Low Rank Matrix Completion
by
Feng Nan
Submitted to the School of Engineering
in partial fulfillment of the requirements for the degree of
Master of Science in Computation for Design and Optimization
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2009
© Massachusetts Institute of Technology 2009. All rights reserved.

Author ....................................................
School of Engineering
August 6 2009

Certified by .....................
V. Parrilo
Professor
Thesis Supervisor

Accepted by ...........................
Jaime Peraire
Professor of Aeronautics and Astronautics
Director, Computation for Design and Optimization Program
Low Rank Matrix Completion

by

Feng Nan

Submitted to the School of Engineering on August 6 2009, in partial fulfillment of the requirements for the degree of Master of Science in Computation for Design and Optimization

Abstract

We consider the problem of recovering a low rank matrix given a sampling of its entries. Such problems are of considerable interest in a diverse set of fields including control, system identification, statistics and signal processing. Although the general low rank matrix completion problem is NP-hard, there exist several heuristic methods that solve the problem approximately by solving the convex relaxation of the original problem. One particularly popular method is to use nuclear norm (sum of singular values) to approximate the rank of the matrix and formulate the problem as a semidefinite program that can be solved efficiently.

In this thesis, we propose a local completion algorithm that searches for possible completion in the neighborhood of each unspecified entry given the rank of the matrix. Unlike existing methods, this algorithm requires only local information of the matrix if the rank is known. Critical in all low rank matrix completion algorithms is the sampling density. The denser the matrix is sampled, the more likely it can be recovered accurately. We then propose a condensation process that increases the sampling density in a specific part of the matrix through elementary row and column re-ordering. Hence we can solve a sub-problem of the original low rank matrix completion problem and gain information on the rank of the matrix. Then the local algorithm is readily applicable to recover the entire matrix. We also explore the effect of additional sampling structures on the completion rate of the low rank matrix completion problems. In particular, we show that imposing regularity in the sampling process leads to slightly better completion rates. We also provide a new semidefinite formulation for a particular block sampling structure that reduces the size of the constraint matrix sizes by a factor of 1.5.

Thesis Supervisor: Pablo. A. Parrilo
Title: Professor
Acknowledgments

I would like to thank my thesis advisor Pablo Parrilo for all the guidance and encouragement throughout the course of this work. I have always enjoyed each meeting with him when he patiently listens to my presentation and gives concrete and constructive advices that help solve many problems. His penetrating insights, clear thinking and enthusiasm of finding things out will continue to influence me for the years to come.

I am also grateful to the Singapore-MIT Alliance (SMA) for providing the fellowship for my study in this master degree program. I cherish each day I spend at MIT, where I have learned knowledge like a thirsty person drinking from a fire hydrant. I’m also privileged to have many precious friendships through classes and Graduate Christian Fellowship. Finally, I also thank Laura Koller for her excellent job in administration and making CDO office a pleasant place to hang out.
# Contents

1 Introduction 13
   1.1 Problem Definition .................................... 13
   1.2 Motivations and Related Work .......................... 15
   1.3 Main Contributions ...................................... 18
   1.4 Structure of the Thesis .................................. 18

2 Local Completion Algorithm 21
   2.1 Rank One Case ($r = 1$) ................................ 24
   2.2 Rank $r > 1$ .............................................. 30

3 Nuclear Norm Minimization 35
   3.1 Nuclear Norm Approximation of Rank ..................... 35
   3.2 Algorithms for Nuclear Norm Minimization ............... 36
      3.2.1 Interior Point Methods for Semidefinite Programming (IPMSDP) 36
      3.2.2 Singular Value Thresholding (SVT) .................... 37
      3.2.3 Sub-gradient Projection (SGP) ......................... 38
   3.3 Sampling schemes ....................................... 40
      3.3.1 Random Uniform Sampling ............................ 40
      3.3.2 Random Matching Sampling ............................ 40
      3.3.3 Blocked Sampling .................................... 41
      3.3.4 A New Block Sampling Formulation .................... 42
   3.4 Numerical Results ....................................... 48
List of Figures

2-1 An illustration of Proposition 5. Two nodes with distance 9 can be connected after 2 iterations. ........................................ 28
2-2 Rank 1 100 × 100 matrix completion rates for local algorithms with various neighborhood sizes ........................................ 29
2-3 Rank 2 matrix completion percentage for various neighborhood sizes using local algorithm ........................................ 33
3-1 Rank 2 100 × 100 matrix completion via nuclear norm minimization using sub-gradient projection method using diminishing step size with sampling density 0.4 ........................................ 39
3-2 Comparing recovery rates of low rank matrix completion with random matching sampling and uniform random sampling schemes. ........................................ 42
3-3 A tree structure for sub-matrix completion. Samples are drawn only from dotted blocks ........................................ 43
3-4 A star structure for sub-matrix completion. Samples are drawn only from dotted blocks ........................................ 44
3-5 Error of rank 2 100 × 100 matrix completion via IPMSDP, average of 20 iterations ........................................ 49
3-6 Rank 2 100 × 100 matrix completion completion percentages using local algorithm ........................................ 50
3-7 Rank 2 100 × 100 matrix completion run time comparison between local and IPMSDP algorithms ........................................ 50
3-8  Sampling schemes comparison, rank 2 500 × 500 matrix completion via SVT, constant step size 1.9, max iterations 150 .......................... 52

4-1  Condensation on a 200x200 matrix at various sampling densities and sub-matrix sizes ......................................................... 62
List of Tables

2.1 Probability of Connectedness for sampling density = 0.1 ................. 27
2.2 Probability of Connectedness for sampling density = 0.4 ................. 27

5.1 Comparison of completion error between SVT and Mixed Algorithm
(rank 2 200 \times 200 matrix) .............................................. 65
Chapter 1

Introduction

1.1 Problem Definition

In this thesis we are interested in solving the Low Rank Matrix Completion (LRMC) problem. Given an \( m \times n \) matrix \( M \), some entries of \( M \) are sampled (observed) while others are not. How does one fill out the unsampled entries (i.e. complete the matrix) such that the resulting matrix has the lowest rank? Let \( M_{ij} \) denote the entry at the \( ith \) row and \( jth \) column of \( M \). Define the sample set \( \Omega \) as the set of all \( (i, j) \) such that \( M_{ij} \) is sampled. In other words, \( M_{ij} \) is sampled if and only if \( (i, j) \in \Omega \). The LRMC problem can be stated as follows.

**Problem 1.1.1.** (LMRC) Given a partially sampled matrix \( M \) and its sample set \( \Omega \), compute the unsampled entries of \( M \) such that the resulting matrix has the lowest rank.

The LRMC problem can be regarded as an optimization problem

\[
\begin{align*}
\text{minimize} & \quad \text{rank}(X) \\
\text{subject to} & \quad X_{ij} = M_{ij} \quad (i, j) \in \Omega.
\end{align*}
\] (1.1)

Here are some illustrative examples.
Example 1.1.2. Consider the partially sampled matrix $M$ equal to
\[
M = \begin{bmatrix}
1 & 2 & 3 \\
2 & 4 & ? \\
? & ? & 9
\end{bmatrix},
\]
where each unsampled entry is denoted as $\,?\,$. There are an infinite number of completions of $M$. One possible way to complete the matrix is given by
\[
X_1 = \begin{bmatrix}
1 & 2 & 3 \\
2 & 4 & 0 \\
1 & 3 & 9
\end{bmatrix}.
\]
Another way to complete the matrix is
\[
X_2 = \begin{bmatrix}
1 & 2 & 3 \\
2 & 4 & 6 \\
3 & 6 & 9
\end{bmatrix}.
\]
Clearly $\text{rank}(X_1) = 3$ and $\text{rank}(X_2) = 1$. Since given the sampled entries of $M$ the lowest rank possible is 1, we conclude that $X_2$ is a solution to the LRMC problem whereas $X_1$ is not. Notice also that $X_2$ is the unique rank 1 completion of $M$.

There are also situations where the LRMC problem does not have a unique solution.

Example 1.1.3. Consider the partially sampled matrix $M$ equal to
\[
M = \begin{bmatrix}
1 & 2 & 3 \\
2 & 5 & ? \\
? & ? & 9
\end{bmatrix}.
\]
The rank of $M$ is at least 2 because the first $2 \times 2$ principal matrix is nonsingular.
It can be seen that

\[ X_1 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 7 & 9 \end{bmatrix} \]

and

\[ X_2 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 6 & 9 \end{bmatrix} \]

are both LRMC solutions for \( M \).

The LRMC problem thus deals with computing the unsampled entries of the matrix based on the sampled ones and ensuring that the completed matrix achieves the lowest rank. The uniqueness of the solution will also be considered.

### 1.2 Motivations and Related Work

In many practical problems, it is of interest to recover a matrix from a sampling of its entries. For example, given a partially filled out survey, one would like to infer the missing entries. Another example can be found in wireless sensor networks, where each sensor can only obtain distance information to its nearest neighbors; the task of localization involves completing the partially observed distance matrix. Without additional information, however, one cannot hope for the recovery of the desired matrix from the knowledge of a subset of its entries because there are infinitely many completions for the unobserved entries. Fortunately in many practical situations, the matrices to be completed have low ranks. This is because the rank of a matrix is often linked to the order, complexity, or dimensionality of the underlying system, which tends to be much smaller than the data size. In the case of inferring missing entries in a survey, it is commonly believed only a few factors contribute to one’s preferences. In the sensor localization problem, the true distance matrix has rank two if the sensors are scattered in a plane or three if they are in a three-dimensional space [16, 1]. Among all the matrices that fit the given data, the one with lowest
rank is therefore often preferred. The LRMC problem is connected with a large body of literature on sparse representation of signals. It is well known that there are many representations for signals, such as sinusoids and wavelets. An effective representation is one that requires very few significant coefficients. It turns out that each one of the representations is effective for some signals but less effective for other ones. It is therefore natural to decompose a signal into a dictionary of several representations for better sparsity. Chen, Donoho and Saunders [6] discovered empirically that minimizing the $l_1$ norm of the coefficients frequently leads to sparse representations of the signals; and in fact when the signal is synthesized from only a few dictionary elements, this method may perfectly recover the specific elements used in the synthesis and their coefficients. This method is known as the Basis Pursuit (BP). Donoho and Huo [8] later proved that if a signal is representable as a highly sparse superposition of elements from such a dictionary, then there is only one such highly sparse representation and it can be obtained using BP. Based on these results, Donoho [7] proposed a compressed data acquisition protocol that performs as if it were possible to directly acquire only the important information about the signal. Specifically, if a signal in $\mathbb{R}^n$ has a sparse representation in some basis with only $N$ coefficients, it is possible to reconstruct the signal exactly from $m = O(N \log(n))$ nonadaptive measurements, much fewer than $n$ measurements. The same idea is presented by Candes, Romberg and Tao, who showed in [5] that it is possible to exactly recover a signal from the partial knowledge of its Fourier coefficients using convex optimization. In fact, when the observed Fourier coefficients are randomly sampled and the signal is sparse, the exact signal can be recovered from a small set of its Fourier coefficients with high probability. All these developments rely on the $l_1$ norm minimization heuristic to recover the sparse signal.

On the other hand, Fazel [9] discovered the nuclear norm (sum of singular values) heuristic for matrix rank minimization, which is the convex relaxation of the rank minimization problem. Authors in [15] showed surprising parallels between the $l_1$ norm minimization used in sparse representation recovery problems and the nuclear
norm minimization used in low rank matrix completion problems. Inspired by these results, Candes and Recht showed in [4] that one can perfectly recover most low-rank matrices from what appears to be an incomplete set of entries using nuclear norm minimization. Specifically, if the given matrix of dimension $n \times n$ with rank $r$ satisfies certain incoherence property, it can be recovered with high probability using only $m = O(n^{1.2} \log n)$ sampled entries from the matrix.

As shown in [4], “most” random matrices satisfy the incoherence property. In particular, consider the singular value decomposition (SVD) of a matrix $M$

$$M = \sum_{k=1}^{r} \sigma_k u_k v_k^*.$$  \hspace{1cm} (1.2)

We could think of a generic low rank matrix as having $\{u_k\}_{1 \leq k \leq r}$ and $\{v_k\}_{1 \leq k \leq r}$ selected uniformly randomly among all families of $r$ orthonormal vectors and we make no assumptions about the singular values $\sigma_k$. This is known as the random orthogonal model, which satisfies the incoherence property required in [4]. Because of the generic nature of this model, the random matrices used in this thesis will follow this model.

Despite the recent progress, the nuclear norm heuristic for solving the LRMC problem has some intrinsic drawbacks.

1. It is often computational expensive when the problem size becomes large. Fazel [9] provided an elegant semidefinite programming formulation of the nuclear norm minimization problem and thus standard semidefinite solvers can be used to solve such problems. But such general-purpose solvers cannot handle large problem sizes. For example, SDPT3 [17] - one of the most advanced semidefinite programming solvers, can only handle $n \times n$ matrices with $n \leq 100$. Although specialized algorithms such as Singular Value Thresholding (SVT) [3] have been proposed that can handle much larger problem sizes, these algorithms generally have slow convergence rates and depend sensitively on the choice of parameters.

2. It requires higher sampling density than needed to recover the low rank matrix.
As we will see in Section 3.4, the nuclear norm heuristic fails to recover the matrix of the lowest rank in situations where the sampling density falls below certain threshold; this threshold could be improved so that exact completion is possible even when the sampling density becomes very low.

1.3 Main Contributions

The main contribution of this thesis are

1. We propose a local completion algorithm that is capable of handling large problem sizes. And because the completion process requires only local information of the matrix, large matrices could be divided into blocks and distributed to multiple processors; each processor can then carry out the completion algorithm locally with minimal amount of communication between neighboring processors. This makes the LRMC problem computationally parallelizable and more scalable.

2. We also propose a condensation process to reduce the sampling density threshold required for perfect recovery. This process condenses the sampled entries in a matrix and raises the sampling density in a sub-block of the given matrix. Since the sampling density is crucial in all matrix completion problems, this technique can be useful in a wide range of settings.

3. We explore additional structures in the sampling process and provide a new formulation of the nuclear norm minimization in a special block sampling case that reduces the constraint semidefinite matrix size. These additional sampling structures could be exploited for better completion rates.

1.4 Structure of the Thesis

- **Chapter 2: The Local Algorithm.** We introduce the local algorithm for matrix completion and study the basic properties such as the condition for
completion and its probabilities. We also resolve a few practical difficulties such as finding potential completions and reducing numerical errors. A C program that implements the local algorithm with variable search neighborhood sizes is also provided in the Appendix.

- **Chapter 3: Nuclear Norm Minimization.** We discuss the popular nuclear norm heuristic and various existing algorithms including Semidefinite Programming, Singular Value Thresholding and Sub-gradient Projection. Then we explore additional sampling structures with their effects on practical performance and problem formulations. We also compare the performance of nuclear norm heuristic and the local algorithm proposed in Chapter 2.

- **Chapter 4: Condensation** The condensation procedure is introduced and its asymptotic behavior is derived. We also demonstrate its effectiveness by a numerical experiment, in which the sampling density can be significantly raised in a sub-block of the matrix.

- **Chapter 5: Mixed Algorithm.** This chapter ties together various components of the thesis into a mixed nuclear norm and local algorithm. We show in a numerical experiment that it can solve large LRMC problems more accurately than existing algorithms.

- **Chapter 6: Conclusions.** This chapter contains the concluding remarks of the thesis and a discussion on future work.
Chapter 2

Local Completion Algorithm

Assume the rank of the matrix is given, a natural way to complete the matrix from its samples is to look for sub-matrices that are *almost complete*. The following example illustrates this simple idea.

**Example 2.0.1.** Recall the matrix in Example 1.1.2, consider the same partially sampled matrix $M$ equal to

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & ? \\ ? & ? & 9 \end{bmatrix}.$$  

Suppose the rank of $M$ is given to be 1, the matrix can be completed by inspecting the $2 \times 2$ sub-matrices

$$X_1 = \begin{bmatrix} 1 & 3 \\ 2 & M_{23} \end{bmatrix},$$

$$X_2 = \begin{bmatrix} 1 & 3 \\ M_{31} & 9 \end{bmatrix},$$

and

$$X_3 = \begin{bmatrix} 2 & 3 \\ M_{32} & 9 \end{bmatrix}.$$  

Clearly knowing the rank of $X_1$, $X_2$ and $X_3$ to be 1 implies $M_{23} = 6$, $M_{31} = 3$, $M_{32} = 9$.
and $M_{32} = 6$. The matrix $M$ can thus be completed as

$$
\begin{bmatrix}
1 & 2 & 3 \\
2 & 4 & 6 \\
3 & 6 & 9
\end{bmatrix}
$$

This motivates a way to complete the matrix by looking for such almost complete sub-matrices in the neighborhood of the unsampled entries. Here we define neighborhood as follows.

**Definition 1.** In a matrix $M \in \mathbb{R}^{m \times n}$, a neighborhood of size $k$ of the entry $M_{ij}$ is a sub-matrix consists of the rows from $i - k$ to $i + k$ and columns from $j - k$ to $j + k$ of the matrix $M$. If $i < k$, the rows start from $m + i - k$ till $m$ and wrap around to $i + k$; if $i + k > m$, the rows start from $i - k$ till $m$ and then wrap around to $i + k - m$. The same wrapping mechanism applies to the columns.

Notice each completion does not increase the rank of the matrix; since the rank of the matrix is given, the completed matrix will be the solution to the LRMC problem. Compared to the nuclear norm heuristic, this method can be carried out locally and therefore has the advantage of being able to handle large matrix sizes. The computation can also be parallelized easily with minimal amount of communication overhead.

A more precise statement of the completion step is given in the following proposition.

**Proposition 1.** If matrix $M$ is of dimension $(k + 1) \times (k + 1)$ with rank $k$ and $M$ can be written in the form:

$$
\begin{bmatrix}
A & b \\
c^T & d
\end{bmatrix},
$$

where $A$ is a $k \times k$ nonsingular matrix, then

$$
d = c^T A^{-1} b. \quad (2.1)
$$
Therefore, suppose $d$ is the unsampled entry to be completed and $A, b, c$ are all sampled (or known), (2.1) can be used to compute $d$ if the hypothesis of Proposition 1 is satisfied. The local algorithm makes use of this result to iterate through the unsampled entries and look for completion in their neighborhoods. To be more explicit, the algorithm is summarized in Algorithm 1.

**Algorithm 1 Local Algorithm for Matrix Completion**

Input: $A$ - input matrix, $\Omega$ - sample set, $r$ - rank of matrix $A$

Output: $A$

while a new completion can be found do
   iterate to find $(i, j) \notin \Omega$
   if no such $(i, j)$ then
      return completed
   else
      look for completion in the neighborhood of $A_{ij}$
      if completion found then
         compute and store the new entry
         $\Omega \leftarrow \Omega \cup \{(i, j)\}$
      end if
   end if
end while

Instead of proving Proposition 1, we shall prove a more general proposition.

**Proposition 2.** Given a block partition of a matrix:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

where $A \in \mathbb{R}^{m_1 \times n_1}$, $B \in \mathbb{R}^{m_1 \times n_2}$, $C \in \mathbb{R}^{m_2 \times n_1}$ and $D \in \mathbb{R}^{m_2 \times n_2}$. Suppose the rank of $M$ is $r$, which is equal to the rank of $A$. Then

$$D = CA^+B,$$  \hspace{1cm} (2.2)

where $A^+$ is the pseudoinverse of $A$.

**Proof.** Since $A$ and $M$ has the same rank, $[CD]$ must be a linear combination of the rows of $[AB]$. Algebraically, there exists a $m_2 \times m_1$ matrix $p$ such that
\[ p[AB] = [CD]. \] Similarly, there exists \( p_c \in \mathbb{R}^{m_2 \times m_1} \) such that \( p_cA = C \); and there exists \( p_b \in \mathbb{R}^{n_1 \times n_2} \) such that \( Ap_b = B \). Now \( CA^+B = p_cAA^+Ap_b = p_cAp_b = p_cB \), where the second equality follows from the property of pseudoinverse. Suppose, to get a contradiction, that \( p_cB \neq D \). From the definitions of \( p \) and \( p_c \) we have \( pA = p_cA = C \), so \( (p - p_c)A = 0 \); whereas \( D = pB \neq p_cB \) so \( (p - p_c)B \neq 0 \). This is a contradiction because \( (p - p_c)B = (p - p_c)Ap_b = 0 \). Thus we have proved \( D = CA^+B \)
\[ \square \]

Next, we discuss the details and related properties of this local completion algorithm when the matrix rank is \( r = 1 \) and \( r > 1 \) in two separate sections.

### 2.1 Rank One Case (\( r = 1 \))

We begin with rank 1 matrices because of their simplicity as far as matrix completion is concerned. The first question we are interested in is: when can a rank 1 matrix be completed using the local algorithm? It is helpful to consider a bipartite graph \( G_{m,n} \) defined as the following.

**Definition 2.** Given a matrix \( M \in \mathbb{R}^{m \times n} \) and its sample set \( \Omega \), its associated completion graph \( G_{m,n} \) is the bipartite graph with \( m \) vertices on the left side, denoted as \( V_1 \), and \( n \) vertices on the right side, denoted as \( V_2 \); with an edge between the \( i \)th vertex on the left and the \( j \)th vertex on the right if and only if \( (i, j) \in \Omega \).

Completing an entry in this case is equivalent to looking for a \( 2 \times 2 \) sub-matrix (after possible row and column permutations) that contains this entry together with 3 other sampled entries. (See Proposition 1). In terms of the completion graph, if there exists a path of length 3 connecting two vertices but there is no edge between them, then an edge can be added to connect the two vertices. We are now ready to answer the first question.

**Proposition 3.** A Rank 1 matrix \( M \in \mathbb{R}^{m \times n} \) can be completed if and only if the corresponding completion graph \( G_{m,n} \) is connected.
Proof. Suppose \( G_{m,n} \) is not connected, then there exist \( a_1 \in V_1, a_2 \in V_2 \) such that there is no path from \( a_1 \) to \( a_2 \). Since completing any entry in the matrix involves adding an edge between 2 vertices that are already connected by a path of length 3, the new edge does not affect the connectedness of the graph. So the graph remains disconnected. In particular, there is still no path from \( a_1 \) to \( a_2 \). So the matrix can not be completed.

Conversely, if the graph \( G_{m,n} \) is connected, for any two vertices \( \forall a_1 \in V_1, \forall a_2 \in V_2 \) there is a path of length \( k \) connecting \( a_1 \) to \( a_2 \), say

\[
a_1, a_2, e_1, a_2, e_2, a_2, e_3, a_2, e_4, a_2, \ldots, e_k, a_2.
\]

Let \( k = 2n + 1 \) and suppose \( k \geq 3 \). Since \( k \) is odd, a new edge can be added between \( a_1 \) and \( e_{x_4} \) and a path from \( a_1 \) to \( a_2 \) becomes \( a_1, e_{1,x_4}, a_{x_4}, e_{x_4}, \ldots, e_{x_k}, a_{x_k}, a_2 \) with length reduced to \( 2n + 1 - 2 = 2n - 1 \). Carry out the same procedure \( n \) times and there will be a path \( a_1, e_{1,2}, a_2 \) with length equals \( 2n + 1 - 2n = 1 \), which means the matrix entry \( M_{a_1,a_2} \) is completed. This works for any \( a_1 \in V_1 \) and \( a_2 \in V_2 \). Therefore the entire matrix can be completed. \( \square \)

Besides connectedness of the completion graph, the neighborhood size plays an important role in the local algorithm. One can show that local algorithm does not work in rank 1 matrix completion if the neighborhood size is too small.

Example 2.1.1. Consider the matrix given by

\[
\begin{pmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0
\end{pmatrix},
\]

where zeros represent unsampled entries. For this matrix, the local algorithm cannot complete the matrix with neighborhood size 1. But the completion graph is connected and the algorithm works if the neighborhood size is big enough to include the entire
If the matrix is uniformly sampled, using Proposition 3 we can compute the probability that the given matrix can be completed by studying its completion graph as shown in Proposition 4.

**Proposition 4.** Given a bipartite graph $G_{m,n}$, suppose the probability of having an edge between any two vertices belonging to the two different vertex sets is $p$, independent of any other edges. Let $q = 1 - p$. Then the probability of the bipartite graph $G_{m,n}$ being connected is $P_{m,n}$, which can be computed recursively via:

$$P_{m,n} = 1 - \sum_{1 \leq k \leq m} \sum_{0 \leq l \leq n} \binom{m-1}{k-1} \binom{n}{l} p_{k,l} q^{l(k-m)+k(n-l)},$$  \hspace{1cm} (2.3)

where $p_{1,0} = P_{0,1} = 1$, $p_{1,1} = p$. \hspace{1cm} (2.4)

**Proof.** Given a bipartite graph $G_{m,n}$ and the nodes are partitioned into two sets $V_1$ and $V_2$. Without loss of generality, pick a node from $V_1$ to be the root node, say $a_1$. Let $K$ and $L$ be the random variables of the number of nodes in the connected component of the graph containing the root node $a_1$ from $V_1$ and $V_2$ respectively. Denote this connected component containing $a_1$ as $A_1$. Then the probability of $G_{m,n}$ being disconnected with $A_1$ having $k$ nodes from $V_1$ and $l$ nodes from $V_2$ is:

$$\binom{m-1}{k-1} \binom{n}{l} p_{k,l} q^{l(k-m)+k(n-l)}.$$  \hspace{1cm} (2.6)

$q^{l(k-m)+k(n-l)}$ ensures that there is no edge linking any node from $A_1$ to any node outside $A_1$. Using the theorem of total probability, the probability of the graph $G_{m,n}$ being disconnected is given by Equation (2.3). \hfill \Box
Tables 2.1 and 2.2 show the probability of the completion graph being connected when any edge being present with probability \( p = 0.1 \) and 0.4, respectively. These probabilities are computed using variable precision arithmetic in Matlab. We can see that as the size of the matrix becomes large, the probability of the completion graph being connected is almost 1 for square or near square matrices when \( p = 0.4 \).

Another property provides an upper bound on the maximum number of iterations for the local algorithm if the neighborhood size is that of the dimension of the matrix; we shall call this as the global-size local completion algorithm.

**Proposition 5.** Assume \( G_{m,n} \) is a connected bipartite graph. Let \( a_1 \in V_1, a_2 \in V_2 \) such that the distance from \( a_1 \) to \( a_2 \), say \( k \), is the longest among all distance \((a_i, a_j)\) where \( a_i \in V_1, a_2 \in V_2 \). Define the diameter of \( G_{m,n} \) to be this distance \( k \). Then the maximum number of iterations for the rank 1 global-size local completion algorithm is upper bounded by \( \log_3 k \).
Figure 2-1: An illustration of Proposition 5. Two nodes with distance 9 can be connected after 2 iterations.

Proof. The distances between any two vertices belonging to the different sides of the bipartite graph has to be odd. Following a similar argument as in the proof of Proposition 3, every iteration reduces all such distances by at least 2 except when the distances are already 1. So every pair of vertices from the opposite sides of the bipartite graph can be connected after at most $\log_3 d$ iterations, where $d$ is their distance. Thus, the entire matrix can be completed after at most $\log_3 k$ iterations. See an illustration in Figure 2-1. \qed

We have run the local completion algorithm on 100 $\times$ 100 rank 1 matrices with variable neighborhood sizes. The same procedure is repeated 100 times for each (density, neighborhood size) pair and the average result is plotted in Figure 2-2, which shows that the bigger the neighborhood size, the better it is able to complete the rank 1 matrix.
Figure 2-2: Rank 1 100 × 100 matrix completion rates for local algorithms with various neighborhood sizes
2.2 Rank $r > 1$

In this section we discuss the implementation of the general local algorithm for matrix completion and present some numerical results. Note that the rank $r$ matrix can be completed if and only if there exists a biclique that includes the edge of the current unsampled entry at each step of the algorithm.

Now we discuss some implementation issues:

1. The program `rankKLocalVN3.c` (attached in the Appendix B) implements the local algorithm and uses Equation (2.1) to perform completion on the unsampled entries of rank $r$ matrices. The program takes in the data matrix, the rank of the matrix $r$, the neighborhood size $k$ and an auxiliary binary matrix with the same size as the data matrix and has value 1 if the corresponding entries in the data matrix is sampled or 0 otherwise. The algorithm works as follows: it keeps a list of the unknown entries of in the data matrix and in each iteration goes through the list. For each unknown entry, it considers the neighborhood of size $(2 \times k + 1) \times (2 \times k + 1)$ centered at the current unknown entry. It then determines if there exists a completion in this neighborhood. If there is no completion in any iteration after scanning the list of unknown entries, the program terminates. Otherwise it repeats as many times as the number of unknown entries in the initial matrix and then terminates.

2. The basic version of the local algorithm results in highly inaccurate completion (often with Inf and NaN's) because of error propagation. To be concrete, consider the following extreme example.

**Example 2.2.1.** Suppose a rank 2 partially sampled matrix $M$ is given by

$$
\begin{bmatrix}
0.980000000000000 & 0.990000000000000 & 0.560000000000000 & 0.310000000000000 & 0.310000000000000 \\
0.140000000000000 & 0.141500000000000 & 0.090000000000000 & 0.043000000000000 & 0.043000000000000 \\
0.449120000000000 & 0.453681500000000 & ? & ? & 0.253649999999999 \\
0.449119999999999 & 0.453681499999999 & 0.253649999999989 & 0.142453000000001 & 0.142453000000001 \\
0.644000000000000 & ? & 0.365000000000000 & 0.204100000000000 & 0.204100000000000
\end{bmatrix}
$$
where the unsampled entries are denoted as ?. And the original matrix \( C \) we wish to recover is actually

\[
\begin{pmatrix}
0.980000000000000 & 0.990000000000000 & 0.560000000000000 & 0.310000000000000 \\
0.140000000000000 & 0.141500000000000 & 0.090000000000000 & 0.043000000000000 \\
0.449120000000000 & 0.453681500000000 & 0.253650000000000 & 0.142453000000000 \\
0.449119999999999 & 0.453681499999999 & 0.253649999999999 & 0.1424530000000001 \\
0.644000000000000 & 0.650550000000000 & 0.365000000000000 & 0.204100000000000 \\
\end{pmatrix}
\]

We can use the local algorithm to compute the entry \( M_{33} \) according to (2.1) with

\[
c = [0.449120000000000, 0.453681500000000],
\]

\[
A = \begin{pmatrix}
0.980000000000000 & 0.990000000000000 \\
0.140000000000000 & 0.141500000000000 \\
0.560000000000000 & 0.090000000000000 \\
\end{pmatrix},
\]

\[
b = \begin{pmatrix}
0.560000000000000 \\
0.090000000000000 \\
\end{pmatrix}.
\]

Similarly, \( M_{34} \) can be computed with

\[
c = [0.449120000000000, 0.453681500000000],
\]

\[
A = \begin{pmatrix}
0.980000000000000 & 0.990000000000000 \\
0.140000000000000 & 0.141500000000000 \\
0.310000000000000 & 0.043000000000000 \\
\end{pmatrix},
\]

\[
b = \begin{pmatrix}
0.310000000000000 \\
0.043000000000000 \\
\end{pmatrix}.
\]

The computed values are \( M_{33} = 0.253649999999999 \) and \( M_{34} = 0.142453000000001 \).
To compute the third unsampled entry $M_{52}$, if we were to use (2.1) with

$$c = [0.3650000000000000, 0.2041000000000000, 0.2536499999999890, 0.1424530000000010, 0.4536815000000000],$$

$$A = \begin{bmatrix}
0.2536499999999890 & 0.1424530000000010 \\
0.2536499999999890 & 0.1424530000000010 \\
0.4536815000000000 & 0.4536815000000000
\end{bmatrix},$$

$$b = \begin{bmatrix}
0.4536815000000000 \\
0.4536815000000000
\end{bmatrix},$$

we would get $M_{52} = \inf$ since $A$ is singular. The trouble here comes from using the newly computed $M_{33}$ and $M_{34}$ in the matrix inverse to further compute other entries.

The modification is made to use only the originally available data in the matrix inverses to complete new entries. This restriction means that computed entries are not allowed to be reused to compute other unsampled entries, which inevitably reduces the completion percentage as a trade-off. Fortunately, by iteratively running the program with the previous outputs as current inputs, accurate matrix completion is achieved.

Figure 2-3 is the completion percentage plot of this local algorithm with rank 2 input data matrices of size $100 \times 100$ under different sampling densities and various neighborhood sizes. This is the average result of 50 identical experiments.
Figure 2-3: Rank 2 matrix completion percentage for various neighborhood sizes using local algorithm
Chapter 3

Nuclear Norm Minimization

The nuclear norm minimization is a powerful heuristic for rank minimization problems and it possess many theoretical and practical advantages (See [9, 15, 4]). In this chapter we first give a brief introduction to the nuclear norm heuristic and existing algorithms. Then we explore new sampling structures and their impact on problem formulation as well as practical performance. The motivation for such studies is to improve completion rate and accuracy given the freedom to sample the matrix in certain structured manner.

3.1 Nuclear Norm Approximation of Rank

As seen from (1.1), the LRMC problem can be formulated as a rank minimization problem. However, such rank minimization problems are non-convex and NP-hard. To approximately solve the original problem, we can consider its convex relaxation problem. Fazel [9] showed that the nuclear norm, or the sum of the singular values of the matrix is the convex envelope of the rank function over the set of matrices with 2-norm less than one, which means over the set of matrices with 2-norm less than one, the nuclear norm function is the largest convex function less than or equal to the rank function. Thus we can use the nuclear norm to approximate the non-convex rank function and obtain a lower bound on the optimal value. And this lower bound is the tightest lower bound by all convex approximations of the rank
Incidentally, it is shown in [9] that the nuclear norm heuristic is related to the widely used trace heuristic. Also, like $l_1$ norm minimization frequently leads to exact sparse representation of signals, the nuclear norm frequently leads to the exact minimum rank solution under some reasonable assumptions discussed in [15]. Thus, the nuclear norm heuristic has gained its popularity in rank minimization problems.

### 3.2 Algorithms for Nuclear Norm Minimization

In this section we consider three different algorithmic methods for solving the nuclear norm minimization problem, namely, Interior Point Methods for Semidefinite Programming (IPMSDP), Singular Value Thresholding (SVT) and Sub-gradient Projection (SGP).

#### 3.2.1 Interior Point Methods for Semidefinite Programming (IPMSDP)

Semidefinite Programming has been a popular research topic in recent years and substantial progress has been made in both theory and application. With the help of powerful solvers that implement efficient algorithms such as interior point methods, SDP has become increasingly mature as a technology that has found wide areas of applications.

It is shown in [9] that the convex problem of nuclear norm minimization

\[
\begin{align*}
\text{minimize} & \quad \|X\|_* \\
\text{subject to} & \quad X_{ij} = M_{ij} \quad (i, j) \in \Omega
\end{align*}
\]  

(3.1)

can be formulated as a semidefinite program; hence it can be solved efficiently using
existing technologies. The problem (3.1) is equivalent to

\[
\begin{align*}
\text{minimize} & \quad t \\
\text{subject to} & \quad \|X\|_* \leq t \\
& \quad X_{ij} = M_{ij} \quad (i, j) \in \Omega
\end{align*}
\]  

(3.2)

By the lemma in [9, Lemma 2], (3.2) is equivalent to:

\[
\begin{align*}
\text{minimize} & \quad t \\
\text{subject to} & \quad \text{Tr}(Y) + \text{Tr}(Z) \leq 2t \\
& \quad \begin{bmatrix} Y & X \\
X' & Z \end{bmatrix} \succeq 0 \\
& \quad X_{ij} = M_{ij} \quad (i, j) \in \Omega
\end{align*}
\]  

(3.3)

, which is now a semidefinite program.

The Interior Point Methods for SDP provides one way of solving the nuclear norm minimization problem. It has the advantage of sound theoretical guarantees and works well in practice. The disadvantage is that it cannot handle large problem sizes.

\subsection{3.2.2 Singular Value Thresholding (SVT)}

The singular value thresholding algorithm [3] is a simple algorithm for minimum rank matrix completion problems. We only give a sketch of the algorithm here. Let \( P_\Omega \) be the orthogonal projector onto the span of matrices vanishing outside of sample set \( \Omega \) so that the \((i, j)\)th component of \( P_\Omega(X) \) is equal to \( X_{ij} \) if \((i, j) \in \Omega \) and zero otherwise. So given \( M \in \mathbb{R}^{m \times n} \), the nuclear norm minimization problem (3.1) can be equivalently expressed as

\[
\begin{align*}
\text{minimize} & \quad \|X\|_* \\
\text{subject to} & \quad P_\Omega(X) = P_\Omega(M)
\end{align*}
\]  

(3.4)
Fix $\tau > 0$ and a sequence $\{\delta_k\}_{k=1}^\infty$ of step sizes. Then starting with $Y^0 = 0 \in \mathbb{R}^{m \times n}$, the algorithm inductively defines

$$
\begin{align*}
X^k &= \text{shrink}(Y^{k-1}, \tau), \\
Y^k &= Y^{k-1} + \delta_k P_{\Omega}(M - X^k)
\end{align*}
$$

(3.5)

until a stopping criterion is reached. In (3.5), $\text{shrink}(Y, \tau)$ is a nonlinear function which applies a soft-thresholding rule at level $\tau$ to the singular values of the input matrix. More precisely, consider the Singular Value Decomposition (SVD) of a matrix $X \in \mathbb{R}^{m \times n}$ of rank $r$

$$
X = U \Sigma V^*, \quad \Sigma = \text{diag}(\{\sigma_i\}_{1 \leq i \leq r}),
$$

(3.6)

where $U$ and $V$ are respectively $m \times r$ and $n \times r$ matrices with orthonormal columns, and the singular values $\sigma_i$ are positive. For each $\tau \geq 0$, the shrinkage operator $\text{shrink}(X, \tau)$ is defined as follows:

$$
\text{shrink}(X, \tau) := UD_\tau(\Sigma)V^*, \quad D_\tau(\Sigma) = \text{diag}(\{\sigma_i - \tau\})_+,
$$

(3.7)

where $t_+$ is the positive part of $t$, namely, $t_+ = \max(0, t)$.

Clearly the SVT algorithm performs only elementary matrix operations and a singular value decomposition at each iteration of the algorithm. The key advantage is that it can deal with much larger matrix sizes than those can be handled by current SDP solvers. It is therefore the method of choice in the next section to compare different sampling schemes.

### 3.2.3 Sub-gradient Projection (SGP)

Let $X$ be an $m \times n$ matrix of rank $r$ and let $X = U \Sigma V'$ be a reduced singular value decomposition. The subdifferential of the nuclear norm at $X$ is given by (see,
Figure 3-1: Rank 2 $100 \times 100$ matrix completion via nuclear norm minimization using sub-gradient projection method using diminishing step size with sampling density 0.4

e.g.,[18])

$$\partial\|X\|_* = \{UV' + W : W \text{ and } X \text{ have orthogonal row and column spaces, and } \|W\| \leq 1\}$$

(3.8)

The Projected subgradient method as mentioned in [15] is implemented by using a simple diminishing step size rule (See [2] for general subgradient projection methods) and using a larger matrix size. The program converges to recover the original low rank matrix as shown in Figure 3-1, which also shows that the norm of the difference with the true matrix converges too slowly to meet the desired accuracy; therefore, SGP is not used in the numerical experiments in this thesis.
3.3 Sampling schemes

Given a matrix $M$, there are different ways of sampling it, either deterministically or randomly. In many situations we do not have any control on how the matrix is sampled and the samples seem to distribute uniformly across the matrix; for example, when we ask customers to fill out a survey, we can not guarantee which questions are answered and which ones are left blank. We can model such sampling scheme as a random uniform sampling (See Section 3.3.1). In other situations we do have control over how sampling is done on the given matrix; for example, in Multi-Dimensional Scaling (MDS) (See [14]) used in psychometrics and statistics, it is of interest to complete an Euclidean Distance Matrix (EDM) representing distances of data points that is not only consistent with the measurements but also requires the smallest number of coordinates to represent the data. The measurement of inter point distances is usually done through a set of experiments; therefore, the aim is to design the set of experiments such that the optimal EDM can be solved with a small amount of measurements. Knowing what sampling schemes lead to better efficiency (better completion rate) is therefore very important. This motivates our study of the effects of different sampling schemes have on the LRMC problem. In this section, random uniform, random matching and block sampling schemes are described. Furthermore, a new semidefinite formulation is presented for a special block sampling scheme.

3.3.1 Random Uniform Sampling

Entries are sampled uniformly across the matrix. Each entry has probability $p$ of being sampled, independent of other entries. $p$ is therefore the sampling density. This is perhaps the most basic type of sampling.

3.3.2 Random Matching Sampling

To add more structure into the sampling procedure, random matching is used instead of the uniform random sampling scheme. Assume the unknown matrix to be completed is $M \in \mathbb{R}^{n \times n}$ and one has available $m$ sampled entries $M_{ij} : (i, j) \in \Omega$ where
\( \Omega \) is a subset of cardinality \( m \). If one has the choice of which entries of \( M \) to sample, one possible way is to have a set of nodes \( A_i, i = 1, 2...n \) on the left with each node representing a row in \( M \), and a set of nodes \( B_i, i = 1, 2...n \) on the right with each node representing a column in \( M \). Now a random matching between the set \( A \) and \( B \) induces a set of edges \( E_{i,j}, (i, j) \in \omega \), where \( A_i \) is matched with \( B_j \). One can then choose to sample \( M \) at the entries in \( \omega \). Clearly each random matching results in \( n \) entries in \( M \) to be sampled. By repeating the random matching multiple times, one can reach \( m \) sampled points in total. To be more precise, suppose the random matching is repeated \( K \) times with \( \omega_i, i = 1, 2...K \) the set of edges in each random matching, then \( \Omega = \bigcup_{i=1}^{K} \omega_i \). The experimental result shows that using SDP, the nuclear norm minimization problem (3.1) has a slightly higher recovery rate when the sampled entries in \( \Omega \) are obtained using random matching than uniform sampling. This is illustrated in the Figure 3-2. This result was obtained by running SDP on matrices of size \( 90 \times 90 \) 20 times at each density and taking the average. The random matching sampling imposes additional structures resulting in stronger regularity; it is therefore expected to lead to better recovery. On the other hand, as matrix size grows larger, the difference between random uniform sampling and random matching sampling becomes smaller, which explains why their recovery rates are not significantly different.

### 3.3.3 Blocked Sampling

Here the original matrix is divided into 10 sub-blocks and two block sampling structures are considered: Star and Tree structures. The tree sampling structure is illustrated in Figure 3-3. And the star sampling structure is illustrated in Figure 3-4. In both cases above, only the sub-blocks with dots are sampled. We wish to study such block sampling schemes because they impose additional regularity on the completion graphs. Specifically, both sampling schemes result in completion graphs with small diameters, which are 4 and 2 for the tree and star structures respectively.
Figure 3-2: Comparing recovery rates of low rank matrix completion with random matching sampling and uniform random sampling schemes.

3.3.4 A New Block Sampling Formulation

When certain blocks of the matrix are unsampled, we now provide in the following proposition a new formulation of the nuclear norm minimization that reduces the constraint semidefinite matrix size.

Proposition 6. Let $M \in \mathbb{R}^{n \times n}$ and

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix},$$

where $M_{ij}$ are one of the $3 \times 3$ blocks of $M$. If no entry in $M_{23}$ and $M_{32}$ is sampled,
Figure 3-3: A tree structure for sub-matrix completion. Samples are drawn only from dotted blocks.
Figure 3-4: A star structure for sub-matrix completion. Samples are drawn only from dotted blocks
then the nuclear norm minimization problem 3.1 is equivalent to the following problem:

\[
\begin{align*}
\text{minimize} & \quad \operatorname{Tr}(W_{11}) + \operatorname{Tr}(W_{22}) + \operatorname{Tr}(W_{33}) + \operatorname{Tr}(Z_{11}) + \operatorname{Tr}(Z_{22}) + \operatorname{Tr}(Z_{33}) \\
\text{subject to} & \quad 
\begin{bmatrix}
W_{11} & W_{12} & X_{11} & X_{12} \\
W_{21} & W_{22} & X_{21} & X_{22} \\
X_{11}' & X_{12}' & Z_{11} & Z_{12} \\
X_{12}' & X_{33}' & Z_{12}' & Z_{22} \\
W_{11} & W_{13} & X_{11} & X_{13} \\
W_{31} & W_{33} & X_{31} & X_{33} \\
X_{11}' & X_{31}' & Z_{11} & Z_{13} \\
X_{13}' & X_{33}' & Z_{13}' & Z_{33}
\end{bmatrix} \succeq 0 \\
X_{ij} = M_{ij} & \quad (i, j) \in \Omega
\end{align*}
\] (3.9)

Before we provide the proof, we introduce some definitions and cite a useful lemma from Grone et al.\cite{13}.

**Definition 3.** For any finite undirected graph \( G = (V, E) \), a \( G \)-partial matrix is defined as a set of complex numbers, denoted by \( [a_{ij}]_G \) or \( A(G) \), where \( a_{ij} \) is defined if and only if \( (i, j) \in E \).

**Definition 4.** A completion of \( A(G) = [a_{ij}]_G \) is a matrix \( M \) which satisfies \( M_{ij} = a_{ij} \) for all \( (i, j) \in E \). \( M \) is a positive semidefinite completion of \( A(G) \) if and only if \( M \) is a completion of \( A(G) \) and \( M \succeq 0 \).

**Definition 5.** A graph \( G \) is positive semidefinite completable if and only if any of its \( G \)-partial positive semidefinite matrix has a positive semidefinite completion.

**Definition 6.** A graph \( G \) is chordal if there are no minimal cycles of length greater than or equal to 4.

An important characterization of positive semidefinite completable matrices is given in the following lemma.

**Lemma 1.** A graph is positive semidefinite completable if and only if it is chordal.
We are now ready to prove Prop.6.

Proof. As shown in [9], the nuclear norm minimization problem (3.1) is equivalent to the Semidefinite problem:

\[
\begin{align*}
\text{minimize} & \quad \text{Tr}(W_{11}) + \text{Tr}(W_{22}) + \text{Tr}(W_{33}) + \text{Tr}(Z_{11}) + \text{Tr}(Z_{22}) + \text{Tr}(Z_{33}) \\
\text{subject to} & \quad A = \begin{bmatrix}
W_{11} & W_{12} & W_{13} & X_{11} & X_{12} & X_{13} \\
W'_{12} & W_{22} & W_{23} & X_{21} & X_{22} & X_{23} \\
W'_{13} & W'_{23} & W_{33} & X'_{31} & X'_{32} & X'_{33} \\
X'_{11} & X'_{21} & X'_{31} & Z_{11} & Z_{12} & Z_{13} \\
X'_{12} & X'_{22} & X'_{32} & Z'_{12} & Z'_{22} & Z'_{23} \\
X'_{13} & X'_{23} & X'_{33} & Z'_{13} & Z'_{23} & Z'_{33}
\end{bmatrix} \succeq 0
\end{align*}
\]

\(W, Z\) are symmetric
\(X_{ij} = M_{ij} (i, j) \in \Omega\)  \(3.10\)

Multiplying a permutation matrix \(\Pi\) on the left and right side of \(A\) gives matrix \(B\):

\[
\Pi = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
B = \Pi' * A * \Pi = \begin{bmatrix}
W_{11} & X_{11} & W_{12} & X_{12} & W_{13} & X_{13} \\
X'_{11} & Z_{11} & X'_{21} & Z_{12} & X'_{31} & Z_{13} \\
W'_{12} & X_{21} & W_{22} & X_{22} & W_{23} & X_{23} \\
X'_{12} & Z'_{12} & X'_{22} & Z_{22} & X'_{32} & Z_{23} \\
W'_{13} & X_{31} & W'_{23} & X_{32} & W_{33} & X_{33} \\
X'_{13} & Z'_{13} & X'_{23} & Z'_{23} & X'_{33} & Z_{33}
\end{bmatrix}
\]

Since \(B\) is symmetric and is similar to \(A\), \(A\) is positive semidefinite if and only if \(B\)
is positive semidefinite. If

\[
B_{12} = \begin{bmatrix}
W_{11} & X_{11} & W_{12} & X_{12} \\
X'_{11} & Z_{11} & X'_{21} & Z_{12} \\
W_{12} & X_{21} & W_{22} & X_{22} \\
X'_{12} & Z'_{12} & X'_{22} & Z_{22}
\end{bmatrix} \succeq 0
\]

and

\[
B_{13} = \begin{bmatrix}
W_{11} & X_{11} & W_{13} & X_{13} \\
X'_{11} & Z_{11} & X'_{31} & Z_{13} \\
W_{13} & X_{31} & W_{33} & X_{33} \\
X'_{13} & Z'_{13} & X'_{33} & Z_{33}
\end{bmatrix} \succeq 0,
\]

then \( B \) is a partial non-negative matrix and the corresponding graph is chordal. By Lemma 1 there exist matrices

\[
\begin{bmatrix}
W_{23} & X_{23} \\
X'_{32} & Z_{23}
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
W'_{23} & X_{32} \\
X'_{23} & Z'_{23}
\end{bmatrix}
\]

such that \( B \succeq 0 \). Since \( X_{23} \) and \( X_{32} \) are entirely unknown by assumption, there is no constraint on them. It is clear that if \( B_{12} \succeq 0 \) and \( B_{13} \succeq 0 \), \( B \) can be made positive semidefinite. On the other hand, \( B \succeq 0 \) implies \( B_{12} \succeq 0 \) and \( B_{13} \succeq 0 \). This completes the proof. \( \square \)

Remark 3.3.5. This new formulation reduces the constraint semidefinite matrix size from \( 2n \times 2n \) in (3.10) to \( \frac{4}{3}n \times \frac{4}{3}n \) in (3.9).

We use CVX, a package for specifying and solving convex programs \([12, 11]\) for solving SDP. A sample code to implement (3.3) is shown below:

%%semidefinite programming:
cvx_begin sdp
  variable X(m,n);

47
3.4 Numerical Results

In this section we present the numerical results from various experiments. First, we compare the local completion algorithm proposed in Chapter 2 with the Interior Point Methods for SDP (IPMSDP) described in Section 3.2.1. We show that the local algorithm can recover the matrix from a lower sampling density than the IPMSDP algorithm; also, the run time for local algorithm does not increase so much as IPMSDP when sampling density is increased. Second, we compare the 4 different sampling structures in terms of their completion error for various sampling densities.

1. Compare the local and IPMSDP matrix completion algorithms. The error of completed matrix for various densities using IPMSDP is shown in the Figure 3-5, which shows that the IPMSDP algorithm recovers the exact matrix with high probability when the density is greater than 0.25 in this case.

Figure 3-6 shows the completion percentages using the local algorithm with various neighborhood sizes. This plot is averaged over 50 runs. In particular, when the neighborhood size is greater than 4, the local algorithm can recover the matrix with high probability if the density is greater than 0.25. This is comparable to that of the IPMSDP algorithm in Figure 3-5. More importantly, using larger neighborhood sizes leads to perfect recovery even when the sampling density falls much lower than the threshold required by the IPMSDP algorithm. The run time comparison reveals that as the computation time using IPMSDP increases nearly quadratically as matrix density increases. As for the local algorithm, the run time first increases then decreases as matrix density becomes larger. For example, to understand the run time plot for neighborhood size 2,
Figure 3-5: Error of rank 2 100 $\times$ 100 matrix completion via IPMSDP, average of 20 iterations

referring to Figure 3-6, the run time initially remains small for density in [0.08, 0.26] because very few entries can be completed; the run time jumps up at density 0.32 to about 270s because about 70% of entries are completed; finally in the density interval [0.32, 0.40] the run time decreases to less than 30s because it becomes easier to find completions as more entries are known. To summarize, the local completion algorithm can achieve the same or better recovery rate than IPMSDP by enlarging the search neighborhood size. And unlike the IPMSDP algorithm, the run time of the local algorithm does not increase as the sampling density increases; it reaches a small peak then decreases. Using larger neighborhood sizes also help reducing the run time as seen from Figure 3-7. This suggests the advantage of using the local algorithm over IPMSDP. Furthermore, the local algorithm can be decomposed into many smaller problems and solved distributely with relatively small amount of communication between sub-problems, which could make this method even faster in practice.
Figure 3-6: Rank 2 $100 \times 100$ matrix completion completion percentages using local algorithm.

Figure 3-7: Rank 2 $100 \times 100$ matrix completion run time comparison between local and IPMSDP algorithms.
2. Compare the 4 different sampling structures. The experiment is done on 500 x 500 matrices using 4 different sampling schemes:

(a) Uniform Random Unrestricted Sampling (URUS): uniform random sampling (See Section 3.3.1) on the entire matrix

(b) Random Matching Unrestricted Sampling (RMUS): random matching sampling (See Section 3.3.2) on the entire matrix

(c) Uniform Random Restricted Sampling - star (URRS-star): uniform random sampling only on the star-structured sampling blocks (See Section 3.3.3)

(d) Uniform Random Restricted Sampling - tree (URRS-tree): uniform random sampling only on the tree-structured sampling blocks (See Section 3.3.3)

Singular Value Thresholding (SVT) is used in these cases as IPMSDP solvers cannot handle matrices bigger than 100 x 100. A comparison on the completion errors averaged over 50 identical runs is shown in Figure 3-8. The blocks that are not sampled are completed using (2.2) in Uniform Random Restricted Sampling - star and Uniform Random Restricted Sampling - tree schemes in Figure 3-8. Notice that the errors for block samplings do not go to 0 even as the sampling density of the blocks are very high. This is because the noise in the sampled blocks is amplified and propagated into the computed unsampled blocks through the pseudoinverse operation. For example, in one instance the error norm of a unsampled 50 x 50 block as computed in the above experiment with sampling density 0.28 is 1.58 but if the original (accurate) sampled blocks are used in the pseudoinverse operation, the resulting block has error norm of only 3.0175e-14. This is when the sampled block being inversed has error norm of 0.1784. To summarize, we can see that URRS-star results in slightly better recovery than URRS-tree. This can be explained by the fact that the star structure has a completion graph of diameter 2 whereas the tree structure has a completion graph of diameter 4; therefore the star structure imposes a stronger
Figure 3-8: Sampling schemes comparison, rank 2 500 × 500 matrix completion via SVT, constant step size 1.9, max iterations 150
regularity condition which leads to better recovery rates. And again the ran-
dom matching sampling increases the rate of recovery compared to uniform random sampling. The restricted block sampling schemes, i.e. URRS-star and URRS-tree have higher recovery errors than the unrestricted sampling scheme, i.e. URUS. And this is mainly because of error propagation involved in the pseudoinverse operation.

The conclusions we draw from these numerical results are the following:

1. The local completion algorithm can achieve better recovery rates than the IPMSDP algorithm given the same sampling density provided the neighborhood size used in the local algorithm is sufficiently large.

2. Perhaps surprisingly, larger neighborhood sizes used in the local algorithm can lead to smaller run times than smaller neighborhood sizes; and the run times of local algorithms tend to be smaller than those of IPMSDP algorithm especially when the sampling density is large.

3. To exploit sampling structure for higher recovery rate, Random Matching Unrestricted Sampling (RMUS) should be used rather than Uniform Random Unrestricted Sampling (URUS); Uniform Random Restricted Sampling - star (URRS-star) should be used rather than Uniform Random Restricted Sampling - tree (URRS-tree). Although the recovery rate differences in these sampling schemes do not differ significantly for large matrix sizes, it is nevertheless noteworthy to have such results in mind in designing specific sampling schemes.
Chapter 4

Condensation

As seen from previous chapters, the sampling density of the matrix plays a critical role in all LRMC problems. The more entries we sample from a matrix, the more likely we can recover the original matrix. Given a partially sampled matrix $M \in \mathbb{R}^{m \times n}$, we are therefore interested in increasing the sample density in a sub-block of the matrix by row and column permutations with the hope that useful information can be gleaned from the condensed sub-block. We shall use the word condensation to refer to this process of concentrating sampled entries into a sub-block of the matrix. It is important to note that such condensation only helps to solve a sub-problem of the original LRMC problem, namely, the sub-block with increased density; the rest of the LRMC problem needs to be solved by other means as we will show in Chapter 5.

To formulate the general condensation problem into an optimization problem, let $A \in \mathbb{R}^{m \times n}$ be a binary matrix of the same size as $M$ with $A_{ij} = 1$ if $M_{ij}$ is sampled and $A_{ij} = 0$ otherwise. Without loss of generality, we consider the permutations to condense the sampled entries into the $k \times k$ sub-matrix on the upper left corner of $M$. Let $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$ be binary vectors with $x_i = 1$ representing the decision of permuting the $i^{th}$ row of $M$ into the first $k$ rows and $x_i = 0$ representing the decision of permuting the $i^{th}$ row of $M$ out of the first $k$ rows; $y_i$'s are similarly defined for the columns. Thus, $x$ and $y$ are decision variables representing the row and column permutations, respectively. Now the condensation problem can be written as follows:
\[
\text{maximize} \quad \sum_i \sum_j x_iz_jA_{ij} \\
\text{subject to} \quad \sum_i x_i = k \\
\sum_j y_j = k \\
x_iy_j \in 0,1
\] (4.1)

This problem is the same as deciding whether or not a bipartite graph contains a biclique of size \(k \times k\). The requirement is that \(|V_1| = |V_2| = k\) for some integer \(k\), where \(|V_1|\) and \(|V_2|\) are the number of vertices on each side of the biclique. (this is called the balanced complete bipartite subgraph problem or balanced biclique problem). Garey and Johnson have shown in their book [10] that this problem is NP-complete.

Some possible approaches to solve the problem include formulating it as a linear integer program and solving it using branch-and-bound. For instance, one could use the big-\(M\) formulation with \(L\) replacing \(M\):

\[
\text{maximize} \quad \sum_i \sum_j W_{ij}A_{ij} \\
\text{subject to} \quad W_{ij} \leq x_i \\
W_{ij} \leq y_j \\
W_{ij} \geq \frac{x_i+y_j-1}{L} \\
\sum_i x_i = k \\
\sum_j y_j = k \\
W_{ij}, x_i, y_j \in 0,1, \forall i, j
\]

where \(L\) is a very large number.

However, in this thesis we adopt a different heuristic approach for solving the condensation problem with the advantage of effectiveness and low computational cost as described next.

Given a matrix \(M \in \mathbb{R}^{n \times n}\), suppose each entry has probability \(p\) of being sampled, independent of other entries. The condensation procedure involves two steps: column and row re-orderings, as described next. First, the column re-ordering sorts the \(n\) columns of matrix \(M\) in descending order with respect to the number of sampled entries in each column. The resulting matrix, denoted as \(M_c\), has denser columns on...
the left than on the right. Next, the row re-ordering sorts the $n$ rows of matrix $M_c$ in descending order with respect to the number of sampled entries in each row up to the first $m$ columns of $M_c$, with $m \leq n$. The resulting matrix, denoted as $M_{cr}$, has denser rows on the top than on the bottom up to the first $m$ columns.

4.1 Column Re-ordering

The number of sampled entries in the $ith$ column of $M$, say $X_i$, is a binomial random variable of parameter $(n,p)$. $X_i, i = 1, 2, ..., n$ has mean $\mu = np$ and variance $\sigma^2 = np(1-p)$. For large $n$, $X_i, i = 1, 2, ..., n$ can be regarded as IID random variables with normal distribution of the above mean and variance, i.e. $N(\mu, \sigma^2)$. The probability that $X_i$ has value $l$ standard deviation above the mean is given by $P_l = P(X_i \geq \mu + l\sigma) = \frac{1}{2} - \frac{1}{2}\text{erf}(\frac{l}{\sqrt{2}})$, where erf$(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is called the error function. Define a new random variable $W_l$ as the number of $X_i's, i = 1, 2, ..., n$ that are larger than $l$ standard deviation above the mean. Clearly $W_l$ is a binomial random variable of parameter $(n, P_l)$. An interesting observation is that as $n$ increases, the expected value $E(W_l) = np_l$ increases. Also, $\sigma^2$ increases; and $X_i$ will be larger than $\mu$ by an increasingly large amount for the same probability $P_l$. However, the density of each column, given by $\rho_i = X_i/n, i = 1, 2, ..., n$, converges to $p$ in probability according to the weak law of large numbers. So the probability that the density in each column deviates from the mean by any given amount is increasingly slim. Moreover, we can show a even stronger result in the following proposition.

**Proposition 7.** Given a matrix $M \in \mathbb{R}^{n \times n}$, suppose each entry has probability $p$ of being sampled, independent of other entries. The probability that at least one of the $n$ columns of $M$ having density deviating from the mean $p$ by any given amount $a > 0$ is approaching 0 as $n \to \infty$.

**Proof.** Let the number of sampled entries in the $ith$ column of $M$ be $X_i$, which can be considered as having normal distribution of parameter $(\mu = np, \sigma^2 = np(1-p))$. 

57
Let \( \rho_i = X_i/n, i = 1, 2, \ldots n. \)

\[
\lim_{n \to \infty} P(\bigcup_{i=1}^{n} (\rho_i \geq p + a)) \leq \lim_{n \to \infty} n \sum_{i=1}^{n} P(\rho_i \geq p + a) \\
= \lim_{n \to \infty} n P(\rho_i \geq p + a) = \lim_{n \to \infty} n P(X_i \geq \mu + na) \\
\leq \lim_{n \to \infty} ne^{-\frac{n^2 a^2}{2\sigma^2}} \\
= \lim_{n \to \infty} ne^{-\frac{3a^2}{2\sqrt{p(1-p)}}} = 0
\]

where the second inequality follows from the Chernoff’s bound for normal random variable \( X \sim N(\mu, \sigma^2) \):

\[
P(X \geq a) \leq e^{-\frac{(a-\mu)^2}{2\sigma^2}}
\]

for \( a > \mu. \)

Prop. 7 suggests that the Step.1 of Condensation that involves column re-ordering is increasingly ineffective as far as column density is concerned when \( n \) goes to infinity.

### 4.2 Row Re-ordering

Here we restrict our attention on the left most \( m \) columns of the matrix \( M_c \), which is obtained after the column re-ordering of \( M \); denote this \( n \times m \) matrix as \( B \). Suppose \( n \) is divisible by \( m, B \) can then be divided into \( n \) blocks, each of size \( m \times m \). Define \( Y_i \) as the number of sampled entries in the \( i \)th block of \( B \). To simplify analysis, we consider the row re-ordering of such blocks of \( B \) instead of individual rows of \( B \). This will provide a conservative estimate of the effectiveness of the actual row re-ordering. Furthermore, we make another simplifying assumption that each entry in \( B \) has probability \( p \) of being sampled, independent of other entries. This is essentially assuming that column re-ordering of \( M \) does not alter the probability distribution of its entries; or even \( B = B_c \). The rationale for this assumption is two-folded: to simplify analysis and that it becomes increasingly accurate as \( n \to \infty \) as seen from Prop. 7.
Under the two assumptions above, $Y_i's, i = 1, 2, \ldots \frac{n}{m}$ are IID binomial random variables of parameter $(m^2, p)$. Let $\bar{Y}_1$ be a random variable such that $\bar{Y}_1 = \max_{i=1,2,\ldots \frac{n}{m}} Y_i$. Thus, the top left $m \times m$ block of $M_{cr}$ will have $\bar{Y}_1$ sampled entries after row reordering. Regarding the density of this corner block of $M_{cr}$, we show the following proposition:

**Proposition 8.** For any fixed $m$, the expected density of the top left corner $m \times m$ block of $M_{cr}$ approaches 1 as $n \to \infty$.

**Proof.**

$$
\lim_{n \to \infty} E \left[ \frac{\bar{Y}_1}{m^2} \right] = \lim_{n \to \infty} \frac{1}{m^2} \sum_{k=0}^{m^2-1} (1 - P(\bar{Y}_1 \leq k))
$$

$$
= \lim_{n \to \infty} \frac{1}{m^2} \sum_{k=0}^{m^2-1} (1 - (P(Y_i \leq k))^{\frac{n}{m}})
$$

$$
= 1
$$

\[ \square \]

### 4.3 Success Probability

In this section, we derive some relationships between $m$, $p$, $n$, and the probability that the top left corner $m \times m$ matrix being fully sampled, denoted as $P(\text{success})$. In particular, we are interested in the question that given a matrix of size $n \times n$, with what sampling density $p$ and block size $m$ can the top left corner $m \times m$ matrix be fully sampled with a desired probability $P(\text{success})$.

Again we restrict our attention to $B$, which is the left most $m$ columns of the matrix $M_c$. Now define $Z_i$ as the number of sampled entries in the $i$th row of $B$. With the same assumption that the probability of each entry being sampled remains $p$ in $B$, $Z_i, i = 1, 2, \ldots, n$ are IID binomial random variables with parameters $(p, m)$. 
Define an indicator random variable:

\[ I_i = \begin{cases} 
1 & \text{if } Z_i = m \\
0 & \text{otherwise}
\end{cases} \]

The sum \( S = \sum_{i=1}^{n} I_i \) is a binomial random variable with parameter \( (p^m, n) \).

\[ P(\text{success}) = P(S \geq m) = 1 - F(m; n, p^m), \]

where \( F(m; n, p^m) \) is the Cumulative Distribution Function of \( S \). To make the above relation more explicit, we make use of the multiplicative form of Chernoff bound (relative error): \[ \text{If } X < (1-\delta)\mu < e^{-\frac{\mu\delta^2}{3}}, \quad \delta \in (0, 1). \] A proof of this bound can be found in the appendix. Setting \( \mu = np^m \) and \( \delta = \frac{np^m - m}{np^m} \) in (4.2), we obtain

\[ P_{\text{suc}} = P(\text{success}) = 1 - F(m; n, p^m) \geq 1 - e^{-\frac{(np^m - m)^2}{2np^m}} \quad (4.3) \]

when \( m \leq np^m \).

To see the relation between sample size, matrix rank and success probability, we use \( r \), the matrix rank, in place of \( m \); and \( s \), the sample size, is equal to \( pn^2 \). Then (4.3) can be written as:

\[ P_{\text{suc}} \geq 1 - e^{-\frac{(ns/n^2)^r - r}{2m(s/n^2)^r}} \quad (4.4) \]

when

\[ r \leq n(s/n^2)^r. \quad (4.5) \]

Suppose the sample size is \( s = cn^\alpha \), where \( c \) is a constant, (4.5) is equivalent to

\[ r \leq c^n n^{(\alpha-2)r+1}. \quad (4.6) \]
For fixed $r$ and large enough $n$, (4.5) can be satisfied if

$$\alpha \geq 2 - 1/r.$$  \hfill (4.7)

We can see that the success probability also becomes increasingly large if (4.7) is satisfied:

$$P_{suc} \geq 1 - e^{-\frac{(n(s/n^2)^r-r)^2}{2n(s/n^2)^r}} = 1 - e^{\frac{2n(s/n^2)^r+\frac{r^2}{an(s/n^2)^r}}{2}} \geq 1 - e^{\frac{2n^{(a-2)r+1-r}}{2}}.$$  \hfill (4.8)

Thus if the sampling size obeys (4.7) is satisfied, we have a high probability of successfully finding a dense $r \times r$ sub-matrix after reordering the original matrix.

### 4.4 A Numerical Example

Here we illustrate the effectiveness of the condensation process by means of a numerical example. Given a $200 \times 200$ matrix sampled at various densities in a uniformly random manner, the condensation process is applied in each case with sub-matrix size $m = 20, 40, 60, 80, 100$. The densities in these sub-matrices after condensation are shown in Figure 4-1, which suggests that the condensation process significantly increases the sampling density at certain part of the matrix. This plot is from a single instance experiment but the trend is already clear. For example, the density is raised by almost 0.2 on the $20 \times 20$ condensed sub-matrix.
Figure 4-1: Condensation on a 200x200 matrix at various sampling densities and sub-matrix sizes
Chapter 5

Mixed Algorithm

In this last chapter, we tie up various parts of the thesis into a new algorithm. We will demonstrate the power of this new algorithm by comparing it with existing methods and show that it can give exact or highly accurate completion of large matrices at low sampling densities where no existing method could achieve accurate completion.

5.1 Motivation and the Mixed Algorithm

Nuclear norm minimization as a heuristic method has the intrinsic drawback that the nuclear norm may be a bad approximation of the rank over certain feasible set. For example, when the sampling density falls below certain threshold, the optimal solution to the nuclear norm minimization problem fails to be minimum rank. (See Figure 3-5 and 3-8.) On the other hand, the local algorithm produces the minimum rank matrix requiring much lower threshold, provided the matrix rank is given, which unfortunately is often unrealistic to know in advance. To make use of both approaches and avoid their shortcomings, we propose the following algorithm.

1. Step 1: Condensation. By reordering the rows and columns of the matrix as described in Chapter 4, one can condense the sampled entries in the upper-left corner.

2. Step 2: Nuclear norm minimization via IPMSDP. Apply IPMSDP to the upper-
left block, which now has a much higher density. Compute the rank of the block via SVD on the completed upper-left block.

3. Step 3: Local completion. Permute back to the original order and use local algorithm to complete the matrix using the rank computed from Step 2.

5.2 Analysis and Discussion

Given a matrix $M \in \mathbb{R}^{n \times n}$, there are a few parameters in this algorithm that warrant some discussion.

First, we consider the choice of $m$, the condensation corner block size in Step 1. (See Section 4.2) In order for this algorithm to work, $m$ has to be greater than or equal to $r$, the rank of the matrix; otherwise it is impossible to get the rank of the matrix in Step 2. Now suppose $m = r$, we have shown in Proposition 8 that the sample density of the top left corner sub-matrix of dimension $r \times r$ after condensation approaches 1 as the dimension of the matrix increases. Also in Section 4.3 we see that the probability of such a $r \times r$ sub-matrix being completely filled with sampled entries after condensation approaches 1, provided enough entries are sampled (See (4.7)). Such analysis suggests for large enough matrix size, we can obtain the rank of the matrix with high probability just by condensation. Step 2 of the above algorithm is therefore likely to recover the upper-left block exactly. However, the choice of $m = r$ is not practical because we do not know $r$ in Step 1 when we perform condensation and choosing a small $m$ runs the risk of $m < r$ with high probability. Even if $m \geq r$ and Step 2 is finished, the number of entries fully recovered in the upper-left block, namely $m^2$, is small. Thus after permutation in Step 3, the known entry density is raised by only a small amount (less than $m^2/n^2$), which does not help the local algorithm to complete the entire matrix. On the other hand, a large $m$ leads to only a small increase in the density of the condensed corner block. (See Figure 4-1 for an example) This means the IPMSDP algorithm in Step 2 has a higher probability to fail to recover the exact upper-left corner block hence the mixed algorithm fails to recover the entire matrix. Essentially, the choice of $m$ has to balance the trade-off
of having higher sampling density in Step 2 for IPMSDP and in Step 3 for the local algorithm.

**Remarks 5.2.1.** Once the upper-left block is completed, the rank of the matrix is known; and the matrix can be completed generically. It is possible to do further permutations and obtain the completion of the entire matrix using nuclear norm minimization; however, the local algorithm we proposed lends itself well here as the rank of the matrix is known.

Second, we consider the choice of the neighborhood sizes in the local algorithm in Step 3. As discussed in Chapter 2 and Section 3.4, larger neighborhood sizes result in higher completion rates in the local algorithm while the run time depends on the sampling density (See Figure 3-7). It is therefore tempting to set large neighborhood sizes. But the locality property of the algorithm can be destroyed by large neighborhood sizes, which is undesirable if we wish to run the local algorithm in parallel, for example.

### 5.3 A Numerical Example

Experiments are carried out on rank 2 200 × 200 matrices \((n = 200)\) with various sampling densities. The condensed block size is set to be \(80 \times 80\) \((m = 80)\). And the completion error are in terms of the 2-norm of the difference between the completed matrix and the true matrix. SVT is used to compare with the Mixed Algorithm and the results averaged over 5 instances are shown in Table 5.1. It shows that the mixed algorithm can solve the LRMC problem more accurately than SVT in this case.

<table>
<thead>
<tr>
<th>Sampling density</th>
<th>0.12</th>
<th>0.13</th>
<th>0.14</th>
<th>0.15</th>
<th>0.18</th>
<th>0.20</th>
<th>0.23</th>
<th>0.25</th>
<th>0.28</th>
<th>0.30</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVT</td>
<td>18.5006</td>
<td>16.4281</td>
<td>5.5544</td>
<td>2.3809</td>
<td>1.1145</td>
<td>0.7284</td>
<td>0.2486</td>
<td>0.2352</td>
<td>0.0494</td>
<td>0.0137</td>
</tr>
<tr>
<td>Mixed</td>
<td>0.025</td>
<td>0.0348</td>
<td>1.8e-4</td>
<td>3.2e-4</td>
<td>7.8e-4</td>
<td>9.9e-5</td>
<td>2.2e-5</td>
<td>1.5e-6</td>
<td>1.28e-4</td>
<td>1.22e-5</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusion

We are concerned with the problem of completing a partially observed matrix so that the completed matrix has the lowest rank. The rank minimization problem is NP-hard but effective heuristics such as the nuclear norm minimization have been proposed to approximately solve the rank minimization problem. In this thesis we propose a local completion algorithm that has a few advantages over the existing methods:

1. It recovers the original low rank matrix exactly if the rank is known and enough entries are sampled. Notice the required number of samples is smaller than the existing methods if large enough search neighborhood is used. (See Section 3.4)

2. The computation time is comparable with that of the Interior Point Methods for Semidefinite Program. (See Figure 3-7)

3. The locality property can be exploited to further speed up the completion by doing the computation in parallel.

We have also compared the existing algorithms and studied the effects of additional sampling structures on the completion rate. In particular, the Interior Point Methods for Semidefinite Programming (IPMSDP) approach has the advantage of fast convergence and good accuracy thanks to the standard solvers. But IPMSDP are limited to small problem sizes. Singular Value Thresholding (SVT) and Sub-gradient
Projection (SGP) can handle much larger problem sizes. SGP has slow convergence rate compared to SVT. But both SGP and SVT require careful selection of parameters such as step sizes. As for the additional sampling structures, we find that the random matching sampling increases recovery rate than uniform random sampling. And we also find that the block sampling schemes tend to result in lower recovery accuracy than unrestricted random sampling. In the block sampling schemes, the star structure shows slightly better recovery accuracy than the tree structure.

We also provide a new semidefinite formulation for a particular block sampling structure. The new formulation reduces the size of the constraint semidefinite matrix by a factor of 1.5.

Finally, we propose a technique called condensation for increasing sampling density in certain part of the matrix by row and column re-ordering. We show that the condensation technique is not only effective in practice but also asymptotically. Furthermore, this can be used as a pre-processing technique in the matrix completion algorithms. One example is to use IPMSDP to complete the condensed sub-matrix and obtain the matrix rank; then use the local algorithm to complete the rest of the matrix. This is the mixed algorithm described in Chapter 5. We also show that it has the advantage of accurate completion over existing methods on large matrices when the sampling density is low.

6.1 Future Work

Although the local algorithm possesses many desirable properties, it still faces the main drawback of numerical instability due to the matrix inverse operations. This problem is worsened when the matrix rank is large. Future work includes avoiding inverting matrices yet preserving the locality property of the algorithm.

Another direction of future work could be addressing the problem of low rank matrix completion using noisy samples of the matrix.
Appendix A

Proof of the multiplicative form of Chernoff bound (relative error)

Proposition 9. Let $X_1, X_2, \ldots X_n$ be independent identically distributed Bernoulli random variables of parameter $p$. Define $X = \sum_{i=1}^{n} X_i$. Let $\mu$ be the expected value of $X$. Then

$$P(X < (1 - \delta)\mu) < e^{-\frac{\delta^2 \mu}{2}}, \quad \delta \in (0, 1) \quad (A.1)$$

Proof.

$$P(X \leq (1 - \delta)\mu) = P(e^{-tX} \geq e^{-t(1-\delta)\mu}) \quad (A.2)$$

$$\leq \min_{t>0} e^{t(1-\delta)\mu} E[e^{-tX}] \quad (A.3)$$

$$= \min_{t>0} e^{t(1-\delta)\mu} (pe^{-t} + 1 - p)^n \quad (A.4)$$

$$= \min_{t>0} e^{t(1-\delta)\mu} (1 - p(1 - e^{-t}))^n \quad (A.5)$$

$$\leq \min_{t>0} e^{t(1-\delta)\mu} e^{(e^{-t}-1)np} \quad (A.6)$$

$$= \min_{t>0} e^{t(1-\delta)\mu + (e^{-t}-1)\mu} \quad (A.7)$$

The first inequality follows from Markov's inequality. The second inequality follows from setting $x = p(1 - e^{-t})$ in the inequality $1 - x \leq e^{-x}$. Now minimize the last
expression over $t > 0$ by setting to 0 the exponent:

$$-\mu e^{-t} + \mu (1 - \delta) = 0$$

We get $t = -\log(1 - \delta)$. Substitute into (A.7) we obtain

$$P(X \leq (1 - \delta)\mu) \leq e^{-\mu[\delta + (1 - \delta)\log(1 - \delta)]} \leq e^{-\frac{\mu\delta^2}{2}}$$ (A.9)

The second inequality can be seen by writing the Taylor expansion of $\log(1 - \delta)$ around 0:

$$\delta + (1 - \delta)\log(1 - \delta) = \delta + (1 - \delta)(-\delta - \delta^2/2 - \delta^3/3 + \cdots)$$

$$= -\delta^2/2 - \delta^3/3 - \delta^4/4 - \cdots$$

$$+ \delta^2 + \delta^3/2 + \delta^4/3 + \cdots$$

$$= \delta^2/2 + \delta^3/6 + \cdots$$
Appendix B

Local Algorithm Codes

The code for rankKLocalVN3.c

```c
#include "mex.h"
#include "matrix.h"
#include <stdlib.h>
/*use pseudo inverses to avoid Inf and NaN*/
/*function to complete the given entry*/
void completeEntry(double *output, int m, bool *binaryA, int *centerIJ, int Rk, int *Jtrack, double *combR,
   int Mcomb, int combIter){
   int i,j,J,I;
   double *Aa, *b, *c, result, *cA, *tolp;
   mxArray *prhsl[2], *plhsl[1], *prhs2[2], *plhs2[2];
   prhsl[0]=mxCreateDoubleMatrix(Rk, Rk, mxREAL);
   Aa=mxGetPr(prhsl[0]);
   prhsl[1]=mxCreateDoubleMatrix(1, 1, mxREAL);
   tolp=mxGetPr(prhsl[1]);
   tolp[0]=pow(10,-8);
   for(j=0;j<Rk;j++){
      J=Jtrack[j];
      for(i=0; i<Rk; i++){
         I=(int)combR[combIter+i*Mcomb];
         Aa[i+j*Rk]=output[I+J*m];
      }
   }
   mexCallMATLAB(1, plhsl, 2, prhsl, "pinv");
   mxDestroyArray(prhsl[0]);
   mxDestroyArray(prhsl[1]);
   /*get c*/
   prhs2[0]=mxCreateDoubleMatrix(1, Rk, mxREAL);
   c=mxGetPr(prhs2[0]);
   for(i=Rk-1;i>=0; i--){
      J=Jtrack[i];
      c[i]=output[I+J*m];
   }
   prhs2[1]=plhsl[0];
   mexCallMATLAB(1, plhs2, 2, prhs2, "mtimes");
   mxDestroyArray(prhs2[0]);
   /*get b*/
   cA=mxGetPr(plhs2[0]);
   b=mxMalloc(Rk, sizeof(double));
   for(i=0;i<Rk; i++){
      I=(int)combR[combIter+i*Mcomb];
      b[i]=output[I+J*m];
   }
   result=0;
   for(i=0; i<Rk; i++){
      result=result+cA[i]*b[i];
   }
}
```
void mexFunction(int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[])
{
  int i, j, m, n, k, ind, iter, maxIter=10000, i1, i2, i3, i4, i5, i6, i7, i8,
  tmp_view_m, i1, i2, i3, i4, i5, i6, i7, i8;
  int ZNew=0, ZOld=0, count=0, fillin=0, Rk=0, neighb=0, locXY=0;
  *nonzRows, *tmp_view, *maxis;
  entryComplete=0, centerIJ[2];
  bool *binaryA, *binaryOrig;
  mxArray *plhs2[1], *prhs2[2], *tmp_viewArr i1;
  if ( mxIsComplex(prhs[0]) || mxIsClass(prhs[0], "sparse") ||
    mxIsChar(prhs[0]) )
    mexErrMsgTxt("Input must be real, full, and nonstring");

  m=mxGetM(prhs[0]);
  n=mxGetN(prhs[0]);
  binaryA=mxGetLogicals(prhs[1]);
  RkP=mxGetPr(prhs[2]);
  neighbP=mxGetPr(prhs[3]);
  Rk=(int)RkP[0];
  neighb=(int)neighbP[0];
  binaryOrig=mxMalloc(m*n, sizeof(bool));
  memcpy(binaryOrig, binaryA, sizeof(bool)*m*n);

  /* output matrix: */
  plhs[0] = mxCreateDoubleMatrix(m, n, mxREAL);
  output=mxGetPr(plhs[0]);
  A=mxGetPr(prhs[0]);
  memcpy(output, A, sizeof(double)*m*n);

  /* count number of zeros in the input matrix*/
  for(j=0;j<m*n;j++){
  if(binaryA[j]==0)
    ZOld++;
  }
  zerosInd=mxMalloc(ZOld, sizeof(int));
  for(j=0;j<m*n;j++){
  if(binaryA[j]==0){
    zerosInd[count]=j;
    count++;
  }
  }
  neighbRows=mxMalloc(2*neighb, sizeof(double));
  Jtrack=mxMalloc(2*neighb, sizeof(int));
  prhs2[1]=mxCreateDoubleMatrix(1,1,mxREAL);
  combKP=mxGetPr(prhs2[1]);
  combKP[0]=(double)Rk;

  for(iter=0; iter<ZOld & iter<maxIter; iter++)
  {
    fillin=0;
    for(i=0; i<ZOld; i++)
    {
      ind=zerosInd[i];
      if(binaryA[ind]==0){

        /* focus on the neighborhood (2*neighb+1)*(2*neighb+1) centered on ind*/
        i=ind/m;
        j=ind%n;
        centerIJ[0]=i;
        centerIJ[1]=j;
        if(j-neighb<0)
          j4=m+(j-neighb);
        else
          j4=j-neighb;
        if(j+neighb>n-1)
          j5=j+neighb-n;
        else
          j5=j+neighb;
        if(i-neighb<0)
          i4=m+(i-neighb);
        else
          i4=i-neighb;

        /* First step: check vertical entries*/
        if(binaryA[ind]==0){

        } else
          "First step: check vertical entries*/
        if(binaryA[ind]==0){

        } else
          "First step: check vertical entries*/
      }
    }
  }
}
/* local algorithm for rank K matrix completion*/
int i, j, m, n, k, ind, iter, maxIter=10000, i1, i2, i3, i4, i5, i6, i7, i8,
  tmp_view_m, i1, i2, i3, i4, i5, i6, i7, i8;
int ZNew=0, ZOld=0, count=0, fillin=0, Rk=0, neighb=0, locXY=0;
  *nonzRows, *tmp_view, *maxis;
  entryComplete=0, centerIJ[2];
  bool *binaryA, *binaryOrig;
  mxArray *plhs2[1], *prhs2[2], *tmp_viewArr i1;
  if ( mxIsComplex(prhs[0]) || mxIsClass(prhs[0], "sparse") ||
    mxIsChar(prhs[0]) )
    mexErrMsgTxt("Input must be real, full, and nonstring");

  m=mxGetM(prhs[0]);
  n=mxGetN(prhs[0]);
  binaryA=mxGetLogicals(prhs[1]);
  RkP=mxGetPr(prhs[2]);
  neighbP=mxGetPr(prhs[3]);
  Rk=(int)RkP[0];
  neighb=(int)neighbP[0];
  binaryOrig=mxMalloc(m*n, sizeof(bool));
  memcpy(binaryOrig, binaryA, sizeof(bool)*m*n);

  /* output matrix: */
  plhs[0] = mxCreateDoubleMatrix(m, n, mxREAL);
  output=mxGetPr(plhs[0]);
  A=mxGetPr(prhs[0]);
  memcpy(output, A, sizeof(double)*m*n);

  /* count number of zeros in the input matrix*/
  for(j=0;j<m*n;j++){
  if(binaryA[j]==0)
    ZOld++;
  }
  zerosInd=mxMalloc(ZOld, sizeof(int));
  for(j=0;j<m*n;j++){
  if(binaryA[j]==0){
    zerosInd[count]=j;
    count++;
  }
  }
  neighbRows=mxMalloc(2*neighb, sizeof(double));
  Jtrack=mxMalloc(2*neighb, sizeof(int));
  prhs2[1]=mxCreateDoubleMatrix(1,1,mxREAL);
  combKP=mxGetPr(prhs2[1]);
  combKP[0]=(double)Rk;

  for(iter=0; iter<ZOld & iter<maxIter; iter++)
  {
    fillin=0;
    for(i=0; i<ZOld; i++)
    {
      ind=zerosInd[i];
      if(binaryA[ind]==0){

        /* focus on the neighborhood (2*neighb+1)*(2*neighb+1) centered on ind*/
        i=ind/m;
        j=ind%n;
        centerIJ[0]=i;
        centerIJ[1]=j;
        /*left most*/
        if(j-neighb<0)
          j4=m+(j-neighb);
        else
          j4=j-neighb;
        /*right most*/
        if(j+neighb>n-1)
          j5=j+neighb-n;
        else
          j5=j+neighb;
        /*upper most*/
        if(i-neighb<0)
          i4=m+(i-neighb);
        else
          i4=i-neighb;

        /*First step: check vertical entries*/
        if(binaryA[ind]==0){

        } else
          "First step: check vertical entries*/
        if(binaryA[ind]==0){

        } else
          "First step: check vertical entries*/
      }
    }
  }
}
for(locXY=1;locXY<=neighb;locXY++){  
//upper center/
if(i-locXY>O)
  i2=m+(i-locXY);
else
  i2=i-locXY;
if(binaryOrig[i2+j2*m]==1){
  neighbRows[neighbRowsCnt]=(double)i2;
  neighbRowsCnt++;
}
/*lower center*/
if(i+locXY>m-l)
  i7=i+locXY-m;
else
  i7=i+locXY;
if(binaryOrig[i7+j7*m]==1){
  neighbRows[neighbRowsCnt]=(double)i7;
  neighbRowsCnt++;
}
if(neighbRowsCnt>=Rk){
  prhs2[0]=mxCreateDoubleMatrix(neighbRowsCnt,1,mxREAL);
  nonzRows=mxGetPr(prhs2[0]);
  memcpy(nonzRows,neighbRows,sizeof(double)*neighbRowsCnt);
  /*get all the combinations of the rows to check*/
  mexCallMATLAB(1,plhs2,2,prhs2,"combntns");
  mxDestroyArray(prhs2[0]);
  Mcomb=mxGetM(plhs2[0]);
  Ncomb=mxGetN(plhs2[0]);
  combR=mxGetPr(plhs2[0]);
  /*check through the combinations*/
  for(combIter=0;combIter<Mcomb;combIter++){
    cnt=0;
    /*left half of the neighborhood*/
    for(J=j4; J<j4+neighb; J++){
      subcnt=0;
      /*only go on if the horizontal left entry nonzero*/
      if(binaryOrig[(JYN)*m+i]==1){
        for(Iloc=0; Iloc<Ncomb; Iloc++){
          if(binaryOrig[(JYN)*m+I]==1){
            subcnt++;
          }
        }
        if(subcnt==Rk){
          Jtrack[cnt]=JYN;
          cnt++;
        }
      }
    }
    /*right half of the neighborhood*/
    for(J=j4+neighb+1; J<j4+2*neighb+1; J++){
      subcnt=0;
      /*only go on if the horizontal right entry nonzero*/
      if(binaryOrig[(JYN)*m+i]==1){
        for(Iloc=0; Iloc<Ncomb; Iloc++){
          if(binaryOrig[(JYN)*m+I]==1){
            subcnt++;
          }
        }
        if(subcnt==Rk){
          Jtrack[cnt]=JYN;
          cnt++;
        }
      }
    }
    /*check if there exists a successful completion*/
    if(cnt>=Rk){
      fillin++;
      entryComplete=1;
      completeEntry(output,m,binaryA,centerIJ,Rk,Jtrack,combR,Mcomb,combIter);
      break;
    }
  }
}
}
if (fillin==0){
  /*couldn't fill in any entry, terminate*/
  break;
}
  /*mexPrintf("Total:%d, iter:%d\n",ZOld, iter);*/
  mxFree(zerosInd);
  mxFree(neighbRows);
  mxFree(Jtrack);
  mxFree(binaryOrig);
}
Bibliography


