Decomposition Approach for Low-rank Matrix Completion and its Applications

Rick Ma, Nafise Barzigar, Aminmohammad Roozgard, and Samuel Cheng

***Abstract*— In this paper, we describe a low-rank matrix completion method based on matrix decomposition. An incomplete matrix is decomposed into sub-matrices which are filled with a proposed trimming step and then are recombined to form a low-rank completed matrix. The divide-and-conquer approach can significantly reduce computation complexity and storage requirement. Moreover, the proposed decomposition method can be naturally incorporated into any existing matrix completion methods to attain further gain. Unlike most existing approaches, the proposed method is not based on norm minimization nor on SVD decomposition. This makes it possible to be applied beyond real domain and can be used in arbitrary fields, including finite fields. The effectiveness of our proposed method is demonstrated through extensive numerical results on randomly generated and real matrix completion problems and a concrete application– video denoising. The numerical experiments show that the algorithm can reliably solve a wide range of problems at a speed significantly faster than recent algorithms. In the proposed denoising approach, we present a patch-based video denoising algorithm by grouping similar patches and then formulating the problem of removing noise using a decomposition approach for low-rank matrix completion. Experiments show that the proposed approach robustly removes mixed noise such as impulsive noise, Poisson noise, and Gaussian noise from any natural noisy video. Moreover, our approach outperforms state-of-the-art denoising techniques such as VBM3D and 3DWTF in terms of both time and quality. Our technique also achieves significant improvement over time against other matrix completion methods.**

I. Introduction

The recovery of an unknown low-rank or approximately low-rank matrix from very limited information is a recent fast growing interest. Consider a large matrix with only a small portion of known entry, an interesting problem is to fill the missing entry assuming the matrix has low-rank. The problem, which is referred to as matrix completion, or more precisely low-rank matrix completion, has gained increasing interests in research communities in recent years. So far, this problem has been studied in many applications such as collaborative filtering [1], system identification [2], computer vision [3], machine learning [4]–[6], global positioning [7] and remote sensing [8]. An example is the famous Netflix challenge where a huge matrix is used to represent the rating of a movie given by a user. Of course, a typical user will only rate very few movie titles. Therefore, an algorithm will be needed to complete the matrix to predict the ratings of all movies among all users.

It has been shown theoretically that under certain assumptions the matrix can be recovered with very high accuracy [9]–[11]. Their approaches convert the rank minimization problem into a nuclear norm minimization problem instead and thus can be solved using semidefinite program (SDP). However, the complexity grows rather rapidly with the size of the matrix n ($\sim n^3$). Candes and Recht [9] showed that one can perfectly recover most low-rank matrices from what appears to be an in complete set of entries, and they proved in some condition, most $n \times n$ matrices of rank r can be perfectly recovered by solving a simple convex optimization program. Also, the authors claimed that their method is accurate even when the few observed entries are corrupted by a small amount of noise. In another work, the problem of recovering low-rank and sparse matrices using a greedy algorithm was discussed for large matrix sizes [12]. Several efficient algorithms have been proposed including Singular Value Thresholding (SVT) [13], Atomic Decomposition for Minimum Rank Approximation (ADMiRA) [14], Fixed Point Continuation with Approximate (FPCA) [15], Accelerated Proximal Gradient (APG) [16], Subspace Evolution and Transfer (SET) [17], Singular Value Projection (SVP) [18], OptSpace [11], and LMaFit [19], where OptSpace and SET are based on Grassmann manifold optimization, SVT and SVP uses iterative hard thresholding (IHT) to facilitate matrix shrinkage, FPCA utilizes Bregman iterative algorithm and Monte Carlo approximate SVD, and LMaFit adopts successive over-relaxation (SOR).

In this paper, we propose a decomposition method to allow very efficient divide-and-conquer approach when known entries are relatively very few. A simple "trimming" method is proposed to recover the decomposed "cluster" matrix. However, the decomposition method can also be combined with any other existing matrix completion techniques to yield further gain. One advantage of the proposed approach is that unlike most existing approaches it does not utilize SVD but only relies on basic vector operations. Therefore, the approach is immediately applicable to matrices of any field (including finite field matrices). This opens up opportunities for new applications. To compare with other methods, we apply to video denoising. The proposed method significantly outperforms the-state-of-the-art denoising techniques such as VBM3D. We also compare it to other denoising techniques using matrix completion. Our method results in comparable performance with significantly lower computation complexity.

The rest of the paper is organized as follows. In the next section, we will introduce the concept of our matrix completion method and the inference algorithm. We further apply our proposed matrix completion to video denoising and will show our simulation results in Section IV, followed by a brief conclusion in Section V.

II. Proposed Matrix Completion Method

This section describes the rationale and the implementation details of our proposed matrix completion method. We will introduce the problem precisely in Section II-A, and present several properties to be used in the later sections. Sections II-B and II-C will describe the decomposition procedures and present our main results. Section II-D will describe the trimming process to recover the decomposed "cluster" matrix.

A. Minimum Rank, Junk Rows and Columns, Equivalence

Let us start with a few words on notation. When things are clear, lines of partition in matrices may be omitted. The ? sign may represent an unknown entry, a row or a column of unknown entry, or a matrix of unknown entry. Similarly, this rule applies to the 0 sign as well. To increase readability, we use bold font for vectors and norman font for other scalars and matrices.

1) Minimum Rank of Incomplete Matrix: Given a finite size matrix M over field $\mathbb{F}$ to be completed, let

$$S(M) = \{\bar{M} | \bar{M} \text{ is a completion of } M\}. \tag{1}$$

If M is already completed, then $S(M) = \{M\}$. We define

$$mr(M) \triangleq \min_{\bar{M} \in S(M)} \text{rank}\bar{M}. \tag{2}$$

Such minimum exists because $\text{rank}(S(M)) \subset \mathbb{N}$ and hence $\exists \bar{M} \in S(M)$ such that

$$\text{rank}\bar{M} = mrM. \tag{3}$$

If $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, then

$$\exists \bar{A} \in S(A) \text{ such that } mr(M) = mr \begin{pmatrix} \bar{A} & B \\ C & D \end{pmatrix}, \tag{4}$$

as we can always find $\bar{A}$ from $\bar{M}$ in (3). We list in the following other obvious properties of $mr(M)$ that will be used later on:

$$mr(M) \leq \text{rank}\bar{M}, \qquad \forall \bar{M} \in S(M), \tag{5}$$

$$mr(M) \leq mr(P) \text{ if } P \text{ is any partial completion of } M, \tag{6}$$

$$mr([A|B]) \leq mrA + mrB, \tag{7}$$

$$mrM^t = mrM, \tag{8}$$

$$mr \begin{pmatrix} A & B \\ C & D \end{pmatrix} \geq mr(A), \tag{9}$$

$$mrM = mrN \text{ if } N \text{ can be obtained from } M \text{ through interchanging of columns/rows.} \tag{10}$$

2) Junk Row and Junk Column:

Definition 1: A row (column) contains entirely either zero or unknown will be referred as a junk row (column).

Certainly, we have

$$mr(\mathbf{J}) = 0 \text{ if } \mathbf{J} \text{ is a junk row (column),} \tag{11}$$

since we can always complete $\mathbf{J}$ entirely by zero entries.

Theorem 1: Let $M = [\mathbf{J}|N]$ where $\mathbf{J}$ is a junk column, then $mrM = mrN$.

Proof: By (7), we have $mrM \leq mr\mathbf{J} + mrN = mrN$. On the other hand $mrM \geq mrN$ by (9). Hence $mrM = mrN$. ∎

Thanks to (8), we have the following corollary:

Corollary 1: $mr\left(\begin{bmatrix} \mathbf{J} \\ N \end{bmatrix}\right) = mrN$ if $\mathbf{J}$ is a junk row.

In essence, junk rows and columns are redundant as their existence does not increase the minimum rank of an incomplete matrix.

3) Equivalence: We say M is equivalent to N and write $M \sim N$ iff N can be obtained from M through row interchanging, column interchanging, and junk rows and columns deletion and augmentation. By Theorem 1 and (10), we have

$$M \sim N \Rightarrow mrM = mrN. \tag{12}$$

B. Unknown-diagonalization

Define

$$\text{u-diag}(B_1, B_2, \cdots, B_n) \triangleq \begin{pmatrix} B_1 & ? & \cdots & ? \\ ? & B_2 & \cdots & ? \\ \vdots & & \ddots & \\ ? & ? & \cdots & B_n \end{pmatrix}. \tag{13}$$

We say M is u-diagonalizable[1] iff $M \sim \text{u-diag}(A, B)$, and both A and B contain at least one nonzero known entry. We want to know if an incomplete matrix is u-diagonalizable as one can complete a u-diagonal matrix efficiently with the following theorem.

Theorem 2: Let $M \sim \text{u-diag}(B_1, \cdots, B_n)$, then $mrM = max_{1 \leq i \leq n} mr(B_i)$.

Proof: By (12) and induction, all we need to show is when $M = \text{u-diag}(A, B)$. Let $mrA = a$ and $mrB = b$, by (9) we have

$$mrM \geq max(a, b) \tag{14}$$

Let $\bar{A}$, $\bar{B}$ be completions of A and B, respectively such that $\text{rank}\bar{A} = a$ and $\text{rank}\bar{B} = b$ (c.f. (3)), then by (6),

$$mrM \leq mr\left(\text{u-diag}(\bar{A}, \bar{B})\right) \tag{15}$$

Combining (14) and (15), we conclude that

$$max(a, b) \leq mrM \leq mr\left(\text{u-diag}(\bar{A}, \bar{B})\right). \tag{16}$$

By (10), we can simply assume the first a columns of $\bar{A}$ form a basis of $Col\bar{A}$; and the first b columns of $\bar{B}$ form a basis of $Col\bar{B}$. Without loss of generality, let us assume $a \geq b$. We complete the matrix u-diag$(\bar{A}, \bar{B})$ by filling up the columns:

$$\begin{bmatrix} \bar{\mathbf{A}}_i \\ ? \end{bmatrix} \Rightarrow \begin{bmatrix} \bar{\mathbf{A}}_i \\ \bar{\mathbf{B}}_i \end{bmatrix},$$

$$\begin{bmatrix} ? \\ \bar{\mathbf{B}}_i \end{bmatrix} \Rightarrow \begin{bmatrix} \bar{\mathbf{A}}_i \\ \bar{\mathbf{B}}_i \end{bmatrix}, \qquad \text{for } 1 \leq i \leq b, \tag{17}$$

$$\begin{bmatrix} \bar{\mathbf{A}}_i \\ ? \end{bmatrix} \Rightarrow \begin{bmatrix} \bar{\mathbf{A}}_i \\ \bar{0} \end{bmatrix}, \qquad \text{for } b+1 \leq i \leq a. \tag{18}$$

[1]We use u in u-diagonalizable as an abbreviation for *unknown*.

For $i > a$, we make use of the fact that $\bar{\mathbf{A}}_i$ is a linear combination of $\bar{\mathbf{A}}_k | k \leq a\}$. Let $\bar{\mathbf{A}}_i = \sum_{k=1}^{a} a_{i,k}\bar{\mathbf{A}}_k$. We fill

$$\begin{bmatrix} \bar{\mathbf{A}}_i \\ ? \end{bmatrix} \Rightarrow \begin{bmatrix} \bar{\mathbf{A}}_i \\ \sum_{k=1}^{a} a_{i,k}\bar{\mathbf{B}}_k \end{bmatrix}, \qquad \text{for } i > a, \tag{19}$$

where $\bar{\mathbf{B}}_k = 0$, $b < k \leq a$. Similarly,

$$\begin{bmatrix} ? \\ \bar{\mathbf{B}}_i \end{bmatrix} \Rightarrow \begin{bmatrix} \sum_{k=1}^{b} b_{i,k}\bar{\mathbf{A}}_k \\ \bar{\mathbf{B}}_i \end{bmatrix}, \qquad \text{for } i > b, \tag{20}$$

where $\bar{\mathbf{B}}_i = \sum_{k=1}^{b} b_{i,k}\bar{\mathbf{B}}_k$.

Now we have a completed u-diag$(\bar{A}, \bar{B})$ and the first a of its columns $\begin{bmatrix} \bar{\mathbf{A}}_1 \\ \bar{\mathbf{B}}_1 \end{bmatrix}$, $\begin{bmatrix} \bar{\mathbf{A}}_2 \\ \bar{\mathbf{B}}_2 \end{bmatrix}, \cdots, \begin{bmatrix} \bar{\mathbf{A}}_b \\ \bar{\mathbf{B}}_b \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{A}}_{b+1} \\ \mathbf{0} \end{bmatrix}, \cdots, \begin{bmatrix} \bar{\mathbf{A}}_a \\ \mathbf{0} \end{bmatrix}$ form a basis for its column space. Hence it has rank a. By (5), $mr\left(\text{u-diag}(\bar{A}, \bar{B})\right) \leq a$ and hence by (16) and $max(a, b) = a$, we get $mrM = a = mr\left(\text{u-diag}(A, B)\right)$ as wanted. ∎

Remark 1: Suppose A has been completed by $\bar{A}$ and $a =$ rank$(\bar{A})$. Then if the number of column of $B = n \leq a$, then we can complete B arbitrarily without increasing the rank of the output and fill the unknowns in B with (17) alone as if $b = n$. More generally, given $M =$ u-diag$(\bar{A}, B)$ with $\bar{A}$ is completed. Then the completing process of B can be stopped once we know that the final rank$B$ will be no greater than rank$\bar{A}$ no matter how we fill in the remaining unknowns in B. For example, if $\mathrm{s}(B) \leq$ rank$\bar{A}$, where

$$\mathrm{s}(B) \triangleq \min(\text{number of column of } B, \text{number of row of } B), \tag{21}$$

then we can complete B arbitrarily to start with.

We will call the submatrices $B_1, B_2, \cdots, B_n$ in Theorem 2 as clusters, where a cluster is a matrix that cannot be further u-diagonalized. We will defer to Section II-D for discussion of how a cluster can be filled. In the following, we will first present a more general decomposition that can be applied to matrices that are not u-diagonalizable.

C. Sub Unknown-diagonalization

Definition 2: A matrix C, not u-diagonalizable, becomes u-diagonalizable after deleting a row or a column is called sub u-diagonalizable. The row (column) is called conjoined row (column).

Before we describe this main theorem in this section, we need to introduce the following lemma.

Lemma 1: $mr[\mathbf{v}|M] = mrM$ if $\forall \bar{M} \in S(M)$, $Col(\bar{M}) \cap S(\mathbf{v}) \neq \phi$,.

Proof: By (1.9) we already have $mr[\mathbf{v}|M] \geq mrM$. Let $\bar{M} \in S(M)$ such that rank$\bar{M} = mrM$ (c.f. (2)). Then pick a $\bar{\mathbf{v}} \in S(\mathbf{v}) \cap Col(\bar{M})$. From (5), we get $mr[\mathbf{v}|M] \leq$ rank$[\bar{\mathbf{v}}|\bar{M}] =$ rank$\bar{M} = mrM$. ∎

Theorem 3: Let $M = \begin{pmatrix} \mathbf{v}_1 & B_1 & ? & \cdots & ? \\ \mathbf{v}_2 & ? & B_2 & \cdots & ? \\ \vdots & \vdots & & \ddots & \\ \mathbf{v}_n & ? & ? & \cdots & B_n \end{pmatrix}$,

where B_i are matrices, $\mathbf{v}_i$ are vectors that $\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix}$ is not a junk column[2], then $mrM = \max_i mr \left(\begin{array}{c|c} \mathbf{v}_i & B_i \\ 1 & \mathbf{?} \end{array} \right)$.

Proof: We will show the case when $n = 2$; cases of higher n are easy to be generalized. Let $M = \begin{pmatrix} \mathbf{u} & A & ? \\ \mathbf{v} & ? & B \end{pmatrix}$ with $\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}$ is not a junk column. If $\mathbf{u}$ is not a junk column, then $\mathbf{u}$ contains a nonzero element. By Lemma 1 (row version),

$$mr \left(\begin{array}{c|c} \mathbf{u} & A \\ 1 & \mathbf{?} \end{array} \right) = mr[\mathbf{u}|A] \leqslant mrM, \text{ (c.f. (9))}. \tag{22}$$

If $\mathbf{u}$ is a junk column, then $\mathbf{v}$ contains a nonzero element λ. Obviously

$$mr \left(\begin{array}{c|c} \mathbf{u} & A \\ 1 & \mathbf{?} \end{array} \right) = mr \left(\begin{array}{c|c} \mathbf{u} & A \\ \lambda & \mathbf{?} \end{array} \right) \leqslant mrM, \text{ (c.f. (9), (10))}. \tag{23}$$

By symmetry, we also have $mr \left(\begin{array}{c|c} \mathbf{v} & B \\ 1 & \mathbf{?} \end{array} \right) \leqslant mrM$ and hence we have shown

$$mrM \geqslant max(mr \left(\begin{array}{c|c} \mathbf{u} & A \\ 1 & \mathbf{?} \end{array} \right), mr \left(\begin{array}{c|c} \mathbf{v} & B \\ 1 & \mathbf{?} \end{array} \right)). \tag{24}$$

Next, we will complete the proof by showing that $\exists \bar{M} \in S(M)$ s.t.

$$\text{rank}\bar{M} = \max(mr \left(\begin{array}{c|c} \mathbf{u} & A \\ 1 & \mathbf{?} \end{array} \right), mr \left(\begin{array}{c|c} \mathbf{v} & B \\ 1 & \mathbf{?} \end{array} \right)). \tag{25}$$

Let $a = mr \left(\begin{array}{c|c} \mathbf{u} & A \\ 1 & \mathbf{?} \end{array} \right) - 1$ and $b = mr \left(\begin{array}{c|c} \mathbf{v} & B \\ 1 & \mathbf{?} \end{array} \right) - 1$. Without loss of generality, we assume $a \geqslant b$. Let $\left(\begin{array}{c|c} \bar{\mathbf{A}}_0 & \bar{A} \\ 1 & \bar{\mathbf{X}} \end{array} \right) \in S\left(\left(\begin{array}{c|c} \mathbf{u} & A \\ 1 & \mathbf{?} \end{array} \right) \right)$ with rank $a + 1$; $\left(\begin{array}{c|c} \bar{\mathbf{B}}_0 & \bar{B} \\ 1 & \bar{\mathbf{Y}} \end{array} \right) \in S\left(\left(\begin{array}{c|c} \mathbf{v} & B \\ 1 & \mathbf{?} \end{array} \right) \right)$ with rank $b + 1$.

By (10), we may assume $\left\{ \begin{bmatrix} \bar{\mathbf{A}}_0 \\ 1 \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{A}}_1 \\ \bar{X}_1 \end{bmatrix}, \cdots, \begin{bmatrix} \bar{\mathbf{A}}_a \\ \bar{X}_a \end{bmatrix} \right\}$ forms a basis for $Col \left(\begin{array}{c|c} \bar{\mathbf{A}}_0 & \bar{A} \\ 1 & \bar{\mathbf{X}} \end{array} \right)$; $\left\{ \begin{bmatrix} \bar{\mathbf{B}}_0 \\ 1 \end{bmatrix}, \begin{bmatrix} \bar{\mathbf{B}}_1 \\ \bar{Y}_1 \end{bmatrix}, \cdots, \begin{bmatrix} \bar{\mathbf{B}}_b \\ \bar{Y}_b \end{bmatrix} \right\}$ forms a basis for $Col \left(\begin{array}{c|c} \bar{\mathbf{B}}_0 & \bar{B} \\ 1 & \bar{\mathbf{Y}} \end{array} \right)$.

Then we complete the matrix $\begin{pmatrix} \bar{\mathbf{A}}_0 & \bar{A} & ? \\ \bar{\mathbf{B}}_0 & ? & \bar{B} \end{pmatrix}$ to $\bar{M}$ by (17)-(20) but with k started from 0. The first $a + 1$ columns of $\bar{M}$ will form a basis for $Col(\bar{M})$ and hence rank$\bar{M} = a + 1$ and we have (25).

∎

[2]Note that $\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix}$ is a conjoined column and M becomes u-diagonalizable without it.

1) How to decompose sub u-diagonalizable matrix:

Definition 3: Given two vectors $\mathbf{v}$ and $\mathbf{w}$ of same length, we say $\mathbf{v}$ is a *donor* for $\mathbf{w}$ ($\mathbf{v} \succeq \mathbf{w}$) iff all of the known positions of $\mathbf{w}$ are also known positions in $\mathbf{v}$. In order words, after some row interchanging $[\mathbf{w}|\mathbf{v}] = \begin{pmatrix} \bar{\mathbf{r}} & \bar{\mathbf{d}} \\ \mathbf{?} & \mathbf{n} \end{pmatrix}$ with $\bar{\mathbf{r}}$ and $\bar{\mathbf{d}}$ contain only known elements. Clearly, $\mathbf{v} \succeq \mathbf{w}$ and $\mathbf{w} \succeq \mathbf{u}$ imply $\mathbf{v} \succeq \mathbf{u}$. However, if $\mathbf{v} \succeq \mathbf{w}$ and $\mathbf{w} \succeq \mathbf{v}$, we do not have $\mathbf{v} = \mathbf{w}$. Vectors $\mathbf{v}$ and $\mathbf{u}$ are said to be *comparable* if either $\mathbf{v} \succeq \mathbf{w}$ or $\mathbf{w} \succeq \mathbf{v}$.

Theorem 4: Conjoined row (column) does not have donors among other rows of the sub u-diag matrix.

Proof: Let C be the sub u-diag matrix, then it must have the following structure (after some row and column interchanging): $C = \begin{pmatrix} A & ? \\ ? & B \\ \mathbf{u} & \mathbf{v} \end{pmatrix}$, where $\mathbf{u}$ cannot be entirely unknown, otherwise C is u-diagonalizable. Now, rows in $[?|B]$ cannot be donors of $[\mathbf{u}|\mathbf{v}]$, the conjoined row. Similarly $\mathbf{v}$ cannot be entirely unknown and hence, rows in $[A|?]$ cannot be donors of $[\mathbf{u}|\mathbf{v}]$ neither. ∎

Therefore if C is a sub u-diagonalizable, we will not miss the chance of decomposing it if we have tested every row and column that does not have a donor. That is to blackout the suspicious row (column) and then carrying out the decomposition mentioned in Section II-B. We would like to call the decomposed components as *sub-clusters*. For example, $\begin{bmatrix} A \\ \mathbf{u} \end{bmatrix}$ and $\begin{bmatrix} B \\ \mathbf{v} \end{bmatrix}$ are sub-clusters of the C in the proposition.

Unlike cluster that cannot be further u-diagonalized, sub-clusters can be sub u-diagonalizable. For example $C = \begin{pmatrix} A & ? & ? \\ \mathbf{u} & \mathbf{v} & \mathbf{?} \\ ? & B & ? \\ \mathbf{?} & \mathbf{x} & \mathbf{y} \\ ? & ? & D \end{pmatrix}$, where both $[\mathbf{u}|\mathbf{v}|\mathbf{?}]$ and $[\mathbf{?}|\mathbf{x}|\mathbf{y}]$ are conjoined rows. In that case, we may first decompose C into sub-clusters $\begin{bmatrix} A \\ \mathbf{u} \end{bmatrix}$ and $\begin{pmatrix} \mathbf{v} & \mathbf{?} \\ B & ? \\ \mathbf{x} & \mathbf{y} \\ ? & D \end{pmatrix}$. Then we may further decompose the later into $\begin{bmatrix} \mathbf{v} \\ B \\ \mathbf{x} \end{bmatrix}$ and $\begin{bmatrix} \mathbf{y} \\ D \end{bmatrix}$, if necessary.

Remark 2: There is a trade-off between the complexity of searching for conjoined rows (columns) and the gain obtained for further decomposition. Even though a conjoined row cannot have any donors from Theorem 4, verifying that for a row can be computationally consuming itself.

However, as one can easily see that the row with the maximum number of known elements will generally have no donor[3]. Therefore, such rows are usually good candidates of a conjoined row and they are easy to find. Thus, in our implementation, we only check such rows for the simplicity of implementation. However, other variation of conjoined row searching can be easily adopted to the sub u-diagonalization approach.

[3]It is not true of course when another row has the same maximum number of known elements and exactly the same locations of known elements.

D. Trimming

Theorem 5: Let $\mathbf{M}_{d_i}$ be the d_i-th column of M and donor (c.f. def 3) of a vector $\mathbf{v}$ for $1 \leq i \leq t$, such that after some row interchanging, $[\mathbf{v}|\mathbf{M}_{d_1}\mathbf{M}_{d_2}\cdots\mathbf{M}_{d_t}] = \left(\begin{array}{c|c} \bar{\mathbf{r}} & \bar{D} \\ \hline \mathbf{?} & N \end{array}\right)$ with $\bar{\mathbf{r}}$ and $\bar{D}$ are completed. If $\bar{\mathbf{r}} \in Col(\bar{D})$, then $mr[\mathbf{v}|M] = mrM$.

Proof: Thanks to (10), we may start with $[\mathbf{v}|M] = \left(\begin{array}{c|c} \bar{\mathbf{r}} & \bar{D}A \\ \hline \mathbf{?} & NB \end{array}\right)$.

Pick $a_i \in \mathbb{F}$ such that

$$\bar{\mathbf{r}} = \sum a_i \bar{\mathbf{D}}_i. \tag{26}$$

Then $\forall \bar{M} = \begin{bmatrix} \bar{D}\bar{A} \\ \bar{N}\bar{B} \end{bmatrix} \in S(M)$, we complete $\mathbf{v}$ to

$$\bar{\mathbf{v}} = \begin{bmatrix} \bar{\mathbf{r}} \\ \sum a_i \bar{\mathbf{N}}_i \end{bmatrix} \in S(\mathbf{v}) \cap Col(\bar{M}). \tag{27}$$

(Note that rank$[\bar{\mathbf{v}}|\bar{M}]$ = rank$\bar{M}$.) Now Lemma 1 implies $mr[\mathbf{v}|M] = mrM$. ∎

1) Trimming Process: We test column by column to see if we can make use of Theorem 5 to trim away any column $\mathbf{v}$ from a given matrix, which can be a cluster or sub-cluster mentioned in the previous section. We call this process as *column trimming*. When we find a column satisfying the condition of Theorem 5, we will mark down the dependency relation between it and its donor (i.e. (26)) in order. Then we black it out and go for the next column.

Similarly, we have row trimming. Notice that the trimming process and the sub u-diagonalizable testing can be carried out together, thanks to Theorem 4. The theorem also tells us that trimmed cluster is still a cluster, i.e., not u-diagonalizable. However, it can be a sub-cluster.

An uninterrupted (c.f. Remark 1) trimming process starts with a column trimming followed by a row trimming, or the other way round. Then we carry out these two kinds of trimming one after the other, until there is no more reduction in the matrix. After the trimmed matrix gets completed, we restore, in reverse order, the blackouts with the completed forms given by (27).

We haven't made any approximation in our completion procedure. Our algorithm can be dovetailed with any other method, without compromising the lowest rank they can find for a given matrix.

E. Implementation

Using the results developed in the earlier subsections, we will now present two prototype matrix completion algorithms: one based on unknown diagonalization and the other based on unknown sub-diagonalization. The key steps of the algorithms are summarized in Algorithm 1 and Algorithm 2, and the detail implementation of each step is presented in the following.

Implementation Details:

- $c = ScanCluster(M)$ finds the clustering information and outputs the row and column indexes of each potential

Algorithm 1 U-DiagFillMatrix(M)

Inputs : an incomplete matrix M

Complete the matrix M: given the incomplete matrix M do:

- $c = ScanCluster(M)$
- $M_a = ArrangeMatrix(M, c)$
- **Cluster trimming and partial filling:** for each cluster B_i in the arranged matrix M_a:
  - $M_t^i = Trimming(B_i)$
  - $F_i = FillCluster(B_i, M_t^i)$
  - $Update(M_a, F_i)$
- $M_f = FillOffDiag(M_a, \{M_t^i\}_{i=1}^n, c)$
- $\hat{M} = RevertMatrix(M_f, c)$

Output : a complete matrix $\hat{M}$

Algorithm 2 SubU-DiagFillMatrix(M) (Theorem 3)

Inputs : an incomplete matrix M

Initialize :

- $c = ScanCluster(M)$
- if c shows that more than one cluster are found from the previous step, output $\hat{M} = U\text{-}DiagFillMatrix(M)$ (Algorithm 1) and exit.
- $i = FindConjoinedColIndex(M)$
- $c_v = ScanCluster(M_{\setminus i})$, where $M_{\setminus i}$ is the submatrix of M excluding column i.
- $M_a = ArrangeMatrix(M_{\setminus i}, c_v)$
- $\mathbf{v} = ArrangeColumn(\mathbf{M}_i, c_v)$, where $\mathbf{M}_i$ is the i^{th} column of M

Complete the matrix M:

- **Cluster trimming and partial filling:** for each cluster B_i in M_a and the respective $\mathbf{v}_i$ in $\mathbf{v}$ (c.f. (29)):
  - $\hat{B}_i = PadCluster(B_i, \mathbf{v}_i)$
  - $\hat{F}_i = U\text{-}DiagFillMatrix(\hat{B}_i)$ (Algorithm 1)
- $\tilde{M} = BuildSubDiagMatrixByClusters(\{\hat{F}_i\}_{i=1}^n)$
- $\hat{M}_{\setminus i} = RevertMatrix(\tilde{M}_{\setminus 1}, c_v)$
- $\hat{\mathbf{M}}_i = RevertColumn(\tilde{\mathbf{M}}_1, c_v)$

Output : a complete matrix $\hat{M}$

cluster as a data structure c. For a target row, we mark all columns that correspond to known entries on that row as columns of the target row's cluster. We then scan through each such column and mark the rows that correspond to known entries on the respective column. The resulting rows will cover all the rows belong to the same cluster of the target row. This step will be repeated to all rows sequentially except those that have been marked to be belonged to some clusters in prior steps. c will then store the total number of clusters and the column and row indices of each cluster.

- $M_a = ArrangeMatrix(M, c)$ rearranges the matrix M into M_a using c by reordering rows and columns. The resulting M_a will have the format of

$$\begin{pmatrix} B_1 & ? & \cdots & ? \\ ? & B_2 & \cdots & ? \\ \vdots & & \ddots & \\ ? & ? & \cdots & B_n \end{pmatrix}, \tag{28}$$

where submatrices $B_1, B_2, \cdots, B_n$ are clusters.

- $M_t^i = Trimming(B_i)$ finds the dependency of each column and row to the others columns and rows. This can be solved efficiently using QR decomposition. When a column/row can be represented by the other columns/rows, it will be removed from the basis choices in the cluster B_i. The data structure M_t^i stores the dependency relationship of columns and rows in the cluster B_i (see Theorem 5). More precisely, M_t^i will tell which column is independent from the rest (basis column). And how a trimmed column should be represented in terms of the basis columns.
- $F_i = FillCluster(B_i, M_t^i)$ fills the unknown elements of the cluster B_i using the dependency relationship M_t^i. Unknowns that lie on a basis choice (i.e., a column vector that cannot be "trimmed" in the previous step) will be filled with zeros. Unknowns that lie on a "trimmed" column will be filled according to Equation (27) of Theorem 5 so that the rank of the cluster will not increase. In other words, a trimmed column will be constructed as linear combination of basis columns according to M_t^i.
- $Update(M_a, F_i)$ replaces the corresponding submatrix in M_a using filled clusters F_i.
- $M_f = FillOffDiag(M_a, \{M_t^i\}_{i=1}^n, c)$ fills the "off-diagonal" elements in the matrix M_a using the dependency relationships $\{M_t^i\}_{i=1}^n$, the clusters relationship c to ensure the entire matrix has a rank no larger than the rank of the lowest rank cluster according to Equations (17)-(20) (see Theorem 2).
- $\hat{M} = RevertMatrix(M_f, c)$ reverts each coefficient in the filled matrix M_f to its original location in the input matrix M using c.
- $i = FindConjoinedColIndex(M)$ finds a potential conjoined column $\mathbf{v}$ as the column vector with the minimum number of unknowns (c.f. Definition 2) and returns the column index.
- $\mathbf{v} = ArrangeColumn(\mathbf{M}_i, c_v)$ reorders the coefficients in $\mathbf{M}_i$, the i^{th} column of M, according to c_v. The matrix $[\mathbf{v}|M_a]$ should be equivalent to M through column and row interchanging, where $M_a = ArrangeMatrix(M_{\setminus i}, c_v)$ and $M_{\setminus i}$ is the submatrix of M excluding $\mathbf{M}_i$. Moreover, we should have $[\mathbf{v}|M_a]$ with the structure

$$[\mathbf{v}|M_a] = \begin{pmatrix} \mathbf{v}_1 & B_1 & ? & \cdots & ? \\ \mathbf{v}_2 & ? & B_2 & \cdots & ? \\ \vdots & \vdots & & \ddots & \\ \mathbf{v}_n & ? & ? & \cdots & B_n \end{pmatrix}. \tag{29}$$

- $\hat{B}_i = PadCluster(B_i, \mathbf{v}_i)$ vertically concatenates the column $\mathbf{v}_i$ to the cluster B_i and then adds to the end a row with all unknowns except the first coefficient being

one, i,e., $\hat{B}_i = \left(\begin{array}{c|c} \mathbf{v}_i & B_i \\ 1 & ? \end{array} \right)$.

- $\tilde{M} = BuildSubDiagMatrixByClusters(\{\hat{F}_i\}_{i=1}^n)$ constructs a sub unknown-diagonized matrix using clusters $\hat{F}_i$, $i = 1, \cdots, n$, according to the proof of Theorem 3 (See explanation after Equation (25)). $\tilde{M}$ should have rank no larger than the rank of the largest rank clusters.
- $\hat{M}_{\backslash i} = RevertMatrix(\tilde{M}_{\backslash 1}, c_v)$ reverts the submatrix of $\tilde{M}$ (excluding the first column) and output to the submatrix of $\hat{M}$ (excluding column i).
- $\hat{\mathbf{M}}_i = RevertColumn(\tilde{\mathbf{M}}_1, c_v)$ reorders the coefficients of the column vector $\tilde{\mathbf{M}}_1$ according to c_v. Note that along with $\hat{M}_{\backslash i} = RevertMatrix(\tilde{M}_{\backslash 1}, c_v)$, $[\hat{\mathbf{M}}_i | \hat{M}_{\backslash i}]$ is equivalent to $[\tilde{\mathbf{M}}_1 | \tilde{M}_{\backslash 1}]$ through row and column exchanges.

F. Complexity analysis

The computational complexity of low-rank matrix completion can be determined by considering the following three steps: 1) scanning matrix to find the clusters, 2) in each cluster, finding the dependency of each column and row to the other columns and rows, respectively, and 3) filling the clusters and "off-diagonal" elements in the matrix.

Assume the input matrix has p rows and q columns. For the worst case, the required time of scanning matrix to find the clusters will be $O(pq)$ since one has to scan through all elements of the matrix. For finding the dependencies and removing the dependent columns and rows, the time depends on the number of cluster n and the size of clusters. One can take advantage of QR decomposition and the complexity will be about $O(p'q'^2)$ for the column trimming of a cluster with size $p' \times q'$, and since the cluster sizes are approximately proportional to $\frac{p}{n}$ and $\frac{q}{n}$, therefore the complexity of column trimming is $O\big(\frac{p}{n}(\frac{q}{n})^2 n\big) = O(\frac{pq^2}{n^2})$. As a result, the time complexity for finding columns and rows dependencies is $O\big(\frac{p^2q+q^2p}{n^2}\big)$.

Similar complexity can be argued for filling the clusters. For filling a column that involved a cluster with width $\hat{q}$, and assuming that the rank of the filled matrix is r, the first r column can be filled directly and so complexity will be just $O(rp)$. But for the rest of the $(\hat{q} - r)$ columns, it needs to be computed as the weighted sum of the first r columns, and so the complexity will be $O\big((\hat{q} - r)rp\big)$. Thus, the total complexity for filling n clusters will be $O\big(n(rp + (\hat{q} - r)rp)\big) \approx O\big(n(rp + (q/n - r)rp)\big) = O\big(rp(n + q - rn)\big) = O(rpq)$ since one may approximate $\hat{q}$ as $\frac{q}{n}$ and $n \leq q$.

Consequently, the total time complexity of Algorithm 1 for the typical case will be $O\big(pq + \frac{p^2q+q^2p}{n^2} + rpq\big)$. For Algorithm 2, excluding the step of $FindConjoinedColIndex(M)$ (of complexity $O(pq)$), the rest essentially has the same complexity $O\big(pq + \frac{p^2q+q^2p}{n^2} + rpq\big)$ as Algorithm 1. However, n will be the number of sub-clusters in this case.

Just to put things into perspective, note that the current implementation requires approximately 0.1 s per matrix with size 1500×1500 for Algorithm 1 and 0.8 s for Algorithm 2 in the most demanding case when running with pure Matlab on a Pentium 3 GHz (11-GB RAM) machine.

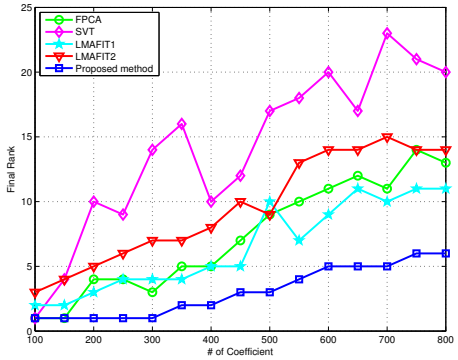


Fig. 1: Final Rank of FPCA [15], SVT [13], LMaFit1 [19], LMaFit2 [19], and the proposed method on small problems with varying rank and number of coefficients. Note that the input matrix has 1000 rows and 1000 columns.

III. Simulation result for Matrix Completion

In this section, to demonstrate how our proposed method is capable to efficiently recover low-rank matrices, we evaluate our proposed method on a series of matrix completion problems. Our experiments can be divided into *Random Matrix Completion Problems* and *Real Matrix Completion Problems*.

1) Experiments on random matrix completion problems: In this subsection, we evaluate our proposed method on a few random matrix completion problems. First, we test the sensitivity of our proposed method to the rank estimation using an increasing number of coefficients strategy. In this test, we maintained the size of the matrices consistent, $q = p = 1,000$ and varied the number of coefficients K from 100 to 800. A summary of computational results is presented in Table I. The final matrix rank "$Rank$" and average CPU time "$Time$" corresponding to this set of K values are reported in this table. This table reveals that our proposed method runs significantly faster than all other solvers to achieve a comparable accuracy. Moreover, the other solvers are essentially dominated by our proposed solver with respect to the obtained final low ranks.

In the second test, we consider applying matrix completion algorithms to randomly generated low-rank matrix approximation problems varying the size and the rank of the matrix, and the number of coefficients K. The purpose of this test is to find a low-rank approximation to a mathematically full-rank matrix, where it can have the lower rank in a short time. We compared our proposed method with LMaFit1 and LMaFit2, where the other solvers APGL, FPCA and SVT were excluded from this comparison since they would have demanded excessive CPU times. A summary of the computational results is presented in Table II. In this table, "$Time$" denotes the average CPU time, and "$Rank$" denotes the rank of the recovered solution for each solver.

2) Experiments on real matrix completion problems: In this subsection, we consider the low-rank matrix approximation problems based on "real data" set, namely, the Jester joke data set[4] [20] and "image inpainting".

[4]The Jester joke data set includes 4.1 million ratings for 100 jokes from 73421 users and is available online on the website: http://www.ieor.berkeley.edu/ goldberg/jester-data/.

TABLE I: Comparison of six solvers on small problems with varying rank and number of coefficients

Problem		**FPCA [15]**		**APGL [16]**		**SVT [13]**		**LMaFit1 [19]**		**LMaFit2 [19]**		**Proposed method**	
$p = q$	K	$Rank$	$Time$	$Rank$	$Time$	$Rank$	$Time$	$Rank$	$Time$	$Rank$	$Time$	$Rank$	$Time$
1000	100	1	11.08	76	3.43	1	44.76	2	0.36	3	0.34	1	0.05
1000	150	1	10.65	113	8.08	4	17.64	2	0.35	4	0.28	1	0.08
1000	200	4	11.96	144	14.43	10	80.37	3	0.40	5	0.33	1	0.11
1000	250	4	11.16	150	29.27	9	98.35	4	0.37	6	0.35	1	0.14
1000	300	3	10.86	150	20.00	14	103.20	4	0.34	7	0.35	1	0.16
1000	350	5	10.28	150	100.97	16	117.12	4	0.28	7	0.26	2	0.17
1000	400	5	10.23	150	58.71	10	136.00	5	0.31	8	0.39	2	0.20
1000	450	7	9.97	150	88.73	12	148.86	5	0.34	10	0.33	3	0.21
1000	500	9	10.08	150	120	17	168.00	10	0.40	9	0.42	3	0.21
1000	550	10	11.61	150	134.30	18	189.40	7	0.28	13	0.33	4	0.23
1000	600	11	10.04	150	130.15	20	211.20	9	0.39	14	0.36	5	0.25
1000	650	12	11.40	150	128.72	17	226.05	11	0.37	14	0.40	5	0.26
1000	700	11	11.52	150	117.34	23	249.85	10	0.53	15	0.50	5	0.26
1000	750	14	10.22	150	134.35	21	253.15	11	0.45	14	0.41	6	0.27
1000	800	13	10.48	150	132.36	20	276.80	11	0.43	14	0.39	6	0.30

TABLE II: Comparison of three solvers on problems with varying size, rank of the matrix and number of coefficients

Problem		**LMaFit1 [19]**		**LMaFit2 [19]**		**Proposed method**	
$p = q$	K	$Rank$	$Time$	$Rank$	$Time$	$Rank$	$Time$
1000	300	4	0.34	7	0.35	1	0.16
1000	400	5	0.31	8	0.39	2	0.20
1000	500	10	0.40	9	0.42	3	0.21
5000	1500	18	0.37	20	0.36	3	1.03
5000	2000	22	0.41	26	0.45	4	1.39
5000	2500	25	0.41	27	0.43	8	1.52
10000	1000	20	2.47	23	3.64	1	0.79
10000	1500	8	1.32	9	1.57	2	1.17
10000	2000	42	2.12	51	2.54	2	1.86
20000	1500	15	7.75	18	8.40	1	1.61

The Jester joke data set contains four problems "jester-1" with 24983 users who have rated 36 or more jokes, "jester-2" with 23500 users who have rated 36 or more jokes, "jester-3" with 24938 users who have rated between 15 and 35 jokes, and "jester-all" with combining all of the first three data sets.

let M be the original incomplete data matrix where the i^{th} row of M corresponds to the ratings given by the i^{th} user on the jokes, and δ be the set of indexes for which M_{ij} is given. Since some of the entries of M are missing, we compute the Normalized Mean Absolute Error (NMAE) to measure the accuracy [16], [19], [20]. The NMAE is defined as

$$NMAE = \frac{1}{r_{max} - r_{min}\,|\delta|} \sum_{(i,j)\in\delta} |M_{ij} - X_{ij}|, \qquad (30)$$

where r_{min} and r_{max} are the lower and upper bounds for the ratings, and M_{ij} and X_{ij} are the estimated and computed ratings of joke j by user i, respectively. Note that we have $r_{min} = -10$ and $r_{max} = 10$ and all ratings are scaled to the range $[-10, +10]$.

Table III shows the results for the LMaFit method [19] and our proposed method on the real matrix completion problem using the jester joke data set. As shown in this table, our proposed method generally returns solutions with lower NMAE than those returned by LMaFit. It is critical to compare two solvers on problem "jester-3" where even LMaFit reports a solution with the lower rank of 43; our solution is more accurate than LMaFit solution and shows the lower NMAE.

To graphically illustrate the effectiveness of our proposed method, we applied it to image inpainting. In grayscale image inpainting, the value of some of the pixels on the image are missing, and the task here is to fill these missing values. Note that the missing pixel positions in the image inpainting are not randomly distributed. If the image is of low-rank, or of numerical low-rank, the matrix completion solvers can be applied on the image inpainting problem to obtain low-rank approximations.

The 512×512 original grayscale image is shown in Fig. 2 (a). Fig. 2 (b) was obtained by truncating the SVD of the images to get the images of rank 40. Fig. 2 (c) is the masked image obtained from Fig. 2 (b), where 9.30% of the pixels were masked in a non-random fashion. The recovered images of Fig. 2 (c) from LMaFit, APGL, and our proposed method are depicted in Figs. 2 (d), (e), and (f).

Table IV shows a summary of the computational results of the image inpainting. Note that in this table, $Rel.err$ [5] denotes the relative error between the original and recovered images. From these figures and the table, we can see our proposed method recover the images significantly better than the other methods with the less relative error.

IV. Application to Video Denoising

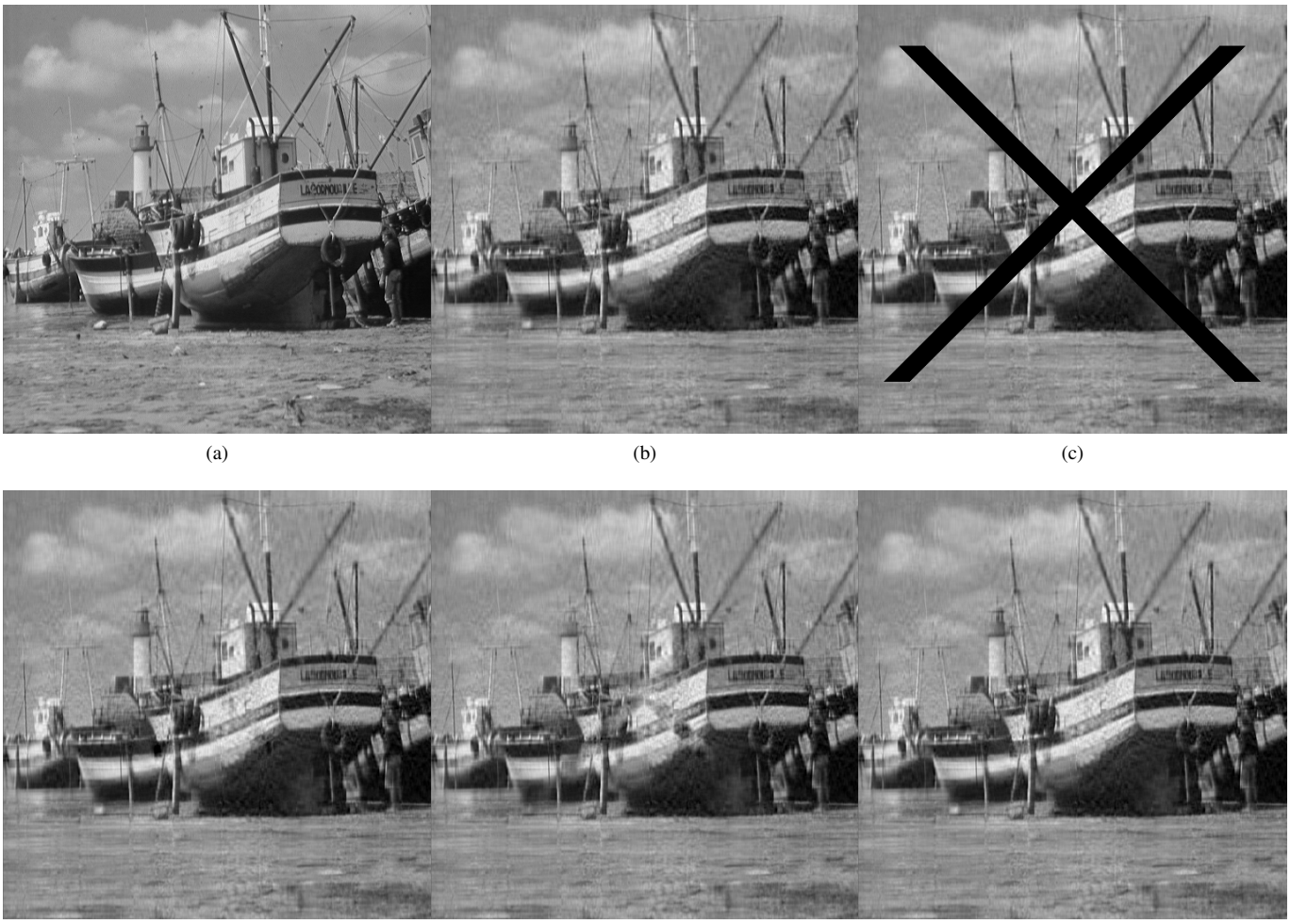
Video sequences are often corrupted by noise during acquisition or transmission. Some noise sources located in camera hardware became active during image acquisition under some lighting conditions. Other noise sources are over transmission channels. Most video denoising algorithms proposed in the literature assume additive white Gaussian noise, which can be categorized into pixel domain and transform domain methods. However, we consider Impulsive/Poisson/Gaussian noise in our work and will show how robust our video denoising method is.

Many video denoising methods have been proposed in the last few decades, e.g., [21]–[24]. One of the first methods to address the denoising problem was the bilateral filter, which

[5] We used the relative error: $Rel.err := \frac{\|M_{rec} - M\|_F}{\|M\|_F}$ to estimate the closeness of M_{rec} to M, where M_{rec} is the "recovered" image produced by the algorithms, and M is the original image.

TABLE III: Numerical results on real data sets.

Problem		**LMaFit [19]**				**Proposed method**		
Name	p/q	$Iter$	$Time$	$NMAE$	$Rank$	$Time$	$NMAE$	$Rank$
jester-1	24983/100	30	3.54	0.1946	62	6.34	0.1942	100
jester-2	23500/100	35	5.98	0.1960	62	5.51	0.1959	100
jester-3	24938/100	169	8.02	0.1970	43	7.34	0.1381	97
jester-all	73421/100	35	11.15	0.1869	62	49.69	0.1865	100



(a) (b) (c)

(d) (e) (f)

Fig. 2: Image inpainting Problem for *Boat* image: (a) Original image; (b) rank 40 image; (c) deterministically 9.30% masked rank 40 image; (d) LMaFit1 [19]; (e) APGL [16]; (f) result of the proposed method.

TABLE IV: Numerical results on image inpainting

Problem		**APGL [16]**				**LMaFit1 [19]**				**Proposed method**		
$\%mask$	r	$Iter$	$Time$	$Rel.err$	$Rank$	$Iter$	$Time$	$Rel.err$	$Rank$	$Time$	$Rel.err$	$Rank$
3.70	40	33	8.37	0.1039	50	195	7.29	0.0818	42	7.80	0.0794	40
9.30	40	29	2.78	0.0354	50	139	1.49	0.0365	40	3.45	0.0248	40
15.81	40	33	3.32	0.1155	50	258	2.74	0.1408	46	5.21	0.0890	40

was proposed by Tomasi and Manduchi [24]. However, this method fails to perform well in when the noise is strong. Selesnick and Li [23] proposed 2D and 3D dual-tree oriented wavelet transforms which give a motion-based multi-scale decomposition for video. They used the proposed transforms for video denoising, where the 2D transform is applied to each frame individually.

Recently, the idea of patch based sparse coding has been applied to video denoising [21], [25]–[27]. Marial *et al.* in [25] suggested to extend the sparse coding approach by proposing that similar patches share the same dictionary elements in their sparse decomposition on denoising. Another recent example based on an enhanced sparse representation in transform domain is block-matching 3-D filter (BM3D) [27]. In BM3D,

similar 2D image blocks are grouped into a 3D data array based on the l_2 norm distance function. Then, the 3D data array is filtered by wavelet shrinkage or Wiener filter in 3D transform domain. The denoised image is produced from all grouped blocks after applying the inverse 3D transform. The concept of BM3D is generalized to video denoising in VBM3D [21]. In VBM3D, the noisy video is processed in a block-wise manner in both spatial and temporal domains. Then, a predictive search block-matching is combined with collaborative hard thresholding or collaborative Wiener filtering.

In this work, we show that the proposed method can operate directly on the raw noisy images that suffer from non-homogeneous noise. The proposed method is similar to that described in [28]. However, we incorporate our proposed matrix completion method into the denoising algorithm and rather than applying a suboptimal block matching algorithm as in [29], we use a near-optimal block matching method [30] with higher complexity. We can afford latter as the proposed matrix completion method runs significantly faster than other matrix completion methods. The goal of our denoising method is to keep only the reliable pixels and get rid of all other un-reliable pixels we find as noise. For each patch in the reference frame, we find the similar patches in the other frames using a block matching algorithm. The found matches will be vectorized and then stacked into a matrix. The reliable pixel values in the matrix are between the mean $\pm$ standard deviation of all elements in the same row. The main step will be done by applying our proposed matrix completion approach on the incomplete matrix. The output of matrix completion is a noise free full matrix. Then, the average value of each row in the full matrix can recover the denoised patch. Repeating the same procedure for all blocks of reference frame can build a denoised frame.

A. Our Method

The problem of video denoising can mathematically be shown as

$$y(x) = z(x) + n(x), \tag{31}$$

where $z(x)$ is the original video signal and $y(x)$ is the observed video after being corrupted by Gaussian/Poisson/Impulsive noise $n(x)$. $x = (i, j, k) \in X$ are coordinates in the spatio-temporal $3D$ domain $X \subset \mathbb{Z}^3$, where the first two components (i, j) are the spatial coordinates and the third one k is the time (frame) index. The main procedure for our proposed denoising method is summarized in Algorithm 3.

Algorithm 3 Video Denoising using matrix completion- estimate version of denoised image $\mathcal{X}$

Inputs : noisy video y, pixel overlap v

Initialize :

- Set $\mathcal{V}$ and $\mathcal{W}$ to be zero images of the same size as the video frame size.

Produce the pre-processing step for removing impulsive noise before patch matching :

- Apply Adaptive Median Filter: $Y^{am} = AMF(y)$

Find the denoised patches: For each coordinate $x \in \Omega$ with v pixel overlap in each direction do:

(a) $S_x = BM\left(Y_x^{am}\right)$

(b) $\hat{\mathcal{Z}_{S_x}} = ReliableElements\left(\mathbf{Y}_{S_x}\right)$

(c) $\check{\mathcal{Z}}_x = DMC\left(\hat{\mathcal{Z}}_{S_x}\right)$

(d) $\hat{z}_x = AVG_{row}(\check{\mathcal{Z}}_x)$

(e) $\mathcal{V} = \mathcal{V} + \hat{z}_x$

(f) $\mathcal{W} = \mathcal{W} + \hat{w}_x$

Normalize : $\hat{\mathcal{Z}} = \mathcal{V}/\mathcal{W}$

Output : a denoised image $\hat{\mathcal{Z}}$

Implementation Details:

- $Y^{am} = AMF(y)$ performs adaptive median filtering using y. Because, the video is corrupted by image noise, applying a patch matching algorithm directly on noisy video generates unreliable result. Specifically, the block matching algorithm will suffer from impulsive noise, and its performance will be seriously degraded by strong impulsive noise. Hence, using a preprocessing step to remove impulsive noise before the block matching step will improve the resulting performance. In our work, we simply use the adaptive median filter proposed by Hwang and Haddad in [31].
- $\Omega \subset X$ is a set that includes the coordinates of the reference blocks. In general, each pixel in the reference image is covered by several patches. We aggregate overlapped patches by a weighted average at each pixel.
- Y_x^{am} denotes a block of size $q \times q$ in Y^{am}, where its center is at x.
- $S_x = BM\left(Y_x^{am}\right)$ presents a block matching algorithm using Y_x^{am} as a reference block, where the result is the set S_x containing the coordinates of the matched blocks. Although there are several methods to find the similar matches [21], [32]–[34], in our work, we use the Adaptive Rood Pattern Search (ARPS) algorithm [30] because of its computational efficiency.
- $\mathbf{Y}_{S_x}$ denotes a matrix formed by stacking the vectorized blocks $Y_{x \in S_x}$ together, where Y_x is a block of size $q \times q$ centered at x in y.
- $\hat{\mathcal{Z}_{S_x}} = ReliableElements\left(\mathbf{Y}_{S_x}\right)$ discards those matrix elements of $\mathbf{Y}_{S_x}$ that are far away from $mean \pm standard\ deviation$ of its corresponding row, designates them as unreliable elements, and then replaces them by zero. Note that those unreliable elements could be the pixels corrupted by Gaussian/Poisson/Impulsive noise or from mismatched patches obtained from previous step (block matching). Also, keeping the reliable elements, lets us recover the full matrix needed for the next step.
- $\check{\mathcal{Z}}_x = DMC\left(\hat{\mathcal{Z}}_{S_x}\right)$ performs a decomposing approach for low-rank matrix completion algorithm (see Section II) using $\hat{\mathcal{Z}}_{S_x}$ and $\check{\mathcal{Z}}_x$ that will be a full matrix with noise

free elements. Recently, many matrix completion methods have been studied [2], [11], [19]. In our work, we use a decomposing approach for low-rank matrix completion algorithm, because of its computational efficiency.

- $\hat{z}_x = AVG_{row}(\check{\mathcal{Z}}_x)$ finds the average value of each row in matrix $\check{\mathcal{Z}}_x$ and converts the obtained vector to a block. Also, $\hat{z}_x$ will be an estimated block of size $q \times q$ centered at x in $\hat{\mathcal{V}}$.
- $\hat{w}_x$ is a patch with the same size as $\hat{z}_x$. Note that, all pixel values in $\hat{w}_x$ are equal to 1.

TABLE V: OUTPUT PSNR OF OUR PROPOSED DENOISING METHOD FOR THE TWO VIDEO SEQUENCES; note that, we kept the Gaussian and Poisson noise constant in all tests

Video name (frame size) / **Impulsive noise density**	**Suzie**	**Coastguard**
0.10	26.3679	22.6124
0.15	26.2852	22.6060
0.20	26.3810	22.5819
0.25	26.3135	22.6281
0.30	26.3545	22.6105
0.35	26.4119	22.6163
0.40	26.2867	22.6346
0.45	26.3257	22.6149
0.50	26.3257	22.5907

B. Experimental Results on Denoising

In this section, we present some video denoising examples to evaluate our performance, using existing sequences such as Miss America, Galleon, and Suzie. All tests in this section were processed in the following manner: All 30 frames were involved in the reconstruction of each image. The block size used for block matching (q) was 20×20 and was not changed for various tests. We obtained a locally consistent solution by allowing patches to overlap, where the overlapped regions (v) were 5 pixels in each direction. Also, for each reference patch, we extracted 5 most similar patches used in each frame using block matching algorithm. For simplicity, we employ the basic version of our algorithm without taking advantage of sub u-diagonalization.

In Fig. 3, we show the PSNR result and a clear visual comparison of the Galleon sequence. The original video is seriously corrupted by a significant mixed noise level with Poisson noise, Gaussian white noise of mean zero and variance 0.02, and Impulsive noise of the noise density 0.03. As shown in Figures. 3 and 5, VBM3D method [21] and tvregv2 [35] generate severe artifacts at edge areas, while our proposed denoising method performs remarkably well for the detail structures and is free of these artifacts.

In Table V, we present the PSNR results of the proposed denoising algorithm for a few sequences, where Impulsive noise is changing. This table shows how our algorithm is robust in denoising the corrupted sequences of serious impulsive noise.

In Graph 4, we compare our denoising method with the VBM3D method [21], which is among the state-of-the-art in video denoising. In this comparison, we apply our denoising method on Coastguard and Suzie sequences, for which we changed the Gaussian noise but kept the Poisson and Impulsive noise constant for all methods. Note that, for a fair comparison and also because the VBM3D method [21] works on removing just the Gaussian noise from the corrupted video, we ran the adaptive median filter method [31] on the test data with a pre-process of removing impulsive noise. In contrast, in our work, we did not use any existing impulsive noise method to detect pixels corrupted by Impulsive noise. The graph in Fig. 7 shows the frame by frame PSNR values of Miss America and vtc1nw. Table VII shows the average PSNR values for our proposed method and the compared methods. Our proposed method surpasses the VBM3D method [21] in all frames by a significant margin for all sequences with more than 2dB. In contrast, while [28] also outperform the VBM3D method but with a significantly smaller margin, we conjecture that the gain is due to the near-optimal block matching method [30] used in our approach.

TABLE VI: PSNR and time comparison for using various matrix completion

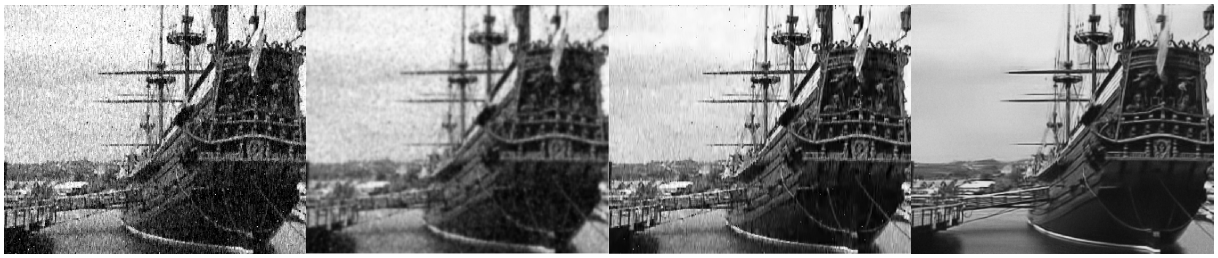
		Tempete	**Galleon**	**Coastguard**
Proposed method	$PSNR[dB]$	23.37	22.73	23.57
	$Time(seconds)$	240	218	110
Denoising method using OptSpace [11]	$PSNR[dB]$	22.79	22.60	23.63
	$Time(seconds)$	1355	1683	538
Denoising method using LMAFIT1 [19]	$PSNR[dB]$	22.78	22.52	21.81
	$Time(seconds)$	220	251	119
Denoising method using FPCA [15]	$PSNR[dB]$	21.38	20.39	20.97
	$Time(seconds)$	5828	7069	1584

We also replaced our proposed decomposition matrix completion with OptSpace [11], LMAFIT1 [19] and FPCA [15] to compare the result and time consumption (see Table VI). It can be seen in Table VI that our method has comparable performance in terms of PSNR, for which it executes much faster than those methods.

V. CONCLUSION

In conclusion, we have proposed a novel and efficient decomposition method for matrix completion. A key idea of our approach is to divide and conquer. The input matrix is partitioned into clusters, and then each cluster is filled separately. A dependency scanning step estimates the lowest possible rank of each cluster by identifying independent rows. The unknown elements of these independent rows can then be filled arbitrarily without increasing the rank of the cluster. The remaining unknown elements of the clusters are filled by the dependency relationship obtained earlier. Finally, the "off-diagonal" elements are filled to ensure the entire matrix has the lowest possible rank.

We have compared our method with several recently introduced techniques using randomly generated and real world matrices. We further proposed a block-based video denoising method using our decomposition approach, in which we keep only reliable pixels and eliminate all unreliable pixels. Our denoising method can remove the serious mixed noise from video sequence, while most of the existing methods have been limited to one specific type of noise. Quantitative and qualitative experiments with video sequences corrupted by mixed noise have shown that the proposed algorithm outperforms the state-of-the-art methods for video denoising tasks.



(a) (b) (c) (d)

Fig. 3: Video denoising for *Galleon* sequence: (PSNR in brackets). From left to right: noisy image; tvregv2 [35] [PSNR: 17.8208]; VBM3D algorithm [21] [PSNR: 17.9226]; result of the proposed denoising algorithm [PSNR: 21.2437].

TABLE VII: AVERAGE PSNR FOR THE TWO VIDEO SEQUENCES

Sequence	Wiener2	VBM3D [21]	3DWTF [23]	tvregv2 [35]	Proposed denoising Method
Miss America	26.6796	30.9090	24.5168	28.1036	32.1931
vtc1nw	25.5855	28.2356	22.1496	27.7033	31.6442

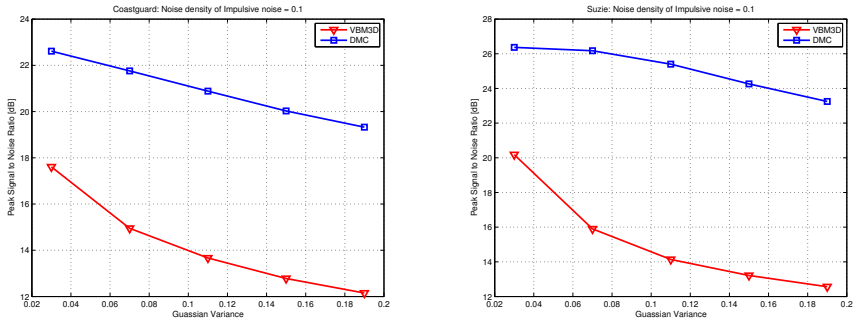
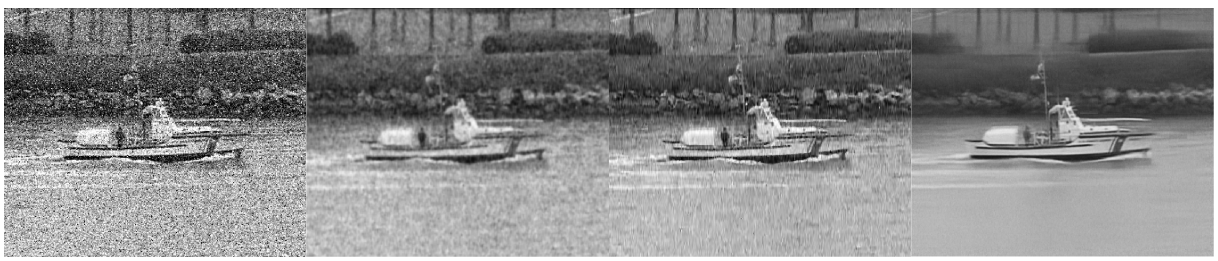


Fig. 4: PSNR values of VBM3D algorithm [21] and proposed denoising method for *Coastguard* and *Suzie* sequences. Note that, we kept the Impulsive noise consistent in all tests.

(a) (b) (c) (d)

Fig. 5: Video denoising for *Suzie* sequence: (PSNR in brackets). (a) noisy image; (b) tvregv2 [35] [PSNR:25.1205]; (c) VBM3D algorithm [21] [PSNR:26.5245]; (d) result of the proposed denoising algorithm [PSNR:29.3254].



(a) (b) (c) (d)

Fig. 6: Video denoising for *Coastguard* sequence: (PSNR in brackets). (a) noisy image; (b) tvregv2 [35] [PSNR:20.9469]; (c) VBM3D algorithm [21] [PSNR:21.0090]; (d) result of the proposed denoising algorithm [PSNR:23.5725].

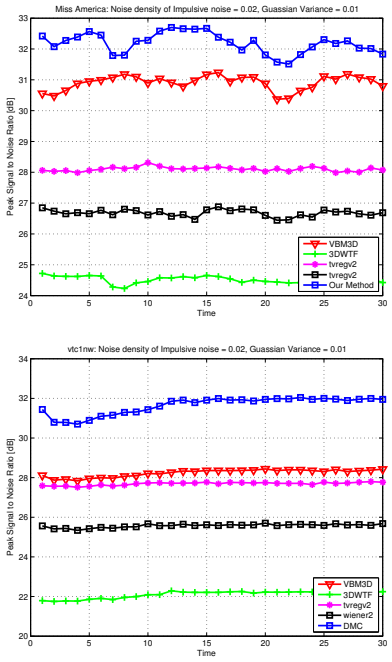


Fig. 7: PSNR values of each denoised frame by VBM3D algorithm [21], 3DWTF [23], tvregv2 [35], wiener2 and the proposed denoising method for (a) the *Miss America* and (b) the *vtc1nw* sequence.

ACKNOWLEDGMENT

The authors would like to thank the associate editor and the anonymous reviewers for their constructive comments and suggestions. We would also like to thank Mrs. Renee Wagenblatt and Ms. Sepideh Darbandi for editing the manuscript.

REFERENCES

[1] N. Srebro, "Learning with matrix factorizations," Ph.D. dissertation, Citeseer, 2004.

[2] E. Candes and Y. Plan, "Matrix completion with noise," *Arxiv preprint arXiv:0903.3131*, 2009.

[3] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: a factorization method," *International Journal of Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992.

[4] T. Graepel, "Kernel matrix completion by semidefinite programming," *Artificial Neural NetworksICANN 2002*, pp. 141–142, 2002.

[5] J. Abernethy, F. Bach, T. Evgeniou, and J. Vert, "Low-rank matrix factorization with attributes," *arXiv preprint cs/0611124*, 2006.

[6] Y. Amit, M. Fink, N. Srebro, and S. Ullman, "Uncovering shared structures in multiclass classification," in *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, vol. 24, 2007, p. 17.

[7] A. Singer, "A remark on global positioning from local distances," *Proceedings of the National Academy of Sciences*, vol. 105, no. 28, p. 9507, 2008.

[8] R. Schmidt, "Multiple emitter location and signal parameter estimation," *Antennas and Propagation, IEEE Transactions on*, vol. 34, no. 3, pp. 276–280, 1986.

[9] E. Candes and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[10] E. Candes and T. Tao, "The power of convex relaxation: Near-optimal matrix completion," *arXiv*, vol. 903, 2009.

[11] R. Keshavan, A. Montanari, and S. Oh, "Matrix completion from a few entries," *Information Theory, IEEE Transactions on*, vol. 56, no. 6, pp. 2980–2998, 2010.

[12] A. Waters, A. Sankaranarayanan, and R. Baraniuk, "Sparcs: Recovering low-rank and sparse matrices from compressive measurements," Technical report, Rice University, Houston, TX, Tech. Rep., 2011.

[13] J. Cai, E. Candes, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *preprint*, 2008.

[14] K. Lee and Y. Bresler, "Admira: Atomic decomposition for minimum rank approximation," *arXiv*, vol. 905, 2009.

[15] S. Ma, D. Goldfarb, and L. Chen, "Fixed point and Bregman iterative methods for matrix rank minimization," *Mathematical Programming*, pp. 1–33, 2009.

[16] K. Toh and S. Yun, "An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems," *preprint*, 2009.

[17] W. Dai and O. Milenkovic, "Set: an algorithm for consistent matrix completion," *Arxiv preprint arXiv:0909.2705*, 2009.

[18] R. Meka, P. Jain, and I. Dhillon, "Guaranteed rank minimization via singular value projection," 2009.

[19] Z. Wen, W. Yin, and Y. Zhang, "Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm."

[20] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

[21] K. Dabov, A. Foi, and K. Egiazarian, "Video denoising by sparse 3d transform-domain collaborative filtering," in *Proc. 15th European Signal Processing Conference*, vol. 1, no. 2. Citeseer, 2007, p. 7.

[22] S. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on Image Processing*, pp. 1532 – 1546, September 2000.

[23] I. Selesnick and K. Li, "Video denoising using 2d and 3d dual-tree complex wavelet transforms," *Wavelets: Applications in Signal and Image Processing X*, vol. 5207, pp. 607–618, 2003.

[24] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998, pp. 839–846.

[25] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Non-local sparse models for image restoration," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2272–2279.

[26] M. Orchard and G. Sullivan, "Overlapped block motion compensation: An estimation-theoretic approach," *Image Processing, IEEE Transactions on*, vol. 3, no. 5, pp. 693–699, 1994.

[27] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *Image Processing, IEEE Transactions on*, vol. 16, no. 8, pp. 2080–2095, 2007.

[28] H. Ji, C. Liu, Z. Shen, and Y. Xu, "Robust video denoising using low rank matrix completion," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1791–1798.

[29] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 3, no. 2, pp. 148–157, 1993.

[30] Y. Nie and K. Ma, "Adaptive rood pattern search for fast block-matching motion estimation," *Image Processing, IEEE Transactions on*, vol. 11, no. 12, pp. 1442–1449, 2002.

[31] H. Hwang and R. Haddad, "Adaptive median filters: new algorithms and results," *Image Processing, IEEE Transactions on*, vol. 4, no. 4, pp. 499–502, 1995.

[32] N. Barzigar, A. Roozgard, S. Cheng, and P. Verma, "Scobep: Dense image registration using sparse coding and belief propagation," *Journal of Visual Communication and Image Representation*, 2012.

[33] S. Zimmer, S. Didas, and J. Weickert, "A rotationally invariant block matching strategy improving image denoising with non-local means," in *Proc. 2008 International Workshop on Local and Non-Local Approximation in Image Processing*, 2008, pp. 135–142.

[34] N. Barzigar, A. Roozgard, P. Verma, and S. Cheng, "A video super resolution framework using scobep," *IEEE Transactions on Circuits and Systems for Video Technology*, 2013.

[35] P. Getreuer. (2009) http://www.getreuer.info/home/tvreg.