Rapid Robust Principal Component Analysis: CUR Accelerated Inexact Low Rank Estimation

HanQin Cai, Keaton Hamm, Longxiu Huang, Jiaqi Li, and Tao Wang, *Senior Member, IEEE*

***Abstract*—Robust principal component analysis (RPCA) is a widely used tool for dimension reduction. In this work, we propose a novel non-convex algorithm, coined Iterated Robust CUR (IRCUR), for solving RPCA problems, which dramatically improves the computational efficiency in comparison with the existing algorithms. IRCUR achieves this acceleration by employing CUR decomposition when updating the low rank component, which allows us to obtain an accurate low rank approximation via only three small submatrices. Consequently, IRCUR is able to process only the small submatrices and avoid the expensive computing on full matrix through the entire algorithm. Numerical experiments establish the computational advantage of IRCUR over the state-of-art algorithms on both synthetic and real-world datasets.**

***Index Terms*—RPCA, principal component analysis, CUR decomposition, low-rank modeling, outlier removal.**

I. Introduction

PRINCIPAL component analysis (PCA) is one of the fundamental tools for dimension reduction. A well-known drawback of the standard PCA approach, viz., singular value decomposition (SVD), is its high sensitivity to outliers. Robust PCA (RPCA) is designed to overcome this shortcoming and enhance the robustness of PCA to potentially corrupted data. RPCA has received a lot of attention in recent years and appears in a wide range of applications, e.g., image alignment and rectification [1], [2], face recognition [3], [4], feature identification [5], [6], sparse graph clustering [7], NMR spectroscopy signal recovery [8], and video background subtraction [9], [10].

Manuscript received September 1, 2020; revised November 10, 2020; accepted December 4, 2020. Date of publication December 11, 2020; date of current version January 21, 2021. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010166001, in part by the AFOSR MURI under Grant FA9550-18-10502, in part by the ONR Grant N0001417121, in part by the ARO Grant W911NF-20-1-0076, in part by the NSF TRIPODS Grant CCF-1740858, in part by the CAREER DMS Grant 1348721, in part by the BIGDATA Grant 1740325, in part by the the Program for Guanddong Introducing Innovative and Entrepreneurial Teams under Grant 2016ZT06D211, and in part by the Cultivation Project of Supercomputing Applications under Grant 67000-18843409. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Xun Chen. *(All authors have contributed equally.) (Corresponding author: Longxiu Huang.)*

HanQin Cai and Longxiu Huang are with the Department of Mathematics, University of California, Los Angeles, CA 90095 USA (e-mail: hqcai@math.ucla.edu; huangl3@math.ucla.edu).

Keaton Hamm is with the Department of Mathematics, University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: keaton.hamm@uta.edu).

Jiaqi Li and Tao Wang are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China (e-mail: lijq63@mail2.sysu.edu.cn; wangtao29@mail.sysu.edu.cn).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LSP.2020.3044130, provided by the authors.

Digital Object Identifier 10.1109/LSP.2020.3044130

We consider the following problem setting for RPCA: given a sparsely corrupted observational data matrix $\boldsymbol{D} \in \mathbb{R}^{n_1 \times n_2}$, which is the sum of the underlying low rank matrix $\boldsymbol{L}$ and sparse outlier matrix $\boldsymbol{S}$, our goal is to recover $\boldsymbol{L}$ and $\boldsymbol{S}$ simultaneously from $\boldsymbol{D} = \boldsymbol{L} + \boldsymbol{S}$. Intuitively, RPCA can be modeled as a non-convex optimization problem:

$$\underset{\boldsymbol{L}',\boldsymbol{S}'}{\text{minimize}} \|\boldsymbol{D} - \boldsymbol{L}' - \boldsymbol{S}'\|_F$$
$$\text{subject to } \operatorname{rank}(\boldsymbol{L}') \le r \quad \text{and} \quad \|\boldsymbol{S}'\|_0 \le \alpha n^2 \tag{1}$$

where r is the rank of the underlying low rank matrix, the symbol $\|\cdot\|_0$ denotes the ℓ_0-norm, and α is the sparsity level of the underlying sparse outlier matrix; for ease of notation, we use $n_1 = n_2 =: n$ through the paper, but emphasize that all results can be easily extended to non-square matrices.

One can see that the RPCA model should handle outliers better than the standard PCA since outliers can be extracted as the sparse matrix $\boldsymbol{S}$; however, the solution of (1) may not be unique if the low rank component is also sparse, or vice versa [11]. To ensure the uniqueness of the solution, the following assumptions are commonly made for RPCA:

Assumption 1 (μ-incoherence of $\boldsymbol{L}$): Let $\boldsymbol{L} = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be the compact SVD of $\boldsymbol{L}$. There exists a constant μ such that

$$\|\boldsymbol{W}\|_{2,\infty} \le \sqrt{\mu r/n} \quad \text{and} \quad \|\boldsymbol{V}\|_{2,\infty} \le \sqrt{\mu r/n}.$$

Assumption 2 (α-sparsity of $\boldsymbol{S}$): $\boldsymbol{S}$ has no more than αn non-zero entries in each of its rows and columns.

Essentially, Assumption 1 ensures the low rank component is not too sparse and Assumption 2 ensures the sparse component is not locally dense. Note that some papers use random support assumptions for $\boldsymbol{S}$ instead of Assumption 2; for example, Assumption 2 will be satisfied when the support of outliers is drawn from some commonly used stochastic process [12, Theorem 1.2]. With these two assumptions, the separation of $\boldsymbol{L}$ and $\boldsymbol{S}$, viz., RPCA, becomes a well-posed problem.

A. Prior Art and Contribution

RPCA was raised and popularized by earlier works [11], [13], [14], wherein convex relaxed formulas for RPCA were proposed and studied. Unfortunately, these earlier approaches only achieved sublinear convergence and thus are computationally intensive [15]. Later, a number of non-convex approaches were studied to solve (1) directly. In particular, [16] proposed an alternating projections based non-convex algorithm and an accelerated version was studied in [17] later. Also, a gradient descent based method was proposed in [18], which was recently

modified for accelerating with ill-conditioned problems [19]. All of the aforementioned non-convex methods offered linear convergence with a complexity of at least $\mathcal{O}(rn^2)$ flops per iteration.

In this letter, we propose a novel rapid non-convex algorithm for directly solving the non-convex RPCA model (1). The key is to integrate the CUR decomposition [20] with the iterative RPCA framework and replace the SVD, which dramatically reduces the complexity to $\mathcal{O}(r^2 n \log^2(n))$ flops per iteration.

II. Background of CUR Decomposition

Consider a noiseless low rank data matrix. The standard dimension reduction tools such as PCA can achieve desired approximation and compression for data; however, the approximation may lose interpretability [20]. One approach for addressing this difficulty is to utilize the self-expressiveness of data, that is, a set of data is generally well-represented via linear combinations of the other data points rather than in some abstract bases, e.g., the singular vectors.

CUR decomposition is an efficient tool to maintain the interpretability in dimension reduction. The classic CUR decomposition problem asks: given a matrix $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ with rank r, can we decompose it into terms involving only some of its columns and some of its rows? Particularly, if r columns of $\boldsymbol{L}$ which span the column space of $\boldsymbol{L}$ and r rows which span the row space of $\boldsymbol{L}$ are chosen, then we can obtain $\boldsymbol{L}$ itself from these submatrices. The answer has been known to be yes for some time.

Theorem 1: Consider row and column indices $\mathcal{I}, \mathcal{J} \subseteq [n]$ with $|\mathcal{I}|, |\mathcal{J}| \geq r$. Denote submatrices $\boldsymbol{C} = \boldsymbol{L}_{:,\mathcal{J}}$, $\boldsymbol{U} = \boldsymbol{L}_{\mathcal{I},\mathcal{J}}$ and $\boldsymbol{R} = \boldsymbol{L}_{\mathcal{I},:}$. If $\operatorname{rank}(\boldsymbol{U}) = \operatorname{rank}(\boldsymbol{L})$, then $\boldsymbol{L} = \boldsymbol{C}\boldsymbol{U}^{\dagger}\boldsymbol{R}$, where $(\cdot)^{\dagger}$ denotes the Moore-Penrose pseudoinverse.

Theorem 1 is essentially folklore; the reader may consult [21] for a history and proof. From the theorem statement, it naturally implies that the success of CUR decomposition heavily relies on whether the rank of the mixing submatrix $\boldsymbol{U}$ equals that of $\boldsymbol{L}$. In fact, with various sampling strategies, this condition can be guaranteed with high probability if we sample an appropriate number of rows and columns. For instance, Theorem 2 presents the sampling complexity for uniform sampling.

Theorem 2 ([22, Theorem 1.1]): Let $\boldsymbol{L}$ satisfy Assumption 1, and suppose we sample $|\mathcal{I}| = \mathcal{O}(r\log(n))$ rows and $|\mathcal{J}| = \mathcal{O}(r\log(n))$ columns uniformly with replacement. Then $\boldsymbol{U} = \boldsymbol{L}_{\mathcal{I},\mathcal{J}}$ satisfies $\operatorname{rank}(\boldsymbol{U}) = \operatorname{rank}(\boldsymbol{L})$ with probability at least $1 - \mathcal{O}(rn^{-2})$.

Various sampling strategies have been proposed for choosing $\mathcal{I}$ and $\mathcal{J}$ including column length and leverage score sampling. The complexity of computing these scores is $\mathcal{O}(n^2)$ while that for uniform scores is $\mathcal{O}(1)$. Since the order of $|\mathcal{I}|, |\mathcal{J}|$ required is the same under incoherence assumptions, we focus solely on uniform sampling here (for more details, see [23, Table 1]).

Note that $\boldsymbol{U}$ is an $\mathcal{O}(r\log(n)) \times \mathcal{O}(r\log(n))$ matrix under uniform sampling. By Theorem 1, the only computational cost is incurred by calculating the pseudo-inverse of $\boldsymbol{U}$, which requires only $\mathcal{O}(r^3\log^2(n))$ flops. In contrast, computing the SVD requires $\mathcal{O}(rn^2)$ flops. This confirms the computational efficiency of CUR decomposition with larger n and smaller r.

We also note that CUR decompositions have been connected with sparse optimization [24], wherein they were shown to be distinct from the standard sparse PCA.

Algorithm 1: Iterated Robust CUR for RPCA (IRCUR).

Input: $\boldsymbol{D}$: observed corrupted data matrix; r: rank; ε: target precision level; ζ_0: initial thresholding value; γ: thresholding decay parameter; $|\mathcal{I}|, |\mathcal{J}|$: sampling number of rows and columns.
Uniformly sample row indices $\mathcal{I}$ and column indices $\mathcal{J}$.
$\boldsymbol{L}_0 = \boldsymbol{0}, \quad \boldsymbol{S}_0 = \boldsymbol{0}, \quad k = 0$
while $e_k > \varepsilon$ (e_k is defined as in (6)) **do**
  (Optional) Resample indices $\mathcal{I}$ and $\mathcal{J}$
  // Phase I: updating sparse component
  $\zeta_{k+1} = \gamma^k \zeta_0$
  $[\boldsymbol{S}_{k+1}]_{:,\mathcal{J}} = \mathcal{T}_{\zeta_{k+1}}[\boldsymbol{D} - \boldsymbol{L}_k]_{:,\mathcal{J}}$
  $[\boldsymbol{S}_{k+1}]_{\mathcal{I},:} = \mathcal{T}_{\zeta_{k+1}}[\boldsymbol{D} - \boldsymbol{L}_k]_{\mathcal{I},:}$
  // Phase II: updating low rank component
  $\boldsymbol{C}_{k+1} = [\boldsymbol{D} - \boldsymbol{S}_{k+1}]_{:,\mathcal{J}}$
  $\boldsymbol{U}_{k+1} = \mathcal{H}_r([\boldsymbol{D} - \boldsymbol{S}_{k+1}]_{\mathcal{I},\mathcal{J}})$
  $\boldsymbol{R}_{k+1} = [\boldsymbol{D} - \boldsymbol{S}_{k+1}]_{\mathcal{I},:}$
  $\boldsymbol{L}_{k+1} = \boldsymbol{C}_{k+1}\boldsymbol{U}_{k+1}^{\dagger}\boldsymbol{R}_{k+1}$ // Do not compute this step
  $k = k + 1$
end while
Output: $\boldsymbol{C}_k, \boldsymbol{U}_k, \boldsymbol{R}_k$: CUR decomposition of $\boldsymbol{L}$.

III. Proposed Algorithm

In this section, we develop a novel rapid yet robust algorithm aiming to solve the non-convex RPCA problem (1) directly. Within the general alternating projections framework for RPCA [16], [17], we propose a CUR-accelerated algorithm, dubbed Iterated Robust CUR (IRCUR). As summarized in Algorithm 1, there are two phases at the $(k+1)$–st iteration of IRCUR: (Phase I) we first project $\boldsymbol{D} - \boldsymbol{L}_k$ to the set of sparse matrices via hard thresholding to update the estimate of $\boldsymbol{S}$, (Phase II) then project $\boldsymbol{D} - \boldsymbol{S}_{k+1}$ to the set of low rank matrices via CUR decomposition to update the estimate of $\boldsymbol{L}$. For ease of presentation, we will discuss Phase II of Algorithm 1 first, then address Phase I.

Phase II: Updating the Estimate of $\boldsymbol{L}$. In prior art, a common approach for updating $\boldsymbol{L}$ is to use the truncated SVD, which can be very costly when n is large. Inspired by Theorem 1, we instead employ CUR decomposition as an inexact low rank approximator here. More specifically, we let

$$\boldsymbol{C}_{k+1} = [\boldsymbol{D} - \boldsymbol{S}_{k+1}]_{:,\mathcal{J}}, \quad \boldsymbol{R}_{k+1} = [\boldsymbol{D} - \boldsymbol{S}_{k+1}]_{\mathcal{I},:}, \quad \text{and} \quad \boldsymbol{U}_{k+1} = \mathcal{H}_r([\boldsymbol{D} - \boldsymbol{S}_{k+1}]_{\mathcal{I},\mathcal{J}}) \tag{2}$$

where the indices $\mathcal{I}, \mathcal{J}$ are generated via uniform sampling, and $\mathcal{H}_r(\cdot)$ denotes the best rank r approximation (i.e., truncated SVD) to the argument. So, the updated estimate of $\boldsymbol{L}$, i.e.,

$$\boldsymbol{L}_{k+1} = \boldsymbol{C}_{k+1}\boldsymbol{U}_{k+1}^{\dagger}\boldsymbol{R}_{k+1}, \tag{3}$$

is of rank r. Per Theorem 2, it costs $\mathcal{O}(r^3\log^2(n))$ flops to obtain $\boldsymbol{U}_{k+1}^{\dagger}$. At first glance, it appears to cost $\mathcal{O}(rn^2\log(n))$ flops to form $\boldsymbol{L}_{k+1}$ itself; however, we actually never need to form the entire $\boldsymbol{L}$ through IRCUR, but merely the CUR components of $\boldsymbol{L}_{k+1}$ to be saved and output. Thus, Phase II of IRCUR costs only $\mathcal{O}(r^3\log^2(n))$ flops. Note that other rank truncation methods have been studied for CUR decomposition [25]–[27], but we

Algorithm 2: Efficient Conversion from CUR to SVD.

1: **Input:** $\boldsymbol{C}, \boldsymbol{U}, \boldsymbol{R}$: CUR decomposition of the matrix.
2: $[\boldsymbol{Q}_C, \boldsymbol{R}_C] = \mathrm{qr}(\boldsymbol{C}); \quad [\boldsymbol{Q}_R, \boldsymbol{R}_R] = \mathrm{qr}(\boldsymbol{R}^T)$
3: $[\boldsymbol{W}_U, \boldsymbol{\Sigma}, \boldsymbol{V}_U] = \mathrm{svd}(\boldsymbol{R}_C \boldsymbol{U}^\dagger \boldsymbol{R}_R^T)$
4: $\boldsymbol{W} = \boldsymbol{Q}_C \boldsymbol{W}_U; \quad \boldsymbol{V} = \boldsymbol{Q}_R \boldsymbol{V}_U$
5: **Output:** $\boldsymbol{W}, \boldsymbol{\Sigma}, \boldsymbol{V}$: SVD of the matrix.

have found the novel method proposed here to be the most computationally efficient.

Phase I: Updating the Estimate of $\boldsymbol{S}$. As shown in [8], [16], [17], the projection to the set of sparse matrices can be achieved via the hard thresholding operator $\mathcal{T}_\zeta$ defined as:

$$[\mathcal{T}_\zeta \boldsymbol{X}]_{i,j} = \begin{cases} \boldsymbol{X}_{i,j}, & \text{if } \ |\boldsymbol{X}_{i,j}| > \zeta, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

At each iteration, with properly chosen thresholding value ζ_{k+1}, we can obtain a sparse $\boldsymbol{S}_{k+1}$ while keeping $\|\boldsymbol{S} - \boldsymbol{S}_{k+1}\|_\infty$ under control. One strategy is to pick $\zeta_{k+1} \geq \|\boldsymbol{L} - \boldsymbol{L}_k\|_\infty$, which implies $\mathrm{supp}(\boldsymbol{S}_{k+1}) \subseteq \mathrm{supp}(\boldsymbol{S})$ and $\|\boldsymbol{S} - \boldsymbol{S}_{k+1}\|_\infty \leq 2\zeta_{k+1}$. In practice, we observe that iteratively decaying thresholding values achieves great success with proper parameter tuning.

We now turn our attention to the computational complexity of Phase I. As discussed in Phase II, to form the CUR components for $\boldsymbol{L}_{k+1}$, we only need the submatrices corresponding to the indices $\mathcal{I}$ and $\mathcal{J}$. Therefore, there is no need to apply thresholding on the entire matrix $\boldsymbol{D} - \boldsymbol{L}_k$, but only on the submatrices that we need to pass to Phase II. That is, computing and passing $[\boldsymbol{S}_{k+1}]_{\mathcal{I},:}$ and $[\boldsymbol{S}_{k+1}]_{:,\mathcal{J}}$ is sufficient. Consequently, we only need to have $[\boldsymbol{L}_k]_{\mathcal{I},:}$ and $[\boldsymbol{L}_k]_{:,\mathcal{J}}$ for this calculation, and this is the reason why we emphasize the whole matrices should never be formed in IRCUR. Recall that only the CUR components of $\boldsymbol{L}_k$ have been computed and saved by the previous iteration. To update $\boldsymbol{S}$ efficiently, we compute

$$[\boldsymbol{L}_k]_{\mathcal{I},:} = [\boldsymbol{C}_k]_{\mathcal{I},:} \boldsymbol{U}_k^\dagger \boldsymbol{R}_k \text{ and } [\boldsymbol{L}_k]_{:,\mathcal{J}} = \boldsymbol{C}_k \boldsymbol{U}_k^\dagger [\boldsymbol{R}_k]_{:,\mathcal{J}}, \tag{5}$$

followed by hard thresholding. Since $|\mathcal{I}| = \mathcal{O}(r \log(n))$ and $|\mathcal{J}| = \mathcal{O}(r \log(n))$, the computational complexity for Phase I of IRCUR is $\mathcal{O}(r^2 n \log^2(n))$.

Moreover, the stopping criterion is developed in terms of the related computing error of the sampled submatrices:

$$e_k = \frac{\|[\boldsymbol{D} - \boldsymbol{L}_k - \boldsymbol{S}_k]_{\mathcal{I},:}\|_F + \|[\boldsymbol{D} - \boldsymbol{L}_k - \boldsymbol{S}_k]_{:,\mathcal{J}}\|_F}{\|\boldsymbol{D}_{\mathcal{I},:}\|_F + \|\boldsymbol{D}_{:,\mathcal{J}}\|_F}. \tag{6}$$

Overall, we can just save and process the submatrices through the entire algorithm. In this regard, IRCUR enjoys both a superior computational complexity (i.e., $\mathcal{O}(r^2 n \log^2(n))$ per iteration) and the memory efficiency.

Finally, at the output stage of IRCUR, a CUR decomposition of the estimated $\boldsymbol{L}$ is returned to the user, which allows for better interpretation of the low rank component. In the case of that the user is more interested in the traditional low rank expression (i.e., SVD), we also provide an efficient method for the conversion from CUR decomposition to SVD, which is summarized in Algorithm 2. This conversion involves two $n \times r$ QR decompositions and a $r \times r$ SVD, which lead its complexity to $\mathcal{O}(r^2 n)$. Hence, this conversion between the two low rank representations does not increase the overall computational complexity.

A. Fixed vs. Resampled Indices

It is optional to resample the indices $\mathcal{I}$ and $\mathcal{J}$ in every iteration and produces two variants of IRCUR: IRCUR-F and IRCUR-R for fixed indices and resampled indices, respectively. IRCUR-F has minimal required data access and therefore less runtime in reality, but we can get stuck with bad submatrices if unlucky, although the chance is very low. On the other hand, IRCUR-R uses more redundant data from different submatrices and thus can correct any one-time bad-luck situations. One can expect IRCUR-R to tolerate corruptions better than IRCUR-F. However, more data access may be forbidden in some application scenarios and also result a bit more computing (e.g., $\|\boldsymbol{D}_{\mathcal{I},:}\|_F$ in (6) will be re-computed for every iteration).

Generally speaking, one should use IRCUR-F if the data access is restricted and extremely fast speed is desired; IRCUR-R should be used when data access is inexpensive and best corruption tolerance is needed. We will further study the empirical performance of these two variants in Section IV.

B. Parameter Tuning

There are a few parameters that need to be tuned in IRCUR. First, for the hard thresholding operator to work properly, we need to take care of two parameters: the initial thresholding value ζ_0, and the thresholding decay parameter γ. Intuitively, ζ_0 should be a positive number that helps us to filter out the irregular values immediately at initialization. An ideal choice of ζ_0 is $\|\boldsymbol{L}\|_\infty$, the maximum magnitude of the underlying low rank component, which implies $\mathrm{supp}(\boldsymbol{S}_0) \subseteq \mathrm{supp}(\boldsymbol{S})$ while keeping the entrywise estimation error of $\boldsymbol{S}$ under control. We can easily estimate $\|\boldsymbol{L}\|_\infty$ by prior knowledge in many real-world problems. For example, in the image and video related applications, any pixels that fall out of the normal range of [0255] should be considered outliers immediately.

On the other hand, the decay parameter γ should be a value in (0, 1) that reflects the anticipated convergence rate. Generally speaking, harder problems (e.g., larger μ, r, α, etc.) will require larger γ for successful recovery, while easier problems can be more quickly recovered with smaller γ and also workable with larger γ. Using a larger value for γ may cause slower convergence but can enhance the robustness of IRCUR. Empirically, it is recommended that $\gamma \in [0.6, 0.9]$.

The other parameters to be tuned are the numbers of rows and columns to be sampled. As stated in Theorem 2, it requires $|\mathcal{I}| = c_I r \log(n)$ and $|\mathcal{J}| = c_J r \log(n)$ for some $c_I, c_J > 0$. Typically, in the presence of noise, more rows and columns need to be sampled to provide robustness (e.g., [28]). In our case, larger α increases the minimum requirement of c_I and c_J. We will further study the empirical relationship between α and the sampling constants in Section IV.

IV. Numerical Experiments

In this section, we compare the empirical performance of two versions of IRCUR, with fixed indices (IRCUR-F) and with resampled indices (IRCUR-R), against the state-of-the-art RPCA algorithms, AccAltProj [17] and GD [18]. Due to the page limit,

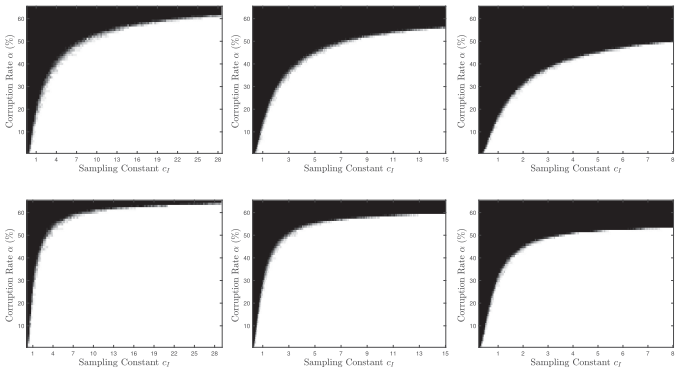

Fig. 1. Empirical phase transition in sampling constant c_I and corruption rate α. A white pixel means all 50 test cases are successfully recovered and a black pixel means all 50 test cases fail the recovery. **Top:** IRCUR-F. **Bottom:** IRCUR-R. **Left:** $r = 5$. **Middle:** $r = 10$. **Right:** $r = 20$.

we defer the details of experimental setting to the supplementary document. Moreover, we provide a sample Matlab code for IRCUR at *https:// github.com/ caesarcai/ IRCUR* .

A. Synthetic Datasets

The experiments in this section follow the setup as in [11], [16]–[18], wherein square $\boldsymbol{D}, \boldsymbol{L}, \boldsymbol{S} \in \mathbb{R}^{n\times n}$ are used for demonstration. Thus, we sample same number of rows and columns in the following experiments, i.e., $|\mathcal{I}| = |\mathcal{J}| = c_I r \log(n)$, but $\mathcal{I}$ may not equal to $\mathcal{J}$.

Empirical Phase Transition: We investigate the recovery ability of IRCUR with different sampling constants c_I and corruption rates α. Taking the problem dimension $n = 1000$, this experiment runs under 3 different rank settings: $r = \{5, 10, 20\}$. For each rank, we conduct 50 random tests for every given pair of (c_I, α), and a recovery is deemed successful if the output of IRCUR satisfies $\|\boldsymbol{C}_k \boldsymbol{U}_k^{\dagger} \boldsymbol{R}_k - \boldsymbol{L}\|_F / \|\boldsymbol{L}\|_F \leq 10^{-3}$. The test results are summarized as Fig. 1, whereas we observe that if more rows/columns are sampled, IRCUR can handle more corruption/outliers. While the increment of sampling constant enhances the robustness rapidly when c_I is small, the tolerance of corruption asymptotically reaches its limit later. This suggests that one should pick a medium value for c_I to ensure robustness and efficient implementation, then $c_I \in [3, 5]$ is a good balanced choice in practice. We also observe that IRCUR-R produces slightly better recovery than IRCUR-F as expected. On the other hand, the maximum tolerance of corruption gets lower as the rank increases since a higher rank results in harder problems.

Computational Efficiency: We evaluate the computational efficiency of the algorithms tested. The experimental settings and results are summarized in Fig. 2. The left subfigure shows that both variants of IRCUR have substantial speed advantage against AccAltProj and GD, especially when n is larger. The middle subfigure confirms that the computational complexity of IRCUR is indeed $\mathcal{O}(n \log^2(n))$ and IRCUR-F is sightly more efficient than IRCUR-R. The right subfigure shows the linear convergence of all the algorithms tested, wherein IRCUR has the lowest runtime per iteration.

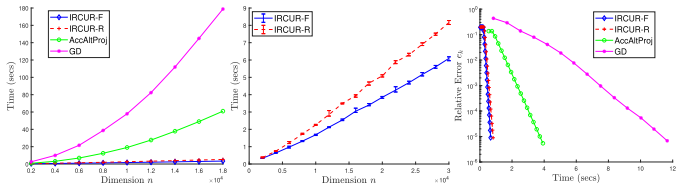

Fig. 2. Runtime comparison. **Left:** dimension n vs runtime: $r = 5$, $\alpha = 0.1$, $c_I = 4$, and $n \in [2000, 18000]$. **Middle:** dimension n vs runtime for only IRCUR: $r = 5$, $\alpha = 0.1$, $c_I = 4$, and $n \in [2000, 30000]$. **Right:** relative error e_k vs runtime: $r = 5$, $\alpha = 0.1$, $c_I = 4$, and $n = 4000$.

Fig. 3. Video background subtraction results. The first corresponds to a frame from *shoppingmall*, and the second row is from *restaurant*. The first column is the original frame, the next four columns are the foreground and background outputted by IRCUR-F and IRCUR-R, respectively.

TABLE I
VIDEO SIZE AND RUNTIME. HERE **S** REPRESENTS THE SHOPPING MALL AND **R** REFERS TO THE RESTAURANT

	frame size	frame number	runtime (sec)			
			IRCUR-F	IRCUR-R	AccAltProj	GD
S	256×320	1000	2.03	2.16	23.04	93.18
R	120×160	3055	0.82	0.88	15.96	58.37

B. Video Background Subtraction

We apply the same algorithms to the task of video background subtraction. Two popular videos, *shoppingmall* and *restaurant*, are used as benchmarks. The video size information and runtime results for all test algorithms are recorded in Table I. We again confirm that both variants of IRCUR are substantially faster than AccAltProj and GD in this real-world benchmark.

Moreover, all the tested algorithms achieve visually desirable results under aforementioned parameter setting. For IRCUR, we present the separated background and foreground for a selected frame from each video in Fig. 3. One can see that both variants of IRCUR enjoy crisp static backgrounds in both videos. Due to the page limit, we defer more visual results to the supplementary document.

V. CONCLUSION AND FUTURE DIRECTION

This letter presents a novel RPCA algorithm, coined IRCUR, that has high computational and memory efficiency. We use a novel CUR decomposition for rapid inexact low rank approximation, which reduces the computational complexity from typical $\mathcal{O}(rn^2)$ to $\mathcal{O}(r^2 n \log^2(n))$. The numerical simulations verify the claimed advantages of IRCUR.

There are two directions for future research. First, we will investigate the theoretical convergence guarantee of IRCUR in the future. Second, it is interesting to study the recovery stability of IRCUR to additive dense noise.

REFERENCES

[1] Y. Peng, A. Ganesh, J. Wright, W. Xu, and Y. Ma, "RASL: Robust alignment by sparse and low-rank decomposition for linearly correlated images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2233–2246, Nov. 2012.

[2] W. Song, J. Zhu, Y. Li, and C. Chen, "Image alignment by online robust PCA via stochastic gradient descent," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 7, pp. 1241–1250, Jul. 2016.

[3] X. Luan, B. Fang, L. Liu, W. Yang, and J. Qian, "Extracting sparse error of robust PCA for face recognition in the presence of varying illumination and occlusion," *Pattern Recognit.*, vol. 47, no. 2, pp. 495–508, 2014.

[4] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb. 2008.

[5] Y. Hu, J.-X. Liu, Y.-L. Gao, and J. Shang, "DSTPCA: Double-sparse constrained tensor principal component analysis method for feature selection," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, early access, Sep. 24, 2019, doi: 10.1109/TCBB.2019.2943459.

[6] J.-X. Liu, Y. Xu, C.-H. Zheng, H. Kong, and Z.-H. Lai, "RPCA-based tumor classification using gene expression data," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 12, no. 4, pp. 964–970, Jul./Aug. 2015.

[7] Y. Chen, S. Sanghavi, and H. Xu, "Clustering sparse graphs," in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2012, pp. 2204–2212.

[8] H. Cai, J.-F. Cai, T. Wang, and G. Yin, "Fast and robust spectrally sparse signal recovery: A provable non-convex approach via robust low-rank Hankel matrix reconstruction," *arXiv:1910.05859*.

[9] W.-D. Jang, C. Lee, and C.-S. Kim, "Primary object segmentation in videos via alternate convex optimization of foreground and background distributions," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 696–704.

[10] B. E. Moore, C. Gao, and R. R. Nadakuditi, "Panoramic robust PCA for foreground–background separation on noisy, free-motion camera video," *IEEE Trans. Comput. Imag.*, vol. 5, no. 2, pp. 195–211, Jun. 2019.

[11] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?," *J. Assoc. Comput. Mach.*, vol. 58, no. 3, pp. 1–37, 2011.

[12] H. Cai, "Accelerating truncated singular-value decomposition: a fast and provable method for robust principal component analysis," Ph.D. dissertation, Dept. Math., Univ. Iowa, Iowa City, IA, USA, 2018.

[13] H. Xu, C. Caramanis, and S. Sanghavi, "Robust PCA via outlier pursuit," in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2010, pp. 2496–2504.

[14] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky, "Rank-sparsity incoherence for matrix decomposition," *SIAM J. Optim.*, vol. 21, no. 2, pp. 572–596, 2011.

[15] S. Ma and N. S. Aybat, "Efficient optimization algorithms for robust principal component analysis and its variants," *Proc. IEEE*, vol. 106, no. 8, pp. 1411–1426, Aug. 2018.

[16] P. Netrapalli, U. Niranjan, S. Sanghavi, A. Anandkumar, and P. Jain, "Non-convex robust PCA," in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2014, pp. 1107–1115.

[17] H. Cai, J.-F. Cai, and K. Wei, "Accelerated alternating projections for robust principal component analysis," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 685–717, 2019.

[18] X. Yi, D. Park, Y. Chen, and C. Caramanis, "Fast algorithms for robust PCA via gradient descent," in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4152–4160.

[19] T. Tong, C. Ma, and Y. Chi, "Accelerating ill-conditioned low-rank matrix estimation via scaled gradient descent," *arXiv:2005.08898*.

[20] M. W. Mahoney and P. Drineas, "CUR matrix decompositions for improved data analysis," *Proc. Nat. Acad. Sci.*, vol. 106, no. 3, pp. 697–702, 2009.

[21] K. Hamm and L. Huang, "Perspectives on CUR decompositions," *Appl. Comput. Harmon. Anal.*, vol. 48, no. 3, pp. 1088–1099, 2020.

[22] J. Chiu and L. Demanet, "Sublinear randomized algorithms for skeleton decompositions," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 3, pp. 1361–1383, 2013.

[23] K. Hamm and L. Huang, "Stability of sampling for CUR decompositions," *Found. Data Sci.*, vol. 2, no. 2, pp. 83–99, 2020.

[24] J. Bien, Y. Xu, and M. W. Mahoney, "CUR from a sparse optimization viewpoint," in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2010, pp. 217–225.

[25] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Fixed-rank approximation of a positive-semidefinite matrix from streaming data," in *Proc. Adv. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1225–1234.

[26] F. Pourkamali-Anaraki and S. Becker, "Improved fixed-rank Nyström approximation via QR decomposition: Practical and theoretical aspects," *Neurocomputing*, vol. 363, pp. 261–272, 2019.

[27] K. Hamm and L. Huang, "Perturbations of CUR decompositions," *arXiv:1908.08101*.

[28] A. Aldroubi, K. Hamm, A. B. Koku, and A. Sekmen, "CUR decompositions, similarity matrices, and subspace clustering," *Front. Appl. Math. Statist.*, vol. 4, no. 65, pp. 1–16, 2019.