Computing Robust Principal Components by A* Search

Swair Shah, Baokun He, Crystal Maung, and Haim Schweitzer
Department of Computer Science, University of Texas (Dallas)
800 W Campbell Road, Richardson, TX 75080
Email: swair@utdallas.edu, bxh150730@utdallas.edu, crystal.maung@gmail.com, hschweitzer@utdallas.edu

***Abstract*—Principal Component Analysis (PCA) is a classic dimensionality reduction technique that computes a low rank representation of the data. Recent studies have shown how to compute this low rank representation from most of the data, excluding a small amount of outlier data. We describe an algorithm that solve this problem by applying a variant of the A* algorithm to search for the outliers. The results obtained by our algorithm are optimal, and more accurate than the current state of the art. This comes at the cost of running time, which is typically slower than the current state of the art. We also describe a related variant of the A* algorithm that runs much faster and produces a solution that is guaranteed to be near the optimal.**

I. Introduction

Principal component analysis (PCA) is arguably the most widely used dimensionality reduction technique. Given a data matrix X of size $m \times n$ and $r \leq m$, the PCA approach computes an orthogonal matrix V of size $m \times r$ (the principal components) and a coefficients matrix A of size $r \times n$ such that:

$$X \approx VA \tag{1}$$

For each column x_i of X this approximation implies:

$$x_i \approx Va_i, \quad i = 1, \ldots, n \tag{2}$$

where a_i is the ith column of A. Thus, when the error of the approximations in (2) is small, one can use the r dimensional vectors a_i as representatives of the m dimensional vectors x_i in the vector space spanned by V. See [1] for many applications of the PCA technique.

Consider as an example the case where $m = 2$ and $r = 1$. In this case the columns of the $2 \times n$ matrix X can be visualized as 2-dimensional points, and the matrix V consists of a single column vector that specifies a 2-dimensional direction. This is illustrated in Figure 1, where the direction V is a line that passes through the origin and approximately passes through all of the points.

The quality of the representation in the reduced dimension depends on the quality of the approximations in (1) and (2). It is known (e.g., [2]) that the reduction to r dimensions that minimizes the approximation error of (1) in the Frobenius norm can be obtained by selecting the r columns of V as the r eigenvectors corresponding to the r largest eigenvalues of the matrix $B = XX^T$.

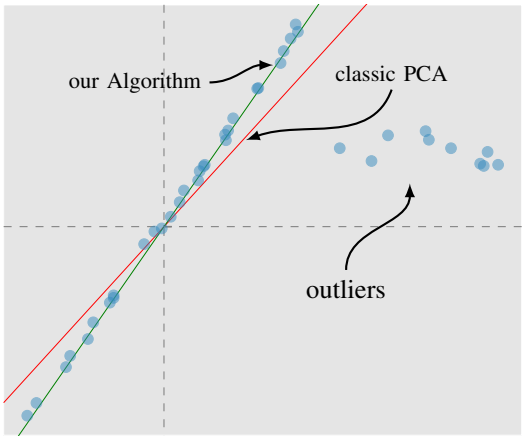

Fig. 1. Visualizing PCA of 2-dimensional points

Recent studies have considered many robust variants of the PCA technique. See [3] for a recent survey with nearly 500 references. We discuss here in detail two important variants. The first variant approximates X as the sum of a sparse matrix and a low rank matrix. See, e.g., [4], [5]. In matrix notation this approximation can be written as:

$$X \approx S + VA \tag{3}$$

where S is sparse and V is $m \times r$. In vector notation:

$$x_i \approx s_i + Va_i, \quad i = 1, \ldots, n \tag{4}$$

where the s_i are sparse. This variant is a good model in cases where the likelihood that a matrix entry is wrong is independent of its location in the matrix. Unlike the first variant, the second variant assumes that the errors are concentrated in a small number of matrix columns, considered to be "outliers". See, e.g., [6], [7]. Let Q be the indicator set of the outlier columns and let P be the indicator set of all the other columns. Let X_P be the matrix created from the columns in P. In matrix notation the approximation is given by:

$$X_P \approx VA \tag{5}$$

where V is $m \times r$. In vector notation:

$$x_i \approx Va_i, \quad i \in P \tag{6}$$

As illustrated in Figure 1, the approximation obtained by PCA can be significantly improved if the approximation is not required to include the outliers.

In this paper we propose two algorithms for the second variant described above. The algorithms use variants of A* search, a classic heuristic search technique in AI. We prove that the first algorithm computes the best possible solution, and that the second algorithm computes a solution that cannot be too far from the best solution. Such performance guarantees are not available for previously proposed algorithm that address the same problem.

The paper is organized as follows. A precise definition of the problem being addressed is given in Section II, which also includes a review of previous work. The main tools that are used to design our algorithms are described in Section III. They include the A* algorithm and specialized eigenvalue routines. The main ideas behind our approach are discussed in Section IV. They include a definition of a subset graph, and the generic A* algorithm that searches the graph. The heuristics that guarantee the performance of the A* algorithm are described in Section V, which also includes a proof of the algorithm optimality. Experimental results are shown in Section VI.

II. The problem being addressed

We consider the following optimization problem. The input is the matrix X of size $m \times n$, a number $k \leq n$ of outliers, and the desired number $r \leq \min(m, n-k)$ of principal components. The output is the index subset P such that $|P| = n - k$, an orthogonal matrix V of size $m \times r$, and a matrix A of size $r \times (n-k)$. The output is computed to minimize the following error:

$$e(P, V, A) = \|X_P - VA\|_F^2 \tag{7}$$

In the above equation X_P is the matrix consisting of the columns of X with index values in P. It is easy to see that when Q, the set of k outlier columns is known, the values of $P, V.A$ that minimize (7) can be easily calculated as follows:

$$\begin{aligned} &P = \{1, \ldots, n\} \setminus Q. \\ &V \text{ is the matrix of the } r \text{ principal components of } X_P. \\ &A = \arg\min \|X_P - VA\|_F^2 = V^+ X_P. \end{aligned} \tag{8}$$

where V^+ is the pseudoinverse of V. This shows that the error in (7) is a function of Q:

$$e(Q) = e(P, V, A) = \|X_P - VA\|_F^2 \tag{9}$$

Therefore, in this variant of robust PCA the challenge is to identify the outlier columns. Since there are $\binom{n}{k}$ possible choices of k column subsets, the exhaustive search approach of evaluating the errors of all such subsets and selecting the smallest one may be unacceptably slow. To the best of our knowledge this paper is the first to propose an algorithm that is guaranteed to find the optimal solution, and runs significantly faster than exhaustive search. We briefly describe previous approaches that address the same problem, but do not guarantee an optimal solution. (In fact, with non-optimal algorithms one has no way of telling whether or not the result is optimal.)

Instead of searching for the outliers, some robust estimation approaches attempt to minimize their influence on the predicted model. For example, in the line fitting example shown in Figure 1 one may attempt to evaluate the error in terms of absolute deviations instead of squared deviations as implied by the Frobenius norm. See, e.g., [8], [9].

A different technique of identifying outliers is based on "leverage scores". Let $X = V\Sigma U^T$ be the SVD of the matrix X. Then U is an $n \times m$ matrix with orthonormal columns. Let $u_1, \ldots, u_k$ be the columns of U corresponding to the largest singular values. Define the leverage score of Column i as:

$$\text{leverage\_score}(i) = \sum_{j=1}^{k} (u_j(i))^2 \quad \text{for } i = 1, \ldots, n \tag{10}$$

It is known that outlier columns have high leverage score, so that the k largest leverage score values identify the desired k outliers. See, e.g., [10], [11].

A recent approach to outlier detection considers a relaxation of the optimization problem in (7) that leads to a convex optimization problem. See [6], [7] for additional details. An experimental comparison between this method and our approach is shown in Section VI.

III. The main tools

Our algorithms are based on the classic A* algorithm with heuristics that require an efficient computation of eigenvalues. These tools are reviewed in this section.

A* algorithms for searching graphs. A* search is a well known heuristic search algorithm for graphs. See, e.g., [12], [13], [14]. In its standard formulation the algorithm computes a path between two given nodes, guided by a heuristic function h_i that can be computed at node n_i. The algorithm expands nodes according to a criterion f_i, which is computed from h_i by the following formula:

$$f_i = g_i + h_i$$

In the above formula g_i is the distance between the initial node and n_i, and the heuristic function h_i is problem specific. The optimality of A* depends on particular properties of h_i. Consistency (see references for definition) of h_i guarantees optimality in a fast variant where each graph node is visited at most once. The weaker condition of admissibility (see references for definition) is sufficient to guarantees optimality in a slower variant that may visit the same node multiple times.

Weighted A* algorithms for searching graphs. The only difference between the A* algorithm and the weighted A* algorithm is a slightly different method of combining the heuristics. Specifically, the algorithm expands nodes according to a criterion $f_i'(\epsilon)$ which is given by the following formula:

$$f_i'(\epsilon) = f_i + \epsilon h_i = g_i + (1 + \epsilon) h_i$$

For $\epsilon > 0$ the weighted A* algorithm is not guaranteed to find an optimal solution, but it typically runs significantly faster than the A* algorithm. It is guaranteed to find a solution within $(1+\epsilon)$ of the optimal. See, e.g., [13].

Rank-one update eigenvalue calculations. The heuristic evaluations in our algorithms use eigenvalues. Let X_0 be an $m \times k$ matrix, and let X_1 be an $m \times (k-1)$ matrix, constructed by removing the single column x from X_0. Suppose we are given the eigenvalues and the eigenvectors of the matrix $B_0 = X_0 X_0^T$. The heuristics in our algorithms require efficient calculations of the eigenvalues of $B_1 = X_1 X_1^T = B_0 - xx^T$.

This rank-one update problem has attracted a lot of attention. It was shown in [15] that the eigenvalues of B_1 are roots of a "secular" equation, that can be constructed from the eigenvalues and the eigenvectors of B_0 and x. Studies of efficient numeric procedures for computing the roots of the secular equation include [16], [17], [18], [19]. In our experiments we use the Gragg method, as described in [18], [19].

IV. Our approach

As shown in Equation (9) it is possible to assign an error value to each column subset of X. Our main idea is to consider a graph that describes the relationship between these subsets, and then perform a graph search for a subset of size k that has the smallest error. The subset graph that we create is the same as the one used in [20], [21]. The search techniques that we propose are also similar to those described in these studies. Specifically, we use a variant of the A* algorithm to compute the optimal solution, and a variant of the weighted A* algorithm to compute a solution with guaranteed bounds on suboptimality. The main difference between these previous studies and the current work is the heuristics that are being used. We propose heuristic functions that are specifically designed to compute the outliers, and provide proofs of optimality and suboptimality for the algorithms that use these heuristic functions.

The subset graph. We create the same column subset graph as in [20], [21], where nodes correspond to column subsets, and there is an edge from subset Q_i to subset Q_j if adding a single column to Q_i creates Q_j. As an example, the graph associated for the matrix $X = (x_1, x_2, x_3)$ is shown in Figure 2.

The A* Algorithm. The algorithm in Figure 3 performs search for column subsets. It is a generic version of the A* algorithm that uses a closed nodes list (and thus visits each node at most once). The algorithm maintains two lists: the list L contains nodes to be examined and the list C contains nodes which are marked as closed and need not be visited again. As we show in Section V, by properly defining the function f' the algorithm finds the desired outliers.

V. The heuristics

In this section we describe the heuristic functions that are needed by the A* algorithm in order to compute the outliers. In typical usage of A* it is enough to define the function h_i

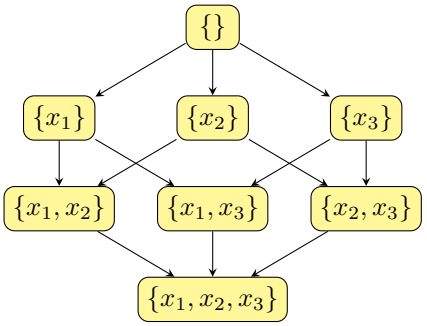

Fig. 2. An example of the column subset graph of 3 columns

Input: X,k,r. **Output:** a column subset Q.
Each node n_i has a subset Q_i of size k_i. The algorithm maintains two lists: the fringe list L, and the closed nodes list C.
Initialization: Put an empty column subset into L.

```
while L is nonempty do
    Pick n_i with the smallest f'_i from L.
    if k_i = k then
        Stop and return n_j with Q_i as the solution subset.
    else
        Add n_i to C.
        for each child n_j of n_i do
            if n_j is not in C or L then
                Compute f'_j and put n_j in L.
            end
        end
    end
end
```

Fig. 3. The generic A* algorithm for column subsets

and then combine it with g_i to construct f_i. In our case there is no obvious choice for g_i and h_i. Instead, we provide a definition for g_i and f_i without using h_i. Since our definition of the heuristic functions is nonstandard, the optimality of the algorithm does not follow from the known theory and must be proved explicitly.

Recall that at the node i of the graph there is a column subset Q_i of k_i columns. We proceed to define the values of f_i and g_i. To simplify notation we write $|M|$ for the number of columns of a matrix M.

$$\begin{aligned} &P_i = \{1, \ldots, n\} \setminus Q_i. \\ &g_i = \min_{V,A} \|X_{P_i} - VA\|_F^2, \text{ subject to } |V| = r \\ &f_i = \min_{U,A} \|X_{P_i} - UA\|_F^2, \text{ subject to } |U| = r + k - k_i \end{aligned} \tag{11}$$

In the above equation X_{P_i} is the matrix created from the columns of X in the subset P_i.

As proved later the result of running A* with f_i as defined in (11) is always optimal. To define the function $f_i'(\epsilon)$ that gives the weighted A* we follow the derivation in [21] and

define it in terms of a general nonnegative function μ_i. As in [21] we use $\mu_i = g_i$ in our experiments. Let μ_i be a nonnegative function which can be evaluated at each node n_i. Define:

$$f'_i(\epsilon) = f_i + \epsilon\mu_i, \quad \epsilon \geq 0 \tag{12}$$

We proceed to show that using $f'(\epsilon)$ as defined in (12) guarantees sub-optimality. The optimality result for f will follow by setting $\epsilon = 0$.

The suboptimality theorem. Let n_* be an optimal solution node with the corresponding values Q^*, P^*, V^*, A^* that minimize the error in (9). Suppose the algorithm in Figure 3 terminates at a node n_{**} with the corresponding values $Q^{**}, P^{**}, V^{**}, A^{**}$. Then:

$$\|X_{P^{**}} - V^{**}A^{**}\|_F^2 \leq \|X_{P^*} - V^*A^*\|_F^2 + \epsilon\mu_{\max} \tag{13}$$

where $\mu_{\max} = \max_i \mu_i$.

The proof will be given in terms of claim 4 stated below. Claims 2 and 3 are used in the proof of Claim 4. Claim 1 is used in the proof of Claim 2. In the proofs we use the following notation. We write $|M|$ for the number of columns of a matrix M, and $[M|x]$ for the matrix formed by appending a column x to the matrix M.

Claim 1. Let $[X|x]$ be the matrix formed by appending a column x to the matrix X, and let $[U|u]$ be the matrix formed by appending a column u to the matrix U. Then for any $t \geq 0$:

$$\min_{A,|U|=t} \|X - UA\|_F^2 \geq \min_{A',u,|U|=t} \|[X|x] - [U|u]A'\|_F^2$$

Proof. Let U_1, A_1 be the minimizers of the left hand side. Take $U_2 = U_1$, $u_2 = x$, and $A_2 = \begin{pmatrix} A_1 & 0 \\ 0 & 1 \end{pmatrix}$. Then:

$$\begin{aligned}
&\min_{A',u,|U|=t} \|[X|x] - [U|u]A'\|_F^2 \\
&\quad \leq \|[X|x] - [U_2|u_2]A_2\|_F^2 \\
&\quad = \|[X|x] - [U_1A_1|x]\|_F^2 \\
&\quad = \|[X - U_1A_1\|_F^2 \\
&\quad = \min_{A,|U|=t} \|X - UA\|_F^2
\end{aligned}$$

□

Claim 2. f_i is monotonically increasing along any path.

Proof. We need to show that if n_j is the child of n_i then $f_j \geq f_i$. Plugging in the definition of f_i as given in (11) we need to show:

$$\min_{A,|U|=r+k-k_j} \|X_{P_j} - UA\|_F^2 \geq \min_{A',|U|=r+k-k_i} \|X_{P_i} - UA'\|_F^2$$

Since n_j is the child of n_i the following two properties hold: $k_j = k_i + 1$, and $[X_{P_j}|x] = X_{P_i}$, where x is a column of X. Therefore, we need to prove:

$$\begin{aligned}
&\min_{A,|U|=r+k-k_i-1} \|X_{P_j} - UA\|_F^2 \\
&\geq \min_{A',|U|=r+k-k_i-1} \|[X_{P_j}|x] - [U|u]A'\|_F^2
\end{aligned}$$

This follows from Claim 1 with $t = r + k - k_i - 1$. □

Claim 3. For any goal node n_i the following holds: $g_i = f_i$.

Proof. At a goal node $k_i = k$, and the result follows trivially from (11). □

Claim 4. Suppose the theorem is false. Then for any node n_z on the path from the root to n_* the following condition holds: $f'_z < f'_{**}$.

Proof. The assumption that the theorem is false can be written as $g_{**} > g_* + \epsilon\mu_{\max}$. The claim can now be proved by the following chain of equalities / inequalities.

$$\begin{aligned}
f'_{**} &= f_{**} + \epsilon\mu_{**} = g_{**} + \epsilon\mu_{**} && \text{(from Claim 3)} \\
&> g_* + \epsilon\mu_{\max} + \epsilon\mu_{**} && \text{(from the assumption)} \\
&= f_* + \epsilon\mu_{\max} + \epsilon\mu_{**} && \text{(from Claim 3)} \\
&\geq f_z(r,k) + \epsilon\mu_{\max} && \text{(from Claim 2)} \\
&= f'_z
\end{aligned}$$

□

Proof of the suboptimality theorem. If the theorem is false then from Claim 4 it follows that all nodes on the path from the root to n_* have smaller f' values than f'_{**}. Since at any given time at least one of them is in the fringe list, they should all be selected before n_{**} is selected. But this means that n_* is selected as the solution and not n_{**}. ■

Clearly, running the algorithm with $\epsilon = 0$ produces an optimal solution.

The optimality theorem. Let n_* be an optimal solution node with the corresponding values Q^*, P^*, V^*, A^* that minimize the error in (9). Suppose the algorithm in Figure 3 is applied with $\epsilon = 0$ and the algorithm terminates at a node n_{**} with the corresponding values $Q^{**}, P^{**}, V^{**}, A^{**}$. Then:

$$\|X_{P^{**}} - V^{**}A^{**}\|_F^2 = \|X_{P^*} - V^*A^*\|_F^2$$

Running time. The running time of the algorithm depends heavily on the calculation of the heuristic values f_i, as defined in (11). In this section we show that the computation involves estimating sums of eigenvalues, and can be done efficiently for the problem at hand. The function f_i is defined as follows:

$$f_i = \min_{U,A} \|X_{P_i} - UA\|_F^2, \text{ subject to } |U| = r + k - k_i$$

Without loss of generality we can assume that U is an orthogonal matrix, which gives the following formula for the minimizer: $A = U^T X_{P_i}$. With the definition $B_i = X_{P_i} X_{P_i}^T$, and exploiting the relationship between the Frobenius norm and the trace operator one can derive the following expression:

$$\begin{aligned}
f_i &= \text{trace}\{(X_{P_i} - UU^T X_{Pi})(X_{P_i} - UU^T X_{Pi})^T\} \\
&= \text{trace}\{B_i\} - \text{trace}\{U^T B_i U\}
\end{aligned}$$

lung cancer dataset ($m = 27$ $n = 57$)		
k	$A_e(k)$ for $r = 2$	$A_e(k)$ for $r = 3$
0 (baseline)	15.438	13.217
1	15.133	12.828
2	14.845	12.455
4	14.213	11.736
6	13.580	10.982
8	12.836	10.081

TABLE II
REDUCTION IN AVERAGE ERROR PER NON-OUTLIER POINT

These traces can be expressed in terms of the eigenvalues of B_i:

$$\text{trace}\{B_i\} = \sum_{t=1}^{m} \lambda_t, \quad \text{trace}\{U^T B_i U\} = \sum_{t=1}^{r+k-k_i} \lambda_t$$

Therefore:

$$f_i = \sum_{t=r+k-k_i+1}^{m} \lambda_t$$

The most expensive computation step of the algorithm is in Line 9, where the function f_j needs to be computed for all the children of n_i. This requires computing eigenvalues of many matrices of the form: $B_j = X_{P_j} X_{P_j}^T = B_i - x_j x_j^T$ where x_j is a column of X. As discussed in Section III this can be done efficiently using rank-one update algorithms.

VI. EXPERIMENTAL RESULTS

In this section we describe the experimental results which verify the applicability of our method on real datasets and show comparison with the Outlier Pursuit and the Leverage Score method. We use the following data sets from UCI Machine Learning repository [22]: *lung cancer, vehicle, spectf* and *libras*. We also show performance on synthetic datasets where number of outliers is known. Finally we show the result on datasets of facial images taken from the Yale Face Database [23]. For the sub-optimal case we use g_i as the function μ_i following [21].

First we show the reduction in the error averaged over non-outlier points as a function k (the number of outliers removed),

$$A_e(k) = \min_{A, V=r} \frac{\|X_P - VA\|_F^2}{(n-k)}$$

The results are shown in Table II. We pick k outlier data points from the *lung cancer* dataset and show the reduction in $A_e(k)$ for $r = 2$ and $r = 3$. First row for $k = 0$ corresponds to the baseline error, when none of the outliers are removed. We can see that $A_e(k)$ reduces with increasing k thus removing outliers results in more accurate principal components.

Accuracy and Speed Comparison

Next we describe the results for accuracy and speed for optimal A^*, sub-optimal A^* with various values of parameter ϵ, Outlier Pursuit [6] and the leverage method. Outlier Pursuit takes the data matrix X as input and a parameter λ as discussed in [6]. It returns two matrices L and C, where left singular vectors of L form the underlying subspace for non-outlier points and columns of C contain the outliers. As described in [6] we take columns of C with k largest l_2 norms as outliers. Outlier Pursuit does not have a way to pass parameter r so we only use parameter k.

The error for all algorithms is measured as the normalized error,

$$\min_{V,A} \frac{\|X_P - VA\|_F^2}{\|X\|_F^2}$$

where X_P is the matrix of just non-outlier columns and V is of rank r. The time is measured in seconds. We point out that just for this comparison the algorithms are run on transposed matrix thus picking outlier features but the principal for picking outliers remains the same. When the algorithm takes more than 100 seconds to run we don't run it further and it is shown by a $-$ in Table I.

We can see that A^* always has the least error but takes more time to run than the others. For example on dataset *vehicle* with parameters $k = 10, r = 5$, Outlier Pursuit gets a result that is 5 times worse than the optimal result. Though Outlier pursuit takes 0.65 seconds to run where as A^* takes 9.07 seconds to produce optimal solution.

Suboptimal A^* is much faster and can achieve an error as low as A^* and is typically much better than Outlier Pursuit. For dataset $vehicle$ with $k = 10, r = 2$ Outlier Pursuit gets a result with two times the error as the Suboptimal A^* with $\epsilon = 2$. Even when the Suboptimal A^* runs three times faster than Outlier Pursuit.

Most of the time the Suboptimal A^* also beats Outlier Pursuit for accuracy and speed. We observe that Leverage Score method is the best in terms of speed as it only requires just one singular value decomposition though it is typically much worse in terms or accuracy as compared to other algorithms.

Comparison on Synthetic Dataset

Since for real datasets with high dimensional points it is difficult to know the exact outliers, we conduct some experiments on synthetic dataset. We follow [6] to generate dataset with a specified rank of non-outlier points and a specified number of outliers. Since A^* is typically slower than sub-optimal A^* we use the latter one and compare it to Outlier Pursuit algorithm. The results for sub-optimal A^* with $\epsilon = 10$ and Outlier Pursuit are shown in Fig 7. Each dataset is of dimension 50×50 and has 15 outliers. The rank of non-outlier points is shown on top of each plot. Since Outlier Pursuit does not take as input the desired rank (the r parameter in our algorithm), we first run their algorithm for a specific value of k. We get L and C as described in [6]. The rank of L (which contains the non-outlier points) is passed as r for our algorithm. Both algorithm have the same value of parameter k for each run. Error is the relative error as described in previous section. We can see that the Suboptimal A^* always achieves a lower error than Outlier Pursuit. An interesting observation is that as the rank of the space of non-outlier points increases, the accuracy of sub-optimal A^* starts to get much better than Outlier Pursuit. For these datasets sub-optimal A^* ran at least three times faster than Outlier Pursuit.

k	r	A^* / time	Sub-optimal $\epsilon = 2$ / time	Sub-optimal $\epsilon = 5$ / time	Sub-optimal $\epsilon = 10$ / time	Outlier Pursuit / time	Leverage Score Method / time
vehicle dataset ($n = 18$, $m = 846$)							
5	2	**5.790E-04** / 1.01	5.910E-04 / 0.22	5.812E-04 / 0.22	**5.790E-04** / 0.22	8.098E-04 / 0.65	8.230E-04 / 0.01
5	3	**3.121E-04** / 0.69	3.442E-04 / 0.22	3.493E-04 / 0.22	3.493E-04 / 0.22	5.604E-04 / 0.64	5.871E-04 / 0.01
10	2	**1.227E-04** / 30.83	**1.227E-04** / 0.34	**1.227E-04** / 0.23	**1.227E-04** / 0.23	2.602E-04 / 0.65	4.138E-04 / 0.01
10	3	**5.820E-05** / 20.91	**5.820E-05** / 0.24	**5.820E-05** / 0.23	**5.820E-05** / 0.23	1.332E-04 / 0.65	2.506E-04 / 0.01
5	5	**9.842E-05** / 0.36	**9.842E-05** / 0.21	**9.842E-05** / 0.22	**9.842E-05** / 0.22	2.379E-04 / 0.64	2.509E-04 / 0.01
10	5	**8.550E-06** / 9.07	8.735E-06 / 0.22	8.735E-06 / 0.22	8.735E-06 / 0.22	4.898E-05 / 0.65	7.723E-05 / 0.01
spectf dataset ($n = 45$, $m = 267$)							
3	2	**1.042E-02** / 2.33	**1.042E-02** / 0.53	**1.042E-02** / 0.26	**1.042E-02** / 0.25	**1.042E-02** / 0.64	1.125E-02 / 0.01
4	2	**9.996E-03** / 84.01	**9.996E-03** / 0.77	**9.996E-03** / 0.26	**9.996E-03** / 0.26	1.013E-02 / 0.66	1.096E-02 / 0.01
10	3	-	6.128E-03 / 0.31	**6.113E-03** / 0.28	**6.113E-03** / 0.29	6.171E-03 / 0.65	7.340E-03 / 0.01
15	3	-	**4.768E-03** / 0.31	**4.768E-03** / 0.30	**4.768E-03** / 0.31	4.806E-03 / 0.65	6.201E-03 / 0.01
15	10	-	1.734E-03 / 0.32	1.734E-03 / 0.32	**1.713E-03** / 0.33	1.841E-03 / 0.65	2.239E-03 / 0.01
20	10	-	1.125E-03 / 0.34	**1.116E-03** / 0.34	**1.116E-03** / 0.34	1.150E-03 / 0.64	1.626E-03 / 0.01
libras dataset ($n = 90$, $m = 360$)							
4	3	-	-	**4.011E-02** / 93.76	**4.011E-02** / 0.83	4.166E-02 / 6.16	4.292E-02 / 0.01
10	3	-	-	**3.189E-02** / 1.28	**3.189E-02** / 0.92	3.550E-02 / 6.27	3.995E-02 / 0.01
10	4	-	-	**2.033E-02** / 47.23	**2.033E-02** / 0.92	2.198E-02 / 6.22	2.411E-02 / 0.01
15	4	-	-	-	**1.770E-02** / 1.02	2.009E-02 / 6.24	2.150E-02 / 0.01
15	10	-	-	**1.471E-03** / 1.06	**1.471E-03** / 0.98	1.769E-03 / 6.18	2.390E-03 / 0.01
20	10	-	-	**1.060E-03** / 1.11	**1.060E-03** / 1.11	1.469E-03 / 6.51	2.166E-03 / 0.01

TABLE I

ACCURACY AND TIME RESULTS FOR OPTIMAL A^* AND SUB-OPTIMAL A^* AND COMPARISON TO THE OUTLIER PURSUIT [6] AND THE LEVERAGE METHOD. THE TIME IS MEASURED IN SECONDS AND THE ERROR IS THE RELATIVE ERROR $\|X_P - VA\|_F^2/\|X\|_F^2$. THE MINIMUM ERROR IS HIGHLIGHTED.

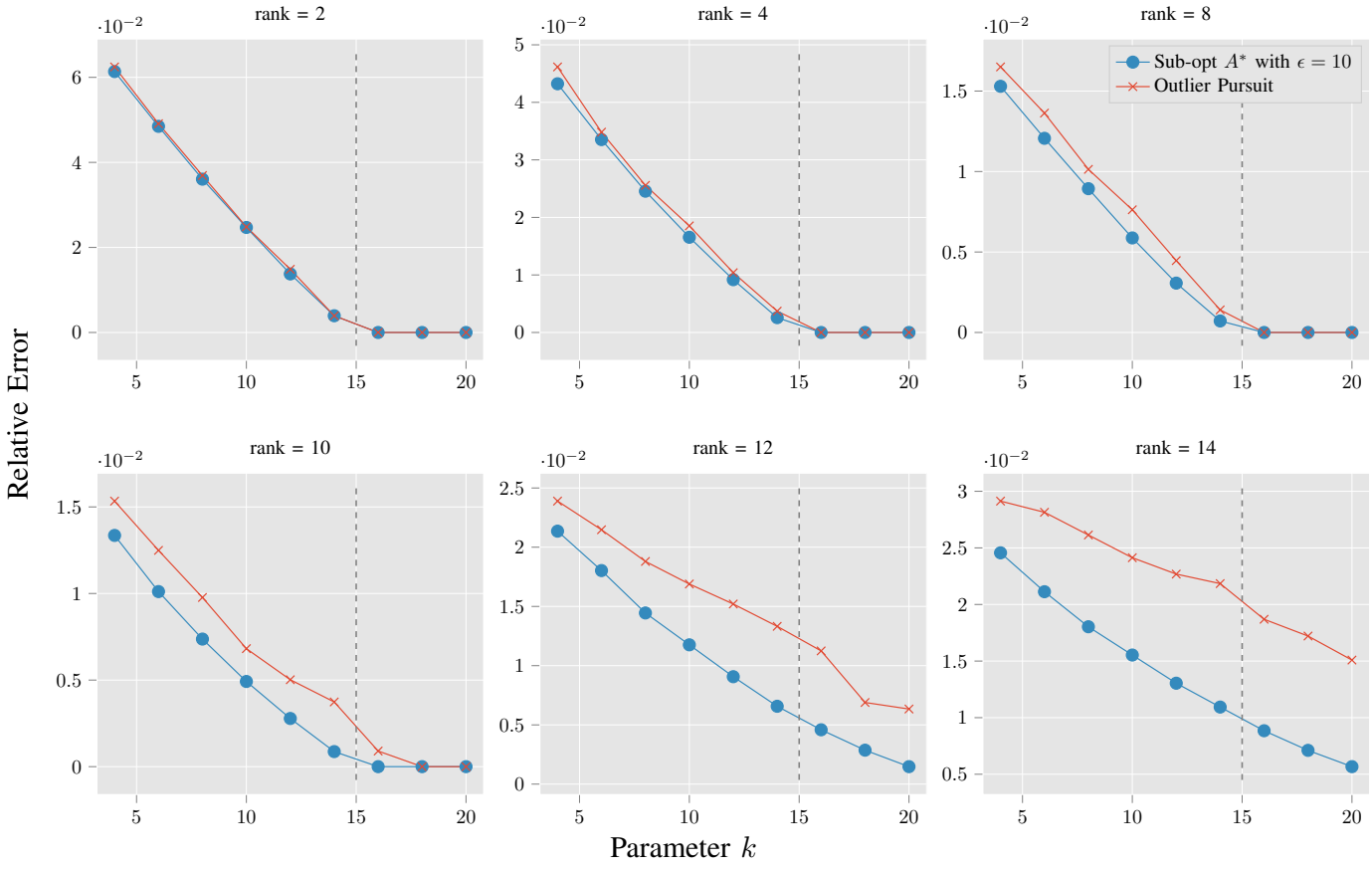

Fig. 7. X is of shape 50×50. Each dataset has 15 outliers and rank of non-outlier points is displayed above each graph.

Fig. 4. Face Dataset 1. Results with $r = 2$ and $k = 4$

.

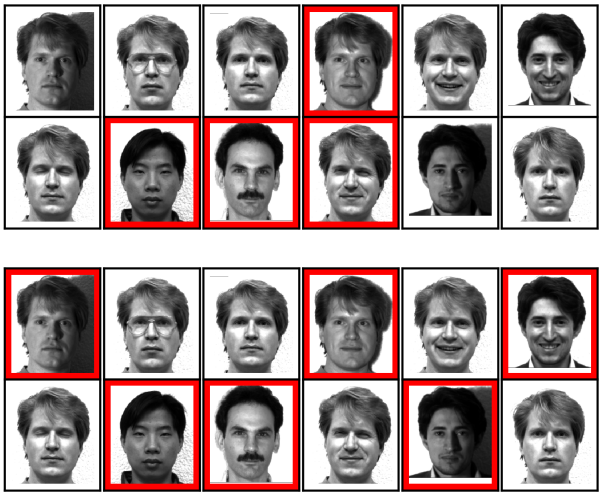

Fig. 5. Face Dataset 1. Top: Results for $r = 3$ and $k = 4$, Bottom : Results for $r = 3$ and $k = 6$

Results on Datasets of Facial Images

We now discuss the results of our algorithm on a dataset of faces. We use the Yale Face Database [23] to create 3 datasets, each with 12 face pictures. Out of those the non-outlier columns are pictures of the same person under different poses, illumination and sometimes with and without glasses. The outlier pictures are pictures of other people. It is known in such a setting the non-outlier images lie in the same low dimensional subspace. We use our algorithm with different values of k and r and show that we successfully detect the low dimensional subspace spanning the non-outlier images.

The results are displayed in Fig 4, 5 and 6. Images with red borders are outliers detected by algorithm. Fig 4 shows the result on first dataset with $r = 2, k = 4$. We can see that our algorithm finds all the outliers. Fig 6 shows successful detection of outliers on Dataset 2 (top) and 3(bottom) with parameters $r = 3, k = 3$ and $r = 3, k = 5$ respectively. Note that the bottom image of Fig 6 has 4 outliers and setting $k = 5$ picks outliers, thus even with overshooting the value of k does successfully recover the subspace underlying non-outlier points. This can also be seen in Fig 5. For $k = 3, r = 4$ we can see that two images out of four detected images are not outliers. We can still recover the exact linear subspace of non-outliers by overshooting parameter k.

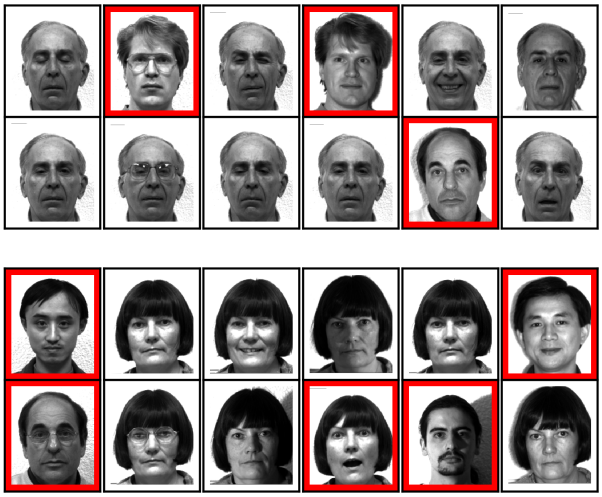

Fig. 6. Face Dataset 2 (top) and 3 (bottom). Successful run on face images with $r = 3$. We use $k = 3$ in the top experiment and $k = 5$ in the bottom one.

VII. Concluding Remarks

Outlier Robust PCA is an important problem in data analytics and machine learning. We take a combinatorial approach to this problem and pose Outlier Robust PCA as a search problem on a graph of the subsets of columns of the data matrix. We use the classic A^* algorithm to find the column subset of outliers of a given data matrix. Comparing our result to an exhaustive search, consider selecting k out of n columns. Exhaustive search requires evaluating $\binom{n}{k}$ subsets. For small k it is $O(n^k)$. More precisely (see [24]), it can be shown that for $k < \sqrt{n}$:

$$\frac{k!}{4} \leq \frac{\binom{n}{k}}{n^k} \leq k!$$

For example, with n=100,k=5 there are roughly $7.5{\cdot}10^7$ subsets to evaluate, and with n=500,k=5 there are roughly $2.5{\cdot}10^{11}$ subsets. As another example, if exhaustive search is attempted to select 4 columns from the *libras* dataset, we estimate the run time to be about one month. This is what we solve in about five minutes. The A^* heuristic search approach finds the optimal solution while evaluating significantly fewer subsets. The sub-optimal A^* works much faster than the optimal variant and still manages to get much better accuracy than the current state of the art.

References

[1] I. T. Jolliffe, *Principal Component Analysis*. Springer-Verlag, 1986.

[2] G. H. Golub and C. F. Van-Loan, *Matrix Computations*, 4th ed. Johns Hopkins University Press, 2013.

[3] T. Bouwmans, A. Sobral, S. Javed, S. K. Jung, and E. Zahzah, "Decomposition into low-rank plus additive matrices for background/foreground separation: A review for a comparative evaluation with a large-scale dataset," *Computer Science Review*, vol. 23, pp. 1–71, 2017.

[4] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM*, vol. 58, no. 3, pp. 11:1–11:37, May 2011.

[5] S. Oymak, A. Jalali, M. Fazel, Y. Eldar, and B. Hassibi, "Simultaneously structured models with applications to sparse and low-rank matrices." *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2886–2908, 2015.

[6] H. Xu, C. Caramanis, and S. Sanghavi, "Robust pca via outlier pursuit," in *Advances in Neural Information Processing Systems*, 2010, pp. 2496–2504.

[7] H. Zhang, Z. Lin, C. Zhang, and E. Y. Chang, "Exact recoverability of robust pca via outlier pursuit with tight recovery bounds," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 3143–3149.

[8] R. Maronna, D. Martin, and V. Yohai, *Robust Statistics: Theory and Methods*. Hoboken, NJ: Wiley, 2006.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge: Cambridge University Press, 2007.

[10] D. C. Hoaglin and R. E. Welsch, "The hat matrix in regression and anova," *The American Statistician*, vol. 32, no. 1, pp. 17–22, 1978.

[11] S. Chatterjee and A. S. Hadi, "Influential observations, high leverage points, and outliers in linear regression," *Statistical Science*, vol. 1, no. 3, pp. 379–393, 1986.

[12] P. E. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heurisitc determination of minimal cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[13] J. Pearl, *Heuristics : intelligent search strategies for computer*. Reading, Massachusetts: Addison-Wesley, 1984.

[14] S. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*. Pearson Education, 2010.

[15] G. H. Golub, "Some modified matrix eigenvalue problems," *SIAM Review*, vol. 15, no. 2, pp. 318–334, 1973.

[16] D. A. Bini and L. Robol, "Solving secular and polynomial equations: A multiprecision algorithm," *Journal of Computational and Applied Mathematics*, vol. 272, pp. 276–292, 2014.

[17] J. R. Bunch, C. P. Nielsen, and D. C. Sorensen, "Rank-one modification of the symmetric eigenproblem," *Numer. Math.*, vol. 31, pp. 31–48, 1978.

[18] A. Melman, "Analysis of third-order methods for secular equations," *Mathematics of Computation*, vol. 67, no. 221, pp. 271–286, Jan. 1998.

[19] C. F. Borges and W. B. Gragg, "A parallel divide and conquer algorithm for the generalized real symmetric definite tridiagonal eigenproblem," in *Numerical Linear Algebra nad Scientific Computing*, L. Reichel, A. Ruttan, and R. S. Varga, Eds., de Gruyter, Berlin, 1993, pp. 11–29.

[20] H. Arai, C. Maung, and H. Schweitzer, "Optimal column subset selection by A-Star search," in *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 2015, pp. 1079–1085.

[21] H. Arai, C. Maung, K. Xu, and H. Schweitzer, "Unsupervised feature selection by heuristic search with provable bounds on suboptimality," in *Proceedings of the 30th National Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, 2016, pp. 666–672.

[22] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[23] A. Georghiades, P. N. Belhumeur, and D. J. Kriegman, "Yale face database," 1997. [Online]. Available: http://cvc.yale.edu/projects/yalefaces/yalefa

[24] R. J. Lipton, "Bounds on binomial coefficents," http://rjlipton.wordpress.com/2014/01/15/bounds-on-binomial-coefficents, 2014.