Heuristic Search for Approximating One Matrix in Terms of Another Matrix

Guihong Wan and **Haim Schweitzer**
Department of Computer Science, The University of Texas at Dallas
Guihong.Wan@utdallas.edu, HSchweitzer@utdallas.edu

Abstract

We study the approximation of a target matrix in terms of several selected columns of another matrix, sometimes called "a dictionary". This approximation problem arises in various domains, such as signal processing, computer vision, and machine learning. An optimal column selection algorithm for the special case where the target matrix has only one column is known since the 1970's, but most previously proposed column selection algorithms for the general case are greedy. We propose the first nontrivial optimal algorithm for the general case, using a heuristic search setting similar to the classical A^* algorithm. We also propose practical sub-optimal algorithms in a setting similar to the classical Weighted A^* algorithm. Experimental results show that our sub-optimal algorithms compare favorably with the current state-of-the-art greedy algorithms. They also provide bounds on how close their solutions are to the optimal solution.

1 Introduction

Let $Y = (y_1 \dots y_N)$ be a "target" matrix of m rows and N columns. Let $X = (x_1 \dots x_n)$ be a "dictionary" matrix of m rows and n columns. We consider the problem of selecting $k \le n$ columns from X that can be used to linearly approximate Y as follows:

$$Y \approx SA, \tag{1}$$

where $S = (x_{s_1} \dots x_{s_k})$ is the selection matrix of size $m \times k$, and A is the coefficient matrix of size $k \times N$. We evaluate the quality of the selected columns in S with the Frobenius norm:

$$E(S) = \min_A \|Y - SA\|_F^2. \tag{2}$$

This formulation includes several well-known special cases. Supervised column subset selection corresponds to the case where $N = 1$. See, e.g. [Hastie *et al.*, 2009; Boutsidis *et al.*, 2013]. When $N > 1$, the problem is sometimes known as sparse multi-target prediction, or simultaneous sparse approximation (e.g., [Tropp, 2004; Maung and Schweitzer, 2015; Wan and Schweitzer, 2021b]). When $Y = X$, it is known as the unsupervised column subset selection, or unsupervised feature selection problem (e.g., [Golub and Van-Loan, 2013; Arai *et al.*, 2015]).

Optimal solutions, as well as approximations within a constant, are known to be NP-hard even for the $N = 1$ case [Natarajan, 1995; Davis *et al.*, 1997; Amaldi and Kann, 1998], and for the unsupervised case where $Y = X$ [Shitov, 2017]. This implies that optimal solutions cannot be efficient. Previous studies have developed optimal algorithms that are exponential in the worst case but improve on exhaustive search, and fast algorithms that are not optimal.

Optimal algorithms for the unsupervised case ($Y = X$) were studied in [Arai *et al.*, 2015; He *et al.*, 2019]. Optimal algorithms for the $N = 1$ case were studied in [Furnival and Wilson, 1974; Bertsimas *et al.*, 2016; Bertsimas *et al.*, 2020]. We are not aware of previous studies of optimal algorithms for the $N > 1$ case that we focus on in this paper.

The case in which the matrix to be approximated contains only one column ($N = 1$) has received a lot of attention. See, e.g. [Furnival and Wilson, 1974; Qian *et al.*, 2015; Bertsimas *et al.*, 2016; Hou *et al.*, 2017; Qin *et al.*, 2018; Huang *et al.*, 2020]. Applications include signal processing (e.g., [Selesnick, 2017; Qin *et al.*, 2018]) and supervised feature selection in linear regression (e.g., [Furnival and Wilson, 1974; Qian *et al.*, 2015; Hou *et al.*, 2017; Bertsimas *et al.*, 2016]). Convex relaxation approaches to feature selection replace some natural constraints (sometimes defined in terms of l_0 norm) with other convex constraints. An example is that the l_1 norm is used in the Lasso technique [Tibshirani, 1996]. Another recent variant is the Pareto technique [Qian *et al.*, 2015], where the authors treated the subset selection as a bi-objective optimization problem. The algorithm was proved to be optimal for data drawn from Exponential Decay distributions.

Heuristic search algorithms were recently applied to the unsupervised case where $Y = X$. See [Arai *et al.*, 2015; Arai *et al.*, 2016; He *et al.*, 2019]. They model the selection as a graph search problem, and apply the A^* or Weighted A^* to solve it. Our method is motivated by this approach. The algorithms developed in these studies can be viewed as special cases of our method.

In the general case, the matrix Y contains $N > 1$ columns. We observe that one cannot simply apply an algorithm for the N=1 case separately to each column of Y. The challenge of the general case is to find columns in X that

Algorithms	Time complexity	Memory complexity
SOMP [Tropp *et al.*, 2006]	$O(kmnN)$	$O(m(N+k))$
SSBR [Belmerhnia *et al.*, 2014]	$O(Tkmn N)$	$O(m(N+k))$
SOLS [Cotter *et al.*, 2005]	$O(kmnN)$	$O(m(n+N))$
CM [Civril and Magdon-Ismail, 2012]	$O(nN(m+k))$	$O(m(n+N))+nN$
ISOLS [Maung and Schweitzer, 2015]	$O(mnN)$	$O(km)+2n$
SPXY [Wan and Schweitzer, 2021b]	$O(km(n+N))$	$O(m(n+N))$
Greedy variant (this work)	$O((k^2+m)r_x n)$	$O(r_x n)$

Table 1: Complexity of various approximate algorithms. T is the number of iterations. $r_x \leq \min(m,n)$.

can simultaneously approximate all columns of Y. Previously proposed algorithms for this general case are generally greedy. They include [Maung and Schweitzer, 2015; Belmerhnia *et al.*, 2014; Civril and Magdon-Ismail, 2012; Tropp *et al.*, 2006]. Some of these algorithms are generated from the greedy algorithms for the $N = 1$ case. For example, the Simultaneous Orthogonal Matching Pursuit (SOMP) Algorithm [Tropp *et al.*, 2006] was generated from the Orthogonal Matching Pursuit (OMP) [Tropp, 2004]. Similarly, the Simultaneous Orthogonal Least Squares (SOLS) Algorithm [Cotter *et al.*, 2005] is a direct generalization of the Orthogonal Least Squares (OLS) Algorithm [Chen *et al.*, 1989]. An algorithm established in [Civril and Magdon-Ismail, 2012] improves the SOLS in term of runtime at the cost of increased memory. A recursive formulation in [Maung and Schweitzer, 2015] is used to improve the speed of the SOLS. A spectral pursuit algorithm presented in [Wan and Schweitzer, 2021b] is efficient when n and N are large. The running time and the memory requirements of these algorithms are summarized in Table 1. The complexity of our greedy variant is discussed in Section 4.

1.1 Our Approach

We propose a heuristic search approach for solving the general problem of selecting a subset of k columns from the dictionary matrix to simultaneously approximate the entire target matrix. Our heuristic functions are based on eigenvalues of related matrices. With proper selection of the heuristics, our algorithm can be tuned to give an optimal solution as well as approximate solutions. The optimal variant runs slow and works only for small k or small dictionary matrices. The suboptimal variants run much faster and produce solutions with guarantees on how close the solutions are to the optima. The main contributions are summarized below:

- We describe the first nontrivial algorithm guaranteed to produce an optimal solution for the general case $N \geq 1$.
- We describe suboptimal algorithms that are more accurate than the greedy variants. Their solutions come with bounds on how far the solutions are from the optimal solution.
- We describe a greedy algorithm that produces results of similar accuracy to the current state of the art with a superior running time. It also comes with a bound on how far the solution is from the optimum.

2 The Proposed Algorithms

Heuristic search algorithms on graphs, such as A^* [Hart *et al.*, 1968] and Weighted A^* [Pohl, 1970], are widely used for solving search problems that can be modeled as graph search.

Algorithm 1 The Search Algorithm.

Input:
Y: a target matrix of N columns.
X: a dictionary matrix of n columns.
k: the desired number of columns to be selected from X.
$f(\cdot)$: a heuristic function.
r_y (optional): the desired rank of Y. Default:$r_y = \min(m,N)$.
r_x (optional): the desired rank of X. Default:$r_x = \min(m,n)$.
Output: a subset S of k columns.
Data Structures: Two global lists of subsets: the fringe list F, and the closed list C.
Preprocessing: Dimensionality reduction.
Initialization: Put the empty subset into F.

```
while F is nonempty do
    Pick node n_i (associated with subset S_i of size k_i) with the smallest heuristic f_i from F. Ties are resolved in favor of the larger k_i.
    if S_i contains k columns then
        Stop and return S_i as the solution subset.
    else
        for each child n_j of n_i do
            if n_j is not in C then
                Compute f_j for n_j.
                Put n_j in C.
                Put n_j in F.
            end if
        end for
    end if
end while
```

The goal is to find a path from a root node to a goal node minimizing the cost associated with the path. The search is guided by functions associated with each node that are known as "Heuristics". With some heuristics, A^* is guaranteed to be optimal, but it may be very slow. Experimental results and theoretical analysis (e.g., [Pearl, 1984]) show that Weighted A^* runs faster than A^*. The algorithm we propose in this paper is similar to the standard (Weighted) A^*.

We describe the search algorithm in terms of a general heuristic function that will be defined later. It is shown in Algorithm 1. The same search procedure (with different heuristics) was also used in [Arai *et al.*, 2015; Arai *et al.*, 2016; He *et al.*, 2019; Wan and Schweitzer, 2021a].

In our case, the graph is constructed as follows. A node contains a subset of selected columns from X and the corresponding f value based on the approximation error of Y. The root node corresponds to the empty subset. A goal node corresponds to a subset of k columns. The children of a node are created by adding a new column into its parent subset. Without loss of generality, we do not distinguish between a node and a subset. The goal of the algorithm is to search a node of k columns on the graph, minimizing the heuristic function f. The fringe list F is a list of nodes that need to be further evaluated, and the closed list C is a list of visited nodes that need not to be added into F again. Employing the standard "best first" strategy the algorithm selects the node from F with the smallest f value to be expanded.

There are two differences between the search procedure in

Algorithm 1 and the typical (Weighted) A^* procedure. The first is in Line 9, where the node is immediately inserted into the closed list, and the second is in Line 10, where the classical A^* algorithm first checks if the node is already in the fringe list, and if so decides whether its value needs to be updated. The justification for these differences is that in our case all paths leading from the root node to a node are equivalent. Once a subset node is added into the fringe list other nodes with the same subset (obtained through a different path) will have the exact same heuristic values and therefore can be ignored. This implies that the fringe list is a subset of the closed list. (These observations show that the closed list can be implemented by a hash table, and the fringe list by a heap.)

3 Heuristic Functions

Recall that the error of approximating Y by a linear combination of k columns in S is given by (2). Let n_i be an arbitrary subset node. Let k_i<k be the number of columns selected at n_i, and let S_i be the matrix formed by these columns. Let $\overline{k}_i$=k−k_i be the number of columns that still need to be selected. We consider the error that can be obtained by completing S_i to size k. Let $\overline{S}_i$ of size $\overline{k}_i$ be such completion. Its error is given by:

$$e(\overline{S}_i) = E(S_i \cup \overline{S}_i) = \min_{A_i, \overline{A}_i} \|Y - S_i A_i - \overline{S}_i \overline{A}_i\|_F^2. \quad (3)$$

In the equation above, $E()$ is the error defined in (2), and $A_i, \overline{A}_i$ are arbitrary coefficient matrices. Thus, the smallest error of a solution subset that contains S_i would be:

$$d_i = \min_{\overline{S}_i} e(\overline{S}_i). \quad (4)$$

This shows that the best choice for the heuristic value f_i for a node n_i is $f_i = d_i$. Unfortunately, computing d_i is too expensive since it requires going over all the subsets $\overline{S}_i$ of size $\overline{k}_i$. However, we show that there is an effective computational approach to approximate d_i. We define two approximations l_i and u_i, and show that they bound d_i from below and above.

$$\begin{aligned} d_i &= \min_{\overline{S}_i, A_i, \overline{A}_i} \|Y - S_i A_i - \overline{S}_i \overline{A}_i\|_F^2, \\ u_i &= \min_{A_i} \|Y - S_i A_i\|_F^2 = \|Y - S_i A_i^*\|_F^2 = \|Y_i\|_F^2, \\ l_i &= \|Y_i - \overline{Y}_i\|_F^2, \end{aligned} \quad (5)$$

where $A_i^* = \arg\min_{A_i} \|Y - S_i A_i\|_F^2$, $Y_i = Y - S_i A_i^*$, $\overline{Y}_i$ is the best rank $\overline{k}_i$ approximation to Y_i.

Lemma 1. For any selection S_i of size $k_i \le k$:

$$l_i \le d_i \le u_i \quad (6)$$

with equality if $k_i = k$.

Proof:

- To prove the right hand side inequality, observe that for any S_i the expression for u_i is the same as d_i if the matrix $\overline{A}_i$ is taken to be identically 0. This ignores the contribution of additional columns that may reduce the error further.
- To prove the left hand side inequality:

$$\begin{aligned} l_i &= \|Y_i - \overline{Y}_i\|_F^2 \le \min_{\overline{S}_i, \overline{A}_i} \|Y_i - \overline{S}_i \overline{A}_i\|_F^2 \quad \text{(note1)} \\ &= \min_{\overline{S}_i, A_i, \overline{A}_i} \|Y - S_i A_i - \overline{S}_i \overline{A}_i\|_F^2 = d_i \end{aligned}$$

  The justification for the inequality in (note1) is that the rank of $\overline{S}_i \overline{A}_i$ is at most $\overline{k}_i$.
- It remains to show that the inequalities become equalities when $k_i = k$. Under this condition, we have: $\overline{k}_i = 0$, which implies that $\overline{Y}_i = 0$ and $\overline{S}_i = 0$. Therefore, l_i, d_i and u_i have same expression: $\|Y_i\|_F^2$. ■

3.1 Three Variants of the Search Algorithm

Clearly, the best choice for the heuristic function is $f_i = d_i$. However since it cannot be efficiently calculated, we consider the linear combination between l_i and u_i. Three options of the heuristic function are considered. The resulting variants are stated as in the following theorems.

Theorem 1. (the optimal variant) If $f_i = l_i$ then the algorithm is guaranteed to terminate with an optimal solution.

Theorem 2. (the greedy variant) If $f_i = u_i$ then the algorithm is greedy and terminates after expanding k nodes.

Theorem 3. (the weighted variant) If $f_i = l_i + \gamma u_i$, where $\gamma \ge 0$, and the algorithm terminates with the subset S^{**} that has an error of e^{**} then

$$e^{**} \le e^* + \gamma \|Y\|_F^2,$$

where e^* is the smallest possible error.

3.2 Lemmas

Our proofs of the theorems require the following results.

Lemma 2. If n_j is a child of n_i then $l_i \le l_j$.

Lemma 3. If n_j is a child of n_i then $u_j \le u_i$.

Lemma 4. Consider the choice $f_i = u_i$. Let n_i be the node picked at Line 2 of the algorithm. Then the following two properties hold:

a. For each child n_j of n_i, the selection size $|S_j|$ is larger than the selection size of all nodes currently in the fringe list.

b. The next node to be picked up will be a child of n_i.

Instead of proving Theorem 3 directly, we prove a stronger version with a tighter bound:

Theorem 4. (the tighter weighted variant) If $f_i = l_i + \gamma u_i$, where $\gamma \ge 0$, and the algorithm terminates at the node n_{**} with the subset S^{**} that has an error of e^{**}, then:

$$e^{**} \le e^* + \gamma(u_{\max} - l_{**})$$

where e^* is the smallest possible error, l_{**} is the value of the lower bound for n_{**}, and $u_{\max}$ be the largest value of the upper bound for the nodes remaining in the fringe after the goal node is reached.

To see that Theorem 4 implies Theorem 3, observe that $u_{\max} \le \|Y\|_F^2$ and $l_{**} \ge 0$.

Lemma 5. Suppose Theorem 4 is false. Then for any node n_z on the path from the root to an optimal goal node n_*, the following condition holds: $f_z < f_{**}$.

Similar theorems were proved in [He *et al.*, 2019] for the unsupervised case. However, the proof technique used in their study does not appear strong enough to prove these theorems for the supervised case. The tool we use to prove the lemmas is the interlacing property of eigenvalues [Golub and Van-Loan, 2013].

The proof of Theorem 1 follows as a corollary of Theorem 3 with $\gamma = 0$. Using Lemma 4, Theorem 2 can be proved. Theorem 3 follows from Theorem 4, and Theorem 4 can be proved by contradiction and using the results in Lemma 5. Detailed proofs are given in a supplementary material.

4 Efficient Heuristics Calculation

The following theorem summarizes the computational formulas for heuristic values.

Theorem 5. At the node n_i where the k_i columns in S_i have already been selected from X, additional $\overline{k}_i{=}k{-}k_i$ columns still need to be selected. Define: $B_i = Y_i Y_i^T$, where Y_i is defined in (5). Let $\lambda_1, \ldots, \lambda_m$ be the eigenvalues of B_i in non-increasing order. Then the values of l_i, u_i defined in (5) can be calculated by:

$$\begin{aligned} u_i &= \sum_{j=1}^{m} \lambda_j = \text{Trace}\{B_i\}, \\ l_i &= \sum_{j=\overline{k}_i+1}^{m} \lambda_j = \text{Trace}\{B_i\} - \sum_{j=1}^{\overline{k}_i} \lambda_j. \end{aligned} \tag{7}$$

The proof is given in a supplementary material. From the formulas in Theorem 5, it is clear that only the top $\overline{k}_i$ eigenvalues of B_i need to be calculated in addition to its trace. But they have to be calculated for each node, which is impractical. We proceed to show that there is a matrix related to B_i with a special structure that enables efficient computation of these eigenvalues.

Observe that we only need heuristics for the children of the picked node (parent) at Line 8 in the algorithm. At that point the parent node n_i is known. We show how to efficiently calculate the heuristics for the children by first performing an expensive eigendecomposition of the parent. However the expensive eigendecomposition of the parent can be reduced by computing eigendecomposition in the initial preprocessing step. In summary, we discuss three different types of eigendecompositions. The first one is the one in the preprocessing step, which only needs to be performed once. The second is for each parent, and the third is for the children, where only the eigenvalues are needed.

4.1 Initial Eigendecomposition

In the preprocessing step we compute the eigenvalues and the eigenvectors of the matrices $B_y = YY^T$ and $B_x = XX^T$. The eigenvalue decompositions give:

$$B_y = U_y D_y U_y^T, \quad B_x = U_x D_x U_x^T,$$

where U_y, U_x are the eigenvectors and D_y, D_x are diagonal matrices with the eigenvalues as the diagonal elements.

Let r_y be the rank of Y then $r_y \leq \min(m, N)$. Let r_x be the rank of X then $r_x \leq \min(m, n)$. As we show later we can ignore zero eigenvalues and their corresponding eigenvectors, and can replace X, Y with the following information:

$$\begin{aligned} &W_x = U_x^T X, \quad \text{where } W_x \text{ is of size } r_x \times n. \\ &D_y \quad \text{reduced to size } r_y \times r_y. \\ &P = D_y^{\frac{1}{2}} U_y^T U_x, \quad \text{where } P \text{ is of size } r_y \times r_x. \end{aligned} \tag{8}$$

With the advancement of randomized algorithms for matrix decompositions, this initial step can be performed efficiently. For example, using the algorithm described in [Halko *et al.*, 2011], the time complexity is $O(mr_yN{+}mr_xn)$; the memory complexity is $O(mr_y{+}mr_x{+}r_xn)$ for the preprocessing step.

4.2 Eigendecomposition for Parent Nodes

Instead of working with the matrix B_i as defined in Theorem 5, we use a related matrix H_i which has same eigenvalues as B_i. The special structure of H_i makes the calculations of these eigenvalues more efficient.

Lemma 6. Let Q_i be an orthonormal basis of the current selection S_i (selected from X) of size k_i. Let $\tilde{S}_i$ be the corresponding selection from $W_x = U_x^T X$, and $\tilde{Q}_i$ be the orthonormal basis of $\tilde{S}_i$, then: $Q_i = U_x \tilde{Q}_i$.

Proof: Straightforward.

Lemma 7. Let B_i be the matrix whose eigenvalues are used in (7) to calculate the heuristics. Let $\tilde{Q}_i$ be an orthonormal basis of $\tilde{S}_i$ of size k_i. Given D_y and P from (8), define the following $r_y \times r_y$ matrix:

$$H_i = D_y - Z_i Z_i^T = D_y - \sum_{j=1}^{k_i} z_j z_j^T, \tag{9}$$

where $Z_i = P\tilde{Q}_i$, $z_i = P\tilde{q}_i$, and $\tilde{q}_i$ is the ith column of $\tilde{Q}_i$. Then H_i and B_i have the same eigenvalues (and trace).

Lemma 7 shows that H_i can be computed from k_i times rank-one updates of the diagonal eigenvalue matrix, which enables specialized routines to compute its eigenpairs (e.g., [Bunch *et al.*, 1978]). The proof is given in a supplementary material.

4.3 Eigenvalues for Children Nodes

To compute the heuristics at Line 8 in Algorithm 1, eigenvalues for children nodes along with their traces are needed. It is the most expensive part of the algorithm. We show how to compute those values efficiently.

Let n_p be the node picked at Line 2 of the algorithm and let $\tilde{S}_p$ of size k_p be the corresponding selection. Let $\tilde{Q}_p$ be the orthonormal basis of $\tilde{S}_p$. Let H_p be the matrix computed according to (9). Suppose a child node n_c is created by adding a column w from W_x to $\tilde{S}_p$. From (9) the associated matrix H_c for the child can be computed by:

$$H_c = H_p - z_c z_c^T \tag{10}$$

where $z_c = P\tilde{q}_c$, $\tilde{q}_c = \bar{w}_c / \|\bar{w}_c\|$, and $\bar{w}_c = (I - \tilde{Q}_p \tilde{Q}_p^T) w$. The key observation here is that H_c is a rank-one modification of H_p. In this case the eigenvalues of the updated matrix can be computed efficiently in $O(kr_y)$ from the eigendecomposition of H_p. See, e.g. [Bunch *et al.*, 1978]. Since computing z_c takes $O(kr_x)$ this adds up to $O(k(r_x + r_y)n)$ for computing the heuristic values for all children of a parent node.

k	f=l	f=l+u	f=l+5u	f=u	Forward	Backward	Leaps	OMP	FoBa	POSS	ISOLS
eunite2001 X:367 × 16 Y:367 × 1											
5	**1.154e6**	**1.154e6**	**1.154e6**	1.169e6	1.169e6	1.161e6	**1.154e6**	1.509e6	1.329e6	1.226e6	1.169e6
10	**0.920e6**	**0.920e6**	0.931e6	1.035e6	1.035e6	**0.920e6**	**0.920e6**	1.043e6	1.043e6	1.013e6	1.035e6
spectf X:267 × 44 Y:267 × 1											
5	**38.64**	**38.64**	**38.64**	39.62	39.62	39.20	**38.64**	40.52	40.52	38.87	39.62
7	**37.73**	**37.73**	38.10	38.36	38.36	38.41	**37.73**	38.74	37.83	**37.73**	38.36
libras X:360 × 90 Y:360 × 1											
3	**5192.12**	**5192.12**	**5192.12**	**5192.12**	**5192.12**	5333.00	**5192.12**	5334.94	5334.94	5194.66	**5192.12**
5	**4723.07**	**4723.07**	4778.88	4796.08	4796.08	5086.07	**4723.07**	4953.25	4953.25	4777.82	4796.08
duke breast cancer X:44 × 7,129 Y:44 × 1											
2	**14.67**	**14.67**	14.94	14.94	error	error	error	14.94	14.94	16.07	14.94
5	-	**4.92**	5.14	5.14	error	error	error	5.36	5.36	5.14	5.14

Table 2: Accuracy comparison for the case $N = 1$. The minimum errors are highlighted. No results (-) is shown if the runtime is longer than 30 minutes. The "error" means that there is an error thrown out during the run of the algorithm. Our optimal variant ($f = l$) and Leaps are guaranteed to produce a best selection. Our weighted variant ($f = l + \gamma u$) are more accurate than other non-optimal algorithms.

4.4 Complexity

Suppose there are T parent nodes, then the overall time complexity is $O(mNr_y + mnr_x + T \cdot k(r_x + r_y)n)$; the memory complexity is $O(mr_y + mr_x + r_x n + T \cdot n)$. For the greedy variant, since $T = k$ and no fringe/closed lists are needed, the time complexity is $O(mr_y N + mr_x n + k^2(r_x + r_y)n)$; the memory complexity is $O(mr_y + mr_x + r_x n)$. Assuming $r_x \geq r_y,\ n \geq N$ and $n \geq m$, then the time complexity is $O((k^2 + m)r_x n)$; the memory complexity is $O(r_x n)$.

5 Bound on Sub-optimality

Both the greedy variant and the weighted variant are not guaranteed to produce an optimal solution. We proceed to show how to obtain bounds on how close their solutions are to the optimal. The technique we use was originally proposed by [Hansen and Zhou, 2007].

Consider a run of a non-optimal variant producing the non-optimal selection S^{**}. Then size$(S^{**}) = k$, and from Lemma 1 it follows that $l_{**} = u_{**} = E(S^{**})$. The value of l_{**} is related to the optimal value $l_* = u_* = E(S^*)$ by: $l_* \leq l_{**}$. Let b be a value satisfying: $l_{**} \leq l_* + b$, or, equivalently $b \geq l_{**} - l_*$. We refer to b as a bound, where a smaller b indicates a better bound, and in particular $b = 0$ implies an optimal solution. An important observation is that in heuristic search one can always compute such values. Let F be the fringe list after the algorithm terminates. Going over all the remaining nodes in the fringe list we can compute: $l_{\min} = \min_{n_i \in F} l_i$. From Lemma 2 it follows that $l_* \geq l_{min}$, so that we can take:

$$b = l_{**} - l_{\min}, \quad \text{where: } l_{\min} = \min_{n_i \in F} l_i. \tag{11}$$

The b is a nontrivial bound on $l_{**} - l_*$, and can be calculated. This gives a provable bound on sub-optimality.

6 Experimental Results

We describe experiments on various datasets that are publicly available. For the N=1 case we compare the proposed algorithm with the following methods: Leaps [Furnival and Wilson, 1974]; Forward [Hastie *et al.*, 2009]; Backward [Hastie *et al.*, 2009]; OMP [Mallat, 1999]; FoBa [Zhang, 2009]; POSS [Qian *et al.*, 2015]. For the general case ($N \geq 1$) we compare our algorithm with the following algorithms: SOMP [Tropp *et al.*, 2006]; SSBR [Belmerhnia *et al.*, 2014]; SOLS [Chen and Huo, 2006]; CM [Civril and Magdon-Ismail, 2012]; ISOLS [Maung and Schweitzer, 2015]. The results for SOLS, CM and ISOLS are same. (They are different in terms of runtime.) The results for ISOLS are shown. The implementations used for Leaps, Forward, and Backward are the functions in the R library "leaps". SOMP and SSBR are implemented in Python. Other implementations are publicly available. Experiments are conducted on iMac with Processor Intel Quad-Core i7 and Memory 32GB.

6.1 Comparison

As discussed in Section 3.1, with different choices of the heuristic function, our algorithm produces optimal, suboptimal and greedy results. We compare those results with the current state-of-the-art algorithms for the N=1 case and the general case where N>1.

Comparison for the N=1 case. The results for various datasets are shown in Table 2. As expected, our optimal variant (f=l) gives identical results to Leaps. Recall that our optimal variant also works when N>1. The errors for the greedy variant (f=u) are same as Forward and ISOLS. The errors for the weighted variant (f=l+γu) are between the optimal solution and the greedy solution. They are more accurate than other non-optimal algorithms.

Comparison for the general case. The results for various datasets are shown in Table 3. The first three datasets are split evenly and experimented without intercept. The remaining datasets are real multi-target datasets tested with intercept. (The constant vector $\mathbf{1}$ is added into the dictionary matrix X.)

Our optimal variant (f=l) is guaranteed to produce optimal solutions. The errors for the greedy variant (f=u) are same as the results of ISOLS. The errors for f=l+γu are between the optimal solution and the greedy solution.

Observe that the sub-optimality bounds are meaningful. In fact, in some cases the sub-optimality bound is 0 which implies that the solution is optimal even though the algorithm is

k	f=l	f=l+u		f=l+2u		f=l+10u		f=u		SOMP	SSBR	ISOLS
	error	error	bound:b	error	bound:b	error	bound:b	error	bound:b			
libras X: 360×45 Y: 360×46												
3	**6,010**	**6,010**	**0**	**6,010**	0.947	**6,010**	0.949	6,169	0.95	6,047	6,169	6,169
5	**5,587**	**5,587**	**0.941**	5,594	0.987	5,623	0.987	5,686	0.987	5,632	5,686	5,686
spectf X: 267×22 Y: 267×23												
5	**423,909**	428,524	**0.635**	433,697	0.639	433,697	0.639	433,697	0.639	431,650	433,697	433,697
10	**374,453**	377,282	0.842	377,282	0.842	377,282	0.842	377,282	0.842	384,156	377,313	377,282
duke breast cancer X: $44 \times 3,565$ Y: $44 \times 3,565$												
5	-	**62,040**	**0.071**	**62,040**	**0.071**	62,191	0.074	62,191	0.074	62,976	62,191	62,191
25	-	**18,040**	**0.159**	18,700	0.189	18,700	0.189	18,700	0.189	213,17	18,700	18,700
scm20d X: $8,966 \times 61$ Y: $8,966 \times 16$												
5	**5.727e9**	**5.727e9**	**0.727**	5.786e9	0.748	6.007e9	0.758	6.007e9	0.758	6.159e9	6.007e9	6.007e9
7	-	**5.205e9**	**0.882**	5.239e9	0.883	5.367e9	0.886	5.367e9	0.886	5.24e9	5.367e9	5.367e9
mediamill X: $43,907 \times 120$ Y: $43,907 \times 101$												
5	**116,292**	**116,292**	**0**	**116,292**	0.395	116,899	0.398	117,894	0.403	118,744	117,894	117,894
10	-	-	-	-	-	**112,975**	0.579	113,931	0.583	114,403	113,931	113,931
oes97 X: 334×263 Y: 334×16												
5	-	**2.120e9**	**0.597**	2.126e9	0.598	2.126e9	0.598	2.126e9	0.598	2.498e9	2.126e9	2.126e9
7	-	**1.704e9**	**0.756**	1.708e9	0.757	1.708e9	0.757	1.708e9	0.757	2.002e9	1.708e9	1.708e9

Table 3: Accuracy comparison for the $N > 1$ case. Our optimal variant ($f = l$) is guaranteed to produce a best selection. The suboptimal results come with bounds. The bounds are normalized by the solution errors.

	SOMP	SSBR	ISOLS	f=l+5u	f=l+u
duke breast cancer: $44 \times 7,129$; $X = Y$					
error / b	58,330	52,629	52,629	52,034 / 0.13	52,629 / 0.14
time (s)	4	30	42	1.9	1.8
Sift: dense $128 \times 1,000,000$; $X = Y$					
error / b	-	-	-	5.15e10 / 0.23	5.18e10 / 0.23
time (min)	-	-	-	17 = 2 + 15	17 = 2 + 15
Day1: sparse $20,000 \times 3,231,957$; $X = Y$; $r_x = r_y = 500$					
error / b	-	-	-	284,222 / 0.097	284,455 / 0.098
time (min)	-	-	-	24 = 17 + 7	24 = 17 + 7

Table 4: Runtime comparison on big datasets with k=20. The "-" indicates that the algorithm did not terminate after 50 minutes.

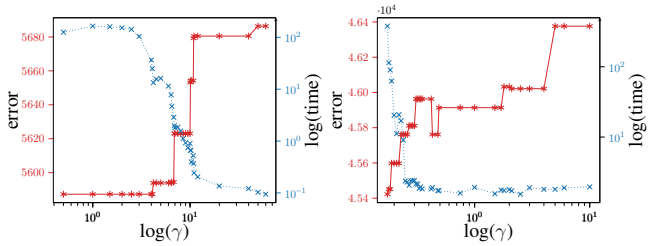

Figure 1: The effect of γ on accuracy and runtime for the general case. The left panel: libras dataset N=46, k=5. The right panel: duke breast cancer N=3,565, k=10.

not guaranteed to produce optimal solutions.

Comparison of the running time. To evaluate the running time we use very large datasets. Table 4 shows the results. Taking f=l+u with the Day1 dataset as an example, the total cost is 24 minutes: 17 minutes for initial processing plus 7 minutes for searching. Other algorithms are not practical for big datasets.

6.2 The Effect of the Parameter γ

We investigated the influence of the parameter γ on the accuracy and runtime. The results are shown in Figure 1. The red curve corresponds to the change of errors as the increase of γ. The blue curve is for the change of the runtime.Observe that as the increase of γ, the error goes up from an optimal solution to a greedy solution, while the runtime decreases.

7 Conclusion

We study a common setting where several columns of one matrix are selected to simultaneously approximate all the columns of another matrix. While many algorithms were developed for the case where the target matrix has a single column, we are only aware of approximate algorithms for the general case. We observe that the general case cannot be reduced to the one column case, as the challenge is to select columns that approximate the entire target matrix.

The algorithms that we develop are similar to the (Weighted) A^* algorithm. We show that with 0 assigned to a weight parameter, the algorithm is guaranteed to be optimal, but its running time may be very slow. Other nonzero choices give practical algorithms that beat the accuracy of the current state of the art algorithms.

In addition to producing a solution, our algorithms calculate bounds on how far the solutions are from the optimum. The quality of these bounds are data dependent. In some cases they provide no useful information, but in other cases it shows that the computed result is very close to the optimal solution. This is the case even for the greedy variant of our algorithm. It produces the same result as other known algorithms but gives the additional information of a bound.

While there is a significant similarity between our algorithm and the classical theory of the (Weighted) A^*, we are not aware of direct applications of A^* to the problem discussed here. In particular, our upper bound does not seem to have a parallel in the general theory.

References

[Amaldi and Kann, 1998] E. Amaldi and V. Kann. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209, 1998.

[Arai *et al.*, 2015] H. Arai, C. Maung, and H. Schweitzer. Optimal column subset selection by A-Star search. In *AAAI'15*, pages 1079–1085, 2015.

[Arai *et al.*, 2016] H. Arai, C. Maung, K. Xu, and H. Schweitzer. Unsupervised feature selection by heuristic search with provable bounds on suboptimality. In *AAAI'16*, pages 666–672, 2016.

[Belmerhnia *et al.*, 2014] Leila Belmerhnia, El-Hadi Djermoune, and David Brie. Greedy methods for simultaneous sparse approximation. In *2014 22nd European Signal Processing Conference (EUSIPCO)*, pages 1851–1855, 2014.

[Bertsimas *et al.*, 2016] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, 2016.

[Bertsimas *et al.*, 2020] Dimitris Bertsimas, Bart Van Parys, et al. Sparse high-dimensional regression: Exact scalable algorithms and phase transitions. *The Annals of Statistics*, 48(1), 2020.

[Boutsidis *et al.*, 2013] C. Boutsidis, P. Drineas, and M. Magdon-Ismail. Near-optimal coresets for least-squares regression. *IEEE Transactions on Information Theory*, 59(10):6880 – 6892, 2013.

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

[Chen and Huo, 2006] J. Chen and X. Huo. Theoretical results of sparse representations of multiple measurement vectors. *IEEE Transactions on Signal processing*, 54(12):4634–4643, 2006.

[Chen *et al.*, 1989] Sheng Chen, Stephen A Billings, and Wan Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of control*, 50(5):1873–1896, 1989.

[Civril and Magdon-Ismail, 2012] A. Civril and M. Magdon-Ismail. Column subset selection via sparse approximation of SVD. *Theoretical Computer Science*, 421:1–14, March 2012.

[Cotter *et al.*, 2005] S. Cotter, B. Rao, K. Engen, and K. Kreutz-Delgado. Sparse solutions to linear inverse problems with multiple measurement vectors. *ASP*, 53(7):2477–2488, 2005.

[Davis *et al.*, 1997] Geoff Davis, Stephane Mallat, and Marco Avellaneda. Adaptive greedy approximations. *Constructive approximation*, 13(1):57–98, 1997.

[Furnival and Wilson, 1974] G. M. Furnival and R. W. Wilson. Regressions by leaps and bounds. *Technometrics*, 16(4):499–511, 1974.

[Golub and Van-Loan, 2013] G. H. Golub and C. F. Van-Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, fourth edition, 2013.

[Halko *et al.*, 2011] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[Hansen and Zhou, 2007] Eric A Hansen and Rong Zhou. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28:267–297, 2007.

[Hart *et al.*, 1968] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Hastie *et al.*, 2009] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, section 3.3.1. Springer, 2009.

[He *et al.*, 2019] B. He, S. Shah, C. Maung, G. Arnold, G. Wan, and H. Schweitzer. Heuristic search algorithm for dimensionality reduction optimally combining feature selection and feature extraction. In *AAAI'19*, pages 2280–2287, California, 2019.

[Hou *et al.*, 2017] C. Hou, Y. Jiao, F. Nie, T. Luo, and Z. Zhou. 2d feature selection by sparse matrix regression. *IEEE Transactions on Image Processing*, 26(9):4255–4268, 2017.

[Huang *et al.*, 2020] S. Huang, Y. Peng, C. Chang, K. Cheng, S. Huang, and B. Chen. Restoration of images with high-density impulsive noise based on sparse approximation and ant-colony optimization. *IEEE Access*, 8, 2020.

[Mallat, 1999] S. Mallat. *A wavelet Tour of Signal Processing*. Academic Press, 1999.

[Maung and Schweitzer, 2015] C. Maung and H. Schweitzer. Improved greedy algorithms for sparse approximation of a matrix in terms of another matrix. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):769–780, March 2015.

[Natarajan, 1995] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal of Computing*, 25(2):227–234, 1995.

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

[Pohl, 1970] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

[Qian *et al.*, 2015] C. Qian, Y. Yu, and Z. Zhou. Subset selection by pareto optimization. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NIPS'15*, pages 1774–1782. Curran Associates, Inc., 2015.

[Qin *et al.*, 2018] Zhijin Qin, Jiancun Fan, Yuanwei Liu, Yue Gao, and Geoffrey Ye Li. Sparse representation for wireless communications: A compressive sensing approach. *IEEE Signal Processing Magazine*, 35(3):40–58, 2018.

[Selesnick, 2017] I. Selesnick. Sparse regularization via convex analysis. *IEEE Transactions on Signal Processing*, 65(17), 2017.

[Shitov, 2017] Y. Shitov. Column subset selection is np-complete, January 2017. arXiv e-print (arXiv:1701.02764[math.CO]).

[Tibshirani, 1996] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B.*, 58(1):267–288, 1996.

[Tropp *et al.*, 2006] J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part I: Greedy pursuit. *Signal Processing*, 86(3):572–588, 2006.

[Tropp, 2004] J. A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[Wan and Schweitzer, 2021a] G. Wan and H. Schweitzer. Accelerated combinatorial search for outlier detection with provable bound on sub- optimality. In *AAAI'21*, 2021.

[Wan and Schweitzer, 2021b] Guihong Wan and Haim Schweitzer. A fast algorithm for simultaneous sparse approximation. In *25th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'21)*, pages 42–54. Springer, 2021.

[Zhang, 2009] T. Zhang. Adaptive forward-backward greedy algorithm for sparse learning with linear models. In *NIPS'09*, pages 1921–1928, 2009.