Optimal Column Subset Selection by A-Star Search

Hiromasa Arai
The University of Texas at Dallas
800 W Campbell Road
Richardson, Texas 75080
hxa112430@utdallas.edu

Crystal Maung
The University of Texas at Dallas
800 W Campbell Road
Richardson, Texas 75080
Crystal.Maung@gmail.com

Haim Schweitzer
The University of Texas at Dallas
800 W Campbell Road
Richardson, Texas 75080
hschweitzer@utdallas.edu

Abstract

Approximating a matrix by a small subset of its columns is a known problem in numerical linear algebra. Algorithms that address this problem have been used in areas which include, among others, sparse approximation, unsupervised feature selection, data mining, and knowledge representation. Such algorithms were investigated since the 1960's, with recent results that use randomization. The problem is believed to be NP-Hard, and to the best of our knowledge there are no previously published algorithms aimed at computing optimal solutions. We show how to model the problem as a graph search, and propose a heuristic based on eigenvalues of related matrices. Applying the A* search strategy with this heuristic is guaranteed to find the optimal solution. Experimental results on common datasets show that the proposed algorithm can effectively select columns from moderate size matrices, typically improving by orders of magnitude the run time of exhaustive search.

1 Introduction

Let Y be an $m \times n$ data matrix. Consider the problem of linearly estimating Y from k of its columns:

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

The matrix $S = (y_{s_1}, \ldots, y_{s_k})$ is formed by the selected columns, and A is the coefficients matrix. Finding S in (1) is known as the "*Column Subset Selection Problem*" (CSSP). In Frobenius norm the approximation error is:

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

Some previous studies of the CSSP with the Frobenius norm include (Cotter et al. 2005; Frieze, Kannan, and Vempala 2004; Dasgupta et al. 2007; Deshpande et al. 2006; Deshpande and Rademacher 2010; Boutsidis, Mahoney, and Drineas 2009; Guruswami and Sinop 2012; Çivril and Magdon-Ismail 2012). Obtaining approximations in other norms, such as the spectral norm or the l_1 norm is considered to be harder. See, e.g. (Gu and Eisenstat 1996; Tropp 2004). Column subset selection gives a sparse approximation to the data matrix, and has found applications in many areas. These include the computation of stable and rank revealing QR factorizations (e.g. (Golub and Van-Loan 1996; Gu and Eisenstat 1996; Boutsidis, Mahoney, and Drineas 2009)), unsupervised feature selection (e.g. (Drineas, Lewis, and Paschou 2010; Maung and Schweitzer 2013), and data mining and knowledge representation (e.g. (Kumar, Mohri, and Talwalkar 2012; Drineas, Lewis, and Paschou 2010)).

Recent studies suggest that the CSSP is most likely NP-hard (Çivril and Magdon-Ismail 2009), and that the optimal solution is closely related to the dominant eigenvectors of YY^T. Let S_k^* denote the best selection of k columns for minimizing the error in (2), and let E_k^* denote the error of the best approximation of Y in terms of a rank-k matrix. Then the following inequalities hold:

$$E_k^* \le E(S_k^*) \le (k+1)E_k^* \tag{3}$$

Calculating E_k^* is easy, since the best rank-k matrix has the k dominant eigenvectors of YY^T as its columns. In particular:

$$E_k^* = \sum_{t=k+1}^{m} \lambda_t(YY^T) = \mathsf{Trace}(YY^T) - \sum_{t=1}^{k} \lambda_t(YY^T)$$

where $\lambda_t(YY^T)$ is the t largest eigenvalue of YY^T. This fact, and the left-hand side inequality in (3) follow from the Courant Fischer theorem (Golub and Van-Loan 1996). The right-hand side inequality is a recent result (Deshpande and Rademacher 2010) and holds only in the Frobenius norm.

Our main result is an A^* search algorithm that uses a generalized form of the inequalities in (3) to compute heuristic estimates. The search algorithm operates on a directed graph, where nodes correspond to column subsets. There is a directed edge from node n_1 to node n_2 if n_2 can be created by adding a single column to n_1. We generalize the inequalities in (3) to the case where some columns have already been selected, so that the inequalities can be applied in intermediate nodes. The left-hand side inequality enables computing a non-decreasing lower bound on the error of the optimal selection, which we use as the heuristic. The right-hand side is used to prune nodes that cannot lead to an optimal solution.

A direct implementation of the above idea requires a huge number of eigenvalue calculations, which makes the algorithm impractical. We are able to exploit a special structure of the relevant matrices that enables a fast calculation

of these eigenvalues. This gives an algorithm that can effectively select a small number of columns even from very large matrices. Running on a standard desktop our algorithm can select several columns from very large matrices taken from standard machine learning depositories. For example, selecting the best 4 columns from the 29,261 columns of the techtc01 dataset (Gabrilovich and Markovitch 2004) took about one minute.

To put things in perspective, current state-of-the-art algorithms for the CSSP, e.g. (Boutsidis, Mahoney, and Drineas 2009), take significantly less time to select hundreds of features from very large datasets. However, the selection is typically non-optimal. (In fact, with non-optimal algorithms one has no way of telling whether or not the result is optimal.) There is a well known optimal algorithm, called *Leaps and Bounds* (Furnival and Wilson 1974), for the related problem of selecting columns for approximating a single vector. That algorithm is based on an entirely different approach, and can handle matrices of no more than 30-40 columns (Hastie, Tibshirani, and Friedman 2009). By contrast, our algorithm which searches the same subsets but for a different goal, can handle much larger matrices, although it can effectively select only a small number of columns.

Attempting to solve the CSSP by exhaustive search requires evaluating $\binom{n}{k}$ subsets. 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 techtc01 dataset, we estimate the run time to be 30 million years. Our algorithm computes the solution in about a minute.

Statement of our main result

We describe a column subset selection algorithm that is optimal in the Frobenius norm. Our algorithm is typically much faster than exhaustive search (and, of course, has the same accuracy). It is more accurate than the current state-of-the-art algorithms, although it runs significantly slower. The main idea is to use spectral lower bounds on the optimal result as heuristics for A^* search, and spectral upper bounds on the optimal result for pruning.

2 Column subset selection by A^*

The search algorithm operates on a directed graph, where nodes correspond to column subsets. There is a directed edge from n_1 to n_2 if n_2 can be created by adding a single column to n_1. An example of the search graph with n=4,k=2 is shown in Figure 1. Observe that the graph has no directed cycles. We proceed to define the values for d, f, g, h which in our case are similar, but not identical, to the standard treatment of A^* in the literature (e.g (Russell and Norvig 2010)). Suppose k_i columns are selected at node n_i. Define:

$d \triangleq$ Smallest error of estimating Y with k columns.

$g_i \triangleq$ Error of estimating Y using the k_i columns in n_i.

$f_i \triangleq$ Smallest error of estimating Y using the k_i columns in n_i and additional $k - k_i$ "best possible" vectors.

$h_i \triangleq g_i - f_i$.

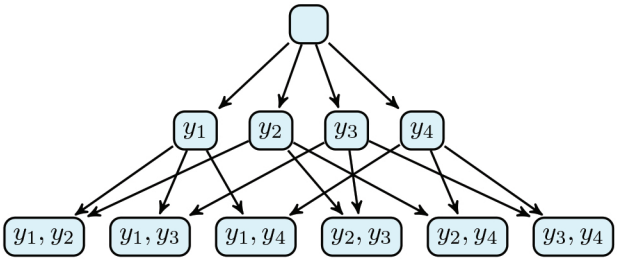

Figure 1: the subsets graph for n=4, k=2

0. Put the root node into F.
1. While F is nonempty and no solution found:
  1.1 Pick n_i with the smallest f_i from F.
  1.2 If n_i has k columns return it as the solution.
  1.3 Otherwise:
    1.3.1 Add n_i to C.
    1.3.2 Examine all children n_j of n_i.
      1.3.2.1 If n_j is in C or F do nothing.
      1.3.2.2 Otherwise if $f_j > P$ put n_j in C.
      1.3.2.3 Otherwise put n_j in F.

Figure 2: the A^* algorithm for optimal subset selection

The following inequalities clearly holds for all n_i: $f_i \leq g_i$, $d \leq g_i$. Also, if n_i is on the path to the optimal solution then $f_i \leq d$. Observe the following differences between the "textbook" definition of f, g, h and ours:

1. The "textbook" definition has g_i as the distance of n_i from the stating point so that it is monotonically increasing along any path. From our definition it is clear that g_i is monotonically decreasing along any path.
2. The relationship $f_i = g_i - h_i$ is different from the "textbook" relationship of $f_i = g_i + h_i$.
3. The values f_i, g_i, h_i depend only on the node n_i and not on the path traversed to reach it.

The A^* algorithm is shown in Figure 2. It keeps a fringe list F of nodes that need to be examined, and a list C of closed nodes, containing nodes that need not be visited again. It also uses a pruning value P, known to be an upper bound on the error of the optimal solution.

Theorem 1: Figure 2 algorithm finds the optimal solution.
Proof: The proof is almost identical to the "textbook" proof for the optimality of A^* on search trees. As before let k_i be the selection size at node n_i.

Claim 1. f_i is monotonically increasing along any path. Proof: if n_j is a child of n_i then f_j is computed from the k_i columns of n_i, one more column, and k-k_i-1 "best possible" vectors. This cannot be better than f_i which is computed from the same k_i vectors and k-k_i "best possible" vectors.

Claim 2. If k_i=k then g_i=f_i. (Immediate from definition.)

Claim 3. Let n_* be an optimal solution, and let n_j be a non optimal solution. Then any node n_b on the path from the root to n_* satisfies: $f_b < f_j$.

Proof: Since n_j is not optimal we have: $g_j > g_*$ so that: $f_j = g_j > g_* = f_* \geq f_b$. The equalities follow from Claim 2, and the right-hand inequality follows from Claim 1.

Claim 4. n_* is selected from the fringe before n_j.

Proof: Suppose the claim is false. Let n_b be a node on the path to n_* that is in the fringe when n_j is selected. But from Claim 3 n_b should be selected before n_j, which leads to a contradiction. This completes the proof of Theorem 1.

3 Calculating heuristic values

In this section we derive formulas for calculating the values of d, f_i, g_i, h_i, the heuristic values of node n_i, that were defined in Section 2.

Recall that our goal is to approximate the matrix Y of size $m \times n$ by a linear combination of k of its columns. The best solution has the following error:

$$d \triangleq E(S_k^*) = \min_{S,A} \|Y - SA\|_F^2$$

where S consists of k columns from Y. Consider the node n_i. Let k_i 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, and let $\overline{S}_i$ denote the matrix formed by these columns. At node n_i our goal is to select the best $\overline{S}_i$ in the sense of minimizing the following error:

$$E_i(\overline{S}_i) \triangleq \min_{A_i, \overline{A}_i} \|Y - S_i A_i - \overline{S}_i \overline{A}_i\|_F^2$$

We proceed to show that this can be viewed as a standard subset selection from a reduced matrix that we call Y_i.

Theorem 2: Let Q_i be an orthonormal basis to S_i. Define:

$$Y_i \triangleq Y - Q_i Q_i^T Y = (I - Q_i Q_i^T) Y \tag{4}$$

Recall that $\overline{S}_i$ is the selection of additional $\overline{k}_i$ columns, and let $\overline{Q}_i$ be an orthonormal basis to $\overline{S}_i$. Let $\tilde{S}_i$ be the selection of the columns in Y_i at the same locations as the $\overline{S}_i$ columns in Y: $\tilde{S}_i \triangleq (I - Q_i Q_i^T)\overline{S}_i$. Let $\tilde{E}$ be the error of estimating Y_i from $\tilde{S}_i$. Then $E_i(\overline{S}_i) = \tilde{E}$.

Proof: Observe that $E_i(\overline{S}_i)$ can be written as:

$$E_i(\overline{S}_i) = \min_{W_i, \overline{W}_i} \|Y - Q_i W_i - \overline{Q}_i \overline{W}_i\|_F^2 \tag{5}$$

where the matrix $(Q_i, \overline{Q}_i)$ is an orthonormal basis for $(S_i, \overline{S}_i)$, and $\overline{Q}_i$ is orthogonal to Q_i. Solving for W_i that minimizes (5) gives: $W_i = Q_i^T Y$. Substituting back in (5):

$$E_i(\overline{S}_i) = \min_{\overline{W}_i} \|Y_i - \overline{Q}_i \overline{W}_i\|_F^2$$

where $\overline{Q}_i$ is orthogonal to Q_i, and spans $(I - Q_i Q_i^T)S$.

By definition $\tilde{E} \triangleq \min_{\tilde{A}_i} \|Y_i - \tilde{S}_i \tilde{A}_i\| = \min_{\tilde{W}_i} \|Y_i - \tilde{Q}_i \tilde{W}_i\|$

where $\tilde{Q}_i$ is an orthonormal basis to $\tilde{S}_i = (I - Q_i Q_i^T)\overline{S}_i$. Therefore we can always take $\overline{Q}_i = \tilde{Q}_i$ and the expressions for $\tilde{E}$ and $E_i(\overline{S}_i)$ become identical. This completes the proof of Theorem 2.

Theorem 2 allows us to generalize the inequalities in (3):

Corollary: At node n_i let $\overline{S}_i^*$ denote the best selection of $\overline{k}_i$ columns after the k_i columns of S_i have already been selected, and let $E^*_{\overline{k}_i}$ denote the error of the best approximation of Y_i in terms of a rank-$\overline{k}_i$ matrix. Then:

$$E^*_{\overline{k}_i} \leq E_i(\overline{S}_i^*) \leq (\overline{k}_i + 1) E^*_{\overline{k}_i} \tag{6}$$

$$E^*_{\overline{k}_i} = \sum_{t=\overline{k}_i+1}^{m} \lambda_t(Y_i Y_i^T) = \mathsf{Trace}(Y_i Y_i^T) - \sum_{t=1}^{\overline{k}_i} \lambda_t(Y_i Y_i^T)$$

Proof: Apply the inequalities in 3 to Y_i.

The corollary shows that $f_i = E^*_{\overline{k}_i}$ is admissible. The following theorem summarizes the computational formulas.

Theorem 3: Searching for k columns, at node n_i where k_i columns of S_i have already been selected, set $\overline{k}_i = k - k_i$. Let Q_i to be an orthonormal basis of S_i and define: $B_i = Y_i Y_i^T$, where Y_i is defined in (4). Let $\lambda_1 \geq \ldots \geq \lambda_m$ be the eigenvalues of B_i. Then the values of f_i, g_i, h_i defined in Section 2 can be calculated by:

$$f_i = \sum_{t=\overline{k}_i+1}^{m} \lambda_t, \qquad h_i = \sum_{t=1}^{\overline{k}_i} \lambda_t,$$

$$g_i = \sum_{t=1}^{m} \lambda_t = \mathsf{Trace}(B_i) = f_i + h_i$$

From the formulas in Theorem 3 it is clear that only the top $\overline{k}_i$ eigenvalues of B_i need to be calculated in addition to its trace. However, this has to be calculated for each node, which is not practical. In the next section we show that there is a matrix related to B_i with a special structure that enables fast calculations of these eigenvalues.

4 Efficient calculation of heuristic values

In this section we discuss fast calculations of the eigenvalues that are needed to calculate the heuristics. We observe that the heuristics are needed only after line 1.3.2.2 of the algorithm in Figure 2. At that point the parent of the node is known. We show how to calculate the children eigenvalues efficiently by performing a more expensive eigenvalue decomposition for the parent, at line 1.3.1 of the algorithm. Furthermore, these more expensive calculations can be reduced by computing an expensive eigenvalue decomposition once, at line 0. To summarize, we discuss three different types of eigenvalue decompositions. The first is calculated once, at the root, the second is calculated for each selected node, and the third is calculated for each child.

4.1 Eigenvalue decomposition at the root

At the root we compute the eigenvalues and the eigenvectors of the matrix $B = YY^T$. The eigenvectors factorization gives:

$$B = V D V^T$$

Here V has orthonormal columns (the eigenvectors), and D is diagonal. We note that V and D can also be calculated

from the SVD of Y, and there are efficient algorithms for computing it. The one we have used is described in (Halko, Martinsson, and Tropp 2011). Let r be the numeric rank of Y then $r \leq \min\{m, n\}$. It is enough to keep V as an $m \times r$ matrix, and retain only the r nonzero eigenvalues.

4.2 Special structure

Instead of working with the matrix B_i of Theorem 3 we describe a related matrix which makes the calculations easier.

Lemma: Define the following $r \times r$ matrix:

$$H_i = D - Z_i Z_i^T = D - \sum_{j=1}^{r} z_j z_j^T \tag{7}$$

where $Z_i = D^{1/2} V^T Q_i$ and $z_j = D^{1/2} V^T q_j$. Then the matrices B_i and H_i have the same eigenvalues (and traces).

Proof: Using the eigenvalue decomposition of B we have:

$$B_i = Y_i Y_i^T = (I - Q_i Q_i^T) B (I - Q_i Q_i^T) = G_i D G_i^T$$

where $G_i = (I - Q_i Q_i^T) V$. As we show later H_i can be written as: $H_i = D^{1/2} G_i^T G_i D^{1/2}$. From this it follows that both H_i and B_i can be viewed as the product of the same two matrices (in different order) which implies that they have the same eigenvalues. To complete the proof it remains to show that the two expressions for H_i are identical. Direct calculation gives:

$$\begin{aligned} G_i^T G_i &= V^T (I - Q_i Q_i^T) V = I - V^T Q_i Q_i^T V \\ D^{1/2} G_i^T G_i D^{1/2} &= D - D^{1/2} V^T Q_i Q_i^T V D^{1/2} \\ &= D - Z_i Z_i^T \end{aligned}$$

This completes the proof of the lemma. The special structure of the matrix H_i, which can be expressed as r rank-1 updates to a diagonal matrix, allows for specialized routines for computing its eigenvectors and eigenvalues. See, e.g (Bini and Robol 2014; Melman 1998; Golub 1973).

4.3 Eigenvalue decomposition for parent nodes

This eigenvalue and eigenvector calculation is performed at line 1.3.1 of the algorithm. As explained later, we need both the eigenvectors and the eigenvalues of the matrix H_i defined in (7). Any of the methods that exploit the special structure can be used.

4.4 Computing eigenvalues for children nodes

The calculation of the heuristic at line 1.3.2.2 of the algorithm is the most time consuming part of the algorithm. We show how to compute these values efficiently from the eigenvalue decomposition of the parent.

Theorem 4: Let n_p be a parent node encountered at line 1.3.1 of the algorithm with selection of size k_p. Let Q_p be an orthonormal matrix that spans these k_p vectors. Let H_p be the matrix computed according to (7). Let Trace_p be the trace of H_p. Suppose a child node n_y is created by adding Column y to the selection of n_p. Then the following procedure computes the heuristic values at n_y without explicitly computing the associated matrix H_y of Equation (7).

1. $\tilde{q}_y = Q_p Q_p^T y, \qquad q_y = \frac{\tilde{q}_y}{|\tilde{q}_y|}$.

2. $z_y = D^{1/2} V^T q_y$.

3. $g_y = \mathsf{Trace}(H_y) = \mathsf{Trace}_p - |z_y|^2$.

4. The top $k - k_p - 1$ eigenvalues of H_y are computed using a specialized method.

5 Compute h_y as the sum of the eigenvalues computed at 4.

6 $f_y = g_y - h_y$.

Proof: From (7) it follows that:

$$H_y = H_p - z_y z_y^T, \quad z_y = D^{1/2} V^T q_y \tag{8}$$

The formula in 3. follows from the linearity of the trace, applied to (8). This gives: $\mathsf{Trace}(H_y) = \mathsf{Trace}(H_p) - \mathsf{Trace}(z_y z_y^T)$, and clearly $\mathsf{Trace}(z_y z_y^T) = |z_y|^2$. The computation of the top eigenvalues in 4. exploits the special structure shown in (8). Our particular implementation uses Gragg's method (Melman 1998). This is an iterative procedure with cubic convergence speed.

5 Pruning

We wish to point out the difference between the left-hand and the right-hand side inequalities in (6) (and in (3)). The left-hand side is a statement about *any* subsets $\overline{S}_i$. It must satisfy: $E^*_{\overline{k}_i} \leq E_i(\overline{S}_i)$. By contrast, the right-hand side inequality implies existence. There exists a subset $\overline{S}_i$ which satisfies $E_i(\overline{S}_i) \leq (\overline{k}_i + 1) E^*_{\overline{k}_i}$.

Since our algorithm is guaranteed to find the optimal solution we can use the right-hand side inequality for pruning. Observe that the inequality can be written as:

$$d \leq E_i(\overline{S}^*_i) \leq (\overline{k}_i + 1) f_i$$

where d is the error of the optimal solution. This holds for all nodes n_i, and we can use it as follows. Set

$$P = \min_i (\overline{k}_i + 1) f_i$$

where at any time i ranges over all the previously computed nodes. Such P must satisfy $d \leq P$, and any node n_j with $f_j > P$ can be pruned.

Observe that this pruning process cannot reduce the number of evaluated nodes. The algorithm evaluates a node n_i with the smallest value of f_i in the fringe list. If such node could have been pruned with the condition $f_i > P$, all the nodes in the fringe can also be pruned, and the algorithm cannot return any solution. However, the pruning can reduce the size of the fringe. This reduces the memory requirement. Depending on the data structure used for implementing the fringe, The pruning can also produce some savings in run time, since the search for a node (Step 1.3.2.1 of the algorithm) is faster when the fringe is smaller.

6 Experimental results

6.1 First experiment

We have tested the proposed algorithm on standard machine learning datasets. These were obtained from the UCI Repository, except for the techtc01 that can be obtained as explained in (Gabrilovich and Markovitch 2004). The results are shown in Figure 3.

k	num of children nodes	num of parent nodes	num parent / $\binom{n}{k}$	runtime	exhaustive / runtime	error TS/A*	error GKS/A*	memory without P / with P
wdbc dataset ($m = 569$, $n = 31$)								
1	31	2	$6.5{\cdot}10^{-2}$	0.001	3.333	1	1	31.00
5	1037	40	$2.4{\cdot}10^{-4}$	0.002	4492.332	1	1	20.81
10	25386	1168	$2.6{\cdot}10^{-5}$	0.036	$1.0\cdot10^{5}$	1	1	4.11
15	556991	32765	$1.1{\cdot}10^{-4}$	5.762	4374.933	1	1	1.53
16	1048470	65530	$2.2{\cdot}10^{-4}$	15.876	1587.679	1	1	2.29
sat dataset ($m = 2000$, $n = 37$)								
1	37	2	$5.4{\cdot}10^{-2}$	0.001	3.000	1.039	1.072	9.25
2	532	20	$3.0{\cdot}10^{-2}$	0.002	42.167	1.068	1	12.85
3	2676	108	$1.4{\cdot}10^{-2}$	0.009	113.171	1.140	1.027	8.43
4	17718	872	$1.3{\cdot}10^{-2}$	0.066	167.055	1.137	1.158	9.76
5	37464	1928	$4.4{\cdot}10^{-3}$	0.158	174.104	1.153	1.244	3.96
6	297806	19003	$8.2{\cdot}10^{-3}$	1.678	87.604	1.137	1.358	4.00
7	1347579	105556	$1.0{\cdot}10^{-2}$	18.850	34.542	1.062	1.114	3.47
SPECTF dataset ($m = 187$, $n = 45$)								
1	45	2	$4.4{\cdot}10^{-2}$	0.001	4.500	1	1.070	11.25
2	1035	47	$4.7{\cdot}10^{-2}$	0.001	17.667	1.029	1.028	21.09
3	11272	499	$3.5{\cdot}10^{-2}$	0.008	35.310	1.066	1.055	11.93
4	100848	5794	$3.9{\cdot}10^{-2}$	0.094	36.603	1.059	1.087	10.32
5	619332	40535	$3.3{\cdot}10^{-2}$	0.888	37.352	1.052	1.059	14.91
6	2298695	187582	$2.3{\cdot}10^{-2}$	20.444	20.717	1.092	1.039	8.44
movement_libras dataset ($m = 360$, $n = 91$)								
1	91	2	$2.2{\cdot}10^{-2}$	0.001	2.250	1	1	91.00
2	445	6	$1.5{\cdot}10^{-3}$	0.004	39.133	1	1.019	4.55
3	8625	188	$1.5{\cdot}10^{-3}$	0.016	379.283	1.002	1.015	25.50
4	182366	3977	$1.5{\cdot}10^{-3}$	0.351	195.656	1.005	1.042	24.18
5	1345415	23963	$5.2{\cdot}10^{-4}$	2.616	457.191	1.034	1.153	10.80
madelon dataset ($m = 2000$, $n = 500$)								
1	500	2	$4.0{\cdot}10^{-3}$	0.011	40.119	1.401	2.798	166.67
2	12675	27	$2.2{\cdot}10^{-4}$	0.508	215.373	1.976	6.247	24.80
3	71309	148	$7.1{\cdot}10^{-6}$	2.950	6150.857	1.441	3.399	11.09
4	234635	485	$1.9{\cdot}10^{-7}$	9.802	$2.3\cdot10^{5}$	1.574	1.308	11.21
5	286725	588	$2.3{\cdot}10^{-9}$	12.351	$1.8\cdot10^{7}$	1.181	1.310	14.03
isolet5 dataset ($m = 1559$, $n = 618$)								
1	618	2	$3.2{\cdot}10^{-3}$	0.014	34.400	1	1	56.18
2	4916	9	$4.7{\cdot}10^{-5}$	0.244	622.926	1.051	1.122	7.77
3	76173	131	$3.3{\cdot}10^{-6}$	4.234	7360.414	1.206	1.177	30.77
CNAE-9 dataset ($m = 1080$, $n = 857$, number of nonzero columns is 695)								
1	695	2	$2.3{\cdot}10^{-3}$	0.020	34.500	1	1	695.00
2	2082	4	$1.1{\cdot}10^{-5}$	0.172	1739.796	1	1	2.97
3	7613	12	$1.1{\cdot}10^{-7}$	0.789	$1.1\cdot10^{5}$	1	1	3.65
4	23456	35	$1.6{\cdot}10^{-9}$	2.615	$7.0\cdot10^{6}$	1.018	1	3.64
5	136765	201	$5.3{\cdot}10^{-11}$	15.665	$2.0\cdot10^{8}$	1.038	1.007	5.06
techtc01 dataset ($m = 163$, $n = 29261$, number of nonzero columns is 9238)								
1	9238	2	$6.8{\cdot}10^{-5}$	0.007	2231.078	1	1	4618.50
2	27711	4	$9.3{\cdot}10^{-9}$	0.017	$1.3\cdot10^{7}$	1	1	3.00
3	240017	27	$6.5{\cdot}10^{-12}$	0.070	$3.0\cdot10^{10}$	1	1	8.66
4	2823249	308	$1.0{\cdot}10^{-14}$	1.000	$1.5\cdot10^{13}$	1.035	1	41.62

Figure 3: Performance comparison on various datasets (see text)

k	error TS/A*	error GKS/A*
1	1.000	1.000
3	1.102	1.204
5	1.129	1.164
7	1.133	1.126
9	1.125	1.125
11	1.125	1.125
13	8.860	1.157
15	8.368	1.125
16	8.122	1.125

Figure 4: Accuracy comparison on a 50×50 Kahan matrix

Column 2 shows the number of children nodes evaluated by the algorithm at Step 1.3.1, and not eliminated at Step 1.3.2.1. This is the number of heuristic values that needs to be calculated. Column 3 shows the number of parent nodes evaluated by the algorithm. These are the nodes that have their children calculated. It is typically much smaller than the number of children nodes.

Column 4 shows the ratio between Column 3 and $\binom{n}{k}$, the number of nodes that need to be evaluated by exhaustive search. Observe that this number is typically very small, especially for the larger datasets.

Columns 5,6 show the run time. Column 5 gives the run time in minutes, and Column 6 gives the ratio between run time of an exhaustive search and Column 5. In most cases the exhaustive search run time was not explicitly calculated. We measured the exhaustive search run time for a small value k_1, and then extrapolated to the desired value of k. This was done by multiplying the time taken for k_1 by $\frac{\binom{n}{k}}{\binom{n}{k_1}}$. Observe that our algorithm is typically much faster than exhaustive search by many orders of magnitude.

To evaluate the gain in accuracy over previously proposed algorithms we compared the results with two other algorithms. The first is the state-of-the-art Two-Stage algorithm as described in (Boutsidis, Mahoney, and Drineas 2009). The second is the classical algorithm of Golub Kelma and Stewart, as described, for example, in (Golub and Van-Loan 1996). In the table we refer to these algorithm as the "TS" and the "GKS".

Column 7 shows the ratio between the error of the Two-Stage algorithm and the error of our algorithm. Column 8 shows the ratio between the error of the GKS and the error of our algorithm. Clearly, these ratios are at least 1, and in most cases they are bigger than 1. In some cases, e.g. when experimenting with the Madelon dataset, the reduction of the error is very significant.

Column 9 shows the memory reduction obtained by the pruning technique discussed in Section 5. The saving is sometimes as small as a factor of 2, and sometimes as large as a factor of 4600.

6.2 Second experiment

The results of the first experiment were performed on real data that comes from real applications. While it clearly shows the accuracy advantage of our algorithm when compared to the TS and the GKS, there were many cases in which the accuracy was identical. (These are indicated by the value of 1 in columns 7,8.) We ran a second experiment, computing column subset selection on a matrix that is known to be a challenge for subset selection algorithms. The matrix structure was proposed by Kahan (Kahan 1966), and the results are shown in Figure 4. Observe that in most cases our results are better by at least 10%.

7 Concluding remarks

This paper describes an optimal heuristic search algorithm for column subset selection. It gives huge run-time improvements over exhaustive search, but it is much slower than the current state-of-the-art non-optimal algorithms.

Our algorithm should not be considered as a competitor to previously proposed fast but non-optimal algorithms. For example, to the best of our knowledge prior to our algorithm there was no guaranteed way of selecting the best 4 columns from a dataset such as the techtc01. In fact, even if an algorithm selects, perhaps by luck, the best 4 columns (e.g., the GKS), there is no current technique that can verify that these are, indeed, the best. By contrast, our algorithm run time for selecting these columns is about 1 minute, and the selection is guaranteed to be optimal.

Thus, our algorithm solves a range of problems that were previously computationally intractable. If one is interested in selecting a small number of columns, running our algorithm for a few minutes may not be much different than running a fast algorithm for under a second. However, our algorithm should not be used when hundreds of columns are needed.

With regard to practical applications, it appears that there are situations where one may be interested in a selection of a small number of features. An example is data mining and knowledge representation, where one may be interested in identifying a small number of features that "cover" the entire data. One can typically "understand" the result if it contains a small number of features, but not if the number of features is large.

A different situation where our algorithm may be used "in practice" is in two-stage algorithms. In the first stage these algorithms apply a fast method for selecting larger than necessary subset of the columns. In the second stage, an accurate algorithm is applied to select the desired number of columns out of those selected in the first stage. Our algorithm can be used for the second stage.

The work described here can be extended in several directions. It may be interesting to identify other search techniques that use admissible or consistent heuristics and apply them with the heuristics developed here. It is also interesting to try and identify other problems in numerical linear algebra that can benefit from the optimality inherent in heuristic combinatorial search.

8 Acknowledgments

The first author would like to thank the Japan Air Self-Defence Force for supporting his studies.

References

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

Boutsidis, C.; Mahoney, M. W.; and Drineas, P. 2009. An improved approximation algorithm for the column subset selection problem. In Mathieu, C., ed., *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, 968–977. SIAM.

Çivril, A., and Magdon-Ismail, M. 2009. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science* 410(47-49):4801–4811.

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

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

Dasgupta, A.; Drineas, P.; Harb, B.; Josifovski, V.; and Mahoney, M. W. 2007. Feature selection methods for text classification. In Berkhin, P.; Caruana, R.; and Wu, X., eds., *KDD*, 230–239. ACM.

Deshpande, A., and Rademacher, L. 2010. Efficient volume sampling for row/column subset selection. In *FOCS*, 329–338. IEEE Computer Society Press.

Deshpande, A.; Rademacher, L.; Vempala, S.; and Wang, G. 2006. Matrix approximation and projective clustering via volume sampling. *Theory of Computing* 2(12):225–247.

Drineas, P.; Lewis, J.; and Paschou, P. 2010. Inferring geographic coordinates of origin for europeans using small panels of ancestry informative markers. *PLoS ONE* 5(8):e11892. doi:10.1371/journal.pone.0011892.

Frieze, A. M.; Kannan, R.; and Vempala, S. 2004. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM* 51(6):1025–1041.

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

Gabrilovich, E., and Markovitch, S. 2004. Text categorization with many redundant features: Using aggressive feature selection to make SVMs competitive with C4.5. In *The 21st International Conference on Machine Learning (ICML)*, 321–328.

Golub, G. H., and Van-Loan, C. F. 1996. *Matrix computations*. The Johns Hopkins University Press, third edition.

Golub, G. H. 1973. Some modified matrix eigenvalue problems. *SIAM Review* 15(2):318–334.

Gu, M., and Eisenstat, S. C. 1996. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM J. Computing* 17(4):848–869.

Guruswami, V., and Sinop, A. K. 2012. Optimal column-based low-rank matrix reconstruction. In Rabani, Y., ed., *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, 1207–1214. SIAM.

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

Hastie, T.; Tibshirani, R.; and Friedman, J. 2009. *The Elements of Statistical Learning*. Springer, second edition. section 3.3.1.

Kahan, W. 1966. Numerical linear algebra. *Canadian Mathematical Bulletin* 9:757–801.

Kumar, S.; Mohri, M.; and Talwalkar, A. 2012. Sampling methods for the nystrom method. *Journal of Machine Learning Research* 13:981–1006.

Maung, C., and Schweitzer, H. 2013. Pass-efficient unsupervised feature selection. In *Advances in Neural Information Processing Systems (NIPS)*, volume 26, 1628–1636.

Melman, A. 1998. Analysis of third-order methods for secular equations. *Mathematics of Computation* 67(221):271–286.

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

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