PARTIAL SEARCH VECTOR SELECTION FOR SPARSE SIGNAL REPRESENTATION

Karl Skretting and John Håkon Husøy

Stavanger University College, Department of Electrical and Computer Engineering
P. O. Box 8002, N-4068 Stavanger, Norway
Phone: +47 51 83 20 16, Fax: +47 51 83 17 50, E-mail: karl.skretting@tn.his.no

ABSTRACT

In this paper a new algorithm for vector selection in signal representation problems is proposed, we call it Partial Search (PS). The vector selection problem is described, and one group of algorithms for solving this problem, the Matching Pursuit (MP) algorithms, is reviewed. The proposed algorithm is based on the Order Recursive Matching Pursuit (ORMP) algorithm, it extends ORMP by searching a larger part of the solution space in an effective way. The PS algorithm tests up to a given number of possible solutions and returns the best, while ORMP is a greedy algorithm testing and returning only one possible solution. A detailed description of PS is given. In the end some examples of its performance are given, showing that PS performs very well, that the representation error is considerable reduced and that the probability of finding the optimal solution is increased compared to ORMP, even when only a small part of the solution space is searched.

1. INTRODUCTION

An overcomplete set of N-dimensional vectors, spanning the space $\mathbb{R}^N$, is a *frame*. The frame vectors, denoted $\{\mathbf{f}_k\}_{k=1}^K$, can be regarded as columns in an $N \times K$ matrix $\mathbf{F}$, where $K \geq N$. A signal vector, $\mathbf{x}$, can be represented, or approximated if the representation is not exact, as a linear combination of the frame vectors,

$$\tilde{\mathbf{x}} = \sum_{k=1}^{K} w(k)\mathbf{f}_k = \mathbf{F}\,\mathbf{w}. \qquad (1)$$

The expansion coefficients, or weights, are collected into a vector $\mathbf{w}$ of length K. The original signal vector is the sum of the signal approximation and an error which may be zero, $\mathbf{x} = \tilde{\mathbf{x}}+\mathbf{r}$. A frame is uniform if all the frame vectors are normalized, i.e. $\|\mathbf{f}_k\| = 1$. Without loss of generality we may assume that the frame in Equation 1 is uniform. The representation is *sparse* when only some few, say s, of the expansion coefficients are non-zero, the *sparseness factor* S, is then given by $S = s/N$.

The vector selection problem (for a sparseness factor given by the maximum number of non-zero weights allowed, s) is to find the weights in Equation 1 such that the norm of the error, $\|\mathbf{r}\| = \|\mathbf{x} - \mathbf{F}\mathbf{w}\|$, is as small as possible. Generally no exact, $\mathbf{x} = \mathbf{F}\mathbf{w}$, solution exists when $s < N$. The choice of weights that minimizes the 2-norm of the residual (error) is called the "optimal solution" or the optimal approximation in the least squares sense. An ϵ-solution is a choice of weights such that the 2-norm of the residual is smaller than a given limit, ϵ. Davis [1] has proved that finding whether an ϵ-solution exists or not is an NP-complete[1] problem, and that finding the optimal approximation for a given s is an NP-hard problem when the 2-norm is used.

The only way to find the optimal solution to the vector selection problem is to examine all the possible combinations for selecting s vectors out of the K frame vectors available. The number of different combinations is $\binom{K}{s}$. Thus a full search algorithm is only practical for small problems. A large amount of work has been done on alternative non-optimal algorithms, the two main alternatives are:
1) Thresholding of the weights found by algorithms returning generally non-sparse solutions. Examples are Basis Pursuit (BP) [2] and FOCal Under-determined System Solver (FOCUSS) [3,4].
2) Greedy algorithms, collectively referred to as Matching Pursuit (MP) algorithms. These algorithms involves the selection of one vector at each step. Variants are Basic Matching Pursuit (BMP), Orthogonal Matching Pursuit (OMP) and Order Recursive Matching Pursuit (ORMP). We will examine them closer in the next section.
Some vector selection algorithms are described and compared in [5-7].

[1] An NP-complete problem is as hard as (can be transformed in polynomial time into) another problem known to be NP-complete. An NP-hard problem is at least as hard as an NP-complete problem.

Basic Matching Pursuit (BMP) algorithm:

```
1   w = BMP(F, x, s)
2     w := 0, r := x
3     while not Finished
4         c := F^T · r
5         find k : |c(k)| = max_j |c(j)|
6         w(k) := w(k) + c(k)
7         r := r − c(k) · f_k
8a        Finished := (||r|| < some Limit)
8b        Finished := (s non-zero entries in w)
9     end
10    return
```

Figure 1: The Basic Matching Pursuit algorithm.

In this paper we present the partial search method for vector selection. It may be seen as a reduction of the full search method, instead of searching all the possible combinations it only searches the combinations containing the most promising frame vectors. Alternatively it may be seen as an extension of the ORMP method: Instead of only selecting the most promising frame vector in each step, it may try several frame vectors in each step. The proposed algorithm use QR factorization the same way as the fast variants of OMP and ORMP. The partial search method was first presented as a recursive function without the fast QR factorization in [8]. In Section 2 we present the MP algorithms and the proposed algorithm for partial search is described in Section 3. In Section 4 some implementation details for the fast variants based on QR factorization are presented. And finally, in Section 5, some numerical examples illustrate the performance of PS compared to BMP, OMP and ORMP.

2. MATCHING PURSUIT ALGORITHMS

The simplest of the Matching Pursuit algorithms is **Basic Matching Pursuit** (BMP), presented and analyzed in [1] and [9]. An overview of the algorithm is in Figure 1. This variant of the BMP algorithm as well as the other algorithms presented here, requires that the frame, $\mathbf{F}$, is uniform. The BMP algorithm has the following four main steps: Initialize, Loop, Select and Update.

In the initialize step the weights, $\mathbf{w}$, are set to zero, and the residual, i.e. the representation error using the current weights, $\mathbf{r} = \mathbf{x} - \mathbf{F}\mathbf{w}$, is set to the given signal vector, line 2 in Figure 1. The main loop, line 3-9, is done until some stop criterion is met. It may be that the representation error is small enough, line 8a, or that the desired number of non-zero entries in $\mathbf{w}$ is reached, line 8b, or that the loop has been executed many times, this third test is not included in Figure 1.

A (new) frame vector is selected to be used in the linear combination which makes up the approximation. As this is a greedy algorithm we select the vector that looks best at the current stage. Within the loop the inner products of the current residual and the frame vectors are calculated, line 4, and the one with the largest absolute value is identified, line 5. Frame vector k is selected. It may happen that this is a vector selected earlier, and thus the number of non-zero entries in $\mathbf{w}$ is not increased but one of the already selected entries is updated and the error is reduced. In fact, we may use this to reduce the error without selecting new frame vectors (as long as the error is not orthogonal to the space spanned by the selected frame vectors) by only considering the frame vectors already selected in line 5.

The weight for the selected frame vector is updated, line 6, and the residual is updated, line 7. We should note that the calculation in line 7 makes the residual orthogonal to the frame vector just selected, but it also add components in the same directions as previously selected frame vectors since these are not orthogonal to the frame vector just selected.

The BMP algorithm is quite simple, but it is not very effective. The loop may be executed many times, often more than s times, and the calculation of the inner products, line 4, is demanding, approximately NK multiplications and additions are done.

Orthogonal Matching Pursuit (OMP) [1,10] is a refinement of BMP. The only difference is in the update step, line 6 and 7 in Figure 1. The approximation after a new frame vector has been selected is made by projecting the signal vector onto the space spanned by the selected frame vectors. When i frame vectors are selected the index set of the selected frame vectors is denoted $I = \{k_1, k_2, \ldots, k_i\}$ and they are collected into a matrix $\mathbf{A} = [\mathbf{f}_{k_1}, \mathbf{f}_{k_2}, \ldots, \mathbf{f}_{k_i}]$ of size $N \times i$. The approximation is $\tilde{\mathbf{x}} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{x} = \mathbf{A}\mathbf{b}$ and the error is as before $\mathbf{r} = \mathbf{x} - \tilde{\mathbf{x}}$. Here, $\mathbf{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{x}$ is the values of the non-zero entries in $\mathbf{w}$ given by the index set I. With this change the residual will always be orthogonal to the selected frame vectors, and the select step will always find a new vector. This ensures that the loop is at most executed s times, but, in a straight forward implementation, the orthogonalization of the residual add extra calculations to the loop.

Order Recursive Matching Pursuit (ORMP) [12-15] is another MP variant. Like OMP this one always keeps the residual orthogonal to all the selected frame vectors. The only difference between ORMP and OMP is that ORMP adjust the inner products before the largest is found, line 5 in Figure 1 will be "find $k : |c(k)/u(k)| = \max_j |c(j)/u(j)|$" instead of "find $k : |c(k)| = \max_j |c(j)|$". Let us explain this in

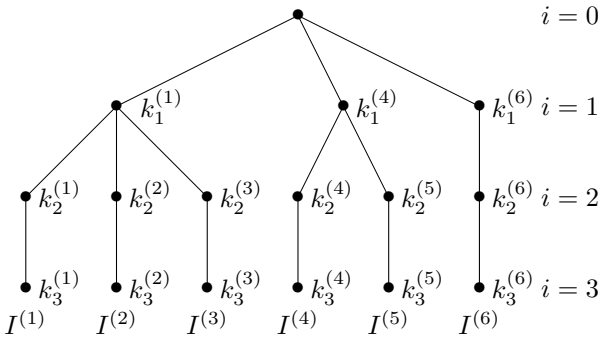

Figure 2: Search tree for the example in Section 3.

more detail:

We denote the subspace of $\mathbb{R}^N$ spanned by the column vectors in $\mathbf{A}$, i.e. the selected frame vectors, as A, and its orthogonal complement as $A^\perp$. A frame vector, $\mathbf{f}_j$, can be divided into two orthogonal components, the part in A of length $a(j)$, and the part in $A^\perp$ of length $u(j)$, and $a(j)^2 + u(j)^2 = \|\mathbf{f}_j\|^2 = 1$, where the latter equality follows since the frame is uniform. Lengthening each remaining or not yet selected frame vector $\mathbf{f}_j$ by a factor $1/u(j)$, makes the lengths of the component in $A^\perp$ equal to 1. This part is the only one we need to consider since the residual, $\mathbf{r}$, is in $A^\perp$. Thus, finding maximum of $|c(j)|/u(j) = (\mathbf{f}_j^T\mathbf{r})/u(j) = (\mathbf{f}_j^T/u(j))\mathbf{r}$, instead of only $|c(j)|$, when selecting the next frame vector is like selecting a vector from a uniform frame in $A^\perp$, i.e. the frame vectors are orthogonalized to A and normalized. ORMP will generally select different frame vectors than OMP, and often the total residual will be smaller. A "straight forward" implementation of ORMP will be even more demanding than OMP.

3. THE PARTIAL SEARCH ALGORITHM

The idea of the Partial Search (PS) algorithm is to search many combinations, i.e. many sets of frame vectors, to find the best solution. In PS we let the number of combinations, NoC, to search be given as an input argument. A particular combination, numbered m, has index set $I^{(m)} = \{k_1^{(m)}, k_2^{(m)}, \ldots, k_s^{(m)}\}$ and the matrix of the selected frame vectors is denoted $\mathbf{A}^{(m)}$. Each combination gives an error, $\mathbf{r}^{(m)} = \mathbf{x} - \tilde{\mathbf{x}}^{(m)} = \mathbf{x} - \mathbf{A}^{(m)}(\mathbf{A}^{(m)T}\mathbf{A}^{(m)})^{-1}\mathbf{A}^{(m)T}\mathbf{x}$. The index set that gives the smallest error also gives the solution, i.e. the sparse weights as in OMP.

The different combinations can be ordered in a search tree as in Figure 2, here $NoC = 6$. Each node, except the root, in the tree represent the selection of one frame vector. The index set given by each leaf is given by the path from the root to the leaf. An index set may have the same start for the index sequence as another index set, i.e. $k_1^{(3)} = k_1^{(2)} = k_1^{(1)}$ and $k_1^{(5)} = k_1^{(4)}$. In the PS algorithm the tree is searched in a depth-first way. The current state is saved each time, on the way down, a node with more than one child is reached, and on the way up the state is retrieved before visiting the next child. At each leaf the norm of the error is calculated and compared to the smallest error found until now.

One important part of the PS algorithm is to decide which frame vectors to select, the indexes $k_i^{(m)}$ to include in the search tree. A function, named "DistributeM", does this task each time a new node is visited, starting at the root where $i = 0$ and at all the levels down to $i = s - 1$. This function distribute the number of combinations available M, at the root ($i = 0$) this is $M = NoC$, by returning a sequence of non-negative integers, M_k for $k = 1, 2, \ldots, K$, such that $M_k > 0$ if frame vector k is to be included and $\sum_{k=1}^{K} M_k = M$. M_k is used as the M value at the next level. This is done according to the rules decided when building the function and it always uses the inner products $\mathbf{c}$ (actually the adjusted inner products $\mathbf{d}$ where $d(j) = |c(j)|/u(j)$ since the ORMP variant is the preferred one), and M. The function may also depend on the number of vectors already selected i, the maximum number of vectors to select s, or the frame size N and K. The best (criterium as in ORMP) frame vector should be the first one, and thus make sure that the ORMP index set is $I^{(1)}$, also the selected indexes should be the best ones in $\mathbf{d}$. In the example in Figure 2 $NoC = 6$ combinations are available at the root, these are distributed so that $M_{k_1^{(1)}} = 3$, $M_{k_1^{(4)}} = 2$ and $M_{k_1^{(6)}} = 1$ and $M_k = 0$ for the rest of the k's. At level $i = 1$ and for node $k_1^{(1)}$ the $M = 3$ combinations to search are distributed such that $M_{k_2^{(1)}} = 1$, $M_{k_2^{(2)}} = 1$ and $M_{k_2^{(3)}} = 1$.

We can note that all the leafs in the tree are at level $i = s$ and that none of the leafs has any siblings. At level $i = s - 1$ it makes no sense to try more than the best frame vector since this is the one that will give the smallest error.

An effective implementation of the PS algorithm should exploit the tree structure to avoid doing the same calculations several times. It is also important that as few calculations as possible is done to find the norm of the error at each leaf, it is not necessary to form the matrices $\mathbf{A}^{(m)}$ nor to do matrix inversion. The same technique as used in QR factorization in OMP and ORMP can be applied in the PS algorithm. The details are in the next section.

OMP/ORMP with QR factorization:

```
1   w = OMPqr/ORMPqr(F, x, s)
2     R := 0
3     i := 0, I := {}, J := {1, 2, ..., K}
4     e := 1, u := 1
5     c := F^T x, d := |c|
6     nx := x^T x, nxLim := nx · 10^-10
7     while 1
8         find k : d(k) = max_j d(j)
9         i := i + 1, I := I ∪ {k}, J := J \ {k}
10        R(i, k) := u(k)
11        nx := nx − c(k)^2
12        if (i ≥ s) or (nx < nxLim) exit loop
13        for j ∈ J
14          R(i, j) := f_k^T f_j
15          for n := 1 to (i − 1)
16            R(i, j) := R(i, j) − R(n, k)R(n, j)
17          end
18          R(i, j) := R(i, j)/u(k)
19          c(j) := c(j)u(j) − c(k)R(i, j)
20          if OMP d(j) := |c(j)|
21          e(j) := e(j) − R(i, j)^2
22          u(j) := sqrt(|e(j)|)
23          c(j) := c(j)/u(j)
24          if ORMP d(j) := |c(j)|
25        end
26        d(k) := 0
27    end
28    w := 0
29    w(I) := R(1 : i, I)^-1 c(I)
30    return
```

Figure 3: The OMP/ORMP algorithm using QR factorization.

Partial Search with QR factorization:

```
1   w = PSqr(F, x, s, NoC)
2     Initialize R, M, e, u, c, d, nx, nxLim,
          nxBest, GoingDown, i, I and J
3     while 1
4         if GoingDown
5           increment i
6           find k : d(k) = max_j d(j)
7           call "DistributeM" function
8           if "two or more children" store state
9         end
10        GoingDown = true
11        I := I ∪ {k}, J := J \ {k}
12        R(i, k) := u(k)
13        nx := nx − c(k)^2
14        if (nx < nxLim) find w by back-
            substitution and exit loop
15        if (i < s)
16          Update R(i, J), c, d, e and u
17        else
18          if (nx < nxBest) update nxBest and
              find w by back-substitution
19          decrement i, set I, J
              until ("sibling" found) or (i = 0)
20          if (i = 0) exit loop
21          set k to index of "sibling"
22          retrieve state
23          GoingDown = false
24        end
25    end
26    return
```

Figure 4: The Partial Search algorithm using QR factorization. More explanation in text.

4. IMPLEMENTATION DETAILS

Fast algorithms based on **QR factorization** have been presented for both OMP and ORMP, [1,13,15]. One such algorithm is given in Figure 3. QR factorization is done on the matrix of the selected frame vectors, denoted $\mathbf{A}$ above. Actually the $\mathbf{Q}$ matrix is not explicitly formed, only the $\mathbf{R}$ matrix is stored. QR factorization is done as in the stable Modified Gram-Schmidt algorithm [15], after the selection of a new frame vector all the remaining frame vectors are orthogonalized to the one just selected.

We will now look closer into the algorithm shown in Figure 3. The matrix $\mathbf{R}$ is of size $s \times K$, where $\mathbf{R}(1 : i, I)$ is the $\mathbf{R}$ matrix in QR factorization of matrix $\mathbf{A}$. I is the index set for the selected vectors and J is the index set of the remaining frame vectors. The vectors $\mathbf{e}$, $\mathbf{u}$, $\mathbf{c}$ and $\mathbf{d}$ in line 4 and 5 are all of length K, $e(k)$ is the length squared of $\mathbf{f}_k$ projected into $A^{\perp}$ and $u(k)$ is the length. $\mathbf{c}$ contains the inner products of the frame vectors and the residual, and $\mathbf{d}$ theirs absolute values (OMP) or the absolute values diveded by the lengths $u(k)$ (ORMP). nx will be updated in line 11 to be $nx = \|\mathbf{r}\|^2$ after the selection of each new frame vector.

The main loop, line 7-27, selects one frame vector each time it is executed, line 8 and 9. If allowed number of vectors are selected, or the norm of the residual is small, the loop is ended. To prepare for selection of more frame vector more work is needed, line 13-26. First (entries for unselected frame vectors in) row i of $\mathbf{R}$ is updated, line 14-18, then vectors $\mathbf{c}$, $\mathbf{d}$, $\mathbf{e}$ and $\mathbf{u}$ are updated, line 19-24. Finally, $d(k)$ is set to zero to ensure that new vectors are selected. When then desired (or needed) number of frame vectors are selected a linear equation system must be solved to find the weights. Since the matrix $\mathbf{R}(1 : i, I)$ is triangular this can be solved quite simply by backward substitution, line 29.

The kernel of the PS algorithm is ORMPqr, it is wrapped in a walk through the search tree. The PS

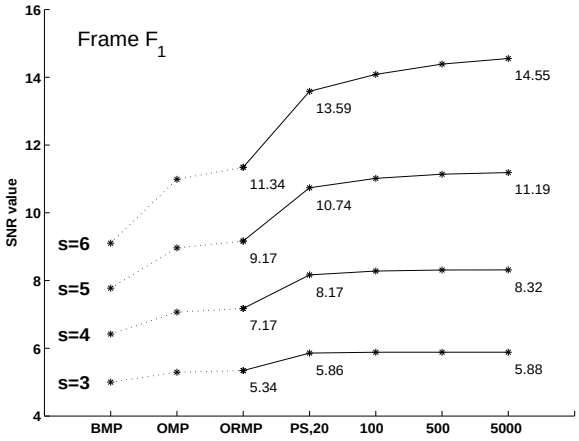

Figure 5: The approximation errors for the random Gaussian data using the "random" frame $\mathbf{F}_1$.

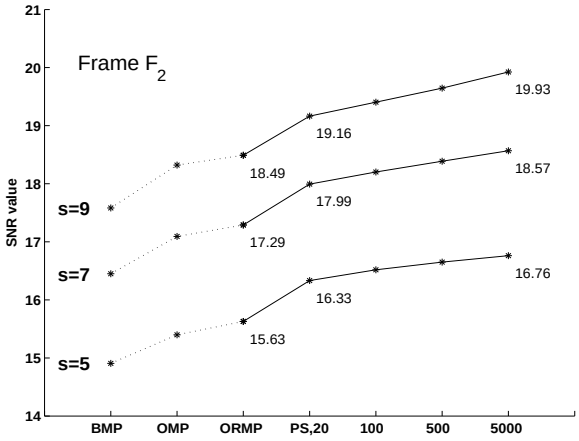

Figure 6: The approximation errors for the AR(1) signal vectors using the "designed" frame $\mathbf{F}_2$.

algorithm is summarized in Figure 4. Initialization in line 2 is like line 2-6 in Figure 3, but some few extra variables are included. $\mathbf{M}$ is a matrix or a table used to store the different M values found by the "DistributeM" function. The state stored in line 8 and retrieved in line 22 is the variables $\mathbf{c}$, $\mathbf{e}$, $\mathbf{u}$ and nx. Note that the state is assumed to be stored in a table, where i is the position, that allows a state stored once to be retrieved several times. Line 16 updates $\mathbf{R}(i, J)$, $\mathbf{c}$, $\mathbf{d}$, $\mathbf{e}$ and $\mathbf{u}$ like line 13-26 in Figure 3. When the walk through the search tree has reached a leaf lines 18-23 are executed, we go up in the search tree (decrement i) until a sibling to the right of the current node is found. The search is finished when the root is reached.

5. SIMULATION EXAMPLES

In this section we will give some examples of the performance of the Partial Search (PS) algorithm. The vector selection algorithms that will be used are BMP, OMP, ORMP, and Partial Search with the allowed number of combinations to search, NoC, set as 20, 100, 500 and 5000. The methods will be along the x-axis in Figures 5 and 6. For a given sparseness factor the results of BMP, OMP and ORMP are loosely connected by a dotted line and the results of ORMP and PS with different values for NoC are connected by a solid line since here intermediate values can be assumed using different values for NoC. Note that ORMP can be regarded as PS with $NoC = 1$.

Having a set of L test vectors, $\mathbf{x}_l$ for $l = 1$ to L, we find sparse approximations to each of the test vectors using the different vector selection methods. The ideal situation now would be if we could compare these results to the optimal solution, but this is only possible for very small problems. Thus we can not find for how many of these L test vectors the optimal solution were found, or how close the resulting error is to the error of the optimal solution. The approximation error for each test vector can be used to compare the different methods to each other. It can be quantified by the Signal to Noise Ratio (SNR) defined (eg. estimated) as

$$SNR = -10 \log_{10} \frac{\sum_{l=1}^{L} \|\mathbf{r}_l\|^2}{\sum_{l=1}^{L} \|\mathbf{x}_l\|^2} = -10 \log_{10} \frac{E_r}{E_x} \quad (2)$$

where $\mathbf{r}_l = \mathbf{x}_l - \tilde{\mathbf{x}}_l$ and the 2-norm is used. The denominator in this fraction is constant for a given set of test vectors. Thus, an increase in the SNR value by 1 is a reduction of the energy in the error, E_r, by 20%. The SNR will be along the y-axis in Figures 5 and 6. As more combinations are searched, NoC increases, the optimal solution is more likely to be found, and the total approximation error will be smaller (higher SNR). When a (considerable) increase in NoC does not give a higher SNR we may expect that the optimal solution is found for most of the test vectors.

In the first example a set of $L = 5000$ random Gaussian vectors is generated. The test vectors are of length $N = 16$, each entry is independent and Gaussian distributed (zero mean and variance $\sigma = 1$). Each of these vectors is approximated by a sparse representation, $s = 3, \ldots, 6$ frame vectors were used for these approximations. The frame vectors in $\mathbf{F}_1$, size 16×48, are selected similar to the test vectors, i.e. they are random Gaussian vectors which are normalized, $\|\mathbf{f}_k\| = 1$. The results are presented in Figure 5.

It is perhaps more interesting to look at the vector selection capabilities for not completely random signals. This is done in the second example. A long

AR(1) signal[2], with $\rho = 0.95$ is divided into $L = 5000$ non-overlapping blocks, each with length $N = 32$, these make up the other set of test vectors, which in Figure 6 is approximated by a sparse representation using $s = 5$, 7 or 9 frame vectors. The frame $\mathbf{F}_2$, of size 32×80, is a frame specially designed for AR(1) signals, the frame design method is described in [8]. The sparse representation results are presented in Figure 6.

The conclusions we can draw from these examples are: 1) Comparing the MP algorithms we see that OMP and ORMP both give better approximations than BMP, and that ORMP is the better one. 2) The PS algorithm is considerable better than the MP algorithms, already for $NoC = 20$ the probability of finding a better (the optimal) solution is considerable higher. 3) For values of $s \leq 5$, or not a *large* number of different possible combinations $\binom{K}{s}$, it seems that PS is likely to find the optimal solution with a small or moderate value of NoC.

One disadvantage of PS is that the execution time is larger than for ORMP (and OMP). This can be seen in the following table for frame $\mathbf{F}_2$ used on 5000 test vectors with $s = 9$.

frame $\mathbf{F}_2$, $s = 9$ Method	Running Time	Time for each combination
ORMP, ($NoC = 1$)	3.23 s	3.23 s
PS, $NoC = 20$	53.6 s	2.68 s
PS, $NoC = 100$	4 min 9 s	2.49 s
PS, $NoC = 500$	19 min 45 s	2.37 s
PS, $NoC = 5000$	3 h 13 min	2.32 s

The execution time increases almost linearly with the number of combinations searched. The time for searching one combination decreases as more combinations are searched, this is natural since a new combination usually has some of the first vectors in common with the previously searched combination, and this is exploited it the algorithm.

6. CONCLUSION

The partial search algorithm presented in this paper finds a solution to the vector selection problem at least as good as the ORMP algorithm. The optimal solution is often found by searching relatively few combinations, often 20 to 100 combinations seems enough. The search among the different combinations is done in a fast and effective way. The improvement in representation when going from ORMP to partial search is the same size as the improvement when going from BMP to ORMP, or better if we allow to search many combinations.

[2] An AR(1) signal is the output of an autoregressive (AR) process of order 1.

7. REFERENCES

[1] G. Davis, *Adaptive Nonlinear Approximations*, Ph.D. thesis, New York University, Sept. 1994.

[2] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Journal of Scientific Computing*, vol. 20, no. 1, pp. 33–61, 1998.

[3] I. F. Gorodnitsky and B. D. Rao, "Sparse signal reconstruction from limited data using FOCUSS: A re-weighted minimum norm algorithm," *IEEE Trans. Signal Processing*, vol. 45, no. 3, pp. 600–616, Mar. 1997.

[4] K. Engan, *Frame Based Signal Representation and Compression*, Ph.D. thesis, Norges teknisk-naturvitenskapelige universitet (NTNU)/Høgskolen i Stavanger, Sept. 2000, Available at `http://www.ux.his.no/~kjersti/`.

[5] B. D. Rao, "Signal processing with the sparseness constraint," in *Proc. ICASSP '98*, Seattle, USA, May 1998, pp. 1861–1864.

[6] B. D. Rao and K. Kreutz-Delgado, "Sparse solutions to linear inverse problems with multiple measrement vectors," in *Proc. DSP Workshop*, Bryce Canyon, Utah, USA, Aug. 1998.

[7] S. F. Cotter, J. Adler, B. D. Rao, and K. Kreutz-Delgado, "Forward sequential algorithms for best basis selection," *IEE Proc. Vis. Image Signal Process*, vol. 146, no. 5, pp. 235–244, Oct. 1999.

[8] K. Skretting, *Sparse Signal Representation using Overlapping Frames*, Ph.D. thesis, NTNU Trondheim and Høgskolen i Stavanger, Oct. 2002, Available at `http://www.ux.his.no/~karlsk/`.

[9] S. G. Mallat and Z. Zhang, "Matching pursuit with time-frequency dictionaries," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[10] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," Nov. 1993, Proc. of Asilomar Conference on Signals Systems and Computers.

[11] S. Singhal and B. S. Atal, "Amplitude optimization and pitch prediction in multipulse coders," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 3, pp. 317–327, Mar. 1989.

[12] S. Chen and J. Wigger, "Fast orthogonal least squares algorithm for efficient subset model selection," *IEEE Trans. Signal Processing*, vol. 43, no. 7, pp. 1713–1715, July 1995.

[13] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM journal on computing*, vol. 24, pp. 227–234, Apr. 1995.

[14] M. Gharavi-Alkhansari and T. S. Huang, "A fast orthogonal matching pursuit algorithm," in *Proc. ICASSP '98*, Seattle, USA, May 1998, pp. 1389–1392.

[15] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, Siam, Philadelphia, PA, USA, 1997.