ONLINE SEARCH ORTHOGONAL MATCHING PURSUIT

Alejandro J. Weinstein and Michael B. Wakin

Colorado School of Mines
Department of Electrical Engineering and Computer Science
1500 Illinois Street, Golden, CO 80401, USA

ABSTRACT

The recovery of a sparse signal x from $y = \Phi x$, where Φ is a matrix with more columns than rows, is a task central to many signal processing problems. In this paper we present a new greedy algorithm to solve this type of problem. Our approach leverages ideas from the field of *online search* on state spaces. We adopt the "agent perspective" and consider the set of possible supports of x as the state space. Under this setup, finding a solution is equivalent to finding a path from the empty support set to the state whose support has both the desired cardinality and the capacity to explain the observation vector y. An empirical investigation on Compressive Sensing problems shows that this new approach outperforms the classic greedy algorithm Orthogonal Matching Pursuit (OMP) while maintaining a reasonable computational complexity.

Index Terms— Compressive Sensing, greedy algorithms, Orthogonal Matching Pursuit (OMP), sparse approximation.

1. INTRODUCTION

Many areas of signal processing, including Compressive Sensing, image inpainting and others, involve solving a sparse approximation problem. This corresponds to solving a system of equations $y = \Phi x$ where the matrix Φ has more columns than rows and x is a sparse vector. An important class of methods for solving this problem are the so called greedy algorithms, for which Orthogonal Matching Pursuit (OMP) is one of the classic representatives [1].

It is possible to think of greedy algorithms as instances of *search problems*. Karahanoğlu and Erdoğan [2] used the A* search method, a well known heuristic search algorithm for finding the shortest path between two nodes in a graph, to design a new greedy solver called A*OMP. This method stores the solution as a tree, where each node represents an index of the estimated support. At each iteration it selects, using a heuristic based on the evolution of the norm of the residue, which leaf node to expand. To avoid an exponential growing of the candidate solutions, this tree is pruned by keeping a relatively small number of leaves.

This work was partially supported by AFOSR grant FA9550-09-1-0465.

In this paper we present a new greedy algorithm for solving sparse approximation problems. Like A*OMP, it frames the recovery of a sparse signal as a search instance. However, instead of using A* search which involves a monolithic planning stage, we formulate the problem as an *online* search, where the planning and execution stages are interleaved. This allows us to achieve a performance significantly better than OMP and similar to A*OMP while maintaining a reasonable computational load. Our simulations confirm this recovery performance with a computational speed $20\times$ faster than A*OMP and less than $2\times$ slower than OMP.

2. PRELIMINARIES

We start this section by introducing some notation and defining the problem we wish to solve. Let x be a real K-sparse vector of dimension N. That is, $x \in \mathbb{R}^N$ with $\|x\|_0 = K$, where $\|\cdot\|_0$ denotes the number of non-zero elements of a vector. Let the support of x be the index set $\mathrm{supp}(x) = \{i \mid x(i) \neq 0\}$. Let $y = \Phi x$, with $\Phi \in \mathbb{R}^{M\times N}$ an $M \times N$ full-rank matrix with $M < N$ and $y \in \mathbb{R}^M$ a measurement or observation vector. Let ϕ_j denote column j of Φ, and we assume that $\|\phi_j\|_2 = 1$ for all j. For any index set Γ, let Φ_Γ be the $M \times |\Gamma|$ matrix corresponding to the columns of Φ indexed by Γ, where $|\Gamma|$ is the cardinality of the index set. We are interested in finding x given y and Φ. Formally, we wish to solve the non-convex optimization problem

$$\widehat{x} = \underset{x\in\mathbb{R}^N}{\operatorname{argmin}} \|x\|_0 \quad \text{s.t.} \quad \Phi x = y. \qquad (P_0)$$

Although this problem is NP-hard, it is possible under appropriate conditions (which depend on the particular values of N, M, K and Φ) to solve it using greedy methods [1].

2.1. Orthogonal Matching Pursuit

One of the classic greedy algorithms for solving this problem is OMP, described in Algorithm 1. OMP is an iterative algorithm that builds an estimate of the support of x by adding one index to this set at a time. The algorithm starts with an empty estimate $\Gamma^{(0)}$. It keeps a residue vector r, initially equal to y, which corresponds to the component of y perpendicular to the column span of Φ_Γ. At each iteration, OMP computes

Algorithm 1 Orthogonal Matching Pursuit

input: Φ, y, stopping criterion
  initialize: $r^{(0)} = y$, $\Gamma^{(0)} = \emptyset$, $\ell = 0$
  while not converged **do**
    match: $h = |\Phi^T r|$
    identify: $\Gamma^{(\ell+1)} = \Gamma^{(\ell)} \cup \operatorname{argmax}_j |h(j)|$
    update: $x^{\ell+1} = \operatorname{argmin}_{z:\ \mathrm{supp}(z)\subseteq\Gamma^{(\ell+1)}} \|y - \Phi z\|_2$
    $r^{\ell+1} = y - \Phi x^{\ell+1}$
    $\ell = \ell + 1$
  end while
output: $\widehat{x} = x^{\ell} = \operatorname{argmin}_{z:\ \mathrm{supp}(z)\subseteq\Gamma^{(\ell)}} \|y - \Phi z\|_2$

the correlation between the current residue and the columns of Φ. The index of the column with the highest correlation is added to the current estimate of the support. Using this new support estimate a new residue is computed. The loop exits when the stopping criterion is met, typically when the norm of the residue is small enough.

We will later exploit the fact that in OMP the norm of the residue vector $r^{(\ell)}$ decays exponentially [3, Sec. 3.1.2]. In particular,

$$\|r^{(\ell)}\|_2^2 \leq (1 - \delta(\Phi))^{\ell} \|y\|_2^2, \tag{1}$$

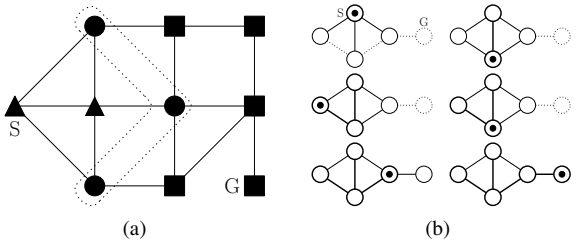
where $\delta(\Phi) \in (0, 1)$ is the universal decay-factor of Φ defined as $\delta(\Phi) = \inf_v \max_{1\leq j\leq M} \frac{|\phi_j^T v|}{\|v\|_2^2}$. Thus, the norm of the residue converges to 0. In fact, it must reach 0 in M or fewer iterations, although a residue of 0 does not necessarily imply that the solution is correct.

2.2. Online search

Search algorithms are methods that solve the problem of finding a minimal cost path between a given pair of start and goal states belonging to a state space [4]. The state space can be described by a graph where nodes represent states and weighted edges represent potential transitions between states.

Search algorithms can be broadly classified into *offline* and *online* algorithms. A way to understand the differences between these two classes is by thinking in terms of an agent that lives in the state space; the agent begins at the initial state and wants to go to the goal state. In the offline approach the agent finds a complete solution to reach the goal in what is called a planning stage, and then executes the corresponding actions. Dijkstra and A* are two classic offline search algorithms. Online approaches, on the other hand, interleave planning and execution of actions.

In offline searching the agent classifies the states into three different classes: *explored states*, *unexplored states*, and the *fringe*. At each iteration, the agent selects one node of the fringe (the criteria used to select the node from the fringe are what differentiate the different offline search algorithms). This node becomes an explored node, and all its successors are added to the fringe. The planning stops when the goal state is removed from the fringe. Figure 1(a) shows an example of a state space during an intermediate step of the planning.



(a) (b)

Fig. 1: Examples of searching in a state space. S and G are the start and goal state, respectively. (a) Offline search during an intermediate stage of execution: explored, unexplored, and fringe states are represented by triangles, squares, and circles, respectively. (b) Evolution of online search. Execution depicted from left to right. A dot indicates the current state. Bold circles and edges represent visited states and transitions, respectively. Dotted lines represent unexplored regions.

As explained above, offline search algorithms must keep a list of the fringe states. When the branching factor (the number of successors for each state) is too large, this approach is unfeasible; colloquially, it said that "you run out of space before you run out of time". Online search algorithms are able to overcome situations like this one since they have memory requirements that do not depend on the number of states or the branching factor.

In online search [5] an agent starts by setting its current state s equal to the start state. Then it performs a local search among its neighbors and it moves to the state that looks most promising. It repeats these two steps, planning and execution, until the current state is equal to the goal state. The agent uses a *value-function* $u(s)$ to store its knowledge about the state space. This function represents the current estimate of the distance to the goal for each state. The first time the value-function is evaluated it is set using a *heuristic function* $h(s)$ that returns an initial estimate of the distance to the goal. When the agent moves from the state s to the state s' it updates $u(s)$ using the value of $u(s')$. Note that the heuristic function is fixed, while the value-function changes as the agent learns about the structure of the state space. Figure 1(b) shows an instance of online search for a simple state space with five states.

3. ONLINE SEARCH OMP

Inspired by OMP and search methods in state spaces, we propose a new algorithm to solve (P_0). One way to think about OMP is that it searches for a support that is able to "explain" the observation vector y. This search proceeds by adding one element to the support at a time, and it is not possible to remove an element once it is added. In other words, OMP does not have the ability to backtrack. On the other hand, online search methods are backtracking algorithms that provide effective mechanisms to search for a solution in a state space. Our algorithm, dubbed "Online Search OMP" (OS-OMP), merges the above mentioned approaches. It combines the greedy addition of indices to the support based on the

value of the residue used in OMP, with the use of a value-function to represent the accumulated knowledge.[1]

We adopt the "agent perspective", typically used in Artificial Intelligence [4], to explain OS-OMP. Consider an agent whose state is an index set Γ. The agent can move to any state corresponding to an index set with one extra element. It can also move to its predecessor state (it may be useful to consider this as an "undo" movement). Thus, the successor function can be written as

$$\text{succs}(\Gamma) = \{\Gamma' \mid \Gamma' \supset \Gamma, |\Gamma'| = |\Gamma|+1\} \cup predecessors(\Gamma),$$

where $predecessors$ is a table that keeps track of the state transitions.

Under this setup, solving (P_0) corresponds to finding, starting at the state $\Gamma = \emptyset$, the Γ with residue 0 and smallest cardinality. OS-OMP is specified entirely in Algorithm 2. It starts with an empty table to store the value function u and another empty table to store the state predecessors. The current support is set equal to the empty set. At each iteration it checks to see whether the current Γ exists in u. If it does not, this Γ is added to the table with a value equal to the heuristic function $h(\Gamma)$ (lines 3 to 5). As the heuristic function we use the norm of the residue corresponding to that support:

$$h(\Gamma) = \min_{z:\ \text{supp}(z) \subseteq \Gamma} \|y - \Phi z\|_2. \tag{2}$$

Note that this is the norm of the residue vector r computed in the *update* step of OMP. Also note that since this function is computed only when a support set does not have an entry in the value-function table, the computational cost of using the norm of the residue as a heuristic is reasonable. Then, OS-OMP computes and stores in the table u_{succs} the value-functions for all the successors of Γ (lines 7 to 12). As before, if there is no entry for a given successor, the value-function table is initialized using the heuristic function h.

The next steps in the algorithm are based on the following observation. As stated by (1), when the new elements added to the support are selected greedily, the norm of the residue decays exponentially. This implies that when the support is not sparse enough, i.e., the estimate is the wrong one, the reduction in the norm of the residue is very small. This behavior of the norm of the residue helps us to identify two search regimes: one during which the norm of the residue decays quickly, and one during which there is little change of the norm of the residue. We consider this last situation as an indication that the current Γ is in the wrong region of the state space and that the algorithm should start backtracking. OS-OMP computes the difference between the maximum and the minimum of the value-function evaluated at the successors of Γ, and compares this difference with a threshold η. If the difference is above this threshold, Γ_{new} is set equal to the $\Gamma' \in \text{succs}(\Gamma)$ with the smallest $u(\Gamma')$; otherwise, Γ_{new} is set equal to the support in the successors with the smallest cardinality (lines 13 to 17).

[1]Note that although methods such CoSaMP [6] and IHT [7] are also able to remove items from the support, they do this by estimating the complete support at each iteration, rather than by adding one element at a time.

Algorithm 2 Online Search OMP

input: Φ, y, $\eta > 0$, stopping criterion
1: **initialize:** $\Gamma = \emptyset$, u: empty table, $predecessors$: empty table
2: **while** not converged **do**
3: **if** $\Gamma \notin u$ **then**
4: $u(\Gamma) = h(\Gamma)$
5: **end if**
6: u_{succs}: empty table
7: **for** $\Gamma' \in succs(\Gamma)$ **do**
8: **if** $\Gamma' \notin u$ **then**
9: $u(\Gamma') = h(\Gamma')$
10: **end if**
11: $u_{succs}(\Gamma') = u(\Gamma')$
12: **end for**
13: **if** $\frac{\max(u_{succs}) - \min(u_{succs})}{\|y\|} > \eta$ **then**
14: $\Gamma_{new} = \text{argmin}(u_{succs})$
15: **else**
16: $\Gamma_{new} = \text{argmin}(\{|\Gamma'| \mid \Gamma' \in succs(\Gamma)\})$
17: **end if**
18: **if** $|\Gamma_{new}| < |\Gamma|$ **then**
19: $u(\Gamma) = +\infty$
20: **end if**
21: **if** $\Gamma_{new} \notin predecessors$ **then**
22: $predecessors(\Gamma_{new}) = \Gamma$
23: **end if**
24: $\Gamma = \Gamma_{new}$
25: **end while**
output: $\widehat{x} = \text{argmin}_{z:\ \text{supp}(z) \subseteq \Gamma} \|y - \Phi z\|_2$

After OS-OMP selects Γ_{new}, it checks to see if this new support is a backtrack move. If that is the case, the value-function of the current Γ is updated to $+\infty$. This guarantees that the path corresponding to this Γ will not be expanded in future iterations (lines 18 to 20). Finally, Γ_{new} is added to the table of predecessors if necessary (lines 21 to 23), and Γ is updated to its new value.

We continue with an example where OS-OMP works successfully but OMP fails. We set the length and sparsity of x to $N = 128$ and $K = 5$, respectively, and the length of y to $M = 19$. The matrix Φ has i.i.d. entries drawn from a standard normal distribution. We compare the evolution of the norm of the residue between OS-OMP and OMP in Fig. 2. Note that, as explained in Sec. 2.1, finding a solution with a residue of norm 0 does not guarantee successful recovery of x. We observe the predicted exponential decay for OMP. For OS-OMP, on the other hand, the algorithm is able to detect that it is going in the wrong direction and backtracks several times until it finds the correct solution.

4. EXPERIMENTAL RESULTS

In this section we empirically evaluate OS-OMP.[2] We also compare with OMP and A*OMP. For all experiments that follow we generate, for each value of the sparsity level K, signals of length $N = 128$ having K non-zero coefficients at locations selected uniformly at random. We fix the value of M and plot the rate of perfect recovery (declared when

[2]Code available at https://github.com/aweinstein/osomp.

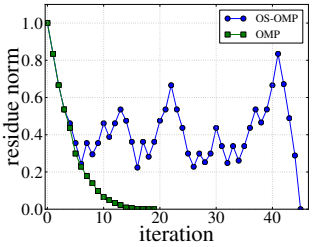


Fig. 2: Comparison of the norm of the residue of OMP and OS-OMP for an instance with $N = 128$, $M = 19$ and $K = 5$. In this example OMP (green squares) fails to recover x, while OS-OMP (blue circles) successes.

$\|x - \widehat{x}\| \leq 10^{-4}$ and estimated using 500 trials) as a function of the sparsity level K. For each trial the matrix Φ has i.i.d. entries drawn from a standard normal distribution. Since greedy methods are sensitive to the distribution of the non-zero signal coefficients, we consider three cases, each one with a different distribution.

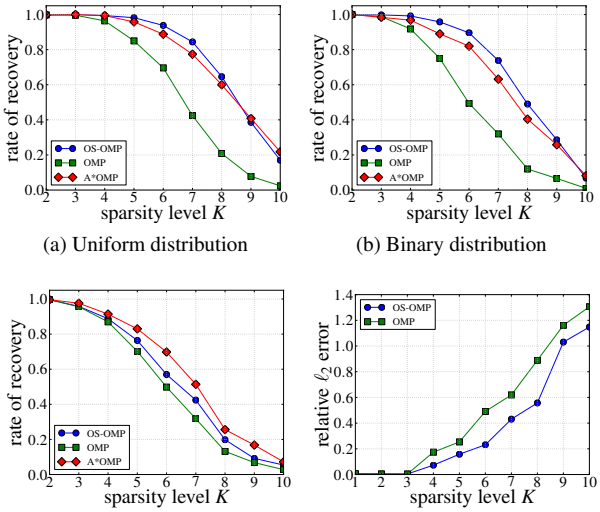
In the first experiment we select the amplitude of the non-zero signal coefficients uniformly at random from the interval $[-2, -1] \cup [1, 2]$ and fix the dimension of y to $M = 30$. Figure 3(a) shows the results. Both OS-OMP and A*OMP perform significantly better than OMP, with OS-OMP performing slightly better than A*OMP for most values of K.

In the second experiment we fix the magnitudes of the non-zero signal coefficients to 1 and set their signs uniformly at random. We fix the dimension of y to $M = 30$. Figure 3(b) shows the results. As in the previous experiment, both OS-OMP and A*OMP perform significantly better than OMP. This time the improvement of OS-OMP over A*OMP is more significant.

In the third experiment the amplitude of the non-zero signal coefficients are i.i.d. drawn at random from a standard normal distribution. We fix the dimension of y to $M = 25$. Figure 3(c) shows the results. This time the difference between the three methods is less significant. OMP is still the method with the poorest performance. This time A*OMP is slightly better than OS-OMP.

Finally, we test OS-OMP in a scenario where the observations are corrupted by additive noise. We set $y = \Phi x + e$, where e is a vector with i.i.d. entries drawn from a zero-mean normal distribution with standard deviation set to $\sigma = 0.1$. To handle this situation, we only need to modify the algorithm to stop when the norm of the residual is smaller than the noise level. We select the amplitude of the non-zero signal coefficients uniformly at random from the interval $[-2, -1] \cup [1, 2]$ and fix the dimension of y to $M = 30$. Figure 3(d) shows the results. We observe that OS-OMP performs better than OMP.

Although OS-OMP and A*OMP exhibit similar performance in terms of rate of recovery, we must stress the fact that the execution time of OS-OMP is significantly faster, roughly by a factor of 20. For instance, to recover a signal of length $N = 128$ and sparsity $K = 5$ from a vector y of length $M = 35$ using OS-OMP takes 5 ms, while recovering the same signal using A*OMP takes 140 ms. OMP takes 3 ms to recover the same signal.



(a) Uniform distribution (b) Binary distribution

(c) Normal distribution (d) Noisy observations

Fig. 3: Experimental results. (a, b, c) Rate of perfect recovery as a function of the sparsity level K using OS-OMP, A*OMP, and OMP for three different distributions of the non-zero coefficients. (d) Relative ℓ_2 error for the recovery of x from noisy observations.

5. CONCLUSIONS

We have presented a new method, called OS-OMP, for recovering a sparse vector x. The new algorithm merges ideas from greedy techniques and from online search methods. OS-OMP performs significantly better than OMP without incurring a significant extra computational load. It has a similar performance to A*OMP, a method also inspired by search algorithms, but it has a much faster execution time.

Future work will include theoretical analysis of OS-OMP. We will also study the possibility of adjusting the parameter η automatically.

6. REFERENCES

[1] J. A. Tropp, "Greed is Good: Algorithmic Results for Sparse Approximation," *IEEE Trans. on Info. Theory*, vol. 50, no. 10, pp. 2231–2242, Oct. 2004.

[2] N. B. Karahanoğlu and H. Erdoğan, "A* Orthogonal Matching Pursuit: Best-First Search for Compressed Sensing Signal Recovery," *Submitted to Elsevier Digital Signal Processing*, 2011.

[3] M. Elad, *Sparse and Redundant Representations*, Springer New York, New York, NY, 2010.

[4] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach (3rd Edition)*, Prentice Hall, Englewood Cliffs, 2009.

[5] S. Koenig, "Exploring unknown environments with real-time search or reinforcement learning," in *Advances in Neural Information Processing Systems (NIPS)*, 1999.

[6] D. Needell and J.A. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Applied and Computational Harmonic Analysis*, vol. 26, no. 3, 2009.

[7] T. Blumensath and M.E. Davies, "A simple, efficient and near optimal algorithm for compressed sensing," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2009.