Local orthogonal greedy pursuits for scalable sparse approximation of large signals with shift-invariant dictionaries

Boris Mailhé, Rémi Gribonval, Frédéric Bimbot  
Projet METISS  
Centre de Recherche INRIA Rennes - Bretagne Atlantique  
IRISA, Campus de Beaulieu  
F-35042 Rennes Cedex, France  
E-mail: firstname.lastname@irisa.fr

Pierre Vanderheynden  
Signal Processing Laboratories (LTS)  
School of Engineering, EPFL  
Station 11, CH - 1015 Lausanne, Switzerland  
E-mail: firstname.lastname@epfl.ch

Abstract—We propose a way to increase the speed of greedy pursuit algorithms for scalable sparse signal approximation. It is designed for dictionaries with localized atoms, such as time-frequency dictionaries. When applied to OMP, our modification leads to an approximation as good as OMP while keeping the computation time close to MP. Numerical experiments with a large audio signal show that, compared to OMP and Gradient Pursuit, the proposed algorithm runs in over 500 less time while leaving the approximation error almost unchanged.

Index Terms—sparse approximation, greedy algorithms, local atoms, orthogonal matching pursuit

I. INTRODUCTION

A sparse approximation of a signal s over a dictionary Φ is a linear approximation of S on a few vectors of Φ called atoms. Finding such a good approximation is a key issue in various domains such as compression, under-determined source separation, and more recently compressed sensing. The problem of finding the closest m-term approximant is NP-Complete due to the combinatorial exploration of all subsets of Φ. Many algorithms have been proposed to obtain good sparse approximations in polynomial time, but even polynomial algorithms can prove too expensive for large signal dimensions.

Today’s most popular approaches are ℓ1 minimization, on the one hand, which is tackled with specialised convex optimization iterative techniques, and greedy algorithms, on the other hand, which iteratively decrease the approximation error by relaxing the sparsity constraint. In this paper we focus on the latter class, which includes Matching Pursuit (MP) [1], Orthogonal Matching Pursuit (OMP) [1], [2], as well as several variants such as Gradient Pursuit (GP) [3] or Relaxed Greedy Algorithm [4]. Roughly speaking, MP is fast but can yield substantially poorer approximation performance than OMP and GP, which however typically have substantially larger running times for large data.

In this paper we propose a way to drastically reduce the complexity of greedy algorithms in the special case of localized atoms. We illustrate the properties of the proposed algorithm by comparing it with MP, OMP and GP on a high dimensional audio signal.

II. GREEDY ALGORITHMS

Let Λ be an Hilbert space of finite dimension N. A dictionary Φ is a set of unit norm vectors ϕ_k of Λ called atoms. We will also use the notation Φ for the matrix that admits the atoms ϕ_k as columns. A sparse approximation of a signal s over a dictionary Φ is a vector x with small approximation error ∥s − Φx∥_2 under a constraint on the number of nonzero coefficients ∥x∥_0, usually denoted with the ℓ0 "norm" ∥x∥_0. Finding the best approximation with ∥x∥_0 ≤ K is an NP-hard problem, and greedy algorithms are sub-optimal iterative algorithms that attempt to solve this problem by successively adding new atoms into a sparse approximation Φ_rx_1 with the objective of minimizing the new residual r_i = s − Φ_rx_i. Each iteration i of a greedy algorithm is composed of two successive steps:

1) selection: find the atom that has the highest scalar product with the residual ϕ_i = argmax_{ϕ_k ∈ Φ} (r_{i-1}, ϕ_k) and add it to the selected atoms Φ_i = Φ_{i-1} ⊕ ϕ_i;
2) update the coefficients r_i (and the residual r_i), trying to minimize the new approximation error ∥r_i∥_2.

Several update rules have been proposed, among which
1) MP: r_i = r_{i-1} − (r_{i-1}, ϕ_i)ϕ_i;
2) OMP: r_i = r_{i-1} − Φ_i^T (Φ_i^T Φ_i)^{-1} Φ_i^T r_{i-1};
3) GP: r_i = r_{i-1} − ||Φ_i^T r_{i-1}||_2^{-2} Φ_i^T r_{i-1}.

MP is the fastest of the above described algorithms, because it only attempts to optimize the coefficient of the last selected atom to minimize the new approximation error. OMP optimizes all coefficients to obtain the minimum error with the set of selected atoms. This can provide a much smaller error, but at the price of significantly more computations. GP essentially attempts to reduce the cost of OMP by performing the first step of a conjugate gradient descent to approximate the full projection OMP would perform.

Table I indicates the complexity order of each step for MP, OMP and GP with a general dictionary. The main quantiﬁes driving the complexity are N (the dimension of the signal...
space), $\alpha \geq 1$ (the redundancy of the dictionary that contains $\alpha N$ atoms), and $i$ (the iteration number, which indicates that $i$ atoms have been selected). Since the goal is to obtain a sparse approximation of the signal, the iteration number $i$ is always lower than the signal dimension $N$.

The selection step involves two substeps: the computation of the $\alpha N$ correlations $\langle r_{i-1}, \varphi_i \rangle$, $\varphi_i \in \Phi$, each of them costing of the order of $N$ multiply-add; the search for the atom with maximum correlation, which requires $\alpha N - 1$ comparisons. The update step for OMP involves computing the Gram matrix $G_i = \Phi_i^T \Phi_i$, which can be updated from the previously computed $\Phi_{i-1}^T \Phi_{i-1}$ but requires the computation of the scalar product of the last selected atom $\varphi_i$ with the $i-1$ previous ones $\Phi_{i-1}$. Then, the new coefficients $x_i$ in OMP can be computed by inverting the Gram matrix with a cost roughly quadratic in the size $i$ of the matrix by reusing the computations done in previous iterations. The exact cost will depend on the chosen inversion method. A more complete study about it, as well as the explanations for GP complexity, can be found in [3]. Eventually, all methods require updating the residual, which involves an $N \times i$ matrix multiply $\Phi_i x_i$ and/or updating the $N$ entries of the vector.

### III. LOCAL DICTIONARIES

Dictionaries are said to be local if the length $L$ of the convex hull of the support of the atoms is much smaller than the signal length: $L \ll N$. Common shift-invariant dictionaries such as Gabor or wavelets dictionaries are local. This structure can be exploited to substantially decrease the algorithms’ complexity through the following tricks:

- the cost for computing a scalar product drops from $N$ to $L$, making the projection step of OMP and GP cheaper.
- with the MP update, between two consecutive iterations, the residual only changes on the support $\text{support}(\varphi_i) := \{t, \varphi_i(t) \neq 0\}$ of the last selected atom. This leaves only about $\alpha L$ correlations to recompute instead of $\alpha N$, since all the other ones are still relevant. This drives down the cost to $\alpha L \log L$.
- previous comparisons between the correlations can also be stored in a tournament tree to make the search of the maximum faster.
- the complexities for the manipulation of $G_i$ are also quite pessimistic as it should be sparse: if 2 atoms have disjoint supports, then their scalar product is null. Fast preselection of the subset of atoms that can possibly be correlated with the last one can be performed within logarithmic complexity if the atoms are sorted by increasing support location.

These tricks are implemented in the Matching Pursuit ToolKit (MPTK\(^1\)) C++ library [5] and enable a speedup of up to a 100 times in decomposition, allowing $I = 1.5 \times 10^6$ iterations of MP on a twenty-minute 44KHz audio signal ($N = 60 \times 10^6$ samples) to be performed in less than twenty minutes of computation time on a standard PC, with an $\alpha = 3$ times overcomplete dictionary of Gabor atoms of length $L = 1024$.

The key property that makes such speedup possible is that the residual update is also local: the residual outside the support of the last selected atom is the same before and after the update. This locality property could certainly be used to speed up the first iterations of OMP or GP, however in OMP/GP the updated support size will grow with the number of iterations until the whole signal support is spanned by selected atoms.

Table II summarizes the complexity order of each step (except the first one which involves computing all correlations) of MP, OMP and GP with a general local dictionary.

### IV. LOCOMP ALGORITHM

#### A. Principle

As described above, in local dictionaries, simple tricks allow to significantly reduce the computational complexity of MP compared to a naive implementations. However, the cost of OMP and GP remains quite high, calling for modified algorithms to handle real-world large-scale signals, where the aimed number of atoms $I$ is somewhat lower than the signal size $N$, but the latter is large enough to discourage naive computation (e.g. for one minute of music sampled at 8 kHz, we already have $N \approx 5 \times 10^5$).

The prohibitive costs for OMP and GP are the ones with strongest dependency in $N$: as shown in Table II the most

---

\(^1\)http://mptk.irisa.fr/
costly steps are the correlation computation and maximum search, which have linear dependency in \( N \). This linear dependency has disappeared in MP by exploiting the locality of the changes in the residual. MP is scalable to large signals because the cost of an iteration depends on \( L \), not on \( N \). This is why we propose an algorithm that only slightly loosens this locality property. To our knowledge, all approaches to decrease OMP complexity emphasize the reduction in the cost of the update step (e.g., by replacing full matrix inversion by conjugate gradient descent as in [3]), not the selection step, so they provide little improvement for local dictionaries.

The main idea of the proposed LocOMP algorithm is to select a sub-dictionary \( \Psi_i \subset \Phi_i \) containing the last selected atom \( \varphi_i \) and to orthogonalize the decomposition only on this sub-dictionary. The simplified algorithm is described in Algorithm 1.

**Algorithm 1** \( x = \text{LocOMP}(s, \Phi) \)

1. \( r_0 = s \)
2. \( \Phi_0 = 0 \)
3. \( x_0 = 0 \)
4. for \( i = 1 \) to \( I \) do
   5. \( \varphi_i = \arg\max_{\varphi \in \Phi} |\langle r_{i-1}, \varphi \rangle| \) [selection]
   6. \( \Phi_i = \Phi_{i-1} \cup \varphi_i \)
   7. \( \Psi_i = \text{neighbour}(\Phi_i, \varphi_i) \) [sub-dictionary selection]
   8. \( \chi_i = (\Psi_i^\top \Psi_i)^{-1} \Psi_i^\top r_{i-1} \) [coefficients of projection on sub-dictionary]
   9. \( x_i = x_{i-1} + \chi_i \) [update coefficients]
10. \( r_i = r_{i-1} - \Psi_i \chi_i \) [update residual]
5. end for
6. return \( x_I \)

**B. Neighbourhood selection**

The key element that determines the behaviour of the algorithm is the \( \text{neighbour()} \) function that performs the sub-dictionary selection:

- MP corresponds to \( \text{neighbour}(\Phi_i, \varphi_i) := \varphi_i \);
- OMP corresponds to \( \text{neighbour}(\Phi_i, \varphi_i) := \Phi_i \);

As seen in Section III, the complexity of the algorithm is driven by the length of the updated residual. If this update interval is fixed, then the best sub-dictionary possible is always the one that all atoms whose support is included in that interval. The sub-dictionary selection problem can then be reduced to the choice of the interval.

In LocOMP, \( \text{neighbour}(\Phi_i, \varphi_i) \) is centered on \( \varphi_i \) with length \( 3L - 2 \): \( \text{neighbour}(\Phi_i, \varphi_i) \) it contains all the atoms \( \varphi \in \Phi_i \) which support intersects with the support of \( \varphi_i \). This choice was mainly led by the observation that, as explained in Section III, this set is already the one that has to be searched for when updating the Gram matrix. Selecting it as the atom’s neighbourhood spares another search.

V. EXPERIMENTAL RESULTS

LocOMP has been tested and compared to MP, OMP and GP on an excerpt from the RWC base\(^2\). It is a one-minute mono-channel jazz guitar audio signal downsampled to 8kHz (\( N \approx 5 \cdot 10^3 \)). Given the high cost of running OMP and GP for comparison (the total running time for each of these algorithms in the first experiment below was roughly 5 \( \cdot \) 10\(^5\) seconds, \( \approx \) 5.7 days), it was not possible to run experiments on more than one signal, and this was also the largest signal dimension we could test. In comparison, the computation time of LocOMP was 854 seconds (\( \approx \) 15 minutes).

A. SNR and computation time

In a first experiment, OMP, GP, LocOMP and MP were run for \( I = 20000 \) iterations\(^3\) to decompose the signal on a fully shift-invariant MDCT dictionary of scale \( L = 32 \) (therefore with redundancy factor \( \alpha = 32 \)) containing \( \alpha N \approx 1.5 \cdot 10^7 \) atoms. The scale roughly corresponds to the smallest scale of the windows used in AAC encoding on 44.1 kHz signals. That is a quite poor dictionary, but we could not afford to actually run OMP and GP on larger ones.

We used the approximation SNR as a measure of the quality of the approximation. Figure 1 shows the SNR reached by each algorithm at each iteration. OMP, GP and LocOMP cannot be distinguished on this plot. The final SNR for LocOMP after 20000 iterations is actually only 0.01dB lower than for OMP and GP, while the final SNR for MP is 0.6dB lower.

The CPU times per iteration evolved linearly for each algorithm. Table III shows their value for the first iteration (which is relatively costly for every algorithm because it involves computing inner products with all atoms of the dictionary), the next beginning iterations, the last iterations and finally the total duration of the complete execution.

---

\(^2\)http://staff.aist.go.jp/m.goto/RWC-MDB/

\(^3\)The iterations of the different algorithms were interleaved on the same process to ensure a similar environment for all. The Matlab\textsuperscript{®} code was compiled and run on a standard PC (2.33 Ghz, 4 Go RAM). It was not fully optimized for runtime speed, especially for the first OMP and GP iterations, but this should not affect the observed time magnitudes.
B. Preliminary application to audio coding

In a second experiment, we investigated the potential use of LocOMP in the scalable coding framework proposed by Ravelli and Daudet [6]. The 8 kHz signal was decomposed on a two-scale fully shift-invariant MDCT dictionary with scales \( L_1 = 32 \) and \( L_2 = 256 \), roughly corresponding at 8kHz to the scales used in AAC encoding at 44.1kHz.

Figure 2 shows the rate/distortion curve of this coding scheme using MP and LocOMP as a transform. At high rates, LoCOMP coding leads to less distortion than MP coding, with a final gain of 1.4dB. However, LoCOMP also brings a degradation at lower rates. This might be partly due to the choice of a much smaller dictionary than the eight-scale dictionary used in [6].

VI. CONCLUSION

We proposed a greedy algorithm called LocOMP for sparse approximation of long signals with large local dictionaries. This algorithm shares the same tractability properties to long signals as MP. It showed the same empirical approximation quality as OMP/GP, with a gain of 0.6 dB over MP, while the computational cost remains 500 times lower than that of OMP. We expect the approximation gain of LocOMP over MP to be more significant for dictionaries more adapted to the decomposed signal (e.g., \( L \) rather of the order of 256, the largest scale used in AAC codecs), however for such scales it no longer seems possible to compare the proposed algorithm with OMP/GP, because of the computational complexity of the latter.

A localized version of Gradient Pursuit is under implementation in MPTK [5] to benefit from other speedups not described here. As the signal is projected on a low-dimension sub-dictionary in LocOMP, it is not clear whether replacing LocOMP by LocGP would change much, but it leads to much lighter code. We believe this implementation will open the door to large scale experiments and applications of sparse approximation that so far seemed unachievable. A prototype is already running, but it is still far from reaching the same code optimization as MP.

VII. ACKNOWLEDGEMENTS

The authors would like to thank Emmanuel Ravelli and Laurent Daudet from the LAM team at University Paris 6 for their help with the audio coding experiments.

REFERENCES


