MPTK: Matching Pursuit made Tractable

Sacha Krstulovic, Rémi Gribonval

MPTK: MATCHING PURSUIT MADE TRACTABLE

Sacha Krstulović and Rémi Gribonval

IRISA-INRIA, Campus de Beaulieu, 35042 Rennes cedex, France – {sacha,remi}@irisa.fr

ABSTRACT

Matching Pursuit (MP) aims at finding sparse decompositions of signals over redundant bases of elementary waveforms. Traditionally, MP has been considered too slow an algorithm to be applied to real-life problems with high-dimensional signals. Indeed, in terms of floating points operations, its typical numerical implementations have a complexity of $\mathcal{O}(N^2)$ and are associated with impractical runtimes. In this paper, we propose a new architecture which exploits the structure shared by many redundant MP dictionaries, and thus decreases its complexity to $\mathcal{O}(N \log N)$. This architecture is implemented in a new software toolkit, called MPTK (the Matching Pursuit Toolkit), which is able to reach, e.g., $0.25\times$real time for a typical MP analysis scenario applied to a 1 hour long audio track. This substantial acceleration makes it possible, from now on, to explore and apply MP in the framework of real-life, high-dimensional data processing problems.

1. INTRODUCTION

The Matching Pursuit (MP) algorithm [1] aims at finding sparse decompositions of signals over redundant bases of elementary waveforms. In particular, it is able to extract high-level signal features, such as harmonic components occurring at various time scales [2], in a direct parametric fashion. Even though MP can be much faster than other global sparse decomposition techniques [3, 4], it has long been considered too slow to be applied to real-life problems. Despite many tricks to accelerate the original iterative algorithm [1] or approximate "fast" variants thereof [5, 6], each step typically costs at least $\mathcal{O}(N)$, N being the number of signal samples. Since the number of steps needed to reach a reasonable reconstruction accuracy is often of the order of a fraction of N, the total complexity usually amounts to at least $\mathcal{O}(N^2)$, and is associated with prohibitive runtimes.

However, we have recently elaborated a global software architecture for MP computation which, by associating algorithmic enhancements with a careful optimization of the code, and helped by the latest increases in CPU clock rates, reaches tractable computation times. For example, running 1.5 million iterations of MP to decompose a one hour long audio signal into Gabor atoms with three different scales can be performed in 15 minutes ($0.25\times$real time) on a Pentium IV@2.4GHz. The acceleration obtained with our program is essentially due to the fact that we were able to decrease the cost of each MP iteration from $\mathcal{O}(N)$ to $\mathcal{O}(\log N)$. Previous implementations of MP (such as LastWave [7] or Atomizer [8]) were not able to handle such large signals, with so many iterations, in a reasonable amount of time and/or with a manageable memory footprint.

The goal of this article is to explain in detail the main ideas and the software architecture leading to such a fast implementation, so that researchers interested in Matching Pursuit and its variants can build MP-based experiments or MP-based applications that run in a tractable time. Our software, called the Matching Pursuit ToolKit (MPTK), performs standard monochannel as well as multichannel Matching Pursuit decompositions [9, 10] of time signals. Its current version implements Gabor dictionaries [1], Harmonic dictionaries [5] as well as Dirac dictionaries.

The rest of the paper is organized as follows. Section 2 describes the Matching Pursuit algorithm, its main computational bottlenecks and some state of the art solutions to reduce its complexity. Section 3 describes our main contribution to a substantial acceleration of the MP algorithm. Section 4 gives some experimental results that illustrate the corresponding gain in speed, while section 5 discusses some practical aspects of the software architecture of MPTK.

2. THE MATCHING PURSUIT ALGORITHM

Matching Pursuit is part of a class of signal analysis algorithms known as Atomic Decompositions. These algorithms consider a signal $\mathbf{x}$ as a linear combination of known elementary pieces of signal $\mathbf{w}_m$, called atoms, chosen within a dictionary $\mathcal{D}$:

$$\mathbf{x} = \sum_{m=1}^{M} \alpha_m \mathbf{w}_m \qquad \text{where } \mathbf{w}_m \in \mathcal{D}. \tag{1}$$

Usually, the dictionary $\mathcal{D}$ is overcomplete: in dimension N, this means that $\mathcal{D}$ has more than N elements and spans the entire space. In this case, the above decomposition is not unique – there may even be an infinite number of solutions. Among all possible decompositions, the preferred ones are the compact (or "sparse") ones, which means that only the first few atoms in Eq. (1), sorted by decreasing weight α_m, are needed to obtain a good approximation of the signal. In general, the bigger the dictionary, the greater the number of potential solutions, and thus the better the chance of finding a more compact signal approximation. However, for general overcomplete dictionaries, finding the globally optimal decomposition according to some pre-determined optimality and compactness criteria is a nontrivial task.

The Matching Pursuit algorithm, originally introduced in [1], is an iterative method which tackles the problem by operating a local optimization, as opposed to global optimization techniques [11, 3, 4] related to heavier computational costs. Below, we recall the principle of the MP algorithm in detail, and we discuss its main computational bottlenecks together with existing state of the art accelerations.

2.1. Principle of the MP algorithm

At each iteration m, the MP algorithm looks for the atom $\hat{\mathbf{w}}_m$ which is the most strongly correlated with the signal $\mathbf{x}$, i.e. which has the highest absolute inner product with the signal. It decomposes along the following steps:

1. initialization: $m = 0$, $\mathbf{x}_m = \mathbf{x}_0 = \mathbf{x}$;

2. computation of the correlations between the signal $\mathbf{x}_m$ and every atom in $\mathcal{D}$, using inner products :

$$\forall \mathbf{w} \in \mathcal{D} : \quad \text{CORR}(\mathbf{x}_m, \mathbf{w}) = |\langle \mathbf{x}_m, \mathbf{w} \rangle| \tag{2}$$

3. search of the most correlated atom, by searching for the maximum inner product:

$$\hat{\mathbf{w}}_m = \arg\max_{\mathbf{w} \in \mathcal{D}} \text{CORR}(\mathbf{x}_m, \mathbf{w}) \tag{3}$$

4. subtraction of the corresponding weighted atom $\alpha_m \hat{\mathbf{w}}_m$ from the signal $\mathbf{x}_m$:

$$\mathbf{x}_{m+1} = \mathbf{x}_m - \alpha_m \hat{\mathbf{w}}_m \tag{4}$$

where $\alpha_m = \langle \mathbf{x}_m, \hat{\mathbf{w}}_m \rangle$;

5. If the desired level of accuracy is reached, in terms of the number of extracted atoms or in terms of the energy ratio between the original signal and the current residual $\mathbf{x}_{m+1}$, stop; otherwise, re-iterate the pursuit over the residual: $m \leftarrow m+1$ and go to step 2.

2.2. Bottlenecks of MP

For completely unstructured dictionaries in dimension N, each inner product $\langle \mathbf{x}_m, \mathbf{w} \rangle$ requires N multiplies and $N-1$ adds. Each iteration requires $\sharp\mathcal{D} \geq N$ ($\sharp\mathcal{D}$ being the cardinality of the dictionary) such inner product computations. Therefore, the cost of computing the inner products could be as high as $\mathcal{O}(N^2)$ at each iteration, making MP completely intractable on high-dimensional signals. Fortunately, many signal dictionaries have some structure which can help make this computation much more efficient.

A first obvious trick is to compute groups of inner products at once using fast transforms such as the FFT [1] , the DCT/DST [11], the Fast Wavelet Transform [12], or efficient filterbanks [13]. Typically, this reduces the cost of step 2 to $\mathcal{O}(\sharp\mathcal{D} \log L)$, where L is the size of the performed FFT (or DCT, DST, etc.). A second trick [1] is to update, at each step, only the inner products that have changed, by observing that $\langle \mathbf{x}_{m+1}, \mathbf{w} \rangle = \langle \mathbf{x}_m, \mathbf{w} \rangle$ for every atom $\mathbf{w}$ with $\langle \mathbf{w}_m, \mathbf{w} \rangle = 0$. When possible, fast analytic computation or preliminary storage of all cross-correlations $\langle \mathbf{w}', \mathbf{w} \rangle$ can decrease the cost of step 2 to at most $\mathcal{O}(\sharp\mathcal{D})$. In some cases, however, it can prove more efficient to wholly recompute the necessary inner products than to use a memory-intensive storage-based approach [13].

Performing enough MP iterations to get an small reconstruction error $\|\mathbf{x}_m\|_2$ often means iterating M times, where M is a constant fraction of the dimension N. At each iteration, step 3 is usually performed by scanning all the stored correlations, yielding a scanning cost of $\mathcal{O}(\sharp\mathcal{D})$ per iteration. For high-dimensional signals, the overall computational cost $\mathcal{O}(M\sharp\mathcal{D})$ of classical MP implementations therefore exceeds $\mathcal{O}(N^2)$, and thus becomes intractable. Popular belief is that MP itself is too computationally intensive to be run in a reasonable computation time on high-dimensional signals. In the next section, we propose a fast search technique which exploits the frame-based nature of many signal dictionaries to decrease the cost of step 3 from $\mathcal{O}(\sharp\mathcal{D})$ to $\mathcal{O}(\log N)$, leading to an implementation of MP with a global computational cost of the order of $\mathcal{O}(N \log N)$.

3. NEW ALGORITHMIC ACCELERATIONS

So far, most efforts to accelerate MP have been on the efficient computation of inner products (step 2). The main idea behind the proposed acceleration is the observation that thanks to previous efforts, the (significantly) most costly part of MP for high-dimensional signals is the search step (step 3). To accelerate it, we mimic the strategy used to reduce the cost of computing inner products by avoiding computing twice the same number. We exploit a typical structure shared by many signal dictionaries.

For example, the multiscale time-frequency Gabor dictionary [1] is a collection $\mathcal{D} = \cup_{j=1}^{J} \mathcal{D}_j$ of "blocks" $\mathcal{D}_j$ of time-frequency atoms at different scales, similar to as many Short Time Fourier Transforms. Each block contains time-frequency atoms obtained by shifting a window $w_j(t)$ of *window length* L_j with a *window shift* $T_j \leq L_j/2$ and modulating them. The waveform of the Gabor atom at scale L_j, time location nT_j and frequency k/K_j ($K_j \geq L_j$ is the *FFT size*) is $w_{j,n,k}(t) := w_j(t - nT_j)\exp(2i\pi kt/K_j)$. Gabor atoms at a given scale L_j and time nT_j and any of the possible K_j different frequencies all share the same temporal support $[nT_j, nT_j + L_j)$ which corresponds to one "time frame" of the signal.

Many other redundant signal dictionaries used for the analysis of large signals have a global structure similar to the multiscale Gabor dictionary: they are the union of groups $\mathcal{D}_n$ of atoms that have their temporal support in a common interval I_n, with the property that:

(a) each interval I_n intersects at most P other intervals.

(b) each interval is of length at most L, where L;

(c) each group contains K_n atoms where K_n;

(d) the dictionary size grows at most linearly with the size N of the analyzed signal.

The very crucial point is that even when the size of the analyzed signal grows arbitrarily, the numbers P, L, K_n above remain bounded. We have seen that scanning at each step *all* atoms of a dictionary to locate the maximally correlated one at each step is quite inefficient. For dictionaries with such a structure, a much more efficient strategy consist in updating and keeping track of the best atom of each group $\mathcal{D}_n$, which can be performed at a severly reduced cost.

At first, the correlation with the best atom of each group (think of the best frequency for a given time-frame at a given scale) is computed

$$\text{CORR}(\mathbf{x}_m, \mathcal{D}_n) := \max_{\mathbf{w} \in \mathcal{D}_n} |\langle \mathbf{x}_m, \mathbf{w} \rangle|$$

and we keep track of which atom is the best within a group

$$\hat{\mathbf{w}}(\mathbf{x}_m, \mathcal{D}_n) := \arg\max_{\mathbf{w} \in \mathcal{D}_n} |\langle \mathbf{x}_m, \mathbf{w} \rangle|$$

Finding the globally best atom is a matter of scanning groups, of which there is often far less than atoms, since $\mathbf{w}_{m+1} = \hat{\mathbf{w}}(\mathbf{x}_m, \mathcal{D}_{n^\star})$ with

$$n^\star := \arg\max_n \text{CORR}(\mathbf{x}_m, \mathcal{D}_n)$$

Typically, if all groups contain of the order of K atoms, scanning all groups to find the best atom costs $\sharp\mathcal{D}/K$ instead of $\sharp\mathcal{D}$ floating point comparisons. This would not bring any speed improvement if we still needed to update $\text{CORR}(\mathbf{x}_m, \mathcal{D}_n)$ for each group at each step. Thanks to the structure of the dictionary, very few of these values need to be updated at each step, and updating them has a negligible cost.

Indeed, after each iteration of MP, we have $\langle \mathbf{x}_{m+1}, \mathbf{w} \rangle = \langle \mathbf{x}_m, \mathbf{w} \rangle$ for nearly all atoms $\mathbf{w}$, except those belonging to at most P groups which atoms have a time supports that (might) intersect the support of the removed atom $\mathbf{w}_m$. Therefore $\text{CORR}(\mathbf{x}_{m+1}, \mathcal{D}_n) = \text{CORR}(\mathbf{x}_m, \mathcal{D}_n)$ is unchanged for nearly all groups (and the best

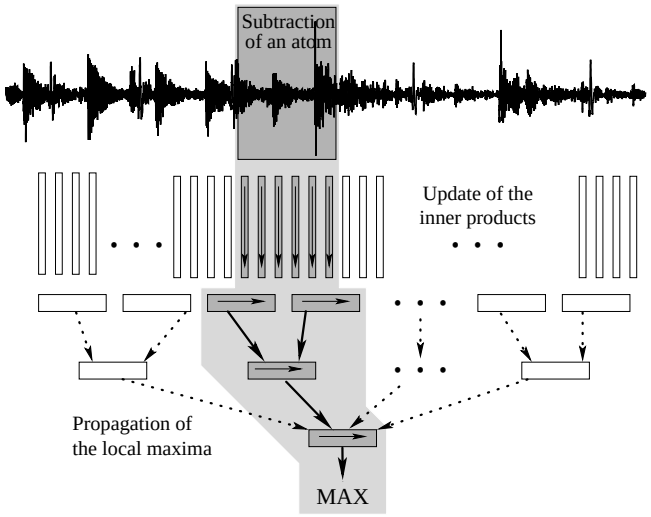

Fig. 1. Update of the inner products and maximum within a block: only the parts concerned with the subtraction of the previous atom (shaded parts) are updated.

atom of those groups is unchanged too). For at most P groups, CORR$(\mathbf{x}_m, \mathcal{D}_n)$ and $\widehat{\mathbf{w}}(\mathbf{x}_m, \mathcal{D}_n)$ must be updated by computing K_n inner products between signals of size at most L, at an update cost not exceeding PKL. Since PKL does not grow with N search time (and memory consumption through the storage of CORR$(\mathbf{x}_m, \mathcal{D}_n)$ and the index of the best atom in each group) for large N are essentially divided by a factor K. For Gabor multiscale dictionaries, the gain factor K is the average FFT size which is often as high as one thousand.

Even with such an acceleration, the complexity of each MP iteration still grows linearly with N: approximately $\sharp\mathcal{D}/K$ groups $\mathcal{D}_n$ are scanned at each step to find the best group, before finding the best atom in the group. The second step to make MP tractable is to use a tree structure to implement this update in $\mathcal{O}(\log \sharp\mathcal{D}/K) = \mathcal{O}(\log N)$.

As illustrated on Figure 1, after updating the P changed values CORR$(\mathbf{x}_m, \mathcal{D}_n)$, we iteratively propagate on a tree the $P/2$, $P/4$, etc., values (and corresponding atom indexes) of the maximum over clusters of 2, 4, etc., nearby groups. At each iteration of MP, few time frames are updated at the leafs of the tree (see Figure 1), and we can propagate the updated values towards the root of the tree by scanning and comparing $\mathcal{O}(\log \sharp\mathcal{D}/K)$ values. Scanning the tree backwards to locate the maximally correlated atom is just as fast.

Figure 1 illustrates the global proposed architecture within each block of the dictionary. To avoid duplicating inner product computations at successive iterations, inner products are updated only for atoms which support intersects portions of the signal which have been modified by the removal the last selected atom. Similarly, the maximum correlation is searched locally over the updated inner products and is propagated across a tree structure which minimizes the amount of memory access. Whereas a plain linear search, browsing all of the inner products, would cost $\mathcal{O}(N)$, it is reduced to $\mathcal{O}(\log N)$ dereferencing operations when using such a tree structure.

4. EXPERIMENTAL RESULTS

To give concrete illustration of the gain of performance that can be expected from our new architecture, we have applied our new implementation to the analysis of a 1 hour long audio signal sampled at 16kHz (about 59 million samples). The analysis was done with 3 Gabor blocks with the following [window length, window shift, fft size] characteristics (in number of samples): [64,32,64], [1024,512,1024] and
[2048,1024,2048]. Given the length of the signal, this amounts to searching a dictionary of 178'773'275 atoms. 1.5 million iterations were performed, capturing about 15dB of the energy of the original signal. The computation was performed on a Pentium IV at 2.4GHz, with enough memory (1GB) to avoid swapping with the hard drive, and under the Linux operating system. With the given parameters, computing, storing and linearly searching all the inner products would take several days and would necessitate significant swapping between the RAM and the hard drive. Restricting the update of the scalar products to the portion of signal where an atom was previously extracted and using the first step of our acceleration approach reduced the computation time to about 20 hours. Using an arborescent search to restrict the number of memory accesses when looking for the maximum inner product reduced the computation time to about 15 minutes ($0.25\times$real time with respect to the original audio signal).

5. THE MPTK IMPLEMENTATION

Older Matching Pursuit implementations, such as the one bundled with LastWave [7], are quite specialized and have become increasingly difficult to maintain. To facilitate our own experiments, we have opted for a complete rewrite of the code which includes the following enhancements: faster code, built-in multichannel signal analysis, flexible specification of the dictionaries (as opposed to hard-coded dictionaries), easy addition of new atom classes, portability across POSIX systems and easy interfacing with any given front-end. The resulting software, written in C++, decomposes into a library, a set of standalone command-line executables and a plain Matlab interface (the interfacing with LastWave has not yet been re-implemented). The complete package has been called MPTK, standing for "the Matching Pursuit ToolKit".

General architecture – The dictionary is implemented as an array of block objects. Each block object knows how to update its inner products along the whole signal or along a particular support, at a particular scale or for a particular transform. This corresponds, e.g., to the application of a Short Time Fourier Transform with a given window length in the case of the Gabor atoms, to a convolution at a particular filter length in the case of atoms based on LPC predictors, or to a wavelet transform restricted to a particular maximum scale and slid along the signal. A block is then able to locate its own maximum inner product, and to emit the associated atom (as a parametric representation or as a waveform). The Matching Pursuit algorithm is implemented according to the iterative cycle depicted in figure 2:

Prospective acceleration factors – The grouping of the inner products in independent blocks, each related to a particular class of atoms, permits the design of **parallel implementations** of the algorithm. An implementation that would take advantage of multi-threading is under study.Besides, access to the FFT is operated through a generic interface which aims at hiding the FFT implementation details and at making it easily interchageable. In particular, some architecture-dependent **performance libraries** deliver FFTs that are optimized

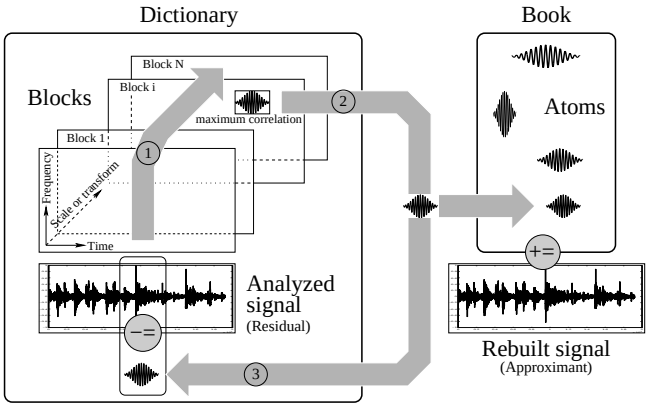

Fig. 2. The general architecture adopted in MPTK. ①: update the inner products and find their maximum; ②: instantiate the corresponding atom; ③: subtract the max atom from the signal; ⇨: re-iterate from the residual. The atoms are stored in a so called "book" and can be summed up to form an approximant.

for a particular processor. A comparative study of the use of the generic FFTW library (from `www.fftw.org`) versus a native Mac OSX FFT is under way.

6. CONCLUSION

The present article describes a new implementation of the Matching Pursuit algorithm which reduces the complexity of the algorithm from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$, and achieves tractable computation times over real-life problems (such as the analysis of audio signals). The corresponding software package, called the Matching Pursuit ToolKit (MPTK), is fast, flexible and open, and it alleviates the burden of rebuilding a complete applicative interface when performing MP-related experiments. This package is distributed [14] under the General Public License, with the hope that it will help researchers and potential users to investigate the properties of the Matching Pursuit algorithm into real-life frameworks.

7. REFERENCES

[1] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3397–3415, 1993.

[2] S. Krstulović, R. Gribonval, P.Leveau, and L. Daudet, "A comparison of two extensions of the matching pursuit algorithm for the harmonic decomposition of sounds," in *Proc.WASPAA'05*, October 2005.

[3] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 33–61, 1999.

[4] I. F. Gorodnitsky and B. D. Rao (1997), "Energy localization in reconstructions using FOCUSS: A recursive weighted norm minimization algorithm," *IEEE Transactions on Signal Processing*, vol. 45, no. 3, 1997.

[5] R. Gribonval and E. Bacry, "Harmonic decompositions of audio signals with matching pursuit," *IEEE Transactions on Signal Processing*, vol. 51, no. 1, pp. 101–111, January 2003.

[6] R. Gribonval and M. Nielsen, "Approximate weak greedy algorithms," *Advances in Computational Mathematics*, vol. 14, no. 4, pp. 361–378, May 2001.

[7] R. Gribonval, *Matching Pursuit package 2.0*, IRISA-INRIA, 2001, into LastWave software, E. Bacry, `http://wave.cmap.polytechnique.fr/soft/LastWave/`.

[8] D. Donoho S. Chen and M. Saunders, "Atomizer for matlab5.x," see `http://www-stat.stanford.edu/~atomizer/`.

[9] R. Gribonval, "Sparse decomposition of stereo signals with matching pursuit and application to blind separation of more than two sources from a stereo mixture," in *Proc. ICASSP'02*, 2002.

[10] R. Gribonval, "Piecewise linear source separation," in *Proc. SPIE'03 – "Wavelets: Applications in Signal and Image Processing"*, 2003, vol. 5207, pp. 297–310.

[11] M. E. Davies and L. Daudet, "Sparse audio representations using the MCLT," *Signal Processing*, 2005 (to appear).

[12] L. Daudet, "Sparse and structured decompositions of signals with the molecular matching pursuit," *IEEE Trans. Speech and Audio Processing*, 2005 (to appear).

[13] M. Goodwin and M. Vetterli, "Matching pursuit and atomic signal models based on recursive filter banks," *IEEE Trans. Sig. Proc.*, vol. 47, no. 7, July 1999.

[14] R. Gribonval and S. Krstulović, "MPTK, The Matching Pursuit Toolkit," see `http://mptk.gforge.inria.fr`.