An exact tree projection algorithm for wavelets

Coralia Cartis and Andrew Thompson

Abstract

We propose a dynamic programming algorithm for projection onto wavelet tree structures. In contrast to other recently proposed algorithms which only give approximate tree projections for a given sparsity, our algorithm is guaranteed to calculate the projection exactly. We also prove that our algorithm has $\mathcal{O}(Nk)$ complexity, where N is the signal dimension and k is the sparsity of the tree approximation.

Index Terms

Sparse representations, wavelets, dynamic programming, complexity analysis, compressed sensing.

I. Introduction

The discrete wavelet transform is a much-used tool in digital signal processing, especially in image processing, since it provides sparse representations for piecewise-smooth signals [1]. Wavelets have a multi-scale structure in which a signal is convolved with dilations and translations of some 'mother wavelet', which induces a natural dyadic tree structure upon the wavelet coefficients. Since wavelets essentially detect discontinuities, large wavelet coefficients of piecewise smooth signals are often propagated down branches of the tree. This suggests that sparse approximations of such signals have additional structure: they are *tree-sparse*, by which we mean that they are supported on a rooted subtree [2], see Section II. This paper concerns the task of finding an exact tree projection, namely a tree-sparse vector of a given sparsity which is closest in Euclidean norm to some given signal vector.

Tree projections, or approximations thereof, predate the arrival of wavelets. An early example of their computation is found in the pruning of decision tree structures known as classification and regression trees (CART) [3], [4], [5]. With the emergence of wavelets, tree approximations also began to be used in the context of wavelet-based compression [6], [7] and orthogonal best-basis methods [8], [5], [9]. More recently, algorithms for tree-based Compressed Sensing have been proposed which require the computation of tree projections [10], [2]. Furthermore, recovery guarantees have been obtained for these algorithms which depend crucially on the assumption that an exact tree projection is computed in every iteration of the algorithm [10], [2].

Several algorithms have been proposed for wavelet tree projection, and the summary by Baraniuk in [11] describes three possible approaches: *Greedy Tree Approximation* (GTA), the *Condensing Sort and Select Algorithm* (CSSA), and the *Complexity-Penalized Sum of Squares* (CPSS). GTA [7] proceeds by growing the tree from the root node in a 'greedy' fashion by making a series of locally optimal choices. However, such an approach is only guaranteed to find an exact tree projection in the idealized case in which the wavelet coefficients are monotonically nonincreasing along the branches [11]. The CSSA was originally proposed by Baraniuk and Jones [12] in the context of optimal kernel design, and solves a linear programming (LP) relaxation of the exact tree projection problem. The CPSS approach has its roots in the *minimal cost-complexity pruning* algorithm for CART [3], and was later developed by Donoho in the context of wavelet approximations [5], [9]. This algorithm solves a Lagrangian relaxation of the exact tree projection problem, in which the sparsity constraint is removed and penalized in the objective function instead. We argue in Section II that, since both the CSSA and CPSS solve relaxations, neither algorithm is guaranteed to find an exact tree projection for a given sparsity. Since the theoretical results mentioned above require an exact tree projection, a crucial question is whether it is possible to guarantee the computation of an exact tree projection in polynomial time.

In this paper, we propose an algorithm for exact tree projection in the context of wavelets. We make use of a Dynamic Programming (DP) approach which has been used before in other contexts. An early reference is Breiman et al. [3], where it is proved that such an algorithm exists in the context of CART decision trees. The algorithm,

C. Cartis is with the School of Mathematics and the Maxwell Institute, University of Edinburgh, Edinburgh, UK (coralia.cartis@ed.ac.uk).
A. Thompson is with the Department of Mathematics, Duke University, Durham, North Carolina (thompson@math.duke.edu).

referred to as *optimal tree pruning*, was then formally stated by Bohanec and Bratko [4], who also proved that it has complexity $\mathcal{O}(N^2)$, where N is the signal dimension. The same exact tree projection approach was also used in the context of orthogonal best-basis selection in [8]. Our algorithm adopts the same basic approach as in [4], but this time in the context of wavelet approximation; further distinctions to [4] are given in Section III. We also prove that our algorithm has low-order polynomial complexity, namely $\mathcal{O}(Nk)$, where k is the sparsity of the tree approximation, which represents an improvement over the result in [4].

Some closely-related recent work [13] proves that there exists a polynomial-time DP algorithm for projection onto any tree structure which has a *loopless pairwise overlapping group* property. We should point out, however, that wavelet transforms may not exhibit this property.

II. Further motivation and preliminaries

We frame the discussion in terms of standard D-dimensional Cartesian-product dyadic wavelets, which have a particular canonical 2^D-ary tree structure. We assume a tree structure in which each coefficient has a maximum of d children, where d is the tree order. The Cartesian-product structure implies that $d = 2^D$, where the transform dimension D is some fixed positive integer, and so d is a fixed integer with $d \geq 2$. Also, let $N = d^J$ for some $J \geq 2$, where J is the number of levels in the tree structure. We assign a numbering $\{i : 1 \leq i \leq N\}$ to the nodes of the tree, starting with the root node and proceeding from coarse to fine levels. Let the root node $i = 1$ be at level 0, and have $d - 1$ children at level 1, namely $\{2, \ldots, d\}$. Let each of the remaining nodes except those in level J (the finest level), that is $\{i : 2 \leq i \leq d^{J-1} = N/d\}$, each have d children, namely $\{d(i-1)+1, \ldots, di\}$ respectively. The nodes in the finest level J, that is $\{i : i > d^{J-1} = N/d\}$, are assumed to have no children; see Figure 1.

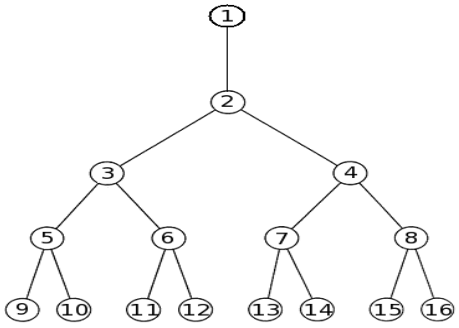

Fig. 1. An illustration of the canonical dyadic tree structure considered here, for the case of $d = 2$ (binary tree) and $J = 4$.

Following [2], we define a *rooted tree of cardinality* k to be some subtree consisting of k nodes of the full tree just described, subject to the restrictions that it must include the root node, and that if any node other than the root node is included then its parent node must also be included. We write $\mathcal{T}_k$ for the set of supports of all rooted trees of cardinality k. Given a vector $y \in \mathbb{R}^N$, we are interested in finding the Euclidean projection $\mathcal{P}_k(y)$ of a vector $y \in \mathbb{R}^N$ onto the set of vectors with support in $\mathcal{T}_k$, namely

$$\mathcal{P}_k(y) = \underset{\{z \in \mathbb{R}^N : \mathrm{supp}(z) \in \mathcal{T}_k\}}{\mathrm{argmin}} \|z - y\|_2, \tag{1}$$

where $\mathrm{supp}(z)$ denotes the support of z. Provided the above numbering scheme is used to distinguish between supports with precisely the same approximation error, $\mathcal{P}_k(y)$ is well-defined. Intuitively, the following lemma establishes that a tree projection preserves the values of selected coefficients.

Lemma 1: Let $\mathcal{P}_k(\cdot)$ be defined as in (1). Then

$$\{\mathcal{P}_k(y)\}_i := \begin{cases} y_i & i \in \Gamma \\ 0 & i \notin \Gamma \end{cases} \quad \text{where} \quad \Gamma := \underset{\Omega \in \mathcal{T}_k}{\mathrm{argmax}} \|y_\Omega\| . \tag{2}$$

Proof: Let $\mathrm{supp}(z) = \Gamma \in \mathcal{T}_k$. Then

$$\|z - y\|^2 = \|(z-y)_\Gamma\|^2 + \|y_{\Gamma^C}\|^2 ,$$

and $\|(z-y)_\Gamma\|^2$ is minimized by setting $z_i = y_i$ for all $i \in \Gamma$. Meanwhile, $\|y_{\Gamma^C}\|$ is minimized by choosing Γ to be the set $\Omega \in \mathcal{T}_k$ which maximizes $\|y_\Omega\|$, and the result now follows. □

Lemma 1 implies that tree projection is a decision problem: since coefficient values are preserved, the projection is obtained by identifying the set $\Gamma \in \mathcal{T}_k$ on which y has maximum energy. It follows that $\mathcal{P}_k(y)$ may also be found by solving the following Integer Program (IP):

$$\max_{\tau \in \mathbb{Z}^N} \sum_{i=1}^{N} y_i^2 \tau_i \quad \text{subject to} \quad \begin{array}{l} \tau \geq 0 \\ \{\tau_i\} \text{ tree-nonincreasing} \\ \sum \tau_i = k \\ \tau_1 = 1. \end{array} \tag{3}$$

Here tree-nonincreasing means that the coefficients do not increase along the branches, a condition which may be translated into a series of linear constraints. Note that the assumption of integrality, together with the constraints, in fact forces $\tau_i \in \{0, 1\}$ for each i, so that τ acts as a decision variable. Denoting the optimal solution of (3) by τ^*, the coefficients of the best tree approximation are then $y_i \tau_i^*$.

We can now be more precise about our claim in Section I that both the CSSA [12] and the CPSS [5] solve relaxations of the tree projection problem, and hence are only guaranteed to give approximate tree projections.

The CSSA solves an LP relaxation of (3) in which one dispenses with the assumption of integrality while retaining all other constraints. The authors of [12] observe that the two solutions are sometimes equal, but not always: a value of $\tau_i = 1$ may be assigned to all but a few coefficients, which are each assigned a value $\bar{\tau}$ for some $0 < \bar{\tau} < 1$. In this case, the result is a tree of size strictly greater than k. Moreover, there is no straightforward method for 'adjusting' the solution *a posteriori* to obtain the optimal k-sparse tree projection.

Meanwhile, the CPSS [5] solves a Lagrangian relaxation of (3) in which the equality constraint $\sum \tau_i = k$ is removed and penalized in the objective instead. This reformulation can be solved using fine-to-coarse dynamic programming on the tree [5], [9]. Because (3) is an IP, its Lagrangian relaxation is also not guaranteed to share the same optimal solution as (3) [14]. A further drawback of the CPSS is the non-obvious relationship between the Lagrange multiplier λ and the required sparsity, which means that the problem must in practice be solved for multiple values of λ in order to find a tree approximation for a given sparsity.

III. The ETP algorithm

Our algorithm (Algorithm 1) falls into the broad category of dynamic programming (DP) algorithms which optimize on directed graphs by utilizing a principle of optimality, namely that optimal solutions at a given node may be determined entirely from optimal solutions at 'preceding' nodes [15]. Our algorithm makes two passes through the tree: firstly, a fine-to-coarse pass finds optimal subtrees at each node for all $\tilde{k} \leq k$. Secondly, a coarse-to-fine pass tracks back to identify the optimal solution.

The algorithm starts at the finest level and moves up the tree, finding optimal subtrees rooted at each node, each decision requiring only the information already obtained within that particular subtree. Following [4], optimal subtrees for a given node are obtained by successively incorporating each of its children, and repeatedly optimizing over all known solutions. To maximize efficiency, optimal subtrees are in fact only found for all cardinalities which could possibly contribute to a rooted tree of cardinality k, which imposes two restrictions. Firstly, by summing the appropriate geometric series, the maximum cardinality of a subtree rooted at a node in level j is $\frac{d^{J+1-j}-1}{d-1}$. Secondly, any subtree rooted at a node in level j of cardinality greater than $k-j$ would necessarily contribute to a rooted tree of cardinality greater than k. Defining

$$l(j) := \min\left(\frac{d^{J+1-j}-1}{d-1}, k-j\right) \quad 1 \leq j \leq J, \tag{4}$$

we therefore need only find optimal subtrees for cardinalities up to $l(j)$ for nodes at level j. Further restrictions are imposed due to the fact that children are incorporated sequentially.

$F(i,l)$ denotes the total energy of the optimal subtree rooted at node i, of size l, and $G(i,l) = [G_r(i,l) : r = 1, \ldots, d] \in \mathbb{R}^d$ is a vector of nonnegative integers, giving the number of nodes contributing to this optimal solution from the subtree rooted at each of the children of node i. We then proceed recursively and eventually determine $F(1,k)$, the energy of the optimal tree of size k.[1] Finally, we use the precedence information to trace the optimal

[1]We also need to define two temporary variables $\tilde{F}$ and $\tilde{G}$ which represent updates to F and G during the incorporation of a child.

solution back along the branches. The algorithm actually does more than is asked for: by the time the root node is reached at the end of the first pass, enough information has been obtained to determine optimal trees for all $\tilde{k} \leq k$.

ETP is closely related, but not identical, to the optimal tree pruning algorithm in [4], where the decision tree context motivates the solution of a slightly different problem from the present tree projection problem. In their setup, if a node is pruned, then so are all other nodes sharing the same parent in the tree structure. Also, an optimal pruning is obtained for a given number of *leaves* (ends of branches of the tree), as opposed to the cardinality of the tree approximant. There is also a difference in how information is stored: the proposal in [4] is to store the full optimal subtrees themselves at each node, essentially incorporating the backtracking phase into the first pass of the algorithm.

Algorithm 1. Exact tree projection (ETP).

Inputs: $\begin{cases} d \in \mathbb{N} & (d \geq 2) \\ y \in \mathbb{R}^N & (N = d^J;\ J \geq 2) \\ k \in \mathbb{N} & (2 \leq k \leq N) \end{cases}$

Find all optimal subtrees:

for $i = (d^{J-1} + 1) : d^J$ **do**
  $F(i,0) = 0$ and $F(i,1) = y_i^2$
end for
for $j = (J-1) : -1 : 1$ **do**
  for $i = (d^{j-1} + 1) : d^j$ **do**
    $F(i,0) = 0$ and $F(i,1) = y_i^2$
    $G(i,0) = 0$ and $G(i,1) = 0$
    for $r = 1 : d$ **do**
      for $l = 2 : \min[l(j), rl(j+1) + 1]$ **do**
        $s_- = \max\{1, l - [(r-1)l(j+1) + 1]\}$
        $s_+ = \min[l - 1, l(j+1)]$
        $\hat{s} = \underset{s_- \leq s \leq s_+}{\operatorname{argmax}} \{F[d(i-1) + r, s] + F(i, l-s)\}$
        $\tilde{F}(i,l) = F[d(i-1) + r, \hat{s}] + F(i, l - \hat{s})$
        $\tilde{G}(i,l) = G(i, l - \hat{s})$, $\tilde{G}_r(i,l) = \hat{s}$
      end for
      for $l = 2 : \min[l(j), rl(j+1) + 1]$ **do**
        $F(i,l) = \tilde{F}(i,l)$ and $G(i,l) = \tilde{G}(i,l)$
      end for
    end for
  end for
end for
for $r = 2 : d$ **do**
  for $l = 2 : \min[k, (r-1)l(1) + 1]$ **do**
    $s_- = \max\{1, l - [(r-2)l(1) + 1]\}$
    $s_+ = \min[l - 1, l(1)]$
    $\hat{s} = \underset{s_- \leq s \leq s_+}{\operatorname{argmax}} [F(r,s) + F(1, l-s)]$
    $\tilde{F}(1,l) = F(r, \hat{s}) + F(1, l - \hat{s})$
    $\tilde{G}(1,l) = G(1, l - \hat{s})$, $\tilde{G}_r(1,l) = \hat{s}$
  end for
  for $l = 2 : \min[k, (r-1)l(1) + 1]$ **do**
    $F(1,l) = \tilde{F}(1,l)$ and $G(1,l) = \tilde{G}(1,l)$
  end for
end for

Backtrack to identify solution:

$\tau = 0$, $\tau_1 = 1$, $\Gamma_1 = k$
for $j = 0 : (J-1)$ **do**

for $i = \max[2, (d^{j-1}+1)] : d^j$ **do**
if $\tau_i = 1$ **then**
for $r = \max(1, 2-j) : d$ **do**
if $G_r(i, \Gamma_i) > 0$ **then**
$\tau_{d(i-1)+r} = 1$
$\Gamma_{d(i-1)+r} = G_r(i, \Gamma_i)$
end if
end for
end if
end for
end for
Calculate solution:
for $i = 1 : N$ **do**
$\hat{y_i} = y_i \tau_i$
end for
Output: $\hat{y} \in \mathbb{R}^N$

IV. Complexity analysis

To establish the complexity of ETP, we note that the dominating operations are additions and comparisons, and therefore it suffices to bound the total number of these operations.

Theorem 2: Given $y \in \mathbb{R}^N$, the ETP algorithm requires at most $(3d^2Nk+N)$ additions/comparisons to calculate $\mathcal{P}_k(y)$, defined in (1).

Proof: Consider the first pass of ETP. In each level j with $1 \leq j \leq J-1$, there are $(d-1)\cdot d^{j-1}$ nodes. For a given node i at level j, each of its d children are incorporated successively. Each time a child is incorporated, we calculate optimal subtrees for each l such that $2 \leq l \leq l_{max}$. Each of these calculations requires at most $(l-1)$ additions, each one accompanied by a comparison, and we have $l_{max} \leq l(j)$, where $l(j)$ is defined in (4). The number of additions/comparisons needed to incorporate a child is therefore bounded by

$$2\sum_{l=2}^{l(j)}(l-1) \leq l(j)[l(j)-1] \leq [l(j)]^2. \tag{5}$$

Combining all these observations, and writing $\mathcal{C}$ for the total number of additions/comparisons in the first pass of the algorithm for levels $1 \leq j \leq J-1$ (i.e. excluding the root node), we have

$$\mathcal{C} \leq \sum_{j=1}^{J-1} \left\{(d-1)d^{j-1}\cdot d \cdot [l(j)]^2\right\}, \tag{6}$$

To bound $l(j)$, observe from (4) that

$$l(j) \leq \min(d^{J+1-j}, k). \tag{7}$$

To determine which of d^{J+1-j} or k gives a tighter bound, let $1 \leq p \leq J-1$ be the unique positive integer such that

$$d^{p-1} \leq k \leq d^p. \tag{8}$$

First let us assume $p > 2$. We have

$$k \leq d^{J+1-j} \iff d^p \leq d^{J+1-j} \iff j \leq J+1-p,$$

which we may combine with (6) and (7) to deduce

$$\mathcal{C} \leq \sum_{j=1}^{J+1-p}(d-1)d^j \cdot k^2 + \sum_{j=J+2-p}^{J-1}(d-1)d^j \cdot d^{2(J+1-j)} \tag{9}$$

$$= (d-1)\left\{ dk^2 \sum_{j=1}^{J+1-p} d^{j-1} + \sum_{j=J+2-p}^{J-1} d^{2J+2-j} \right\}$$

$$= (d-1)\left\{ dk^2 \left(\frac{d^{J+1-p}-1}{d-1} \right) + d^{J+p} \left[\frac{1-\left(\frac{1}{d}\right)^{p-2}}{1-\frac{1}{d}} \right] \right\}. \tag{10}$$

Since

$$(d-1)\left[\frac{1-\left(\frac{1}{d}\right)^{p-2}}{1-\left(\frac{1}{d}\right)} \right] \leq \left(\frac{d-1}{1-\frac{1}{d}} \right) = d,$$

and since $d^{J+1-p} - 1 \leq d^{J+1-p}$, we can further deduce from (10) that

$$\mathcal{C} \leq k^2 \cdot d^{J+2-p} + d^{J+1+p},$$

to which we can make the substitution $N = d^J$ and apply the bounds (8), concluding that

$$\mathcal{C} \leq k^2 \cdot \frac{d^2 N}{k} + d^2 N k = 2d^2 N k. \tag{11}$$

On the other hand, if $p \leq 2$, then $k \leq d^2 \leq d^{J+1-j}$ for all $j \leq J-1$, and so the second summation in (9) is empty. We may then follow the same argument to bound the first summation, obtaining $\mathcal{C} \leq 2d^2 N k$ in this case also. Meanwhile, the root node has $(d-1)$ children, which may be combined with (5) and (7) to deduce that the number of additions/comparisons for the root node is bounded by

$$(d-1)k^2 \leq d^2 N k. \tag{12}$$

Finally, note that the second pass involves no additions and at most N comparisons, which combines with (11) and (12) to yield the result. □

It follows from Theorem 2 that ETP has complexity $\mathcal{O}(Nk)$.[2] We may compare the complexity of ETP with that of the other approximate tree projection methods discussed in Section II: GTA [7] and the CSSA [12] are both $\mathcal{O}(N \log N)$ while the CPSS [5] is $\mathcal{O}(N)$. Provided $\log N \ll k$, we can see that the price we pay for guaranteeing an exact tree projection is an increased order of complexity.

The algorithm is easy to implement[3] and we have experimented with random tests successfully; its cost is not disappointing compared to approximate methods such as the CSSA and the CPSS.

V. Conclusion

We have presented an algorithm for exact tree projection in the context of wavelets and have proved that it has $\mathcal{O}(Nk)$ worst-case complexity. The DP algorithm described here could also be applied to any tree-based signal representations. Approximate algorithms may still offer an advantage in terms of computational efficiency, and therefore the practitioner's choice of tree projection algorithm will depend upon whether guaranteed precision or algorithm running time is the overriding priority.

References

[1] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed. Academic Press, 2009.

[2] R. Baraniuk, V. Cevher, M. Duarte, and C. Hegde, "Model-based compressive sensing," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1982–2001, 2010.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and regression trees*, 1st ed. Chapman and Hall, 1984.

[4] M. Bohanec and I. Bratko, "Trading accuracy for simplicity in decision trees," *Machine Learning*, vol. 15, no. 3, pp. 223–250, 1994.

[5] D. Donoho, "Cart and best ortho-basis: A connection," *Annals of Statistics*, vol. 25, no. 5, pp. 1870–1911, 1997.

[6] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

[7] A. Cohen, W. Dahmen, I. Daubechies, and R. Devore, "Tree approximation and optimal encoding," *Applied and Computational Harmonic Analysis*, vol. 11, no. 2, pp. 192–226, 2001.

[2] Note that, except for the constant involved, the result is independent of d.

[3] Matlab code is available to download from `www.math.duke.edu/~thompson`.

[8] R. Coifman and M. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 713–718, 1992.
[9] D. Donoho, N. Dyn, D. Levin, and T. Yu, "Smooth multiwavelet duals of alpert bases by moment-interpolating refinement," *Applied and Computational Harmonic Analysis*, vol. 9, no. 2, pp. 166–203, 2000.
[10] T. Blumensath and M. Davies, "Sampling theorems for signals from the union of finite-dimensional linear subspaces," *IEEE Transactions on Information Theory*, vol. 55, no. 4, pp. 1872–1882, 2009.
[11] R. Baraniuk, "Optimal tree approximation with wavelets," in *Proceedings of the SPIE Technical Conference on Wavelet Applications in Signal Processing VII*, 1999.
[12] R. Baraniuk and D. Jones, "A signal-dependent time-frequency representation: Fast algorithm for optimal kernel design," *IEEE Transactions on Signal Processing*, vol. 42, no. 12, pp. 3530–3535, 1994.
[13] N. Bhan, L. Baldassarre, and V. Cevher, "Tractability of interpretability via selection of group-sparse models," 2013, preprint.
[14] G. Nemhauser and L. Wolsey, *Integer and combinatorial optimization*, 1st ed. Wiley, 1988.
[15] R. Bellman, *Dynamic Programming*. Courier Dover Publications, 2003.