Polyphase decompositions and shift-invariant discrete wavelet transforms in the frequency domain

Alle Meije Wink [a] , Jos B. T. M. Roerdink [b,*]

[a]*Imaging Sciences Department and MRC Clinical Sciences Centre, Hammersmith Campus, Imperial College London, United Kingdom*

[b]*Institute for Mathematics and Computing Science, University of Groningen, The Netherlands*

Abstract

Given a signal and its Fourier transform, we derive formulas for its polyphase decomposition in the frequency domain and for the reconstruction from the polyphase representation back to the Fourier representation. We present two frequency-domain implementations of the shift-invariant periodic discrete wavelet transform (SI-DWT) and its inverse: one that is based on frequency-domain polyphase decomposition and a more efficient 'direct' implementation, based on a reorganisation of the *à trous* algorithm.

We analyse the computational complexities of both algorithms, and compare them to existing time-domain and frequency domain implementations of the SI-DWT. We experimentally demonstrate the reduction in computation time achieved by the direct frequency domain implementation of the SI-DWT for wavelet filters with non-compact support.

Key words: polyphase decomposition, frequency domain implementation, shift-invariant discrete wavelet transform.

---

* Corresponding author. Tel: +31-50-3633931; Fax: +31-50-3633800
Email addresses: `allemeije.wink@csc.mrc.ac.uk` (Alle Meije Wink),

1 Introduction

The discrete wavelet transform (DWT) is extensively used in signal processing applications. Some versions of the DWT have been implemented in the frequency domain, see *e.g.* [7, 9, 10, 12, 13]. Westenberg and Roerdink used this Fourier implementation of the DWT for a frequency domain implementation of the wavelet X-ray transform [14]. A problem of the original DWT is that it is not shift-invariant, *i.e.*, the DWT of a shifted signal cannot be found by shifting the DWT coefficients.

A shift-invariant discrete wavelet transform (SI-DWT) was introduced independently by Holschneider and Shensa [3, 8], who called it the *à trous* algorithm, and by Coifman and Donoho [2], who called it *cycle spinning*. This algorithm is based on computing the wavelet coefficients for all possible circularly shifted copies of the input signal. Extensions of the standard DWT have been proposed which are approximately shift-invariant [4]; its implementation in the Fourier domain is considered in [6]. Here we consider the frequency domain implementation of the exactly shift-invariant DWT.

Making the wavelet transform shift-invariant requires a large amount of additional computation. For this reason, high-speed hardware implementations have been proposed [1]. The Rice Wavelet Toolbox [`www.dsp.rice.edu/software/RWT`] (RWT) contains an implementation of the SI-DWT that is based on the time-domain FWT. Another way to speed up computation of the SI-DWT similarly to the DWT, is doing the convolutions in the Fourier domain. A frequency domain SI-DWT is more efficient than the time-domain implementation for filters that have non-compact support. Such implementations have been considered in the past, notably

`j.b.t.m.roerdink@rug.nl` (Jos B. T. M. Roerdink).

by Rioul and Duhamel [7], based on polyphase decompositions. Their method, henceforth referred to as the RD algorithm, performs the convolution steps of the SI-DWT in the Fourier domain, while computing the downsampling and shift operations in the time domain, for all J levels (octaves) of the wavelet decomposition. This requires a large number of forward and backward FFT steps for each level of the wavelet decomposition, and leads to an algorithm which is somewhat complex to implement. The RD algorithm is targeted towards handling very long (potentially infinite) data sequences, and processes the data by dividing it in blocks and performing the required operations (Fourier transform, complex multiplication, downsampling) on each block at a time. Furthermore, the RD algorithm computes the *aperiodic* SI-DWT. Rioul and Duhamel [7] describe an extension of the RD algorithm, based on the Vetterli algorithm [11], which gathers a certain number J_0 (where $J_0 < J$) of consecutive octaves in one step by performing the subsampling in the frequency domain, thus avoiding subsequent forward and backward FFTs.

The goal of this paper is to present a simple and efficient alternative to the RD frequency domain implementation for the case of the periodic SI-DWT. Our algorithm for the SI-DWT, called the 'direct' implementation, employs all upsampling and downsampling of signals in the frequency domain, for arbitrary sampling factors and signal shifts. It uses only a single initial FFT and one final IFFT; all other steps consist of simple copying and multiplications of matrix elements. In that sense, our approach is the equivalent to application of the Vetterli algorithm for *all* octaves, i.e., $J_0 = J$. We describe another SI-DWT implementation using explicit polyphase decompositions (the 'polyphase' implementation), and show that the direct implementation has superior efficiency. Although the essential ingredients of the direct implementation have been known for a long time, the simple algorithm presented here has, to the best of our knowledge, not been described before. The algorithm can be easily extended to higher dimensions by using tensor product wavelet bases.

We analyse the time complexity of the direct and polyphase methods and compare them to the RD algorithm (slightly modified for periodic DWTs) and the time-domain SI-DWT, using non-compact filters (*i.e.*, with length N). For a fixed J, the time-domain implementation of the SI-DWT is quadratic in N. The frequency domain implementations are all order $N \log_2 N$, of which the RD and direct algorithms are faster, and of those two the direct algorithm is most efficient.

For large N the speed gain of the direct algorithm over the RD method equals $3J/(J+2)$. Although this is not an order of magnitude difference, this is a noticeable improvement for processing large data sets. An example is wavelet analysis of functional magnetic resonance imaging (fMRI) data, which actually prompted our interest in efficient SI-DWT implementations. Here the SI-DWT is computed for each time series in a sequence of image volumes. As each volume consists of several millions of data points, this means computing millions of SI-DWTs [15]. The direct algorithm proposed in this paper significantly reduces computation time for long wavelet filters.

The organisation of this paper is as follows. Section 2.1 summarises the polyphase decomposition and monophase reconstruction in the frequency domain. Section 3 reviews the shift-invariant wavelet transform (SI-DWT) and section 4 presents the three frequency domain implementations. The complexity analysis is carried out in section 5. Conclusions are drawn in section 6.

2 Polyphase decomposition in the frequency domain

2.1 *Upsampling and downsampling*

Implementations of the DWT by the fast wavelet transform (FWT) use the convolution operator, as well as up/downsampling by a factor of 2. Downsampling corresponds to biphase decomposition, and discarding the second phase. These operations can all be implemented in the Fourier domain [12, 14].

It has been shown [14] that given the *Z-transform* $\mathfrak{X}[z]$ of a discrete signal $x(n)$ of length N, the even and odd samples of a signal x are given by $\mathfrak{X}^{\text{even}}[z] = \frac{1}{2}\left(\mathfrak{X}[z^{\frac{1}{2}}] + \mathfrak{X}[-z^{\frac{1}{2}}]\right)$ and $\mathfrak{X}^{\text{odd}}[z] = \frac{1}{2}z^{\frac{1}{2}}\left(\mathfrak{X}[z^{\frac{1}{2}}] - \mathfrak{X}[-z^{\frac{1}{2}}]\right)$, respectively. The Z-transform of the signal upsampled by a factor 2 is given by $\mathfrak{X}^{\text{up},2}[z] = \mathfrak{X}\left[z^2\right]$. On the unit circle in the complex plane, the Z-transform $\mathfrak{X}[e^{2\pi i k/N}]$ coincides with the element $X(k) = \sum_{n=0}^{N-1} x(n)\, e^{\frac{-2\pi i k\, n}{N}}$ of the discrete Fourier transform (DFT) of x.

2.2 *Polyphase decomposition and monophase reconstruction*

Here we consider a more general subsampling scheme which decomposes a signal into Q phases, where all phases are retained. A discrete signal $x(n)$ can be subsampled by a factor $Q \in \mathbb{N}^+$ in Q different ways (if Q is a divisor of N) by shifting the signal over $0, \ldots, Q-1$ positions, respectively. The signal is then in its *polyphase* form; we will refer to the operation above as the *polyphase decomposition*. The original signal is retrieved via the *monophase reconstruction*, which interleaves the Q signals after upsampling by a factor Q.

Given a signal $x(n)$ of length $N = 2^m$, $m \in \mathbb{N}^+$ and a number Q that is a divisor of

N, the *polyphase decomposition* is defined as

$$x^{Q,0} = (x(0),\ x(Q),\ x(2Q),\ \ldots,\ x(N-Q))$$

$$x^{Q,1} = (x(1),\ x(Q+1),\ x(2Q+1),\ \ldots,\ x(N-Q+1))$$

and so on, splitting x into *phase components* $x^{Q,q}$ as follows:

$$x^{Q,q}(n) = x(Q\,n+q), \qquad q = 0,1,\ldots,Q-1, \qquad n = 0,1,\ldots,N/Q-1.$$

Conversely, the *monophase reconstruction*

$$x(n) = x^{Q,n \bmod Q}(n \operatorname{div} Q), \quad n = 0,1,\ldots,N-1 \tag{1}$$

restores a signal $x(n)$ from its polyphase components $x^{Q,q}$. That is, to collect the elements $x(n)$: for $n = 0\ldots Q-1$, take elements $x^{Q,0}(0)$, $x^{Q,1}(0)\ldots x^{Q,Q-1}(0)$; for $n = Q\ldots 2Q-1$ take elements $x^{Q,0}(1)$, $x^{Q,1}(1)\ldots x^{Q,Q-1}(1)$, and so on.

2.2.1 The Z-transform

The Z-transform of each $x^{Q,q}$ is denoted by $\mathfrak{X}^{Q,q}[z]$:

$$\mathfrak{X}^{Q,q}[z] = \sum_{n=0}^{N/Q-1} x(Q\,n+q)z^{-n}.$$

The decomposition of x in Q phases has the the Z-transform

$$\begin{aligned}\mathfrak{X}[z] &= \sum_{n=0}^{N-1} x(n)z^{-n}\\ &= \sum_{q=0}^{Q-1}\sum_{n'=0}^{N/Q-1} x(Q\,n'+q)z^{-(Qn'+q)}\\ &= \sum_{q=0}^{Q-1} z^{-q}\,\mathfrak{X}^{Q,q}[z^Q]\end{aligned} \tag{2}$$

which is a generalisation of the equations in [14], and can be used to represent the relation between the Z-transform and the DFT of a phase component.

Theorem 1 *Let $x(n)$ be a signal of length N with Z-transform $\mathfrak{X}[z]$. Let $x^{Q,q}$ denote the signal downsampled by a factor Q and shifted over index q. Then:*

$$\mathfrak{X}^{Q,q}[z] = \frac{1}{Q} z^{\frac{q}{Q}} \sum_{\ell=0}^{Q-1} e^{\frac{2\pi i \ell q}{Q}} \mathfrak{X}\left[e^{\frac{2\pi i \ell}{Q}} z^{\frac{1}{Q}}\right]. \tag{3}$$

Proof: Insert (2) into the sum (denoted *SUM*) in the right-hand side of (3):

$$\begin{aligned}
\mathit{SUM} &= \sum_{\ell=0}^{Q-1} e^{\frac{2\pi i \ell q}{Q}} \mathfrak{X}\left[e^{\frac{2\pi i \ell}{Q}} z^{\frac{1}{Q}}\right] \\
&= \sum_{\ell=0}^{Q-1} e^{\frac{2\pi i \ell q}{Q}} \sum_{m=0}^{Q-1} e^{-\frac{2\pi i \ell m}{Q}} z^{-\frac{m}{Q}} \mathfrak{X}^{Q,m}[z] \\
&= \sum_{m=0}^{Q-1} z^{-\frac{m}{Q}} \mathfrak{X}^{Q,m}[z] \underbrace{\sum_{\ell=0}^{Q-1} e^{\frac{2\pi i \ell (q-m)}{Q}}}_{Q\,\delta_{q,m}} \\
&= z^{-\frac{q}{Q}} \mathfrak{X}^{Q,q}[z]\, Q,
\end{aligned}$$

where $\delta_{q,m}$ are Kronecker deltas. This completes the proof. ■

2.2.2 *The DFT: polyphase decomposition*

Let $X(k)$ be the N-point DFT of $x(n)$, let and $X^{Q,q}(k)$ the N/Q-point DFT of phase component $x^{Q,q}$, *i.e.*, :

$$X^{Q,q}(k) = \mathfrak{X}^{Q,q}\left[e^{\frac{2\pi i k}{N/Q}}\right] = \sum_{n=0}^{N/Q-1} x(Qn+q)\, e^{-\frac{2\pi i k\, n}{N/Q}}, \quad k = 0, \ldots, N/Q-1. \tag{4}$$

Application of formula (3) to (4) yields

$$X^{Q,q}(k) = \frac{1}{Q} e^{\frac{2\pi i k q}{N}} \sum_{\ell=0}^{Q-1} e^{\frac{2\pi i \ell q}{Q}} \mathfrak{X}\left[e^{\frac{2\pi i (k+\ell N/Q)}{N}}\right].$$

Using the fact that $\mathfrak{X}[e^{2\pi i k/N}] = X(k)$, we therefore find that the equation which expresses the frequency domain polyphase decomposition (FPD) is given by:

$$X^{Q,q}(k) = \frac{1}{Q} e^{\frac{2\pi i k q}{N}} \sum_{\ell=0}^{Q-1} e^{\frac{2\pi i \ell q}{Q}} X\left(k + \frac{\ell N}{Q}\right). \tag{5}$$

This formula expresses the DFT coefficients of the phase components $X^{Q,q}$ in terms of the DFT $X(k)$ of signal $x(n)$.

2.2.3 Monophase reconstruction

The frequency domain monophase reconstruction (FMR) transforms the DFT coefficients of the polyphase components of a signal back into the DFT coefficients of the signal itself. It is given by the following equation:

$$X(K) = \sum_{q=0}^{Q-1} e^{\frac{-2\pi i q K}{N}} X^{Q,q}(k), \quad K = k + \frac{\ell N}{Q}, \tag{6}$$

for $k = 0, \ldots, N/Q - 1$, $\ell = 0, \ldots, Q - 1$. Its proof is similar to theorem 1.

3 The shift-invariant discrete wavelet transform

In the wavelet representation, a signal is a superposition of transient waveforms, which are basis functions of a sequence of nested function spaces [12]. A multiresolution representation of a discrete signal $c^0 := (c^0(0), c^0(1), \ldots, c^0(N-1))$ is made by repeatedly splitting signals c^j $(j \geq 0)$ into approximation c^{j+1} and detail d^{j+1} parts, by filtering with lowpass and bandpass filters h and g, respectively.

The fast wavelet transform (FWT) with filters of length L changes a discrete signal of N points to its wavelet representation in $\mathcal{O}(L\,N)$ time, by recursively downsampling c^j and d^j after filtering. Its inverse upsamples c^{j+1} and d^{j+1} and filters the upsampled signals with *dual* filters $\widetilde{h}$ and $\widetilde{g}$, before adding them together (see Fig. 1a-b). If h and g define an orthogonal wavelet basis, the dual filters are defined as $\widetilde{h}(n) = \overline{h(-n)}$ and $\widetilde{g}(n) = \overline{g(-n)}$, $\overline{x}$ denoting the complex conjugate of x.

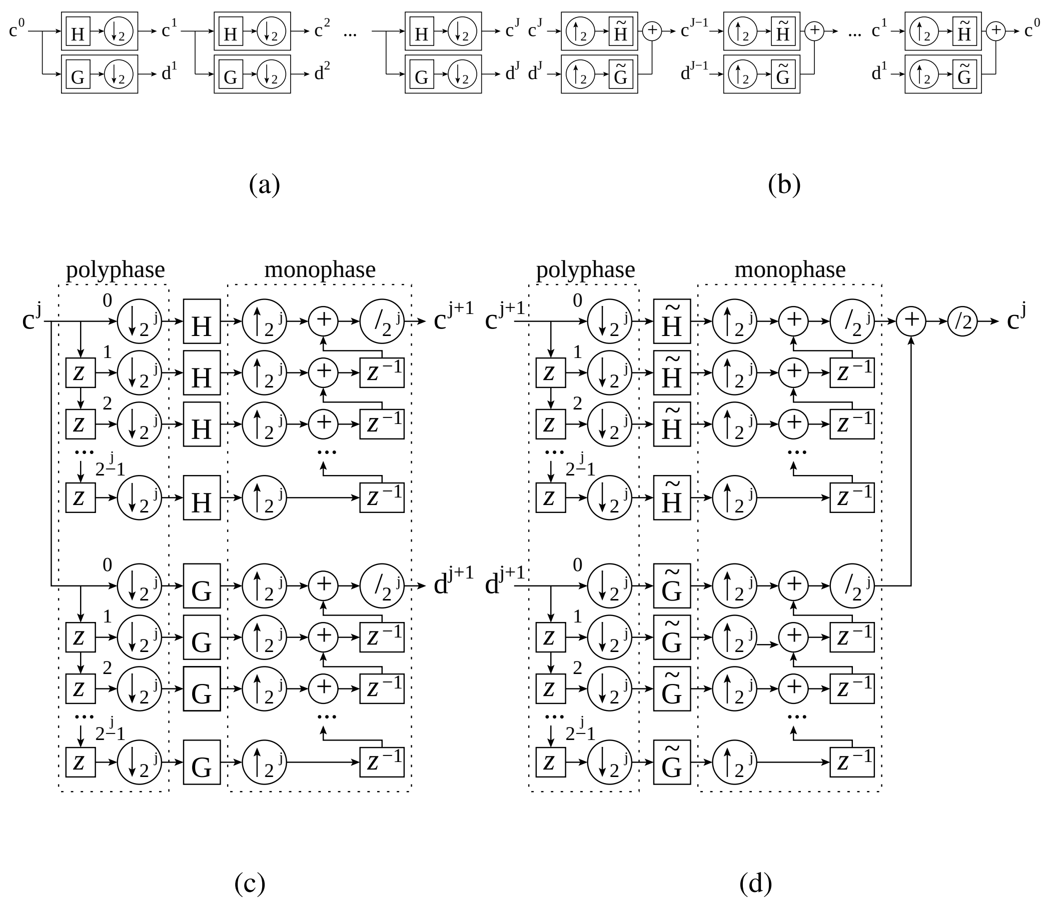


Fig. 1. *Graphical representations of the FWT (a), the IFWT (b), one level of the SI-DWT (c) and one level of the SI-IDWT (d).*

3.1 Definition of the SI-DWT

We introduce the SI-DWT as defined in Eq. (3.14) of Shensa's paper [8] describing the *à trous* algorithm. Let h, g be the scaling and wavelet filters of an orthonormal wavelet basis (this will guarantee that perfect reconstruction holds). The definition of the SI-DWT with J levels (octaves) then is [1] :

$$c^{j+1} = (\uparrow_Q h) * c^j, \quad d^{j+1} = (\uparrow_Q g) * c^j \tag{7}$$

for $Q = 2^j, j = 0, 1, \ldots, J{-}1$. Here, $*$ denotes discrete convolution, and the operation $\uparrow_Q x$ denotes upsampling of x by a factor Q, *i.e.*, inserting $Q{-}1$ zeros between

[1] We write d^{j+1} instead of $\tilde{w}^j$ $(j = 0, 1, \ldots)$ as in [8]

each pair of elements of x. Input is a vector c^0; output are vectors d^1,d^2,. . .,d^J,c^J. The d^j_k are the detail coefficients of the expansion of c^0 and the c^J_k are the approximation coefficients on the coarsest level. If the length of the input signal c^0 equals N, then for all levels j the length of c^j and d^j is N as well. Many coefficients of the filter after upsampling are zero (*à trous* filter = filter with holes).

The original signal is reconstructed recursively, starting at level J, by upsampling the dual filters $\widetilde{h}$ and $\widetilde{g}$, followed by the convolution

$$c^{j-1} = (\uparrow_Q h') * c^j + (\uparrow_Q g') * d^j \tag{8}$$

for $Q = 2^j; j = L, J-1, \ldots, 1$. Here $h' = \widetilde{h}/2$ and $g' = \widetilde{g}/2$, *i.e.*, the reconstruction filter coefficients are divided by 2 to account for the fact that the data size is not reduced by a factor of 2 in each step, but remains constant [8].

3.2 Polyphase transform

An efficient SI-DWT implementation skips multiplications of zero filter coefficients. To show this in more detail, (7) can be written as:

$$\begin{aligned} c^{j+1}(k) &= \sum_n h(-n)c^j(Q\,n+k), \qquad & d^{j+1}(k) &= \sum_n g(-n)c^j(Q\,n+k), \text{ or} \\ c^{j+1,q} &= h * c^{j,q}, & d^{j+1,q} &= g * c^{j,q} \end{aligned} \tag{9}$$

where $c^{j,q}(n) = c^j(Q\,n+q)$ and $d^{j,q}(n) = d^j(Q\,n+q)$ are the polyphase components of c^j and d^j, respectively. In step j the data vector c^j is split into $Q = 2^j$ blocks (polyphase decomposition), and each block is convolved with filter $\widetilde{h}$ (for c^{j+1}) or with $\widetilde{g}$ (for d^{j+1}). All blocks are assembled into new vectors c^{j+1} and d^{j+1}, with the same dimensions as c^0, by the monophase transform (1). The inverse transform (SI-IDWT) does the same polyphase filtering procedure, with $\widetilde{h}$ and $\widetilde{g}$.

The upsampled and filtered signals c^{j+1} and d^{j+1} are added together to produce c^j (see Fig. 1c-d).

In summary, the implementations of the shift-invariant discrete wavelet transform (SI-DWT) by the *à trous* algorithm [3,8] or *cycle spinning* [2], contain the following steps: subsample for all possible shifts (polyphase decomposition), filter the phase signals separately, and merge the filtered phase signals (monophase reconstruction).

4 Frequency domain SI-DWT

We now describe the implementations of the SI-DWT in the frequency domain. The FFT implementation produces a *cyclic* convolution. To avoid wrap-around effects, the signal has to be extended by zero elements (zero padding). We will look in more detail at this when we consider the computational complexity in section 5.

First we look at the method of Rioul and Duhamel [7]. Then we describe our two new implementations.

4.1 The method of Rioul and Duhamel

The approach of Rioul and Duhamel [7] is based on the polyphase representation (9). This representation contains three steps for each octave j: a polyphase decomposition, a convolution, and a monophase reconstruction. Only the convolution is done in the frequency domain. That is, after the polyphase decomposition an FFT is applied, followed by a multiplication of the Fourier coefficients and an IFFT. The monophase transform then gives the next octave $j+1$. This procedure is repeated until the maximum number of octaves J has been reached.

4.2 Direct Fourier-domain filtering

Our alternative method starts with an initial FFT of the input signal, and afterwards works purely in the frequency domain for all octaves. We start by an N-point FFT of the input signals and filters. The formulas for the SI-DWT (7) then become:

$$C^{j+1} = H^Q \bullet C^j, \quad D^{j+1} = G^Q \bullet C^j, \tag{10}$$

for $j = 0, 1, \ldots, J-1$, $Q = 2^j$. Input is a vector C^0; output are vectors D^1, D^2, ..., D^J, C^J. Here $x \bullet y$ denotes pointwise multiplication of vectors x and y, and C^j, D^j, H^Q and G^Q denote the N-point DFT vectors of c^j, d^j, $\uparrow_Q h$ and $\uparrow_Q g$, respectively. The vectors H^Q and G^Q contain the Fourier coefficients of the upsampled filters. In particular, if $j = 0$, that is, $Q = 1$, H^1 equals the DFT vector H of h.

Note that the operations (10) are all just successive multiplications of frequency domain vectors: there are no intermediate FFTs or IFFTs between octaves. After all octaves have been computed, the time domain vectors d^1, d^2, ..., d^J, c^J may be obtained by taking N-point IFFTs of $D^1, D^2, \ldots, D^J, C^J$, respectively.

Let us consider the elements of the DFT vector H^Q in more detail.

$$\begin{aligned} H^Q(k) &= \sum_{n=0}^{N-1} (\uparrow_Q h)(n)\, e^{-\frac{2\pi\, i\, k\, n}{N}} \\ &= \sum_{m=0}^{M-1} h(m)\, e^{-\frac{2\pi\, i\, k\, m\, Q}{N}} = H(k\, Q), \end{aligned} \tag{11}$$

for $k = 0, \ldots, N-1$. Here the filter h is assumed to have length M. Note that the index k in (11) runs from 0 to $N-1$, independent of level j, *i.e.*, the filter length stays constant. For example, for $j = 1$ (*i.e.*, $Q = 2$):

$$H^2 = [H(0)\, H(2)\, \ldots H(N-2) \quad H(0)\, H(2)\, \ldots H(N-2)]$$

In general,

$$H^Q = \overbrace{\left[(\downarrow_{2^j} H)\ (\downarrow_{2^j} H) \cdots (\downarrow_{2^j} H) \right]}^{Q\ times}$$

So, in iteration j of the decomposition, the DFT vector H^Q is obtained by down-sampling the DFT vector H by a factor $Q = 2^j$, and then Q times repeating this reduced vector of length N/Q to again get a filter of length N. Alternatively, two copies of the even-numbered samples of the filter values in the previous iteration $j - 1$ (*i.e.*, $H^{Q/2}$) are concatenated to obtain H^Q. The case of G^Q is analogous.

From (8), reconstruction in the Fourier domain is obtained by

$$C^{j-1} = H'^Q \bullet C^j + G'^Q \bullet D^j,$$

where H'^Q and G'^Q are obtained in the same way from h' and g' as described above for H^Q.

This frequency-domain implementation of the SI-DWT does not use the polyphase decomposition, which at higher levels of the wavelet decomposition substantially reduces the number of required computations. Matlab code of the SI-DWT and SI-IDWT routines is given in Algorithm 1.

For separable filters, generalisation of this algorithm to higher dimensions follows directly from the time-domain version and the convolution property. In the 2D case, the input C is a 2D frequency spectrum, and the 4 filters HH, HG, GH, and GG are the tensor products of the 1D filters. Filtering becomes multiplication, and higher-level filters are made by subsampling the originals by a factor $1/Q \times 1/Q$ and repeating the subsampled filter $Q \times Q$ times. For every next level, each of the filters is applied to the current approximation.

Algorithm 1. *Frequency-domain implementation of SI-DWT and SI-IDWT in Matlab code*

Frequency-domain SI-DWT

$$\left.\begin{array}{l}\textbf{Inputs}\text{: level } J\text{, signal } S\text{, filters } H\text{, } G\text{;}\\[2em] \textbf{Outputs}\text{: approx. } C\text{, detail } D(j,:)\ (j=1,2,\ldots,J)\end{array}\right\}\begin{array}{l}\text{all signals and filters}\\[2em]\text{are DFT vectors}\end{array}$$

```
1: len=length($S$);
2: $C$=$S$;
3: for $j$ = (1:$J$)
4:   $C$ = $H$ .* $C$;
5:   $D(j,:)$ = $G$ .* $C$;
6:   $H$ = [$H$(1:2:len) $H$(1:2:len)];
7:   $G$ = [$G$(1:2:len) $G$(1:2:len)];
8: end;
```

Frequency-domain SI-IDWT

$$\left.\begin{array}{l}\textbf{Inputs}\text{: level } J\text{, approx. } C\text{, details } D(j,:)\text{, dual filters } H'\text{, } G'\\[2em] \textbf{Outputs}\text{: reconstruction } R \qquad\qquad (j=1,2,\ldots,J)\end{array}\right\}\begin{array}{l}\text{all signals and filters}\\[2em]\text{are DFT vectors}\end{array}$$

```
1: len=length($C$);
2: $R$=$C$;
3: for $j$ = ($J$:-1:1)
4:   $Q$ = 2^(j-1);
5:   $H'_s$ = repmat ( $H'$(1:$Q$:len), 1, $Q$ );
6:   $G'_s$ = repmat ( $G'$(1:$Q$:len), 1, $Q$ );
7:   $R$ = $H'_s$ .* $R$ + $G'_s$ .* $D(j,:)$;
8: end;
```

Special symbols: u **.*** v: pointwise multiplication of u and v;

repmat(A,m,n): m-by-n tiling of copies of A.

4.3 Fourier-domain polyphase filtering

Another method is based on a frequency domain version of (9):

$$C^{j+1,q} = H \bullet C^{j,q}, \quad D^{j+1,q} = G \bullet C^{j,q} \tag{12}$$

for $j = 0, 1, \ldots, J-1$, $q = 0, 1, \ldots, Q-1$, $Q = 2^j$. The notation is as in the previous subsection, i.e., $C^{j,q}$, $D^{j,q}$, H and G are the DFT vectors of $c^{j,q}$, $d^{j,q}$, h and g, respectively, all of length N/Q.

The algorithm starts by an initial FFT of the input signal c^0, and from then on perform all operations (polyphase transform, filtering, and monophase reconstruction) in the frequency domain. At each octave j, the steps are:

(1) Compute $\{C^{j,q}\} =$ FPD(C^j), $q = 0, 1, \ldots, Q-1$,
(FPD denotes frequency domain polyphase decomposition)

(2) Compute $C^{j+1,q}$ and $D^{j+1,q}$ for all q by (12)

(3) Compute $C^{j+1} =$ FMR$(\{C^{j+1,q}\})$ and $D^{j+1} =$ FMR$(\{D^{j+1,q}\})$,
(FMR denotes frequency domain monophase reconstruction)

The frequency domain polyphase and monophase transforms are based on the formulas of section 2.2. In contrast to the direct method, the explicit polyphase transform in this algorithm needs to process the entire signal for every phase component, making it computationally expensive when the downsampling factor Q is large (at high levels of decomposition). See section 5 for more details.

5 Complexity analysis

We consider the time complexity of the algorithms described above, in the case where the filter length L is the same order as the signal length $N = 2^m$, and J is the number of octaves (decomposition levels). We express the arithmetic complexity in terms of the total number of flops, i.e., real multiplications and additions.

5.1 Time domain

A direct implementation in the time domain which avoids multiplication by zero coefficients can be based on the polyphase decomposition (9). This requires $2L-1$ operations (L multiplications and $L-1$ additions) for each filter on each octave, i.e., $2(2L-1)\,N$ operations per octave. The total complexity is

$$2J(2L-1)\,N.$$

For a number of octaves $J = \log_2 N$ and $L = \mathcal{O}(N)$, the asymptotic complexity for the time-domain SI-DWT is $4N^2 \log_2 N$.

5.2 Rioul -Duhamel method

The complexity of the RD algorithm can be found from the formulas in [7, p. 582, eq. (58)]. To enable comparison with our periodic frequency domain implementations, we note that in the periodic case the FFT-length N used in the RD algorithm equals the block length B, so that the denominators in eq. (58) of [7] equal 2^m. This means that the total number of flops per octave for N points equals $6 \cdot 2^m(m-1) + 18$. For a total number of J octaves we thus find ($N = 2^m$) that the complexity equals

$$6J(N(\log_2 N - 1) + 3).$$

For a number of octaves $J = \log_2 N$, the asymptotic complexity for the RD method is $6N(\log_2 N)^2$.

5.3 Frequency domain polyphase filtering

For octave j, the number of operations is as follows. The FPD needs to process Q blocks of length N/Q. Each block element $X^{Q,q}(k)$ in (5) requires Q complex multiplications and $Q - 1$ additions for computing the sum, followed by one complex and one real multiplication. Since each complex multiplication (addition) requires six (two) real operations, the total number of flops per block element is $8Q + 5$. This makes $Q \cdot N/Q \cdot (8Q + 5) = N \cdot (8Q + 5)$ operations in total. The filters have length N/Q and there are Q blocks, this makes N complex multiplications or $6N$ real operations per filter, so $12N$ operations in total. Finally, the FMR (6) again has Q blocks of length N/Q, each requiring Q complex multiplications and $Q - 1$ additions, hence $N\,(8Q - 2)$ real operations. In total that is (remember that $Q = 2^j$) $15N + 16N \cdot 2^j$ operations per octave j. Summing this over $j = 0, 1, \ldots J - 1$ octaves gives a complexity of $15N\,J + 16N\,(2^J - 1)$.

The get the total complexity, we have to add the operation count of the initial FFT of the input c^0, and $J+1$ IFFTs of the outputs $D^1, D^2, \ldots, D^J, C^J$ in the frequency domain. The operation count in [7], i.e., for a split-radix FFT of a real sequence of length N, is $2N(\log_2 N - 2) + 6$. The same holds for a complex sequence with complex conjugate symmetry, which applies to $D^1, D^2, \ldots, D^J, C^J$, since these are multiplications of DFTs of real signals. The FFTs thus require a total of $(J + 2)(2N(\log_2 N - 2) + 6)$ flops.

In summary, this implementation has an overall complexity of

$$\begin{aligned}&(J+2)(2N(\log_2 N-2)+6)+15N\,J+16N\,(2^J-1)\\ &=2(J+2)(N\log_2 N+3)+N(11J-24+2^{J+4}).\end{aligned}$$

For fixed J, the asymptotic complexity is $\mathcal{O}(N\log_2 N)$ in the signal length. However, it is clear that the operation count increases exponentially as the number J of octaves increases. When $J=\log_2 N$, the asymptotic complexity is $\mathcal{O}(N^2)$, making this implementation less attractive than the direct method. However, for low decomposition levels and very long filters ($L\sim N$) this algorithm is still more efficient than the time domain implementation.

5.4 Direct frequency domain filtering

Two length N filters are applied at each octave, both consisting of N complex multiplications. This makes $12N$ real operations per octave and a total of $12J\,N$ operations summed over all octaves. Again the complexity of the initial and final FFTs, i.e., $(J+2)(2N(\log_2 N-2)+6)$ flops, needs to be added, giving a total of

$$\begin{aligned}&(J+2)(2N(\log_2 N-2)+6)+12N\,J\\ &=2(J+2)(N\log_2 N+3)+8N(J-1).\end{aligned}$$

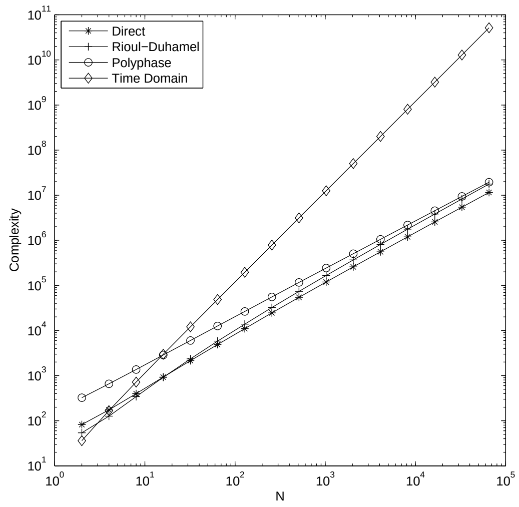
This implementation is clearly much more efficient than the frequency domain polyphase filtering method. It is also more efficient than the time domain implementation for large filter size L, *i.e.*, of the order of N, where the complexity of the convolution becomes quadratic in N. Frequency domain convolution is unaffected, since it does not depend on the filter length.

The direct algorithm has the same asymptotic complexity $\mathcal{O}(N(\log_2 N)^2)$ as the

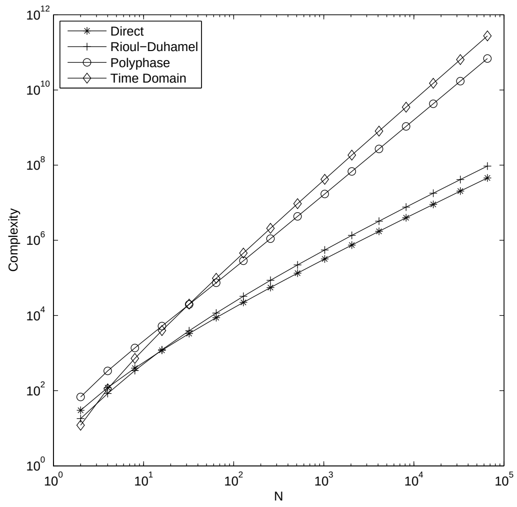
RD method, but with a smaller constant: for large N the speed gain is a factor of $3J/(J+2)$. For a number of octaves $J = \log_2 N$, the asymptotic complexity of the direct algorithm is $2N(\log_2 N)^2$, *i.e.*, a factor of 3 smaller than the RD algorithm.

5.5 Summary

Table 1 shows the complexities of all algorithms, for filter length N. If the decomposition level J is fixed, the time domain implementation of the polyphase decomposition is quadratic in N, while the frequency domain implementations are both of order $N \log_2 N$. The direct algorithm is much more efficient than the polyphase implementation, especially for large J. The direct algorithm is also faster than the RD algorithm: for large N the speed gain equals $3J/(J+2)$. For the highest possible decomposition level, i.e., $J = \log_2 N$, the polyphase frequency domain algorithm has complexity $\mathcal{O}(N^2)$, while the direct and RD algorithms both have $\mathcal{O}(N(\log_2 N)^2)$, the direct algorithm in turn being three times faster than the RD algorithm.



(a) $J = 3$



(b) $J = \log_2 N$

Fig. 2. *Doubly logarithmic plot of the complexity of the time-domain implementation (with $L = N$) and frequency domain implementations of the SI-DWT.*

Table 1

Arithmetical complexity of the SI-DWT for the time domain (TD) and frequency domain methods (Direct, Polyphase, RD) expressed in terms of the total number of floating points operations. Here N is the signal length, L the filter length, J the number of octaves. The third column ('Asymptotic') contains the asymptotic number of flops when $L = N$, $J = \log_2 N$, with N very large.

Method	Nr of flops	Asymptotic
TD	$2J(2L-1)\,N$	$4N^2\log_2 N$
Direct	$2(J+2)(N\log_2 N+3)+8N(J-1)$	$2N(\log_2 N)^2$
Polyphase	$2(J+2)(N\log_2 N+3)+N(11J-24+2^{J+4})$	$16N^2$
RD	$6J(N(\log_2 N-1)+3)$	$6N(\log_2 N)^2$

A log-log plot of the complexity as a function of the input length for decomposition level $J = 3$ and filter length $L = N$ is shown in Fig. 2. The frequency domain algorithms all scale similarly for large N, and the direct method is fastest. The quadratic scaling of the time-domain implementation is clearly evident from the larger slope of the plot.

5.6 Experimental results

We implemented our direct and polyphase frequency-domain versions of the SI-DWT as Matlab MEX-routines and compared the computation times with those of the time-domain implementation provided by the Rice Wavelet Toolbox. No implementation of the RD algorithm was available to us, therefore this method was excluded from the experiments. The test was done for a four-level SI-DWT, once with the Daubechies-4 filter [12], and once with the symmetric orthogonal cubic spline wavelet filter [5] (the support of orthogonal spline wavelet filters has the same size as the signal). For each signal length, 16384 (2^{14}) signals were decomposed and reconstructed. The computation times we obtained were in excellent agreement with the theoretical complexity estimates. For the Daubechies-4 filter, all algorithms scale linearly in N, but the polyphase implementation is significantly slower. For the cubic spline wavelet, the time-domain implementation scales quadratically, and the direct algorithm is again much faster than the polyphase implementation.

6 Conclusion

We have analysed the implementation of the periodic shift-invariant wavelet transform (SI-DWT) and its inverse in the time and frequency domain. We have described two frequency domain implementations, one based on explicit polyphase decompositions carried out entirely in the frequency domain (the 'polyphase' implementation), and one that employs all upsampling and downsampling of signals, for arbitrary sampling factors and signal shifts, in the frequency domain (the 'direct' implementation). Both methods only use one initial FFT and one final IFFT, all other steps consist of simple copying and multiplication of matrix elements. The implementation of the direct algorithm is very simple; explicit (Matlab-like) pseudo-code has been presented.

We performed a complexity analysis of our algorithms, comparing them to the algorithm by Rioul and Duhamel (RD method), which performs the convolution steps of the SI-DWT in the Fourier domain, while computing the downsampling and shift operations in the time domain, for all octaves of the wavelet decomposition [7]. We found that for long filter lengths (of the order of the signal length) the 'direct' and RD algorithms are the most efficient, both being of order $\mathcal{O}(N \log_2 N)$. In addition, the direct algorithm is faster than the RD algorithm: for large N the speed gain equals $3J/(J+2)$. For high number of octaves, i.e., $J = \log_2 N$, the polyphase frequency domain algorithm has complexity $\mathcal{O}(N^2)$, while the direct and RD algorithms both are of order $\mathcal{O}(N(\log_2 N)^2)$, the direct algorithm being three times as fast as the RD algorithm. In applications like the analysis of fMRI data, where the SI-DWT transform is performed millions of times, a significant speedup is achieved by using the direct algorithm.

7 Acknowledgements

This research was part of the project 'Wavelets and their applications', funded by the Dutch National Science Foundation (NWO), project no. 613.006.570.

References

[1] H. Y. Chuang, D. P. Birch, L.-C. Liu, J.-C. Chien, S. P. Levitan, C. C. Li, A high speed shift-invariant wavelet transform chip for video compression, in: IEEE Computer Society Annual Symposium on VLSI, April 25 - 26, 2002.

[2] R. R. Coifman, D. L. Donoho, Translation-invariant denoising, in: A. Antoniadis, G. Oppenheim (eds.), Wavelet and Statistics, Lecture Notes in Statistics, Springer, 1995, pp. pp. 125–150.

[3] M. Holschneider, R. Kronland-Martinet, J. Morlet, P. Tchamitchian, A real-time algorithm for signal analysis with the help of the wavelet transform, in: J. M. Combes, A. Grossmann, P. Tchamitchian (eds.), Wavelets, Time-Frequency Methods and Phase Space, 1989.

[4] N. G. Kingsbury, Complex wavelets for shift invariant analysis and filtering of signals, Journal of Applied and Computational Harmonic Analysis 10 (3) (2001) 234–253.

[5] S. G. Mallat, A theory for multiresolution signal decomposition: The wavelet representation, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (7) (1989) 674–693.

[6] J. Neumann, G. Steidl, Dual–tree complex wavelet transform in the frequency domain and an application to signal classification, International Journal of Wavelets, Multiresolution and Information Processing 3 (1) (2005) 1–23.

[7] O. Rioul, P. Duhamel, Fast algorithms for discrete and continuous wavelet transforms, IEEE Transactions on Information Theory 38 (2) (1992) 569–486.

[8] M. J. Shensa, The discrete wavelet transform: wedding the *à trous* and Mallat algorithms, IEEE Transactions on Signal Processing 40 (10) (1992) 2464–2482.

[9] G. Strang, T. Nguyen, Wavelets and Filter Banks, Wellesley-Cambridge Press, 1996.

[10] P. Vaidyanathan, Multirate systems and filter banks, Prentice Hall, Englewood Cliffs, NJ, USA, 1993.

[11] M. Vetterli, C. Herley, Wavelets and filter banks: Relationships and new results, in: Proc. Int. Conf. Acoust., Speech, Signal Processing, Albuquerque, NM, 1990.

[12] M. Vetterli, C. Herley, Wavelets and filter banks: Theory and design, IEEE Transactions on Signal Processing 40 (9) (1992) 2207–2232.

[13] M. Vetterli, J. Kovačević, Wavelets and Subband Coding, Prentice Hall, Englewood Cliffs, NJ, 1995.

[14] M. A. Westenberg, J. B. T. M. Roerdink, Frequency domain volume rendering by the wavelet X-ray transform, IEEE Transactions on Image Processing 9 (7) (2000) 1249–1261.

[15] A. M. Wink, H. Hoogduin, J. B. T. M. Roerdink, Data-driven haemodynamic response function extraction using Fourier-wavelet regularised deconvolution, BMC Medical Imaging (8:7), doi:10.1186/1471-2342-8-7.
URL `http://www.biomedcentral.com/1471-2342/8/7`

Alle Meije Wink received the M. Sc. (1999) in computing science from the University of Twente, The Netherlands, and a Ph. D. (2004) in computing science, with a focus on fMRI analysis, from the University of Groningen, The Netherlands. He has worked as a scientific programmer (1999) at the University Medical Centre Groningen, and as a research associate at the Neuroimaging Centre Groningen (2004) and the Brain Mapping Unit, University of Cambridge, United Kingdom (2004–2007). He is currently an fMRI scientist at the Imaging Sciences Department, Imperial College, London, United Kingdom. His research interests include image analysis methods and functional imaging methodology.

Jos B. T. M. Roerdink received his M.Sc. (1979) in theoretical physics from the University of Nijmegen, the Netherlands. Following his Ph. D. (1983) from the University of Utrecht and a two-year position (1983-1985) as a Postdoctoral Fellow at the University of California, San Diego, both in the area of stochastic processes, he joined the Centre for Mathematics and Computer Science in Amsterdam. There he worked from 1986-1992 on image processing and tomographic reconstruction. He was appointed associate professor (1992) and full professor (2003), respectively, at the Institute for Mathematics and Computing Science of the University of Groningen, where he currently holds a chair in Scientific Visualization and Computer Graphics. His current research interests include biomedical visualization, neuroimaging and bioinformatics.