EFFICIENT ALGORITHMS FOR THE DISCRETE GABOR TRANSFORM WITH A LONG FIR WINDOW

PETER L. SØNDERGAARD

ABSTRACT. The Discrete Gabor Transform (DGT) is the most commonly used signal transform for signal analysis and synthesis using a linear frequency scale. The development of the Linear Time-Frequency Analysis Toolbox (LTFAT) has been based on a detailed study of many variants of the relevant algorithms. As a side result of these systematic developments of the subject, two new methods are presented here. Comparisons are made with respect to the computational complexity, and the running time of optimised implementations in the C programming language. The new algorithms have the lowest known computational complexity and running time when a long FIR window is used. The implementations are freely available for download. By summarizing general background information on the state of the art, this article can also be seen as a research survey, sharing with the readers experience in the numerical work in Gabor analysis.

Key words and phrases : Discrete Gabor Transform, algorithm, implementation

2010 AMS Mathematics Subject Classification — 42C15, 42C40

1. INTRODUCTION

The finite, discrete Gabor transform (DGT) of a signal f of length L is given by

$$c(m,n,w) = \sum_{l=0}^{L-1} f(l,w)\overline{g(l-an)}e^{-2\pi iml/M}. \tag{1.1}$$

Usually, g is a window (filter prototype) that localizes the signal in time and in frequency, i.e. typically a real-valued, smooth function of good decay, symmetric around the origin (hence with a Fourier transform with similar properties), but the definition of the DGT imposes no rescrictions on g. This property has been used to lower the computational complexity of sparse compression techniques [28, ?]. The DGT is equivalent to a Fourier modulated filter bank with M channels and decimation in time a [7], and it is a valuable tool for time-frequency analysis when a linear frequency scale is desired. Because of its popularity, it has many names, including the *windowed Fourier transform* or the *short-time Fourier transform*. In this paper, we will exclusively refer to it as the DGT, and use the name *the short-time Fourier transform* (STFT) to refer to the DGT with $a = 1$ and $M = L$. A popular choice for the window g is a finite impulse response (FIR) window that is well localised in the time-frequency plane. As all windows that we will use in this paper are technically FIR, we will use the term to refer to windows which have a significantly shorter support than the transform length.

Many algorithms have been proposed for the computation of the DGT. There are three structures in the DGT with an FIR window which are typically exploited by algorithms:

(1) Coefficients are regularly spaced in time
(2) Coefficients are regularly spaced in frequency
(3) The window is FIR

The discovery of the fast Fourier transform (FFT) [12] made it possible to exploit the time- and frequency structure of the DGT. A classical method to compute the DGT is to window a signal in the time domain, and then compute an FFT of the windowed samples. This algorithm is commonly known as weighted overlap-add [34], and it exploits the FIR and frequency structure of the DGT. It is possible to improve upon this simple algorithm using Poisson summation, which was first done by Portnoff [29]. The Portnoff algorithm does not exploit the time-structure of the DGT, and therefore it can be used to compute DGTs which are not structured in time, so-called non-stationary Gabor transforms [4].

Another approach is to view the DGT as a filter bank, and then compute each subband signal using the overlap-add algorithm [21, 36]. This algorithm exploits the time and FIR structure, but not the frequency structure. It is not as computationally efficient as the Portnoff algorithm, but because it does not exploit the frequency structure, it can be used for all filter banks that have a regular sampling in time, so-called uniform filter banks [8]. Many of these early DGT algorithms were discovered in search for an efficient method to compute the phase vocoder [17]. A unified view of these algorithms and the extension to DGT synthesis was presented in [2].

A third approach would be to simultaneously exploit the time- and frequency structure of the DGT. For a non-redundant Gabor transform ($a = M$ in (1.1)) this can be done efficiently by a Zak-transform of the signal and window [3]. In the case when the window and signal have the same length, a factorization of the frame operator matrix was found by Zibulski and Zeevi in [41]. The method was initially developed in the $L^2(\mathbb{R})$ setting, and was adapted for the finite, discrete setting by Bastiaans and Geilen in [5]. They extended it to also cover the analysis/synthesis operator. A simple, but not so efficient, method was developed for the Gabor analysis/synthesis operator by Prinz in [30]. Strohmer [37] improved the method and obtained the lowest known computational complexity for computing the Gabor *frame* operator, but Strohmer did not provide a method to carry out the actual computation of the DGT. In this paper we extend Strohmer's method to also cover the Gabor *analysis* and *synthesis* operators providing a fast way to compute the DGT. We shall refer to this algorithm as the *factorization* algorithm. These methods do not exploit the FIR property of the window, meaning that they can be used to efficiently calculate transforms in which the signal and window have the same length. There are several advantages of using long windows: it is possible to do efficient signal processing without worrying about errors introduced by truncating a window and it is possible to use windows that have a slow decay in the time domain. As an example, non-compactly supported windows like the Gaussian and its tight and dual windows can be used without truncation. While the time-windowing and OLA algorithms are part of the curriculum of a typical course on digital signal processing, the combined time-frequency algorithms are far less well-known.

The aforementioned algorithms all exploit two out of the three structures of the DGT with an FIR window. In this paper we present an algorithm that exploits all three structures, by using the factorization algorithm within an overlap-add algorithm. The result is an algorithm that can efficiently handle long windows and block-processing of signals.

The factorization algorithm uses fewer floating point operations than the algorithm presented in [5], although they have the same asymptotic computational complexity, $O(NM\log M)$, where M is the number of channels and N is the number of time steps. The overlap-add-factorization algorithm has a linear running time with respect to signal length. A more accurate flop count is presented later in the paper.

In Section 2 we present the mathematical definitions used in the rest of the paper. In Section 3 we derive the Portnoff and factorization algorithm, and the extension by overlap-add is presented in Section 4. In Section 5 we discuss how to adapt the algorithms for real-valued signals and windows. In Section 6 we present extensions and specializations of the factorization algorithm. Section 7 presents a detailed analysis of the computational complexity of each algorithm, and Section 8 deals with the implementation issues presenting actual timings of the algorithms on a standard desktop computer.

2. Definitions

To make the formulas in his paper shorter and more readable, we shall denote the set of integers between zero and some number L by $\langle L\rangle = 0,\ldots,L-1$. We use the "$\cdot$" notation in conjunction with the DFT to denote the variable over which the transform is to be applied. Another related notation used in this paper is the ":" notation used to denote all elements indexed by a variable. As an example, if $C\in\mathbb{C}^{M\times N}$ then $C_{:,1}$ is a $M\times 1$ column vector, $C_{1,:}$ is a $1\times N$ row vector and $C_{:,:}$ is the full matrix. This notation is commonly used in Matlab and Fortran programming and in some textbooks, [19].

In this paper, the unitary Discrete Fourier Transform (DFT) of a signal $f\in\mathbb{C}^L$ is defined by

$$(\mathcal{F}_L f)(k) = \frac{1}{\sqrt{L}}\sum_{l=0}^{L-1} f(l)e^{-2\pi ikl/L}. \tag{2.1}$$

The cyclic convolution $f*g$ of two functions $f,g\in\mathbb{C}^L$ and the involution f^* is given by

$$(f*g)(l) = \sum_{k=0}^{L-1} f(k)\,g(l-k)\,,\quad l\in\langle L\rangle \tag{2.2}$$

$$f^*(l) = \overline{f(-l)},\quad l\in\langle L\rangle\,. \tag{2.3}$$

The cross-correlation is given by convolution with an involuted function. It can be computed efficiently using the discrete Fourier transform:

$$(f * g^*)(l) = \sqrt{L}\mathcal{F}_L^{-1}\left((\mathcal{F}_L f)(\cdot)\overline{(\mathcal{F}_L g)(\cdot)}\right)(l). \tag{2.4}$$

In this setting the Poisson summation formula looks as follows

$$\mathcal{F}_M\left(\sum_{k=0}^{b-1} g(\cdot + kM)\right)(m) = \sqrt{b}\,(\mathcal{F}_L g)(mb), \tag{2.5}$$

where $g \in \mathbb{C}^L$, $L = Mb$ with $b, M \in \mathbb{N}$.

A family of vectors e_j, $j \in \langle J\rangle$ of length L is called a *frame* if constants $0 < A \leq B$ exist such that

$$A\left\|f\right\|^2 \leq \sum_{j=0}^{J-1} \left|\langle f, e_j\rangle\right|^2 \leq B\left\|f\right\|^2, \quad \forall f \in \mathbb{C}^L. \tag{2.6}$$

An equivalent condition is to require that the family is a set of generators for $\mathbb{C}^L$. The constants A and B are called lower and upper frame bounds. If $A = B$, the frame is called *tight.* If $J > L$, the frame is redundant (oversampled). Finite- and infinite dimensional frames are described in [11].

A finite, discrete *Gabor system* (g, a, M) is a family of vectors $g_{m,n} \in \mathbb{C}^L$ of the following form

$$g_{m,n}(l) = e^{2\pi i l m/M} g(l - na), \quad l \in \langle L\rangle \tag{2.7}$$

for $m \in \langle M\rangle$ and $n \in \langle N\rangle$ where $L = aN$ and $M/L \in \mathbb{N}$. A Gabor system that is also a frame is called a *Gabor frame.* The analysis operator $C_g : \mathbb{C}^L \mapsto \mathbb{C}^{M\times N}$ associated to a Gabor system (g, a, M) is the DGT given by (1.1). The Gabor synthesis operator $D_\gamma : \mathbb{C}^{M\times N} \mapsto \mathbb{C}^L$ associated to a Gabor system (γ, a, M) is given by

$$f(l) = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} c(m,n)\, e^{2\pi i m l/M}\gamma(l - an). \tag{2.8}$$

In (1.1), (2.7) and (2.8) it must hold that $L = Na = Mb$ for some $M, N \in \mathbb{N}$. Additionally, we define $c, d, p, q \in \mathbb{N}$ by

$$c = \gcd(a, M) \quad , \quad d = \gcd(b, N), \tag{2.9}$$

$$p = \frac{a}{c} = \frac{b}{d} \quad , \quad q = \frac{M}{c} = \frac{N}{d}, \tag{2.10}$$

where GCD denotes the greatest common divisor of two natural numbers. With these numbers, the *redundancy* of the transform can be written as $L/(ab) = q/p$, where q/p is an irreducible fraction. It holds that $L = cdpq$. The Gabor *frame operator* $S_g : \mathbb{C}^L \mapsto \mathbb{C}^L$ of a Gabor frame (g, a, M) is given by the composition of the analysis and synthesis operators $S_g = D_g C_g$. The Gabor frame operator is important because it can be used to find the *canonical dual window* $g^d = S_g^{-1} g$ and the *canonical tight window* $g^t = S_g^{-1/2} g$ of a Gabor frame. The canonical dual window is important because D_{g^d} is a left inverse of C_g, so the canonical dual window can be used to perfectly reconstruct a signal from its DGT coefficients. The canonical tight window gives perfect reconstruction if it is used for both analysis and

Algorithm 1 Window factorization

```
                    WFAC$(g,a,M)$

 (1) for $r=\langle c\rangle$ , $k=\langle p\rangle$, $l=\langle q\rangle$
 (2)     for $s=\langle d\rangle$
 (3)         $tmp\,(s) \leftarrow g\,(r+c\cdot(k\cdot q-l\cdot p+s\cdot p\cdot q \textbf{ mod } d\cdot p\cdot q))$
 (4)     end for
 (5)     $Phi\,(r,k,l,:) \leftarrow$DFT$(tmp)$
 (6) end for
 (7) return Phi
```

Algorithm 2 Discrete Gabor transform by factorization

```
                    DGT$(f,g,a,M)$
 (1) $Phi =$WFAC$(g,a,M)$
 (2) for $r=\langle c\rangle$
 (3)     for $k=\langle p\rangle$, $l=\langle q\rangle$, $w=\langle W\rangle$
 (4)         for $s=\langle d\rangle$
 (5)             $tmp\,(s) \leftarrow f\,(r+(k\cdot M+s\cdot p\cdot M-l\cdot h_a\cdot a \bmod \mathrm{L})\,,w)$
 (6)         end for
 (7)         $Psitmp\,(k,l+w\cdot q,\cdot) \leftarrow$DFT$(tmp)$
 (8)     end for
 (9)     for $s=\langle d\rangle$
(10)         $G \leftarrow Phi\,(:,:,r,s)$
(11)         $F \leftarrow Psitmp\,(:,:,s)$
(12)         $Ctmp\,(:,:,s) \leftarrow G^T\cdot F$
(13)     end for
(14)     for $u=\langle q\rangle$, $l=\langle q\rangle$, $w=\langle W\rangle$
(15)         $tmp \leftarrow$IDFT$(Ctmp\,(u,l+w\cdot q,:))$
(16)         for $s=\langle d\rangle$
(17)             $coef\,(r+l\cdot c,u+s\cdot q-l\cdot h_a \bmod N,w) \leftarrow tmp\,(s)$
(18)         end for
(19)     end for
(20) end for
(21) for $n=\langle N\rangle$,$w=\langle W\rangle$
(22)     $coef\,(:,n,w) \leftarrow$DFT$(coef\,(:,n,w))$
(23) end for
(24) return $coef$
```

synthesis. For more information on Gabor systems and properties of the operators C, D and S see [20, 15, 16].

We shall study the DGT applied to multiple signals at once. This is for instance a common subroutine in computing a multidimensional DGT. The DGT defined by (1.1) works on a multi-signal $f\in\mathbb{C}^{L\times W}$, where $W\in\mathbb{N}$ is the number of signals.

3. The factorization algorithm

To derive a faster DGT, one approach is to consider the analysis operator C_g as a matrix, and derive a faster algorithm through unitary matrix factorizations of this matrix [37, 31]. Another approach is to consider properties of the Zak transform [5]. The Zak transform method has the downside that values outside the fundamental domain of the Zak transform require an additional step to compute. In the present paper we have chosen to derive the algorithm by directly manipulating the sums in the definition of the DGT.

To find a more efficient algorithm than (1.1), the first step is to recognize that the summation and the modulation term in (1.1) can be expressed as a DFT:

$$c\left(m,n,w\right)=\sqrt{L}\mathcal{F}_L\left(f(\cdot,w)\overline{g\left(\cdot-an\right)}\right)(mb)\,. \tag{3.1}$$

If the window used is an FIR window, and if the length of the support of the window is equal to M, then (3.1) is the weighted overlap-add algorithm [34]. We can improve upon this because we do not need all the coefficients computed by the DFT appearing in (3.1), only every b'th coefficient. Therefore, we can rewrite by the Poisson summation formula (2.5):

$$\begin{aligned} c\left(m,n,w\right) &= \sqrt{M}\mathcal{F}_M\left(\sum_{\tilde{m}=0}^{b-1} f(\cdot+\tilde{m}M,w)\overline{g\left(\cdot+\tilde{m}M-an\right)}\right)(m) \\ &= \left(\mathcal{F}_M K\left(\cdot,n,w\right)\right)(m)\,, \end{aligned} \tag{3.2}$$

where

$$K\left(j,n,w\right)=\sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(j+\tilde{m}M,w\right)\overline{g}\left(j+\tilde{m}M-na\right), \tag{3.3}$$

for $j\in\langle M\rangle$ and $n\in\langle N\rangle$. From (3.2) it can be seen that computing the DGT of a signal f can be done by computing K followed by DFTs along the first dimension of K. This is the algorithm reported by Portnoff in [29]. The equation similar to (3.3) for the frame operator is known as the *Walnut representation* of the frame operator [39].

To further lower the complexity of the algorithm, we wish to express the summation in (3.3) as a cross-correlation.

We split the variable j into two new variables r and l using $j=r+lc$ with $r\in\langle c\rangle$, $l\in\langle q\rangle$ and introduce $h_a,h_M\in\mathbb{Z}$ such that the following is satisfied:

$$c=h_M M-h_a a. \tag{3.4}$$

This equation is known as Bézouts identity, and the two integers h_a and h_M that solve the equation can be found by the extended Euclidean algorithm for computing the GCD of a and M.

Using (3.4) and the splitting of j we can express (3.3) as

$$\begin{aligned} & K\left(r+lc,n,w\right) \\ (3.5) = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+lc+\tilde{m}M,w\right)\overline{g}\left(r+l\left(h_M M-h_a a\right)+\tilde{m}M-na\right) \\ (3.6) = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+lc+\tilde{m}M,w\right)\overline{g}\left(r+\left(\tilde{m}+lh_M\right)M-\left(n+lh_a\right)a\right) \end{aligned}$$

We substitute $\tilde{m}+lh_M$ by $\tilde{m}$ and $n+lh_a$ by n and get

$$\begin{aligned} & K\left(r+lc,n-lh_a,w\right) \\ (3.7) \quad = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+lc+\left(\tilde{m}-lh_M\right)M,w\right)\overline{g}\left(r+\tilde{m}M-na\right) \\ (3.8) \quad = & \sqrt{M}\sum_{\tilde{m}=0}^{b-1} f\left(r+\tilde{m}M+l\left(c-h_M M\right),w\right)\overline{g}\left(r+\tilde{m}M-na\right) \end{aligned}$$

We split $\tilde{m}=k+\tilde{s}p$ with $k\in\langle p\rangle$ and $\tilde{s}\in\langle d\rangle$ and $n=u+sq$ with $u\in\langle q\rangle$ and $s\in\langle d\rangle$ and use that $M=cq$, $a=cp$ and $c-h_M M=-h_a a$:

$$\begin{aligned} K\left(r+lc,u+sq-lh_a,w\right) \quad = \quad & \sqrt{M}\sum_{k=0}^{p-1}\sum_{\tilde{s}=0}^{d-1} f\left(r+kM+\tilde{s}pM-lh_a a,w\right)\times \\ (3.9) \qquad\qquad & \times\overline{g}\left(r+kM-ua+\left(\tilde{s}-s\right)pM\right) \end{aligned}$$

Having expressed the variables j, $\tilde{m}$, n using the variables r, s, $\tilde{s}$, k, l, u we have now indexed f using $\tilde{s}$ and g using $(\tilde{s}-s)$. This means that we can view the summation over $\tilde{s}$ as a cross-correlation, which can be efficiently computed using a DFT. If we define

$$(3.10) \qquad \Psi^f_{r,s}\left(k,l+wq\right)=\mathcal{F}_d f\left(r+kM+\cdot pM-lh_a a,w\right),$$

$$(3.11) \qquad \Phi^g_{r,s}\left(k,u\right)=\sqrt{M}\mathcal{F}_d g\left(r+kM+\cdot pM-ua\right),$$

then by using (2.4) we can write (3.9) as

$$\begin{aligned} & K\left(r+lc,u+\tilde{s}q-lh_a,w\right) \\ (3.12) \qquad = & \sqrt{d}\sum_{k=0}^{p-1}\mathcal{F}_d^{-1}\left(\Psi^f_{r,\cdot}\left(k,l+wq\right)\overline{\Phi^g_{r,\cdot}\left(k,u\right)}\right)\left(\tilde{s}\right) \\ (3.13) \qquad = & \sqrt{d}\mathcal{F}_d^{-1}\left(\sum_{k=0}^{p-1}\Psi^f_{r,\cdot}\left(k,l+wq\right)\overline{\Phi^g_{r,\cdot}\left(k,u\right)}\right)\left(\tilde{s}\right) \end{aligned}$$

If we consider $\Psi^f_{r,s}$ and $\Phi^g_{r,s}$ as matrices for each r and s, the sum over k in the last line can be written as matrix products. The factorization algorithm (Algorithm 2) follows from this.

4. Extension by overlap-add

The factorization algorithm requires all data to be available beforehand. However, if the window is finitely supported (FIR), then a convolution can be computed by an overlap-add (OLA) algorithm [36, 21]. The OLA algorithm is a classical algorithm to convolve a long signal with an FIR window by splitting the convolution into smaller convolutions, which can each be efficiently calculated by an FFT.

The OLA algorithm works by partitioning a system of length L into blocks of length L_b such that $L = L_b N_b$, where N_b is the number of blocks. The block length must be longer than the support of the window, $L_b > L_g$. To perform the computation we take a block of the input signal of length L_b and zero-extend it to length $L_x = L_b + L_g$, and compute the convolution the extended signal using the similarly extended window. Because of the zero-extension of the window and signal, the computed coefficients will not be affected by the periodic boundary conditions, and it is therefore possible to overlay and add the computed convolutions of length L_x together to form the complete convolution of length L. The downside to this is that the total length of the small convolutions is longer than the long convolution by a fraction of

$$\rho = \frac{L_x}{L_b} = \frac{L_g + L_b}{L_b}. \tag{4.1}$$

Therefore, a trade-off between the block length L_b and the window length L_g must be found, so that the block length is long enough for (4.1) to be close to one, but at the same time small enough to not impose a too long processing delay.

The total computational effort of the small FFTs is less than computing the large convolution by a direct approach, or by zero-extending the filter and performing a long FFT. More importantly for applications, the OLA algorithm can be performed in a block-processing setting, where the input data is continuously produced.

The DGT can be expressed using convolutions as

$$c(m,n) = e^{-2\pi imn/M} f * (M_{mb} g^*), \tag{4.2}$$

where M_{mb} is the modulation operator

$$(M_{mb} g)(l) = e^{2\pi ilmb/L} g(l), \quad l = \langle L \rangle. \tag{4.3}$$

When computing a DGT by the OLA algorithm, it is only necessary to compute the FFT of the input signal once as it can be reused for all channels. Furthermore, the Poisson summation formula can be used, as we only need a subsampling (by the hop-size a) of the IFFT. These are major reductions in computational complexity over the verbatim OLA algorithm applied to each channel. The flop count of this algorithm is listed in Table 1 under the label 'OLA'.

To use the OLA algorithm with Algorithm (2), one replaces the small convolutions in the OLA algorithm by computation of the DGT with an FIR window. In the rest of this paper, we refer to this algorithm as the *Fac-OLA* algorithm.

5. The DGT for real-valued signals

It is a common case that both the signal and the desired window for a DGT are real-valued. In this case, (3.3) is also real-valued and hence the

Algorithm 3 Canonical Gabor dual window

```
GABDUAL$(g, a, M)$
(1) $Phi$ =WFAC$(g, a, M)$
(2) for $r = \langle c \rangle$, $s = \langle d \rangle$
(3)     $G \leftarrow Phi\,(:, :, r, s)$
(4)     $Phi^d\,(:, :, r, s) \leftarrow \left(G \cdot G^T\right)^{-1} \cdot G$
(5) end for
(6) $g^d$ =IWFAC$\left(Phi^d, a, M\right)$
(7) return $g^d$
```

DGT coefficients along the frequency axis have the same symmetry as DFT coefficients of a real-valued signal. It is therefore common practice in applications to only calculate and store the DGT coefficients corresponding to the positive frequencies. This makes it possible to cut the memory usage and computational complexity in half, and still maintain a direct relationsship to the DGT. Another common strategy for treating real-valued signals efficiently [38] is to use an approach based on the discrete cosine or sine transforms [1], but the Poisson summation formula for these transforms differs from (2.5) [33] so the factorization algorithm cannot be used unmodified.

As can be seen from (3.2), computing the DGT for real-valued signals can be done by substituting the final DFT by a DFT that only calculates the coefficients corresponding to the positive frequencies, see [6, 9] for a simple algorithm.

For the Portnoff algorithm, this is all that is needed to get the full reduction in computational complexity, as (3.3) is calculated using only real arithmetic. For Algorithm (2), the aforementioned positive-frequency DFT can be used to speed up line 7 and 15 in the algorithm, and the loop over s in line 9 runs only through the integers $\langle d/2 + 1 \rangle$ (for even d).

6. Extensions and special cases

6.1. **The synthesis operator.**

The algorithm just described can also be used to calculate the synthesis operator D_γ. This is done by applying Algorithm 2 in the reverse order and inverting each line. The only lines that are not trivially invertible are lines 10-12, which become

```
10) $\Gamma \leftarrow Phi^d\,(:, :, r, s)$
(11) $C \leftarrow Ctmp\,(:, :, s)$
(12) $Psitmp\,(:, :, s) \leftarrow \Gamma \cdot C$
```

where the matrices $Phi^d\,(:, :, r, s)$ should be left inverses of the matrices $Phi\,(:, :, r, s)$ for each r and s, if perfect reconstruction is desired.

The matrices $Phi^d\,(:, :, r, s)$ can be computed by Algorithm 1 applied to a dual Gabor window γ of the Gabor frame (g, a, M). It also holds that all dual Gabor windows γ of a Gabor frame (g, a, M) must satisfy that $Phi^d\,(:, :, r, s)$ are left inverses of the matrices $Phi\,(:, :, r, s)$. This criterion was reported in [23, 24].

A special left-inverse in the *Moore-Penrose pseudo-inverse*. Taking the pseudo-inverses of $Phi\,(:, :, r, s)$ yields the factorization associated with the canonical dual window of (g, a, M) [10]. This is Algorithm 3. Taking the polar decomposition of each matrix in $\Phi^{g}_{r,s}$ yields a factorization of the canonical tight window (g, a, M). For more information on these methods, as well as iterative methods for computing the canonical dual/tight windows, see [25]. In [27] the authors have presented an efficient method for synthesis, where the analysis window must be a short FIR window, but where the dual system need not be explicitly calculated.

6.2. **Special cases.** We shall consider two special cases of the algorithm:

The first case is integer oversampling. When the redundancy is an integer then $p = 1$. Because of this we see that $c = a$ and $d = b$. This gives (3.4) the appearance

$$a = h_M q a - h_a a, \tag{6.1}$$

indicating that $h_M = 0$ and $h_a = -1$ solves the equation for all values of a and q. The algorithm can be simplified accordingly, and reduced to the well known Zak-transform algorithm for this case [22].

The second case is the short time Fourier transform. In this case $a = b = 1$, $M = N = L$, $c = d = 1$, $p = 1$, $q = L$ and as in the previous special case $h_M = 0$ and $h_a = -1$. In this case the algorithm can be reduced to a very simple and well-known algorithm for computing the STFT.

7. Computational complexity

When computing the flop (floating point operations) count of the algorithm, we will assume that a complex FFT of length M can be computed using $4M \log_2 M$ flops. A review of flop counts for FFT algorithms is presented in [26]. Table 1 shows the flop count for Algorithm 2 and compares it with the definition of the DGT (1.1), with the Portnoff algorithm (3.2), with the Zak-transform based algorithm published in [5], and with the factorization-OLA algorithm presented in Section (4). When computing the flop count, we assume that both the window and signal are complex valued, except for the transforms for real signals, where both the signal and window must be real-valued.

The flop count for definition (1.1) is that of a complex matrix-vector multiplication. All the other algorithms share the $4MN \log_2 M$ term coming from the application of an FFT to each block of coefficients and only differ in how (3.3) is computed. The Portnoff algorithm is very fast for a small overlapping factor L_g/a, but turns into an $\mathcal{O}\left(L^2\right)$ algorithm for a full length window. The term $L\left(8q + 1 + \frac{q}{p}\right)$ in the [5] algorithm comes from calculation of the needed Zak-transforms, and the $4L\left(1 + \frac{q}{p}\right)\log_2 N$ term comes from the transform to and from the Zak-domain. Compared to (3.10) and (3.11) this transformation uses longer FFTs. The factorization algorithm does away with the multiplication with complex exponentials in the algorithm described in [5], and so the first term reduces to $L\left(8q\right)$. For real-valued signals, the factorization algorithm uses half the flops of the complex valued version. The Portnoff algorithm sees an even greater improvement, as the computation of (3.3) can be done using only a quarter of the flops of the complex valued version.

TABLE 1. Flop counts for different ways of computing the DGT. First column list the algorithm, second column the flop count for the particular algorithm. The term L_g denotes the length of the window used so L_g/a is the overlapping factor of the window. Note for comparison that $\log_2 N = \log_2 d + \log_2 q$. For the OLA algorithms, ρ is given by (4.1). For the real-valued transforms, the linear algebra algorithm use 1/4 of the flops of the complex-valued transform, the Portnoff algorithm has the flop count specified in the table, and all other algorithms use half as many flops as their complex counterparts.

Alg.:	Flop count
Linear algebra. (1.1)	$8MNL$
Portnoff. (3.2)	$8L\frac{L_g}{a} + 4NM\log_2 M$
Portnoff, real	$2L\frac{L_g}{a} + 2NM\log_2 M$
Bastiaans et. al. [5]	$L\left(8q+1+\frac{q}{p}\right) + 4L\left(1+\frac{q}{p}\right)\log_2 N + 4MN\log_2(M)$
OLA. [21, 36]	$\rho\left(8LM + 4L\log_2(\rho L_b) + 4MN\log_2(\rho N)\right)$
Factorization. Alg. 2	$8Lq + 4L\left(1+\frac{q}{p}\right)\log_2 d + 4MN\log_2(M)$
Fac-OLA.	$\rho\left(8Lq + 4L\left(1+\frac{q}{p}\right)\log_2(\rho d) + 4MN\log_2 M\right)$

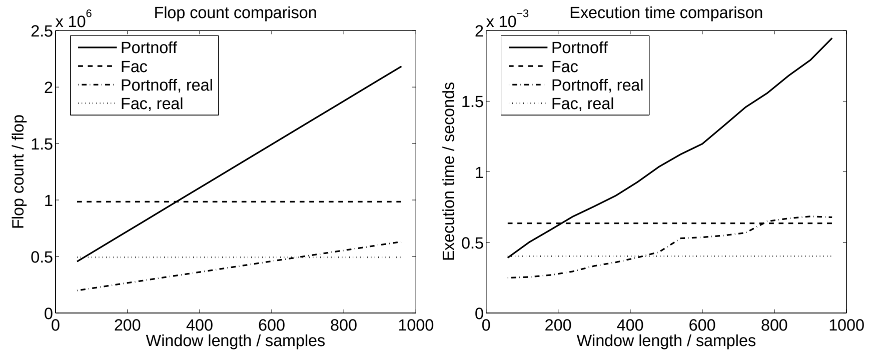

FIGURE 8.1. Comparison of the flop count and the actual running time of the factorization and Portnoff algorithms. The Gabor system used has a time shift of $a = 40$, $M = 60$ frequency channels, system length of $L = 1800$ and works on $W = 4$ signals at once. The flop count and running time are shown as functions of the window length L_g.

8. Implementation and timing

The reason for defining the factorization algorithm on multi-signals, is that several signals can be handled at once in the matrix product in line 12 of Algorithm 2. This is a matrix product of two matrices size $q \times p$ and $p \times qW$, so the second matrix grows when multiple signals are involved.

Doing it this way reuses the $\Phi^g_{r,s}$ matrices as much as possible, and this is an advantage on standard, general purpose computers with a deep memory hierarchy, see [14, 40]. It also makes it possible to reuse the results of the indexing computations done in the permutations.

The benefit of expressing Algorithm 2 in terms of loops (as opposed to using the Zak transform or matrix factorizations) is that they are easy to reorder. The algorithm presented as Algorithm 2 is just one among many possible algorithms depending on the order in which the r, s, k and l loops are executed. For a given platform, it is difficult a priory to estimate which ordering of the loops will turn out to be the fastest. The ordering of the loops presented in Algorithm 2 is the variant that uses the least amount of extra memory.

Implementations of the algorithms described in this paper can be found in the Linear Time Frequency Analysis Toolbox (LTFAT) available from `http://ltfat.sourceforge.net` (see also the paper [35]). An appropriate algorithm will be automatically invoked when calling the `dgt` or `dgtreal` functions. The implementations are done in both the MATLAB / OCTAVE scripting language and in C. A range of different variants of Algorithm 2 has been implemented and tested, and the one found to be the fastest on a small set of computers is included in the toolbox.

We have not created an efficient implementation of the Bastiaans & Geilen algorithm [5] in C, because it is always inferior to the factorization algorithm. Similarly, the OLA algorithm is always inferior to the Fac-OLA algorithm, so neither of these two algorithms have been timed.

A comparison of the flop count and running time of the Portnoff and factorization based algorithms are shown on Figure 8.1. From the figure it can be seen that for the particular setup, the factorization algorithm (for complex arithmetic) will be faster than the Portnoff algorithm if the window is longer than 250 samples, which is close to what is predicted by the flop-count. For real arithmetic, the cross-over point where the Portnoff algorithm becomes slower than the factorization algorithm appears later because the Portnoff algorithm gets a larger gain from using real arithmetic. All tests were done on an Intel Core2 CPU operating at 2.4 GHz.

The speed of the factorization algorithm is highly dependent on the speed of the underlying FFT library. A key issue is therefore to avoid FFT sizes for which computation of the FFT is slow. This is usually the case when the size of the FFT is prime or consists of a few large prime factors. In such cases, the regular Cooley-Tukey algorithm [12] cannot be used directly, and special algorithms [32] for prime sized factors must be used to reduce the problem size to a highly composite number. When timing the factorization algorithm, only FFT sizes of the form $2^a3^b5^c7^d$, where $a, b, c, d \in \mathbb{N}$ (such as $L = 44100 = 2^23^25^27^2$) have been used. This technique was also used in [18]. The result of running the factorization and Portnoff algorithm on increasing problem sizes is shown on Figure 8.2. For the chosen setup, the factorization algorithm is consistently faster than the Portnoff algorithm, and as the input signals become longer, the factorization-OLA algorithms starts to outperform the basic factorization algorithm.

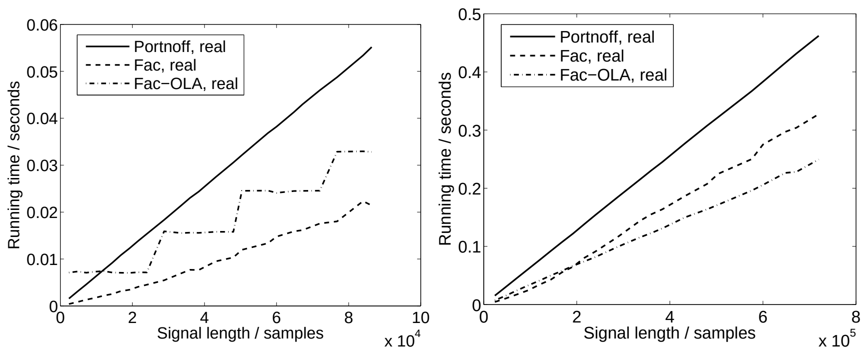

FIGURE 8.2. Comparison of the running time of the Portnoff, factorization and fac-OLA algorithms for real valued signals. The Gabor system used has a time shift of $a = 40$, $M = 60$ frequency channels, window length $L_g = 2400$, block length of $L_B = 24000$ and works on $W = 4$ signals at once. The figures on the left and right show the behaviour of short and long signals, respectively.

The cross-over point where one algorithm is faster than the other is highly dependent on the interplay between the algorithm and the computer architecture. Experience from the ATLAS [40], FFTW [18] and SPIRAL [13] projects, show that in order to have the highest performance, is it necessary to select the algorithm for a given problem size based on previous tests done on the very same machine. Performing such an optimization is beyond the scope of this paper, but will be the target of future work.

References

[1] N. Ahmed, T. Natarajan, and K. Rao. Discrete cosine transfom. *Computers, IEEE Transactions on*, 100(1):90–93, 1974.
[2] J. Allen and L. Rabiner. A unified approach to short-time Fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.
[3] L. Auslander, I. Gertner, and R. Tolimieri. The discrete Zak transform application to time-frequency analysis and synthesis of nonstationary signals. *IEEE Trans. Signal Process.*, 39(4):825–835, 1991.
[4] P. Balazs, M. Dörfler, N. Holighaus, F. Jaillet, and G. Velasco. Theory, Implementation and Application of Nonstationary Gabor Frames. *Journal of Computational and Applied Mathematics*, accepted for publication, 2011.
[5] M. J. Bastiaans and M. C. Geilen. On the discrete Gabor transform and the discrete Zak transform. *Signal Process.*, 49(3):151–166, 1996.
[6] G. Bergland. Numerical Analysis: A Fast Fourier Transform Algorithm for Real-valued Series. *Communications of the ACM*, 11(10):703–710, 1968.
[7] H. Bölcskei, F. Hlawatsch, and H. G. Feichtinger. Equivalence of DFT filter banks and Gabor expansions. In *SPIE 95, Wavelet Applications in Signal and Image Processing III*, volume 2569, part I, San Diego, july 1995.
[8] H. Bölcskei, F. Hlawatsch, and H. G. Feichtinger. Frame-theoretic analysis of oversampled filter banks. *Signal Processing, IEEE Transactions on*, 46(12):3256–3268, 2002.
[9] W. Briggs. *The DFT: an owner's manual for the discrete Fourier transform.* Society for Industrial Mathematics, 1995.
[10] O. Christensen. Frames and pseudo-inverses. *J. Math. Anal. Appl.*, 195:401–414, 1995.
[11] O. Christensen. *An Introduction to Frames and Riesz Bases.* Birkhäuser, 2003.

[12] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput*, 19(90):297–301, 1965.

[13] F. de Mesmay, Y. Voronenko, and M. Püschel. Offline library adaptation using automatically generated heuristics. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2010.

[14] J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16(1):1–17, 1990.

[15] H. G. Feichtinger and T. Strohmer, editors. *Gabor Analysis and Algorithms.* Birkhäuser, Boston, 1998.

[16] H. G. Feichtinger and T. Strohmer, editors. *Advances in Gabor Analysis.* Birkhäuser, 2003.

[17] J. L. Flanagan, D. Meinhart, R. Golden, and M. Sondhi. Phase Vocoder. *The Journal of the Acoustical Society of America*, 38:939, 1965.

[18] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[19] G. H. Golub and C. F. van Loan. *Matrix computations, third edition.* John Hopkins University Press, 1996.

[20] K. Gröchenig. *Foundations of Time-Frequency Analysis.* Birkhäuser, 2001.

[21] H. Helms. Fast Fourier transform method of computing difference equations and simulating filters. *IEEE Transactions on Audio and Electroacoustics*, 15(2):85–90, 1967.

[22] A. J. E. M. Janssen. The Zak transform: a signal transform for sampled time-continuous signals. *Philips Journal of Research*, 43(1):23–69, 1988.

[23] A. J. E. M. Janssen. On rationally oversampled Weyl-Heisenberg frames. *Signal Process.*, pages 239–245, 1995.

[24] A. J. E. M. Janssen. The duality condition for Weyl-Heisenberg frames. In Feichtinger and Strohmer [15], Chap. 1, pages 33–84.

[25] A. J. E. M. Janssen and P. L. Søndergaard. Iterative algorithms to approximate canonical Gabor windows: Computational aspects. *J. Fourier Anal. Appl.*, 13(2):211–241, 2007.

[26] S. Johnson and M. Frigo. A Modified Split-Radix FFT With Fewer Arithmetic Operations. *IEEE Trans. Signal Process.*, 55(1):111, 2007.

[27] S. Moreno-Picot, M. Arevalillo-Herráez, and W. Diaz-Villanueva. A linear cost algorithm to compute the discrete Gabor transform. *IEEE Trans. Signal Process.*, 58(5):2667–2674, 2010.

[28] G. Pfander, H. Rauhut, and J. Tanner. Identification of matrices having a sparse representation. *IEEE Trans. Signal Process.*, 56(11):5376–5388, 2008.

[29] M. Portnoff. Implementation of the digital phase vocoder using the fast Fourier transform. *IEEE Trans. Acoust. Speech Signal Process.*, 24(3):243–248, 1976.

[30] P. Prinz. Calculating the dual Gabor window for general sampling sets. *IEEE Trans. Signal Process.*, 44(8):2078–2082, 1996.

[31] S. Qiu. Discrete Gabor Transforms: The Gabor-Gram Matrix Approach. *J. Fourier Anal. Appl.*, 4(1):1–17, 1998.

[32] C. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.

[33] K. Rao and P. Yip. *Discrete Cosine Transform, Algorithms, Advantages, Applications.* Academic Press, 1990.

[34] R. Schafer and L. Rabiner. Design and Simulation of a Speech Analysis-Synthesis System based on Short-Time Fourier Analysis. *IEEE Transactions on Audio and Electroacoustics*, 21(3):165–174, 1973.

[35] P. L. Søndergaard, B. Torrésani, and P. Balazs. The Linear Time Frequency Analysis Toolbox. *International Journal of Wavelets, Multiresolution Analysis and Information Processing*, 10(4), 2012.

[36] T. Stockham Jr. High-speed convolution and correlation. In *Proceedings of the April 26-28, 1966, Spring joint computer conference*, pages 229–233. ACM, 1966.

[37] T. Strohmer. Numerical algorithms for discrete Gabor expansions. In Feichtinger and Strohmer [15], Chap. 8, pages 267–294.

[38] L. Tao and H. Kwan. Novel DCT-based real-valued discrete Gabor transform and its fast algorithms. *IEEE Trans. Signal Process.*, 57(6):2151–2164, 2009.

[39] D. F. Walnut. Continuity properties of the Gabor frame operator. *Journal of mathematical analysis and applications*, 165(2):479–504, 1992.

[40] R. C. Whaley and A. Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, February 2005.

[41] Y. Y. Zeevi and M. Zibulski. Oversampling in the Gabor scheme. *IEEE Trans. Signal Process.*, 41(8):2679–2687, 1993.

Department of Electrical Engineering, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark, ps@elektro.dtu.dk