SUBMITTED TO APJ
Preprint typeset using LaTeX style emulateapj v. 12/16/11

AN ACCURATE AND EFFICIENT ALGORITHM FOR DETECTION OF RADIO BURSTS WITH AN UNKNOWN DISPERSION MEASURE, FOR SINGLE DISH TELESCOPES AND INTERFEROMETERS

Barak Zackay and Eran O. Ofek
Benoziyo Center for Astrophysics, Weizmann Institute of Science, 76100 Rehovot, Israel
submitted to APJ

ABSTRACT

Astronomical radio bursts disperse while traveling through the interstellar medium. To optimally detect a short-duration signal within a frequency band, we have to precisely compensate for the pulse dispersion, which is a computationally demanding task. We present the Fast Dispersion Measure Transform (FDMT) algorithm for optimal detection of such signals. Our algorithm has a low theoretical complexity of $2N_fN_t + N_tN_d\log_2(N_f)$ where N_f, N_t and N_d are the numbers of frequency bins, time bins, and dispersion measure bins, respectively. Unlike previously suggested fast algorithms our algorithm conserves the sensitivity of brute force dedispersion. Our tests indicate that this algorithm, running on a standard desktop computer, and implemented in a high-level programming language, is already faster than the state of the art dedispersion codes running on graphical processing units (GPUs). We also present a variant of the algorithm that can be efficiently implemented on GPUs. The latter algorithm's computation and data transport requirements are similar to those of two-dimensional FFT, indicating that incoherent dedispersion can now be considered a non-issue while planning future surveys. We further present a fast algorithm for sensitive dedispersion of pulses shorter than normally allowed by incoherent dedispersion. In typical cases this algorithm is orders of magnitude faster than coherent dedispersion by convolution. We analyze the computational complexity of pulsed signal searches by radio interferometers. We conclude that, using our suggested algorithms, maximally sensitive blind searches for such pulses is feasible using existing facilities. We provide an implementation of these algorithms in Python and MATLAB.

1. INTRODUCTION

When a radio pulse propagates through the interstellar and intergalactic plasma, different frequencies travel at different speeds. This phenomenon, known as dispersion, hinders the detection of radio pulses. This is because integrating over many frequencies during a given time frame dilutes the signal with noise, as only a single frequency contributes signal at any given interval within the integration frame. The solution to this problem is to dedisperse the signal (i.e., to apply frequency dependent time delays to the signal prior to integration). Since in most cases the dispersion is a priori unknown, we need to test a large number of dispersions. The best dispesrion is the one that maximizes the signal-to-noise ratio of the pulse. A different way to look at this problem is that we need to integrate the flux along many dispersion curves in the frequency-time domain.

The difference in pulse arrival time between two frequencies is given by:

$$\Delta t = t_2 - t_1 = 4.15\mathrm{DM}(f_1^{-2} - f_2^{-2})\,\mathrm{ms}, \qquad (1)$$

where DM is called the dispersion measure of the signal and it is traditionally measured in units of $\mathrm{pc\,cm^{-3}}$. f_i are frequencies measured in GHz and t_i is the arrival time of the signal at frequency f_i. For brevity, throughout the paper, we will use d to denote the dispersion measure in which all the dimensional constants are absorbed, i.e., the relation is given by

$$\Delta t = t_2 - t_1 = d(f_1^{-2} - f_2^{-2}). \qquad (2)$$

bzackay@gmail.com
Eran.ofek@weizmann.ac.il

The raw input from a radio reciever is a time series of voltage measurments sampled typically at high frequency (e.g., $\sim 100\,\mathrm{MHz}$). We denote the sampling interval by τ. In order to generate a spectrum ($I[t,f]$) as a function of time (t) and frequency (f) the time series is divided to blocks of size N_f samples, and each block is then Fourier transformed (a process known as Short Time Fourier Transform, or STFT) and the absolute value squared at each frquency is saved[1].

There are two distinct processes that we can apply to dedisperse a signal: incoherent dedispersion and coherent dedispersion. The term incoherent dedeispersion refers to applying frequency-dependent time delays to the $I(t,f)$ matrix, while coherent dedispersion involves applying a frequency-dependent phase shift directly to the Fourier transform of the raw signal. This subtle difference is important – incoherent dedispersion is only an approximation that is valid under certain conditions that we are going to review shortly (§2). A typical input matrix ($I[t,f]$) to incoherent dedispersion is presented in the top panel of Figure 1, while on the bottom we show a zoom in on the output of the transform.

The exact mathematical description of signal dispersion is the multiplication of the Fourier transform of the raw signal with the phase-only filter

$$\hat{H}(f) = \exp\left(\frac{2\pi i d}{f + f_0}\right) \qquad (3)$$

(Lorimer & Kramer 2012) , where f_0 is the base-band frequency. In order to exactly dedisperse the signal, we

[1] Sometimes, an additional stage of binning is then applied to reduce the time resolution.

can apply the inverse of this shift. This process is referred as coherent dedispersion.

The computational requirements of incoherent dedispersion are more tractable than those of coherent dedispersion, and therefore whenever a blind search for astrophysical pulses is done, incoherent dedispersion is usually the method of choice (e.g. van Leeuwen & Stappers 2010). The main practical motivation for improving dedispersion algorithms is to allow efficient analysis of modern radio interferometers data. When a blind search for new sources is conducted using a multi-component radio interferometer, coherent dedispersion of a large number of synthesized beams is the most sensitive detection method, but usually this is unfeasible. The standard solution for this problem is to either combine the power from all antennas incoherently, or to use only a small core of the interferometer for blind searches. Both approaches result in a significant loss of sensitivity and angular resolution that can amount to an order of magnitude sensitivity loss. It is important to improve upon these methods, especially when searching for non-repeating radio transients such as fast radio bursts (FRBs; Lorimer et al. 2007; Thornton et al. 2013). Efficient detection of such objects requires both high sensitivity and a good spatial localization that is calculated in real time. This is crucial for the multi-wavelength followup of these illusive transients (e.g., Petroff et al. 2014).

In this paper, we present the Fast discrete Dispersion Measure Transform algorithm (henceforth FDMT) to solve the problem of incoherent dedispersion. FDMT is a transform algorithm, having (generally) equal sizes for both input and output, and complexity that is only logarithmically larger than the input itself.

In addition, we present a hybrid algorithm that achieves both the sensitivity of coherent dedispersion, and the computational efficiency of incoherent dedispersion. Finally, we show that using this algorithm, it is feasible to perform blind searches with modern radio interferometers, and consequently to open new frontiers in the search for pulsars and radio transients.

The structure of the paper is as follows. In §2 we analyze the sensitivity of incoherent dedispersion. In §3 we review the existing approaches for incoherent dedispersion. In §4 we describe the proposed algorithm, along with its complexity analysis. In §5 we present a variant of the algorithm that utilizes the fast Fourier transform to make the algorithm much more parallel friendly. In §6 we compare the runtime of the implementation we provide with existing implementations of brute force dedispersion. In §7 we propose a new hybrid algorithm for detection of pulses shorter than sensitively detectable by incoherent dedispersion. In §8 we discuss the application of the proposed algorithms for interferometers, and show that sensitive detection of short pulses, with maximal resolution, using all the elements of the interferometer, is feasible with current facilities. We conclude in §9.

2. SENSITIVITY ANALYSIS OF INCOHERENT DEDISPERSION

In this subsection, we develop the condition on the pulse length, the sampling interval and the dispersion delay that allows sensitive detection with incoherent dedispersion. This will be of importance in section §7.

We denote by x the raw voltage signal, and by N_s the

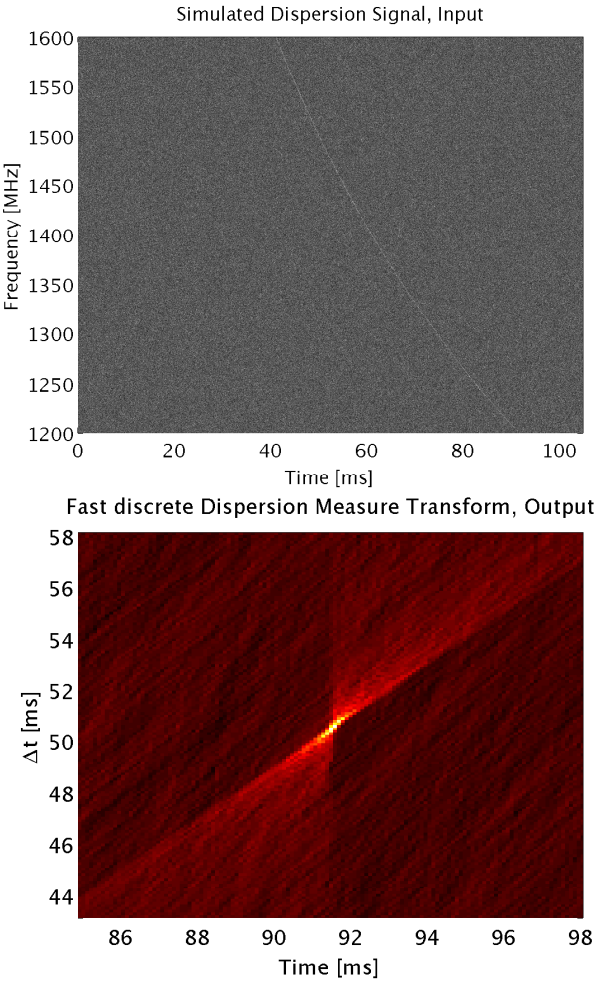

Figure 1. A dispersed signal and its dispersion transform (FDMT), based on simulated data. The top panel shows a 0.1ms wide dispersed pulse with $D = 40\,\mathrm{pc\,cm^{-3}}$. The bottom panel shows a zoom in on the significant part of the dedispersion transform. Notice that the Y axis has units of time because the dispersion measure is parametrized by the total time delay between the pulses entrance and exit. Note that because the pulse is so thin, any slight error on the dispersion path will immediately result in significant loss of the pulse power.

total number of samples. We further denote the pulse duty time (length) by $t_p = N_p\tau$, where N_p is the number of samples within the pulse. The optimal score for pulse detection is the sum of the squared voltage measurements within the pulse start time (t_0) and end time,

$$S_{\mathrm{opt}} = \sum_{j=0}^{N_p} x(t_0 + j)^2. \tag{4}$$

We further define the dispersion time t_d to be the total delay of the pulse within the band and we define N_d to be the dispersion time in units of samples, i.e. $t_d = N_d\tau$.

An important property of the dispersion kernel $H(f)$ is that it is power preserving ($|\hat{H}(f)|^2 = 1$). Another important property is that the majority of pulse power will lay within the dispersion curve in $I(t, f)$. By summing over the dispersion curve in I, we also sum the power of the noise outside the pulse, which its total variance is proportional to the number of added I bins–i.e., to the length of the dispersion curve. The total length of the

dispersion curve can be approximated by

$$\sim \max\left(1, \frac{N_p}{N_f}\right)\sqrt{{N_f}^2 + \frac{{N_d}^2}{{N_f}^2}}, \quad (5)$$

assuming the dispersion curve is close to linear.

Therefore, the ratio of the noise power summed by incoherent dedispersion and the noise power within the pulse (assuming $N_p < N_f$) is

$$\frac{\sqrt{{N_f}^2 + \frac{{N_d}^2}{{N_f}^2}}}{N_p}. \quad (6)$$

Immediately, we get that the choice of N_f that maximizes sensitivity is

$$N_f = \sqrt{N_d}, \quad (7)$$

regradless of pulse length N_p. In order for the sensitivity loss to be less than a factor of $\sqrt{2}$, we need both

$${N_d}^2 < {N_f}^2 {N_p}^2 \quad (8)$$

and

$${N_f}^2 < {N_p}^2. \quad (9)$$

This implies that

$$N_d < N_p N_f < {N_p}^2. \quad (10)$$

This transforms to an important relation between the minimal pulse duration t_p, the maximal dispersion time t_d, and the sampling time τ.

$$\frac{t_d}{\tau} < \frac{t_p^2}{\tau^2} \quad (11)$$

or, simplified,

$$t_p > \sqrt{t_d \tau}. \quad (12)$$

Using a standard order of magnitude value for τ and t_d, $\tau = 10^{-8}$ s (100 MHz sampling) and $t_d = 0.1$ s we get that $t_p > 10^{-4.5}$s. This means that incoherent dedispersion analysis of pulses shorter than about $10^{-4.5}$ s usually loses sensitivity relative to coherent dedispersion.

3. EXISTING ALGORITHMS FOR INCOHERENT DEDISPERSION

Algorithmically, there are two leading approaches for inchorerent dedispersion of single-dish data streams. These are the the tree dedispersion (Taylor 1974) and brute force dedispersion (e.g., Magro et al. 2011;Barsdell et al. 2012; Clarke et al. 2013).

The tree dedispersion algorithm, efficiently calculates the integrals of all the straight line paths with slopes between 45° and 90° through the input time vs. frequency matrix (this is similar to the discrete Radon transform, Gotz & Druckmuller 1996; Brady 1998). The computational complexity of this algorithm is $N_t N_f \log_2 N_f$, where we use the notation $N_t = N_s/N_f$ (note that N_t and N_f are the dimensions of $I(t, f)$). However, since the dispersion curve is not linear, the use of this method results in a substantial loss of sensitivity. This can be somewhat mitigated by applying the algorithm to many small sub-bands[2] of the data, and then combining the results with a dedicated algorithm. This approach is not exact, and it increases the computational complexity of the algorithm. For more details on the sensitivity analysis and computational complexity of this algorithm we refer readers to Barsdell et al. (2012).

The brute force dedispersion algorithm simply scans all the trial dispersion measures, one at a time, integrating its path on the input map and finding curves with excess power. This method is exact, but has the high complexity of $N_\Delta N_f N_t$, where N_Δ is the number of trial dispersion measures scanned (note that $N_\Delta = N_d/N_f$). In order to expedite the search speed, the algorithm was implemented on graphical processing units (GPUs), and this method is now capable of analyzing single beam data in real time (Barsdell et al. 2012). The maximal sensitivity, along with the possibility of real time analysis using GPUs, makes this method likely to be the most popular algorithm for dedispersion.

Here we present an algorithm that combines both maximal sensitivity and low computational complexity. A comparison of all these mentioned algorithms is summarized in Table 1.

4. THE FDMT ALGORITHM

The input to the FDMT algorithm is a two dimensional array of intensities, denoted by $I(t, f)$. The FDMT algorithm calculates the integral over all curves defined by Equation 2. A dispersion curve can be uniquely defined by the arrival time of the signal at the lowest frequency (t_0) and the time delay between the arrival times of the lowest and highest frequencies (Δt).

Therefore, the FDMT result can be expressed as a two dimensional array that contains the integrals along dispersion curves as a function of t_0 and Δt

$$A_{f_{\min}}^{f_{\max}}(t_0, \Delta t) = \sum_{f=f_{\min}}^{f_{\max}} I(t_0 - d\left[\frac{1}{f_{\min}^2} - \frac{1}{f^2}\right], f), \quad (13)$$

where $f_{\min}$ and $f_{\max}$ are the minimum and maximum frequencies in the base-band.

To compute the FDMT transform of the input, the algorithm works in $\log_2(N_f)$ iterations. The inputs of the i'th iteration are the FDMT transforms of a partition of the original band into $N_f/2^{(i-1)}$ sub-bands of size $2^{(i-1)}$ frequencies. The outputs of i'th iteration are the FDMT transforms of a partition of the original input into $N_f/2^i$ sub-bands of size 2^i frequencies. Every two successive sub-bands are combined using the addition rule described below. After $\log_2(N_f)$ iterations, we have the FDMT over the whole band.

The FDMT combining process of two successive sub-bands into $A_{f_0}^{f_2}(t_0, \Delta t)$, is given by the following addition rule:

$$A_{f_0}^{f_2}(t_0, \Delta t) = A_{f_0}^{f_1}(t_0, t_0 - t_1) + A_{f_1}^{f_2}(t_1, t_1 - t_2). \quad (14)$$

Here, $A_{f_0}^{f_1}$ and $A_{f_1}^{f_2}$ are part of the output of the previous iteration and t_1 is the intersection time of the dispersion curve at the central frequency $f_1 = \frac{f_2 - f_0}{2}$. t_1 is uniquely determined by the formula

$$t_1 \equiv t_0 - \Delta t \frac{f_1^{-2} - f_0^{-2}}{f_2^{-2} - f_0^{-2}} \equiv t_0 - C_{f_2, f_0}\Delta t, \quad (15)$$

[2] A sub-band of the data is a part of the data that is both limited and continuous in frequency.

Table 1
Algorithm comparison

	FDMT	Brute force	Tree
Computational complexity	$\max\{N_t N_\Delta \log_2(N_f), 2N_f N_t\}$	$N_f N_t N_\Delta$	$N_f N_t \log_2(N_f)$
Information efficient	Yes	Yes	No
Memory access friendly	Yes	Yes	Yes
Parallelization friendly	Yes	Yes	No

Note. — Comparison of the FDMT algorithm with two other approaches to incoherent dedispersion, brute force (e.g.,Barsdell et al. 2012), and tree dedispersion (Taylor 1974). It is clear that the FDMT algorithm dominates in all parameters.

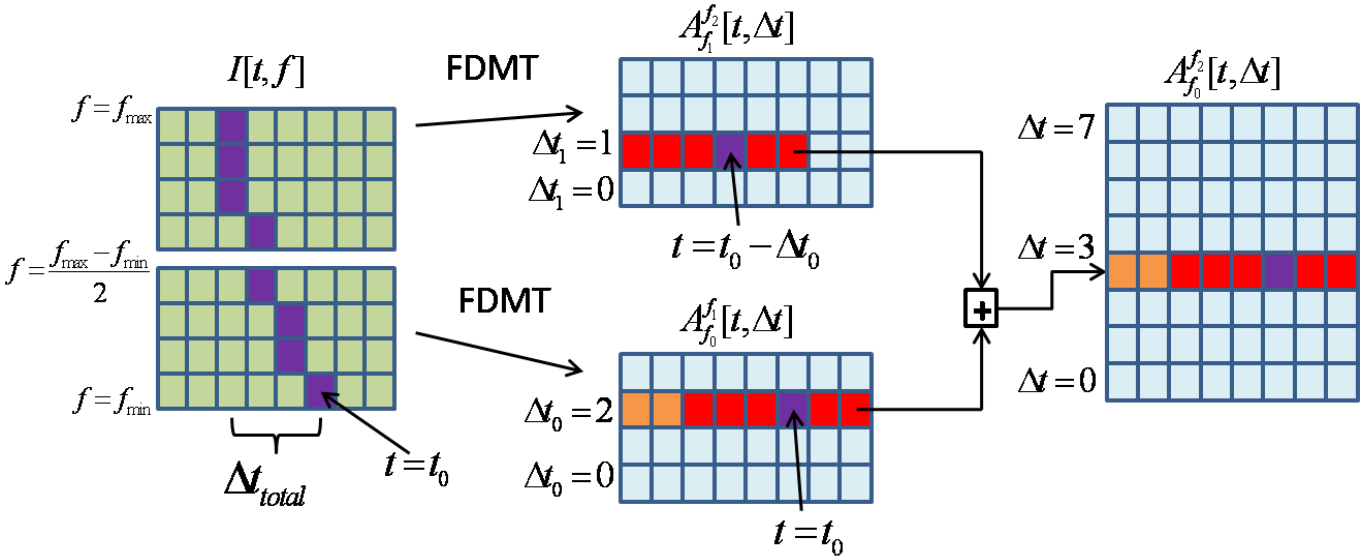

Figure 2. Illustration of a single iteration of the FDMT algorithm: On the left side, the input frequency vs. time input table is drawn. An example dispersion curve is highlighted on purple. The input table is divided methodically to two sub-bands. The right table shows the final dedispersion transform of the input (left), where the integral over the purple dispersion curve (left) is marked by the purple cell on the right. On the middle, the dispersion measure transform of the two sub-bands (which are assumed to be calculated in the previous iteration) is drawn, where each of the two sub-bands contains in each cell the sum of the unique dispersion trail with exit point t_0 and total delay Δt through the corresponding sub-bands. The cells that contain the partial sums of the two halves of the purple dispersion curve on the left are highlighted in purple. Highlighted in red, is a row in the dispersion tables that contributes to the calculation of the red cells on the right. Notice that we can add the lines highlighted in red as vectors, in order to implement the algorithm in a vectorized form. Highlighted in orange, are the cells that use the alternative addition rule, in the case when the dispersion trail exits the boundaries of the input table.

where

$$C_{f_2,f_0} \equiv \frac{f_1^{-2} - f_0^{-2}}{f_2^{-2} - f_0^{-2}}. \tag{16}$$

By definition, $A_{f_0}^{f_1}(t_0, t_0 - t_1)$ is the calculated sum over the unique dispersion curve between the coordinates (t_0, f_0) and (t_1, f_1), and $A_{f_1}^{f_2}(t_1, t_1 - t_0)$ is the same for (t_1, f_1) and (t_2, f_2). After an FDMT iteration, the only dispersion curve passing through $(t_0, f_0), (t_2, f_2)$, will be given by $A_{f_0}^{f_2}(t_0, t_0 - t_2)$.

For sufficiently early t_0, the time t_1 will be smaller than zero. In that case we just copy – i.e, use the alternative addition rule

$$A_{f_0}^{f_2}(t_0, \Delta t) = A_{f_0}^{f_1}(t_0, t_0 - t_1). \tag{17}$$

The operation of one iteration of the algorithm is graphically illustrated in Figure 2.

The only thing left to deal with is the data initialization. This is done prior to the first iteration, generating $A_{f_0}^{f_1}$ for every two consecutive frequencies. If the maximum dispersion delay between two consecutive frequencies is smaller than the width of a time bin, then we can use the simple initialization:

$$A_f^{f+\delta_f}(t_0, 0) = I(f, t), \tag{18}$$

where

$$\delta_f = \frac{f_{\max} - f_{\min}}{N_f}. \tag{19}$$

Otherwise, the energy of the signal at some frequencies is not located at a single bin. This can be compensated for by two ways:

1. By computing partial sums over the time axis. i.e

$$A_f^{f+\delta_f}(t, \Delta t) = \sum_{i=0}^{\Delta t} I(f, t + i). \tag{20}$$

2. If for a certain d, the time delay within each single frequency bin is larger than one time bin, we can simply reduce the time resolution (i.e., bin). Note

that this implies $N_\Delta >> N_f$, which we show in §2 to be suboptimal in terms of sensitivity.

In the MATLAB and Python codes we provide, we use Option 1. The maximal time delay within each frequency bin is uniquely determined by $d_{\max}$, the maximal d we want to scan, and is given by

$$\Delta t_{\max}(f_0) = d_{\max}\frac{f_0^{-2} - (f_0+\delta_f)^{-2}}{f_{\min}^{-2} - f_{\max}^{-2}}. \quad (21)$$

Therefore, this decision can be made prior to the computation. A pseudo code of FDMT is given in Algorithm 1. In addition, we provide implementation for the algorithm in Python and MATLAB.[3] Note that so far we did not treat rounding and binning issues, these are discussed in §4.2 and are implemented in the codes we provide.

Algorithm 1 The FDMT Algorithm

Input: $I(f,t)$ input matrix (possibly packed), t axis is continuous in memory.
Output: Packed dispersion measure scores arranged in a two dimensional table $A_{f_{\min}}^{f_{\max}}(t,\Delta t)$ where Δt represents the dispersion measure axis.

Initiate the table by $A_f^{f+\frac{f_{\max}-f_{\min}}{N_f}}(t,\Delta t) = \sum_{i=0}^{\Delta t} I(f,t+i)$
for iteration $i = 1$ to $i = log_2 N_f$ **do**
  for f_0 in the range $[f_{\min}, f_{\max}]$ with steps $2^i\delta_f$ **do**
    $f_2 = f_0 + 2^i\delta_f$
    $f_1 = \frac{f_2+f_0}{2}$
    $C_{f_2,f_0} = \frac{f_1^{-2}-f_0^{-2}}{f_2^{-2}-f_0^{-2}}$
    $\Delta t_{\max}(i,f_0) = N_\Delta \frac{f_0^{-2}-(f_0+2^i\delta_f)^{-2}}{f_{\min}^{-2}-f_{\max}^{-2}}$
    for Δt in the range $[0, \Delta t_{\max}(i,f_0)]$ **do**
      for t_0 in range $[C_{f_2,f_0}\Delta t, N_t]$ **do**
        $t_1 = t_0 - C_{f_2,f_0}\Delta t$
        $A_{f_0}^{f_2}(t_0,\Delta t) = A_{f_0}^{f_1}(t_0, t_0-t_1) + A_{f_1}^{f_2}(t_1, t_1-t_2)$
      end for
      for t_0 in the range $[0, C_{f_2,f_0}\Delta t]$ **do**
        $A_{f_0}^{f_2}(t_0,\Delta t) = A_{f_0}^{f_1}(t_0, C_{f_2,f_0}\Delta t)$
      end for
    end for
  end for
end for

4.1. *Computational Complexity*

To calculate the computational complexity, we need to trace the number of operations done throughout the algorithm. The amount of additions in iteration i is bounded from above by $N_b N_t \Delta t_{\max}(i)$ where N_b is the number of sub-bands processed at the current iteration, and $\Delta t_{\max}(i)$ is the maximum time shift within a single sub-band at iteration i for the curve with highest dispersion measure:

$$\Delta t_{\max}(i,f_0) = N_\Delta \frac{f_0^{-2} - (f_0+2^i\delta_f)^{-2}}{f_{\min}^{-2} - f_{\max}^{-2}}, \quad (22)$$

[3] The codes are available from https://sites.google.com/site/barakzackayhomepage/home

$$\Delta t_{\max}(i) = \max_{f_0}\{\Delta t_{\max}(i,f_0)\} = \quad (23)$$
$$= N_\Delta \frac{f_{\min}^{-2} - (f_{\min}+2^i\delta_f)^{-2}}{f_{\min}^{-2} - f_{\max}^{-2}}.$$

As a first approximation, one can assume that the dispersion curve is almost linear, meaning that the number of Δt's needed in iteration i is roughly twice the number needed in iteration $i+1$. In the last iteration, $|\Delta t| = N_\Delta$, and therefore, in iteration i,

$$\Delta t_{\max}(i) \approx \frac{N_\Delta 2^i}{N_f}. \quad (24)$$

The number of bands (N_b) in each iteration is $N_b = \frac{N_f}{2^i}$. Therefore, under the approximation of almost linear dispersion (or narrow band), the following approximation is correct:

$$N_b \Delta t_{\max}(i) \approx \max\{N_\Delta, N_b\} \quad (25)$$

Summing this for all iterations, and assuming N_Δ is dominant in all iterations, and taking into account the number of entries in each added row (N_t), we get the complexity

$$C_{\text{FDMT}} = N_t N_\Delta \log_2(N_f). \quad (26)$$

If we assume N_b is dominant, we get

$$C_{\text{FDMT}} = N_t N_f + \frac{N_t N_f}{2} + ... = 2N_t N_f. \quad (27)$$

Therefore, the total complexity of the algorithm is bounded from above by:

$$C_{\text{FDMT}} \le 2N_t N_f + N_t N_\Delta \log_2(N_f). \quad (28)$$

Casting the complexity analysis of the FDMT algorithm with the more naturally defined N_s, N_d, using $N_s = N_f N_t$ and $N_\Delta = N_d/N_f$ we get

$$C_{\text{FDMT}} \le 2N_f \frac{N_s}{N_f} + \frac{N_d}{N_f}\frac{N_s}{N_f}\log_2(N_f) = \quad (29)$$
$$= 2N_s + \frac{N_s N_d}{{N_f}^2}\log_2(N_f).$$

Adding to the above the complexity of data preparation by STFT, $N_s \log_2(N_f)$, and the fact that if we chose $N_f < N_p$ we can effectively bin (or low pass, see §5.1) to size N_p, we get

$$C_{\text{FDMT}} \le N_s \log_2 N_f + \frac{2N_f N_s}{N_p} + \frac{N_d N_s}{{N_p}^2}\log_2(N_f). \quad (30)$$

Here we can see that the data preparation complexity dominates the operation count of the algorithm whenever incoherent dedispersion is maximally sensitive (i.e, $N_d < {N_p}^2$).

4.2. *Implementation details*

In this subsection we consider implementation issues, such as rounding and binning, pulse profile convolution and using an arbitrary number of frequencies. In addition, it is important to implement the tricks of the trade,

in order to transform the theoretical complexity reduction to a real speedup.

Rounding and binning: The exact formulas written above need to take into account discreteness of both frequency and time axes. To keep the formulas readable, we did not include these considerations in the algorithm description and pseudo-code. However, we include them in the implementation we provide, and we advise readers who want to implement the FDMT to pay attention to the discretization process. That is because an incorrect choice might lead to a significant reduction in accuracy.

An example of the most important discretization issue, is that when combining two sub-bands, the point t_1, where the dispersion curve travels from one sub-band to the next might not be well defined. This can happen, because the dispersion curve might travel one bin between the end frequency of the first band $f_0 + (2^i - 1)\delta_f$ and the start frequency of the second band $f_0 + 2^i\delta_f$. The implemented solution for this problem is to calculate two versions of C_{f_0,f_2}, one with the end frequency of the lower sub-band, and the other with the start frequency of the upper sub-band. Using the different versions of C_{f_0,f_2} in the two different uses of t_1, we can account for a time shift between the added bands, approximating the dispersion curve better.

Machine word utilization: One can utilize the machine word width (or the width of the SSE registers) to pack few instances of the dedispersion procedure into one computation (since modern computers operate on machine words of 64 bits, this will result in a speedup factor of 4–8 depending on the number of bits per frequency and the pulse maximum allowed strength).

Memory access: An important issue in run-time reduction, is the continuity of memory access. The FDMT algorithm never performs any re-ordering action along the time axis. Therefore, it is recommended to store the time axis continuously in order to speed up the memory access operations.

Different range of dispersions: Sometimes, we have a prior knowledge of the range of dispersion measures needed to scan. In that case, one can still employ the FDMT algorithm after an additional preparation of applying either a frequency dependent shift to the input (according to some $d_{\min}$), or a coherent dedispersion of the signal (using $d_{\min}$).

Pulse profile: Sometimes we have prior knowledge on the pulse width or profile (might be a different profile for each frequency, like in pulse scattering). By applying a matched filter approach, one can convolve each frequency time series with the predicted profile for that frequency and employ the FDMT at the end. For wide enough pulse shapes, one might consider binning the time resolution.

We note that convolution of the time axis with a uniform pulse profile (for all frequencies) commutes with the entire FDMT operation. Therefore, we can test a few pulse profiles per FDMT without repeating the dedispersion process.

Dealing with the case of $N_f \neq 2^k$: The algorithm presented above assumes that the number of frequencies is strictly a power of two. This assumption can be abandoned by slightly adjusting the addition rule to allow a merge of non-equal size sub-bands. The only change needed is to switch f_1 in Equation 16 from being the middle frequency between f_0 and f_2 to be the border frequency between the sub-bands.

Applying FDMT for other functional forms: The dispersion equation (Eq. 2) is used only in the preparation of C_{f_2,f_0}. One can easily extend the FDMT algorithm to search for other functional forms, for example,

$$\Delta t = f_1^{\gamma} - f_2^{\gamma}. \tag{31}$$

The only required modification is to change the power of the frequency in Equation 16 from -2 to γ. Furthermore, it could be extended to any family of curves that satisfies the condition that there is only one curve passing between any two points in the input data. Using this, one can calculate the required C_{f_2,f_0}, by finding the only curve passing through both (t_0, f_0) and (t_2, f_2), and defining t_1 to be the intersection time of the curve with the frequency f_1. While the complexity of the algorithm, may change with the functional form, for a sufficiently regular functional form, the complexity will close to $N_d N_t \log N_f$.

5. ELIMINATING SHIFTS BY FFT'ING THE TIME AXIS

In modern computers and GPU's, memory access is frequently the bottleneck of many algorithms, especially when programming transforms, where the computational complexity is only slightly larger than the data size. Efficient implementation of transform algorithms is non-trivial and requires architecture dependent changes in order to avoid cache misses [4] (in a general CPU setup) or to avoid communication when using distributed computing.

While it is probably possible to control the algorithms behavior as presented above, it is non-trivial to distribute the data between different processing units while avoiding duplication and communication issues.

In this subsection, we present a variant of the algorithm which is easily parallelized on all architectures and where the memory access pattern is as parallelization friendly as possible.

The algorithm, as it is described in §4, has only one core operation: adding a complete shifted "time" row. It is the shift operation which makes the data transfer and memory management of the algorithm challenging, and therefore we wish to eliminate shifts from the algorithm. In order to do that, we can Fourier transform the time axis. This makes the shift operation become a multiplication with a "shift vector" which is the Fourier transform of a shifted delta function. In this version, all additions are of numbers from the same (Fourier transformed) time coordinate. Therefore, we can assign different parts of the (Fourier transformed) time axis to different processing units, and consequently reduce the need for shared memory or data transport. At the end, we need to Fourier transform back the time axis. We call this algorithm FFT–FDMT and it is summarized in Algorithm 2. Tracking the data in this algorithm, we can see that there are only two "global" steps and that they are both transpose operations of the data. To perform all other steps of the algorithm we need only to access memory that is not larger than one row or one column of the

[4] In modern computers, the fastest memory buffer is the L1 cache. An access to a value that is not stored in the L1 cache causes a memory read from slower storage media such as L2 cache or the RAM memory, and is sometimes called a "cache miss".

input. Since present L1 cache architectures can contain more than a typical row or column of data, the algorithm can be computing-power limited. The run-time of this algorithm on any machine is comparable to the run-time of two dimensional convolution, because of the similar number of operations and data access patterns. We note that in the basic preparation of radio data, one often applies Fourier transforms (for example, when applying filters or screening for radio frequency interferences). Therefore, if we have the computational ability to prepare the input table from the raw data, FFT–FDMT is also feasible.

Algorithm 2 The FFT–FDMT Algorithm

Input: $I(f,t)$ input matrix (possibly packed), t axis is continuous in memory.
Output: Packed dispersion measure scores arranged in a two dimensional table $A^{f_{\max}}_{f_{\min}}(t,\Delta t)$ where Δt represents the "dispersion measure" axis.

1: Initiate the table by
$$A_f^{f+\frac{f_{\max}-f_{\min}}{N_f}}(t,\Delta t) = \sum_{i=0}^{\Delta t} I(f,t+i)$$
2: Initiate the "shift vector" $V(\tilde{t_0},\Delta T) = \mathcal{F}(\delta(\Delta T))(\tilde{t_0})$ where $\delta(x)$ is a vector containing one at position x and zeros everywhere else, $\mathcal{F}$ is the FFT operator, and $\tilde{t_0}$ is the index of the Fourier transformed time axis.
3: Fourier transform the time axis
$$B_f^{f+\delta_f}(\tilde{t},\Delta t) = \mathcal{F}(A_f^{f+\delta_f}(:,\Delta t))$$
4: Transpose the data. after this action, the frequency and Δt axes should be continuous in memory, time axis should be distributed across all computing units.
5: **for** $\tilde{t_0}$ in the range $[0, N_t]$ **do**
6: **for** i in the range $[1, \log_2 N_f]$ **do**
7: **for** f_0 in the range $[f_{\min}, f_{\max}]$ with steps $2^i\delta_f$ **do**
8: $f_2 = f_0 + 2^i\delta_f$
9: $f_1 = \frac{f_2+f_0}{2}$
10: $C_{f_2,f_0} = \frac{f_1^{-2}-f_0^{-2}}{f_2^{-2}-f_0^{-2}}$
11: $\Delta t_{\max}(i,f_0) = N_\Delta \frac{f_0^{-2}-(f_0+2^i\delta_f)^{-2}}{f_{\min}^{-2}-f_{\max}^{-2}}$
12: **for** Δt in the range $[0, \Delta t_{\max}(i,f_0)]$ **do**
13: $\Delta t_1 = C_{f_2,f_0}\Delta t$
14:
$$B_{f_0}^{f_2}(\tilde{t_0},\Delta t) = B_{f_0}^{f_1}(\tilde{t_0},\Delta t_1) + B_{f_1}^{f_2}(\tilde{t_0},\Delta t-\Delta t_1)V(\tilde{t_0},\Delta t_1)$$
15: **end for**
16: **end for**
17: **end for**
18: **end for**
19: Transpose the data back. Now, time is again continous in memory.
20: Perform inverse Fourier transform on the time axis.
$$A^{f_{\max}}_{f_{\min}}(\tilde{t},\Delta t) = \mathcal{F}^{-1}(B^{f_{\max}}_{f_{\min}}(:,\Delta t))$$

5.1. *Comments on the Implementation of the FFT-FDMT algorithm*

The FFT-FDMT algorithm is designed to increase the amount of computation per cache replacement. To completely optimize the algorithm for this property, we have to consider special implementation details like cache size and processing units communication geometry. Though important to an efficient implementation of the algorithm, these details are considered out of scope for this paper as they are architecture dependent. We note that all the details discussed in §4.2 are valid also for the FFT-FDMT version, except for the changes listed below.

Machine word utilization: The long integer data type is the optimal choice for the regular FDMT algorithm in order to fully utilize the machine word capability. In the Fourier transformed version of the algorithm, we have to use the complex floating point data type. Using the floating point data type, we have to leave unused the bits of the exponent field, and leave some more bits unused to retain the floating point precision needed to perform the Fourier transform operations. Furthermore, some architectures such as GPUs, have a clear optimization preference for the 32 bit floating point data type. However, it is possible to pack another algorithm instance in the complex field of the input vector. Since the result of the FDMT algorithm is real (as a sum of real numbers), packing another input to the imaginary part of the input is possible. The imaginary part of the result will be the second algorithm instance.

Pulse profile: In addition to the ability to test several pulse profiles per FDMT operation, as explained in §4.2, we can further exploit the use of the Fourier transformed time domain. If the pulse width is slightly larger than one bin, reducing the computational load by binning loses information. Instead, we can effectively apply a low-pass filter on the time axis by either keeping less (time-domain) frequencies or multiplying with a filter. This can be both more sensitive than binning the time axis and more efficient than having a high sampling rate.

Handling Large dispersion measures: If the maximum dispersion broadens the pulse to more than one time bin per frequency bin, the initialization phase of the algorithm inflates the data from size N_tN_f to size $N_\Delta N_f$ (note that the use of $N_\Delta >> N_f$ is losing sensitivity, and therefore this part is not considered a crucial part of the algorithm). The partial sum operation of the initialization phase is equivalent to an application of a low-pass filter on the time axis. This allows natural reduction of computation and memory by saving a differential amount of Fouriered time bins for different Δt's. This can be used in the case of large dispersion measures to reduce the algorithm's complexity from $N_\Delta N_t \log_2(N_f)$ to $2N_tN_f\log_2(\frac{N_\Delta}{N_f})$.

Zero padding: Since convolution is a cyclical operation, all the shifts done in this algorithm are cyclical shifts. Therefore, we have to pad the time axis with N_Δ zeros prior to the Fourier transform. This operation can increase by a factor of two the complexity of the algorithm if $N_\Delta = N_t$. To avoid this we can choose $N_t >> N_\Delta$. This is usually possible if the size of the input table is not too close to the maximum memory (or cache) capacity of the machine used.

6. RUN TIME AND BENCHMARKING

Accurate benchmarking of algorithms should use a mature code, and contain architecture dependent adaptations. However, it is important to demonstrate that the code we present, running on a single standard CPU, is competitive with the brute force dedispersion implementations on GPU's. Therefore, we provide a simple benchmark for the provided code.

The benchmark we use is the run-time of performing FDMT on data with the following properties: $N_f = 2^{10}$, $N_t = 5 \times 2^{16}$ and $N_\Delta = 2^{10}$. This volume of input is similar to the one used in the "toy observation" defined in (Barsdell et al. 2012; Magro et al. 2011), $N_f = 2^8$, $N_t = 2^{19}$ and $N_\Delta = 500$. Although, we modified the partition between N_f, N_t, and increased N_d by a factor of two[5]. The run-time we achieve on this data is 3.5 seconds, on a standard Intel Core i-5 4690 processor. For example, these numbers can represent a real time dedispersion of 8 seconds of input data with 40 MHz bandwidth and 1024 dispersions. To get this benchmark, we pack five instances of the algorithm to the 64 bit machine word, allocating 12 bits to each instance. The resulting packed data has dimensions $N_t = 2^{16}, N_f = 2^{10}$, and serves as input to the FDMT implementation. Using this scheme, we find that our run-time is already shorter than that of the state of the art brute force implementations on GPU's reported in (Barsdell et al. 2012; Magro et al. 2011). A comparison between the run-times is shown in Table 2.

7. BRIDGING THE GAP BETWEEN COHERENT AND INCOHERENT DEDISPERSION

Since some interesting transient sources such as pulsar giant pulses are in the regime $1 << N_p < \sqrt{N_d}$, it is of importance to find a feasible and sensitive algorithm for their exact dedispersion. Coherent dedispersion was, until now, the only sensitive alternative. The noise power summed when searching for a pulse that is dispersed with a dispersion measure d is

$$N_p + N_d. \tag{32}$$

The noise power summed when searching for a non-dispersed pulse is N_p, and therefore the largest dispersion tolerable for sensitive pulse detection satisfies

$$N_d = N_p. \tag{33}$$

Therefore, for sensitive detection, the number of dispersion measure trials we need to process is

$$\frac{N_d}{N_p}. \tag{34}$$

The convolution operation performed for coherent dedispersion can be efficiently calculated with Fourier transforms of size N_d, and therefore the complexity of coherent dedispersion is:

$$C_{\text{coherent}} = \frac{N_d}{N_p} N_s \log_2(N_d). \tag{35}$$

Noting that the computational complexity of coherent dedispersion scales with N_d/N_p and that of incoherent dedispersion scales with N_d/N_p^2, we see that using coherent dedispersion is not computationally efficient for resolved pulses (i.e $N_p > 1$).

7.1. *Hybrid algorithm for dedispersion*

[5] This is a more realistic choice, since using large $N_\Delta > N_f$ usually looses sensitivity (see §2), and the number of frequencies is usually larger than 2^{10}.

In order to have both the detection sensitivity of coherent dedispersion and the computational complexity of FDMT, we propose the following solution: Coherently dedisperse the raw signal with coarse trial dispersion values (with steps δd), and then apply STFT and absolute value squared, followed by FDMT with the maximal dispersion being the next coarse-trial coherent dedispersion. This process ensures that the FDMT will not lose sensitivity, relative to coherent dedispersion.

We denote by $N_{\delta d}$ the number of bins of length τ that a delta function pulse will spread upon when dispersed by δd:

$$N_{\delta d} = \frac{4.15 \delta d (f_{\min}^{-2} - f_{\max}^{-2}) \text{ms}}{\tau} \tag{36}$$

As shown in §2, in order to retain sensitivity the maximal dispersion residual to be processed by the following FDMT must be bounded from above by

$$N_{\delta d} = N_p^2 \tag{37}$$

Therefore, the number of trial dispersions we need to coherently dedisperse is

$$N_{\text{coherent}} = \frac{N_d}{N_p^2}. \tag{38}$$

This process is approaching maximum sensitivity, and its complexity is:

$$C_{\text{hybrid}} = \frac{N_d}{N_p^2} N_s (\log_2(N_d) + \log_2(N_f)) + \\ + \frac{2 N_d N_s}{{N_p}^2} + \frac{N_d N_s}{{N_p}^2} \log_2(N_f). \tag{39}$$

Simplifying, we get the computational complexity for detection of a pulse of length N_p:

$$C_{\text{hybrid}} = \frac{N_d N_s}{{N_p}^2} (2 + \log_2(N_d) + 2 \log_2(N_p)). \tag{40}$$

This complexity is near optimal, because the number of uncorrelated scores is $\frac{N_d N_s}{{N_p}^2}$, which is only a logarithmic factor smaller than the computational complexity. Therefore, there is not much room for further reduction of computational complexity. The algorithm is summarized in Algorithm 3.

Algorithm 3 Coherent hybrid FDMT dedispersion algorithm

Input: Antenna voltage series $x(t)$.
Output: Score table for all dispersions $d < N_d$ with steps N_p and all exit times $t_0 < N_s$ with steps N_p.

1: **for** dispersion d_0 in the range $[0, d_{\max}]$ in steps of $\frac{{N_p}^2}{N_d} d_{\max}$ **do**
2: Create the signal $y(t)$ by applying the filter $\hat{H}_{d_0}(f) = \exp\left(\frac{2\pi i d_0}{f + f_0}\right)$ to $x(t)$.
3: Apply STFT with block size N_p on $y(t)$, to obtain $I(t, f)$.
4: Apply FDMT to I, with maximal $\Delta t_{\max} = N_p$, and output the partial result $A_{f_{\min}}^{f_{\max}}(d_0 + d, t_0)$ for $d < {N_p}^2$ with steps N_p, and t_0 in the range $[0, N_s]$ with steps N_p.
5: **end for**

Table 2
Runtime comparison

	This work	Magro et al. (2011)	Barsdell et al. (2012)
Machine used	Intel Core i5 4690	Tesla C1060 GPU	Tesla C1060 GPU
Programming language	Python (anaconda + accelerate)	C	C
Number of instances packed	5	1	1
Runtime	3.5s	4.8s	2.1s
$N_f, N_t(\text{total}), \text{N}_\text{d}$	$2^{10}, 5\times 2^{16}, 2^{10}$	$2^8, 2^{19}, 500$	$2^8, 2^{19}, 500$
$N_f N_t N_d$	5×2^{36}	2^{36}	2^{36}
Algorithm used	FDMT (non-FFT version)	Brute force	Brute force
Algorithm theoretical complexity	$N_t N_f + N_t N_d \log_2(N_f)$	$N_f N_t N_d$	$N_f N_t N_d$

Note. — The FDMT algorithm has a different computational complexity scaling than the brute-force dedispersion it is compared to. Even with standard CPU's and with a high-level programming language, the FDMT implementation we present here is faster than existing GPU implementations of brute force dedispersion.

7.2. *Implications*

Using this algorithm, it is possible to perform blind searches for pulses with duration in the $1\,\mu\text{s} - 1\,\text{ms}$ regime (which implies $N_p = 10^2 - 10^5$ for standard searches). Together with the low computational complexity of FDMT, this can be efficiently employed in blind searches for FRBs and giant pulses, both reducing the computational load, and increasing sensitivity.

8. FDMT FOR RADIO INTERFEROMETERS

In this section, we analyze the complexity of blind searches of short astrophysical signals with radio interferometers using the FDMT. We first calculate the computational and communication complexity of applying the FDMT algorithm after the imaging operation. Afterwards, we offer a way to reduce the communication complexity by applying the FDMT algorithm after the correlator operation and before the imaging operation. We show that in principle, using our scheme, it is possible to use modern radio interferometers to detect and locate short astrophysical pulses in real-time without the knowledge of the dispersion measure.

We start by introducing some additional notation. In the scenario of a blind dispersed pulse search with a radio interferometer, we have signals of several telescopes. We denote the raw voltage signal from the j'th telescope by x_j. We further denote by N_a the number of antennas, and by N_l the number of distinct locations in the sky, or pixels, in the optimal image resolution of the interferometer. The desired statistic that we need to calculate for efficient detection is given by:

$$S(t_0, p_x, p_y) = \sum_{t=t_0}^{t=t+N_p} \sum_{i=0}^{N} (x_i \otimes H)(t + u_i(p_x, p_y)), \quad (41)$$

where $u_i(p_x, p_y)$ represents the time delay of the signal at antenna i, H is the dedispersion filter needed to be convolved with to correct for dispersion, and $\otimes$ represents convolution. We wish to calculate this score for all combinations of sky positions (which we denote their number by N_l), dispersions $\frac{N_d}{N_p}$, and start times $\frac{N_s}{N_p}$. Therefore, the number of calculated scores is $\frac{N_l N_s N_d}{{N_p}^2}$.

We estimate the number of computations required by using general modern radio interferometer parameters such as: $\nu = 100\,\text{MHz}$, $t_d = 0.1\,\text{s}$, $t_p = 0.1\,ms$. Using $N_a = 300$ antennas of diameter 10 m, spread out to a maximal baseline of 10 km. $N_l = 10^6$, $N_s = 10^8$, $N_p = 10^4$, $N_d = 10^7$ we get 10^{13} scores per second, which requires a computing power of 10 TFlop/s to process. The computational requirement of the solution we propose in the next section is only logarithmically larger than this computation rate. Therefore, it is feasible with current facilities to perform a blind search using modern radio interferometers. For example, the computing facility of the Australian Commonwealth Scientific and Industrial Research Organisation[6] (known as CSIRO) has computing power equivalent to 260 TFlop/s [7].

8.1. *The standard approaches to pulse blind search with interferometers*

There are two existing approaches to blind search interferometry. The first is to add antennas incoherently and then dedisperse. This process loses the angular resolution and reduces the sensitivity by a factor of $\sqrt{N}$. However, this is considered to be computationally feasible, and it is sensitive to the interferometer's entire field of view.

The second approach is to "beam-form" and dedisperse, i.e for every searched location (p_x, p_y), shift all the signals from all antennas with the correct shift for position (p_x, p_y), add them up, and perform dedispersion. To mitigate the computational load of this process, it is custom to use only a small subset of all N_l sky locations at a time, considerably reducing the overall survey speed of the instrument.

Another possibility is to use a combination of both approaches by dividing the interferometers to closely packed stations, beam-forming all stations to a subset of all possible directions, and then incoherently adding the stations. All methods trade the computational unfeasibility with a significant sensitivity reduction. A consideration of those approaches can be found in (van Leeuwen & Stappers 2010).

8.2. *The proposed solution*

First, we quickly review the standard imaging process of interferometry, using the approximations of flat sky and short observation. Assuming there is no dispersion,

[6] http://www.atnf.csiro.au/
[7] Taken from the Top500 website, http://www.top500.org/list/2014/06/?page=2

the desired score is

$$S(t_0,p_x,p_y) = \sum_{t=t_0}^{t=t_0+N_p}\left|\sum_{j=0}^{N} x_j(t+u_j(p_x,p_y))\right|^2 \quad (42)$$
$$= \sum_{f=f_{\min}}^{f_{\max}}\left|\sum_{j=0}^{N}\hat{x}_j(t_0,f)\exp(-2\pi i f u_j(p_x,p_y))\right|^2$$
$$= \sum_{f=f_{\min}}^{f_{\max}}\sum_{j,k=0}^{N}\hat{x}_j(t_0,f)\hat{x}_k^*(t_0,f)\times$$
$$\exp\left(-2\pi i f(u_j(p_x,p_y)-u_k(p_x,p_y))\right),$$

denoting by $\hat{x}$ the Fourier transform of x. To efficiently calculate this score for every pixel p_x, p_y, it is useful to use the relation

$$u_j(p_x,p_y) - u_k(p_x,p_y) \propto (L_j - L_k)\cdot(p_x,p_y), \quad (43)$$

denoting by L_j the two dimensional location of antenna j on the plane (under the approximation of having all antennas on the same plane). Now, we can calculate the score at all positions (p_x, p_y) at the same time, by a two dimensional fourier transform of the array:

$$\hat{S}(t_0,p_u,p_v) = \sum_{j,k,f}\hat{x}_j(t_0,f)\hat{x}_k(t_0,f)\mathbb{1}((L_j-L_k)f,(p_u,p_v)) \quad (44)$$

$$S(t_0,p_x,p_y) = \mathcal{F}^{-1}(\hat{S}(t_0,p_u,p_v)), \quad (45)$$

where $\mathbb{1}((a,b),(c,d))$ is equal to one if $(a,b)=(c,d)$ (to the desired approximation), and zero otherwise.

The summation in Equation 44 is a sum of squares. This means that coherent dedispersion operations must be performed before correlating[8], because the imaging process calculates the sum of square absolute values of frequencies.

Incorporating dedispersion into this, we can see that the block size N_f we used earlier is transformed in this framework to the size of the Fourier transform done by the correlator. As a result, the imaging process cannot be done simultaneously in all frequencies, as different frequency sets should be used for different dispersion measures. Naively, this means that we need to image separately at each frequency, performing many two dimensional Fourier transform imaging operations, followed by an FDMT for every pixel. Denoting the complexity of the i'th step of the algorithm by C_i, the complexity of the coherent dedispersion + STFT of all individual antennas is

$$C_1 = \max\left(1,\frac{N_d}{{N_p}^2}\right)N_a N_s(\log_2(N_d)+\log_2(N_f)). \quad (46)$$

The complexity correlating all pairs of antennas is

$$C_2 = \max\left(1,\frac{N_d}{{N_p}^2}\right)\frac{N_a(N_a-1)}{2}N_s. \quad (47)$$

[8] The process of calculating $\hat{x}_i(t_0,f)\hat{x}_j(t_0,f)$ is referred to as "correlating" in the literature, and is calculated by a computing infrastructure usually called "the correlator".

The complexity of the imaging process is

$$C_3 = N_l\log_2(N_l)\frac{N_s N_d}{{N_p}^2}. \quad (48)$$

The complexity of the FDMT algorithm (without the STFT part which was already done in this context) is

$$C_4 = N_l\frac{N_s N_d}{{N_p}^2}\log_2(N_f). \quad (49)$$

So, the total complexity of this process is

$$C = C_1 + C_2 + C_3 + C_4. \quad (50)$$

While the complexity of this process is indeed "optimal", in the sense that it is only a logarithmic factor larger than the number of independant results, implementing this will result in a reduced computational efficiency. This is due to data transport between the imaging stage and the dedispersion stage. Between these stages, $\frac{N_l N_s N_d}{{N_p}^2}$ complex numbers are being transported.

This could be mitigated by the fact that dedispersion can be done before the imaging operation, if the condition

$$(f_{\max}-f_{\min})(u_i(p_u,p_v)-u_j(p_u,p_v)) < 1 \quad (51)$$

holds[9]. If the band is wide, this condition will not hold for pairs of far away antennas. In this case, it is necessary to split the frequencies into sub-bands that are narrow enough to maintain Condition 51. Since the FDMT's input and output dimensions have the same size, the communication complexity of the proposed solution is $\frac{N_a(N_a-1)N_s N_d}{2{N_p}^2}$, which should (if $N_a^2 << N_l$) make the algorithm's run-time be computation limited, and thus feasible.

Another interesting point, is that if we are in the regime of $N_\Delta < N_f$, then the FDMT is volume shrinking, and performing it only after the imaging process will result in excessive computation in the imaging stage. This scenario is sometimes plausible, for example, when looking for pulsars in a globular cluster, where we sometimes have a relatively good guess of the dispersion measure, or if we are using the choice of $N_f > \sqrt{N_d}$ (with some sacrifice of sensitivity, if $N_f > N_p$),

This process is summarized in Algorithm 4.

[9] sometimes, if the Condition 51 doesn't hold, the resulting image is said to suffer from a "chromatic aberration".

Algorithm 4 Finding short pulses with interferometers

Input: Antenna voltage series Output: $S(d, t_0, p_x, p_y)$ for every time, dispersion, and sky location. Standard choice of N_f is N_p.

for dispersion d_0 in the range $[0, N_d - \frac{N_d}{{N_f}^2}]$ in steps of $\frac{N_d}{{N_f}^2}$ **do**
  for antenna index i. **do**
    Create the signal x_i by convolving the $i'th$ antenna signal with the dedispersion filter with index d_0.
    Apply STFT with block size of N_f on the signal x_i, to obtain $\hat{x}_i$
  end for
  for every pair of antennas i, j calculate $\hat{x}_i^d \hat{x}_j^d$ **do**
    for each populated point on the $\hat{S}(t_0, f, p_u, p_v)$ matrix **do**
      Generate the time vs. frequency map of all the frequencies[10]that enter into the same cell (p_u, p_v).
      Apply FDMT with maximal $N_d = {N_f}^2$
    end for
    Data "transpose operation", each processing unit now holds all the p_u, p_v $\hat{S}(t_0, d, p_u, p_v)$ cells, for the same d, t_0.
    for each dispersion $d < {N_f}^2$ and each time t_0 **do**
      Perform two dimensional inverse Fourier transform to calculate $S(t_0, d, p_x, p_y) = \mathcal{F}^{-1}(\hat{S}(t_0, d, p_u, p_v))$
      If for some point, the power is statistically significant.
    end for
  end for
end for

9. CONCLUSION

We present the FDMT algorithm, that performs exact incoherent dedispersion transform with the complexity of $N_f N_t \log_2(N_f)$. We show that regular implementation tricks of the trade can be combined with the FDMT algorithm to achieve significant computation speedup. We also present a variant of the FDMT algorithm that is slightly more computationally intensive, but concentrates all memory access operations to two global transpose operations, and might present further speedup on massively parallel architectures such as GPUs. We show that the FDMT algorithm dominates all other known algorithms for incoherent dedispersion and has comparable complexity to the signal processing operations required to generate its input data. Therefore, we conclude that incoherent dedispersion can now be considered a non-issue for future surveys. We provide implementations of the FDMT algorithm in high level programming languages, with a faster runtime than the state of the art implementations of brute-force dedispersion on GPUs.

We further present an algorithm that bridges the gap between coherent and incoherent dedispersion, and show that the computational complexity of this algorithm is orders of magnitude lower than that of coherent dedispersion for pulses of resolvable duration. Using this algorithm, it will be possible to perform blind searches for FRBs and giant pulse emitting pulsars with the sensitivity of coherent dedispersion searches.

Last, we compute the operation count for a blind search of short astrophysical searches with modern radio interferometers and arrive to the conclusion that it is computationally feasible using existing facilities.

BZ would like to express his deep thank to Gregg Halinnan for introducing him the problem of dedispersion. The authors would like to express their thanks to Avishay Gal-Yam, Shrinivas Kulkarni, Matthew Bailes, David Kaplan, Ora Zackay and Gil Cohen for their useful comments and advice regrading the paper. E.O.O. is incumbent of the Arye Dissentshik career development chair and is grateful to support by grants from the Willner Family Leadership Institute Ilan Gluzman (Secaucus NJ), Israeli Ministry of Science, Israel Science Foundation, Minerva and the I-CORE Program of the Planning and Budgeting Committee and The Israel Science Foundation.

REFERENCES

Barsdell, B. R., Bailes, M., Barnes, D. G., & Fluke, C. J. 2012, MNRAS, 422, 379
Brady, Martin L. "A fast discrete approximation algorithm for the Radon transform." SIAM Journal on Computing 27, no. 1 (1998): 107-119.
Clarke, N., Macquart, J.-P., & Trott, C. 2013, ApJS, 205, 4
Clarke, N., D'Addario, L., Navarro, R., & Trinh, J. 2014, Journal of Astronomical Instrumentation , 3, 50004
Gotz, W. A., and H. J. Druckmuller. "A fast digital Radon transform. An efficient means for evaluating the Hough transform." Pattern Recognition 29, no. 4 (1996): 711-718.
Manchester, R. N., Lyne, A. G., D'Amico, N., et al. 1996, MNRAS, 279, 1235
Magro, A., Karastergiou, A., Salvini, S., et al. 2011, MNRAS, 417, 2642
McLaughlin, M. A., & Cordes, J. M. 2003, ApJ, 596, 982
Lorimer, D. R., Bailes, M., McLaughlin, M. A., Narkevic, D. J., & Crawford, F. 2007, Science, 318, 777
Lorimer, D. R., & Kramer, M. 2012, Handbook of Pulsar Astronomy, by D. R. Lorimer , M. Kramer, Cambridge, UK: Cambridge University Press, 2012,
Petroff, E., van Straten, W., Johnston, S., et al. 2014, ApJ, 789, L26
Taylor, J. H. 1974, A&AS, 15, 367
Thompson, D. R., Wagstaff, K. L., Brisken, W. F., et al. 2011, ApJ, 735, 98
Thornton, D., Stappers, B., Bailes, M., et al. 2013, Science, 341, 53
van Leeuwen, J., & Stappers, B. W. 2010, A&A, 509, A7

ACKNOWLEDGMENTS

[10] Also, it is possible for certain geometric configurations that several antennas will contribute to the same cell in the U,V plane