Optimized Spectrum Permutation for the Multidimensional Sparse FFT

André Rauh and Gonzalo R. Arce, *Fellow, IEEE*

***Abstract*—A multidimensional sparse fast Fourier transform algorithm is introduced via generalizations of key concepts used in the one-dimensional (1-D) sparse Fourier transform algorithm. It is shown that permutation parameters are of key importance and should not be chosen randomly but instead can be optimized. A connection is made between the sparse Fourier transform algorithm and lattice theory, thus establishing a rigorous understanding of the effect of the permutations on the algorithm performance. Lattice theory is then used to optimize the set of parameters to achieve a more robust and better performing algorithm. Other algorithms using pseudorandom spectrum permutation can also benefit from the methods developed in this paper. The contributions address the case of the exact k-sparse Fourier transform but the underlying concepts can be applied to the general case of finding a k-sparse approximation of the Fourier transform of an arbitrary signal. Simulations illustrate the efficiency and accuracy of the proposed algorithm. The optimizations of the parameters and the improved performance are shown in simulations for the 2-D case where worst case and average case peak signal-to-noise ratio (PSNR) improves by several decibels.**

***Index Terms*—Fast Fourier transforms.**

I. Introduction

THE Discrete Fourier Transform (DFT) calculates the spectrum representation of input signals and has become ubiquitous in signal processing applications. It is used broadly throughout digital signal processing, partial differential equations, polynomial multiplication [1], and audio processing. In certain time constrained applications or when the data is very large, there is a natural desire to accelerate the DFT with algorithms that perform increasingly faster and with less power usage [2].

Formally the DFT of a one dimensional signal of N elements is defined as:

$$\hat{x}_j = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi ijn/N}, \tag{1}$$

where x_n is the signal in the time domain and $\hat{x}_k$ the DFT in the frequency domain. The complexity of a naive and straightforward implementation of the DFT has a runtime complexity of $O(N^2)$ for a one dimensional signal of length N. This is readily noticed by interpreting the calculation as a matrix vector multiplication.

The FFT reduces the runtime complexity of the DFT to $O(N \log N)$ and is therefore an important algorithm used in signal processing. While the FFT algorithm does not make any assumptions about the structure of the signal, very often the signals of interest are obtained from a structured source resulting in a nearly sparse Fourier spectrum. This fact is the basis for signal compression and is used among others in the popular MP3 codec. Other examples of structured signals include images, video and in general samples of most systems over time or space. In general it is likely that the signals encountered in nature are often structured unless the signal in question is purely random and thus just noise. These input signals frequently result in a sparse spectrum, i.e., most of the Fourier coefficients of a signal are very small or equal to zero.

Assuming that a signal of length N is k-sparse ($k < N$) in the Fourier domain, the signal can be described with only k coefficients. And thus it seems natural that there should be a better performing algorithm that exploits this property of the signal. The last 20 years has seen advances in algorithms aimed at improving the runtime for the sparse Fourier transform. The first notable in [3] and several other algorithms have been proposed with this goal in mind [4]–[7]. A recent approach is the so called sparse FFT (sFFT) which lowered the computational complexity significantly. It was introduced in [8] and improved the runtime complexity to $O(k \log N)$ to make it faster than the FFT for a given sparsity factor $k \leq O(N)$. A survey of sparse FFT methods can be found in [9].

The applicability of the sparse FFT is only limited by the sparseness of the signal in the Fourier domain. For instance, the standard FFT algorithm could be chosen to perform the compression of an audio signal for when a high audio quality is needed during playback. On the other hand, the sparse FFT could be used if the audio signal is highly compressed into a lower quality signal which may be desirable for speech. There are also applications which inherently work on sparse signals. For instance GPS deals with extremely sparse signals and the sparse FFT can be used to significantly speed up signal acquisition [2]. Another application is GHz-Wide sensing [10] of signals. Additionally, a very popular field that deals with sparse signals is compressive sensing. Recovery algorithms for compressive sensing are often limited by the performance of the sparse basis transformation. This is where the sparse FFT would come into play and therefore allow faster recovery and recovery of large problems sizes. One particularly interesting application within compressive sensing is the reconstruction of Magnetic Resonance Imaging (MRI) images which rely on the two or three

Manuscript received September 24, 2015; revised March 2, 2016, June 2, 2016, and July 18, 2016; accepted July 24, 2016. Date of publication August 10, 2016; date of current version October 31, 2016. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Wenwu Wang.

The authors are with the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19711 USA (e-mail: rauh@udel.edu; arce@udel.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2016.2599483

dimensional Fourier transform. Another similar application is to apply the 2D sparse FFT to reconstruct a more sparsified image in 2D Magnetic Resonance Spectroscopy [11]. Computational Lithography uses very large scale two dimensional Fourier transforms to calculate the intensity of every source point of the Wafer [12]. A sparse Fourier transform of sizes up to 10^{33} or more is necessary for the detection of pulsars [13].

The algorithm introduced in [8] handles only one dimensional signals and does not explicitly state how to extend the algorithm to solve the multidimensional sparse FFT problem. Being a separable transform the 1D DFT can easily be extended to multiple dimension by applying the 1D algorithm multiple times on each slice along each dimension. Applying the sparse FFT in this naive way, however, would negate all gains due to the repetitions necessary along one dimension. Some of the state of the art sparse FFT algorithms, such as [14], use permutations in order to obtain well distributed Fourier coefficients. This paper shows that these permutation can be further improved by choosing permutation parameters that are not randomly chosen but instead semi-deterministic. Hence, the proposed method can be applied to a wide range of existing and future algorithms that use permutations to solve the clustering of Fourier coefficients which are often found in real world signals.

Assuming a signal of dimensionality d with N elements along each dimension, the complexity of such a naive algorithm would be $kdN \log N$. The term N which is introduced by the repeated computation of the 1D sFFT is prohibitively fast growing compared to the desired logarithmic term $\log N$. Thus, a more efficient approach is shown in our work by extending the sparse FFT algorithm itself to multiple dimensions. In addition, the algorithm in [8] introduces many parameters which are left to be chosen randomly. This works well for the assumption of a randomly distributed input spectra but is sub-optimal for signals inhibiting structure. Often however, choosing those parameters randomly is far from optimal and becomes a much more pronounced problem with the multidimensional sFFT.

In this paper it is shown that the parameters in question should not be chosen randomly and Lattice theory is used to optimize these parameters. In fact, the findings are also applicable to other algorithms which use pseudorandom spectrum permutation [5], [8], [15]–[18]. Note that the proposed algorithm and finding in this paper are not limited to the exact sparse FFT (sFFT-3.0) but are also applicable to other and more recent developed algorithms such as [19] which also proposes a multidimensional sparse FFT. Other multidimensional sparse FFTs employing Lattices have been proposed [20] but differ in the approach.

II. Lattices

Lattices have various mathematical definitions depending on the context and application field. In this work, lattices are used as a discrete additive subgroup of $\mathbb{R}^n$ also known as point lattices. An additive subgroup has the property:

$$\forall x, y \in \mathcal{L} \rightarrow -x, x + y \in \mathcal{L} \tag{2}$$

i.e. for any two points in the lattice $\mathcal{L}$, the points $-x$ and $x + y$ are also part of that lattice.

Equivalently, a lattice is the $\mathbb{Z}$-linear span of a set of n linearly independent vectors:

$$\mathcal{L}(\boldsymbol{B}) = \{a_1\boldsymbol{b}_1 + a_2\boldsymbol{b}_2 + \cdots + a_n\boldsymbol{b}_n : a_1, a_2, \ldots, a_n \in \mathbb{Z}\}. \tag{3}$$

The vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n$ are called the basis vectors of the lattice and similarly the matrix $\boldsymbol{B}$ is called the basis of the lattice. In theory a lattice is an infinite set, however in practical applications the set of numbers will be on a finite domain.

The fundamental parallelepiped of a basis $\boldsymbol{B}$ is defined as:

$$\mathcal{P}(\boldsymbol{B}) = \left\{ \sum_{i=1}^{n} c_i \boldsymbol{b_i} : c_i \in \left[-\frac{1}{2}, \frac{1}{2}\right) \right\}.$$

Another fundamental region of particular interest is the Voronoi cell of a lattice. It is defined as the set of all points in $\mathbb{R}^n$ that are closer to the origin than to any other lattice point:

$$\mathcal{V}(\boldsymbol{B}) = \{\boldsymbol{x} \in \mathbb{R}^n : ||\boldsymbol{x}|| < ||\boldsymbol{x} - \boldsymbol{v}|| \, \forall \boldsymbol{v} \in \mathcal{L} \setminus \{\boldsymbol{0}\}\}. \tag{4}$$

Note, that the basis of a lattice is not unique. For any matrix $\boldsymbol{U} \in \mathbb{Z}^{n\times n}$, $\det \boldsymbol{U} = \pm 1$ the matrix resulting from $\boldsymbol{B} \cdot \boldsymbol{U}$ will also be a basis for the lattice $\mathcal{L}(\boldsymbol{B})$.

Some basis are more desirable and interesting than other basis due to their mathematical properties. Basis reduction is the process of taking a base and reducing it such that it still generates the same lattice. In lattice theory a vast effort has been made to find such reduced basis sets [21]. Given a basis, the problem of finding the shortest near-orthogonal vectors is called the shortest vector problem (SVP). The exact problem of finding the shortest vector within a lattice is known to be NP-hard [22]. However, various approximate solutions with polynomial time exist. In fact, this NP-hardness result gave rise to a new field called lattice based cryptography. However, various algorithms to find good approximations in polynomial time complexity have been reported [21], [23], [24].

The minimum basis of a lattice is the basis with the shortest and nearly orthogonal basis vectors. The fundamental parallelepiped of the minimum basis of a lattice is called the fundamental domain of the lattice. The determinant of the fundamental lattice is defined as the volume of the fundamental parallelepiped.

An example of a two dimensional lattice is the well known hexagonal tiling which is obtained from a lattice with basis vectors of equal length and an inner angle of $60 \deg$.

The rank of a lattice is the dimension of its linear span $m = \dim(\text{span}(\boldsymbol{B}))$. In the case of $m = n$ the lattice is called a full rank lattice. The rank of the lattice is easily visualized: It is the number of vectors necessary to generate the lattice. In two dimensions, two well known full rank lattices are the square and the hexagonal lattice.

Another recently emerging application is the usage in public key cryptography due to the high computational complexity of the lattice basis reduction problem [25]. In communication, lattices find use in demodulation and quantization both of which seek to solve the closest vector problem. In computer graphics lattices are applied in sampling patterns and pixel antialiasing [26]. Pseudo random number generators are closely

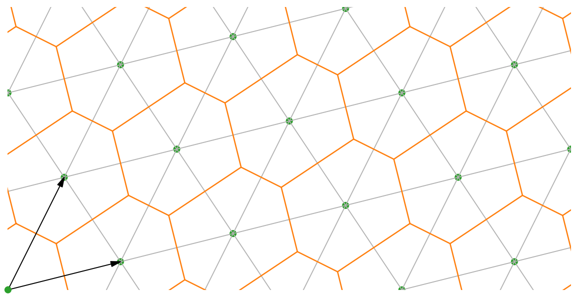

Fig. 1. The solid dots (green) are the lattice points generated by the base vectors shown in the bottom left. The solid connected lines (orange) surrounding the lattice points is the Voronoi tessellation. The straight lines (grey) crossing through the lattice are the Delaunay tessellation also known as the dual graph. The fundamental region is spanned by the two basis vectors in the bottom left. Also note, the lattice points are the centroids of the Voronoi parallelepiped.

connected to lattices and find applications in numerical integration [27].

This paper discusses lattices in the context of multidimensional sparse FFT. While the focus is on the two dimensional case, all arguments are straightforward to extended to arbitrary dimensions unless otherwise noted. An example of a two dimensional lattice is depicted in Fig. 1. It also depicts the Voronoi cell, the Delaunay tessellation and the two basis vectors that generate the lattice points. The fundamental parallelepiped is spanned by the two basis vectors and the origin.

A. *Dual Lattice*

The Nyquist-Shannon sampling theorem is the corner stone of sampling a continuous signal. It states that a signal is completely determined when sampled at twice the bandwidth. Fulfilling this constraint, it is possible to reconstruct the signal perfectly. The original theorem is stated for the one dimensional case where the samples are equidistant. It is straightforward to extend the theorem to multiple dimension by treating each dimension independently. For instance, a two dimensional signal which has a rectangular shaped spectrum with different bandwidths along each dimension, implies that the signal can be sampled at a lower rate along one dimension and must be sampled at a higher rate along the other dimension.

Naturally, it is interesting to ask what the optimal sampling lattice is for a given arbitrary shaped spectrum? Note, that any kind of rectangular sampling in the time domain will result in the repetition of the spectrum along a rectangular shaped lattice in the frequency domain. Thus, the standard rectangular sampling grid method does not allow us to answer this question in an interesting way. As introduced in the previous section, a lattice can be used to sample a continuous signal. This, in turn, raises the question of the effect on the signal spectra as it is sampled by a lattice. The answer to this question is the *Dual Lattice*.

One well known fact is that the optimal sampling lattice of circular shaped spectra is the hexagonal lattice. This is due to two facts:

1) The hexagonal lattice is self-dual, i.e. the dual of a hexagonal lattice is again a hexagonal lattice.
2) The hexagonal lattice optimizes the arrangement of spheres thus achieving the highest density. This fact was proven by Gauss in 1831 [28].

It is also helpful to think of the Voronoi tessellation of the hexagonal lattice—which is a hexagon—as the best approximation to a circle among all two dimensional lattices. Also note that the hexagonal lattice is popular due to the fact that it is optimal if the spectrum is assumed to be isotropic which is a good assumption for most real world signals.

In general, the signal is sampled in such a way that the resulting spectra of the samples signals do not overlap and thus the signal can be reconstructed perfectly. If the sampling constraints are relaxed from ordinary rectangular lattices to arbitrary lattices however, the way the signal is sampled is still constrained. As shown later, the effect of signals that are sampled according to a lattice in the time domain is that their spectrum is reduplicated according to the dual lattice in the frequency domain.

Given this constraint it is natural to ask what the optimal sampling lattice is, given an *a priori* knowledge of the shape of the spectrum. Reduplication of the spectrum in the frequency domain along a lattice $\mathcal{L}$ means that overlapping of the spectrum occurs if the spectrum extends beyond the Voronoi cell of the lattice $\mathcal{L}$. This is a straightforward result from the definition of the Voronoi tessellation (4) which quasi tiles the space according to the proximity to a lattice point. Here, the term "quasi-tiling" is used since the Voronoi cell does not include the boundary points so the tiling is not complete. The above assumes symmetric spectra or else a more general approach must be taken which will be noted later in this section.

Formally, the dual lattice of the lattice $\mathcal{L}$ is denoted as $\mathcal{L}^T$. It is defined as the set of real vectors $\boldsymbol{h} \in \mathbb{R}^n$ with:

$$\mathcal{L}^T := \{\boldsymbol{h} \in \mathbb{R}^n \,|\boldsymbol{h} \cdot \boldsymbol{x} \in \mathbb{Z} \text{ for all } \boldsymbol{x} \in \mathcal{L}\} \tag{5}$$

which means that the dual of a lattice $\mathcal{L}$ is the set of points whose inner product with any points of the lattice is an integer. The dual lattice is—as its name suggests—again a lattice. For instance the dual of the lattice $2\,\mathbb{Z}^n$ is the lattice $\frac{1}{2}\,\mathbb{Z}^n$. As this example shows, it is also common to refer to the dual lattice as the reciprocal lattice.

If the lattice is a full rank lattice, then the dual lattice can be generated with the basis matrix of the inverse of the transpose of $\boldsymbol{B}$, i.e. $(\boldsymbol{B}^T)^{-1}$. This follows directly from the definition of the dual lattice. For the use-case of improving the sparse FFT, it should be noted that we always deal with full rank lattices.

In general, the algorithm of finding an optimal sampling lattice provided an expected shape of the spectrum is the following:

1) Given the expected shape of the spectrum S_g (which, if no a-priori information is given, should be chosen as an n-sphere)
2) find a lattice whose Voronoi tessellation (4) best approximates the spectrum shape S_g. Let this lattice be called $\mathcal{L}_g$.
3) Calculate the dual lattice $\mathcal{L}_g^T$ of the lattice $\mathcal{L}_g$ and use the lattice $\mathcal{L}_g^T$ to sample the signal.

Step 2 of the algorithm follows from the sampling theorem and can be done by a simple brute force search by generating

```
procedure DualPermCandidates($\boldsymbol{L}$)
    $\boldsymbol{L}' \leftarrow \boldsymbol{L}^{-T}$
    $s \leftarrow ||\boldsymbol{L}'||_{\max}$                 ▷ Maximum matrix element
    $S \leftarrow \emptyset$, $n \leftarrow 0$
    for $m \leftarrow 1, 2, \ldots$ do
        $\tilde{\boldsymbol{P}} \leftarrow \mathrm{round}(ms\boldsymbol{L}')$
        Ensure $\det(\tilde{\boldsymbol{P}})$ is odd              ▷ See Fig. 5
        if $\tilde{\boldsymbol{P}} \notin S$ then
            $S \leftarrow S \cup \tilde{\boldsymbol{P}}$
            $\mathcal{M}_n \leftarrow \tilde{\boldsymbol{P}}$
            $n \leftarrow n + 1$
        end if
    end for
    return $\mathcal{M}$
end procedure
```

Fig. 2. The proposed iterative algorithm which generates an infinite sequence of candidates for permutation matrices $\tilde{\boldsymbol{P}}$. The only parameter needed is the lattice basis approximating the expected spectrum shape. Note that the evaluation of the "goodness" of the candidates is deferred until it is defined what constitutes a good permutation matrix. Also note that despite generating an infinite sequence an actual implementation would not realize the candidates in the sequence eagerly.

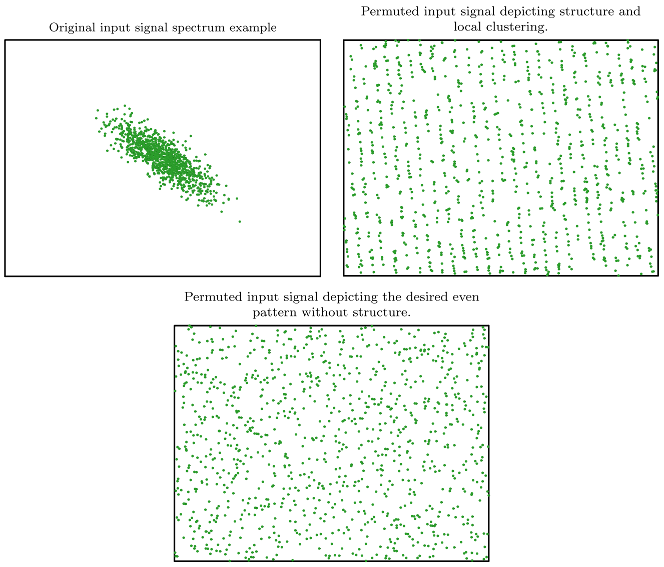

Fig. 3. Good and bad permutation of a permuted input spectrum. *Top left*: The original input spectrum generated with a multivariate Gaussian distribution. *Top right*: An example of a bad permutation due to choosing the parameters randomly. Notice the clustering of the lattice points leading to collisions in the sparse FFT algorithm. *Bottom*: An example of using a permutation obtained from using the optimal permutation with Algorithm 2. Note the very uniform distribution of the coefficients which is desirable to reduce collisions in the sparse FFT.

lattices and their Voronoi tessellation and measuring the error. The error can be measured according to some user defined error measure, for instance by calculating the area of the given spectrum that lies outside the Voronoi tessellation. This is a simple and effective approach if the dimensionality of the signal is low, otherwise this approach suffers from the popular curse of dimensionality and more sophisticated approaches must be employed [29].

Lastly, it is noted that in some cases finding an optimal lattice requires a more sophisticated algorithm. For instance, if the spectrum of the signal is non-symmetric the step 2 of the algorithm tries to match a Voronoi cell—which is always symmetric—with the non-symmetric spectrum. This, in turn, will result in a poor fitting lattice which is larger than necessary and hence results in non-optimal lattice.

In this case it is recommended, that the optimal lattice should be found by brute force iterating over lattices and reduplicate the input spectrum around the neighborhood of the origin point of the lattice. The error measure used to find a good matching lattice is the intersection of the reduplicated spectra. In general, more elaborated methods to deal with non-symmetric spectral characteristics can yield slightly better lattices.

B. Permutation Candidates Algorithm

In this Subsection an iterative algorithm is introduced which is based on the dual lattice introduced in the previous section. The pseudo code of the algorithm is given in Fig. 2. The algorithm takes the basis $\boldsymbol{L}$ of a lattice $\mathcal{L}$ and returns a sequence of matrices of the same size as the basis. Note that, one step within the algorithm ensures that the determinant of the integer matrix is odd. An algorithm to ensure an odd determinant is given in Section III.

To this end, the algorithm first calculates the dual and generates integer matrices based on this dual. This sequence of matrices can be used to permute the spectrum of the shape of the input lattice. Each candidate matrix $\boldsymbol{M}$ is evaluated by applying the permutation operator P' to the spectrum:

$$(P'\hat{x})_{\boldsymbol{j}} = \hat{x}_{\boldsymbol{M}\boldsymbol{j}}. \tag{6}$$

Each spectrum permutation is then evaluated by measuring the error. The error measure can vary by the application and is user defined. For instance it may be preferable to choose the overall PSNR or a simple inter point distance as an error measure.

Next, the algorithm is evaluated with simulations. To this end, multiple sparse input spectra are generated whose shape is derived from a multivariate Gaussian distribution. MATLAB was used for implementing the simulation. The setup was as follow: The input size was chosen to be $N \times N = 1024 \times 1024$ with a sparsity of 0.01%. One realization of such an input spectrum is depicted in Fig. 3 (left).

For each input spectrum the simulation permutes the spectrum according to (6). The permutation matrix $\boldsymbol{M}$ is either chosen randomly or from the sequence of matrices obtained from Algorithm 2.

One example of a random permutation is depicted in Fig. 3 (middle) with an example of a good permutation obtained via the proposed algorithm depicted on the right. Note the structured layout of the coefficients which are unfavorable to avoid collisions in the sparse FFT algorithm.

In order to measure the "goodness" of a permuted spectrum the distance to the closest neighbor is calculated for each nonzero coefficient:

$$D_{\boldsymbol{k}}^{\min} = \min_{\boldsymbol{j} \neq \boldsymbol{k}} ||\hat{x}_{\boldsymbol{k}} - \hat{x}_{\boldsymbol{j}}||_2. \tag{7}$$

Note that, this is different from the minimum distance of a lattice. This error measure was chosen due to being correlated to the number of collisions in the sparse FFT algorithm. Close

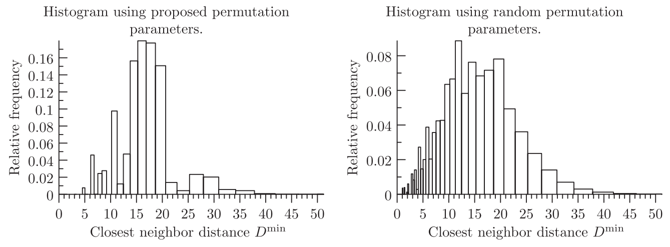

Fig. 4. Histograms depicting the difference between using different methods of choosing permutation parameters. Over 400 input spectra are generated in the shape of Fig. 3 and permuted them either randomly (left) or according the dual lattice method (right). The euclidean distance between each non-zero coefficient in the spectrum and its nearest neighbor is measured of how "good" a permutation is. The right image has a mean minimum distance of 16:1 which is a 21% improvement over using random permutation parameters (left) which has a mean minimum distance of 13:3.

neighbors can lead to collisions which in turn yield to non-optimal performance. A total of 400 simulations were run and the results summarized with a histogram which is depicted in Fig. 4. The result is a 21% improvement in the mean closest neighbor distance.

C. Complexity Analysis

The proposed algorithm generates a sequence of permutation matrices. To this end the dual matrix is calculated and subsequently multiplied by a a factor. The input size is a matrix of $d \times d$ integer elements. However, the complexity is not measured in term of d since d is usually small and considered constant for the sparse FFT algorithm. The actual complexity comes from the dimension of the input spectrum that is to be permuted. That is, the elements of the returned permutation matrix are at most N, where N is the number of elements in each dimension. It does not make sense for the elements to be $>N$ since the input spectrum is a cycling signal due to the definition of the DFT.

A naive algorithm would try all possible permutation matrices which has a complexity of $O(N^{dd})$. This means that the complexity is exponential in the number of dimensions. This is no surprise due to the well known curse of dimensionality. However, evaluating all possible matrices is impossible even for $d = 2$ unless N is small which is of no interest to the sparse FFT algorithm. The proposed Algorithm 2 reduces this complexity for any dimensionality d to a linear search algorithm of $O(N)$ by only evaluating the matrices that are likely to yield a good performance. This massive reduction in complexity allows for a much more sensible subset of possible permutation matrices which yield good performance as will be shown later. It also makes the problem tractable for higher dimensions due to being independent of d.

III. Odd Determinant Algorithm

In this Section a novel algorithm is introduced that ensures that the determinant of a square matrix is odd. This constraint is to be fulfilled in order for the permutation generated by the matrix to be invertible. Normally a matrix is invertible if the determinant is nonzero. However, the permutation applies a

```
procedure EnsureOddDeterminant($\boldsymbol{M}$)
    Assert $\boldsymbol{M} \in \mathbb{N}^{d\times d}$
    if $|\boldsymbol{M}|$ is odd then
        return $\boldsymbol{M}$
    end if
    if $d = 1$ then
        return $\boldsymbol{M}+ 1$
    end if
    Let $m$ and $\boldsymbol{N}$ be the expansion such that:
    $|M| = \sum_{i=1}^{d} (-1)^{i+1} m_{1,i} \boldsymbol{N}_{1,i}$
    if $\exists i$ for which $m_{1,i}\boldsymbol{N}_{1,i}$ is odd then
        return $\boldsymbol{M}$ with the entry $m_{1,i} \leftarrow m_{1,i} - 1$
    end if
    Find $i$ for which $|\boldsymbol{N}_{1,i}|$ is odd
    if $i$ was found then
        return $\boldsymbol{M}$ with the entry $m_{1,i} \leftarrow m_{1,i} + 1$
    end if
    if There is no $i$ for which $m_{1,i}$ is odd then
        Set $m_{1,1} \leftarrow m_{1,1} + 1$
    end if
    Find $i$ for which $m_{1,i}$ is odd
    Update $\boldsymbol{N}_{1,i} \leftarrow$ EnsureOddDeterminant($\boldsymbol{N}_{1,i}$)
    return EnsureOddDeterminant($\boldsymbol{M}$)
end procedure
```

Fig. 5. Recursive algorithm to turn an integer matrix with even determinant into a similar matrix with odd determinant. Note that the algorithm potentially recurses on a matrix of the same input size but guarantees termination after only one more call due to the conditions preceding the recursion. The algorithm works by flipping bits carefully such that the Laplace expansion of the determinant has an odd number of odd terms.

modulo N operation to the indices due to the implicitly cyclic DFT. This results in a system of linear congruences which are invertible if the determinant is relatively prime to N [30]. Due to the further constraint of N being a power of two, the constraint is for the determinant to be odd. The algorithm works with any square matrix $\boldsymbol{M} \in \mathbb{N}^{d\times d}$. The approach is to flip bits of as few of the matrix entries as possible thus changing some entries from even to odd and vice versa.

The basis for the algorithm approach is the Laplace expansion of the determinant calculation of a matrix:

$$|\boldsymbol{M}| = \sum_{i=1}^{d} (-1)^{i+j} m_{i,j} \boldsymbol{N}_{i,j}$$

where $m_{i,j}$ are the matrix entries and $\boldsymbol{N}_{i,j}$ are the minors of $\boldsymbol{M}$ obtained by generating a matrix of the elements of $\boldsymbol{M}$ omitting the entries of the row i and column j. Thus effectively crossing-out the entries of the ith row and jth column. Note that the size of the minor $\boldsymbol{N}$ is $d - 1 \times d - 1$. This technique is also sometimes referred to as the expansion by minors.

The other obvious observation is that only an odd number of odd terms yield an odd number. The algorithm is most easily understood with recursion which reduces the problem size in each iteration by one and ends in the trivial case of a 1×1 matrix which is just an integer. Such a matrix can be made odd by subtracting or adding one in case the number is even.

A full pseudo code description of the algorithm is given in Fig. 5.

A detailed description of the algorithm is given: First, the two trivial cases which stop the recursion: 1. The determinant of the input is odd 2. The input size is a single number ($d = 1$).

The second case can be solved by adding one to the input number which makes the number odd. The algorithm proceeds with inspecting the input's Laplace expansion. Subsequently, the number of odd terms in the expansion are counted. Note that, at that stage of the algorithm the number of odd terms must be even. If the number of odd terms is two or more, the solution is easy: Make one of the terms even and the overall determinant will become odd. Note, however, that the minor matrices of the terms are not disjoint and have common elements. Thus, flipping one of the elements of the minor matrices can result in a change in multiple terms. This in turn means that a flip in the matrices could result in the determinant staying even. For this reason, the factor $m_{1,i}$ of the expansion is used to make the overall term odd. It is know—due to the term being odd—that the factor $m_{1,i}$ must be odd as well. This stage solves the problem and terminates the recursive algorithm.

The next stage is to handle the case if the number of odd terms is zero. This means that the determinant can be made odd by making one single term odd. Overall, it is desirable to change the least amount of matrix entries which implies that the focus is on changing the factors $m_{1,i}$ as opposed to the minor matrices. Thus, the algorithm tries to find a term with a minor matrix which has an odd determinant. If successful, the factor $m_{1,i}$ is made odd as well. This terminates the algorithm since this will make one term odd and thus the overall determinant odd.

At this stage, there is no minor matrix with odd determinant, which implies that recursion on one minor matrix is necessary. The first step however, is to ensure that at least one factor $m_{1,j}$ exists which is odd. If this fails, then the first factor $m_{1,1}$ of the matrix is set to become odd. The next step is to find an odd factor $m_{1,j}$ and its accompanying minor matrix $\boldsymbol{N}_{1,j}$. It is known that the minor matrix $\boldsymbol{N}_{1,j}$ has an even determinant. Thus, the algorithm calls itself recursively which ensures that minor matrix has an odd determinant. Note that the problem size is now reduced by one since the minor matrix has size $d - 1 \times d - 1$.

The final line of the algorithm requires some attention: As noted earlier, changing a minor matrix's entries affects the other $d - 2$ minor matrices' entries. This, in turn, can lead to other terms of the expansion become odd. For instance, if there were no terms which were odd and one minor matrix is modified then this modification could yield two 2 odd terms. Thus effectively negating what was desired: An overall odd determinant. Interestingly, the case of two or more odd terms was already covered earlier in the algorithm. The matrix can be made odd by simply calling the algorithm recursively. Note that, the recursive call is *not* being done with a reduced input size but with the same $d \times d$ input matrix size. It first seems that this could potentially result in an infinite loop. However, the recursive call will *never* call itself recursively and is thus safe to do. This observation also proves the termination of the algorithm since the other recursion reduces the problem size each time.

Note that, there is a slightly more complex version of this algorithm possible. The proposed algorithm only does one Laplace expansion along the first row of the input matrix. A more sophisticated algorithm could potentially reduce the number of bit flips even more by considering other expansion along the rows and columns.

A. Error and Complexity Analysis

In this Section the runtime complexity of Algorithm 5 as well as the error introduced by the bit flipping is analyzed. The worst case runtime of the proposed algorithm is $O(d!)$. This is due to the complexity of the Laplace expansion which also has a complexity of $O(d!)$. In the worst case the algorithm recurses on one smaller minor matrices and expands each. Note, however, that for the use case of generating permutation matrices this seemingly high complexity is not an issue. For instance for a 3-dimensional sparse FFT the input matrix is only 3×3 and in fact considered constant for the algorithmic complexity of the sparse FFT algorithm.

Since the algorithm changes the entries of the input matrix it is of interest how much the output of the algorithm differs from the input. The absolute maximum error is bound by one. This is due to the careful choice of adding one to the even entries and subtracting one from the odd entries which effectively can be implemented as a bit flipping operation. This means, that even if the recursive algorithm changes entries multiple times the error is guaranteed to be at most one.

IV. Sparse FFT Algorithm

In this Section the core ideas of the one dimensional sparse FFT algorithm as introduced by [8] in 2012 are described. A more in depth explanation of all steps is described in Section V. Specifically, the proposed algorithm is based on what is often referred to as the exact k-sparse algorithm also called sFFT-3.0. However, the proposed changes to the algorithm are not specific to this version and are also applicable to the general k-sparse algorithm which is also sometimes referred to as sFFT-4.0. Firstly, the notation is introduced. The notation will partly be re-used for the multidimensional version of the algorithm in Section V.

Given a signal x of length N its discrete Fourier transform is denoted as $\hat{x}$. A signal is considered to be k-sparse if there are only k non-zero components in $\hat{x}$. Furthermore $\omega = e^{-2\pi\imath/N}$ is defined as the Nth root of unity. The set $\{0, \ldots, N-1\}$ is defined as $[N]$ and further $[N] \times [N]$ as $[N]^2$. The number of bins that are used to hash the Fourier coefficients is denoted by B.

The following paragraph describes the main ideas of one iteration of the sFFT-3.0 algorithm. The key idea of the sFFT algorithm is to hash the k coefficients into few buckets in sublinear time. This is achieved by using a carefully designed filter that is concentrated in time as well as in the frequency domain. Due to the sparsity of the signal and the careful selection of the number of bins, each bin is likely to only contain one coefficient after being hashed. After the coefficients of each bin are obtained the actual positions in the frequency domain are recovered by *locating* and *estimating*. The algorithm does this hashing twice and "encodes" the frequency of the coefficient into the phase difference between the two hashed coefficients. This technique achieves the *locating* part of the algorithm by decoding the phase and obtaining the frequency. Before the coefficients are hashed into buckets, the procedure (HashToBins) permutes the signal

x in the time domain by applying the permutation operator $P_{\sigma,a,b}$ which is defined as

$$(P_{\sigma,a,b}x)_i = x_{\sigma(i-a)}\omega^{\sigma b i}, \tag{8}$$

where the parameter b is uniformly random between 1 and N, σ is uniformly random odd between 1 and N, and a is 0 for the first hashing operation (HashToBins) and 1 for the second call to HashToBins. The constraining to odd values for σ is necessary in order for the permutation to be invertible.

With the use of some basic properties of the Fourier transform the following can be proved (page 5 of [8]):

$$\widehat{P_{\sigma,a,b}x}_{\sigma(i-b)} = \hat{x}_i\omega^{a\sigma i}. \tag{9}$$

Later a multidimensional version of this equation is derived. Informally, this equation states the following: A permutation, defined by equidistant sub-sampling in the time domain in addition to applying a linear phase, results in a (different) permutation in the frequency domain with a (different) linear phase. By carefully choosing the parameters of (9) it is possible to design the permutation such that the phase difference between the two hashed coefficients is linear in frequency. This property is then used to recover the coefficient exactly by using the quotient of two measurements with different parameter a.

A high level overview of the functions that divide the key steps of the sFFT-3.0 algorithm are the following [8]:

- HashToBins permutes the spectrum of $\widehat{x-z}$, then hashes to B bins, where z is the already recovered signal which is initially all zero.
- NoiselessSparseFFTInner runs HashToBins twice and *estimates* and *locates* "most" of $\widehat{x-z}$'s coefficients.
- NoiselessSparseFFT runs NoiselessSparseFFTInner multiple times until it finds $\hat{x}$ exactly.

The function NoiselessSparseFFTInner generates the random parameters for the permutation (among others) and passes it to HashToBins. The permutations are $P_{\sigma,0,b}$ for the first call of HashToBins and $P_{\sigma,1,b}$ for the second call respectively. The number of bins is denoted by B and gradually reduced with each call of NoiselessSparseFFTInner. HashToBins performs the low pass filtering on the signal which has a complexity of $O(B\log N)$. By carefully reducing B per iteration the 1D sFFT algorithm runs in time $O(k\log N)$. The reader is advised to see [8] for a more in depth description and proves of the 1D sFFT algorithm.

V. Multidimensional sparse FFT Algorithm

In this Section it is described how to extend the one dimensional sparse FFT from Section IV to multiple dimensions. A comprehensive pseudo code description of the d-dimensional algorithm is given in Fig. 6. First it is described what extensions to the concepts are necessary. Consecutively the algorithm is described in more detail.

As stated earlier, for simplicity the symbols are reused and the notation is redefined for the d-dimensional case. Let x be an $[N]^d$-dimensional signal with sparsity k. It is assumed that each Fourier coefficient $\hat{x}_{\boldsymbol{i}} \in -L,\ldots,L$ where $L \leq N^c$ for some constant $c > 0$. Let the number of bins that are used to hash the coefficients be $[B]^d$.

```
procedure HashToBins($x$, $\hat{z}$, $P_{M,a,b}$, $B$, $\delta$, $\alpha$)
    Compute $\hat{y}_{\boldsymbol{j}N/B}$ for $\boldsymbol{j} \in [B]^d$,
    where $y = G_{B,\alpha,\delta} \cdot (P_{M,a,b}x)$
    $\hat{y'}_{\boldsymbol{j}N/B} = \hat{y}_{\boldsymbol{j}N/B} - (\widehat{G'_{B,\alpha,\delta}} * \widehat{P_{M,a,b}z})_{\boldsymbol{j}N/B}$, $\boldsymbol{j} \in [B]^d$
    return $\hat{u}$ given by $\hat{u}_{\boldsymbol{j}} = \widehat{y'}_{\boldsymbol{j}N/B}$
end procedure
procedure NoiselessSparseFFTInner($x$, $k'$, $\hat{z}$, $\alpha$)
    Let $B = k'/\beta$, for sufficiently small constant $\beta$.
    Let $\delta = 1/(4N^2L)$
    Choose $\boldsymbol{M}$ by Algorithm 2                ▷ precomputed
    Choose $\boldsymbol{b}$ uniformly at random from $[N]^d$.
    for $i \leftarrow 0,1,\ldots,d$ do
        $a \leftarrow$ according to (11)
        $\hat{\boldsymbol{u}}_i \leftarrow$ HashToBins($x, \hat{z}, P_{M,a,b}, B, \delta, \alpha$)
    end for
    Compute $J = \{j : |\hat{u}_{0,j}| > 1/2\}$
    for $j \in J$ do
        for $p \leftarrow 1,2,\ldots,d$ do
            $\boldsymbol{a}_{p-1} \leftarrow \phi(\hat{u}_{0,j}/\hat{u}_{p,j})$  ▷ $\phi(\cdot)$ denotes the phase
        end for
        $\nu \leftarrow \mathrm{round}(\boldsymbol{a}\boldsymbol{M}^{-1}\frac{n}{2\pi})$
        Find $v$ in $\boldsymbol{M}^T(v-b) \mod N = \nu$       ▷ See (12)
        $\hat{w}_v \leftarrow \mathrm{round}(\hat{u}_o)$
    end for
    return $\hat{w}$
end procedure
procedure NoiselessSparseFFT($x$, $k$)
    $\hat{z} \leftarrow 0$
    for $t \in 0,1,\ldots,\log k$ do
        $k_t \leftarrow k/2^t$
        $\alpha_t \leftarrow \Theta(2^{-t})$
        $\hat{z} \leftarrow \hat{z}+$NoiselessSparseFFTInner($x, k_t, \hat{z}, \alpha_t$)
    end for
    return $\hat{z}$
end procedure
```

Fig. 6. Exact k-sparse d-dimensional algorithm.

The low pass filter—which has a general form approximating a rectangular in one dimension—needs to be extended to multiple dimensions. There are two popular and straightforward options: A hypersphere or a hypercube. The extension to multiple dimensions need to be performed very carefully due to the constraints of the filter. It is crucial that the filter has limited support in *both*, time domain as well as Fourier domain. It turns out that this poses a significant problem when defining the multidimensional filter which transition from one to zero not along a principal axis. For instance, a circular shaped low pass filter contains transitions along each direction (0° to 180°) whereas the rectangular filter only contains transitions along 0° and 90°.

Due to this unique limitation the low pass filter is defined as the dyadic product of the one dimensional low pass filter:

$$\boldsymbol{G}_{B,\alpha,\delta} = \bigotimes_{i=1}^{d} \boldsymbol{g'}_{B,\alpha,\delta} \tag{10}$$

where $\boldsymbol{g'}_{B,\alpha,\delta}$ denotes the same filter vector as described in Section VII of [8]. Thus approximating a hypercube.

Note that the actual complexity of this is much less than the seemingly $O(N^d)$ due to the limited support of the vector $\boldsymbol{g'}$ which is $O(B\log N)$ in the one dimensional case. In d dimensions the support of $\boldsymbol{G}_{B,\alpha,\delta}$ is thus $O(B^d\log^d N)$.

The fact that the phase difference between the two hashes is always a one dimensional entity even in a d-dimensional sample poses a problem. To be able to recover the frequencies in d-dimensions it is necessary to hash a total of $d+1$ times and encode each dimension in the calls $1, 2, \ldots, d$ to HASHTOBINS. This allows to *locate* the coefficient in d-dimensions by decoding each frequency component along each dimension separately.

The most interesting part of the algorithm and the focus of this paper is the very first part of each outer iteration: The permutation. It is necessary to extend the permutation (9) to multiple dimensions which is done with the following definition of the permutation operator $P_{\boldsymbol{M},\boldsymbol{a},\boldsymbol{b}}$:

$$\left(P_{\boldsymbol{M},\boldsymbol{a},\boldsymbol{b}}\boldsymbol{x}\right)_v = x_{\boldsymbol{M}(\boldsymbol{v}-\boldsymbol{a})}\omega^{\boldsymbol{v}^T \boldsymbol{M}^T \boldsymbol{b}} \tag{11}$$

where $\boldsymbol{M}$ is a matrix of size $d \times d$ that stretches the input signal $\boldsymbol{x}$. And $\boldsymbol{a}$ and $\boldsymbol{b}$ are the d-dimensional vectors counterparts of the one dimensional definition in (9). Note also that all of the vectors and matrices in (11) only contain integers. In order to recover the original Fourier coefficients, i.e. perform a reversible permutation the determinant of the matrix $\boldsymbol{M}$ needs be odd. For two dimensions, the matrix $\boldsymbol{M}$ can be interpreted as applying a shear and scale to the multidimensional input signal. This interpretation allows for some intuition regarding the optimal parameter: If the parameters are chosen randomly, and the shear along one dimension happens to dominate the transformation, what happens to an isotropic input spectrum? The answer is that in such an unfortunate case the permuted spectrum would results in a "banding" like accumulation of Fourier coefficients. This can be fatal for the performance of the algorithm since such bands of accumulated coefficients results in many collisions. The answer in choosing the optimal matrix $\boldsymbol{M}$ lies in the proposed Algorithm 2.

Furthermore, the parameter $\boldsymbol{a}$ is the only parameter that differs among each call to HASHTOBINS. For a given iteration i and vector index q:

$$a_{i,q} = \begin{cases} 0 & \text{if } i = 0 \\ 0 & \text{if } i \neq 0 \text{ and } q \neq i \\ 1 & \text{if } i \neq 0 \text{ and } q = i \end{cases} \tag{12}$$

where the iteration index i ranges from 0 to d.

Another part of the multidimensional algorithm requiring special attention is one step within NOISELESSSPARSEFFTINNER where the inverse to above permutation is needed. Again, the extension to multiple dimensions is not straightforward. Firstly, note that in the one dimensional sparse FFT algorithm the inverse is found with the extended Euclidean algorithm with the constraint of σ being odd which is the counterpart of the constraint that the determinant of the permutation matrix is odd. In order to simplify this Section the focus is only on the expression $\boldsymbol{M}\boldsymbol{v}$. The result is straightforward to extend to the form of (11). Remember that all elements in $\boldsymbol{M}$ are integers and further $\det \boldsymbol{M}$ must be odd for the expression to be a bijection. The goal is to find $\boldsymbol{v}$ in $\boldsymbol{M}\boldsymbol{v} \mod N = \boldsymbol{y}$. Where the modulo appears due to the simple fact that the DFT of a signal is implicitly periodic with the signal length N.

This problem turns out to be a congruence equation which can be solved by reducing this problem to a linear Diophantine equation [31]. First, a trick is used to get rid of the modulo operation:

$$\begin{gathered} \boldsymbol{M}\boldsymbol{x} \mod N = \boldsymbol{y} \\ \begin{bmatrix} m_{1,1} & \cdots & m_{1,d} & \mu_1 \\ \vdots & \ddots & \vdots & \vdots \\ m_{d,1} & \cdots & m_{d,d} & \mu_d \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \\ N \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} \boldsymbol{M}'\boldsymbol{v}' = \boldsymbol{y} \end{gathered} \tag{13}$$

where $\boldsymbol{\mu}$ is a vector of arbitrary integers. In this simple form the solution to the linear Diophantine equation can be found by computing the Hermite normal form of the matrix $\boldsymbol{M}'$ [32]. Note that the complexity of computing the Hermite normal form is quite high with $O(d^6)$. However, the dimensionality is usually quite low and considered constant for the complexity of the sparse FFT algorithm. Thus, it does not increase the complexity of the overall sparse FFT algorithm. In fact, the permutation matrices as well as its inverse permutation obtained with the HNF form can be precomputed by an application.

Next the relationship between the original signal $\boldsymbol{x}$ and the effect of the permutation (11) to the spectrum is derived:

$$\begin{aligned} (\widehat{P_{M,a,b}x})_{M^T(v-b)} &= \sum_{u\in[N]^d} \omega^{u^T M^T (v-b)} (P_{M,a,b}x)_u \\ &= \sum_{u\in[N]^d} \omega^{u^T M^T (v-b)} x_{M(u-a)} \omega^{u^T M^T b} \\ &= \omega^{v^T M a} \sum_{u\in[N]^d} \omega^{v^T M (u-a)} x_{M(u-a)} \\ &= \hat{x}_v \omega^{v^T M a} \end{aligned} \tag{14}$$

Again, this states that the applied permutation (11) in the time domain results in a permutation in the Fourier domain. Both, the frequency as well as the time domain, also apply a (different) phase, which is exactly what is needed for the algorithm to function by encoding the frequency in the phase component.

A. Complexity Analysis

The time complexity of the d-dimensional sparse FFT algorithm is similar to the one dimensional algorithm. Besides the straightforward extensions of the various parameters and entities of the one dimensional sparse FFT to multiple dimensions, there are a few steps that require more careful attention.

First, the $d+1$ calls to HASHTOBINS which encode the d dimensions of each coefficient need to be taken into account. Secondly, the d-dimensional FFT within HASHTOBINS results in a complexity of $O(B^d \log B)$. Taking into account that B is chosen as $O(k^{1/d})$ the result is a complexity of $O(k \log k^{1/d})$ for the FFT calculation within HASHTOBINS.

The main cost of the algorithm is the first iteration. Within this first iteration the application of the time domain filter is the dominating cost with $O(B^d \log^d N)$. Again, taking into account that B is chosen as $O(k^{1/d})$ the result is a complexity of $O(k \log^d N)$ Thus, given that $d+1$ iterations of

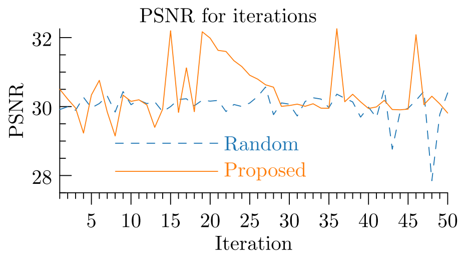

Fig. 7. An example graph depicting the PSNR over 50 iterations. The PSNR was calculated as the average over 40 generated input spectra. The input size $N \times N = 8192 \times 8192$ and the sparsity $k = 1600$. *Top*: In each iteration a randomly generated permutation *Bottom*: A subset of the proposed method DUALPERMCANDIDATES was used. Note: With only very few iteration the proposed algorithm finds a very good permutation matrix. The proposed algorithm also avoids bad permutation with low PSNR. A random strategy might or might not find a good permutation. This non-deterministic behavior is undesirable for real applications.

HASHTOBINS is necessary for one iteration and the fact that the complexity is dominated by the first iteration of NOISELESSSPARSEFFTINNER, the algorithm has an overall time complexity of $O(k \log^d N)$.

In comparison, the proposed method in [20] also uses Lattice theory to find the high-dimensional FFT and achieves a complexity of $O(dk^3 + dk^2 k^2 N \log(kN))$ for $\sqrt{N} \lesssim k \lesssim N^d$ and $O(dk^3)$ otherwise.

VI. RESULTS

The proposed algorithms are validated by simulating the sparse FFT algorithm in two dimensions. To this end the 2D sFFT algorithm was implemented in MATLAB. The simulation setup was as follows: Input spectra were generated with a given sparsity k. Each spectrum had an isotropic shape which is the most common shape among real world signals and should be assumed if no a-priori knowledge of the input signal is given. One iteration of the overall algorithm is run, i.e. one call to NOISELESSSPARSEFFTINNER. This allows one to compare the performance of the permutation. As an error measure the PSNR of the recovered signal after one iteration is calculated. A good permutation will reduce the number of collisions and result in a higher PSNR. Also note that the algorithm runtime is dominated by the first iteration.

Algorithm 2 is used to generated candidates for the permutation matrix of Algorithm 6. Each candidate is evaluated with 40 generated input spectra. An example of the evaluation of 50 candidates is depicted in Fig. 7. As a comparison 60 random candidates where evaluated and are depicted on the left side of Fig. 7. It can be seen that our proposed algorithm performs better finding a better maximum PSNR.

Fig. 8 shows the improvements for the proposed algorithm for different input sizes N. Again, the PSNR is compared to choosing random permutation matrices. It can be seen that the proposed algorithm finds permutation matrices that improve the PSNR of the sparse FFT algorithm by roughly 2 dB across the shown input sizes.

Similarly, Fig. 9 depicts the PSNR improvement over different sparsity k for and input spectrum of size $N \times N =$

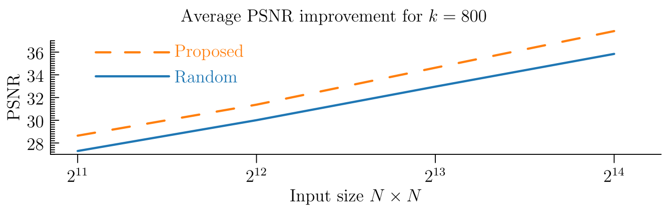

Fig. 8. 40 iterations of the proposed algorithm are used and compared it with a random strategy. The PSNR was calculated as the average over 40 generated input spectra. For the each iteration the average PSNR was calculated and the best performing permutation matrix chosen. For the shown graph the sparsity k was kept constant at 800. The graph shows that the proposed method improves the PSNR by roughly 2 dB.

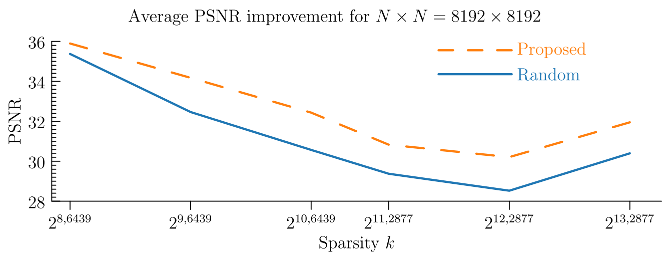

Fig. 9. This graph shows the improvement in PSNR for an input spectrum of $N \times N = 8192 \times 8192$ with different signal sparsity k ranging from 400 to 10000. The test setup is the same as the one of Fig. 8.

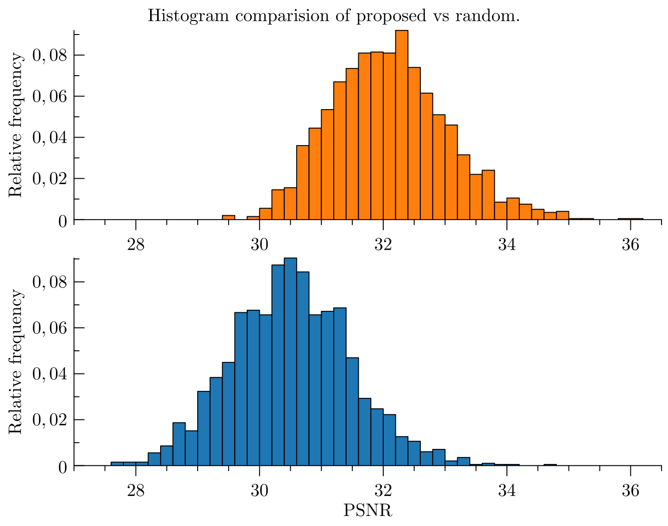

Fig. 10. This graph compares two histograms obtained from the optimal permutation matrix from the proposed method (top) and the optimal matrix obtained from randomly generating permutation matrices (bottom). The histogram shows the distribution of 2000 generated input spectra. It can be seen that the PSNR of the proposed method is well contained and improves upon the random permutation by roughly 2 dB.

8192×8192. Again, the improvement is around 2dB compared to a random permutation strategy.

Fig. 10 depicts the histogram of PSNR values over 2000 simulations. The top histogram shows the distribution of the PSNR with the permutation obtained from the proposed algorithm whereas the bottom shows the optimal permutation obtained from randomly selected permutations. Again, 40 iterations where employed. This shows that the PSNR values follow

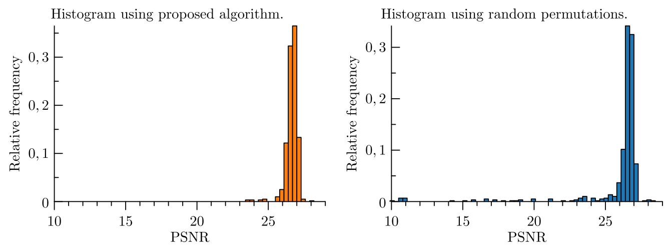

Fig. 11. Histograms depicting the difference between using different methods of choosing permutation parameters. Over 600 input spectra are generated with an isotropic input spectrum with a sparsity $k = 400$ and dimensions of 8192×8192. The PSNR is measured after one iteration of the sFFT algorithm in order to compare the performance of the permutation. *Left*: Choosing a random element of the DUALPERMCANDIDATES procedure as the permutation matrix. *Right*: Choosing a completely random permutation element. The left histogram shows that the PSNR is concentrated and successfully avoids very poor permutations which can occur with random parameters (right) and result in unpredictable algorithmic performance. The average PSNR is 26.65 dB on the left and 25.97 dB on the right. The minimum PSNR is 23.50 dB on the left and 10.05 dB on the right.

TABLE I
THIS TABLE SHOW THE IMPROVEMENT OF THE PROPOSED ALGORITHM BY AVOIDING COLLISION GENERATING PARAMETERS WHICH CAN RESULT IN A VERY POOR PSNR 600 INPUT SPECTRA WERE GENERATED AND ONE ITERATION OF THE ALGORITHM WAS RUN. THE TABLE SHOW THE MINIMUM PSNR ACROSS ALL 600 SIMULATIONS. σP AND σR SHOW THE CORRESPONDING STANDARD DEVIATIONS OF THE PROPOSED METHOD AND THE RANDOM METHOD RESPECTIVELY.

Sparsity k	Min. PSNR proposed	Min. PSNR random
200	22.93	8.35
300	24.49	13.47
400	23.50	10.05
600	22.52	9.85

a Gaussian like distribution which is desirable for real world applications that require predictable performance.

Next, a different approach is investigated. Instead of iterating over the sequence generated by Algorithm 2 and finding the optimal permutation the strategy of using a random sample of the sequence is used and is compared to taking a completely random permutation matrix. This turns out to be a better strategy since it avoids very poor permutations which in turn result in very poor PSNR. The result is depicted in Fig. 11. The histogram shows the PSNR of 600 simulations of an input spectrum of 8192×8192 and a sparsity of $k = 400$. This histogram shows that the very poor PSNR values are avoided. Note that this does not increase the runtime since picking a random element of the sequence of Algorithm 2 does not require generating the entire sequence and is thus a very cheap (constant time) operation. Thus, this strategy could be employed by libraries which have no prior knowledge of the input data.

Furthermore, a comparison of the minimum PSNR is shown in Table I. Showing that a randomly chosen permutation can result in poor performance. The proposed method mitigates these poor permutations successfully which is crucial for real world applications of the sparse FFT.

Concluding, an example of applying the two dimensional sparse FFT to an image is shown. Fig. 12. shows a crop of 1000 $\times$ 1000 pixels of an image of 32768 $\times$ 32768. Note that the

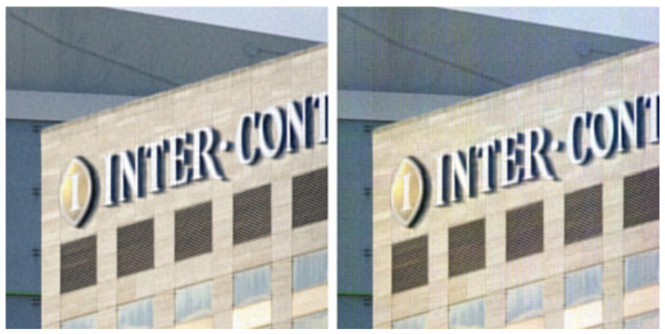

Fig. 12. *Left*: A 1000×1000 pixel crop of a 32768×32768 image with a sparsity of 1%. *Right*: The image after running the proposed sFFT algorithm. The PSNR is 27.81 dB when compared to the original image.

sparse FFT was independently applied to each RGB channel. Other spectral shapes, such as anisotropic shaped spectra were also evaluated and yielded similar results to the presented ones.

A. Higher Dimensions

Due to implementation details, no higher dimensional simulations ($d > 2$) were performed. However, intuitively the higher the dimensions the more likely it is to get a "banding" like accumulation of Fourier coefficients after applying the permutation. This is due to the fact that the permutation applies a shear to each dimension and only one unlucky parameter could result in large shear along one dimension and thus a banding like structure. This, in turn, would result in many collisions which can be avoided by using the proposed method.

VII. CONCLUSION

In this paper the one dimensional exact sparse FFT introduced in [8] is extended to multiple dimensions. Further the focus is on the permutation part of the algorithm which is the main subtlety and crucial to achieve good performance. This is well known in the Computer Science community where good performance for many algorithms are obtained by minimizing the collision rate of the hashing functions [1]. The focus is on the shape of the spectra of real world signals and it is shown that the performance can be optimized by carefully choosing the permutation parameters as opposed to the widely spread notion of randomly choosing the parameters.

The permutation operation is interpreted as generating a lattice which helps to argue the optimal parameters for the shape of many real world signal spectra. The results showed successfully that the proposed method avoids poor hashing permutation which can result in many collision. This is crucial in real world applications which often require a reliable performing algorithm. A clear understanding of the permutation and its inverse for the general d-dimensional case was established by solving a system of linear congruence equations.

Further, an algorithm was proposed which modifies a matrix to ensure that the determinant is odd. This novel algorithm was necessary in order for the congruence equations to be invertible. Practical guidelines were established for the permutation parameters in our proposed algorithm which are optimized for real world signals. The proposed method successfully avoids "bad" hashing permutation parameters which results in a more robust and consistent performing algorithm.

REFERENCES

[1] T. H. Cormen *et al.*, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001, vol. 2.

[2] H. Hassanieh, F. Adib, D. Katabi, and P. Indyk, "Faster GPS via the sparse Fourier transform," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 353–364.

[3] E. Kushilevitz and Y. Mansour, "Learning decision trees using the Fourier spectrum," *SIAM J. Comput.*, vol. 22, no. 6, pp. 1331–1348, 1993.

[4] A. Akavia, S. Goldwasser, and S. Safra, "Proving hard-core predicates using list decoding," in *Proc. 44th Annu. Symp. Found. Comput. Sci.*, 2003, vol. 44, pp. 146–159.

[5] A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss, "Near-optimal sparse Fourier representations via sampling," in *Proc. 34th Annu. ACM Symp. Theory Comput.*, 2002, pp. 152–161.

[6] M. A. Iwen, "Combinatorial sublinear-time Fourier algorithms," *Found. Comput. Math.*, vol. 10, no. 3, pp. 303–338, 2010.

[7] Y. Mansour, "Randomized interpolation and approximation of sparse polynomials," *SIAM J. Comput.,* vol. 24, no. 2, pp. 357–368, 1995.

[8] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Nearly optimal sparse Fourier transform," in *Proc. 44th Annu. ACM Symp. Theory Comput.*, 2012, pp. 563–578.

[9] A. C. Gilbert, P. Indyk, M. Iwen, and L. Schmidt, "Recent developments in the sparse Fourier transform: A compressed Fourier transform for big data," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 91–100, Sep. 2014.

[10] H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi, "GHz-wide sensing and decoding using the sparse Fourier transform," in *Proc. IEEE INFOCOM*, 2014, pp. 2256–2264.

[11] L. Shi, O. Andronesi, H. Hassanieh, B. Ghazi, D. Katabi, and E. Adalsteinsson, "MRS sparse-FFT: Reducing acquisition time and artifacts for in vivo 2D correlation spectroscopy," in *Proc. Int. Soc. Magn. Resonance Med. Annu. Meeting Exhib.*, 2013, p. 2019.

[12] X. Ma and G. R. Arce, *Computational Lithography*. New York, NY, USA: Wiley, 2011, vol. 77.

[13] W. Atwood, M. Ziegler, R. Johnson, and B. Baughman, "A time-differencing technique for detecting radio-quiet gamma-ray pulsars," *Astrophys. J. Let.*, vol. 652, no. 1, p. L49–L52, 2006.

[14] B. Ghazi, H. Hassanieh, P. Indyk, D. Katabi, E. Price, and L. Shi, "Sample-optimal average-case sparse Fourier transform in two dimensions," in *Proc. 51st Annu. IEEE Allerton Conf. Commun., Control, Comput. (Allerton)*, 2013, pp. 1258–1265.

[15] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Simple and practical algorithm for sparse Fourier transform," in *Proc. 23rd Annu. ACM-SIAM Symp. Discrete Algorithms*, 2012, pp. 1183–1194.

[16] A. C. Gilbert, S. Muthukrishnan, and M. Strauss, "Improved time bounds for near-optimal sparse Fourier representations," *Proc. SPIE*, vol. 4914, 2005, Art. no. 59 141A.

[17] A. C. Gilbert *et al.*, "A tutorial on fast Fourier sampling," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 57–66, Mar. 2008.

[18] A. Rauh and G. R. Arce, "Sparse 2D fast Fourier transform," in *Proc. 10th Int. Conf. Sampling Theory Appl.*, Jul. 2012.

[19] P. Indyk and M. Kapralov, "Sample-optimal Fourier sampling in any constant dimension," in *Proc. 55th IEEE Annu. Symp. Found. Comput. Sci.*, 2014, pp. 514–523.

[20] D. Potts and T. Volkmer, "Sparse high-dimensional FFT based on rank-1 lattice sampling," *Appl. Comput. Harmon. Anal.*, vol. 41, pp. 713–748, 2015.

[21] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, no. 4, pp. 515–534, 1982.

[22] M. Ajtai, "The shortest vector problem in L2 is NP-hard for randomized reductions (Extended Abstract)," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, 1998, pp. 10–19. [Online]. Available: http://doi.acm.org/10.1145/276698.276705

[23] R. Kannan, "Improved algorithms for integer programming and related lattice problems," in *Proc. 15th Annu. ACM Symp. Theory Comput.*, 1983, pp. 193–206. [Online]. Available: http://doi.acm.org/10.1145/800061.808749

[24] C.-P. Schnorr, "A hierarchy of polynomial time lattice basis reduction algorithms," *Theor. Comput. Sci.*, vol. 53, no. 2, pp. 201–224, 1987.

[25] P. Q. Nguyen and J. Stern, "The two faces of lattices in cryptology," in *Cryptography and Lattices*. New York, NY, USA: Springer, 2001, pp. 146–180.

[26] S. Dammertz, "Rank-1 lattices in computer graphics," Ph.D. dissertation, Inst. of Media Informatics, Ulm Univ., Ulm, Germany, 2009.

[27] J. Stoer and R. Bulirsch, *Numerische Mathematik*. New York, NY, USA: Springer, 1989, vol. 8.

[28] C. F. Gauß, "Besprechung des Buchs von LA Seeber: Intersuchungen über die Eigenschaften der positiven ternären quadratischen Formen usw," *Göttingsche Gelehrte Anzeigen*, vol. 2, pp. 188–196, 1831.

[29] S. Dammertz, H. Dammertz, and A. Keller, "Efficient search for two-dimensional rank-1 lattices with applications in graphics," in *Monte Carlo and Quasi-Monte Carlo Methods 2008*. New York, NY, USA: Springer, 2009, pp. 271–287.

[30] T. M. Apostol, *Introduction to Analytic Number Theory*. New York, NY, USA: Springer, 2013.

[31] L. J. Mordell, *Diophantine Equations*. New York, NY, USA: Academic Press, 1969, vol. 30.

[32] R. K. Martin, *Large Scale Linear and Integer Optimization: A Unified Approach*. New York, NY, USA: Springer, 1999.

André Rauh received the Dipl.Ing. degree from the University of Applied Sciences Esslingen, Esslingen am Neckar, Germany, in 2011 and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Delaware, Newark, Germany, in 2015.

His research interest include signal and image processing, compressive sensing and robust, nonlinear and statistical signal processing.

Gonzalo R. Arce (F'00) received the Ph.D. degree from Purdue University, West Lafayette, IN, USA. He is the Charles Black Evans Distinguished Professor of electrical and computer engineering with the University of Delaware, Newark. He served as the Department Chairman from 1999 to 2009. He held the 2010 Fulbright-Nokia Distinguished Chair in Information and Communications Technologies with Aalto University, Helsinki, Finland. He has held Visiting Professor appointments with Tampere University of Technology and the Unisys Corporate Technology Center. His research interests include statistical signal processing and computational imaging. He is the coauthor of books *Nonlinear Signal Processing* (Wiley, 2004), *Modern Digital Halftoning* (CRC Press, 2008), and *Computational Lithography* (Wiley, 2010). He has served as an Associate Editor of several journals of the IEEE, OSA, and SPIE.