ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Learning Set Functions that are Sparse in Better Non-Orthogonal Fourier Bases

Bachelor Thesis

E. Brusoni

August 21, 2022

Advisors: Prof. Dr. M. Püschel, B. Seifert, C. Wendler

Department of Computer Science, ETH Zürich

## Abstract

A recent line of work developed a family of algorithms for learning set functions efficiently under certain conditions. These algorithms exploit the sparsity of set functions in non-orthogonal Fourier domains. We contribute in two ways to this work. First, we introduce a new non-orthogonal Fourier transform and the associated extension to the mentioned algorithms. The new transform introduces weights along the Fourier basis vectors resulting in a better interpretable Fourier basis when we regard the Fourier domain as a Kernel space. The new resulting transform matrix also has a considerably lower condition number compared to its predecessors. Second, we provide implementations of six different types of set functions: Sensor placement tasks, preference functions for measuring contamination in water networks, preference elicitation in auctions, fitness functions, random forest regressors on binary input data and compiler flag optimization tasks, and use them to evaluate our novel algorithm.

# Contents

**Contents** **iii**

**1 Introduction** **1**

**2 Background** **3**
- 2.1 Discrete Set Function Fourier Transforms . . . . . . . . . . . . 3
- 2.2 Modified Tanimoto Kernel . . . . . . . . . . . . . . . . . . . . . . 7
- 2.3 Condition Number . . . . . . . . . . . . . . . . . . . . . . . . . 8

**3 Weighted Fourier Transform for Set Functions** **9**
- 3.1 Weighted DSFT3 . . . . . . . . . . . . . . . . . . . . . . . . . . 9
- 3.2 SSFTW3 Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . 12
  - 3.2.1 Determining the coefficients with known support . . . 13
  - 3.2.2 Determining the sequence of restrictions . . . . . . . . 14
  - 3.2.3 Efficient sampling . . . . . . . . . . . . . . . . . . . . . 15

**4 Empirical Evaluation** **19**
- 4.1 Sensor Placement . . . . . . . . . . . . . . . . . . . . . . . . . . 20
- 4.2 Learning Preference Functions . . . . . . . . . . . . . . . . . . . 20
- 4.3 Preferece Elicitation in Auctions . . . . . . . . . . . . . . . . . 22
- 4.4 Fitness Functions . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  - 4.4.1 Empirical Fitness Functions . . . . . . . . . . . . . . . . 24
  - 4.4.2 Simulated Fitness Functions . . . . . . . . . . . . . . . 26
- 4.5 Learning Random Forest Regressors . . . . . . . . . . . . . . . 27
- 4.6 Compiler Optimization Tasks . . . . . . . . . . . . . . . . . . . 30

**5 Conclusion** **33**

**Bibliography** **37**

Chapter 1

# Introduction

Numerous problems in machine learning on discrete domains involve learning so called set functions [1]. Set functions are mappings from subsets to the real numbers and appear naturally in many applications. For example in sensor placement tasks, where one wants to find most informative subset of sensors [1], or as the graph cut function, which associates with every subset of nodes in a weighted graph the sum of the edge weights that need to be cut to separate them [2], in recommender systems a set function can quantify the utility of every subset of items [3], in auction design every bidder can be modeled as a set function [4], and as fitness functions, which quantify the positive or negative effects of mutations on an organism [5].

But working with these functions is challenging, because of their exponential nature. Hence to make things tractable its crucial to exploit any structure available. One key property for efficiently learning set functions is sparsity in the Fourier domain [1, 6, 7]. A recent line of work, aiming at porting core signal processing theory to the discrete set domain, derived multiple Fourier transforms for set functions [8]. Applying those transforms on some relevant classes of set functions has been shown to result in a very sparse Fourier spectrum. Furthermore algorithms have been developed that compute these transforms efficiently, under certain conditions on the Fourier coefficients [1, 9]. Still these transforms have issues, one of the most notable ones is their really high condition number. Furthermore it is still of great interest to discover new classes of set functions, which are sparse in the Fourier domains from [8].

**Contributions.** Our contributions from this work are twofold. First we present an extension to the algorithms from [1] based on a novel Fourier transform. The new transform introduces weights along the Fourier basis vectors. When interpreting the Fourier basis as the kernel feature space for discrete sets, the introduction of weights allows for the definition of a much more interpretable similarity measure between the embedded samples. The resulting weighted Fourier basis has a considerably lower condition number compared to previous transforms, which could possibly help in reducing the susceptibility to noise. The second contribution is the testing of the newly introduced extension, and other algorithms from [1], on different classes of set functions. Some of these set functions contain noise, which is a problem, as due to the high condition number this could negatively effect the sparsity in the Fourier domains. Of the in total six function classes three were already evaluated in [1]: Sensor placement tasks, preference functions and auction elicitations. The other three classes are newly implemented and evaluated in this work: fitness functions, random forest regressors with binary input data and size of compiled programs with different flag activations.

In chapter 2 we will present the background knowledge necessary for understanding this work and in which context it was introduced. Then in chapter 3 we present our new Fourier basis and give a detailed description of the corresponding sparse set function Fourier transform (SSFT) derived from [1]. In chapter 4 we will then evaluate the sparsity of multiple transforms on the different classes of set functions to asses their practical relevance.

Chapter 2

# Background

In this chapter we present various topics related to signal processing for discrete sets, machine learning and linear algebra necessary for understanding this work. First, we will briefly describe what a set function is, which is central for us, since in this work we talk about learning set functions. Then we will present two of the set function Fourier transforms derived from discrete-set signal processing theory [8]. After that we will explain what Fourier sparsity is and how it can help us to represent a set function in a compact manner. Finally, in the last two sections we will introduce the modified Tanimoto kernel and the definition and meaning of the condition number of a matrix. These last two concepts are crucial to derive and motivate the new Fourier transform we introduce in this work.

## 2.1 Discrete Set Function Fourier Transforms

In this section we present a brief, simplified derivation of two of the Fourier transforms for set functions associated with discrete-set signal processing (SP) theory [8]. Discrete-set SP ports core SP concepts, such as convolutions, Fourier transforms and shifts, to the discrete-set index domain. In contrast to discrete-time signal processing where one works with signals indexed by time, in discrete-set SP we work with set function signals indexed by subsets of a discrete set.

**Definition 2.1 (Set Function)** *Let $N = \{x_1, \dots, x_n\}$ be a set of size $n$ and let $2^N$ be the power set of $N$, then a set function $s$ is a function of the form*

$$s : 2^N \to \mathbb{R};\ A \mapsto s(A) \tag{2.1}$$

*and $s$ can be identified with the signal vector*

$$\mathbf{s} = (s(A))_{A \subseteq N} \tag{2.2}$$

*of length* $2^n$*.*

Note that to define the signal vector $\boldsymbol{s} = (s(A))_{A \subseteq N}$ we need to choose an order on the subsets. In discrete-time SP signals are ordered in ascending time, as there is no clear ascending order for subsets we need an alternative. As in [8], we choose the lexicographic ordering on the Cartesian product. For example, $n = 3$ yields the following ordering:

$$\{\}, \{x_1\}, \{x_2\}, \{x_1, x_2\}, \{x_3\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1, x_2, x_3\}. \tag{2.3}$$

Discrete-set SP is derived from algebraic signal processing theory (ASP) [10, 11]. The aim of ASP is to enable the use of SP techniques to solve tasks such as coding, estimation, detection, compression, filtering and others, in other index domains other then the classical space or time domains. The axiomatic concept in ASP from which all others are derived is the shift operation.

**Discrete-set Shifts.** Recall the notion of shifts in the discrete-time index domain: Given a signal $(t(i))_{i \in \mathbb{N}}$ indexed by time, a shift by $x$ results in a signal $(t'(i))_{i \in \mathbb{N}} = (t(i - x))_{i \in \mathbb{N}}$. Similarly one can define a shift for signals indexed by subsets. Discrete-set SP defines multiple notions of shift in the set index domain. We now present two such shifts from [8] obtained from set union and difference operations.

**Definition 2.2 (Shift 3)** *Let* $(s(A))_{A \subseteq N}$ *be a set function signal defined on the powerset of* $N$ *and* $x_i \in N$*. Then the signal* $\boldsymbol{s}^{(3)}$ *obtained by applying shift 3 on* $s$ *is of the form*

$$\boldsymbol{s}^{(3)} = (s^{(3)}(A))_{A \subseteq N} = (s(A \setminus \{x_i\}))_{A \subseteq N} \tag{2.4}$$

As one can see shift 3 results in a naturally delayed signal. Similarly we define shift 4.

**Definition 2.3 (Shift 4)** *Let* $(s(A))_{A \subseteq N}$ *be a set function signal defined on the powerset of* $N$ *and* $x_i \in N$*. Then the signal* $\boldsymbol{s}^{(4)}$ *obtained by applying shift 4 on* $s$ *is of the form*

$$\boldsymbol{s}^{(4)} = (s^{(4)}(A))_{A \subseteq N} = (s(A \cup \{x_i\}))_{A \subseteq N} \tag{2.5}$$

Shift 4 results in a perfectly advanced signal. Note that shifts 3 and 4 have a corresponding shift matrix [8], which when multiplied with the signal vector results in the shifted signal vector. From shift 3 and shift 4 [8] then derives two different signal models, which we denote as models 3 and 4. In the following section we present the two Fourier transforms associated with models 3 and 4.

**Fourier Transforms.** In this section we present the Fourier transforms associated with the shifts 3 and 4.
A proper notion of Fourier transform jointly diagonalizes all filter matrices derived from the shift [8]. These filter matrices and their construction are presented in detail in [8] and are omitted here. To note is that filters are shift equivariant linear mappings and that to diagonalize the filter matrices it is sufficient to diagonalize the shift matrices. The matrices $\mathcal{F}^{(3)}$ and $\mathcal{F}^{(4)}$ defined below diagonalize the matrices of shift 3 and shift 4 respectively and are thus Fourier transform matrices.

**Definition 2.4 (DSFT3)** *The Fourier transform derived from shift 3 of a signal $\mathbf{s} = (s(A))_{A \subseteq N}$, with $|N| = n$, is defined as*

$$\widehat{\mathbf{s}}^{(3)} = \mathcal{F}^{(3)}\mathbf{s} \tag{2.6}$$

*with $\mathcal{F}^{(3)}_{BA} = (-1)^{|A|-|B|} \iota_{A \subseteq B}$ and $\mathcal{F}^{(3)-1}_{AB} = \iota_{B \subseteq A}$, resulting in Fourier and inverse Fourier transform matrices of the form*

$$\mathcal{F}^{(3)} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}^{\otimes n} \text{ and } \mathcal{F}^{(3)-1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n} \tag{2.7}$$

*where $\bigotimes n$ denotes the n-fold Kronecker product.*

Note that the **DSFT3** defined here is not the same as introduced in [8], but the Fourier basis vectors only differ by a scalar product and thus both versions span the same space. The formula for computing the coefficient $\widehat{s}^{(3)}(B)$ at a particular frequency $B \subseteq N$ is then:

$$\widehat{s}^{(3)}(B) = \sum_{A \subseteq B} (-1)^{|A|-|B|} s(A) \tag{2.8}$$

and the formula to evaluate $s(A)$ given the Fourier spectrum is then:

$$s(A) = \sum_{B \subseteq A} \widehat{s}^{(3)}(B). \tag{2.9}$$

**Definition 2.5 (DSFT4)** *The Fourier transform derived from shift 4 of a signal $\mathbf{s} = (s(A))_{A \subseteq N}$, with $|N| = n$, is defined as*

$$\widehat{\mathbf{s}}^{(4)} = \mathcal{F}^{(4)}\mathbf{s} \tag{2.10}$$

*with $\mathcal{F}^{(4)}_{BA} = (-1)^{A \cap B} \iota_{A \cup B = N}$ and $\mathcal{F}^{(4)-1}_{BA} = \iota_{A \cap B = \emptyset}$, resulting in Fourier and inverse Fourier transform matrices of the form*

$$\mathcal{F}^{(4)} = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix}^{\otimes n} \text{ and } \mathcal{F}^{(4)-1} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{\otimes n} \tag{2.11}$$

*where $\bigotimes n$ denotes the n-fold Kronecker product.*

The formula for computing the coefficient $\widehat{s}^{(4)}(B)$ of a particular frequency $B \subseteq N$ is then:

$$\widehat{s}^{(4)}(B) = \sum_{A \subseteq N,\ A \cup B = N} (-1)^{|A \cap B|} s(A) \tag{2.12}$$

and the formula to evaluate $s(A)$ given the Fourier spectrum is then:

$$s(A) = \sum_{B \subseteq N, A \cap B = \emptyset} \widehat{s}^{(4)}(B) \tag{2.13}$$

As mentioned discrete-set SP associates the **DSFT3** and **DSFT4** with the signal models 3 and 4. Hence sometimes we will be referring to them as Fourier transforms w.r.t model 3 and 4 respectively. Note that equivalent forms of these two transforms were also derived when studying specific applications [5, 12, 13]. For example the Taylor series for fitness functions in [5] is equal to the **DSFT3**. Also in [12] the derived transform, introduced to compute marginal posterior probabilities in Bayesian networks, is equal to the **DSFT4**.

**Fourier Sparsity.** In this short section we introduce the concept of (Fourier) sparsity.

**Definition 2.6** *A set function s is called k-sparse if*

$$supp(\widehat{s}) = \{B : \widehat{s}(B) \neq 0\} = \{B_1, ..., B_k\}, \tag{2.14}$$

*where $\widehat{s}$ is the Fourier spectrum of s.*

We call the set of frequencies associated with non-zero coefficients $supp(\widehat{s}) = \{B : \widehat{s}(B) \neq 0\}$ the support of s. Exactly learning a sparse set function is equivalent to finding its support and the values of the corresponding coefficients. We can then evaluate $s$ at every index in time linear in $k$. Thus in the case of $k \ll n$ knowing the support of $s$ and its corresponding coefficients is useful information, as we do not need an explicit look up table with $2^n$ entries or whatever costly way we had to evaluate $s$. For example, querying set functions involving hyperparameter or compiler flag optimizations can be very costly.

**Sparse Set Function Transforms.** Even in the case of $k \ll n$ a key challenge is to determine the support of $s$ and its coefficients efficiently. By simply performing the naive Fourier transform via matrix multiplication we would still evaluate the whole set function and perform a prohibitive number of operations. Hence more efficient methods are needed to perform Fourier transforms for set functions with a sparse spectrum.
There are various different methodologies and algorithms for solving this

task, see [1] for a more in depth discussion of related work. In our case we apply the algorithms presented in [1], which we collectively denote as **SSFT** algorithms. For our purposes we need the algorithms performing the Fourier transforms 3 and 4. We will denote these algorithms as **SSFT3** and **SSFT4** respectively. Note that all **SSFT** algorithms have the same query and time complexity.

**Lemma 2.7 (SSFT queries and complexity)** *[1, Theorems 10 and 11] Under the assumption that no cancellations between the Fourier coefficients occur, the algorithmic complexity of* **SSFT** *is $O(nk^2)$ and it requires $O(nk - k\log k)$ queries for k-sparse set functions.*

Note that the time and query complexity depend directly on the sparsity of the set function, i.e., for **SSFT** to work it is expected that $k \ll 2^n$.

**Weighted Fourier Basis.** Consider a ground set $N$ and two subsets $A, B \subseteq N$. It turns out, that when introducing weights of the form $w_{BA}$ to each Fourier basis entry $(\mathcal{F})^{-1}_{AB}$, where $\mathcal{F}^{-1}$ is a discrete-set Fourier basis, we obtain again a new Fourier basis [14]. We will exploit this fact in the next chapter by introducing a new Fourier basis with weights of this form.
Later we will motivate the introduction of the weights. For that reason in the following two sections we present the modified Tanimoto kernels and the notion of condition number, which will be useful to motivate new Fourier bases.

## 2.2 Modified Tanimoto Kernel

In this section we introduce so called (modified) Tanimoto kernel [15], which encodes the Jaccard similarity between two sets. But first we give a brief introduction to kernel methods.

**Kernel Methods.** Kernel methods are often applied in machine learning for enabling the use of linear methods for solving non-linear tasks. The dataset is embedded in some higher dimensional space using an embedding function $\Phi : \mathbb{R}^d \to \mathbb{R}^k$, where our original samples are in $\mathbb{R}^d$ and $k > d$. The idea is that while the task is not solvable with linear methods in $\mathbb{R}^d$, it will be in $\mathbb{R}^k$. With every embedding $\Phi$ there is an associated similarity measure, or kernel function, $\kappa$.

**Definition 2.8 (Similarity Measure)** *Given an embedding function $\Phi : \mathbb{R}^d \to \mathbb{R}^k$, where $k > d$, a similarity measure $\kappa$ is a function*

$$\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \kappa(x, y) \mapsto \sum_{i=1}^{k} \Phi(x)_i \Phi(y)_i \,. \tag{2.15}$$

Then the task is solved in using the similarity measure and without actually calculating the embedding, i.e, by finding a closed form for $\kappa$.

**Similarity between Sets.** A measure for representing the similarity between sets is the Jaccard similarity:

$$J(A_1, A_2) = \frac{|A_1 \cap A_2|}{|A_1 \cup A_2|} \,. \tag{2.16}$$

Two sets are the more similar the more elements they have in common, normalized by the number of elements in both sets. This normalization is introduced to compensate for varying set sizes.
Motivated by this intuitive measure, we can define useful similarity measures based on the Jaccard similarity, such as the generalized Tanimoto kernel [15]. In this work we use a modified version of the Tanimoto Kernel

**Definition 2.9 (Modified Tanimoto kernel)** *The modified Tanimoto kernel function $\kappa$ encodes the similarity between two sets $A_1, A_2$ as*

$$\kappa(A_1, A_2) = \frac{2^{|A_1 \cap A_2|}}{2^{|A_1 \cup A_2|}} \,. \tag{2.17}$$

## 2.3 Condition Number

Let $A$ be a matrix in $\mathbb{R}^{n \times n}$ and $x$ and $b$ two vectors in $\mathbb{R}^n$. Consider the linear equation $Ax = b$. The condition number *cond* is a value associated with the matrix $A$.

**Definition 2.10 (Condition Number)** *Let $A \in \mathbb{R}^{n \times n}$, the condition number of $A$ is then defined as*

$$cond(A) = \|A\| \, \|A^{-1}\|. \tag{2.18}$$

$cond(A)$ quantifies how sensitive the solution $x$ is to perturbations in $b$. To see this consider the following linear system:

$$A(x + \delta x) = b + \delta b, \tag{2.19}$$

where $\delta b$, $\delta x$ are the absolute errors in $b$ and $x$ respectively. We then have the following inequality:

$$\frac{\|\delta x\|}{\|x\|} \leq cond(A) \frac{\|\delta b\|}{\|b\|}. \tag{2.20}$$

One can see that given a relative error $b$, the relative error in the solution can be up to $cond(A)$ times that error. But also note that inequality (2.20) is only an upper-bound, i.e, the error is not bound to propagate with a factor of $cond(A)$ to the solution.

Chapter 3

# Weighted Fourier Transform for Set Functions

In this chapter we introduce a new Fourier transform. The transform we present, denoted as **WDSFT3**, is a weighted version of the **DSFT3** transform shown in the background section. We will start by motivating the introduction of weights and then culminate in the specific weights. With those weights we will define the **WDSFT3** transform. Finally we will present its associated **SSFT** algorithm, denoted as **SSFTW3**, for efficiently learning set functions that are sparse in the Fourier domain of **WDSFT3**.

## 3.1 Weighted DSFT3

In this section we first motivate the introduction of weights along the Fourier basis vectors of the inverse **DSFT3**, then we define the new Fourier transform **WDSFT3**.

**Fourier kernel space** To understand the motivation behind the weighted Fourier Transform we interpret (some) Fourier basis $\mathcal{F}^{-1}$ as a kernel space for subsets. Recall from the background section the embedding function $\Phi$, we then have the feature space embedding

$$\Phi : 2^N \to \mathbb{R}^{2^n};\ A \mapsto (\mathcal{F}^{-1}_{AB})_{B\subseteq N} \tag{3.1}$$

Recall also that for every embedding there is an associated similarity measure $\kappa$. Consider $\mathcal{F}^{-1} = \mathcal{F}^{(3)-1}$, for the embedding shown above the resulting similarity between two subsets $A_1$, $A_2$ is then

$$\kappa(A_1, A_2) = 2^{|A_1 \cap A_2|}. \tag{3.2}$$

A useful kernel used for dealing with powersets is the Tanimoto kernel [14] presented in the background section. A kernel function very similar to it, is the modified Tanimoto kernel we defined earlier. The Tanimoto kernel represents clearly a better similarity measure between subsets compared the one from equation (3.2). Hence we wish for a Fourier basis which results in such a similarity measure. Indeed there is a matrix $\mathcal{F}^{(W3)-1}$ where the resulting similarity measure is equal to the modified Tanimoto kernel [14].

**Lemma 3.1** *Consider $w_{BA} = \sqrt{3}^{|B|} 2^{-|A|}$. For*

$$\mathcal{F}^{(W3)-1}_{AB} = (w_{BA}) i_{B \subseteq A} \tag{3.3}$$

*we have that*

$$\kappa(A_1, A_2) = \frac{2^{|A_1 \cap A_2|}}{2^{|A_1 \cap A_2|}} \tag{3.4}$$

**Proof** The proof for this lemma was provided to us by the authors of [14]:

$$\begin{aligned}
\kappa(A_1, A_2) &= \sum_{B \subseteq A_1 \cap A_2} w_{BA_1} w_{BA_2} \\
&= \sum_{B \subseteq A_1 \cap A_2} \sqrt{3}^{|B|} 2^{-|A_1|} \sqrt{3}^{|B|} 2^{-|A_2|} \\
&= 2^{-(|A_1|+|A_2|)} \sum_{B \subseteq A_1 \cap A_2} 3^{|B|} \\
&= 2^{-(|A_1|+|A_2|)} \sum_{|B|=0}^{|A_1 \cap A_2|} \binom{|A_1 \cap A_2|}{|B|} 3^{|B|} \\
&= 2^{-(|A_1 \cup A_2| + |A_1 \cap A_2|)} (3+1)^{|A_1 \cap A_2|} \\
&= \frac{2^{|A_1 \cap A_2|}}{2^{|A_1 \cup A_2|}}
\end{aligned} \qquad \square$$

Note that the structure $\mathcal{F}^{(W3)-1}$ is equal to the one structure the Fourier basis of **DSFT3**, the only difference is the, always non-zero, weight $w_{BA}$.

**Condition number of the weighted version.** As we saw in the background section another property one wishes from a matrix is a low condition number. The following lemma states the condition number of $\mathcal{F}^{(W3)}$. Note that $cond(\mathcal{F}^{(W3)}) = cond(\mathcal{F}^{(W3)-1})$.

**Lemma 3.2** *The matrix $\mathcal{F}^{(W3)}$ has a condition number of $cond(\mathcal{F}^{(W3)}) = \sqrt{3}^n$, where $2^n$ is the number of rows and columns of $\mathcal{F}^{(W3)}$.*

**Proof** First recall the formula for the condition number $cond(A)$ of an arbitrary matrix $A$

$$\kappa(A) = \|A\|_2 \cdot \|A^{-1}\|_2 \tag{3.5}$$

We have that $\|A\|_2 = \sqrt{\max_{\lambda \in \sigma(A^T A)} \lambda}$. Where the value of the maximum eigenvalue $\lambda_{max}$ can be easily computed using the identities ${U^{\otimes n}}^T = {U^T}^{\otimes n}$, $(U \otimes U')(V \otimes V') = (UV \otimes U'V')$ and $\sigma(U \otimes V) = \{\mu\nu \mid \mu \in \sigma(U) \wedge \nu \in \sigma(V)\}$. We have

$$\begin{pmatrix} 1 & 0 \\ -\frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}} \end{pmatrix}^T \begin{pmatrix} 1 & 0 \\ -\frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}} \end{pmatrix} = \begin{pmatrix} 1 & -\frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} & \frac{5}{3} \end{pmatrix} \tag{3.6}$$

which has eigenvalues of 2 and $\frac{2}{3}$. With the same steps we see that $\mathcal{F}^{(W3)-1}$ has eigenvalues $\frac{3}{2}$ and $\frac{1}{2}$. Then we have $\sqrt{3} = \sqrt{\frac{3}{2}}\sqrt{2}$. $\square$

This is a considerable improvement over the Fourier basis of the **DSFT3**. Analogously as above it can be shown that $\mathcal{F}^{(3)}$ has a condition number of $\left(\frac{3+\sqrt{5}}{2}\right)^n$. To get a feeling of the improvement, take for example $n = 10$. Then $\kappa(\mathcal{F}^{(3)}) \approx 15127$ and $\kappa(\mathcal{F}^{(W3)}) = 243$.
With the better interpretable Kernel similarity measure and the lower condition number we now have a "better" non-orthogonal Fourier basis.

**WDSFT3.** As we saw there are various advantages in using a weighted Fourier basis. We now define the new weighted Fourier transform.

**Definition 3.3 (WDSFT3)** *The Fourier transform derived from introducing weights on $\mathcal{F}^{(3)}$ of a signal $\mathbf{s} = (s(A))_{A \subseteq N}$, with $|N| = n$, in the discrete-set index domain is defined as*

$$\widehat{\mathbf{s}}^{(W3)} = \mathcal{F}^{(W3)} \mathbf{s} \tag{3.7}$$

*with $\mathcal{F}^{(W3)}_{BA} = (-w_{BA})^{|B|-|A|} i_{A \subseteq B}$ and $\mathcal{F}^{(W3)-1}_{AB} = w_{BA} i_{B \subseteq A}$, resulting in Fourier and inverse Fourier transform matrix of the form*

$$\mathcal{F}^{(W3)} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}} \end{bmatrix}^{\otimes n} \text{ and } \mathcal{F}^{(W3)-1} = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix}^{\otimes n} \tag{3.8}$$

*where $\bigotimes n$ denotes the n-fold Kronecker product.*

The resulting Fourier and inverse Fourier transforms are then:

$$\widehat{s}^{(W3)}(B) = \sum_{A \subseteq B} (-1)^{|A|-|B|} \frac{1}{w_{BA}} \cdot s(A) \tag{3.9}$$

$$s(A) = \sum_{B \subseteq A} w_{BA} \cdot \widehat{s}^{(W3)}(B) \tag{3.10}$$

Note that for this Fourier transform, as for the previous two, there is an associated signal model. We denote this model as W3. Now that we have our Fourier transform we introduce the corresponding extension to the **SSFT** algorithms.

## 3.2 SSFTW3 Algorithm

Recall from the background section the definition of k-sparsity for set functions: A function is sparse if it has k frequency components, i.e., if it has k non-zero Fourier coefficients. Also recall that the aim of all **SSFT** algorithms is to calculate the support and Fourier coefficients of a sparse set function $s : 2^N \to \mathbb{R}$ with groundset $N = \{x_1, ..., x_n\}$ efficiently w.r.t. some Fourier transform.
In this section we will give a rough outline of how the **SSFTW3** algorithm works and present the most important ideas and results. As all **SSFT** algorithms are identical in their structure, we will first present a rough outline of how they operate. Then from this skeleton algorithm we will instantiate the **SSFTW3** algorithm. Note that all fundamental lemmas for implementing the **SSFT3** algorithm were already presented in [1], we only need to do some minor adjustments as we have a weighted version of the basis of **DSFT3**.

**Algorithm outline.** Algorithm 1 presents the skeleton of a **SSFT** algorithm. Let $\mathcal{F}$ be the transform matrix of some set function Fourier transform and $s_i : M_i \to \mathbb{R}$ a set function, where $M_i = \{x_1, \ldots, x_i\} \subseteq N$. We call $s_i$ is a restriction of $s$.
In short, the algorithm works the following way: It iterates once from 1 to $n$. At iteration $i$ it constructs the support of $s_i$ from the support of $s_{i-1}$ (lines 6,7), we discuss the details later. Then by querying $s$ and solving a linear system of equations it determines the Fourier coefficients of $s_i$,i.e., it reconstructs $\widehat{s_i}$ (lines 9,10). Finally it stores the support of $s_i$ for the next iteration (lines 11,12). At the $n$-th iteration we return $\widehat{s_n} = \widehat{s}$. Note that each $s_i$ is defined on $M_i$, thus at every iteration we add an element of $N$ to the domain of the restriction.
There are two main issues which have to be resolved to transform the skeleton algorithm to the **SSFTW3** algorithm:

1. Let $\mathcal{F}^{-1}_{\mathcal{AB}}$ be the submatrix in Algorithm 1 (line 10) of $\mathcal{F}^{-1}$ obtained by selecting the rows indexed by $\mathcal{A}$ and columns indexed by $\mathcal{B}$. Let $\mathcal{B}$ be the support of $s$. To reconstruct $s$ we have to determine $\mathcal{A}$ such that $\mathcal{F}^{-1}$ is invertible.

2. We have to define the sequence of restrictions of $s$, i.e., we have to define $(s_0, \ldots, s_n)$. As one can see from the pseudocode at line 11 in algorithm 1, for later iterations we only consider the frequencies in the support of a restriction. Hence it must hold that, if a frequency is not in the support of $s_i$ then it, or a superset of it, isn't in the support of $s_{i+1}$. In other words it must hold that

$$\widehat{s_i}(B) = \widehat{s_{i+1}}(B) + \widehat{s_{i+1}}(B \cup \{x_{i+1}\}) \tag{3.11}$$

for all $0 \leq i < n$. If the left side of the equation is zero, then the right side must be zero and so $B$ and $B \cup \{x_i\}$ cannot be in the support of $s_{i+1}$. Of course this assuming that there are no cancellations on the right hand side. This is a condition that must hold for the **SSFT** algorithms to work.

In the next two sections we will solve these two issues.

**Algorithm 1** Sparse set function Fourier transform of $s$

```
1: $M_0 \leftarrow \emptyset$
2: $s_0(\emptyset) \leftarrow s(I)$
3: for $i = 1, \dots, n$ do
4:     $M_i \leftarrow M_{i-1} \cup \{x_i\}$
5:     $\mathcal{B} \leftarrow \emptyset$
6:     for $B \in \text{supp}(\widehat{s_{i-1}})$ do
7:         $\mathcal{B} \leftarrow \mathcal{B} \cup \{B, B \cup \{x_i\}\}$
8:     end for
9:     $\mathbf{s}_{\mathcal{A}} \leftarrow (s(A))_{A \in \mathcal{A}}$
10:    $\mathbf{x} \leftarrow$ solve $\mathbf{s}_{\mathcal{A}} = \mathcal{F}^{-1}_{\mathcal{AB}}\mathbf{x}$ for $\mathbf{x}$
11:    for $B \in \mathcal{B}$ with $\mathbf{x}_B \neq 0$ do
12:        $\widehat{s_i}(B) \leftarrow \mathbf{x}_B$
13:    end for
14: end for
15: return $s_n$
```

### 3.2.1 Determining the coefficients with known support

First, we start with the easier task to determine the Fourier coefficients of a set function knowing its support. Given the support, or a super set of it, $\mathcal{B} \supseteq supp(\widehat{s})$ and an appropriate choice of $\mathcal{A} \subseteq N$, one can then exactly determine $\widehat{s}$, by solving the following system of linear equations:

$$s_{\mathcal{A}} = \mathcal{F}^{(W3)-1}_{\mathcal{AB}} \widehat{s}_{\mathcal{B}} \tag{3.12}$$

Where, as before, $s_{\mathcal{A}}$ and $\widehat{s}_{\mathcal{B}}$ denote the vectors of set function evaluations indexed by $\mathcal{A}$ and set function coefficients indexed by $\mathcal{B}$ respectively. $\mathcal{F}^{(W3)-1}_{\mathcal{AB}}$ is the inverse Fourier transform matrix where we only selected the rows indexed by $\mathcal{A}$ and the columns indexed by $\mathcal{B}$. Note that all non-zero coefficients of $\widehat{s}$ are in $\widehat{s}_{\mathcal{B}}$.
To compute the Fourier coefficients vector $\widehat{s}$ we need an appropriate choice of $\mathcal{A}$. Namely we want to select an $\mathcal{A}$ such that we can uniquely determine $\widehat{s}_{\mathcal{B}}$, i.e, we want $\mathcal{F}^{(W3)-1}_{\mathcal{AB}}$ to be invertible.

Let $\mathcal{B} = \{b_1, \dots, b_k\} \subseteq 2^N$. We consider the associated sampling operator

$$\mathcal{P}_{\mathcal{B}} : \mathbb{R}^{2^n} \to \mathbb{R}^k;\ \boldsymbol{s} \mapsto \boldsymbol{s}_{\mathcal{B}} = (s_{b_1}, \dots, s_{b_k})^T \tag{3.13}$$

**Lemma 3.4 (Sampling Lemma)** *Let s be k-Fourier-sparse with* $supp(\widehat{s}) = \{B_1, ..., B_k\} = \mathcal{B}$. *Then* $\mathcal{F}^{(W3)-1}_{\mathcal{B}\mathcal{B}}$ *is invertible and s can be perfectly reconstructed from the queries* $\boldsymbol{s}_{\mathcal{B}} = \mathcal{P}_{\mathcal{B}}\boldsymbol{s}$. *Namely,* $s = \mathcal{I}_{\mathcal{B}} s_{\mathcal{B}}$ *with* $\mathcal{I}_{\mathcal{B}} = \mathcal{F}^{(W3)-1}_{2^N\mathcal{B}} (\mathcal{F}^{(W3)-1}_{\mathcal{B}\mathcal{B}})^{-1}$.

**Proof** As $s$ has Fourier support $\mathcal{B}$. We have that

$$\boldsymbol{s} = \mathcal{F}^{(W3)-1}_{2^N\mathcal{B}} \widehat{\boldsymbol{s}}_{\mathcal{B}} \tag{3.14}$$

Applying the sampling operator on both sides yields

$$\boldsymbol{s}_{\mathcal{B}} = \mathcal{F}^{(W3)-1}_{\mathcal{B}\mathcal{B}} \widehat{\boldsymbol{s}}_{\mathcal{B}}. \tag{3.15}$$

What remains is to show that $\mathcal{F}^{(W3)-1}$ is invertible. $\mathcal{F}^{(W3)-1}$ is lower-triangular with diagonal elements at indices $(B, B)$ for $B \subseteq N$. Hence $\mathcal{F}^{(W3)-1}_{\mathcal{B}\mathcal{B}}$ is lower-triangular and thus invertible. □

Thus for **SSFTW3** at lines 9-10 we will have $\mathcal{A} = \mathcal{B}$.

### 3.2.2 Determining the sequence of restrictions

We have seen in the previous section that if we know the support of $s$, we just have to solve a linear system of equations to determine the Fourier coefficients. Now what remains is to determine the support of a set function $s$. For that purpose define a sequence $(s_0, \dots, s_n)$ of restrictions. We use the following restriction on $s$ as presented in [1]. Let $M \subseteq N$ with $|M| = m$ and $L = N \backslash M$, we have

$$s \downarrow_{L \cup 2^M} : 2^M \to \mathbb{R}; A \mapsto s(L \cup A) \tag{3.16}$$

Our sequence of restrictions is then

$$(s_0 = s \downarrow_{L\cup 2^{\emptyset}},\ s_1 = s \downarrow_{L \cup 2^{\{x_1\}}} \dots, s_n = s \downarrow_{L \cup 2^N}). \tag{3.17}$$

This restriction was first introduced in [1] to enable the **SSFT3** algorithm. Since we are using a weighted version of the **DSFT3** transform, it also works for us. This is shown formally by the following lemma, which relates the Fourier coefficients of a restriction with the coefficients of $s$.

**Lemma 3.5** *Modified from [1, Lemma 9]*

$$\widehat{s \downarrow_{M^c \cup 2^M}}^{(W3)}(B) = 2^{-|M^c|} \sum_{A \subseteq M^c} (\sqrt{3})^{|A|} \widehat{s}^{(W3)}(A \cup B) \tag{3.18}$$

Where $M^c = N \setminus M$.

**Proof** We have $\widehat{s\downarrow_{M^c\cup 2^M}}^{(W3)}(C) = s(C)$ per definition, for all $C \in 2^M$. Performing the Fourier expansion on both sides yields

$$\begin{aligned}\sum_{B\subseteq C} \sqrt{3}^{|B|} 2^{-|C|} \widehat{s\downarrow_{M^c\cup 2^M}}^{(W3)}(B) &= \sum_{B\subseteq M^c\cup C} \sqrt{3}^{|B|} 2^{-|M^c\cup C|} \widehat{s}(B) \\ &= \sum_{B\subseteq C} \sqrt{3}^{|B|} 2^{-|C|} \sum_{A\subseteq M^c} \sqrt{3}^{|A|} 2^{-|M^c|} \widehat{s}(A\cup B)\,.\end{aligned} \tag{3.19}$$

(3.18) is the unique solution for the system of $2^{|M|}$ equations given by (3.19).□

From the above lemma follows the equality

$$\widehat{s\downarrow_{M^c\cup 2^M}}^{(W3)}(B) = \frac{1}{2}\left(\widehat{s\downarrow_{(M\cup\{x\})^c\cup 2^{(M\cup\{x\})}}}^{(W3)}(B) + \widehat{s\downarrow_{(M\cup\{x\})^c\cup 2^{(M\cup\{x\})}}}^{(W3)}(B\cup\{x\})\right) \tag{3.20}$$

for some $x \in N \setminus M$. Note that this is exactly the property we wished for, as displayed in equation (3.11). Again, we now know that, assuming the right side doesn't cancel, if the left side is zero, the two components on the right side must also be zero. So $B$ and $B\cup\{x\}$ cannot be in $\text{supp}(\widehat{s\downarrow_{(M\cup\{x\})^c\cup 2^{(M\cup\{x\})}}}^{(W3)})$. We then conclude that we can construct $\mathcal{B} \supseteq supp(\widehat{s\downarrow_{(M\cup\{x\})^c\cup 2^{(M\cup\{x\})}}}^{(W3)})$ by performing the following union operation:

$$\mathcal{B} = \bigcup_{B\in\text{supp}(\widehat{s\downarrow_{M^c\cup 2^M}}^{(W3)})} \{B, B\cup\{x\}\} \tag{3.21}$$

Concretely, we can retrieve $\widehat{s}^{(W3)}$ by iterating through the chain of $n$ subproblems:

$$s\downarrow_{N\cup 2^{\emptyset}} = \widehat{s\downarrow_{N\cup 2^{\emptyset}}}^{(W3)}, \widehat{s\downarrow_{N\setminus\{x_1\}\cup 2^{\{x_1\}}}}^{(W3)}, \ldots, \widehat{s\downarrow_{\emptyset\cup 2^N}}^{(W3)} = \widehat{s}^{(W3)} \tag{3.22}$$

This under the assumption that no cancellations occur in the sum (3.18). Note that $s\downarrow_{N\cup 2^{\emptyset}}(\emptyset) = s(N)$, so in **SSFTW3** we have $I = N$ at line 2. By applying the lemmas in this and the previous section we result in the **SSFTW3** algorithm.

### 3.2.3 Efficient sampling

In the previous section we presented a rough outline of how the algorithm works. But in order to achieve the time and query complexity as in lemma (2.7) we need go in more detail.
In every iteration of the **SSFTW3** algorithm we have to solve the linear system

$$\mathbf{s}_{\mathcal{B}} = \mathcal{F}_{\mathcal{B}\mathcal{B}}^{(W3)-1} \widehat{\mathbf{s}}_{\mathcal{B}} \tag{3.23}$$

for $\widehat{\mathbf{s}}$. It turns out that at each iteration we can reuse some of the queries performed at the previous iteration. To be precise we can reuse exactly half of the queries. We introduce the notation $\mathcal{B} + x = \{B \cup \{x\} : B \in \mathcal{B}\}$ for sets of subsets $\mathcal{B} \subseteq 2^N$ and $\mathcal{B}_k = \operatorname{supp}(\widehat{s \downarrow_{M_k^c \cup 2^{M_k}}}^{(W3)})$. Further we observe that given $\mathcal{B}_{i-1} = \operatorname{supp}(\widehat{s \downarrow_{M_{i-1}^c \cup 2^{M_{i-1}}}}^{(W3)})$ and $T_{i-1} = \mathcal{F}^{(W3)-1}_{\mathcal{B}_{i-1}\mathcal{B}_{i-1}}$, we can compute $\mathcal{B}_i$ from a linear system constructed from $T_{i-1}$.

**Lemma 3.6** *Modified from [1, Lemma 8] Let $\mathcal{B} = \mathcal{B}_{i-1} \cup (\mathcal{B}_{i-1} + x_i)$. We have $\operatorname{supp}(\widehat{s \downarrow_{M_i^c \cup 2^{M_i}}}^{(W3)}) \subseteq \mathcal{B}$.*
*Let*

$$\mathbf{q}^{x_i} = (s\,(B))_{\mathcal{B}_{i-1}+x_i} \quad \textit{and} \quad \mathbf{q}^{\overline{x}_i} = (s\,(B))_{\mathcal{B}_{i-1}}. \tag{3.24}$$

*Then*

$$\mathcal{F}^{(W3)-1}_{\mathcal{B}\mathcal{B}} = \begin{pmatrix} T^{i-1} & 0 \\ \frac{1}{2}T^{i-1} & \frac{\sqrt{3}}{2}T^{i-1} \end{pmatrix} \tag{3.25}$$

*and the solution of*

$$\begin{pmatrix} T^{i-1} & 0 \\ \frac{1}{2}T^{i-1} & \frac{\sqrt{3}}{2}T^{i-1} \end{pmatrix} \begin{pmatrix} \widehat{\mathbf{q}}^{\overline{x}_i} \\ \widehat{\mathbf{q}}^{x_i} \end{pmatrix} = \begin{pmatrix} \mathbf{q}^{\overline{x}_i} \\ \mathbf{q}^{x_i} \end{pmatrix} \tag{3.26}$$

*contains the Fourier coefficients of $\widehat{s \downarrow_{M_i^c \cup 2^{M_i}}}^{(W3)}$ and is given by*

$$\begin{aligned} \widehat{\mathbf{q}}^{\overline{x}_i} &= (T^{i-1})^{-1}\mathbf{q}^{\overline{x}_i} \quad \textit{and} \\ \widehat{\mathbf{q}}^{x_i} &= (T^{i-1})^{-1}\frac{1}{\sqrt{3}}(2\mathbf{q}^{x_i} - \mathbf{q}^{\overline{x}_i}). \end{aligned} \tag{3.27}$$

**Proof** The proof is analogous to the one in [1]. Here we only list the most important differences. First as in [1] we introduce the functions $\Phi$ and $\rho$. Let $\varphi : 2^N \to \{0,1\}^n \subseteq \mathbb{R}^n$ be the mapping from sets to indicator vectors, i.e., for $A \subseteq N$, $\varphi(A)_i = 1$ if $x_i \in A$ and $\varphi(A)_i = 0$ if $x_i \notin A$. Let $\Phi$ denote the mapping from sets of subsets $\mathcal{A}$ to indicator matrices, i.e., $\Phi(\mathcal{A}) = (\varphi(A)^T)_{A\in\mathcal{A}} \in \{0,1\}^{|\mathcal{A}|\times n} \subseteq \mathbb{R}^{|\mathcal{A}|\times n}$. Let,

$$\rho : \mathbb{R} \to \mathbb{R}; \qquad a \mapsto \begin{cases} 1 & \text{if } a = 0, \\ 0 & \text{else.} \end{cases}$$

Then the first difference to [1] is the construction of the Fourier matrix. Which is to be expected since we use a different transform. Let $C_{\mathcal{A}\mathcal{B}}$ be a matrix indexed by elements of $2^N$ where $(C_{\mathcal{A}\mathcal{B}})_{AB} = 2^{-|A|}\sqrt{3}^{|B|}$ for $A \in \mathcal{A}$ and $B \in \mathcal{B}$. With the introduced notation in place, and the fact that if $B_1 \subseteq B_2$ then $B_2^c \cap B_1 = \emptyset$, we have

$$\mathcal{F}^{(W3)-1}_{\mathcal{B}\mathcal{B}} = \rho(\rho(\Phi(\mathcal{B}))\Phi(\mathcal{B})^T) \odot C_{\mathcal{B}\mathcal{B}}. \tag{3.28}$$

We also observe that

$$\Phi(\mathcal{B}) = \begin{pmatrix} \Phi(\mathcal{B}_{i-1}) \\ \Phi(\mathcal{B}_{i-1} + x_i) \end{pmatrix}. \tag{3.29}$$

With this insights and the introduction of $\mathcal{C}$ the proof becomes trivially similar to the proof of lemma 8 in [1]. □

Let $s_{i-1} := \widehat{s\downarrow_{M^c_{i-1}\cup 2^{M_{i-1}}}}^{(W3)}$ and $s_i := \widehat{s\downarrow_{M^c\cup 2^M}}^{(W3)}$. Recall that $\mathbf{q}_{i-1} = (s_{i-1}(B))_{B\in\mathcal{B}_{i-1}}$ and $s_{i-1}(B) = s(\{x_i \dots x_n\} \cup B)$. From the definition of set function restriction, it follows that:

$$s_i(B \cup \{x_i\}) = s(\{x_{i+1} \dots x_n\} \cup (B \cup \{x_i\}) = s_{i-1}(B)$$

Which means that some evaluations of $s_i$ are equal to evaluations of $s_{i-1}$, hence:

$$\mathbf{q}^{x_i} = \mathbf{q}_{i-1} \tag{3.30}$$

we can then conclude that we can reuse queries from iteration $i-1$ in iteration $i$.

Chapter 4

# Empirical Evaluation

We evaluate the **SSFTW3** algorithm together with **SSFT4** and **SSFT3** on six set function classes. The first three classes are the same as in [1]: Objective functions for sensor placement tasks, preference functions for contamination detection in water networks and functions representing simulated bidders in auctions.
After that we evaluate the three transforms on three new classes. First, we study the Fourier sparsity of empirical [16, 17] and simulated [18] fitness functions. After that we learn random forest regressors trained on binary input data obtained from [19]. Finally, we study set functions mapping subsets of gcc compiler's optimization options to the size in bytes of the compiled program. The object file sizes were obtained using compiler gym [20].
Note that in all our experiments we constrained the **SSFT** algorithms to make at most 1000 set function query accesses per iteration. This implies that at most 2000 Fourier coefficients are returned. We also report the threshold we used for each experiment, i.e., the value below which a coefficient is considered as zero.

**SSFT+ algorithm.** Before presenting the results we briefly introduce **SSFT+**, a variation of the **SSFT** algorithms for mitigating the effect of cancellations. Recall that for the **SSFT** algorithms to work, we assume that the Fourier coefficients do not cancel out.
The **SSFT**+ algorithms were introduced in [1]. They work by modulating the coefficients of $s$ so that they don't cancel by applying a one-hop filter on $s$. Then transforming the filtered function and attaining the true Fourier coefficients by applying the convolution theorem. The drawback is an increased query and time complexity. As always **SSFT3+** and **SSFT4+** refer to **SSFT+** instantiated in the respective models.

## 4.1 Sensor Placement

The first set function class we evaluate was already studied in [1] for assessing the performance of the **SSFT4** algorithm. Here we consider a discrete set of sensors located at different positions measuring a quantity of interest, e.g., temperature, amount of rainfall, or traffic data and want to find an informative subset of sensors subject to a budget constraint on the number of sensors selected.
In [1] the informativeness of a subset of sensors $A \subseteq N$ is quantified with the formula:

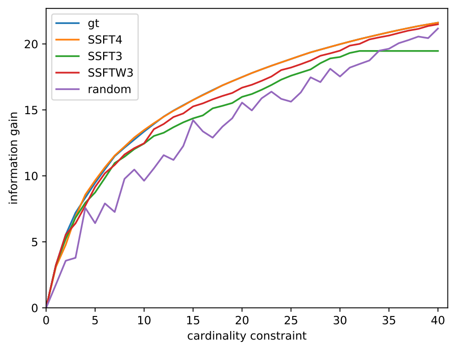
$$G(A) = \frac{1}{2} \log |I_{|A|} + \sigma^{-2}(K_{ij})_{i,j \in A}|, \tag{4.1}$$

where $(K_{ij})_{i,j \in A}$ is the submatrix of the covariance matrix $K$ that is indexed by the sensors $A \subseteq N$ and $I_{|A|}$ the $|A| \times |A|$ identity matrix. As said we want to find an informative subset of sensors constrained on the number of sensors. $G$ is approximately maximized by $A^* \approx \arg\max_{A \subseteq N : |A| \leq d} G(A)$ [1]. We wish to learn $G$ by finding its support in the Fourier domain using the **SSFT** algorithms. Let $s$ be the Fourier-sparse surrogate of $G$. We then also maximize $s$: $A^+ \approx \arg\max_{A \subseteq N : |A| \leq d} s(A)$ and compute $s(A^+)$. As in [1] we use $G(A_{rand})$ as baseline, where $A_{rand}$ is a random subset of $d$ sensors.
In [1] three set functions were constructed this way for temperature measurements from 46 sensors at Intel Research Berkeley, for velocity data from 357 sensors deployed under a highway in California and for 167 detecting rainfall amounts [1].

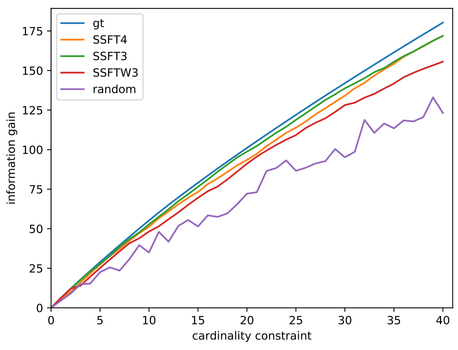
**Interpretation of results.** As was already shown in [1] model 4 performs really well in the sensor placement tasks. Nevertheless model 3 is comparable to it for the datasets *Rain* and *California*, thus indicating that set functions of this class are also sparse w.r.t model 3. For the *Rain* dataset model 3 performs even slightly better than model 4. Only for *Berkeley* model 3 performs poorly. **SSFTW3** performs reasonably well only on the *Berkeley* dataset, where it is better than model 3. But for datasets *Rain* and *California* it doesn't compare to models 3 and 4.

## 4.2 Learning Preference Functions

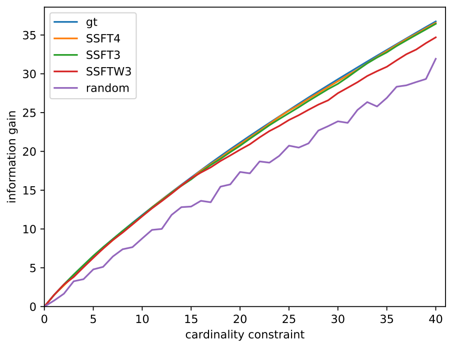
Again we look at a class of set functions already discussed in [1]. We consider a class of preference functions that are used for the cost-effective contamination detection in water networks (citation in [1]). A more understandable explanation of the set function class is presented in [1]. Here we only mention that we want to determine a cost-effective subset of sensors, by quantifying



**(a)** *Berkeley*, $n = 46$



**(b)** *California*, $n = 357$



**(c)** *Rain*, $n = 167$

**Figure 4.1:** Comparison of learnt surrogate objective functions on submodular maximization tasks subject to cardinality constraints (x-axis); On the y-axis we plot the information gain achieved by the informative subset obtained by the respective method. Threshold at 1e−3.

the informativeness of each subset of sensors with a set function of the form

$$p : 2^N \to \mathbb{R}; A \mapsto \sum_{\ell=1}^{L} \max_{i \in A} r_{\ell i}, \tag{4.2}$$

where $r$ is a matrix in $\mathbb{R}_{\geq 0}^{L \times n}$. Each row corresponds to an event (e.g., contamination of the water network at any junction) and the entry $r_{\ell i}$ quantifies the utility of the $i$-th sensor in case of the $\ell$-th event. We want to learn $p$.

We use the same $r \in \mathbb{R}^{3424 \times 12527}$ utility matrix as in [1]. For its source we refer to [1]. From this utility matrix we obtain subnetworks by selecting the columns that provide the maximum utility, i.e., we select the $|N| = n$ columns $j$ with the largest $\max_\ell r_{\ell j}$.
We applied the **SSFT** algorithm w.r.t models 3, 4 and W3 and show the amount of queries needed to retrieve all non-zero Fourier coefficients, the

**Table 4.1:** Comparison of model 4, 3 and weighted 3 sparsity of facility locations functions in terms of reconstruction error $\|\mathbf{p} - \mathbf{p}'\| / \|\mathbf{p}\|$ for varying $|N|$ and threshold at 1e−3.

| $\|N\|$ | $\alpha$ | queries | time (s) | $k$ | $\|\mathbf{p} - \mathbf{p}'\| / \|\mathbf{p}\|$ |
|---|---|---|---|---|---|
| 10 | **SSFT4** | 129 | 0.02 | 36 | 0.0 |
| | **SSFT3** | 113 | 0.03 | 65 | 0.08 |
| | **SSFT3+** | 1,466 | 0.21 | 279 | 0.02 |
| | **SSFTW3** | 1,024 | 0.13 | 1,023 | 0.0 |
| 20 | **SSFT4** | 735 | 0.24 | 102 | 0.0 |
| | **SSFT3** | 1,379 | 0.25 | 577 | 0.17 |
| | **SSFT3+** | 26,411 | 2.34 | 1,999 | 0.036 |
| | **SSFTW3** | 11,024 | 2.06 | 1,999 | 0.82 |
| 50 | **SSFT4** | 9,788 | 2.88 | 648 | 0.0 |
| | **SSFT3** | 27,421 | 5.10 | 1,999 | 2.3 |
| | **SSFT3+** | 613,913 | 70.48 | 1,999 | 2.26 |
| | **SSFTW3** | 41,024 | 14.27 | 1,999 | 1.0 |

number of coefficients retrieved, the execution time and the reconstruction error. The results can be seen in table 4.1.

**Interpretation of results.** Clearly models 3 and W3 do not perform as well as model 4. This is to be expected, since this class of set functions is by construction sparse w.r.t model 4 [1]. Only for $n = 10$ can **SSFT3** achieve comparable results with low reconstructions errors. Also the observation that **SSFT3+** has a lower reconstruction error than **SSFT3** indicates that cancellations occur. This becomes even more clear for $n = 20$, where the reconstruction error of **SSFT3** is significant, but still doesn't reconstruct the maximum number coefficients, whereas **SSFT3+** does.
**SSFTW3** is not able to achieve any good results in this function class, indicating that the class is not sparse w.r.t model W3. Only for $n = 10$ we get a reconstruction error of 0, but this is just due to the fact that **SSFTW3** reconstructed all $2^{10}$ coefficients. The only positive note being that there are apparently no cancellations.

## 4.3 Preferece Elicitation in Auctions

In combinatorial auctions a set of goods $N = \{x_1, \dots, x_n\}$ is auctioned to a set of $m$ bidders. Each bidder $j$ is modeled as a set function $b_j : 2^N \to \mathbb{R}$ that maps each bundle of goods to its subjective value for this bidder. The problem of learning bidder valuation functions from queries is known as the preference elicitation problem (citation in [1]). The experiment from [1], which we repeat with our **SSFT** algorithms, sketches an approach under the assumption of Fourier sparsity.

As described in [1] we resort to simulated bidders. Specifically, we use the multi-region valuation model (MRVM) from the spectrum auctions test

**Table 4.2:** Multi-region valuation model ($n = 98$). Each row corresponds to a different bidder type. Threshold at 1e−3.

| | number of queries (in thousands) | | | Fourier coefficients recovered | | | relative reconstruction error | | |
|---|---|---|---|---|---|---|---|---|---|
| B. type | **SSFT4** | **SSFT3** | **SSFTW3** | **SSFT4** | **SSFT3** | **SSFTW3** | **SSFT4** | **SSFT3** | **SSFTW3** |
| local | $4 \pm 4$ | $6 \pm 1$ | $89 \pm 0$ | $121 \pm 126$ | $278 \pm 75$ | $1,976 \pm 30$ | $0.5063 \pm 0.4937$ | $0.1184 \pm 0.0186$ | $1.0 \pm 0$ |
| regional | $17 \pm 1$ | $16 \pm 2$ | $89 \pm 0$ | $561 \pm 30$ | $739 \pm 83$ | $1,999 \pm 0$ | $0.0121 \pm 0.0034$ | $0.1109 \pm 0.0150$ | $1.0 \pm 0$ |
| national | $75 \pm 0$ | $22 \pm 0$ | $89 \pm 0$ | $1,027 \pm 3$ | $1,026 \pm 8$ | $1,999 \pm 0$ | $0.0115 \pm 0.0018$ | $0.1717 \pm 0.0902$ | $1.0 \pm 0$ |

| | number of queries (in thousands) | | Fourier coefficients recovered | | relative reconstruction error | |
|---|---|---|---|---|---|---|
| B. type | **SSFT4+** | **SSFT3+** | **SSFT4+** | **SSFT3+** | **SSFT4+** | **SSFT3+** |
| local | $232 \pm 51$ | $216 \pm 74$ | $323 \pm 72$ | $493 \pm 174$ | $0.0 \pm 0.0$ | $0.0303 \pm 0.0140$ |
| regional | $608 \pm 11$ | $548 \pm 102$ | $807 \pm 33$ | $1038 \pm 2$ | $0.0 \pm 0.0$ | $0.1382 \pm 0.1450$ |
| national | $3,543 \pm 2,881$ | $816 \pm 20$ | $1,037 \pm 4$ | $1024 \pm 11$ | $0.0103 \pm 0.0080$ | $0.4099 \pm 0.0693$ |

suite (citation in [1]). In MRVM, 98 goods are auctioned off to 10 bidders of different types (3 local, 4 regional, and 3 national). We learn these bidders using the prior Fourier-sparse learning algorithms, including **SSFT3+** and **SSFT4+**. Table 4.2 shows the results: means and standard deviations of the number of queries required, Fourier coefficients recovered, and relative error (estimated using 10,000 samples) taken over the bidder types and 10 runs.

**Interpretaion of results.** Also in this case the set functions class in sparser w.r.t. model 4. Cancellations occur for both models 3 and 4, but when applying the **SSFT+** algorithms model 4 is still considerably better than model 3. **SSFTW3** reaches the limit of computing 2000 coefficients, while having a huge relative error, which means that the underlying function is not sufficiently sparse for the algorithms to work.

## 4.4 Fitness Functions

Fitness functions, or fitness landscapes, map genotype to phenotype. The genotype of an organism is its complete set of genetic data. The phenotype is an observable trait. In our case it is a real number and a measure for fitness. For example, suppose we are interested in bacterial resistance to antibiotics. Given an ancestor genotype of the bacteria, we perform mutations on the bacteria, thus changing its genotype. We then, through experiments, measure the resistance to antibiotics after the mutation. This results in a fitness function mapping various genotypes of our bacteria to its phenotype (the antibiotic resistance).

Special cases of fitness functions are set functions. We then have a set $N = \{x_1, ..., x_n\}$ of "mutations" and a function $s : 2^N \to \mathbb{R}$ mapping from genotype to phenotype. For example $s(\{x_1, x_3\})$ is the fitness value if mutations $x_1$ and $x_3$ occurred in the genome. $s(\emptyset)$ is the fitness of the ancestor genotype, e.g., with no mutations.

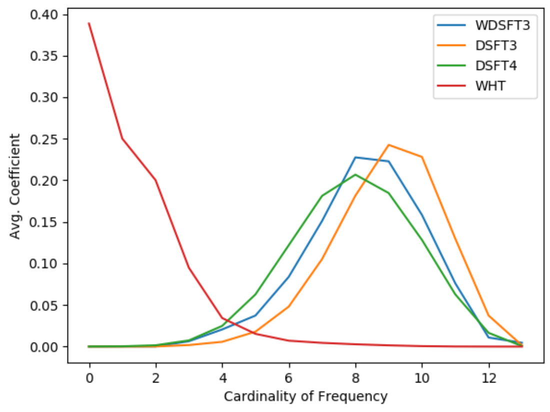


**Figure 4.2:** *Poelwijk*, $n = 13$

In the following sections we will briefly introduce the fitness function datasets, describe them shortly and present our results. First we will present the results on the empirical fitness functions. Then we will present our results on simulated fitness functions. In our case we used the widely adopted NK-model [18] to simulate fitness functions.

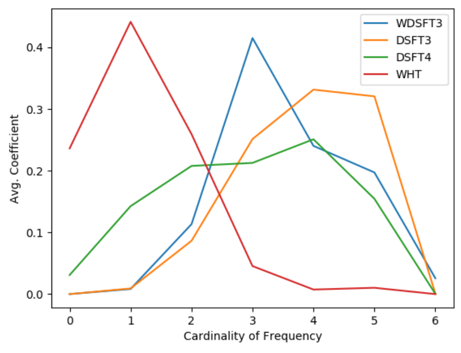
### 4.4.1 Empirical Fitness Functions

In this section we study fitness functions generated from empirical studies. Since the ground sets of all of these fitness functions isn't very large, we applied the "naive" discrete-set Fourier transform and simply multiplied the vectorized set function with the corresponding transform matrix. We did this for models 3, 4 and W3. We also transformed the set functions with the Walsh-Hadamard Transform (WHT), which is widely used for studying biological fitness functions [17, 21, 16]. We then plotted the scaled, average magnitude of all coefficients of a specific frequency cardinality.

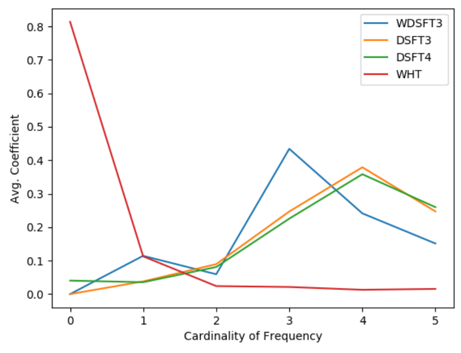
**The Poelwijk dataset.** This dataset, created by the authors of [16], maps all mutations at 13 different positions in a protein to the brightness of the protein. The results of applying our transforms can be seen in figure 4.2

**Collection of datasets.** This collection of many combinatorially complete empirical biological fitness functions is presented in [17]. The single datasets are from different studies, for their respective sources we refer to [17]. In figure 4.3 we present our results.

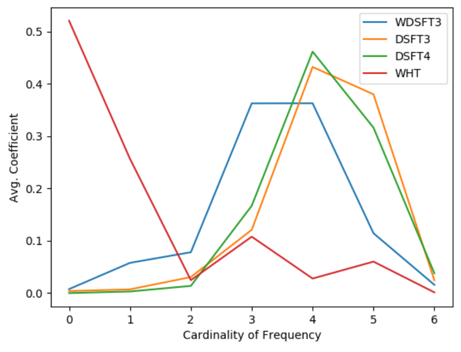
**Interpretation of results.** As one can clearly see our transforms are not performing too well on the empirical fitness functions. Only in a few datasets



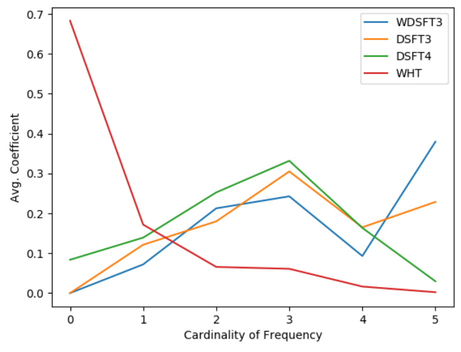
**(a)** *Bank*, $n = 6$



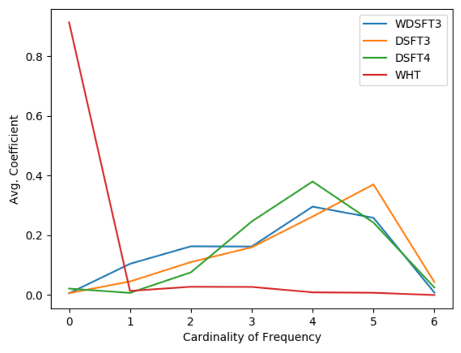
**(b)** *Devisser*, $n = 5$



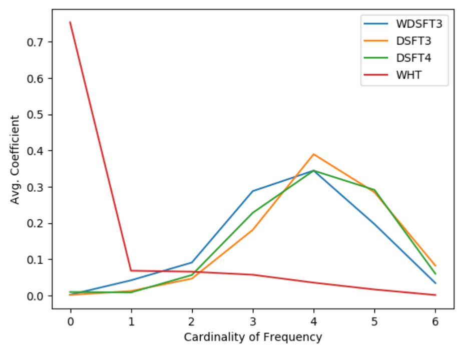
**(c)** *Hall* , $n = 6$



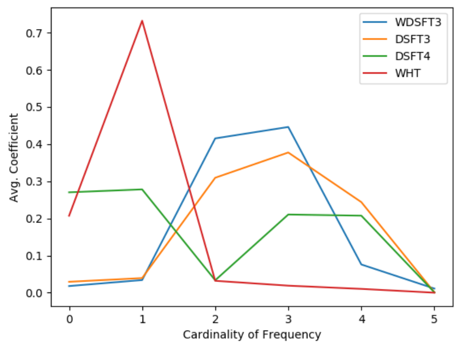
**(d)** *Khan*, $n = 5$



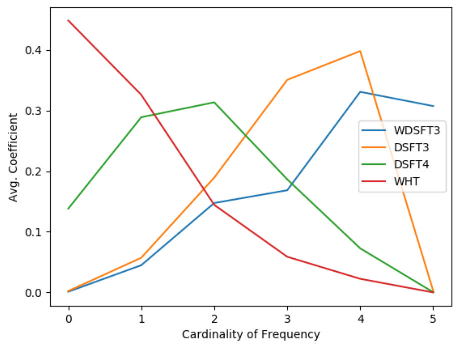
**(e)** *Omaille*, $n = 6$



**(f)** *Palmer*, $n = 6$



**(g)** *Weinreich*, $n = 5$



**(h)** *Whitlock*, $n = 5$

**Figure 4.3:** Magnitude of Fourier coefficients of various empirical fitness functions. On the x-axis we have the discrete-set "frequencies" in ascending order of cardinality, on y-axis the scaled absolute value of the corresponding coefficients.

such as *Devisser*, *Hall*, *Khan* and *Weinreich* one can see a resemblance of sparsity in the lower frequencies. Unfortunately this sparsity is not sufficient for the **SSFT** algorithms to reconstruct the function with a low reconstruction error, without using all frequencies. The WHT peforms better, and while it is not the focus of this work it is to note that SSFT can be performed with the WHT [1], but significantly better algorithms exist [6].

### 4.4.2 Simulated Fitness Functions

As mentioned above we also transformed simulated set functions generated using the NK-model. The NK-model, first introduced in [18], is widely used as a fitness function surrogate.
As explained in [22] a NK landscape, a fitness landscape generated with the NK-model, contains two parameters: $n$ and $k$. $n$ is the size of the ground set which corresponds to a set $N = \{x_1, \dots x_n\}$ of mutations. $k$ describes the number of epistatic interactions. Epistasis can be understood as the dependence of the effect of a mutation from the presence or absence of other mutations [22]. So clearly $0 \leq k \leq n-1$, where $k = 0$ means that all mutations are independent from each other and $k = n-1$ means that all mutations depend from each other.
It is easier to understand the role of the parameter $k$ when looking at the fitness value $NK(m)$ of a particular subset $m \subseteq N$ [22]. Where we interpret $m$ as a indicator sequence, which is 1 at index $i$ if mutation $x_i$ occurs and 0 otherwise.

$$NK(m) = \frac{1}{n} \sum_{i=1}^{n} f_i(n_i(m)) \tag{4.3}$$

(formula presented in [22]) with $n_i : C(n) \to C(k+1)$, $f_i : C(k+1) \to \mathbb{R}$ where $C(n)$ is the set of all binary sequences of length $n$, $n_i$ outputs the indicator sequence for $x_i$ and all mutations that interact epistatically with $x_i$ from the input sequence $s$ and $f_i$ calculates the fitness contribution of the $i$-th epistatic component.
For example consider $n = 3$ and $k = 1$. Then we have $N = \{x_1, x_2, x_3\}$. Assume $x_i$ and $x_{i+1 \mod 3} + 1$ build an epistatic component. The fitness value of $s = [1, 0, 1]$, which corresponds to the set $\{x_1, x_3\}$ is then

$$NK([1,0,1]) = \frac{1}{3}(f_1([1,0]) + f_2([0,1]) + f_3([1,1])). \tag{4.4}$$

The larger $k$ is the more variable and complex the NK landscape becomes, which means that the larger $k$ becomes the more rugged the NK landscape is [22].
For our evaluation we created NK landscapes where the epistatic components where chosen randomly and each value of $f_i$ was drawn at random from a normal distribution. The code implementation of the NK landscapes is based on the work of the authors of [23]. For our experiments we chose $n = 10$ and

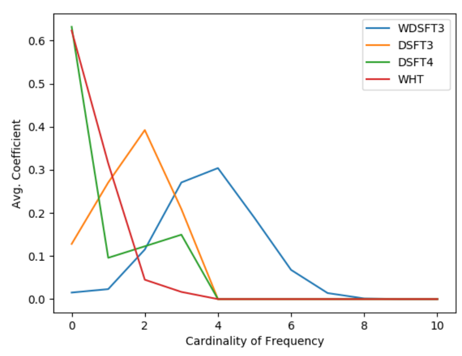
$k$ values from 2 to 4. As before we performed **DSFT3**, **DSFT4**, **WDSFT3** and WHT via matrix multiplications. The results can be seen in figure 4.4. We also transformed NK landscapes with a noise component, those are on the right side of figure 4.4.

**Interpretation of results.** Looking only at the exact NK landscapes (left column) we can see **DSFT3**, **DSFT4** and WHT perform quite well and behave equally regarding sparsity. The NK landscapes are very sparse for low $k$ values, but as expected get less and less sparse for larger $k$ as more and more randomness comes into play. Unfortunately the NK landscapes don't seem to be sparse in the Fourier domain w.r.t model W3.
WHT is the only transform that performs well for noisy NK landscapes, which is to be expected since the WHT matrix as defined in discrete-set SP theory [8] has a condition number of exactly one. This may explain why all transforms apart from WHT were not sparse for the empirical landscapes. Note that the plots in the noisy fitness landscapes for our transforms look fairly similar to the ones of the empirical fitness landscapes. This could indicate that noise is indeed the issue. Also consider that of course NK landscapes don't model empirical landscapes perfectly [22].
A motivating argument for **WDSFT3** was the low condition number and thus the hope of being less susceptible to errors, but fitness functions do not appear to be sparse w.r.t model W3 even in the noiseless case, so its hard to draw conclusions. Nevertheless the coefficient magnitude "shifts" to larger frequencies for model W3 too in the noisy landscapes, which could be an indication that its still strongly affected by noise.

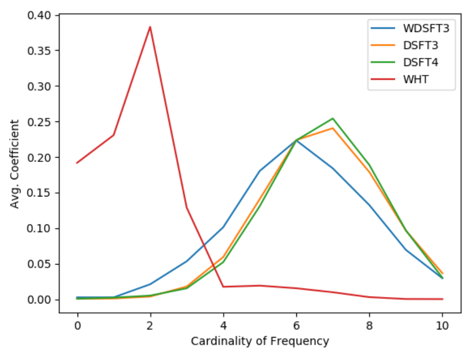
## 4.5 Learning Random Forest Regressors

For our next function class we perform the Fourier transform(s) of a random forest regressor (RFR). Recall that given a dataset $D \in \mathbb{R}^{n \times d}$, with $n$ samples and $d$ features, and labels $y \in \mathbb{R}^n$ the RFR aims to learn the function $f : \mathbb{R}^n \to \mathbb{R}$, where $f(D_i) = y_i$. For this task we use a dataset mapping 81 features of 21263 different superconductors to their corresponding critical temperatures from [19]. The set function $s$ to learn for this task was constructed the following way: First we binarized the features in the dataset $D$, obtaining $D_{bin} \in \{0,1\}^{n \times d'}$. Then we trained a RFR on $D_{bin}$ and $y$, obtaining a trained RFR $s$. The set function we learn is then $s : N \to \mathbb{R}$, with $N = \{x_1, \dots, x_{d'}\}$.

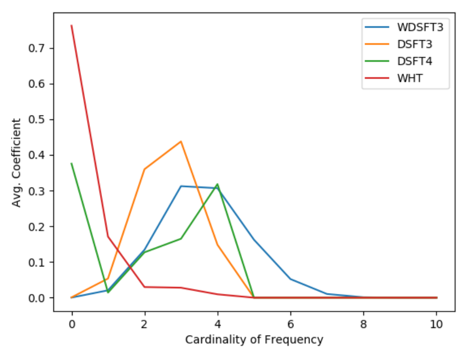
Each feature was binarized independently from one another to $m$ bit vectors, by applying the binarization method presented in [24]. Consider a feature vector $v = D_{:,i}$, e.g. the i-th column of $D$. Let $a$ and $b$ be the minimum and maximum values respectively in $v$. We then split the interval $[a, b]$ in $m$



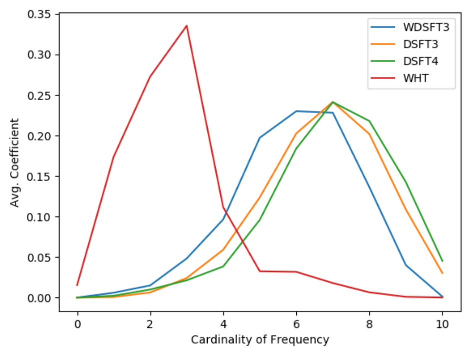
**(a)** $n = 10,\ k = 2$



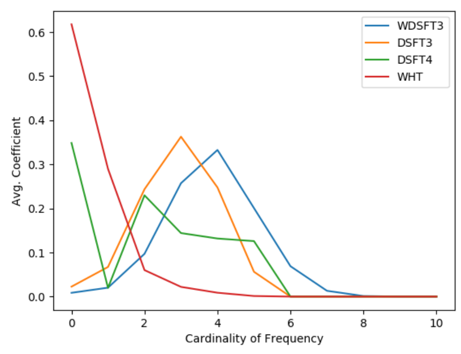
**(b)** $n = 10,\ k = 2$ (noisy)



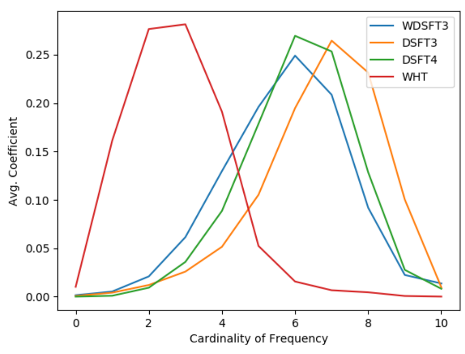
**(c)** $n = 10,\ k = 3$



**(d)** $n = 10,\ k = 3$ (noisy)



**(e)** $n = 10,\ k = 4$



**(f)** $n = 10,\ k = 4$ (noisy)

**Figure 4.4:** Avg. magnitude of all Fourier coefficients of frequencies of the same magnitude of NK landscapes. On the x-axis we have the frequency cardinality, on y-axis the scaled average absolute value of the corresponding. coefficients.

**Table 4.3:** Comparison of model 4, 3 and weighted 3 sparsity of random forest regressors in terms of reconstruction error $\|s - s'\|/\|s\|$ for varying parameters $m$, $maxdepth$ and threshold at $1\mathrm{e}{-8}$.

| $m$ | $maxdepth$ | | queries | time (s) | $k$ | $\|s - s'\|/\|s\|$ |
|---|---|---|---|---|---|---|
| 5 | 2 | **SSFTW3** | 396k | 60.73 | 2000 | 1.0 |
| | | **SSFT4** | 215k | 52.16 | 1000 | 0.017 |
| | | **SSFT3** | 220k | 46.62 | 1000 | 0.014 |
| 5 | 3 | **SSFTW3** | 396k | 46.75 | 2000 | 1.0 |
| | | **SSFT4** | 295k | 46.96 | 1000 | 0.043 |
| | | **SSFT3** | 296k | 38.37 | 1000 | 0.046 |
| 5 | 4 | **SSFTW3** | 396k | 48.12 | 2000 | 1.0 |
| | | **SSFT4** | 322k | 46.36 | 1000 | 0.058 |
| | | **SSFT3** | 326k | 42.04 | 1000 | 0.098 |
| 10 | 2 | **SSFTW3** | *failed* | | | |
| | | **SSFT4** | 458k | 86.97 | 1000 | 0.018 |
| | | **SSFT3** | 474k | 95.08 | 1000 | 0.012 |
| 10 | 3 | **SSFTW3** | *failed* | | | |
| | | **SSFT4** | 595k | 104.02 | 1000 | 0.039 |
| | | **SSFT3** | 610k | 107.02 | 1000 | 0.097 |
| 10 | 4 | **SSFTW3** | *failed* | | | |
| | | **SSFT4** | 656k | 102.22 | 1000 | 0.049 |
| | | **SSFT3** | 661k | 121.74 | 1000 | 0.23 |

evenly spaced intervals

$$[p_1 = a, p_2), [p_2, p_3), \dots, [p_{m-1}, p_m = b]. \tag{4.5}$$

Then naturally each entry of $v$ will be converted to a sequence of length $m$, with the $i$-th bit set to one if the entry is contained in the i-th interval. $D_{bin}$ will then be a matrix $\{0,1\}^{n \times md}$.
Again we applied the **SSFT3**, **SSFT4** and **SSFTW3** algorithms. We did this for various values of $m$ and different maximal depth ($maxdepth$) values. The results can be seen in table 4.3.

**Interpretion of results.** Both **SSFT3** and **SSFT4** perform well for shallow tress, e.g. maximal depths of 2 and 3. However when the trees become deeper both of the algorithms struggle to reconstruct the set functions perfectly. For the case $m = 10$ and $maxdepth = 4$ the error of **SSFT3** becomes too large, whereas for model 4 its considerably lower.
**SSFTW3** does not perform well on this class of functions. For $m = 5$ it reconstructs the maximal amount of coefficients, but the relative error is still maximal. We omitted the results for $n = 10$ as this set function class is clealy not sparse w.r.t model W3.

**Table 4.4:** Comparison of model 4, 3 and W3 sparsity of object file size task in terms of reconstruction error $\|\mathbf{s}-\mathbf{s}'\|/\|\mathbf{s}\|$ for varying $n$ and threshold at 1e−8.

| | $n$ | queries | time (s) | $k$ | $\|\mathbf{s}-\mathbf{s}'\|/\|\mathbf{s}\|$ |
|---|---|---|---|---|---|
| **SSFTW3** | 10 | 1,024 | 24.02 | 1,024 | 0.0 |
| | 20 | *failed* | | | |
| | 50 | *failed* | | | |
| | 100 | *failed* | | | |
| **SSFT4** | 10 | 19 | 19.70 | 2 | 0.0 |
| | 20 | 41 | 51.63 | 3 | 0.0 |
| | 50 | 304 | 324.77 | 15 | 0.0009 |
| | 100 | 1,238 | 1536.36 | 23 | 0.0011 |
| **SSFT3** | 10 | 19 | 24.02 | 2 | 0.0 |
| | 20 | 43 | 55.34 | 4 | 0.0 |
| | 50 | 272 | 342.37 | 15 | 0.0012 |
| | 100 | 1,614 | 1783.37 | 30 | 0.0011 |
| **SSFT4+** | 50 | 6,349 | 7965.34 | 34 | 0.0004 |
| | 100 | 148,686 | 270,455.60 | 143 | 0.0003 |
| **SSFT3+** | 50 | 8,178 | 8966.83 | 52 | 0.0004 |
| | 100 | *timeout* | | | |

## 4.6 Compiler Optimization Tasks

Another interesting class of set functions are mappings from a set of options flags of a compiler to various properties of the compiled program, such as size or execution time.

For our experiment we mapped a set of optimization flags of the gcc compiler to the size in bytes of the resulting compiled C program. As tool to extract this data we used compiler gym [20].
We then considered a set function $s : 2^N \to \mathbb{R}$ where $N = \{x_1, \dots, x_n\}$ is a set of $n$ randomly chosen option flags. $s$ maps each subset of activated flags to the size of the compiled object file.
In table 4.4 you can see our results for different $n$. Note that some flags have more than one option, we just chose the default option.

**Interpretation of results.** This class of set functions seems particularly sparse w.r.t models 3 and 4. Furthermore as the reconstruction error is always zero there occur no cancellations. The impressive performance of **SSFT4** and **SSFT3** can be partly attributed to the observation, that most of the option flags do not affect the size of object file. Hence it could be more interesting to perform these transforms with programs susceptible to the chosen ground set of flags. However **SSFT4+** and **SSFT3+** achieve slightly better results for $n = 50$, so there appear to be some cancellations. **SSFT4+** is also better than **SSFT4** for $n = 100$, where it discovers more coefficients and has a smaller relative error. **SSFT3+** for $n = 100$ timed out after we ran the experiment for three days.

**SSFTW3** does not perform well. For $n = 10$ it reconstructs all coefficients. We omitted results for greater $n$ as the functions are clearly not sparse w.r.t model W3.

Chapter 5

# Conclusion

We have implemented a new sparse set function Fourier transform and evaluated its performance, together with other algorithms of the same family, on multiple classes of set functions.

When looking at the Fourier space as a kernel space the **WDSFT3** seems a more sensible choice compared to the plain unweighted non-orthogonal transforms. However **SSFTW3** does not perform well on any of our set function classes. For the case of the exact function classes, such as the sensor placement task, auction elicitation functions, preference functions and object file sizes, the set functions were simply not sparse enough in the Fourier domain of **DSFTW3**. Only for one of the sensor placement tasks was it comparable to **SSFT3** and **SSFT4**. In the case of the empirical fitness landscapes one could attribute the poor performance to noise in the signal. But still even the noiseless NK-landscapes were dense w.r.t model W3, which suggests that the noise is not the only issue. Relevant set functions that are sparse w.r.t model W3 remain to be discovered. The conjecture that the sparsity w.r.t model W3 is less negatively affected by noise in the signal, because of the lower condition number, also remains to be confirmed or denied, as we couldn't observe any sparsity.
On the other hand most of the newly implemented function classes: the exact NK landscapes, object file size functions and random forest regressors, appear to be very sparse w.r.t models 3 and 4. Which further motivates the study of non-orthogonal transforms as in [8].

# Acknowledgements

I thank Prof. Dr. Markus Püschel for giving me the opportunity to work on this interesting topic.

I thank Dr. Bastian Seifert and Dr. Chris Wendler for supervising me, for their time and patience when answering my questions, for giving me access to a lot of useful resources and for revising my write up.

# Bibliography

[1] C. Wendler, A. Amrollahi, B. Seifert, A. Krause, and M. Püschel, "Learning set functions that are sparse in non-orthogonal Fourier bases," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 10283–10292, May 2021.

[2] M. H. Karci and M. Demirekler, "Minimization of monotonically levelable higher order mrf energies via graph cuts," *IEEE Transactions on Image Processing*, vol. 19, no. 11, pp. 2849–2860, 2010.

[3] S. Tschiatschek, J. Djolonga, and A. Krause, "Learning probabilistic submodular diversity models via noise contrastive estimation," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (A. Gretton and C. C. Robert, eds.), vol. 51 of *Proceedings of Machine Learning Research*, (Cadiz, Spain), pp. 770–779, PMLR, 09–11 May 2016.

[4] D. C. Parkes, *Iterative combinatorial auctions*. MIT press, 2006.

[5] E. D. Weinberger, "Fourier and Taylor series on fitness landscapes," *Biological cybernetics*, vol. 65, no. 5, pp. 321–330, 1991.

[6] A. Amrollahi, A. Zandieh, M. Kapralov, and A. Krause, "Efficiently learning Fourier sparse set functions," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[7] P. Stobbe and A. Krause, "Learning Fourier sparse set functions," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics* (N. D. Lawrence and M. Girolami, eds.), vol. 22 of *Proceedings of Machine Learning Research*, (La Palma, Canary Islands), pp. 1125–1133, PMLR, 21–23 Apr 2012.

[8] M. Püschel and C. Wendler, "Discrete signal processing with set functions," *CoRR*, vol. abs/2001.10290, 2020.

[9] R. E. Schapire and L. M. Sellie, "Learning sparse multivariate polynomials over a field with queries and counterexamples," *Journal of Computer and System Sciences*, vol. 52, no. 2, pp. 201–213, 1996.

[10] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory," *CoRR*, vol. abs/cs/0612077, 2006.

[11] M. Puschel and J. M. F. Moura, "Algebraic signal processing theory: 1-d space," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3586–3599, 2008.

[12] P. Parviainen and M. Koivisto, "Bayesian structure discovery in Bayesian networks with less space," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 589–596, PMLR, 13–15 May 2010.

[13] D. Rockmore, P. Kostelec, W. Hordijk, and P. F. Stadler, "Fast Fourier transform for fitness landscapes," *Applied and Computational Harmonic Analysis*, vol. 12, no. 1, pp. 57–76, 2002.

[14] Anonymous, "Causal Fourier analysis on directed acyclic graphs and posets," *submitted for publication*.

[15] S. S. E. Bach, "On the generalization of Tanimoto-type kernels to real valued functions," *CoRR*, vol. abs/2007.05943, 2020.

[16] F. J. Poelwijk, M. Socolich, and R. Ranganathan, "Learning the pattern of epistasis linking genotype and phenotype in a protein," *Nature communications*, vol. 10, no. 1, pp. 1–11, 2019.

[17] D. M. Weinreich, Y. Lan, J. Jaffe, and R. B. Heckendorn, "The influence of higher-order epistasis on biological fitness landscape topography," *Journal of statistical physics*, vol. 172, no. 1, pp. 208–225, 2018.

[18] S. A. Kauffman and E. D. Weinberger, "The NK model of rugged fitness landscapes and its application to maturation of the immune response," *Journal of Theoretical Biology*, vol. 141, no. 2, pp. 211–245, 1989.

[19] K. Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," *Computational Materials Science*, vol. 154, pp. 346–354, 2018.

[20] C. Cummins, B. Wasti, J. Guo, B. Cui, J. Ansel, S. Gomez, S. Jain, J. Liu, O. Teytaud, B. Steiner, Y. Tian, and H. Leather, "CompilerGym: Robust, Performant Compiler Optimization Environments for AI Research," in *CGO*, 2022.

[21] D. M. Weinreich, Y. Lan, C. S. Wylie, and R. B. Heckendorn, "Should evolutionary geneticists worry about higher-order epistasis?," *Current opinion in genetics & development*, vol. 23, no. 6, pp. 700–707, 2013.

[22] E. Pitzer and M. Affenzeller, *A Comprehensive Survey on Fitness Landscape Analysis*, pp. 161–191. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[23] A. C. Mater, M. Sandhu, and C. Jackson, "The NK landscape as a versatile benchmark for machine learning driven protein engineering," *bioRxiv*, 2020.

[24] Anonymous, "Beyond shapley values: Interpreting black-box models via Fourier sparsity," *submitted for publication*.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

LEARNING SET FUNCTIONS THAT ARE SPARSE IN BETTER NON-ORTHOGONAL FOURIER BASES

**Authored by** (in block letters):
*For papers written by groups the names of all authors are required.*

| | |
|---|---|
| **Name(s):** | **First name(s):** |
| BRUSONI | ENRICO |

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

| | |
|---|---|
| **Place, date** | **Signature(s)** |
| Zürich, 27.08.2022 | [illegible] |

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*