INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road Ann Arbor MI 48106-1346 USA
313 761-4700 800 521-0600

Order Number 9029986

Computing discrete Fourier transforms on a rectangular array consisting of ordinary and crystallographic-invariant data

Tsai, Dwen-Ren, Ph.D.

City University of New York, 1990

Copyright ©1990 by Tsai, Dwen-Ren. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Computing Discrete Fourier Transforms on a Rectangular Array Consisting of Ordinary and Crystallographic-Invariant Data.

by

Dwen-Ren Tsai

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1990

©1990

DWEN-REN TSAI

All Rights Reserved

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

April 6, 1990 — Date

Chair of Examining Committee

April 6, 1990 — Date

Executive Officer

Prof. Michael Vulis

Prof. Stefan A. Burr

Supervisory Committee

The City University of New York

Abstract

Computing Discrete Fourier Transforms on a Rectangular Array Consisting of Ordinary and Crystallographic-Invariant Data.

by

Dwen-Ren Tsai

Adviser: Professor Michael Vulis

The Weighted Redundancy Transform (WRT,[4]) algorithm for computing the multi-dimensional Discrete Fourier Transform (DFT) is generalized to the case in which the sample size (blocklength) is not the same on every axis. The proposed algorithm, similar to the WRT algorithm, is based on the one-dimensional Fast Fourier Transform (FFT) and, compared to traditional techniques of computing the multi-dimensional DFT, offers substantial savings in the number of one-dimensional FFT procedure calls. While the algorithm is applicable to transforms of any dimensions, only the two-dimensional case is explored in details in this thesis. Then, a customized algorithm for computing crystallographic-invariant DFT was constructed based on the WRT algorithm. This new algorithm has a number of advantages over existing crystallographic transforms and is applicable toward a substantially larger set of problems.

Preface

Much of the progress made in science and engineering during the past hundred years would not have been possible without the invention of Fourier transform. It is the most convenient and powerful tool in analyzing sinusoidal data in many fields of applied science, for example, radar, sonar, signal filtering , X-ray crystallography, ecology, and weather prediction.

The importance of the Fourier transform was not reflected in numerical computation until 1965 when Cooley and Tukey published a new algorithm, known as the Fast Fourier transform (FFT) algorithm, in their short paper [9]. This algorithm reduced the number of computations (multiplications) from $O(N^2)$ (straightforward calculation) to $O(NlogN)$. Many different refining algorithms were developed subsequently, however, all with the same order of computational complexity. A decade later, a potentially more efficient algorithm (from the view point of reducing the number of multiplications), known as Winograd Fourier transform algorithm (WFTA), was developed by Winograd [11,12]. Winograd's work not only reduces the computational complexity to $O(N)$ (for the prime-size case) but also provides a much deeper understanding of computing the DFT.

Multi-dimensional DFTs arise naturally from problems that are inherently multidimensional and also arise from the way a one-dimensional DFT is computed. Traditionally, a multi-dimensional DFT is computed by sequentially applying one-dimensional DFTs to all dimensions. This requires $s \cdot n^{s-1}$ calls of a one-dimensional DFT subroutine in computing an s-dimensional DFT of size n. The Winograd-based algorithms ([2], [3]) tend to minimize algebraic complexity. However, minimally does not guarantee efficient implementation. Moreover, in most cases these algorithms are defined only for prime-size DFTs, while most of the problems arising in real life require efficient subroutines for composite-size DFTs. It is true that one can use nesting (or tensor product) structures to combine these algorithms to produce composite size DFTs; however, the extra effort of matrix manipulations and bookkeeping outweigh the advantages arising from their arithmetic minimally.

In 1989, a new and relatively simpler and more efficient algorithm, named the Weighted Redundancy transformation (WRT), for computing multi-dimensional DFT of any size was proposed by Vulis [4]. This algorithm offers many advantages over others.

The problem of computing a multi-dimensional DFT with different

blocklengths on each axis (rectangular) frequently aries in real life. However, most of the efficient algorithms available, for example Nussbaumer-Quandalle FFT algorithm ([15], [16]), were designed just for the case in which blocklength is the same on each axis. This serves as the original motivation of my research in constructing an efficient general multi-dimensional DFT algorithm capable of handling all cases: prime or composite and square or rectangular.

In Chapter 1 of this thesis, we first generalize the WRT algorithm for computing the multi-dimensional DFT to the case where the blocklength may be different on each axis. Then, we apply this algorithm to customize a new multi-dimensional DFT for the crystallographic-invariant data in Chapter 2.

Acknowledgements

I would like to express my deep appreciation to my advisor Prof. M. Vulis for his continuous guidance throughout the course of this study, for his invaluable support and friendship, and for the opportunity to learn and work with him.

I would also like to thank the members of my Examining Committee, Prof. Anshel and Prof. Burr, for their constructive advice. Dr. Feig of IBM, who attended my second exam, is also appreciated.

Thank to Prof. Lucci of City College who offered me an adjunct job at CCNY that solved my financial problem.

Huey-Wen Su who added one more dimension to my life. I am grateful for those wonderful time we spent together. Without her help and constant encouragement I will still be several years before graduation.

In memory of my father

and

to my mother

Table of Content

Chapter 1.
Rectangular Data Array DFT

1.1. Introduction.

In this chapter we are going to construct a new general algorithm for computing the multi-dimensional Discrete Fourier Transform (DFT) where the blocklength may be different in each axis. Most of the results of this chapter were published in [6].

Let $A(x_1, x_2, \ldots x_s)$ be an s-dimensional complex-valued n_1 by n_2 by $\ldots$ n_s array, where $\forall i = 1, \ldots s,\ 0 \leq x_i < n_i$. Then the Discrete Fourier Transform of A, $\hat{A}$, is the complex-valued array defined as:

$$\hat{A}(y_1, y_2, \ldots y_s) = \sum_{x_1=0}^{n_1-1} \sum_{x_2=0}^{n_2-1} \cdots \sum_{x_s=0}^{n_s-1} \omega_{n_1}^{x_1 y_1} \omega_{n_2}^{x_2 y_2} \cdots \omega_{n_s}^{x_s y_s} A(x_1, x_2, \ldots x_s) \qquad (q1)$$

where $\omega_m = e^{\frac{2\pi i}{m}}$ and $0 \leq y_j < n_j$.

The traditional approach to the s-dimensional DFT is to consecutively apply one-dimensional n_j-point Discrete Fourier Transforms to all one-dimensional subarrays of A. For instance, in the two-dimensional case ($s = 2$),

one first computes the one-dimensional n_1-point DFT on all (n_2) rows of A and then computes the one-dimensional n_2-point DFT on all (n_1) rows of the result. In the "square" case ($n = n_1 = n_2$), $2n$ n-point one-dimensional DFT calls are required. Winograd based multi-dimensional algorithms ([2],[3]) reduce the number of these one-dimensional calls to $n+1$ for prime n, the theoretically minimal algebraic complexity. The implementations of these algorithms indeed demonstrate the theoretically-expected improvements for the prime size DFTs. Nesting (or tensor producting) of these algorithms produces multi-dimensional composite-size DFT algorithms. However, in the composite-size case, the extra effort needed for matrix manipulations and bookkeeping has been observed to increase superlinearly.

The WRT generalizes these savings to the case of non-prime n ([4]). It is a relatively efficient and easily programmable algorithm which approaches (and in a number of cases achieves) the minimal algebraic complexity. Further, the WRT is applicable to any data samples sizes. The WRT was introduced in [4] only for the "square" cases, i.e. when $n_1 = n_2 = \ldots = n_s$. This thesis treats the more general "rectangular" case. While the sequence of computations in the rectangular WRT (r-WRT) is similar to the one in the square case, the

underlying data structures are substantially more complicated, and therefore more theory is needed to validate the algorithm.

To keep the notation simple, in this thesis we present details of the algorithm and of the underlying theory only for the two-dimensional case. Furthermore, to keep the discussion general, we do not discuss either efficient machine-dependent implementations of the algorithm or the improvements possible for particular sample sizes. This information will be presented in future papers.

In Section 2, we introduce the notation and terminology for this algorithm. In Section 3, the r-WRT algorithm is described in detail. The derivation of the first major parameters of this algorithm – the number of lines (which is the number of one-dimensional DFT routine calls required) –is detailed in Section 4; while the derivation of the second major parameter – the weight of each point (which is the divisor of precomputed divisions) – is spread over Section 6 and the Appendix. (This computation is similar to the derivation of the number of lines.) Section 5 provides the point mapping from prime cases to the corresponding composite case. A sample FORTRAN program can be found in the Appendix.

1.2. Notation

We first introduce the terminology and notation needed for the algorithm.

Definition. Let $I = I_{mn}$ be the direct sum of Z/mZ and Z/nZ, $Z/mZ \oplus Z/nZ$. I_{mn} is said to be an m-by-n index rectangle.

The underlying idea of the r-WRT is first introduced informally. A multi-dimensional DFT algorithm that uses the one-dimensional DFT has to "cover" the index rectangle by "one-dimensional objects" on which the one-dimensional DFT will take place. The straightforward multi-dimensional DFT covers the rectangle with rows and columns. The WRT algorithm instead uses **lines**.

Definition. By a **line** we mean a maximal cyclic subgroup in I, $\lambda(\vec{t}) = \{a \cdot \vec{t} | a \in Z\}$, where $\vec{t} \in I$. We say that $\vec{t}$ is a generator of $\lambda(\vec{t})$. The order of the subgroup generated by $\vec{t}$, or the "length" of the line $\lambda(\vec{t})$ will be denoted as $l(\vec{t})$.

Example 1.1. There are four lines in $I_{4,2}$:

$$\lambda((1,0)) = \{(0,0),\ (1,0),\ (2,0),\ (3,0)\},$$

$$\lambda((1,1)) = \{(0,0),\ (1,1),\ (2,0),\ (3,1)\},$$

$$\lambda((0,1)) = \{(0,0),\ (0,1)\},$$

$$\lambda((2,1)) = \{(0,0),\ (2,1)\}.$$

Notice that $\{(0,0),\ (2,0)\}$ is a subgroup of I but not a line, since it is properly contained in $\lambda((1,0))$.

Definition. Let $\Psi(m,n)$ denote the total number of different lines in rectangle I_{mn}. Let $\Psi_k(m,n)$ denote the number of different k-element lines.

Example 1.2. $\Psi(4,2) = 4$; $\Psi_2(4,2) = \Psi_4(4,2) = 2$ (see above).

Remark. In [4], $\Psi(n)$ denoted the number of lines in an $n \times n$ square. In §4 we obtain the formulas for computing $\Psi(n,m)$ which generalize those for computing $\Psi(n)$ in [4].

Definition. Let $W_{m,n}(\vec{t})$ denote the number of different lines in I_{mn} which contains $\vec{t} \in I_{mn}$. We call $W_{m,n}(\vec{t})$ the weight of $\vec{t}$. We will sometimes abbreviate $W_{mn}(\vec{t})$ as $W(\vec{t})$.

Example 1.3. In the 4×2 rectangle, the weights of elements (x, y) are as follows:

$x \backslash y$	0	1
0	4	1
1	1	1
2	2	1
3	1	1

The values of Ψ and W are crucial for the implementation of the algorithm. The greater part of this thesis is a rather technical computation of these values (§1.4–§1.6). We will, however, delay this computation and first present the r-WRT algorithm.

1.3. The Rectangular WRT Algorithm.

Any element $\vec{x}$ can be written in the form $\vec{x} = c \cdot \vec{t}$ where $\vec{t}$ is a generator and c is an integer. This representation is not unique; in fact, there are $W(\vec{x})$ representations that involve different generators $\vec{t}$

Assuming that in (q1)

$\vec{y} = (y_1, y_2, \ldots, y_s) = d \cdot \vec{t}_j = d \cdot (t_{j_1}, t_{j_2}, \ldots, t_{j_s})$ and that $\vec{x} = (x_1, x_2, \ldots, x_s) = c \cdot \vec{t}_k = c \cdot (t_{k_1}, t_{k_2}, \ldots, t_{k_s})$, we rewrite (q1) as

$$\hat{A}(d\cdot\vec{t}_j)=\sum_{i=1}^{\Psi(n_1,n_2,\ldots,n_s)}\sum_{c=0}^{l(\vec{t}_i)-1}\omega_{n_1}^{d\cdot c\cdot t_{j_1}t_{a_1}}\omega_{n_2}^{d\cdot c\cdot t_{j_2}t_{a_2}}\ldots\omega_{n_s}^{d\cdot c\cdot t_{j_s}t_{a_s}}B(c\cdot\vec{t}_k),\qquad(q2)$$

where

$$B(\vec{u})=A(\vec{u})/\,W_{n_1,n_2,\cdots n_s}(\vec{u}).\qquad(q3)$$

The division in (q3) is dictated by non-uniqueness of the above representations.

Equation (q2) can be simplified by bringing the terms to the same base. Assume that $n=\mathrm{lcm}(n_1,n_2,\ldots,n_s)$ and that ω_n is the n-th primitive root of 1, $\omega_n=e^{2\pi i/n}$. We rewrite (q2) as

$$\begin{aligned}\hat{A}(d\cdot\vec{t}_j)&=\sum_{i=1}^{\Psi(n_1,n_2,\ldots,n_s)}\sum_{c=0}^{l(\vec{t}_i)-1}\omega^{(n/n_1)d\cdot c\cdot t_{j_1}t_{a_1}}\omega^{(n/n_2)d\cdot c\cdot t_{j_2}t_{a_2}}\ldots\omega^{(n/n_s)d\cdot c\cdot t_{j_s}t_{a_s}}B(c\cdot\vec{t}_k)\\&=\sum_{i=1}^{\Psi(n_1,n_2,\ldots,n_s)}\sum_{c=0}^{l(\vec{t}_i)-1}\omega^{cd\cdot(\sum_{m=1}^{s}(\frac{n}{n_m})t_{jm}t_{am})}B(c\cdot\vec{t}_k)\\&=\sum_{i=1}^{\Psi(n_1,n_2,\ldots,n_s)}\sum_{c=0}^{l(\vec{t}_i)-1}\omega^{cd\cdot(C(j,k))}B(c\cdot\vec{t}_k),\end{aligned}\qquad(q4)$$

where

$$C(j,k) = \sum_{m=1}^{s} (\frac{n}{n_m}) t_{jm} t_{km} \qquad (q5)$$

Notice that the coefficients $C(j,k)$ do not depend on the data and can be pre-computed and hard-wired in the Transform.

The computations given by equations (q4) are subdivided into two steps. One first computes the one-dimensional Discrete Fourier Transform on lines, forming the array $\hat{B}$:

$$\hat{B}(d) = \sum_{c=0}^{l(t_k)-1} \omega^{dc} B(c \cdot \vec{t}_k). \qquad (q6)$$

Notice that

$$\hat{B}(d \cdot C(j,k)) = \sum_{c=0}^{l(\vec{t}_k)-1} \omega^{d \cdot c C(j,k)} B(c \cdot \vec{t}_k), \qquad (q7)$$

Substituting (q7) into (q4), we obtain

$$\hat{A}(d \cdot \vec{t}_j) = \sum_{i=1}^{\Psi(n_1,n_2,\ldots,n_s)} \hat{B}(d \cdot C(j,k)). \qquad (q8)$$

The computation of the multi-dimensional Discrete Fourier Transform proceeds as follows:

- First, the data are normalized, using equations (q3).
- The one-dimensional Discrete Fourier Transform on lines is computed, forming the array $\hat{B}$ (equations (q6)).
- The multi-dimensional Transform is then computed by (q8).

Remark. A sample **FORTRAN** program is given in Appendix.

In addition to these three computational steps, one has to precompute line generators $\vec{t}_j$, weights W, and coefficients $C(j,k)$. Furthermore, the algorithm assumes that the data are ordered according to lines; thus an input and/or output permutation(s) may be required.

In the rest of this thesis we derive the formulas for the Ψ function and the weights W that are necessary for the pre-computational steps. To keep the notation simple, we will limit the discussion to the two-dimensional case.

1.4. Lines

In this section we derive some properties of the Ψ function and the formulas for its computation.

Definition. Let $\#(G)$ denote the order of the group G.

Let G_1 and G_2 be two rectangles; let H_1 be a subgroup of G_1 and H_2 be a subgroup of G_2. We will assume

$$gcd(\#(G_1), \#(G_2)) = 1. \qquad (q10)$$

Definition. Let π_1 and π_2 be the canonical projections from $G_1 \oplus G_2$ to G_1 and G_2, respectively.

We may write $H_1 = \pi_1(H_1 \oplus H_2)$ and $H_2 = \pi_2(H_1 \oplus H_2)$.

The following observations follow immediately from (q10):

Lemma 1.1. $\#(H_1 \oplus H_2) = \#(H_1) \cdot \#(H_2)$.

Lemma 1.2. $H_1 \oplus H_2$ is cyclic if and only if H_1 and H_2 are both cyclic.

Lemma 1.3. $H_1 \oplus H_2$ is a line in $G_1 \oplus G_2$ if and only if H_1 and H_2 are lines in G_1 and G_2 respectively.

Proof. Let $H_1 \oplus H_2$ be a line; assume that H_1 is not a line and is properly contained in a line λ_1. Then $\lambda_1 \oplus H_2$ is a cyclic subgroup that properly contains $H_1 \oplus H_2$; this contradicts $H_1 \oplus H_2$ being a line. Conversely, let H_1

and H_2 be lines; let λ be a line that contains $H_1 \oplus H_2$. Then $H_1 \subset \pi_1(\lambda)$ and $H_2 \subset \pi_2(\lambda)$. Since H_1 and H_2 are maximal, $H_1 = \pi_1(\lambda)$ and $H_2 = \pi_2(\lambda)$; by **Lemma 1.1** we have $H_1 \oplus H_2 = \lambda$. ∎

Remark. Most of the other proofs in this section are similar to the one above; we will therefore omit them.

Lemma 1.4. There exists a one-to-one correspondence between pairs of lines in G_1 and G_2 and lines in $G_1 \oplus G_2$.

Lemma 1.5. If $m = m_1 \cdot m_2$, $n = n_1 \cdot n_2$ and $gcd(m_1 n_1, m_2 n_2) = 1$, then

$$\Psi(m, n) = \Psi(m_1, n_1)\Psi(m_2, n_2). \tag{q11}$$

The last lemma implies that Ψ is a multiplicative function in each variable. Formally, we have shown

Theorem 1.6. Let $m = p_1^{a_1} \cdot p_2^{a_2} \ldots p_r^{a_r}$ and $n = p_1^{b_1} \cdot p_2^{b_2} \ldots p_r^{b_r}$, the p_i are distinct primes and $a_i, b_i \geq 0$. Then

$$\Psi(m, n) = \prod_{i=1}^{r} \Psi(p_i^{a_i}, p_i^{b_i}). \tag{q12}$$

The multiplicative property also holds for $\Psi_k(m, n)$.

Theorem 1.7.

$$\Psi_k(m, n) = \sum_{\substack{k_1, k_2, \ldots k_r \\ k_1 k_2 \ldots k_r = k}} \prod_{i=1}^{r} \Psi_{k_i}(p_i^{a_i}, p_i^{b_i}) \qquad (q13)$$

Example 4. $\Psi(12, 6) = \Psi(3, 3) \cdot \Psi(4, 2)$. Here $\Psi(4, 2) = 4$ (see Example 1). $\Psi(3, 3)$ is also 4; the lines in $I_{3,3}$ are

$$\lambda((1, 0)) = \{(0, 0), (1, 0), (2, 0)\}$$
$$\lambda((1, 1)) = \{(0, 0), (1, 1), (2, 2)\}$$
$$\lambda((0, 1)) = \{(0, 0), (0, 1), (0, 2)\}$$
$$\lambda((1, 2)) = \{(0, 0), (1, 2), (2, 1)\}$$

Thus $\Psi(12, 6) = 16$ by Theorem 1.6. Further, since $\Psi_2(4, 2) = 2$ and $\Psi_4(4, 2) = 2$, $\Psi_{12}(12, 6) = 6$ and $\Psi_6(12, 6) = 6$ by Theorem 1.7.

Here are the lines:

$\lambda((1,0))=\{(0,0),(1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(7,0),(8,0),(9,0),(10,0),(11,0)\}$

$\lambda((1,1))=\{(0,0),(1,1),(2,2),(3,3),(4,4),(5,5),(6,0),(7,1),(8,2),(9,3),(10,4),(11,5)\}$

$\lambda((1,2))=\{(0,0),(1,2),(2,4),(3,0),(4,2),(5,4),(6,0),(7,2),(8,4),(9,0),(10,2),(11,4)\}$

$\lambda((1,3))=\{(0,0),(1,3),(2,0),(3,3),(4,0),(5,3),(6,0),(7,3),(8,0),(9,3),(10,0),(11,3)\}$

$\lambda((1,4))=\{(0,0),(1,4),(2,2),(3,0),(4,4),(5,2),(6,0),(7,4),(8,2),(9,0),(10,4),(11,2)\}$

$\lambda((1,5))=\{(0,0),(1,5),(2,4),(3,3),(4,2),(5,1),(6,0),(7,5),(8,4),(9,3),(10,2),(11,1)\}$

$\lambda((9,1))=\{(0,0),(9,1),(6,2),(3,3),(0,4),(9,5),(6,0),(3,1),(0,2),(9,3),(6,4),(3,5)\}$

$\lambda((9,4))=\{(0,0),(9,4),(6,2),(3,0),(0,4),(9,2),(6,0),(3,4),(0,2),(9,0),(6,4),(3,2)\}$

$\lambda((10,5))=\{(0,0),(10,5),(8,4),(6,3),(4,2),(2,1)\}$

$\lambda((10,3))=\{(0,0),(10,3),(8,0),(6,3),(4,0),(2,3)\}$

$\lambda((10,1))=\{(0,0),(10,1),(8,2),(6,3),(4,4),(2,5)\}$

$\lambda((4,1))=\{(0,0),(4,1),(8,2),(0,3),(4,4),(8,5)\}$

$\lambda((4,3))=\{(0,0),(4,3),(8,0),(0,3),(4,0),(8,3)\}$

$\lambda((4,5))=\{(0,0),(4,5),(8,4),(0,3),(4,2),(8,1)\}$

$\lambda((6,1))=\{(0,0),(6,1),(0,2),(6,3),(0,4),(6,5)\}$

$\lambda((0,1))=\{(0,0),(0,1),(0,2),(0,3),(0,4),(0,5)\}$

The remainder of this section deals with the computation of $\Psi(p^a, p^b)$ for a fixed prime p and any $a, b \geq 0$. To compute $\Psi(p^a, p^b)$ we will choose a special set of line generators and then enumerate this set. We break the construction into a series of simple lemmas and omit most of the proofs.

Lemma 1.8. $\Psi(m, n) = \Psi(n, m)$ for all m and n.

Due to Lemma 1.8 we will assume $a \geq b$. The simplest case is $b = 0$,

when there exist only one line (of length p^a). The case $a = b$ is presented in [4]. To handle the general case, we will examine and list the generators t_j.

Definition. We write $j \mid k$ if j divides k; we write $j \nmid k$ otherwise.

Lemma 1.9. A cyclic subgroup $H \subset I_{p^a,p^b}$ is a line if and only if H is generated by (x, y) and either $p \nmid x$ or $p \nmid y$.

Proof. An element $\vec{t} = (p \cdot t_1, p \cdot t_2)$ does not generate a line since the cyclic subgroup generated by (t_1, t_2) properly contains the one generated by $\vec{t}$. Conversely, if $\vec{t} \in \lambda(\vec{u})$ and $\lambda(\vec{t})$ is not a line, then $\vec{t} = m \cdot \vec{u}$, where $\gcd(m, p) \neq 1$. ∎

Lemma 1.10. Any line is generated by either an element of the form $(1, u)$ or an element of the form $(v, 1)$ where $p \mid v$.

Proof. Consider a line generated by $\vec{t} = (t_1, t_2)$. By the previous lemma, either $\gcd(t_1, p) = 1$ or $\gcd(t_2, p) = 1$. In the former case, there exists an integer k, such that $k t_1 \equiv 1 (\mathrm{mod} p^a)$. Then $\lambda(\vec{t}) = \lambda(k \cdot \vec{t}) = \lambda((1, k \cdot t_2))$. Otherwise, $t_1 = p \cdot t_1'$ and there exists an integer k such that $k t_2 \equiv 1 (\mathrm{mod} p^b)$; therefore $(k \cdot t_1, 1) = (p(k \cdot t_1'), 1)$ generates $\lambda(\vec{t})$. ∎

Lemma 1.11. $\lambda(1,u) \neq \lambda(1,v)$ unless $u = v$.

Proof. Let $(1,v) \in \lambda(1,u)$. Then for some k $k \equiv 1(\mathrm{mod}p^a)$ and $kv \equiv u(\mathrm{mod}p^b)$. Because of the assumption $a \geq b$ this implies $u = v$. ∎

Corollary 1.12. There are p^b lines generated by elements of the form $(1,u)$. Each of them contains p^a elements.

It is more difficult to enumerate the line generators of the form $(v,1)$, since a similar lemma does not hold for them. For instance, in the $I_{16,2}$ example $(p = 2, a = 4, b = 1)$, elements $(2,1)$, $(6,1)$, $(10,1)$, and $(14,1)$ all generate the same line.

Lemma 1.13. Let $\gcd(u,p) = \gcd(v,p) = 1$ and $i \neq j$. Then $\lambda(p^i u,1) \neq \lambda(p^j v,1)$.

Proof. Assume $i > j$. Let $(p^i u,1) = c(p^j v,1)$ for some integer c. Then $p^{i-j}u \equiv cv(\mathrm{mod}p^{a-j})$, which contradicts the fact that c and v are relatively prime to p. ∎

Lemma 1.14. Let $i > a - b$. Then $\lambda(p^i u,1) \neq \lambda(p^i v,1)$, unless $u = v$.

Proof. Assume $(p^i u,1) = c(p^i v,1)$. Then $c \equiv 1(\mathrm{mod}p^b)$ and $u \equiv cv(\mathrm{mod}$

p^{a-i}). Since $a - i < b$, the latter equation implies that $u \equiv v \pmod{p^b}$. ∎

Corollary 1.15. There exist $\phi(p^{a-i}) = (p-1)p^{a-i-1}$ lines generated by elements of the form $(p^i u, 1)$, $i > a - b$. ($\phi(n)$ is the Euler's ϕ function; it computes the number of integers k, $1 \leq k \leq n$ that are relatively prime to n. (See [1], for instance).

Each of these lines contains p^b elements. The total number of lines accounted for in this case is $\sum_{i=a-b+1}^{a}(p-1)p^{a-i-1} = p^{b-1}$

It now remains to enumerate the lines generated by elements $(p^i u, 1)$, where $i \leq a - b$. The following lemma can be proven in the same fashion as **Lemma 1.14**:

Lemma 1.16. A line $\lambda(p^i t, 1)$, $i \leq a - b$, contains exactly p^{a-b-i} elements $(p^i u, 1)$.

Corollary 1.17. For any $i \leq a-b$, there are $\Phi(p^{a-i})/p^{a-b-i}$ lines generated by elements $(p^i t, 1)$. For all values of i, we count $\sum_{i\geq 1}^{a-b} \Phi(p^{a-i})/(p^{a-b-i}) = (a-b)\Phi(p^b)$ distinct lines. Each of these lines contains p^{a-i} elements.

Combining corollaries 1.12, 1.15, and 1.17, we obtain

Theorem 1.18.

$$\Psi(p^a, p^b) = p^b + p^{b-1} + (a-b)\phi(p^b) \qquad (q14)$$

where $\phi(p^b) = (p-1)p^{b-1}$.

Remark. This is consistent with the special case $\Psi(p^a, p^a) = (p+1)p^{a-1}$, treated in [4].

1.5. Mapping

From the results of the previous sections one may easily compute $\Psi(m, n)$ for any m and n. However, point-to-point mapping from G_1 and G_2 to $G_1 \oplus G_2$ is not as easy as the other way around. This section describe this mapping.

Definition. Let i be an element of a ring. i is called an idempotent of the ring if $i^2 = i$.

Lemma 1.19. Suppose that $m = m_1 \times m_2$ where $gcd(m_1, m_2) = 1$. Then elements $i_1, i_2 \in Z/m$ can be found such that

$$\begin{cases} i_1 \equiv 1 (\mathrm{mod}\, m_1) \\ i_1 \equiv 0 (\mathrm{mod}\, m_2) \end{cases} \qquad \begin{cases} i_2 \equiv 1 (\mathrm{mod}\, m_2) \\ i_2 \equiv 0 (\mathrm{mod}\, m_1) \end{cases} \qquad (q15)$$

Proof. The Chinese Remainder Theorem.

Furthermore, one can get

Corollary 1.20.

$$i_1^2 \equiv i_1 (mod\ m) \qquad i_2^2 \equiv i_2 (mod\ m)$$

Thus i_1 and i_2 are idempotents of Z/m.

Lemma 1.21. Any two distinct idempotents i_1 and i_2 in Z/m possess the following properties:

$$i_1 \cdot i_2 = 0, \tag{q16}$$

$$i_1 + i_2 = 1. \tag{q17}$$

Lemma 1.22. Suppose that i_1 and i_2 are two idempotents of $G_1 \oplus G_2$. Let u_1 and u_2 be elements of G_1 and G_2, respectively. Then an element u in $G_1 \oplus G_2$ which satisfies $\pi_1(u) = u_1$ and $\pi_2(u) = u_2$ is uniquely determined by

$$u = u_1 \cdot i_1 + u_2 \cdot i_2. \tag{q18}$$

Example 1.5. Let's consider the case of the rectangle $I_{72,6}$. By using the Chinese Remainder Theorem, we can view $I_{72,6}$ as the direct sum of

rectangles $I_{8,2}$ and $I_{9,3}$. Thus, one may define

$$\pi_1(x, y) = (x \bmod 8, y \bmod 2)$$

and

$$\pi_2(x, y) = (x \bmod 9, y \bmod 3),$$

where $(x, y) \in I_{72,6}$.

Assume that $i_1 = (\alpha_1, \alpha_2)$ and $i_2 = (\beta_1, \beta_2)$ are idempotents satisfying $\pi_1(i_2) = (0,0)$ and $\pi_2(i_1) = (0,0)$. One may compute α_1, α_2, β_1, and β_2 as follows:

$$\left\{\begin{array}{l} 1 \equiv \alpha_1 (mod\ 8) \\ 0 \equiv \alpha_1 (mod\ 9) \end{array}\right\} \quad \Longrightarrow \quad \alpha_1 = 9\,,$$

$$\left\{\begin{array}{l} 1 \equiv \alpha_2 (mod\ 2) \\ 0 \equiv \alpha_2 (mod\ 3) \end{array}\right\} \quad \Longrightarrow \quad \alpha_2 = 3\,,$$

$$\left\{\begin{array}{l} 0 \equiv \beta_1 (mod\ 8) \\ 1 \equiv \beta_1 (mod\ 9) \end{array}\right\} \quad \Longrightarrow \quad \beta_1 = 64\,,$$

$$\left\{\begin{array}{l} 0 \equiv \beta_2 (mod\ 2) \\ 1 \equiv \beta_2 mod 3 \end{array}\right\} \quad \Longrightarrow \quad \beta_2 = 4\,.$$

To illustrate our results, let us check a point mapping. Assume $(x_1, y_1) = (2, 1)$ and $(x_2, y_2) = (3, 1)$. Then

$$x = (2 \cdot 9 + 3 \cdot 64) \bmod 72 = 66,$$

$$y = (1 \cdot 3 + 1 \cdot 4) \bmod 6 = 1.$$

Then, let us check from the other direction:

$$x_1 = 66 \bmod 8 = 2$$

$$y_1 = 1 \bmod 2 = 1$$

$$x_2 = 66 \bmod 9 = 3$$

$$y_2 = 1 \bmod 3 = 1$$

Moreover, we find that $(66, 1)$ generates a line with length 12 in $I_{72,6}$, whereas $(2, 1)$ generates a line with length 4 in $I_{8,2}$ and $(3, 1)$ a line with length 3 in $I_{9,3}$. The generators in these three rectangles are shown in the following table. The subscript of each entry is the length of the line it generates.

$I_{8,2}$

	$(1,1)_8$	$(1,0)_8$	$(2,1)_4$	$(4,1)_2$	$(0,1)_2$
$(1,1)_9$	$(1,1)_{72}$	$(1,4)_{72}$	$(10,1)_{36}$	$(28,1)_{18}$	$(64,1)_{18}$
$(1,2)_9$	$(1,5)_{72}$	$(1,2)_{72}$	$(10,5)_{36}$	$(28,5)_{18}$	$(64,5)_{18}$
$(1,0)_9$	$(1,3)_{72}$	$(1,0)_{72}$	$(10,3)_{36}$	$(28,3)_{18}$	$(64,3)_{18}$
$(3,1)_3$	$(57,1)_{24}$	$(57,4)_{24}$	$(66,1)_{12}$	$(12,1)_6$	$(48,1)_6$
$(6,1)_3$	$(33,1)_{24}$	$(33,4)_{24}$	$(42,1)_{12}$	$(60,1)_6$	$(24,1)_6$
$(0,1)_3$	$(9,1)_{24}$	$(9,4)_{24}$	$(18,1)_{12}$	$(36,1)_6$	$(0,1)_6$

$I_{9,3}$

$I_{72,6}$

1.6. Weights

In this section, we derive the formulas for computing the weights $W(\vec{t})$.

Since the derivation of the formulas is a complicated, lengthy, and unexciting exercise in elementary number theory, we only sketch the proofs.

Similarly to the case with lines, one can show multiplicative properties of the weight function and therefore bring the discussion down to the prime case.

Lemma 1.23. If $m = m_1 \cdot m_2$, $n = n_1 \cdot n_2$, and $\gcd(m_1 n_1, m_2 n_2) = 1$, then

$$W_{m,n}(x, y) = W_{m_1,n_1}(x_1, y_1) \cdot W_{m_2,n_2}(x_2, y_2) \qquad (q19)$$

where $x_1 = x \bmod m_1$, $x_2 = x \bmod m_2$, $y_1 = y \bmod n_1$, and $y_2 = y \bmod n_2$.

Example 1.6. We can compute the weights in $I_{12,6}$ by looking at the weights in $I_{4,2}$ and $I_{3,3}$. The weights of all elements in these three rectangles are shown below:

$x_1 \backslash y_1$	0	1
0	4	1
1	1	1
2	2	1
3	1	1

$W_{4,2}$

$x_2 \backslash y_2$	0	1	2
0	4	1	1
1	1	1	1
2	1	1	1

$W_{3,3}$

$x\backslash y$	0	1	2	3	4	5
0	16	1	4	4	4	1
1	1	1	1	1	1	1
2	2	1	2	1	2	1
3	4	1	1	4	1	1
4	4	1	4	1	4	1
5	1	1	1	1	1	1
6	8	1	2	4	2	1
7	1	1	1	1	1	1
8	4	1	4	1	4	1
9	4	1	1	4	1	1
10	2	1	2	1	2	1
11	1	1	1	1	1	1

$W_{12,6}$

Every entry in $W_{12,6}$ is the product of two corresponding entries in $W_{4,2}$ and $W_{3,3}$. For instance, $W_{12,6}(0,0) = W_{4,2}(0,0) \cdot W_{3,3}(0,0) = 4 \times 4 = 16$, $W_{12,6}(6,0) = W_{4,2}(2,0) \cdot W_{3,3}(0,0) = 2 \times 4 = 8$, etc..

The previous lemma naturally generalizes into the following:

Lemma 1.24. If $m = p_1^{a_1} \cdot p_2^{a_2} \dots p_r^{a_r}$ and $n = p_1^{b_1} \cdot p_2^{b_2} \dots p_r^{b_r}$ where the p_i are different prime numbers and a_i, $b_i \geq 0$, then

$$W_{m,n}(x, y) = \prod_{i=1}^{r} W_{p_i^{a_i} p_i^{b_i}}(x_i, y_i),$$

where $x_i \equiv x (\mathrm{mod}\, p_i^{a_i})$, $y_i \equiv y (\mathrm{mod}\, p_i^{b_i})$.

It now remains to compute $W_{p^a,p^b}(x,y)$, where p is a prime. To shorten the notation, we will assume $W = W_{p^a,p^b}$.

Theorem 1.25. Let $(x,y) = (p^r c_1, p^s c_2)$ be an element in $I_{p^a p^b}$ where p is prime, $a \geq b$, and c_1, c_2 are not divisible by p. Then

(1) $W(0,0) = \Psi(p^a, p^b)$.

(2) $W(x,y) = p^s$, if $r > s > b$.

(3) $W(x,y) = p^b + (r-b)\Phi(p^b)$, if $r > s$ and $s = b$.

(4) $W(x,y) = p^r$, if $r \leq s$.

The derivation and proof of this result requires careful analysis of all cases and is shown in Appendix.

1.7. Advantages of the WRT Algorithm

In general, $s \cdot n^{s-1}$ one-dimensional DFT calls are required in computing an s-dimensional FFT of size n on each axis, while the WRT algorithm only needs Ψ one-dimensional DFT calls. When $s = 2$, the ratio $\Psi/s \cdot n^{s-1} \approx 0.75$ (see [4]). This ratio decreases as s increases for DFT of any size. Here are some examples:

	$n=4$	$n=5$	$n=7$	$n=15$
$s=2$	$6/8=0.78$	$6/10=0.60$	$8/14=0.57$	$24/30=0.80$
$s=3$	$28/48=0.53$	$31/75=0.41$	$57/167=0.34$	$403/675=0.59$
$s=4$	$120/256=0.46$	$131/500=0.26$	$400/1370=0.29$	$5240/13500=0.31$

In the table above, the numerators are values of Ψ, and the denominators are values of $s \cdot n^{s-1}$. The performance of r-WRT is somehow similar to the square case.

In the traditional sequential computer, multiplications were much slower than additions, so that algorithms were developed to minimize the number of multiplications at the expense of other operations (e.g. addition and indices swapping). Nowadays, newer hardware has a relatively faster multiplication capability, making such concerns less important. Moreover, parallelization has become the main concern in judging the performance of algorithms on mainframe computer. FFT algorithms have been found more efficient than WFTA-based algorithms on the new computer, for example the Cray-1 (see [14]), owing to its more parallel structure. The WRT algorithm is a highly parallelizable algorithm. It is inherently a one pass algorithm in any case (no matter how many dimensions the problem has), while FFT algorithms require s passes for the s-dimensional problem. Therefore, the WRT algorithm can

be expected to be very efficient on the new vector and parallel computers.

The advantages of WRT algorithm can be summarized as follows:

- It is applicable to DFTs of any size.
- In most cases, it requires fewer multiplications than an FFT algorithm.
- It has substantially less "structural" complexity than WFTA-based algorithm.
- It allows substantial reduction in computation in cases when the data satisfy a crystallographic symmetry (which is discussed in Chapter 2).
- It is expected to have a superior performance on vector and parallel computers.
- It has potential for real time problems.

Chapter 2.
Crystallographic-invariant DFT

2.1. Introduction.

This chapter introduces a new way of computing the crystallographic-invariant Discrete Fourier Transform. The new algorithm is based on the generic Weighted Redundancy Transform algorithm. In this chapter, the algorithm is tailored and optimized for the tasks arising in X-ray crystallography and the study of solid materials, where the sampled data are not arbitrary but are invariant under the actions of the group of symmetries of the crystal (the point group). The crystallographic WRT(c-WRT) holds a number of advantages over existing crystallographic transforms and is applicable to a large set of problems.

A few Discrete Fourier Transform algorithms have been introduced for handling symmetric data. The simplest (as well as the most common and most important case) is the case of the even Discrete Fourier Transform, where the data sample satisfies 180° rotational symmetry. The one-dimensional N-point

data sample array $A(0)$, $A(1)$, ..., $A(N-1)$ is said to be even if

$$\forall k \quad A(k) = A(N-k). \tag{r1}$$

This definition naturally generalizes for the multi-dimensional transform. The even Discrete Fourier Transform can be handled by direct application of the Fast Fourier Transform [9]; see [10] for an extensive discussion. The only other case which can be handled by direct application of the Fast Fourier Transform is the case where the data satisfy the 90° rotational symmetry. We say that a two-dimensional N-point data sample $A(k_1, k_2)$, $0 \leq k_1, k_2 < N$ is invariant under the 90° rotation if

$$\forall k_1, k_2 \quad A(k_1, k_2) = A(k_2, -k_1). \tag{r2}$$

This possibility was studied by Lynn Ten Eyck in [13].

Later algorithms designed for handling other invariances, such as 60°, 90°, 120° rotations (see, for example, [7]) were based on the one-dimensional Winograd Finite Fourier Transform Algorithm (WFTA) rather than on the Fast Fourier Transform . The c-WRT uses the Fast Fourier Transform and, therefore, can be implemented rather efficiently using existing optimized libraries.

Remark: This thesis only describes the algorithms; it does not deal with the questions of efficient implementation and hardware-dependent optimization of the transform. Also, to simplify our discussion, in this chapter, we consider only the square case.

2.2. Crystallographic Invariance.

In this section, we introduce notation for symmetry.

Definition. We say that a two-dimensional data array is R-symmetric if

$$A(R(x, y)) = A(x, y), \quad x, y \in Z$$

where the linear transformation

$$R(x, y) = (ax + by, cx + dy) = (R \cdot (x, y)^T)^T, \quad x, y \in Z,$$

assuming $R = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

We are not interested in studying R-symmetric functions in general. In practice, the eigenvalues of R must be conjugates, $e^{2\pi i\theta}$ and $e^{-2\pi i\theta}$, lying on the unit circle. Since R is an integer matrix, $2\cos(2\pi\theta) \in Z$.

This can be the case only if

$$\theta = 0, \frac{1}{6}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}$$

Therefore, R must have order 1, 2, 3, 4, or 6.

Definition. Let R_n denote an R having order n.

The case $R = R_1$ is nonsymmetric and therefore trivial. We only need to consider the cases of R_2, R_3, R_4, and R_6.

Lemma 2.1.

$$R_2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{r3}$$

$$R_3 = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix}, \tag{r4}$$

$$R_4 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \tag{r5}$$

$$R_6 = \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}, \tag{r6}$$

where R_2, R_3, R_4, and R_6 corresponds to π, $\pi/3$, $\pi/4$, and $\pi/6$ rotation symmetry, respectively.

$R_2 = -I$ maps a line back to the line itself and, therefore, will not be considered in this thesis.

Example 2.1. Consider the case of $I_{5,5}$ under R_3.

$$R_3 \cdot (1,2)^T = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = (-2(mod\ 5), -1(mod\ 5))^T = (3,4)^T,$$

$$R_3^2 \cdot (1,2)^T = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix} = (-4(mod\ 5), -1(mod\ 5))^T = (1,4)^T,$$

$$R_3^3 \cdot (1,2)^T = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 4 \end{pmatrix} = (-4(mod\ 5), -3(mod\ 5))^T = (1,2)^T,$$

where $A(1,2) = A(3,4) = A(1,4)$ under R_3.

2.3. Lines and Symmetries

WRT naturally incorporates crystallographic symmetries because of the following crucial observations:

Lemma 2.2. If R is any linear transformation of I, and λ is a line in I, then $R(\lambda)$ is a line.

The idea of the c-WRT is to "merge" lines λ, $R(\lambda)$, $R^2(\lambda), \ldots$ since they index the same data. More precisely, if Λ is the set of all lines, we would like to construct a set of representatives Λ_R such that

1. $\forall \lambda \in \Lambda\ \exists \lambda' \in \Lambda_R$ so that $\lambda = R^k(\lambda')$ for some $k \in Z$.
2. $\forall \lambda, \lambda' \in \Lambda_R$, $\forall k \in Z\ \lambda \neq R^k(\lambda')$.

We will further say that the lines λ and λ' are R-equivalent if $\exists k \in Z$, $\lambda' = R^k(\lambda)$.

The one-dimensional Discrete Fourier Transform will be computed only on the elements of a set of representatives Λ_R.

Example 2.2. Λ contains 6 lines in $I_{4,4}$. They are

$$\lambda_0 = \lambda((0,1)) = \{(0,0),\ (0,1),\ (0,2),\ (0,3)\}$$

$$\lambda_1 = \lambda((1,0)) = \{(0,0),\ (1,0),\ (2,0),\ (3,0)\}$$

$$\lambda_2 = \lambda((1,1)) = \{(0,0),\ (1,1),\ (2,2),\ (3,3)\}$$

$$\lambda_3 = \lambda((1,2)) = \{(0,0),\ (1,2),\ (2,0),\ (3,2)\}$$

$$\lambda_4 = \lambda((2,1)) = \{(0,0),\ (2,1),\ (0,2),\ (2,3)\}$$

$$\lambda_5 = \lambda((1,3)) = \{(0,0),\ (1,3),\ (2,2),\ (3,1)\}$$

Let $R = R_3$. A set of representatives Λ_R can be chosen to be $\{\lambda_0, \lambda_3\}$, since $\lambda_2 = R_3(\lambda_0)$, $\lambda_1 = R_3^2(\lambda_0)$, $\lambda_4 = R_3(\lambda_3)$, and $\lambda_5 = R_3^2(\lambda_3)$. In this example there are two families of R_3-equivalent lines: $\{\lambda_0, \lambda_1, \lambda_2\}$ and $\{\lambda_3, \lambda_4, \lambda_5\}$.

In general, families of R-equivalent lines may contain different numbers of lines. The task of this section is to identify these families.

Lemma 2.2. If $R = R_3$, each R-equivalent family contains either 1 or 3 lines. If $R = R_4$, each R-equivalent family contains either 1 or 2 lines. If $R = R_6$, each R-equivalent family contains either 1 or 3 lines.

Proof. The number of lines in each family divides the order of the rotation matrix. The result now follows from the observation that $R_4^2 = R_6^3 = -I$; multiplication by $-I$ leaves lines invariant.∎

Our next task is to identify the cases which produce 1-line R-equivalent families. As will be seen later, these cases require special handling.

Definition. We say that λ is an R-invariant line if $\lambda = R(\lambda)$.

Remark. The problem of finding 1-line R-equivalent families is a discrete analog of the problem of finding the eigenvectors of a matrix. The difference is that the computations carried over Z_N, which is generally not a field.

We will detail the discussion only for the case $R = R_3$. The other two cases can be handled in a similar fashion.

Let $\vec{k} = (x, y) \in I$, $\lambda = \lambda(\vec{k})$ be an R-invariant line. Then $R(\vec{k}) = c \cdot \vec{k}$ for some $c \in Z$. Substituting R_3 for R, one sees that $-y \equiv c \cdot x \pmod{N}$ and $x - y \equiv c \cdot y \pmod{N}$. Since $\vec{k}$ is primitive, $\gcd(x, y, N) = 1$. Because of $-y \equiv c \cdot x \pmod{N}$, $\gcd(x, N) = 1$ and $\gcd(y, N) = 1$. We can conclude that

$$c^2 + c + 1 \equiv 0 \pmod{N} \qquad (r7)$$

1-line R_3-equivalent families occur only when the characteristic polynomial ($r7$) of R_3 has solutions, i.e. R_3 has eigenvalues over Z_N.

Lemma 2.3. 1-line R_3-equivalent families occur if and only if the equation $c^2 + c + 1 \equiv 0 \pmod{N}$ has solutions, i.e. when $N = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, where p_i are different prime numbers such that $p_i = 3$ or $p_i \bmod 3 = 1$. In this case there are 2^k solutions.

Similar reasoning for the $R = R_4$ and $R = R_6$ cases leads to the following lemmas:

Lemma 2.4. 1-line R_6-equivalent families occur if and only if the equation $c^2 - c + 1 \equiv 0 \pmod{N}$ has solutions, i.e. when $N = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, where p_i are different prime numbers such that $p_i = 3$ or $p_i \bmod 3 = 1$. In this case there are 2^k solutions.

Lemma 2.5. 1-line R_4-equivalent families occur if and only if the equation $c^2 + 1 \equiv 0 \pmod{N}$ has solutions, i.e. when

a. $N = 2$ (solution $c = 1$), or

b. $N = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, or $N = 2 \cdot p_1 \cdot p_2 \cdot \ldots \cdot p_k$, where p_i is prime; $p_i \bmod 4 = 1$.

In this case, there are 2^k solutions.

We will assume for now that N is such that all R-equivalent families have the same number of elements. Special cases listed in lemma 3, 4 and 5 require only slight "patches" to the algorithm but would make the notation look more cumbersome. These cases will be discussed later in the thesis. It should be noted, however, that special cases do not include traditionally used data sample sizes, such as powers of 2. For instance, for $R = R_3$ and $R = R_6$, 1-line families do not occur if N is even. For $R = R_4$, 1-line families do not occur if N is divisible by 4.

Example 2.3. The following data sample sizes will require special handling for $N \leq 100$ and $R = R_3$ or $R = R_6$: 1, 3, 7, 13, 19, 21, 31, 37, 39, 43, 49, 57, 61, 67, 73, 79, 91, 93, and 97. For $R = R_4$, special handling is required for 1, 2, 5, 10, 13, 17, 25, 26, 29, 34, 37, 41, 50, 53, 58, 61, 65, 73, 74, 82, 85, 89, and 97.

The central idea of the WRT is to carry out the computations not in standard orthogonal coordinates, but to index the data points by the lines and the position of the element in the lines. More precisely, any element $\vec{u} \in I$ can be (not uniquely) presented as $c \cdot \vec{t_j}$, where $\vec{t_j}$ generates line λ_j. In the absence of symmetry, the computation proceeds on $\Psi(N)$ lines. By symmetry,

the lines $\lambda(\vec{t}_j)$, $\lambda(R(\vec{t}_j))$, $\lambda(R^2(\vec{t}_j))$,... index the same data. It will therefore be possible to perform the computation only once for each R-equivalent family.

Definition. Let $\Psi_{(R)}(N)$ denote the number of R-equivalent families.

Remark. If N does not fall into one of the special cases listed in Lemmas 2.3, 2.4 and 2.5, $\Psi_{(R_3)}(N) = \Psi_{(R)}(N)/3$, $\Psi_{(R_6)}(N) = \Psi_{(R)}(N)/3$, and $\Psi_{(R_4)}(N) = \Psi_{(R)}(N)/2$. The formula is somewhat more complex in the special cases.

2.4. c-WRT for 120° Rotation

In this section we assume $R = R_3$.

The computation of the 2-dimensional Discrete Fourier Transform on an $N \times N$ point data array A is given by the following equations:

$$\hat{A}(\vec{v}) = \sum_{\vec{u} \in I(n)} \omega^{<\vec{v},\vec{u}>} \cdot A(\vec{u}), \tag{r8}$$

where $\omega = e^{2\pi i/N}$. Each index tuple $\vec{u}$ lies in $W(\vec{u})$ lines. Assuming that $\vec{t}_j$, $j = 1,2,\ldots,\Psi(N)$, and $\vec{s}_j$, $j = 1,2,\ldots,\Psi(N)$ are two (not necessarily different) systems of line generators, we can rewrite the above equations as

$$\hat{A}(d \cdot \vec{s}_j) = \sum_{i=1}^{\Psi(n)} \sum_{c=0}^{n-1} \omega^{<d \cdot \vec{s}_j, c \vec{t}_i>} \cdot B(c \cdot \vec{t}_i), \qquad (r9)$$

where

$$B(\vec{u}) = A(\vec{u}) / W(\vec{u}). \qquad (r10)$$

This normalizing division is required since $\vec{u}$ occurs in $W(\vec{u})$ lines. Let

$$B'_j(d \cdot \vec{t}_i) = \sum_{c=0}^{n-1} \omega^{<d \cdot \vec{s}_j, c \vec{t}_i>} \cdot B(c \cdot \vec{t}_i), \qquad (r11)$$

Then

$$\hat{A}(d \cdot \vec{s}_j) = \sum_{i=1}^{\Psi(n)} B'_j(d \cdot \vec{t}_i) \qquad (r12).$$

Let $\mu = \Psi_{(R_3)}$ and $\nu = \Psi'_{(R_3)}$. The system of line generators $\vec{t}_i$ can be chosen in such a way that the first 3μ generators $\vec{t}_i$ belong to the 3-element R_3-equivalent families;

$$\vec{t}_{1+\mu} = R_3 \cdot \vec{t}_1; \quad \vec{t}_{1+2\mu} = R_3 \cdot \vec{t}_1$$

$$\vec{t}_{2+\mu} = R_3 \cdot \vec{t}_2; \quad \vec{t}_{2+2\mu} = R_3 \cdot \vec{t}_2$$

$$\cdots\cdots\cdots \qquad (r13)$$

$$\vec{t}_{\mu-1+\mu} = R_3 \cdot \vec{t}_{\mu-1}; \quad \vec{t}_{\mu-1+2\mu} = R_3 \cdot \vec{t}_{\mu-1}$$

$$\vec{t}_{2\mu} = R_3 \cdot \vec{t}_{\mu}; \quad \vec{t}_{3\mu} = R_3 \cdot \vec{t}_{\mu}$$

The remaining ν generators $\vec{t}_i$ (in most cases $\nu = 0$) form 1-element R_3-equivalent families, i.e. $\vec{t}_i = R_3 \cdot \vec{t}_i$.

We will further assume that the generators $\vec{s}_j$ are similarly ordered with respect to the transposed R_3, $\hat{R}_3^t$:

$$\vec{s}_{1+\mu} = \hat{R}_3 \cdot \vec{s}_1; \quad \vec{s}_{1+2\mu} = \hat{R}_3 \cdot \vec{s}_1$$

$$\vec{s}_{2+\mu} = \hat{R}_3 \cdot \vec{s}_2; \quad \vec{s}_{2+2\mu} = \hat{R}_3 \cdot \vec{s}_2$$

$$\cdots\cdots\cdots \qquad (r14)$$

$$\vec{s}_{\mu-1+\mu} = \hat{R}_3 \cdot \vec{s}_{\mu-1}; \quad \vec{s}_{\mu-1+2\mu} = \hat{R}_3 \cdot \vec{s}_{\mu-1}$$

$$\vec{s}_{2\mu} = \hat{R}_3 \cdot \vec{s}_{\mu}; \quad \vec{s}_{3\mu} = \hat{R}_3 \cdot \vec{s}_{\mu}$$

We can rewrite equation (r12) as

$$\hat{A}(d \cdot \vec{s}_j) = \sum_{i=1}^{\nu} (B'_j(d \cdot \vec{t}_i) + B'_j(d \cdot R_3 \vec{t}_i) + B'_j(d \cdot R_3^2 \vec{t}_i)) + \sum_{i=3\mu+1}^{3\mu+\nu} B'_j(d \cdot \vec{t}_i). \qquad (r15)$$

The values $B'_j(d \cdot \vec{t}_i)$ are computed by the one-dimensional DFT on lines. Since

$$\hat{B}(d \cdot \vec{t}_i) = \sum_{c=0}^{n-1} \omega^{cd} B(c \cdot \vec{t}_i), \tag{r16}$$

we rewrite equation (r10) as

$$B'_j(d \cdot \vec{t}_i) = \hat{B}(< \vec{t}_i, \vec{t}_j > d \cdot \vec{t}_i). \tag{r17}$$

Let $\kappa_{ij} =< \vec{t}_i, \vec{t}_j >$. Substituting (r17) into (r15) we obtain

$$\begin{aligned} \hat{A}(d \cdot \vec{s}_j) = \sum_{i=1}^{\nu} (\hat{B}(< \vec{t}_i, \vec{t}_j > d \cdot \vec{t}_i) + \hat{B}(< \vec{t}_i, R_3 \vec{t}_j > d \cdot \vec{t}_i) \\ + \hat{B}(< \vec{t}_i, R_3^2 \vec{t}_j > d \cdot \vec{t}_i)) + \sum_{i=3\mu+1}^{3\mu+\nu} \hat{B}(< \vec{t}_i, \vec{t}_j > d \cdot \vec{t}_i). \end{aligned} \tag{r18}$$

Notice that the coefficients $< \vec{t}_i, \vec{t}_j >$ can be precomputed. In order to simplify the notation in the last equation, we define

$$\kappa_{0ij} =< \vec{t}_i, \vec{t}_j >, \ \kappa_{1ij} =< \vec{t}_i, R_3 \vec{t}_j >, \text{and } \kappa_{2ij} =< \vec{t}_i, R_3^2 \vec{t}_j >, \tag{r19}$$

and rewrite (r18) as

$$\begin{aligned} \hat{A}(d \cdot \vec{s}_j) = \sum_{i=1}^{\nu} (\hat{B}(\kappa_{0ij} d \cdot \vec{t}_i) + \hat{B}(\kappa_{1ij} d \cdot \vec{t}_i) + \hat{B}(\kappa_{2ij} d \cdot \vec{t}_i)) \\ + \sum_{i=3\mu+1}^{3\mu+\nu} \hat{B}(\kappa_{0ij} d \cdot \vec{t}_i). \end{aligned} \tag{r20}$$

These equations describe the line algorithm for the 120°-degree rotation invariant data. Observe that the computations have to be carried out only for $j = 1,2,3,\ldots,\mu-1,\mu,3\mu+1,3\mu+2,\ldots,3\mu+\nu$, since for $j = 1,2,3,\ldots,\mu-1,\mu$,

$$\hat{A}(d\cdot\vec{s}_{j+\mu}) = \hat{A}(d\cdot\vec{s}_{j+2\mu}) = \hat{A}(d\cdot\vec{s}_j) \qquad (r21)$$

Similarly, the redundancy in the input data allows one to alter (r10) to

$$B(d\cdot\vec{t}_i) = A(d\cdot\vec{t}_i)/\,W(d\cdot\vec{t}_i), \qquad (r22)$$

where the computation should be carried out only for $j = 1,2,3,\ldots,\mu-1,\mu,3\mu+1,3\mu+2,\ldots,3\mu+\nu$.

In addition to saving the computations, the data redundancy effectively reduces the storage requirements by a factor of approximately three.

The computations should be organized as follows:

1. Compute the line generators t_i and s_j, the tables of weights W, and the inner products κ. It is best to pre-compute these values and "hard-wire" them into the program.
2. Permute the input data into lines (see below).

3. Normalize the data, following (r22). This step can be combined with the previous one.
4. Compute the one-dimensional Discrete Fourier Transform on $\mu+\nu$ lines, following (r16).
5. Add up the results, following (r20).
6. Permute the output data.

The input permutation step may be done as follows. Assuming that the line generators $\vec{t}_i = (t_{i_1}, t_{i_2})$ are stored in a two-dimensional $\Psi(n) \times 2$ array INPUTL, we compute

```
for i := 1 to μ + ν do
  for c := 0 to n − 1 do
    ALINES[i,c] := A [INPUTL[i,1]*c mod n, INPUTL[i,2]*c mod n];
```

2.5. Advantages of the c-WRT Algorithm

Some of the advantages of the c-WRT over other crystallographic Discrete Fourier Transforms are summarized here:

- The c-WRT is applicable to *all* point groups. While this thesis discusses

only two- and three-dimensional rotations and with or without translations, a similar approach can be used with other point groups.

- The c-WRT is applicable to *any* data sample sizes that are consistent with the structure of the point group. WFRT-based algorithms require special handling for each sample size and usually are efficient only for prime or prime power sample sizes. c-WRT works in the same fashion on any data.
- In particular, the c-WRT can be used with *very large* data samples, involving millions of the data points, as happens in X-ray studies of viral and protein material.
- The c-WRT's reduction in the computational complexity is proportional to the size of the point group; for very large problems it may be even more important that a similar reduction also occurs in the data flow.
- The c-WRT is a relatively *simple* algorithm. As such it can be speedily implemented on any current computer. The use of the Fast Fourier Transform allows the usage of standard libraries and, therefore, cuts down the implementation effort.
- Like the WRT, c-WRT can be implemented on modern vector and

parallel computers.

- The c-WRT can be applied in a few other areas, including holography.

Appendix

A.1. A Sample Program

Here we present fragments of a FORTRAN program that computes the Discrete Fourier Transform of a 9 by 3 data array A by using the r-WRT algorithm.

The data is indexed by 6 lines ($\Psi(9,3) = 6$):

$$\lambda(t_0 = (1,0)) = \{(0,0),(1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(7,0),(8,0)\}$$

$$\lambda(t_1 = (1,1)) = \{(0,0),(1,1),(2,2),(3,0),(4,1),(5,2),(6,0),(7,1),(8,2)\}$$

$$\lambda(t_2 = (1,2)) = \{(0,0),(1,2),(2,1),(3,0),(4,2),(5,1),(6,0),(7,2),(8,1)\}$$

$$\lambda(t_3 = (0,1)) = \{(0,0),(0,1),(0,2)\}$$

$$\lambda(t_4 = (3,1)) = \{(0,0),(3,1),(6,2)\}$$

$$\lambda(t_4 = (6,1)) = \{(0,0),(6,1),(3,2)\}$$

```
      SUBROUTINE FFT9(X)
      COMPLEX X(0:8)
* This subroutine computes 9-point DFT. X is both the input
* and the output array.
      ..............
      RETURN
```

```
      END
*
      SUBROUTINE FFTDUP9(X)
      COMPLEX*16 X(0:8),A
      REAL*8 Y1(2),Y2(2),Y3(2),Y4(2),Y7(2)
      REAL*8 B,C,Z1,Z2,Z3,Z4
      EQUIVALENCE (A,Y1),(X(1),Y2),(X(2),Y3)
      EQUIVALENCE (X(3),Y4),(X(6),Y7)
      DATA B/1.73205080756890/
      A=X(0)
      C=Y1(1)+Y1(1)+Y1(1)
      X(0)=(X(0)+X(1)+X(2))*3
      Z1=-(Y2(1)+Y3(1))
      Z2=(Y3(2)-Y2(2))*B
      Z3=-(Y2(2)+Y3(2))
      Z4=(Y2(1)-Y3(1))*B
      Y4(1)=(Z1+Z2)*1.5+C
      Y4(2)=(Z3+Z4)*1.5+Y1(2)+Y1(2)+Y1(2)
      Y7(1)=(Z1-Z2)*1.5+C
      Y7(2)=Y4(2)
      X(1)=(0.,0.)
      X(4)=(0.,0.)
      X(7)=(0.,0.)
      X(2)=(0.,0.)
      X(5)=(0.,0.)
      X(8)=(0.,0.)
      RETURN
      END
*
      SUBROUTINE WRT93(A,AHAT)
      COMPLEX*16 A(0:8,0:2),AHAT(0:8,0:2)
      COMPLEX*16 L(0:8,0:2),LHAT(0:8,0:2)
      INTEGER C(0:5,0:5)
*
```

```
* Array C contains coefficients
*
      DATA C/
     0  3,3,3,0,0,0,
     1  3,4,5,1,1,7,
     2  3,5,7,2,2,2,
     3  0,1,2,1,1,1,
     4  0,1,2,1,1,1,
     5  0,7,2,1,1,1/
* Divided by weights (when needed)
      A(0,0)=A(0,0)/6.
      A(3,0)=A(3,0)/3.
      A(6,0)=A(6,0)/3.
      A(0,1)=A(0,1)/3.
      A(0,2)=A(0,2)/3.
      A(3,1)=A(3,1)/3.
      A(3,2)=A(3,2)/3.
      A(6,1)=A(6,1)/3.
      A(6,2)=A(6,2)/3.
      * Assign into lines
      L(0,0)=A(0,0)
      L(1,0)=A(1,0)
      L(2,0)=A(2,0)
      L(3,0)=A(3,0)
      L(4,0)=A(4,0)
      L(5,0)=A(5,0)
      L(6,0)=A(6,0)
      L(7,0)=A(7,0)
      L(8,0)=A(8,0)
      ............
* Calling one-dimensional FFT subroutines.
      CALL FFT9(L(0,0))
      CALL FFT9(L(0,1))
      CALL FFT9(L(0,2))
      CALL FFTDUP9(L(0,3))
```

```
      CALL FFTDUP9(L(0,4))
      CALL FFTDUP9(L(0,5))
* Initialize the output
      DO 1 I=0,8
      DO 1 J=0,5
    1  LHAT(I,J)=(0.,0.)
* Additions and permutation.
      DO 2 LINEIN=0,5
      DO 2 LINEOT=0,5
      DO 2 I=0,8
    2  LHAT(I,LINEOT)=LHAT(I,LINEOT)+
    &    L(MOD(I*C(LINEIN,LINEOT),4),LINEIN)
* Assignment from lines
      AHAT(0,0)=LHAT(0,0)
      AHAT(1,0)=LHAT(1,0)
      AHAT(2,0)=LHAT(2,0)
      AHAT(3,0)=LHAT(3,0)
      AHAT(4,0)=LHAT(4,0)
      AHAT(5,0)=LHAT(5,0)
      AHAT(6,0)=LHAT(6,0)
      AHAT(7,0)=LHAT(7,0)
      AHAT(8,0)=LHAT(8,0)
      .................
      RETURN
      END
```

A.2. Derivation of the Weight Function

Since $(0,0)$ belongs to all the lines, obviously its weight is equal to the value of the Ψ function. Based on a result from §1.4, it is reasonable to consider all points (x, y), excluding $(0,0)$, in four categories:

I. $x = 0,\ y \neq 0.$

II. $x \neq 0,\ y = 0.$

III. $x \neq 0,\ y \neq 0,$ but the p-power in $x >$ the p-power in $y.$

IV. $x \neq 0,\ y \neq 0,$ but the p-power in $x \leq$ the p-power in $y.$

To further simplify our discussion we observe that

Lemma A.1. In $I_{p^r p^s}$,

(1) $W(p^r c_1, 0) = W(p^r, 0)$ and

(2) $W(0, p^s c_2) = W(0, p^s),$

where c_1 and c_2 are prime to $p.$

Proof: (1) One can easily build a one-to-one correspondence between the set of all lines that contain the element $(p^r, 0)$ and the set of all lines containing $(p^r c_1, 0)$ by using the mapping $f : u \rightarrow cu$, where $u = (p^r, 0)$. ∎

(2) can be proved similarly.

From the discussion in §1.4, we know that line generators must be either $(1, c_1)$ or $(p^i c_2, 1)$, where $i > 0.$

Case I. $(x, y) = (0, p^s c)$ where c is prime to p.

For simplicity, one need consider only $(x, y) = (0, p^s)$.

These elements can only be generated by $(p^i c_2, 1)$-type generators. Explicitly, we know any $(p^i c_2, 0)$ will do as long as $i \geq a - s$. Since the number of the generators of this type is known, the answer to this case will be

$$\frac{p^a}{p^{a-s}} = p^s.$$

Case II. $(x, y) = (p^r c, 0)$ where c is not divisible by p.

Again one may just consider the case $(x, y) = (p^r, 0)$. Here, we further divide this case, according the value of r, into two subcases.

(i) If $r \leq b$, then $(p^r, 0)$ can only be generated by $(1, c_1)$ type generators. Thus we have

$$p^r c_1 \equiv 0 (\mathrm{mod}\, p^b).$$

Then

$$c_1 \equiv 0 (\mathrm{mod}\, p^{b-r}).$$

There are only p^r c_1 in Z_{p^b} that satisfy the equation above. Therefore the answer is p^r.

(ii) If $r > b$, then $(p^r, 0)$ can be generated by both $(1, c_1)$- and $(p^i c_2, 1)$-type generators. Actually any $(1, c_1)$ and any $(p^i c_2, 1)$ with $i \leq r - b$ generates $(p^r, 0)$. Therefore the solution to this subcase is $p^b + (r - b)\Phi(p^b)$. (Recall the results in §1.4.)

Case III. $(x, y) = (p^r c_1, p^s c_2)$ with $r > s$, where $x, y \neq 0$.

For the sake of simplicity, one may as well assume $(x, y) = (p^r d, p^s)$ where d is a constant and is prime to p. (This can be done by multiplying $(p^r c_1, p^s c_2)$ by a constant.)

In this case we need the concept of equivalence classes.

Definition. Let EC-(η) denote the equivalence class under the operation η.

It can be easily shown that only $(p^i c, 1)$-type generators with $i = r - s$ can possibly generate $(p^r d, p^s)$. So now we will concentrate on the generators of the type $(p^i c, 1)$ with i fixed as $i = r - s$. However, different $(p^i c, 1)$ may generate the same line.

Suppose that $(p^i c, 1)$ and $(p^i c', 1)$ generate the same line.

Then

$$p^i c \equiv p^i c'(1 + kp^b)(\mathrm{mod}\, p^a) \quad \Longleftrightarrow$$

$$c \equiv c'(1 + kp^b)(\mathrm{mod}\, p^{a-i}) \quad \Longleftrightarrow$$

$$c \equiv c'(\mathrm{mod}\, p^b) \qquad \text{(with respect to } (\mathrm{mod}\, p^{a-i})$$

Thus $(p^i c, 1)$ and $(p^i c', 1)$ will generate the same line *if and only if* when we regard c and c' as elements of $Z_{p^{a-i}}$, c and c' must be in the same EC-$(\mathrm{mod}\, p^b)$.

Note: When $p^b > p^{a-i}$, each EC-(mod p^b) (of the set $Z_{p^{a-i}}$) actually contains only one element.

Now back to Case III, noting that $i = r - s$.

Assume that $(p^r d, p^s)$ is generated by $(p^i c, 1)$.

Then, we have

$$(p^r c, p^s) = (p^s + kp^b) \cdot (p^i c, 1) \quad \Longleftrightarrow$$

$$p^r d \equiv p^i c(p^s + kp^b)(\mathrm{mod}\, p^a) \quad \Longleftrightarrow$$

$$p^r d \equiv p^r c(1 + kp^{b-s})(\mathrm{mod}\, p^a) \quad \Longleftrightarrow$$

$$d \equiv c(1 + kp^{b-s})(\mathrm{mod}\, p^{a-r}) \quad \Longleftrightarrow$$

$$d \equiv c(\mathrm{mod}\, p^{b-s}) \quad \text{(with respect to mod}\, p^{a-r}) \Longleftrightarrow$$

the EC-(mod p^{b-s}) of c intersects the EC-(mod p^{a-r}) of d.

However, we know that any EC-(mod p^α) and EC-(mod p^β) are either $\subset$ or $\supset$ or disjoint. So the necessary and sufficient condition for $(p^r d, p^s)$ to be generated by $(p^i c, 1)$ will be:

EC-(mod p^{b-s}) of c either contains or is contained in the EC-(mod p^{a-r}) of d.

Recall that each EC-(mod p^b) represents a line. Thus our problem is equivalent to the following question:

Question: In $Z_{p^{a-i}}$, how many EC's-(mod p^b) of c, when imbedded as EC-(mod p^{b-s}), will contain or are contained in the particular EC-(mod p^{a-r}) of element d ?

Let

N_1 be the number of elements (of $Z_{p^{a-i}}$) in each EC-(mod p^b),

N_2 be the number of elements (of $Z_{p^{a-i}}$) in each EC -(mod p^{b-s}), and

N_3 be the number of elements (of $Z_{p^{a-i}}$) in each EC -(mod p^{a-r}).

It is easy to show that

$$N_1 = max(\frac{p^{a-i}}{p^b}, 1),$$
$$N_2 = max(\frac{p^{a-i}}{p^{b-s}}, 1),$$
$$N_3 = max(\frac{p^{a-i}}{p^{a-r}}, 1) = p^{r-i} = p^s$$

If the EC-(mod p^{a-r}) of d contains the EC-(mod p^{b-s}) of c, the answer to the above question is

$$\frac{N_3}{N_2} \cdot \frac{N_2}{N_1} = \frac{N_3}{N_1} \qquad (A1)$$

On the other hand, if the EC-(mod p^{a-r}) of d is contained in the EC-(mod p^{b-s}) of c, the answer to the above question is

$$\frac{N_2}{N_1}. \qquad (A2)$$

Therefore, combining equations ($A1$) and ($A2$), the answer to Case III can be put in a single formula:

$$[max(\frac{N_3}{N_2}, 1)] \cdot \frac{N_2}{N_1}$$

$$= max[\frac{p^s}{max(p^{a-i-b+s}, 1)}, 1] \cdot \frac{max(p^{a-i-b+s}, 1)}{max(p^{a-i-b}, 1)}$$

$$= max[\frac{p^s}{max(p^{a-b-r+2s}, 1)}, 1] \cdot \frac{max(p^{a-b-r+2s}, 1)}{max(p^{a-b-r+s}, 1)}. \qquad (A3)$$

Given different values a, b, r, and s, we would expect to get different answers. However, all results lead to a simple answer p^s.

Remark: Since the answer to this case is independent of d,

$$W(p^r c_1, p^s c_2) = W(p^r, p^s)$$

when $r > s$.

Case IV. $(x, y) = (p^r c_1, p^s c_2)$ with $r \leq s$, where $x, y \neq 0$.

Here one may assume that $(x, y) = (p^r, p^s d)$. (This can be achieved by multiplying $(p^r c_1, p^s c_2)$ by a constant.) These elements can only be generated by $(1, c)$-type generators.

So we have

$$p^s d \equiv p^r c (\mathrm{mod}\, p^b) \quad \Longleftrightarrow$$

$$c \equiv p^{a-r} d (\mathrm{mod}\, p^{b-r}).$$

The problem becomes counting the number of the EC's-(mod p^{b-r}) in Z_{p^b}. The answer to this case is p^r.

Remark: Once again since the answer to this case is independent of the variable d, and combining with the remark of the previous case, one may conclude that $W(p^r c_1, p^s c_2) = W(p^r, p^s)$ for all cases.

Lemma A.2. In $I_{p^a p^b}$,

$$W(p^r c_1, p^s c_2) = W(p^r, p^s) \qquad (A4)$$

. where c_1 and c_2 are not divisible by p.

Finally, let us conclude this section by following theorem:

Theorem A.3. Let $(x, y) = (p^r c_1, p^s c_2)$ be an element in I_{p^a, p^b} where p is prime and $a \geq b$. Then

(1) $W(x, y) = p^s$ if $x = 0$ and $y \neq 0$,

(2.1) $W(x, y) = p^r$ if $y = 0$ and $x \neq 0$ and $r \leq b$;

(2.2) $W(x, y) = p^b + (r - b)\Phi(p^b)$ if $y = 0$ and $x \neq 0$ and $r > b$,

(3) $W(x, y) = p^s = max[\frac{p^r}{max(p^{a-b-r+2s}, 1)}, 1] \cdot \frac{max(p^{a-b-r+2s}, 1)}{max(p^{a-b-r+s}, 1)}$ if $x \cdot y \neq 0$ and $r > s$, and

(4) $W(x, y) = p^r$ if $x \cdot y \neq 0$ and $r \leq s$.

By analyzing all cases, one may conclude Theorem 1.25.

References

[1] K. H. Rosen, "Elementary Number Theory and its applications," Addison-Wesley, (1985): 167.

[2] L. Auslander, E. Feig, and S. Winograd, "New Algorithms for the Multi-Dimensional Discrete Fourier Transform, "*IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-31, (1983): 388-403.

[3] M. Vulis, "Ring Structures and the Discrete Fourier Transform," *Advances of Applied Math.*, **6**, (1985): 350-372.

[4] M. Vulis, "The Weighted Redundancy Transform," *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-37, (1989): 1687-1692.

[5] M. Vulis, "Ring Structures and the Discrete Fourier Transform," submitted to *IEEE Trans. Acoust., Speech, Signal Processing.*

[6] D.-R. Tsai and M. Vulis, "Computing Discrete Fourier Transform on a Rectangular Data Array," *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-38, (1990): 271-276.

[7] R. Tolimieri, M. Vulis, "Orbital Algorithm: A Symmetry Invariant Transform Algorithm, Part II: Customized Version – Handling Rotational Symmetry," Submitted to *IEEE Trans. Acoust., Speech, Signal Processing.*

[8] H. Nussbaumer,"Fast Fourier Transform and Convolution Algorithms," Spring-Verlag, Berlin, (1982).

[9] J.W. Cooley, J.W. Tukey, "An Algorithm for the Machine Computation of Complex Fourier Series," *Math Comp.*, 19, (1965): 297-301.

[10] J. W. Cooley, P. A. Lewis, and P. D. Welch, "The Fast Fourier Transform Algorithm. Programming Considerations in the Calculation of Sine, Cosine and Laplace Transform Transforms,"*J. Sound Vib.*, 12, (1970): 315-337.

[11] S. Winograd, "On Computing Discrete Fourier Transform," *Proc. Nat. Acad. Sci. U.S.A.* 73, (1976): 1005-1006.

[12] S. Winograd, "On Computing Discrete Fourier Transform," *Math. Comp.*, 32, (1978): 175-199.

[13] L. F. Ten Eyck, Acta Crystallographica, A 29, (1973): 183-191.

[14] C. Temperton, "A Note on Prime Factor FFT Algorithms,"*J. of Comp. Phy.*, 52, (1983): 198-204.

[15] H. J. Nussbaumer, and P. Quandalle, "Fast Computation of Discrete Fourier Transforms Using Polynomial Transforms," *IEEE Trans. Acoust., Speech, Signal Processing*, 27, (1979): 169-181.

[16] H. J. Nussbaumer, and P. Quandalle, "New Polynomial Transform Algorithms for Fast DFT Computations," *Proc. IEEE 1979 Internat. Trans. Acoust., Speech, Signal Processing Conf.*, (1979): 510-513.

[17] R. E. Blahut, "Fast Algorithms for Digital Signal Processing," Addison-Wesley, New York, (1985).

[18] I. N. Herstein, "Topics in Algebra," Xerox College Publishing, 2nd ed., Lexington, MA., (1975).