Notes on the Truncated Fourier Transform

Joris van der Hoeven

Dépt. de Mathématiques (Bât. 425)
Université Paris-Sud
91405 Orsay Cedex
France
Email: `joris@texmacs.org`

February 1, 2005

In a previous paper [vdH04], we introduced a truncated version of the classical Fast Fourier Transform. When applied to polynomial multiplication, this algorithm has the nice property of eliminating the "jumps" in the complexity at powers of two. When applied to the multiplication of multivariate polynomials or truncated multivariate power series, a non-trivial asymptotic factor was gained with respect to the best previously known algorithms.

In the present note, we correct two errors which slipped into the previous paper and we give a new application to the multiplication of polynomials with real coefficients. We also give some further hints on how to implement the TFT in practice.

Keywords: Fast Fourier Transform, jump phenomenon, truncated multiplication, FFT-multiplication, multivariate polynomials, multivariate power series.

A.M.S. subject classification: 42-04, 68W25, 42B99, 30B10, 68W30.

1. Introduction

Let $\mathcal{R} \ni 1/2$ be an effective ring of constants (i.e. the usual arithmetic operations $+$, $-$ and $\times$ can be carried out by algorithm). If $\mathcal{R}$ has a primitive n-th root of unity with $n = 2^p$, then the product of two polynomials $P, Q \in \mathcal{R}[X]$ with $\deg P Q < n$ can be computed in time $O(n \log n)$ using the Fast Fourier Transform or FFT [CT65]. If $\mathcal{R}$ does not admit a primitive n-th root of unity, then one needs an additional overhead of $O(\log \log n)$ in order to carry out the multiplication, by artificially adding new root of unity [SS71, CK91].

Besides the fact that the asymptotic complexity of the FFT involves a large constant factor, another classical drawback is that the complexity function admits important jumps at each power of two. These jumps can be reduced by using $(k\, 2^p)$-th roots of unity for small k. They can also be smoothened by decomposing $(n + \delta) \times (n + \delta)$-multiplications as $n \times n$-, $n \times \delta$- and $(n + \delta) \times \delta$-multiplications. However, these tricks are not very elegant, cumbersome to implement, and they do not allow to completely eliminate the jump problem. The jump phenomenon becomes even more important for d-dimensional FFTs, since the complexity is multiplied by 2^d whenever the degree traverses a power of two.

In [vdH04], the author introduced a new kind of "Truncated Fourier Transform" (TFT), which allows for the fast evaluation of a polynomial $P \in \mathcal{R}[X]$ in any number n of well-chosen roots of unity. This algorithm coincides with the usual FFT if n is a power of two, but it behaves smoothly for intermediate values. Moreover, the inverse TFT can be carried out with the same complexity and the approach generalizes to higher dimensions.

Unfortunately, two errors slipped into the final version of [vdH04]: in the multivariate TFT, we forgot certain crossings. As a consequence, the complexity bounds for multiplying polynomials (and power series) in d variables of total degree $< n$ only holds when

$d = o(\log \log n)$. Moreover, the inverse TFT does not generalize to arbitrary "unions of intervals".

The present note has several purposes: correcting the above mistakes in [vdH04], providing further details on how to implement the FFT and TFT in a sufficiently generic way (and suitable for univariate as well as multivariate computation) and a new extension to the case of TFTs with real coefficients. Furthermore, a generic implementation of the FFT and TFT is in progress as part of the standard C++ library MMXLIB of MATHEMAGIX [vdHea05]. We will present a few experimental results, together with suggestions for future improvements. More details will be provided in a forthcoming paper and we refer to [vdH04] for further references.

2. THE FAST FOURIER TRANSFORM

Let $\mathcal{R}$ be an effective ring of constants, $n = 2^p$ with $p \in \mathbb{N}$ and $\omega \in \mathcal{R}$ a primitive n-th root of unity (i.e. $\omega^{n/2} = -1$). The discrete Fast Fourier Transform (FFT) of an n-tuple $(a_0, \ldots, a_{n-1}) \in \mathcal{R}^n$ (with respect to ω) is the n-tuple $(\hat{a}_0, \ldots, \hat{a}_{n-1}) = \mathrm{FFT}_\omega(a) \in \mathcal{R}^n$ with

$$\hat{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij}.$$

In other words, $\hat{a}_i = A(\omega^i)$, where $A \in \mathcal{R}[X]$ denotes the polynomial $A = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1}$. We also say that $(\hat{a}_0, \ldots, \hat{a}_{n-1})$ is the FFT $\hat{A}$ of A.

The F.F.T can be computed efficiently using binary splitting: writing

$$(a_0, \ldots, a_{n-1}) = (b_0, c_0, \ldots, b_{n/2-1}, c_{n/2-1}),$$

we recursively compute the Fourier transforms of $(b_0, \ldots, b_{n/2-1})$ and $(c_0, \ldots, c_{n/2-1})$

$$\begin{aligned}
\mathrm{FFT}_{\omega^2}(b_0, \ldots, b_{n/2-1}) &= (\hat{b}_0, \ldots, \hat{b}_{n/2-1})\,; \\
\mathrm{FFT}_{\omega^2}(c_0, \ldots, c_{n/2-1}) &= (\hat{c}_0, \ldots, \hat{c}_{n/2-1}).
\end{aligned}$$

Then we have

$$\begin{aligned}
\mathrm{FFT}_\omega(a_0, \ldots, a_{n-1}) &= (\hat{b}_0 + \hat{c}_0, \ldots, \hat{b}_{n/2-1} + \hat{c}_{n/2-1}\, \omega^{n/2-1} \\
&\ \ \hat{b}_0 - \hat{c}_0, \ldots, \hat{b}_{n/2-1} - \hat{c}_{n/2-1}\, \omega^{n/2-1}).
\end{aligned}$$

This algorithm requires $n\,p = n \log_2 n$ multiplications with powers of ω and $2\,n\,p$ additions (or subtractions).

In practice, it is most efficient to implement an in-place variant of the above algorithm. We will denote by $[i]_p$ the bitwise mirror of i at length p (for instance, $[3]_5 = 24$ and $[11]_5 = 26$). At step 0, we start with the vector

$$x_0 = (x_{0,0}, \ldots, x_{0,n-1}) = (a_0, \ldots, a_{n-1}).$$

At step $s \in \{1, \ldots, p\}$, we set

$$\begin{pmatrix} x_{s,im_s+j} \\ x_{s,(i+1)m_s+j} \end{pmatrix} = \begin{pmatrix} 1 & \omega^{[i]_s m_s} \\ 1 & -\omega^{[i]_s m_s} \end{pmatrix} \begin{pmatrix} x_{s-1,im_s+j} \\ x_{s-1,(i+1)m_s+j} \end{pmatrix}. \tag{1}$$

for all $i \in \{0, 2, \ldots, n/m_s - 2\}$ and $j \in \{0, \ldots, m_s - 1\}$, where $m_s = 2^{p-s}$. Using induction over s, it can easily be seen that

$$x_{s,im_s+j} = (\mathrm{FFT}_{\omega^{m_s}}(a_j, a_{m_s+j}, \ldots, a_{n-m_s+j}))_{[i]_s},$$

for all $i \in \{0, \ldots, n/m_s - 1\}$ and $j \in \{0, \ldots, m_s - 1\}$. In particular,

$$\begin{aligned} x_{p,i} &= \hat{a}_{[i]_p} \\ \hat{a}_i &= x_{p,[i]_p} \end{aligned}$$

for all $i \in \{0, \ldots, n-1\}$. This algorithm of "repeated crossings" is illustrated in figure 1.

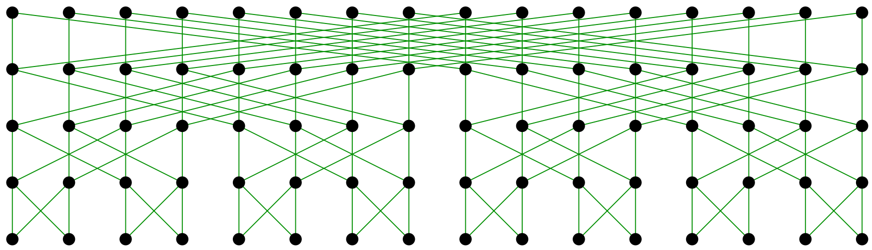

Figure 1. Schematic representation of a Fast Fourier Transform for $n = 16$. The black dots correspond to the $x_{s,i}$, the upper row being $(x_{0,0}, \ldots, x_{0,15}) = (a_0, \ldots, a_{15})$ and the lower row $(x_{4,0}, \ldots, x_{4,15}) = (\hat{a}_0, \hat{a}_8, \hat{a}_4, \hat{a}_{12}, \ldots, \hat{a}_{15})$.

A classical application of the FFT is the multiplication of polynomials $A = a_0 + \cdots + a_{n-1} X^{n-1}$ and $B = b_0 + \cdots + b_{n-1} X^{n-1}$. Assuming that $\deg A B < n$, we first evaluate A and B in $1, \omega, \ldots, \omega^{n-1}$ using the FFT:

$$\begin{aligned} (A(1), \ldots, A(\omega^{n-1})) &= \mathrm{FFT}_\omega(a_0, \ldots, a_{n-1}) \\ (B(1), \ldots, B(\omega^{n-1})) &= \mathrm{FFT}_\omega(b_0, \ldots, b_{n-1}) \end{aligned}$$

We next compute the evaluations $(A(1)\, B(1), \ldots, A(\omega^{n-1})\, B(\omega^{n-1})))$ of $A B$ at $1, \ldots, \omega^{n-1}$. We finally have to recover $A B$ from these values using the inverse FFT. But the inverse FFT with respect to ω is nothing else as $1/n$ times the direct FFT with respect to ω^{-1}. Indeed, for all $(a_0, \ldots, a_{n-1}) \in \mathcal{R}^n$ and all $i \in \{0, \ldots, n-1\}$, we have

$$\mathrm{FFT}_{\omega^{-1}}(\mathrm{FFT}_\omega(a))_i = \sum_{k=0}^{n-1} \sum_{j=0}^{n-1} a_i\, \omega^{(i-k)j} = n\, a_i, \tag{2}$$

since

$$\sum_{j=0}^{n-1} \omega^{(i-k)j} = 0$$

whenever $i \neq k$. This yields a multiplication algorithm of time complexity $O(n \log n)$ in $\mathcal{R}[X]$, when assuming that $\mathcal{R}$ admits enough primitive 2^p-th roots of unity. In the case that $\mathcal{R}$ does not, then new roots of unity can be added artificially [SS71, CK91, vdH02] so as to yield an algorithm of time complexity $O(n \log n \log \log n)$.

3. THE MULTIVARIATE FFT

Given a multivariate polynomial $A \in K[X_1, \ldots, X_d]$ with $\deg_{X_i} A < n_i = 2^{p_i}$ for all i, we may also consider its FFT with respect to each of its variables:

$$\hat{A}_{i_1, \ldots, i_d} = A(\omega_{n_1}^{i_1}, \ldots, \omega_{n_d}^{i_d}),$$

where ω_{n_i} is an n_i-th root of unity for each i. Usually the ω_{n_i} are chosen such that $\omega_2^2 = \omega_1$, $\omega_4^2 = \omega_2$, $\omega_8^2 = \omega_3$, etc. The multivariate FFT is simply computed by taking the univariate FFT with respect to each variable.

Instead of using multi-indices $(i_1, \ldots, i_d)$ for multivariate FFTs, it is more efficient to encode A and $\hat{A}$ using a single array of size $n = n_1 \cdots n_d = 2^p$. This can be done in several ways. Assume that we have ordered $n_1 \geqslant \cdots \geqslant n_d$. The *lexicographical encoding* of the multi-index $(i_1, \ldots, i_d)$ is given by

$$\mathrm{lex}_{n_1, \ldots, n_d}(i_1, \ldots, i_d) = i_1 + i_2 n_1 + i_3 n_1 n_2 + \cdots + i_d n_1 \cdots n_{d-1}.$$

The *simultaneous encoding* of $(i_1, \ldots, i_d)$ is recursively defined by

$$\mathrm{sim}_{n_1, \ldots, n_d}(i_1, \ldots, i_d) = \mathrm{hi}(i_d)\, 2^{p-1} + \cdots + \mathrm{hi}(i_1)\, 2^{p-d} + \mathrm{sim}_{n_1/2, \ldots, n_{d'}/2}(\mathrm{lo}(i_1), \ldots, \mathrm{lo}(i_{d'})),$$

where $\mathrm{hi}(i_j) = i_j \operatorname{div} 2^{p_j - 1}$, $\mathrm{lo}(i_j) = i_j \bmod 2^{p_j - 1}$ and d' is maximal with $n_{d'} \geqslant 4$. For each $i < p_1$, we denote by d_i the maximal number with $n_{d_i} \geqslant 2^i$.

Using one of these encodings, one may use the above in-place algorithm for the computation of the multivariate FFT, when replacing the crossing relation by

$$\begin{pmatrix} x_{s, i m_s + j} \\ x_{s, (i+1) m_s + j} \end{pmatrix} = \begin{pmatrix} 1 & {}^s\omega^{[i]_s m_s} \\ 1 & -{}^s\omega^{[i]_s m_s} \end{pmatrix} \begin{pmatrix} x_{s-1, i m_s + j} \\ x_{s-1, (i+1) m_s + j} \end{pmatrix}. \tag{3}$$

Here

$$({}^0\omega, \ldots, {}^{p-1}\omega) = (\omega_2, \ldots, \omega_{n_d}, \omega_2, \ldots, \omega_{n_{d-1}}, \ \ldots\ , \omega_2, \ldots, \omega_{n_1})$$

in case of the lexicographical encoding and

$$({}^0\omega, \ldots, {}^{p-1}\omega) = (\omega_2, \overset{d_0 \times}{\ldots}, \omega_2, \omega_4, \overset{d_1 \times}{\ldots}, \omega_4, \ \ldots\ , \omega_{n_1}, \overset{d_{p_1 - 1} \times}{\ldots}, \omega_{n_1})$$

in case of the simultaneous encoding. Translated back into terms of monomials, crossings at stage s involve monomials M and $M Y_s$, where

$$(Y_0, \ldots, Y_{p-1}) = (X_d^{n_d/2}, \ldots, X_d, X_{d-1}^{n_{d-1}/2}, \ldots, X_{d-1}, \ \ldots\ , X_1^{n_1/2}, \ldots, X_1)$$

in case of the lexicographical encoding and

$$(Y_0, \ldots, Y_{p-1}) = (X_{d_0}^{n_{d_0}/2}, \ldots, X_1^{n_1/2}, X_{d_1}^{n_{d_1}/4}, \ldots, X_1^{n_{d_1}/4}, \ \ldots\ , X_{d_{p_1 - 1}}, \ldots, X_1)$$

in case of the simultaneous encoding.

4. Implementing the FFT

A challenge for concrete implementations of the FFT in computer algebra systems is to achieve both optimal speed and genericity. Also, we would like to allow for univariate as well as multivariate FFTs. A first attempt in this direction is now part of MMXLIB, the standard C++ library of MATHEMAGIX [vdHea05]. The genericity is obtained using templates. Let us comment some aspects of our implementation.

Roots of unity. There exist at least three important models for the computation with roots of unity during the FFT (see also the explanations and discussion at the end of this section):

1. The Schönhage-Strassen model.
2. The nice prime number model.
3. The floating-point model.

In our implementation, we have created a special template class `fft_root<C>` for roots of unity in `C`, which may be specialized to any of the above models. The class `fft_root<C>` essentially implements methods for multiplication and multiplication with elements of `C`.

The FFT-profile. All necessary data for an FFT of length $n = 2^p$ are stored in a class `fft_transformer<C>`. This comprises the roots of unity ${}^0\omega, \ldots, {}^{p-1}\omega$ for the cross relations (3) and precomputations of $1, \omega, \ldots, \omega^{n'-1}$, where $n' = 2^{p'}$ is the highest order among one of the ${}^i\omega$. These data will be called the *FFT-profile*.

Of course, the precomputation of $1, \omega, \ldots, \omega^{n'-1}$ requires some additional space. Nevertheless, this does not harm if n' is significantly smaller than n (say $n' \leqslant n/4$) or when the encoding of roots of unity requires significantly less space than storing elements of `C` (for instance, in the Schönhage-Strassen model, it suffices to store the shifts). In the remaining case, some space may be saved by precomputing only $1, \omega^4, \ldots, \omega^{n'-4}$ and by computing the remaining values only on demand during the two last steps.

It should also be noticed that we really precompute the array

$$\omega^{[0]_{n'}}, \omega^{-[0]_{p'}}, \omega^{[2]_{p'}}, \omega^{[2]_{p'}}, \omega^{[4]_{p'}}, \omega^{[4]_{p'}}, \ldots, \omega^{[n'-2]_{p'}}, \omega^{-[n'-2]_{p'}} \tag{4}$$

There are several reasons for doing so. First of all, we really need the binary mirrored power $\omega^{[i]_s}$ in the cross relation (3). Secondly, $\omega^{[2i+1]_{n'}} = -\omega^{[2i]_{n'}}$ is easily deduced from $\omega^{[2i]_{n'}}$. Finally, precomputation of the $\omega^{-[2i]_{n'}}$ is useful for the inverse transform.

Partial FFT steps. The core of the FFT is an efficient implementation of one FFT step, i.e. carrying out the cross relations (3) for $i\, m_s + j \in \{0, \ldots, n-1\}$ and fixed s. More generally, for the purpose of preservation of locality (see below) and the TFT (see next sections), it is interesting to allow for ranges $i\, m_s + j \in \{a\, 2^r, \ldots, (a+1)\, 2^r - 1\}$ with $r < p$ and $a \in \{0, \ldots, 2^{p-r} - 1\}$. This key step should be massively inlined and loop-unrolled. For special types `C` and `fft_root<C>`, the operations in `C` and `fft_root<C>` may be written in assembler. Notice that the ${}^s\omega^{[i]_s m_s}$ involved in the cross-relations correspond to a subarray of (4).

Preservation of locality. For very large n, the cross relations (3) typically involve data which are stored at locations which differ by a multiple of the cache size. Consequently, the FFT may give rise to a lot of cache misses and swapping. One remedy to this problem is to divide the FFT in two (or more) parts. The first part regroups the first $P = \lfloor p/2 \rfloor$ steps. For each $i \in \{0, \ldots, n-1\}$, we collect the data $a_i, a_{N+i}, \ldots, a_{n-N+i}$ with $N = 2^P$ in an array stored in contiguous, and perform the FFT on this array. After putting the results back at the original places, we proceed as usual for the remaining $\lceil p/2 \rceil$ steps.

At present, this strategy has not yet been implemented in MMXLIB, since we noticed no substantial slow-down due to cache misses on our computer. However, when memory fills up, we did observe significant swapping, so the above trick might be used in order to obtain a good performance even in critical cases. Also, we currently only tested our implementation with coefficients modulo $3 \cdot 2^{30} + 1$. It may be that the cost of cache misses is negligible w.r.t. the cost of multiplications and divisions modulo this number.

Choosing the appropriate FFT-model. In the Schönhage-Strassen model, we recall that multiplications with roots of unity simply correspond to shiftings, which leads to almost optimal speed. However, the inner FFT multiplication step usually becomes expensive in this model. Indeed, in order to obtain a large number of roots of unity, one has to allow for large constants in `C`. Nevertheless, given a number $2^b - 1$ which fits into a machine word, it should be noticed that Schönhage-Strassen's method still provides b-th roots of unity. Therefore, the inner multiplication step is efficient for certain higher dimensional FFTs. Moreover, in this case, one may exploit the fact that multiplications modulo $2^b - 1$ are fast.

On the other hand, the nice prime number and floating-point models admit many roots of unity, but genuine multiplications (and divisions) have to be carried out during the FFT step. In the nice prime number model, one computes modulo a prime number like $p = 3 \cdot 2^{30} + 1$, which has many 2^i-th roots of unity, yet still fits into a single machine word. One may also use several such prime numbers, in combination with Chinese remaindering. In the floating-point model, one uses floating point approximations of complex numbers. This is particularly useful if fast floating point arithmetic is available, but one always has to carefully deal with rounding errors.

In practice, it follows that the choice of an optimal model for the FFT depends very much on the application. When using the FFT for multiplication, a general rule of the dumb is to balance the costs of the FFT-step and the inner multiplication step. If the inner multiplication step is carried out just once, then Schönhage-Strassen's model is usually optimal. This is for instance the case when multiplying two large integers. If one FFT corresponds to several inner multiplications, as in the case of matrix multiplication with large integer coefficients, one should opt for the nice prime or floating-point model, depending on the efficiency of division, floating-point operations and the available amount of space.

In between, it may sometimes be useful to use Schönhage-Strassen's method with cube roots (instead of square roots) of the number of digits (or degree) [vdH02, Section 6.1.3]. Similarly, one may force an extra iteration (i.e. immediately use fourth roots, but loose an additional factor 2). Finally, in the case of multivariate FFTs (we consider that large integer coefficients account for an additional dimension), one may artificially generate additional roots of unity. For instance, if $\mathcal{R}$ has a 2^p-th root of unity ω, then one may multiply bivariate polynomials in $\mathcal{R}[x, y]/(x^{2^p} - 1, y^{2^{p+1}} - 1)$, by using x as a 2^p-th root of ω during the FFT w.r.t. y.

5. THE TRUNCATED FOURIER TRANSFORM

Let $n = 2^p$, $l \in \{n/2 + 1, \ldots, n\}$ and let $\omega \in \mathcal{R}$ be a primitive n-th root of unity. The Truncated Fourier Transform (TFT) from [vdH04, Section 3] takes a tuple $(a_0, \ldots, a_{l-1})$ on input and produces $(\tilde{a}_0, \ldots, \tilde{a}_{l-1})$ with $\tilde{a}_i = \hat{a}_{[i]_p}$ for all i. More generally, if $\mathcal{S}$ is a subset of $\{0, \ldots, n-1\}$ and $a: \mathcal{S} \to \mathcal{R}$, then we define $\tilde{a} = \mathrm{TFT}_{\mathcal{S},\omega}(a)$ by

$$\begin{array}{rcl} \tilde{a}: \mathcal{S} & \longrightarrow & \mathcal{R} \\ i & \longmapsto & \tilde{a}_i = \hat{a}_{[i]_p} = \sum_{j \in \mathcal{S}} a_j \, \omega^{j [i]_p} \end{array}$$

Even more generally, one may select a target subset $\mathcal{T}$ of $\{0, \ldots, n-1\}$ which is different from $\mathcal{S}$, and define $\tilde{a} = \mathrm{TFT}_{\mathcal{S} \to \mathcal{T},\omega}(a)$ by

$$\begin{array}{rcl} \tilde{a}: \mathcal{T} & \longrightarrow & \mathcal{R} \\ i & \longmapsto & \tilde{a}_i = \hat{a}_{[i]_p} = \sum_{j \in \mathcal{S}} a_j \, \omega^{j [i]_p} \end{array}$$

In order to compute the TFT, we simply consider the computation graph of a complete FFT and throw out all computations which do not contribute to the result $\tilde{a}$ (see figures 2, 3 and 4). If the set $\mathcal{S}$ is "sufficiently connected", then the cost of the computation of $\tilde{a}$ is $O((|\mathcal{S}| + |\mathcal{T}|)\, p)$. For instance, for $\mathcal{S} = \{0, \ldots, l-1\}$, we have proved [vdH04]:

THEOREM 1. *Let $n = 2^p$, $l \leqslant n$ and let $\omega \in \mathcal{R}$ be a primitive n-th root of unity in $\mathcal{R}$. Then the TFT of an l-tuple $(a_0, \ldots, a_{l-1})$ w.r.t. ω can be computed using at most $l\,p + n$ additions (or subtractions) and $\lceil (l\,p + n)/2 \rceil$ multiplications with powers of ω.*

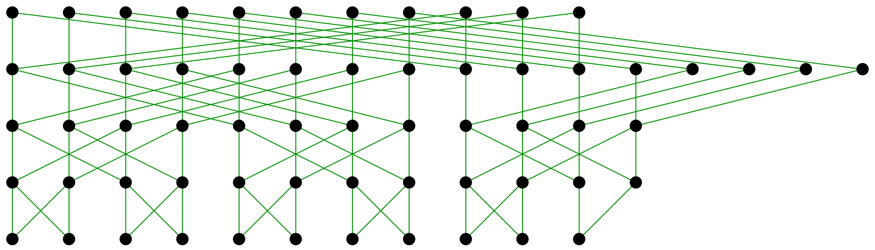

Figure 2. Schematic representation of a TFT for $n=16$ and $l=11$.

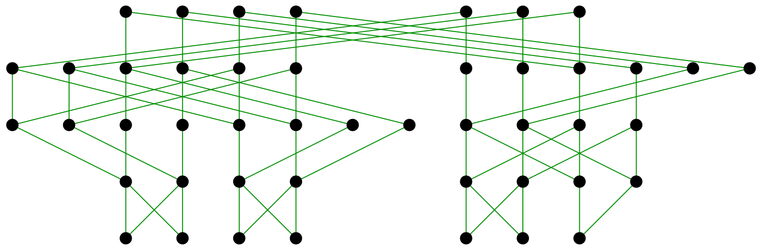

Figure 3. Schematic representation of a TFT for $n=16$ and $\mathcal{S}=\{2,3,4,5,8,9,10\}$.

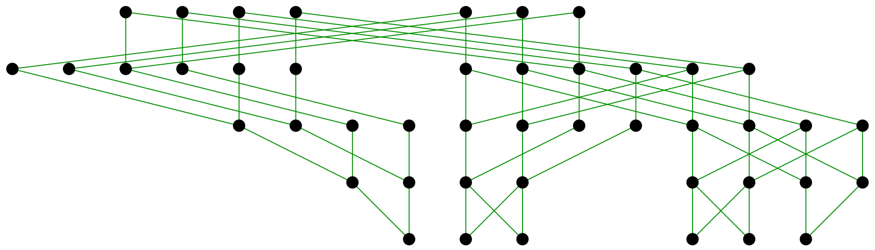

Figure 4. Schematic representation of an asymmetric TFT for $n=16$ with $\mathcal{S}=\{2,3,4,5,8,9,10\}$ and $\mathcal{T}=\{7,8,9,12,13,14\}$.

6. Inverting the Truncated Fourier Transform

In order to use the TFT for the multiplication of numbers, polynomials or power series, we also need an algorithm to compute the inverse transform. In this section, we give such an algorithm in the case when $\mathcal{T}=\mathcal{S}$ and when $\mathcal{S}$ is an initial segment for the (partial) "bit ordering $\preccurlyeq$" on $\{0,\ldots,n-1\}$: given numbers $i=i_0+i_1 2+\cdots+i_{p-1}2^{p-1}$ and $j=j_0+j_1 2+\cdots+j_{p-1}2^{p-1}$ (with $i_k,j_k\in\{0,1\}$), we set $i\preccurlyeq j$ if $i_k\leqslant j_k$ for all k. We recall that an initial segment of $\{0,\ldots,n-1\}$ is a set $\mathcal{S}$ with $a\in\mathcal{S}\wedge b\preccurlyeq a\Rightarrow b\in\mathcal{S}$.

The inverse TFT is based on the key observation that the cross relation can also be applied to compute "one diagonal from the other diagonal". More precisely, given a relation

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & \alpha \\ 1 & -\alpha \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix}, \tag{5}$$

where $\alpha=\omega^i$ for some i, we may clearly compute (a,b) as a function of (c,d) and vice versa

$$\begin{pmatrix} c \\ d \end{pmatrix} = 2^{-1}\begin{pmatrix} 1 & 1 \\ \alpha^{-1} & -\alpha^{-1} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}, \tag{6}$$

but we also have

$$\begin{pmatrix} b \\ c \end{pmatrix} = \begin{pmatrix} 2 & -\alpha \\ 1 & -\alpha \end{pmatrix} \begin{pmatrix} a \\ d \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} a \\ d \end{pmatrix} = \begin{pmatrix} -1 & 2 \\ -\alpha^{-1} & \alpha^{-1} \end{pmatrix} \begin{pmatrix} b \\ c \end{pmatrix} \tag{8}$$

Moreover, these relations only involve shifting (multiplication and division by 2), additions, subtractions and multiplications by roots of unity.

In order to state our in-place algorithm for computing the inverse TFT, we will need some more notations. At the very start, we have $\tilde{a}: \mathcal{S} \to \mathcal{R}$ and $b: \overline{\mathcal{S}} \to \mathcal{R}$ on input, where $\overline{\mathcal{S}}$ is the complement of $\mathcal{S}$ in $\mathcal{N} = \{0, \ldots, n-1\}$ and b is the zero function. Roughly speaking, the algorithm will replace $\tilde{a}$ by its inverse TFT a and b by its direct TFT $\tilde{b}$. In the actual algorithm, the array b will be stored in a special way (see the next section) and only a part of the computation of $\tilde{b}$ will really be carried out.

Our algorithm proceeds by recursion. In the recursive step, $\mathcal{N}$ is replaced by a subset of $\{0, \ldots, n-1\}$ of the form $\{a\, 2^{n-s}, \ldots, (a+1)\, 2^{n-s} - 1\}$, where s is the current step and $a \in \{0, \ldots, 2^s - 1\}$. The sets $\mathcal{S}$ and $\overline{\mathcal{S}}$ will again be subsets of $\mathcal{N}$ with $\mathcal{N} = \mathcal{S} \amalg \overline{\mathcal{S}}$, and $\mathcal{S} - a\, 2^{n-s}$ is recursively assumed to be an initial segment for the bit ordering. Given a subset $\mathcal{A}$ of $\mathcal{S}$, we will denote $\mathcal{A}^{\downarrow} = \mathcal{A} \cap \mathcal{N}^{\downarrow}$ and $\mathcal{A}^{\uparrow} = \mathcal{A} \cap \mathcal{N}^{\downarrow}$, where

$$\begin{aligned} \mathcal{N}^{\downarrow} &= \{2\, a\, 2^{n-s-1}, \ldots, (2\, a+1)\, 2^{n-s-1} - 1\} \\ \mathcal{N}^{\uparrow} &= \{(2\, a+1)\, 2^{n-s-1}, \ldots, (2\, a+2)\, 2^{n-s-1} - 1\}. \end{aligned}$$

Similarly, given $u: \mathcal{A} \to \mathcal{R}$, we will denote by $u^{\downarrow}$ and $u^{\uparrow}$ the restrictions of u to $\mathcal{A}^{\downarrow}$ resp. $\mathcal{A}^{\uparrow}$. We now have the following recursive in-place algorithm for computing the inverse TFT (see figure 5 for an illustration of the different steps).

Algorithm ITFT$(\tilde{a}: \mathcal{S} \to \mathcal{R}, b: \overline{\mathcal{S}} \to \mathcal{R}, s)$

If $s = p$ or $\mathcal{S} = \varnothing$ then return
If $\overline{\mathcal{S}} = \varnothing$ then apply the partial inverse FFT on $\tilde{a}$ and return
For $i \in \overline{\mathcal{S}}^{\downarrow}$, cross b_i with b_{i+m_s} using (5)
ITFT$(\tilde{a}^{\downarrow}, b^{\downarrow}, s+1)$
For $i \in \mathcal{S}^{\downarrow} \cap (\overline{\mathcal{S}}^{\uparrow} - m_s)$, cross $\tilde{a}_i$ with b_{i+m_s} using (8)
ITFT$(\tilde{a}^{\uparrow}, b^{\uparrow}, s+1)$
For $i \in \mathcal{S}^{\downarrow} \cap (\mathcal{S}^{\uparrow} - m_s)$, cross $\tilde{a}_i$ with $\tilde{a}_{i+m_s}$ using (6)

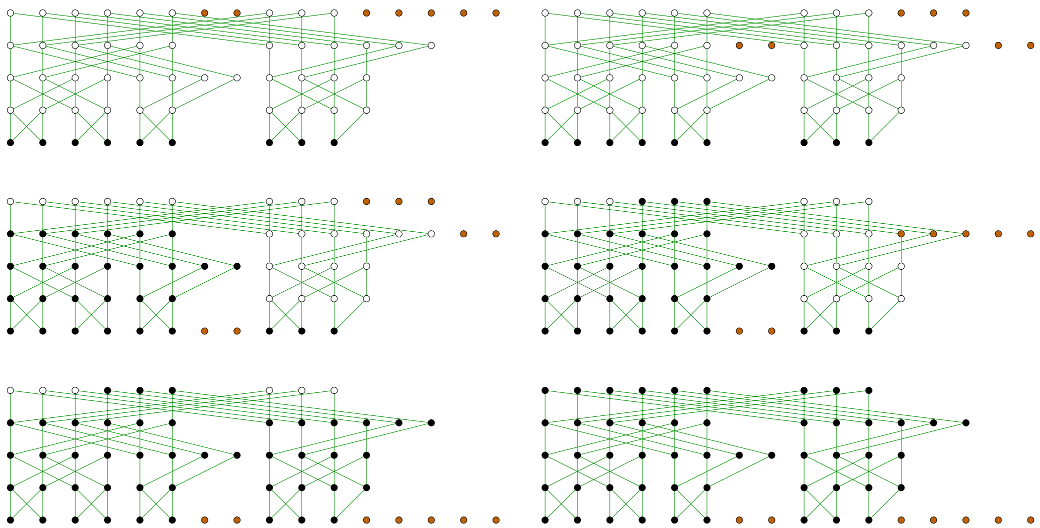

Figure 5. Schematic representation of the recursive computation of the inverse TFT for $n = 16$ and $\mathcal{S} = \{1, 2, 3, 4, 5, 6, 9, 10, 11\}$. The different images show the progression of the known values $x_{i,j}$ (the black dots) during the different computations at stage $s = 0$. Between the second and third image, we recursively apply the algorithm, as well as between the fourth and fifth image.

Applying the algorithm for $\mathcal{S} = \{0, \ldots, l-1\}$ with $n/2 \leqslant l < n$, we have

THEOREM 2. *Let $n=2^p$, $l \leqslant n$ and let $\omega \in \mathcal{R}$ be a primitive n-th root of unity in $\mathcal{R}$. Then the l-tuple $(a_0, \ldots, a_{l-1})$ can be recovered from its TFT w.r.t. ω using at most $l\,p+n$ shifted additions (or subtractions) and $\lceil (l\,p+n)/2 \rceil$ multiplications with powers of ω.*

REMARK 3. Even though it seems that the linear transformation $\mathrm{TFT}_{\mathcal{S},\omega}\colon a \mapsto \tilde{a}$ has a non-zero determinant for any $\mathcal{S}$, an algorithm of the above kind does not always work. The simplest example when our method fails is for $n=4$ and $\mathcal{S}=\{1,2\}$. Nevertheless, the condition that $\mathcal{S}$ is an initial segment for the bit ordering on $\{0,\ldots,n-1\}$ is not a necessary condition. For instance, a slight modification of the algorithm also works for final segments and certain more general sets like $\mathcal{S}=\{0,1,3,4,5,7\}$ for $n=8$. We will observe in the next section that the bit ordering condition on $\mathcal{S}$ is naturally satisfied when we take multivariate TFTs on initial segments of $\mathbb{N}^d$.

7. THE MULTIVARIATE TFT

Let $n_1=2^{p_1} \geqslant \cdots \geqslant n_d=2^{p_d}$ and let $\omega_{n_i} \in \mathcal{R}$ be a primitive n_i-th root of unity for each i. Given subsets $\mathcal{S},\mathcal{T}$ of $\mathcal{N}=\{0,\ldots,n_1-1\} \times \cdots \times \{0,\ldots,n_d-1\}$ and a mapping $a\colon \mathcal{S} \to \mathcal{R}$, the multivariate TFT of a is a mapping $\tilde{a}=\mathrm{TFT}_{\mathcal{S}\to\mathcal{T},\omega_{n_1},\ldots,\omega_{n_d}}(a)\colon \mathcal{T} \to \mathcal{R}$ defined by

$$\begin{array}{rcl} \tilde{a}\colon \mathcal{T} & \longrightarrow & \mathcal{R} \\ (i_1,\ldots,i_d) & \longmapsto & \tilde{a}_{i_1,\ldots,i_d} = \displaystyle\sum_{(j_1,\ldots,j_d)\in\mathcal{S}} a_{j_1,\ldots,j_d}\, \omega_{n_1}^{j_1[i_1]_{p_1}} \cdots \omega_{n_d}^{j_d[i_d]_{p_d}} \end{array}$$

See figure 6 for an example with $\mathcal{T}=\mathcal{S}$ in dimension $d=2$. The multivariate TFT and its inverse are computed in a similar way as in the univariate case, using one of the encodings from see section 3 of tuples in $\mathcal{N}$ by integers in $\{0,\ldots,n_1\cdots n_d-1\}$ and using the corresponding FFT-profile determined by ${}^0\omega,\ldots,{}^{p-1}\omega$ instead of $\omega^{n/2},\omega^{n/4},\ldots,\omega$.

We generalize the bit-ordering $\preccurlyeq$ on sets of the form $\{0,\ldots,2^p-1\}$ to sets $\mathcal{N}$ of indices by

$$(i_1,\ldots,i_d) \preccurlyeq (j_1,\ldots,j_d) \Leftrightarrow i_1 \preccurlyeq j_1 \wedge \cdots \wedge i_d \preccurlyeq j_d.$$

If ϕ is one of the encodings $\mathrm{lex}_{n_1,\ldots,n_d}$ or $\mathrm{sim}_{n_1,\ldots,n_d}$ from section 3, then it can be checked that

$$(i_1,\ldots,i_d) \preccurlyeq (j_1,\ldots,j_d) \Leftrightarrow \phi(i_1,\ldots,i_d) \preccurlyeq \phi(j_1,\ldots,j_d)$$

If $\mathcal{T}=\mathcal{S}$ is an initial segment for the bit ordering, this ensures that the inverse TFT also generalizes to the multivariate case.

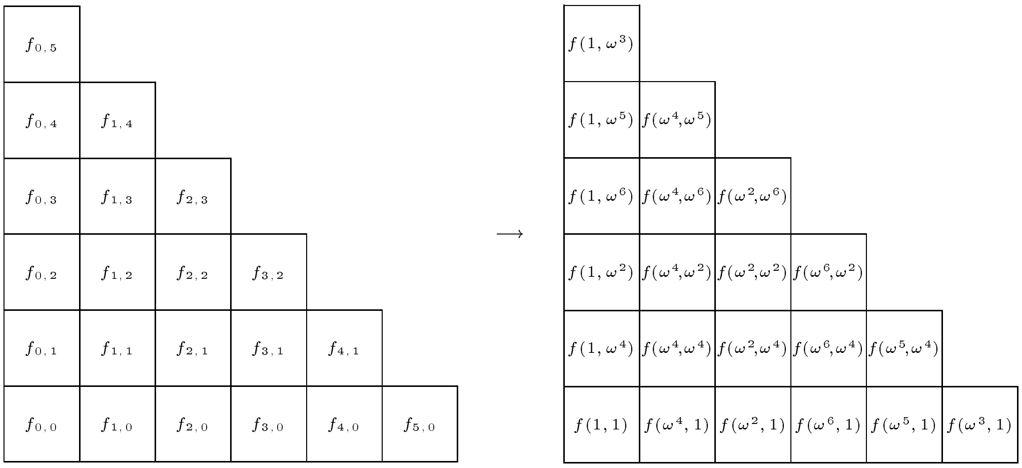

Figure 6. Illustration of a TFT in two variables ($\omega=\omega_8$).

From the complexity analysis point of view, two special cases are particularly interesting. The first *block case* is when $\mathcal{S} = \{0, \ldots, l_1 - 1\} \times \cdots \times \{0, \ldots, l_d - 1\}$ with $n_i/2 < l_i \leqslant n_i$ for all i. Then using the lexicographical encoding, the multivariate TFT can be seen as a succession of d univariate TFTs whose coefficients are mappings $c: \mathcal{S}_i \to \mathcal{R}$, with

$$\mathcal{S}_i = \{0, \ldots, l_1 - 1\} \times \cdots \times \{0, \ldots, l_{i-1} - 1\} \times \{0, \ldots, l_{i+1} - 1\} \times \cdots \times \{0, \ldots, l_d - 1\}.$$

Setting $n = n_1 \cdots n_d$, $l = l_1 \cdots l_d$ and $p = p_1 \cdots p_d$, theorems 1 and 2 therefore generalize to:

THEOREM 4. *With the above notations, the direct and inverse TFT of a mapping $a: \mathcal{S} \to \mathcal{R}$ can be computed using at most $\sigma = l\,(p + \frac{n_1}{l_1} + \cdots + \frac{n_d}{l_d})$ shifted additions (or subtractions) and $\lceil \sigma/2 \rceil$ multiplications with powers of the ω_i.*

REMARK 5. The power series analogue of the block case, i.e. the problem of multiplication in the ring $\mathcal{A} = \mathcal{R}[z_1, \ldots, z_d]/(z_1^{l_1}, \ldots, z_d^{l_d})$ is also an interesting. The easiest instance of this problem is when $l_1 = \cdots = l_d = l = 2^p$ for some p. In that case, we may introduce a new variable t and compute in $\mathcal{R}[z_1, \ldots, z_d]/(z_1^l - t, \ldots, z_d^l - t, t^k - 1)$ instead of $\mathcal{A}$, where k is the smallest power of two with $k \geqslant d + 1$. This gives rise to yet another FFT-profile of the form $(\omega_{kl}, \ldots, \omega_{2l}, \omega_l, \ldots, \omega, \ldots, \omega_l, \ldots, \omega)$ and a complexity in $O(d\, l^d \log l^d)$. The trick generalizes to the case when the l_i are all powers of two, but the general case seems to require non-binary FFT steps.

The other important *simplicial case* is when $n_1 = \cdots = n_d$ and $\mathcal{S} = \{(i_1, \ldots, i_d): i_1 + \cdots + i_d < l\}$ for some fixed l with $n_1/2 < l \leqslant n_1$ (and more generally, one may consider weighted degrees). In this case, we choose the simultaneous encoding for the multivariate TFT (see figure 7 for an example). Now the size of $\mathcal{S}$ is $s_{l,d} = \binom{l+d-1}{d}$. At stages $d\,(i-1), \ldots, d\,i - 1$, we notice that the number of "active nodes" for the TFT is bounded by $B_i = \binom{d + 2^i - 1}{d} \min\,(s_{l,d}, (n_1/2^i)^d)$. When $d = o(\log \log l)$, we have

$$B_1 + \cdots + B_{p_1} = O(s_{l,d} \log s_{l,d}),$$

which proves:

THEOREM 6. *With the above notations, and assuming that $d = o(\log \log l)$, the direct and inverse TFT of a mapping $a: \mathcal{S} \to \mathcal{R}$ can be computed using $O(s_{l,d} \log s_{l,d})$ shifted additions (or subtractions) and multiplications with powers of ω_{n_1}.*

REMARK 7. The complexity analysis of the simplicial case in [vdH04, Section 5] contained a mistake. Indeed, for the multivariate TFT described there, the computed transformed coefficient $\tilde{a}_{l-1,0,\ldots,0}$ does not depend on $a_{0,\ldots,0,l-1}$, which is incorrect.

REMARK 8. Theorem 6 may be used to multiply formal power series in a similar way as in [vdH04]. This multiplication algorithm requires an additional logarithmic overhead. Contrary to what was stated in [vdH04], we again need the assumption that $d = o(\log \log l)$.

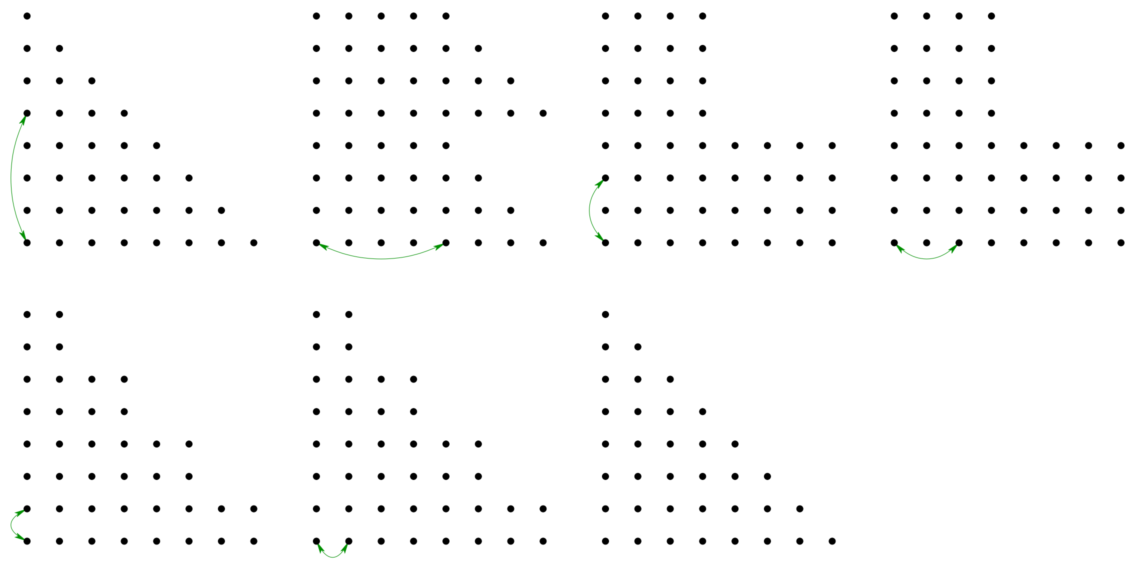

Figure 7. Illustration of the different stages of a bivariate simplicial TFT for $n_1 = n_2 = l = 8$. The black dots correspond to the active nodes at each stage. The arrows indicate two nodes which will be crossed between the current and the next stage.

8. IMPLEMENTING THE TFT

In order to implement the TFT and its inverse, one first has to decide how to represent the sets $\mathcal{S}$ and $\mathcal{T}$, and mappings $a: \mathcal{S} \to \mathcal{R}$. A convenient approach is to represent $\mathcal{S}$, $\mathcal{T}$ and a (for the direct TFT) or $\mathcal{S}$, $\tilde{a}$ and b (for the inverse TFT) by a single binary "TFT-tree". Such a binary tree corresponds to what happens on a subset $\mathcal{I} \subseteq \{0, \ldots, n-1\}$ of the form $\mathcal{I} = \{a\, 2^r, \ldots, (a+1)\, 2^r - 1\}$ with $a, r \in \mathbb{N}$. The binary tree is of one of the following forms:

Leafs. When the tree is reduced to a leaf, then it explicitly contains a map $u: \mathcal{I} \to \mathcal{R}$, which is represented by an array of length 2^r in memory. By convention, if u is reduced to a null-pointer, then u represents the zero map. Moreover, the node contains a flag in order to indicate whether $\mathcal{I} \subseteq \mathcal{S}$ or $\mathcal{I} \subseteq \mathcal{T}$ (for leafs, it is assumed that we either have $\mathcal{I} \subseteq \mathcal{S}$ or $\mathcal{I} \cap \mathcal{S} = \varnothing$ and similarly for $\mathcal{T}$).

Binary nodes. When the tree consists of a binary node with subtrees t_0 and t_1, then t_0 encodes what happens on the interval $\mathcal{I}^{\downarrow}$, whereas t_1 encodes what happens on the interval $\mathcal{I}^{\uparrow}$. For convenience, the tree still contains a flag to indicate whether $\mathcal{I} \cap \mathcal{S} \neq \varnothing$ or $\mathcal{I} \cap \mathcal{T} \neq \varnothing$.

Notice that the bounds of the interval $\mathcal{I}$ need not to be explicitly present in the representation of the tree; when needed, they can be passed as parameters for recursive functions on the trees.

Only the direct TFT has currently been implemented in MMXLIB. This implementation roughly follows the above ideas and can be used in combination with arbitrary FFT-profiles. However, the basic arithmetic with "TFT-trees" has not been very well optimized yet.

In tables 1–4, we have given a few benchmarks on a 2.4GHz AMD architecture with 512Mb of memory. We use $\mathbb{Z}/(3 \cdot 2^{30} + 1)\, \mathbb{Z}$ as our coefficient ring and we tested the simplicial multivariate TFT for total degree $< n$ and dimension d. Our table both shows the size of the input $s = |\mathcal{S}|$, the real and average running times t_{tot} and $t_{\mathrm{av}} = t_{\mathrm{tot}}/s$, the theoretical number of crossings to be computed c_{tot}, its average $c_{\mathrm{av}} = c_{\mathrm{tot}}/s$ and the ratio $t_{\mathrm{av}}/c_{\mathrm{av}} = t_{\mathrm{tot}}/c_{\mathrm{tot}}$. The number ρ correspond to the running time divided by the running time of the univariate TFT for the same input size. The tables both show the most favourable cases when n is a power of two and the worst cases when n is a power of two plus one.

d	n	s	t_{tot} in ms	t_{av} in µs	c_{tot}	c_{av}	$t_{\mathrm{av}}/c_{\mathrm{av}}$	ρ
1	16	16	0.01090	0.681	32	2	0.341	1
1	17	17	0.01899	1.117	63	3.71	0.301	1
1	256	256	0.1452	0.567	1024	4	0.142	1
1	257	257	0.2314	0.901	1535	5.97	0.151	1
1	4096	4096	3.206	0.783	24576	6	0.131	1
1	4097	4097	4.413	1.077	32767	8.00	0.135	1
1	65536	65536	68.33	1.043	524288	8	0.130	1
1	65537	65537	88.67	1.353	655359	10.00	0.135	1
1	1048576	1048576	1371	1.307	10485760	10	0.131	1
1	1048577	1048577	1710	1.631	12582911	12	0.136	1

Table 1. Timings for the direct univariate TFT of degree $< n$.

d	n	s	t_{tot} in ms	t_{av} in µs	c_{tot}	c_{av}	$t_{\mathrm{av}}/c_{\mathrm{av}}$	ρ
2	4	10	0.0153	1.531	27	2.70	0.567	1.25
2	5	15	0.0434	2.891	101	6.73	0.430	3.30
2	16	136	0.1671	1.229	724	5.32	0.231	1.43
2	17	153	0.4566	2.984	1758	11.49	0.260	3.83
2	64	2080	2.393	1.150	15824	7.61	0.151	1.15
2	65	2145	5.857	2.731	31498	14.68	0.186	2.74
2	256	32896	44.8	1.362	319296	9.71	0.140	1.05
2	257	33153	90.67	2.735	566330	17.08	0.160	2.14
2	1024	524800	851	1.622	6159616	11.74	0.138	1.03
2	1025	525825	1578	3.001	10096890	19.20	0.156	1.91

Table 2. Timings for the direct bivariate TFT of total degree $< n$.

d	n	s	t_{tot} in ms	t_{av} in µs	c_{tot}	c_{av}	$t_{\mathrm{av}}/c_{\mathrm{av}}$	ρ
3	4	20	0.0941	4.704	109	5.45	0.863	5.18
3	5	35	0.2797	7.992	509	14.54	0.550	8.50
3	16	816	2.2222	2.723	9324	11.43	0.238	3.68
3	17	969	9.4546	9.757	25807	26.63	0.366	13.90
3	64	45760	143	3.125	729008	15.93	0.196	2.76
3	65	47905	478	9.978	1640523	34.25	0.291	8.85
3	256	2829056	9005	3.183	55049920	19.46	0.164	2.05
3	257	2862209	234173	81.816	111116603	38.82	2.108	52.97

Table 3. Timings for the direct three-dimensional TFT of total degree $< n$. The last colored rectangles correspond to an input for which the computer started heavy swapping.

d	n	s	t_{tot} in ms	t_{av} in µs	c_{tot}	c_{av}	$t_{\text{av}}/c_{\text{av}}$	ρ
4	2	5	0.0175809	3.51617	20	4	0.879	1.90
4	3	15	0.155039	10.3359	206	13.73	0.753	11.88
4	8	330	2.09375	6.3447	5346	16.2	0.392	8.10
4	9	495	13.4	27.0707	20327	41.06	0.659	41.21
4	32	52360	508	9.70206	1436052	27.43	0.354	8.61
4	33	58905	2262	38.4008	3820448	64.86	0.592	24.59
5	32	376992	9541	25.3082	19603488	52.00	0.487	19.04
6	16	54264	3941	72.6264	3962120	73.02	0.995	65.96
7	8	3432	325	94.697	254493	74.15	1.277	112.13
8	8	6435	2051	318.726	730912	113.58	2.806	345.22

Table 4. Assorted timings for higher dimensional simplicial TFTs.

9. Improving the implementation of the TFT

A few conclusions can be drawn from tables 1–4. First of all, in the univariate case, the TFT behaves more or less as theoretically predicted. The non-trivial ratios $t_{\text{av}}/c_{\text{av}}$ in higher dimensions indicate that the set $\mathcal{S}$ is quite disconnected. Hence, optimizations in the implementation of TFT-trees may cause non-trivial gains. The differences between the running times for the best and worse cases in higher dimensions show that the multivariate TFT is still quite sensible to the jump phenomenon (even though less than a full FFT). Finally, in the best case, the ratio ρ be comes reasonably small. Indeed, ρ should theoretically tend to 1 and should be bounded by $d!$ in order to gain something w.r.t. a full FFT. However, ρ can become quite large in the worse case. The various observations suggest the following future improvements.

Avoiding expression swell. In the case of simplicial TFTs, the number of active nodes swells quite dramatically at the first stages of the TFT (the size may roughly be multiplied by a factor 2^d). In this special case, it may therefore be good to compress the first d steps into a single step. Moreover, one benefits from the fact that this compression can be done in a particularly efficient way. Indeed, let $\acute{a}_{i_1,\ldots,i_d}$ be the transform of $a_{i_1,\ldots,i_d}$ after d steps. We have

$$\acute{a}_{i_1,\ldots,i_d} = a_{i_1,\ldots,i_d} + a_{i_1+n/2,\ldots,i_d} + \cdots + a_{i_1,\ldots,i_d+n/2}$$

for all $(i_1, \ldots, i_d) \in \mathcal{S} \cap \{0, \ldots, n/2-1\}^d$ and

$$\acute{a}_{i_1,\ldots,i_{j-1},i_j+n/2,i_{j+1},\ldots,i_d} = \acute{a}_{i_1,\ldots,i_d} - 2\, a_{i_1,\ldots,i_{j-1},i_j+n/2,i_{j+1},\ldots,i_d}$$

for each $j \in \{1, \ldots, d\}$. It can be hoped that this acceleration approximately reduces the current running times for the worst case to the current running times for the best case. For high dimensions, it may be useful to apply the trick a second time for the next d stages $d+1, \ldots, 2\,d$; however, the corresponding formulas become more complicated. We hope that further study will enable the replacement of the condition $d = o(\log \log l)$ in theorem 6 by a better condition, like $d = O(\log l)$.

Avoiding small leafs. At the other back-end, the overhead in the manipulation of TFT-trees may be reduced by allowing only for leafs whose corresponding arrays have a minimal size of 2, 4, 8, or more. Also, if the intersection of $\mathcal{S}$ with the interval $\mathcal{I}$ of a TFT-tree has a high density (say $>1/2$), then we may replace the TFT-tree by a leaf.

Reducing the jump phenomenon. When the TFT is used for polynomial or power series multiplication, and n is slightly above a power of two 2^p, then classical techniques may be used for reducing the jump phenomenon. For instance, the polynomials A and B may be decomposed $A = A^{\downarrow} + A^{\uparrow}$ and $B = B^{\downarrow} + B^{\uparrow}$, where $A^{\downarrow}$ is the part of total degree $< 2^p$, $A^{\uparrow} = A - A^{\downarrow}$ and similarly for B. Then $A^{\downarrow} B^{\downarrow}$ is computed using the TFT at order 2^p and the remainder $A^{\uparrow} B^{\downarrow} + A^{\downarrow} B^{\uparrow} + A^{\uparrow} B^{\uparrow}$ by a more naive method.

10. THE TFT FOR REAL COEFFICIENTS

Assume now that $\mathcal{R}$ is a ring such that $\mathcal{R}[\mathrm{i}]$ has many 2^p-th roots of unity. Typically, this is the case when $\mathcal{R}$ is the "field" of floating point numbers. A disadvantage of the standard FFT or TFT in this context is that the size of the input is doubled in order to represent numbers in $\mathcal{R}[\mathrm{i}]$. This doubling of the size actually corresponds to the fact that the FFT or TFT contains redundant information in this case. Indeed, given $A = a_0 + \cdots + a_{n-1} X^{n-1}$ with $n = 2^p$, we have $A(\omega^{-1}) = \overline{A(\omega)}$ for any n-th root of unity ω. This suggests to evaluate A either in ω or ω^{-1}.

With the notations from section 5, let $\mathcal{S} \subseteq \{0, \ldots, n-1\}$ be an initial segment for the bit ordering $\preccurlyeq$. As our target set $\mathcal{T}$, we take

$$\mathcal{T} = \mathcal{S} \cap \{0, 1, 2, 4, 5, 8, 9, \ldots, 12, \ldots, n/2, \ldots, 3\,n/4 - 1\},$$

The "real TFT" now consists of applying the usual TFT with source $\mathcal{S}$ and target $\mathcal{T}$. It can be checked that crossings of "real nodes" introduce "complex nodes" precisely then when one of the two nodes disappears at the next stage (see figure 8). It can also be checked that the inverse TFT naturally adapts so as to compute the inverse real TFT. For "sufficiently connected" sets $\mathcal{S}$, like $\mathcal{S} = \{0, \ldots, l-1\}$, the real TFT is asymptotically twice as fast as the complex TFT.

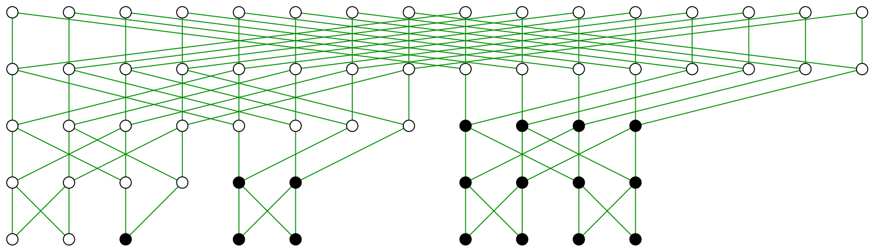

Figure 8. Schematic representation of a real FFT. The white nodes correspond to real numbers and the black nodes to complex numbers.

BIBLIOGRAPHY

[CK91] D.G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. Acta Informatica, 28:693–701, 1991.

[CT65] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. Math. Computat., 19:297–301, 1965.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. Computing 7, 7:281–292, 1971.

[vdH02] J. van der Hoeven. Relax, but don't be too lazy. JSC, 34:479–542, 2002.

[vdH04] J. van der Hoeven. The truncated Fourier transform and applications. In J. Gutierrez, editor, Proc. ISSAC 2004, pages 290–296, Univ. of Cantabria, Santander, Spain, July 4–7 2004.

[vdHea05] J. van der Hoeven et al. Mmxlib: the standard library for Mathemagix, 2002–2005. `ftp://ftp.mathemagix.org/pub/mathemagix/targz`.