The Truncated Fourier Transform

Paul Vrbik
University of Western Ontario

December 10, 2009

The Fast Fourier Transform

Let:

- $\mathcal{R}$ be an *effective ring* (meaning that there are effective procedures for computing the sum, difference and product of two elements of $\mathcal{R}[x]$),
- $\omega \in \mathcal{R}[x]$ be a primitive $n = 2^p$-th root of unity (i.e. $\omega^{n/2} = -1$).

Definition (FFT - technical)

The discrete Fast Fourier Transform (FFT) maps

$$(a_0, \ldots, a_{n-1}) \in \mathcal{R}^n \xrightarrow{\text{FFT}} (\hat{a}_0, \ldots, \hat{a}_{n-1}) \in \mathcal{R}^n$$

with

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

Definition (FFT - useful)

Let $A = a_0 + a_x x + \cdots + a_{n-1}x^{n-1} \in \mathcal{R}[x]$ then FFT maps

$$A(x) \xrightarrow{\text{FFT}} (A(\omega^0), A(\omega^1), \ldots, A(\omega^{n-1}))$$

or $\hat{a}_i = A(\omega^i)$.

The FFTcan be computed efficiently using binary splitting: Write

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

and do

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

Then we have

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

In practice it is more efficient to implement an *in-place* algorithm rather than a recursive one.

First we define the bitwise mirror of i at length p, denoted $[i]_p$.

Example

$[3]_5 = 24$ because $3 = 00011_2$ whose reverse is $11000_2 = 24$.

$[11]_5 = 26$ because $11 = 01011_2$ whose reverse is $11010_2 = 26$.

At step $s \in \{1, \ldots, p\}$ with $m_s = 2^{p-s}$ we set

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

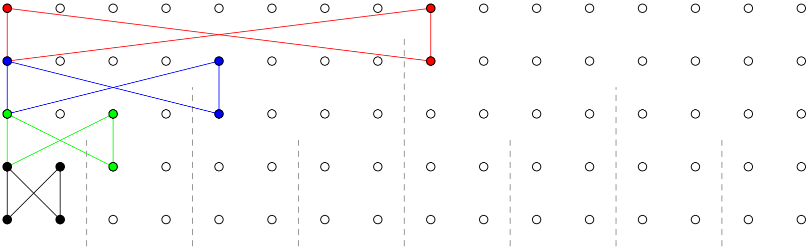


Figure: Butterflies. The black dots correspond to the $x_{s,i}$. The top row corresponding to $s = 0$. In this case $n = 16 = 2^4$.

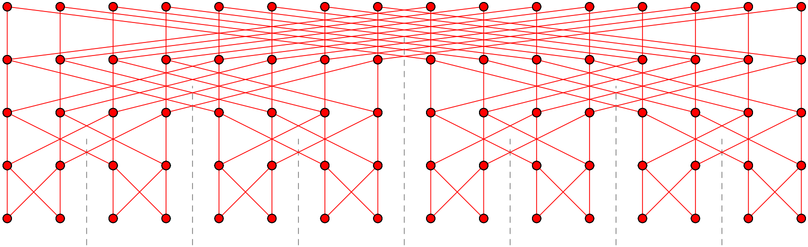


Figure: Butterflies. The black dots correspond to the $x_{s,i}$. The top row corresponding to $s = 0$. In this case $n = 16 = 2^4$.

For every step s we calculate $x_s = (x_{0,0}, \ldots, x_{0,n-1})$ from x_{s-1} (note $x_0 = (a_0, \ldots, a_{n-1})$).

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

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

for all $i \in \{0, \mathbf{2}, \ldots, n/m_s - 2\}$ and $j \in \{0, \ldots, m_s - 1\}$, where $m_s = 2^{p-s}$.

Theorem

For $i \in \{0, \ldots, n/m_s - 1\}$

$$x_{p,i} = \hat{a}_{[i]_p}$$

$$\hat{a}_i = x_{p,[i]_p}$$

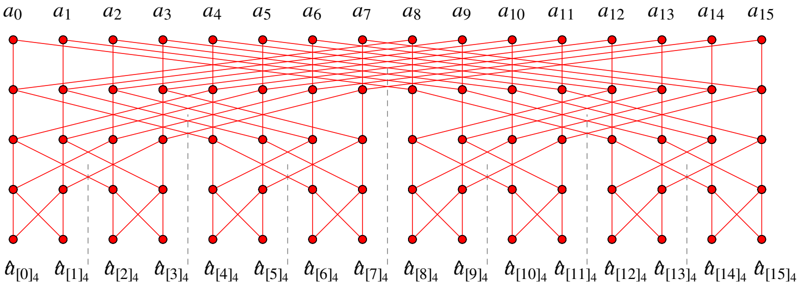


Figure: The “Regular” Fast Fourier Transform.

It is also straightforward to recover **a** from $\hat{\mathbf{a}}$

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

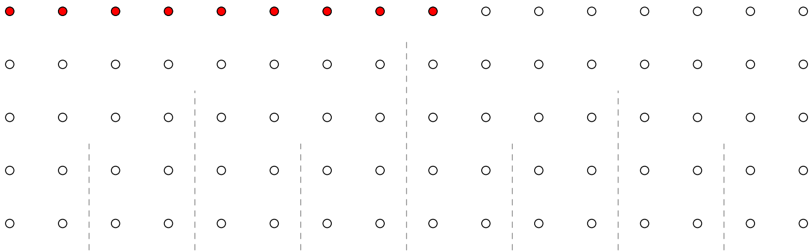
since

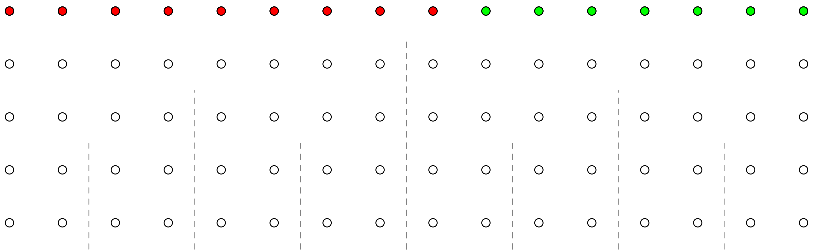
$$\sum_{j=0}^{n-1} \omega^{(i-k)j} = 0$$

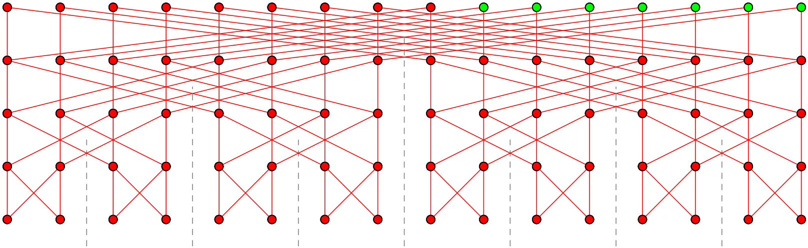
whenever $i \neq k$.
This yields a polynomial multiplication algorithm of time complexity $O(n \log n)$ in $\mathcal{R}[x]$.

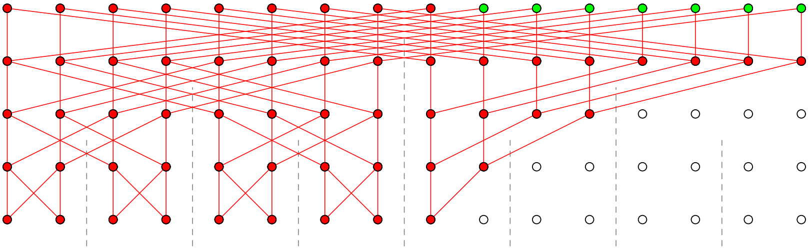
The Truncated Fast Fourier Transform

The algorithm for FFTrequires that n is a power of two. If we want to multiply two polynomials $A, B \in \mathcal{R}[x]$ such that $\deg(AB) + 1 = n + \delta$ we would need to carry out the FFT at precision $2n$.









The idea behind the TFTis to only do the work required to get the desired output (eliminating a factor of *2*).

Let $n = 2^p$, $\ell < n$ (usually $\ell \geq n/2$) and let ω be a primitive n-th root of unity.

$$\mathrm{TFT}_\omega\left(A = a_0 + \cdots + a_\ell x^{\ell-1}\right) \xrightarrow{\mathrm{TFT}} \left(A(\omega^{[0]_p}), \ldots, A(\omega^{[\ell-1]_p})\right).$$

We use the in-place algorithm from the previous section in order to compute the TFT. The claim is that this means many of the $x_{s,i}$ can be skipped. Indeed, at stage s, it suffices to compute the vector $(x_{s,0}, \ldots, x_{s,(\lfloor(\ell-1)/m_s\rfloor+1)m_s-1})$

Theorem

The TFTof an ℓ-tuple w.r.t. ω can be computed using at most $\ell p + n$ additions and $\lfloor (lp+n)/2 \rfloor$ multiplications with powers of ω.

The Inverse Truncated Fourier Transform

Unfortunately (quite unfortunately), the inverse TFTcannot be computed using a similar formula. Simply put, we are missing information and have to account for this.

We will use the fact that whenever one value among

$$x_{s,im_s+j}, x_{s-1,im_s+j}$$

and one value among

$$x_{s,(i+1)m_s+j}, x_{s-1,(i+1)m_s+j}$$

are known then we can deduce the other values.

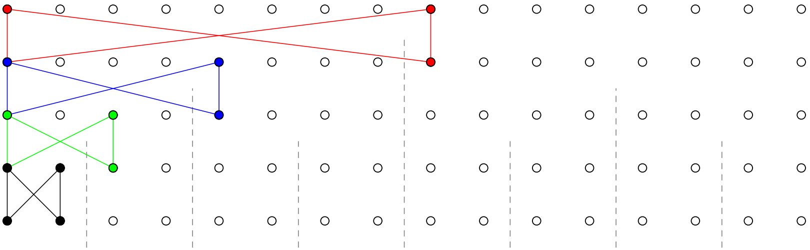


Figure: You can deduce any part of the butterfly from two known parts.

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

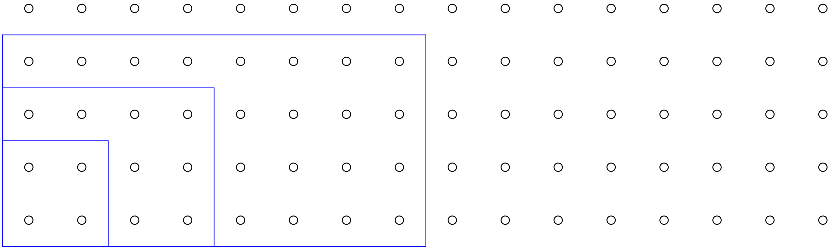


Figure: The computations in the boxes are self contained.

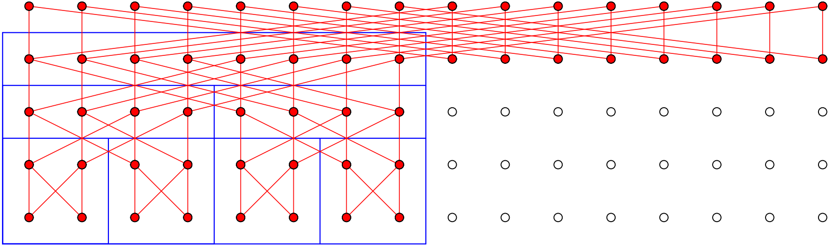
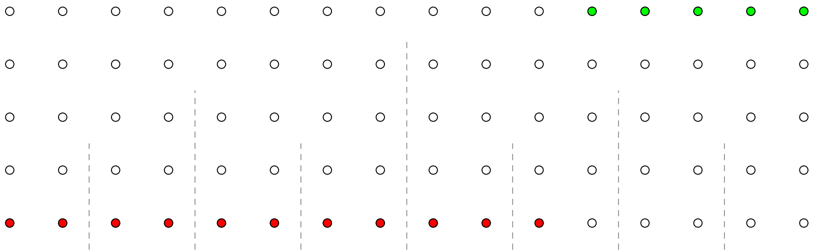
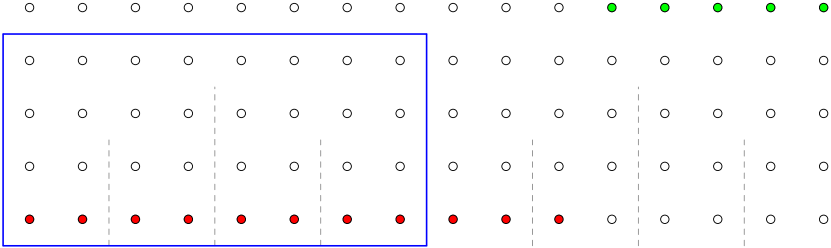
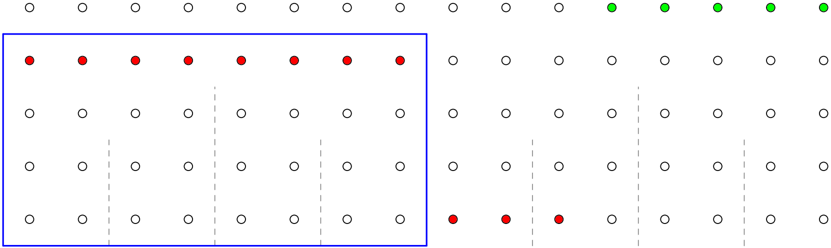
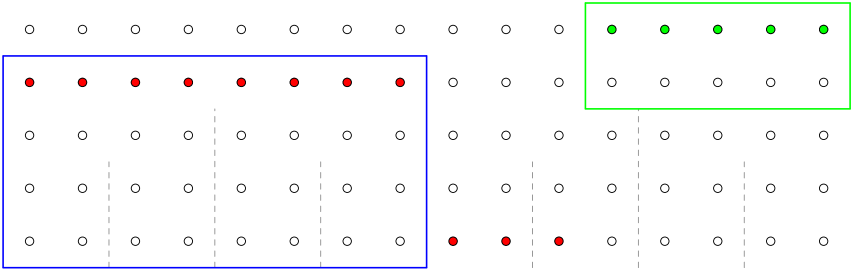


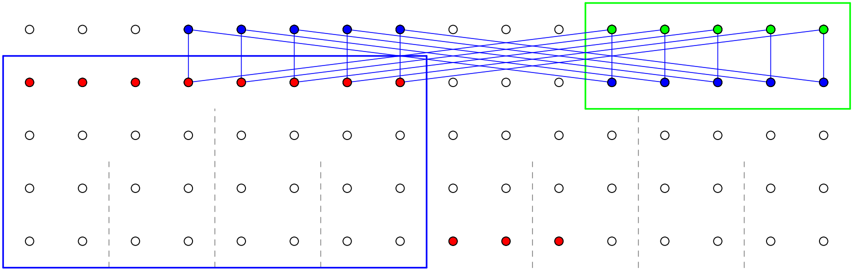
Figure: The computations in the boxes are self contained.

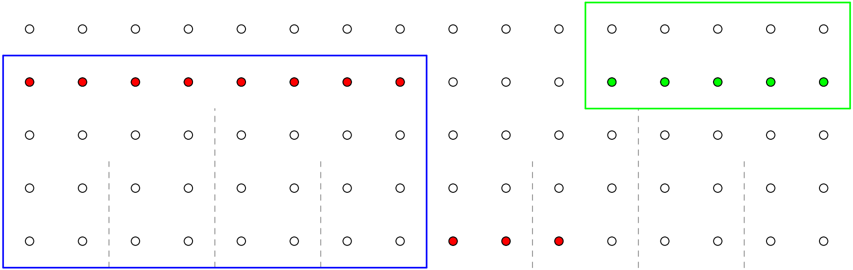


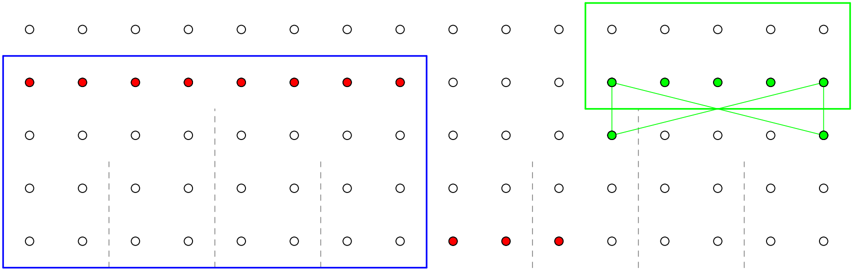


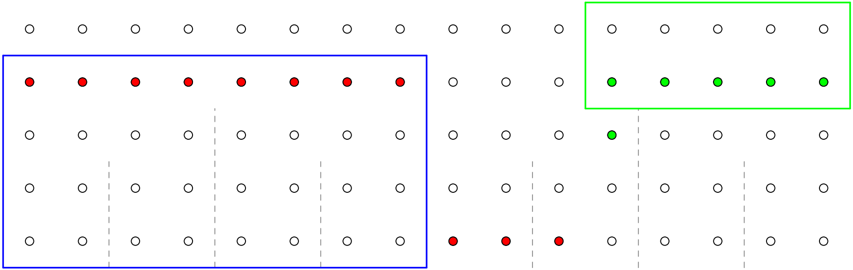


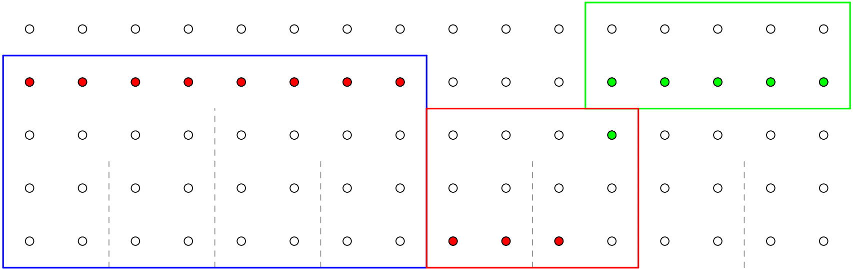


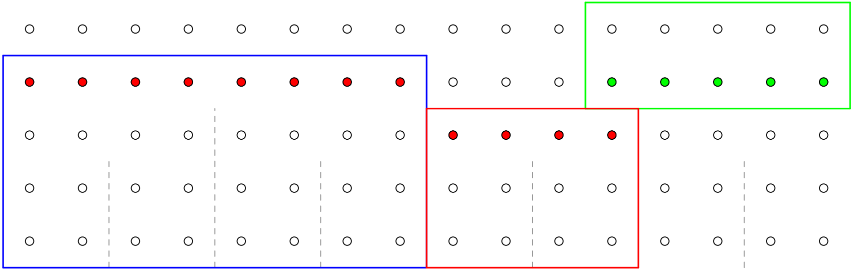


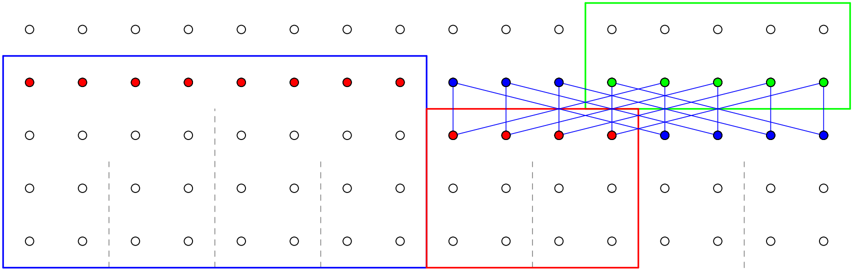


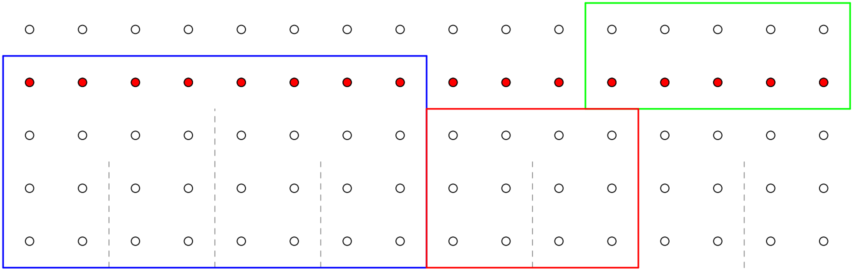


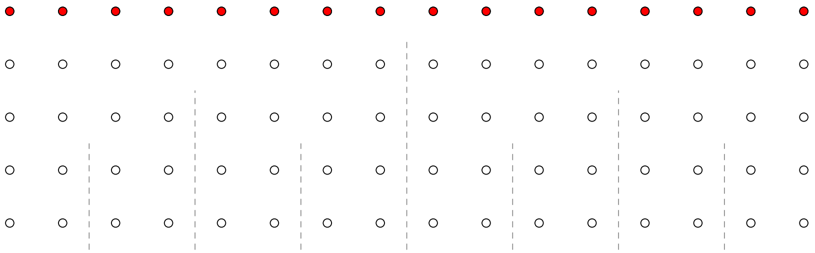












Theorem

The ℓ-tuple $(a_0, \ldots, a_{\ell-1})$ can be recovered from its TFTwith respect to ω using at most $\ell p + n$ shifted additions (or subtractions) and $\lfloor (\ell p + n)/2 \rfloor$ multiplications with powers of ω.

"Future Work"

- Paper for Eric.
- Finish my MAPLE implementation (almost there).

Done.