

Fast component-by-component construction of rank-1 lattice rules with a non-prime number of points[★]

Dirk Nuyens, Ronald Cools

*Dept. of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, 3001 Heverlee, Belgium*

Abstract

The component-by-component construction algorithm constructs the generating vector for a rank-1 lattice one component at a time by minimizing the worst-case error in each step. This algorithm can be formulated elegantly as a repeated matrix-vector product, where the matrix-vector product expresses the calculation of the worst-case error in that step. As was shown in an earlier paper, this matrix-vector product can be done in time $O(n \log(n))$ and with memory $O(n)$ when the number of points n is prime. Here we extend this result to general n to obtain a total construction cost of $O(sn \log(n))$ and memory of $O(n)$ for a rank-1 lattice in s dimensions with n points. We thus obtain the same big-Oh result as for n prime.

As was the case for n prime, the main calculation cost is significantly reduced by using fast Fourier transforms in the matrix-vector calculation. The number of fast Fourier transforms is dependent on the number of divisors of n and the number of prime factors of n . It is believed that the intrinsic structure present in rank-1 lattices and exploited by this fast construction method will deliver new insights in the applicability of these lattices.

Key words: Numerical integration, Quadrature and cubature formulas, Quasi-Monte Carlo, Rank-1 lattices, Fast component-by-component construction, Analysis of algorithms

1991 MSC: 65D30, 65D32, 68W40

[★] This research is part of a project financially supported by the Onderzoeksfonds K.U.Leuven / Research Fund K.U.Leuven

Email addresses: dirk.nuyens@cs.kuleuven.ac.be (Dirk Nuyens),
ronald.cools@cs.kuleuven.ac.be (Ronald Cools).

1 Introduction

The application of this paper is the approximation of an s -dimensional integral over the unit cube by an equal weight cubature rule,

$$I(f) = \int_{[0,1]^s} f(\mathbf{x}) \, d\mathbf{x} \approx Q(f) = \frac{1}{n} \sum_{\mathbf{x}_k \in P_n} f(\mathbf{x}_k), \quad (1)$$

where the n evaluation points are a rank-1 lattice

$$P_n = \left\{ \frac{k \cdot \mathbf{z}}{n} : 0 \leq k < n \right\}, \quad (2)$$

and the integer vector \mathbf{z} is called the generating vector of the lattice. By $k \cdot \mathbf{z}$ we mean (componentwise) multiplication modulo n . Both k and the components z_j are taken from \mathbb{Z}_n , the integers modulo n . Whenever we write $a \cdot b$ with $a, b \in \mathbb{Z}_n$ in the rest of this paper, we mean multiplication modulo n ; if ambiguity arises about the modulus we will write it explicitly. The components of \mathbf{z} are further restricted to the set

$$U_n = \{z \in \mathbb{Z}_n : \gcd(z, n) = 1\}.$$

This set contains the *units* of \mathbb{Z}_n and assures us that $(k \cdot z_j)/n$ is a permutation of the equispaced distribution k/n in each dimension j , where $k = 0, \dots, n-1$. This guarantees that the lattice has n distinct points in $[0, 1]^s$.

The component-by-component algorithm constructs this generating vector component by component, in each step minimizing a certain error measure for the cubature rule (1) and keeping all previously chosen components fixed. Here we minimize the worst-case error for all functions in the unit ball of a reproducing kernel Hilbert space $\mathcal{H}(K)$ with reproducing kernel K :

$$e(P_n, \mathcal{H}(K)) = \sup_{\substack{f \in \mathcal{H}(K) \\ \|f\| \leq 1}} |I(f) - Q(f)|.$$

The most important property of such a function space is the reproducing property of the kernel: $f(\mathbf{x}) = \langle f(\cdot), K(\mathbf{x}, \cdot) \rangle_{\mathcal{H}(K)}$, $\forall f \in \mathcal{H}(K), \forall \mathbf{x} \in [0, 1]^s$. In this paper we are however not so interested in the function space setting, but more in the technical realization of the component-by-component algorithm.

We take the same scenery as in [11], that is we use a shift-invariant tensor product reproducing kernel Hilbert space. Since the kernel is shift-invariant it is only a function of one argument $K(\mathbf{x}, \mathbf{y}) = K(\mathbf{x} - \mathbf{y}, \mathbf{0}) = K(\mathbf{x} - \mathbf{y})$, furthermore the s -dimensional kernel for a tensor product space is given by the product of the 1-dimensional kernels which build up the space. We thus

consider reproducing kernels of the form

$$K_{s,\gamma,\beta}(\mathbf{x}) = \prod_{j=1}^s K_{1,\gamma_j,\beta_j}(x_j), \quad K_{1,\gamma_j,\beta_j}(x_j) = \beta_j + \gamma_j \omega(x_j).$$

The specific form of the 1-dimensional kernels that we use here stems from the fact that these kernels are mostly studied in the Fourier domain, where the β_j is then just the *constant* frequency part and the γ_j is a weighting of the *variable* frequency part, i.e. $K_{1,\gamma_j,\beta_j}(x_j) = \beta_j + \sum_{h \in \mathbb{Z} \setminus \{0\}} \hat{k}_h \exp(2\pi i h x_j)$. In what follows we normalize the kernel such that $\beta_j = 1$ and leave out this subscript with no loss of generality (see the scaling in the Sobolev example at the end of this section).

In these so called weighted function spaces the γ -weights denote the relative importance of certain coordinates in the function space. The weights used here are product-type weights [15] where the γ_j are taken as a decaying sequence of positive weights,

$$\gamma_1 \geq \gamma_2 \geq \dots \geq \gamma_s \geq 0,$$

to denote that successive coordinates are less and less important and this assumes that the problem is (re)formulated in such a way. Under certain conditions on these weights, the lattice rules constructed by a component-by-component algorithm achieve the optimal rate of convergence [10], hereby justifying the usage of the algorithm.

The main advantage of using a reproducing kernel Hilbert space is the fact that an explicit expression for the worst-case error in such a function space exists. From [9] we know that for $\mathcal{H}(K)$ a shift-invariant reproducing kernel Hilbert space and P_n a lattice, the worst-case error can be written as

$$e(P_n, \mathcal{H}(K_{s,\gamma})) = \left(- \int_{[0,1]^s} K_{s,\gamma}(\mathbf{x}) \, d\mathbf{x} + \frac{1}{n} \sum_{\mathbf{x}_k \in P_n} K_{s,\gamma}(\mathbf{x}_k) \right)^{1/2}.$$

Under the assumption that $1 + \gamma_j \omega(x_j)$ is the normalized Fourier expansion of $K_{1,\gamma_j}(x_j)$ and using $K_{s,\gamma}(\mathbf{x}) = \prod_j K_{1,\gamma_j}(x_j)$, the integral above equals 1 (since the constant frequency part equals the integral of the function). We find that the worst-case error for n lattice points in s dimensions in such a function space can be written in terms of the generating vector \mathbf{z} as

$$e_{n,s}(z_1, z_2, \dots, z_s) = \left(-1 + \frac{1}{n} \sum_{k=0}^{n-1} p_{s-1}(k) \left(1 + \gamma_s \omega \left(\frac{k \cdot z_s}{n} \right) \right) \right)^{1/2}, \quad (3)$$

and the n -vector \mathbf{p}_{s-1} is recursively defined as

$$p_{s-1}(k) = p_{s-2}(k) \left(1 + \gamma_{s-1} \omega \left(\frac{k \cdot z_{s-1}}{n} \right) \right), \quad p_0(k) = 1. \quad (4)$$

The kernel which defines the shift-invariant function space can be taken arbitrary, see also [12]. To make this a little bit more concrete we take a look at the two most often used function spaces: the weighted Korobov space and the weighted Sobolev space. The reproducing kernel for a Korobov space with smoothness parameter $\alpha = 2$ is given by

$$K_{1,\gamma_j}(x_j) = 1 + \gamma_j 2\pi^2 B_2(x_j), \quad B_2(x_j) = x_j^2 - x_j + 1/6,$$

and is shift-invariant. The kernel of a weighted Sobolev space, which is a space of non-periodic functions, is a function of two parameters $K(\mathbf{x}, \mathbf{y})$ and the lattice point set takes the form

$$P_n = \left\{ \left(\frac{k \cdot \mathbf{z}}{n} + \mathbf{\Delta} \right) \bmod 1 : 0 \leq k < n \right\}.$$

These are so-called shifted lattice rules, with the added complication that a component-by-component construction now also has to search the optimal shifts $\Delta_j \in [0, 1)$. This kernel is not shift-invariant and thus formula (3) is not applicable. However with each such kernel we can associate a shift-invariant kernel by averaging over all possible shifts, see [9] and also [13]. The shift-invariant kernel then becomes

$$K_{1,\gamma_j}(x_j) = 1 + \frac{3\gamma_j}{3 + \gamma_j} B_2(x_j), \quad B_2(x_j) = x_j^2 - x_j + 1/6,$$

and the calculated error should now be scaled by $\prod_j (1 + \gamma_j/3)$. This has the added benefit that on application the shifts can be randomly chosen, since they were not fixed by the construction algorithm. Using independent replications this gives the possibility of obtaining a statistical error estimate in actual computations (this is mostly also done for the natural shift-invariant spaces). In these spaces and under appropriate conditions on the weights the component-by-component lattice rules achieve the optimal rate of convergence, see [10] for an analysis in Korobov and Sobolev spaces where these rates are $O(n^{-\alpha/2+\delta})$, $\alpha > 1$, and $O(n^{-1+\delta})$, $\delta > 0$.

In Section 2 we will introduce the component-by-component algorithm in its matrix-vector form. After introducing some necessary theory in Section 3 and Section 4, we will show in Section 5 how to obtain a fast, $O(sn \log(n))$, component-by-component construction algorithm by providing a fast matrix-vector multiplication for general n . In Section 6 we will illustrate the techniques which were introduced in the previous sections for some choices of n . We conclude the paper in Section 7.

2 A matrix-vector form of the component-by-component algorithm

Our goal is to select a generating vector \mathbf{z} which tries to minimize the worst-case error, defined in (3). The component-by-component algorithm (introduced in [14]) finds values for the components of the generating vector one component at a time, while keeping the previously made choices fixed. We start with z_1 to construct a 1-dimensional rule, then go on to find z_2 for a 2-dimensional rule and so on, in each step minimizing (3).

So in iteration s we have to calculate the worst-case error (3) for each possible candidate $z_s \in U_n$. Abstracting out the γ_s and the summation of the constant 1 in (3) gives a formula of the form

$$v_s(z) = \sum_{k=0}^{n-1} p_{s-1}(k) \omega\left(\frac{k \cdot z}{n}\right), \quad \forall z \in U_n,$$

which can easily be identified with a matrix-vector product (for all z at once):

$$\mathbf{v}_s = \mathbf{\Omega}_n \mathbf{p}_{s-1}, \quad \mathbf{\Omega}_n = \left[\omega\left(\frac{k \cdot z}{n}\right) \right]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}.$$

We have taken the liberty to define this matrix $\mathbf{\Omega}_n$ without specifying the iteration order of the elements $z \in U_n$ and $k \in \mathbb{Z}_n$ which define it. For now we take this order arbitrary, although for correctness the ordering of \mathbf{v}_s and \mathbf{p}_{s-1} must match those of U_n and \mathbb{Z}_n respectively, and this is silently assumed from now on.

After this matrix-vector product we search the minimum value of \mathbf{v}_s , i.e. $v_s(z^*)$, and choose the corresponding z candidate as the optimal choice for $z_s = z^*$. It can be seen easily that the minimum index z^* for \mathbf{v}_s is also the index for the worst-case error \mathbf{e}_s with minimum $e_s(z^*)$. Thus:

$$z_s = \operatorname{argmin}_{z \in U_n} e_s(z).$$

Once we know z_s we can overwrite the old \mathbf{p} -vector, \mathbf{p}_{s-1} , with the new \mathbf{p}_s using (4). This is in fact (almost) the same as multiplying each element in \mathbf{p}_{s-1} by the corresponding element of the row of $\mathbf{\Omega}_n$ which corresponds to our choice of z_s :

$$p_s(k) = (1 + \gamma_s \mathbf{\Omega}_n(z_s, k)) p_{s-1}(k), \quad \forall k \in \mathbb{Z}_n.$$

This can be written in matrix-vector lingo as a product with a diagonal matrix

$$\mathbf{p}_s = \operatorname{diag}(\mathbf{1}_{1 \times n} + \gamma_s \mathbf{\Omega}_n(z_s, \cdot)) \mathbf{p}_{s-1},$$

where $\Omega_n(z_s, :)$ means the z_s -th row of the matrix Ω_n .

This brings us to a short and concise formulation in Algorithm 1 of the component-by-component algorithm.

Algorithm 1 The component-by-component algorithm

```

p0 = 1
for  $s = 1$  to  $s_{\max}$  do
   $e_s^2 = -\mathbf{1}_{\phi(n) \times 1} + n^{-1} (\mathbf{1}_{\phi(n) \times (n-1)} + \gamma_s \Omega_n) \mathbf{p}_{s-1}$ 
   $z_s = \operatorname{argmin} e_s^2(z)$ 
   $\mathbf{p}_s = \operatorname{diag}(\mathbf{1}_{1 \times n} + \gamma_s \Omega_n(z_s, :)) \mathbf{p}_{s-1}$ 
end for

```

By simple inspection of the algorithm, we find that the major cost in constructing such a rank-1 lattice rule is concentrated in the matrix-vector multiplication. A general matrix-vector product has time complexity $O(n^2)$ for a matrix of order n , and so the obvious component-by-component construction of a rank-1 lattice with n points in s dimensions is $O(sn^2)$. A more precise construction cost can be derived by using the actual size of the matrix Ω_n , which is $|U_n| \times |\mathbb{Z}_n| = \phi(n) \times n$, and thus the construction cost is $O(s\phi(n)n)$, where ϕ is the Euler totient function. This means that the construction cost is $O(sn^{2-\delta})$ with $0 < \delta \leq 1/2$ when using a general matrix-vector product.

Since our matrix Ω_n has at most n different elements, and since in such a case it is often possible to do a matrix-vector product in $O(n \log(n))$ instead of $O(n^2)$, we could of course hope that component-by-component construction could be done in $O(sn \log(n))$. Such a technique was introduced in [11] for n prime.

3 Preliminaries

Define the index-matrix Ξ_n to represent the structure of Ω_n . This index-matrix has the same size as Ω_n where at position (z, k) we do not have the value of $\omega((k \cdot z)/n)$, but just the index $i = k \cdot z \bmod n$:

$$\Xi_n = \left[k \cdot z \bmod n \right]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}.$$

We can form the matrix Ω_n (assuming the same ordering of the index sets U_n and \mathbb{Z}_n) in a simple way from Ξ_n , by the application of the kernel function ω operating elementwise

$$\Omega_n = \omega(\Xi_n/n).$$

There is a matrix homomorphism from the matrix Ξ_n to the matrix Ω_n , i.e. the modulo multiplication structure present in Ξ_n is preserved in Ω_n . We will formalize this matrix homomorphism in the following definition where we use the term *codomain* to denote the set of entries in the matrix and the term *domain* to denote its index set.

Definition 1 (Matrix homomorphism). *A mapping φ from the codomain of \mathbf{A} onto the codomain of \mathbf{B} defines a matrix homomorphism if there exist mappings σ_r and σ_c from the domain of \mathbf{A} onto the domain of \mathbf{B} such that*

$$\forall (i, j) \in \text{domain}(\mathbf{A}) : \mathbf{A}(i, j) = t \Leftrightarrow \mathbf{B}(\sigma_r(i), \sigma_c(j)) = \varphi(t).$$

Similarly we can define a *matrix isomorphism* which states that two matrices are isomorphic when the mappings σ_r , σ_c and φ are one-to-one and onto. Note that these definitions are chosen in such a way that the Cayley tables of two isomorphic groups A and B are isomorphic matrices and vice versa. This will play an important role in what follows.

For completeness we will now give some basic abstract algebra results which we will use further on, and which can all be found in a standard algebra textbook, e.g. [7].

Definition 2 (Cyclic group). *A group G is called cyclic whenever all its elements can be generated by the powers of an element $g \in G$, called a generator, and thus $G = \langle g \rangle = \{g^k : k \in \mathbb{Z}\}$.*

Corollary 1 (Cyclicness of U_n). *The multiplicative group $U_n = \{z \in \mathbb{Z}_n : \gcd(z, n) = 1\}$, with order given by the Euler totient function as $|U_n| = \phi(n)$, is cyclic whenever*

$$n = 2, 4, p^k \text{ or } 2p^k,$$

with p an odd prime. A generator for the cyclic group U_n is called a primitive root modulo n .

An algorithm to find a primitive root modulo n can be found in [1]. If we have a generator g for the cyclic group U_n (with n given as in Corollary 1) then we can list the elements of U_n in natural order of this generator as

$$U_n = \langle g \rangle = \{g^0, g^1, g^2, \dots, g^{\phi(n)-1}\}.$$

We will now make a connection between the Cayley table of a cyclic group and a circulant matrix. For more about circulant matrices see e.g. [2].

Definition 3 (Circulant matrix). *A circulant matrix $\mathbf{C}_m = \text{circ}(\mathbf{c})$ of order*

m is a matrix defined by the m elements in the vector \mathbf{c} , indexed from 0, as

$$[\mathbf{C}_m]_{k,\ell} = c_{k-\ell \bmod m}.$$

In other words every diagonal of a circulant matrix consists of the same element and each column is a cyclic downshifted version from the previous column. Thus the first column is just the vector \mathbf{c} and the last row is this vector in reverse order.

The Cayley table of a cyclic group can be made to look like a circulant matrix. Consider a cyclic group G with a generator g . We can then picture the Cayley table as the left part in (5).

$$\begin{array}{c|ccccc} \cdot & g^0 & g^1 & g^2 & \cdots & g^{-1} \\ \hline g^0 & g^0 & g^1 & g^2 & \cdots & g^{-1} \\ g^1 & g^1 & g^2 & g^3 & \cdots & g^0 \\ g^2 & g^2 & g^3 & \cdots & \cdots & g^1 \\ \vdots & \vdots & \cdots & \cdots & \cdots & \vdots \\ g^{-1} & g^{-1} & g^0 & g^1 & \cdots & g^{-2} \end{array} \simeq \begin{array}{c|ccccc} \cdot & g^0 & g^{-1} & g^{-2} & \cdots & g^1 \\ \hline g^0 & g^0 & g^{-1} & g^{-2} & \cdots & g^1 \\ g^1 & g^1 & g^0 & g^{-1} & \cdots & \vdots \\ g^2 & g^2 & g^1 & \cdots & \cdots & g^{-2} \\ \vdots & \vdots & \cdots & \cdots & \cdots & g^{-1} \\ g^{-1} & g^{-1} & \cdots & g^2 & g^1 & g^0 \end{array} \quad (5)$$

By inspection we see that using the natural order of a generator results in constant anti-diagonals. By using the negative powers of the generator for the columns of the Cayley table, and keeping the positive powers of the generator for the rows, we obtain the table depicted at the right of (5). We now have constant diagonals and this table can be interpreted as a circulant matrix.

Theorem 1 (Diagonalization of a circulant matrix). *A circulant matrix has a similarity transform with the Fourier matrix as its eigenvectors*

$$\mathbf{C}_m = \mathbf{F}_m^{-1} \mathbf{D} \mathbf{F}_m,$$

and its eigenvalues are given by the discrete Fourier transform of its defining elements in the vector \mathbf{c}

$$\mathbf{D} = \text{diag}(\mathbf{F}_m \mathbf{c}).$$

Corollary 2 (Fast matrix-vector product with a circulant matrix). *A matrix-vector product with a circulant matrix \mathbf{C}_m takes time $O(m \log(m))$ instead of $O(m^2)$ when using its similarity transform and a fast Fourier algorithm:*

$$\begin{aligned} \mathbf{C}_m \mathbf{x} &= \mathbf{F}_m^{-1} \mathbf{D} \mathbf{F}_m \mathbf{x} \\ &= \text{IFFT}(\text{diag}(\text{FFT}(\mathbf{c})) \text{FFT}(\mathbf{x})). \end{aligned}$$

Note that a fast Fourier transform in time $O(m \log(m))$ is always possible when an m -point discrete Fourier transform is necessary. For such an implementation see e.g. [6].

4 Partitioning the index-matrix into circulant blocks

The technique which we will use for the fast construction will be based on block-partitioning the matrix Ξ_n into smaller matrices which are isomorphic to circulant matrices (and thus isomorphic to cyclic groups). The derived techniques will work for any matrix Ω_n as long as this matrix is homomorphic to Ξ_n (i.e. we will only use the multiplicative structure).

In the following we will represent $v \in \mathbb{Z}_n$ as numbers in a “residue number system” (e.g. [8, Section 4.7]), where we use the remainders of v with respect to moduli n_i that are prime to each other and $n = n_1 n_2 \cdots n_r$. The most natural way is to use the prime factorization of n , with the $n_i = p_i^{k_i}$, so

$$\begin{aligned} v &\simeq (v \bmod n_1, v \bmod n_2, \dots, v \bmod n_r) \\ &\simeq (v_1, v_2, \dots, v_r). \end{aligned}$$

The Chinese remainder theorem then tells us that this representation is unique (whenever the n_i are prime to each other) and we can thus map from v to (v_1, v_2, \dots, v_r) and back (an algorithm can be found in [1]). Note that in this representation the units take a special form:

$$u \simeq (u_1, u_2, \dots, u_r) \in U_n \Leftrightarrow u_i \in U_{n_i}.$$

4.1 Partitioning of U_n

First we partition U_n in terms of smaller cyclic groups.

Theorem 2 (Structure of U_n). *The multiplicative group U_n is isomorphic to the external direct product of groups U_{n_i} where $n = n_1 n_2 \cdots n_r$ is a factorization of n and the n_i are prime to each other*

$$U_n \simeq U_{n_1} \oplus U_{n_2} \oplus \cdots \oplus U_{n_r}.$$

Corollary 3. *Every group U_n can be written as*

$$U_n \simeq \begin{cases} U_{n_1} \oplus U_{n_2} \oplus \cdots \oplus U_{n_r}, & \text{if } 8 \nmid n, \\ (\langle 2^{k-1} - 1 \rangle_{2^k} \oplus \langle 5 \rangle_{2^k}) \oplus U_{n_2} \oplus \cdots \oplus U_{n_r}, & \text{if } 8 \mid n, n_1 = 2^k, k \geq 3, \end{cases}$$

where every subgroup is a cyclic multiplicative group.

Proof. A proof can be found in most abstract algebra books (e.g. [7, page 155]) mostly in the proximity of the fundamental theorem of Abelian groups, but since we need some details for prime factors of the form 2^k later on, we sketch the outline.

Consider the group U_n and a prime factorization of $n = n_1 n_2 \cdots n_r$, $n_i = p_i^{k_i}$. When $8 \nmid n_i$ the group U_{n_i} is already cyclic (see Corollary 1). We only need to consider the case where one of the prime factors, say n_1 , is 2^k with $k \geq 3$. For such a group we use the “pseudo generator” 5 which generates half of U_{2^k} and from which we can easily derive the other half. This gives

$$U_{2^k} = \{1 \cdot \langle 5 \rangle \bmod 2^k, (2^{k-1} - 1) \cdot \langle 5 \rangle \bmod 2^k\} \simeq \langle 2^{k-1} - 1 \rangle_{2^k} \oplus \langle 5 \rangle_{2^k}.$$

This comes from the well known isomorphism $U_{2^k} \simeq \mathbb{Z}_2 \oplus \mathbb{Z}_{2^{k-2}}$. We note that such a power of 2 thus makes an isomorphic copy of half of U_n . This proves the corollary. \square

So, given a group U_n we can find r smaller cyclic groups U_{n_i} of order $\phi(n_i)$ and generators g_i (we make abstraction of the special case for $8 \mid n_i$ since it would clutter the explanations following, however the reasoning still holds and an example follows in Section 6.2). We can then order the elements of $\oplus_i U_{n_i}$ in lexicographical order of the powers of these generators

$$\oplus_i U_{n_i} = \left\{ \begin{array}{l} (g_1^0, \dots, g_{r-1}^0, g_r^0), \dots, (g_1^0, \dots, g_{r-1}^0, g_r^{-1}), \\ (g_1^0, \dots, g_{r-1}^1, g_r^0), \dots, (g_1^0, \dots, g_{r-1}^1, g_r^{-1}), \\ \vdots \\ (g_1^{-1}, \dots, g_{r-1}^{-1}, g_r^0), \dots, (g_1^{-1}, \dots, g_{r-1}^{-1}, g_r^{-1}) \end{array} \right\},$$

as well as in order of the negative powers of these generators to build a Cayley table. As an example consider U_n , where $n = p_1 p_2$, a generator g_1 for U_{p_1} and g_2 for U_{p_2} , then the Cayley table can be made to look like:

·	$(g_1^0, \mathbf{g}_2^{-1})$	$(g_1^{-1}, \mathbf{g}_2^{-1})$	⋯	$(g_1^1, \mathbf{g}_2^{-1})$, $\mathbf{C}_2 = \left[\begin{array}{cccc} g_2^0 & g_2^{-1} & \cdots & g_2^1 \\ g_2^1 & g_2^0 & \cdots & g_2^2 \\ \vdots & \vdots & \ddots & \vdots \\ g_2^{-1} & g_2^{-2} & \cdots & g_2^0 \end{array} \right]$.
(g_1^0, \mathbf{g}_2)	(g_1^0, \mathbf{C}_2)	(g_1^{-1}, \mathbf{C}_2)	⋯	(g_1^1, \mathbf{C}_2)	
(g_1^1, \mathbf{g}_2)	(g_1^1, \mathbf{C}_2)	(g_1^0, \mathbf{C}_2)	⋯	(g_1^2, \mathbf{C}_2)	
⋮	⋮	⋮	⋱	⋮	
(g_1^{-1}, \mathbf{g}_2)	(g_1^{-1}, \mathbf{C}_2)	(g_1^{-2}, \mathbf{C}_2)	⋯	(g_1^0, \mathbf{C}_2)	

Here \mathbf{C}_2 is the circulant form of the Cayley table for U_{p_2} and the notation (g_1^k, \mathbf{C}_2) means that we have to combine g_1^k with every entry from \mathbf{C}_2 , likewise the notation (g_1^k, \mathbf{g}_2) means to combine every element from $\mathbf{g}_2 = [g_2^0, g_2^1, \dots, g_2^{-1}]$ with g_1^k and similarly $\mathbf{g}_2^{-1} = [g_2^0, g_2^{-1}, \dots, g_2^1]$. The complete Cayley table can now be seen as a block circulant matrix with circulant blocks.

This can obviously be extended to more than 2 factors and we would get that the Cayley table for r factors could be seen as an r times nested block circulant matrix. A matrix-vector product with such a (nested block) circulant matrix with r levels of nesting can be done in time $O(rn \log(n))$, see Lemma 3 in Section 5 later on in this paper.

4.2 Partitioning of \mathbb{Z}_n

We will now split \mathbb{Z}_n by considering a group action φ by the transformation group U_n on \mathbb{Z}_n . For completeness we will introduce the necessary definitions.

Definition 4 (Group action and orbit). *A group G acts on a set X by a group action $\varphi : G \times X \rightarrow X$ such that for every $x \in X$:*

- (1) $\varphi(e, x) = x$, where e is the identity element of G ,
- (2) $\varphi(g, \varphi(h, x)) = \varphi(gh, x)$ for all $g, h \in G$.

The orbit of $x \in X$ is defined as $\text{orb}(x) = \{\varphi(g, x) : g \in G\}$ and is the subset of X which can be reached by x under the transformations of G .

For our purpose the group $G = U_n$ acts on the set $X = \mathbb{Z}_n$, and the group action φ is multiplication modulo n . This is in fact a very natural view on the generation of the point set P_n as given in (2), where the components of the generating vector are selected from the units of \mathbb{Z}_n , i.e. U_n , to assure that $(k \cdot z_j)/n$ is a permutation of the equispaced distribution k/n in each dimension j , where $k = 0, \dots, n-1$.

We can now see every possible choice of z_s in the optimization process for dimension s as a possible permutation of the n 1-dimensional points. From algebra we know that a given action G on X defines an equivalence relation where the different orbits are the partitions. We will now show that the divisors of n are valid representers for these orbits on \mathbb{Z}_n .

Theorem 3 (Partitioning of \mathbb{Z}_n). *The union of all orbits generated by the divisors of n under U_n form a partition of \mathbb{Z}_n*

$$\bigcup_{d|n} \text{orb}(d) = \bigcup_{d|n} dU_n = \mathbb{Z}_n,$$

where the partition represented by d has size $|\text{orb}(d)| = \phi(n/d)$. Thus

$$n = \sum_{d|n} |\text{orb}(d)| = \sum_{d|n} |dU_n| = \sum_{d|n} \phi(n/d).$$

Proof. Note that since $n \equiv 0 \pmod{n}$ we conveniently use n for 0 , and vice versa, whichever is appropriate.

Observe that the orbit of d can be specified in different ways:

$$\begin{aligned} \text{orb}(d) &= \{d \cdot u : u \in U_n\}, && \text{the definition,} \\ &= dU_n, && \text{as a coset of } U_n, \\ &= \{v \in \mathbb{Z}_n : \gcd(v, n) = d\}, && \text{having a common gcd.} \end{aligned}$$

The last form is the most interesting for us (and follows directly from observing the prime powers). Since for all $v \in \mathbb{Z}_n$

$$\gcd(v, n) = d \in \text{divisors}(n),$$

this shows that the divisors are representers for the partitions.

Left to prove is $|\text{orb}(d)| = \phi(n/d)$. We will consider $v \in \mathbb{Z}_n$ in the residue number system with moduli given by the prime factorization of $n = \prod_i p_i^{k_i}$, so that

$$\text{orb}(d) = dU_n \simeq \oplus_i d_i U_{p_i^{k_i}}, \quad \text{with } d_i = d \pmod{p_i^{k_i}}.$$

As such the total number of elements in dU_n for a divisor $d = \prod_i p_i^{\ell_i}$, with $0 \leq \ell_i \leq k_i$, is the product of the number of elements in the cosets $d_i U_{p_i^{k_i}}$ (modulo $p_i^{k_i}$) and these are given by

$$|d_i U_{p_i^{k_i}}| = \phi(p_i^{k_i}/p_i^{\ell_i}), \quad \text{since } \gcd(d, p_i^{k_i}) = \gcd(d_i, p_i^{k_i}) = p_i^{\ell_i}.$$

It follows that

$$|\text{orb}(d)| = |dU_n| = \prod_i \phi(p_i^{k_i}/p_i^{\ell_i}) = \phi(n/d).$$

□

4.3 Block-partitioning of Ξ_n

If we combine Theorem 2 and Theorem 3 then we can partition Ξ_n in blocks which are isomorphic to the Cayley table of groups $U_{n/d}$. Let us first introduce the following lemma.

Lemma 1. *The following are equivalent for $n = \prod_i n_i$ and all n_i prime to each other and d a divisor of n :*

$$\begin{aligned} dU_n \bmod n &= dU_{n/d} \bmod n \\ &\simeq \bigoplus_i \left(d_i U_{n_i/g_i} \bmod n_i \right) \\ &\simeq \left(d \bigoplus_i U_{n_i/g_i} \right) \bmod n, \end{aligned}$$

with $d_i = d \bmod n_i$ and $g_i = \gcd(d_i, n_i)$, all with operation modulo n (or modulo n_i for the parts in the residue number system). Moreover

$$\begin{aligned} wU_n \bmod n &= wU_{n/g} \bmod n \\ w\mathbb{Z}_n \bmod n &= w\mathbb{Z}_{n/g} \bmod n, \quad \text{with } g = \gcd(w, n) \text{ and } w \in \mathbb{Z}. \end{aligned}$$

Proof. We will prove that $w\mathbb{Z}_n \bmod n = g\mathbb{Z}_{n/g} \bmod n$ (where w does not necessarily divide n , since that is trivial). The rest then follows easily. First set $g = \gcd(w, n)$ such that $\gcd(w/g, n/g) = 1$ and $g \mid n$. Consequently $w/g \in U_{n/g}$ and multiplication of the complete set $\mathbb{Z}_{n/g}$ with an element from $U_{n/g}$ returns the complete set $\mathbb{Z}_{n/g}$. We then find

$$\begin{aligned} w\mathbb{Z}_n \bmod n &= \{w \cdot v \bmod n : v \in \mathbb{Z}_n\} \\ &= \{g \cdot (w/g \cdot v) \bmod n : v \in \mathbb{Z}_n\} \\ &= \{g \cdot (w/g \cdot v \bmod n/g) \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= \{g \cdot v \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= g\mathbb{Z}_{n/g} \bmod n. \end{aligned}$$

Note that the g up front can be changed into a w by keeping w/g outside of the modulo n/g . \square

In the next theorem we use the symbol $\mathbf{1}_{t \times 1}$ to denote a vector of length t with all components equal to one and we use the symbol \otimes to denote the Kronecker tensor product. So $\mathbf{1}_{t \times 1} \otimes \mathbf{B}$ can be read as t replications of the matrix \mathbf{B} on top of each other. Wherever we write $dU_{n/d}$ in the next theorem, we mean modulo n as in the spirit of Lemma 1.

Theorem 4 (Block-partitioning of Ξ_n). *We can block-partition the matrix*

$$\Xi_n = \left[k \cdot z \right]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}$$

by considering the divisors of n (denoted as $d^{(1)}, d^{(2)}, \dots, d^{(\nu)}$, with $\nu = d(n)$ the number of divisors of n), into vertical partitions \mathbf{A}_d per divisor d ,

$$\Xi_n \simeq [\mathbf{A}_{d^{(1)}} | \mathbf{A}_{d^{(2)}} | \dots | \mathbf{A}_{d^{(\nu)}}], \quad \mathbf{A}_d = \left[k \cdot z \right]_{\substack{z \in U_n \\ k \in dU_{n/d}}},$$

of sizes $\phi(n) \times \phi(n/d)$. These partitions \mathbf{A}_d can each separately be horizontally partitioned into t_d identical square blocks \mathbf{B}_d for which

$$\mathbf{A}_d \simeq \begin{bmatrix} \mathbf{B}_d \\ \vdots \\ \mathbf{B}_d \end{bmatrix} = \mathbf{1}_{t_d \times 1} \otimes \mathbf{B}_d, \quad \mathbf{B}_d = \left[k \cdot z \right]_{\substack{z \in U_{n/d} \\ k \in dU_{n/d}}} = \left[d \cdot k \cdot z \right]_{\substack{z \in U_{n/d} \\ k \in U_{n/d}}},$$

of size $\phi(n/d) \times \phi(n/d)$, and so $t_d = \phi(n)/\phi(n/d)$. The blocks \mathbf{B}_d are isomorphic with the Cayley tables of $U_{n/d}$.

Proof. The vertical partitioning follows directly from Theorem 3 and the equality $dU_n \bmod n = dU_{n/d} \bmod n$ from Lemma 1.

We will now prove that these vertical partitions can be partitioned horizontally in $t_d = \phi(n)/\phi(n/d)$ identical square blocks which are isomorphic to the Cayley table of $U_{n/d}$. We will shift our problem to the residue number system, with moduli given by the prime factorization of $n = n_1 n_2 \cdots n_r$, $n_i = p_i^{k_i}$. From Lemma 1 we have that

$$dU_{n/d} \simeq \oplus_i d_i U_{n_i/g_i} \simeq d \oplus_i U_{n_i/g_i}, \quad \text{with } d_i = d \bmod n_i \text{ and } g_i = \gcd(d_i, n_i),$$

and it follows that

$$|d_i U_{n_i/g_i}| = |U_{n_i/g_i}| = \phi(n_i/g_i).$$

Now consider the matrix \mathbf{B}_{d_i} as a set B_{d_i} (i.e. the codomain of \mathbf{B}_{d_i}) and follow the same reasoning as in the proof of Lemma 1

$$\begin{aligned} B_{d_i} &= \{d_i \cdot k_i \cdot z_i \bmod n_i : z_i \in U_{n_i}, k_i \in U_{n_i/g_i}\} \\ &= d_i U_{n_i/g_i} \bmod n_i, \end{aligned}$$

which has size $|B_{d_i}| = \phi(n_i/g_i)$. We observe that the $\phi(n_i)$ elements of U_{n_i} can be partitioned in $t_{d,i} = \phi(n_i)/\phi(n_i/g_i)$ equivalence classes which have the elements of U_{n_i/g_i} as representers, i.e. B_{d_i} is isomorphic to U_{n_i/g_i} .

We thus have that the number of elements horizontally and vertically is both $\phi(n_i/g_i)$ and that there are only $\phi(n_i/g_i)$ distinct elements in \mathbf{B}_{d_i} , having a modulo multiplication structure. Using the Chinese remainder theorem, $B_d \simeq \oplus_i B_{d_i}$, it follows that we have $t_d = \phi(n)/\phi(n/d)$ copies and that \mathbf{B}_d is isomorphic to the Cayley table of $U_{n/d}$. \square

The previous theorem has not filled in the details of how exactly the rows and the columns of the matrix $\mathbf{\Xi}_n$ should be permuted. It only states that this is possible for each vertical partition \mathbf{A}_d (cf. the wording *each separately*).

The following corollary states that we can fix these two permutations for the complete matrix at once.

Corollary 4. *If we fix the ordering of the rows for all partitions \mathbf{A}_d and arrange the ordering of the columns so that $\mathbf{B}_1 \simeq \mathbf{C}_n$, where \mathbf{C}_n is the (nested block) circulant form of the Cayley table of the group U_n , then the interleaving of the (nested block) circulant matrices $\mathbf{B}_d \simeq d \mathbf{C}_{n/d}$ for a certain divisor d of n in \mathbf{A}_d is given by*

$$\mathbf{A}_d = (\otimes_i \mathbf{R}_{d,i}) d \mathbf{C}_{n/d},$$

where the i indices go over the prime factors of n (with the exceptional case for powers of 2 as given in Corollary 3) and the $\mathbf{R}_{d,i}$ matrices are defined as

$$\mathbf{R}_{d,i} = \mathbf{1}_{t_{d,i} \times 1} \otimes \mathbf{I}_{\phi(n_i/g_i)},$$

where $d_i = d \bmod n_i$, $g_i = \gcd(d_i, n_i)$, $t_{d,i} = \phi(n_i)/\phi(n_i/g_i)$ and $\mathbf{I}_{\phi(n_i/g_i)}$ is the identity matrix of order $\phi(n_i/g_i)$.

Proof. By simple calculation. □

This corollary gives a straightforward method to know where every element of a matrix \mathbf{B}_d arrives in its corresponding matrix \mathbf{A}_d . We just have to interpret what multiplication with a matrix $\mathbf{R}_d = \otimes_i \mathbf{R}_{d,i}$ means. This interpretation is quite natural since $dU_{n/d} = \oplus_i d_i U_{n_i/g_i}$. If we look at the basic case of multiplication with one matrix $\mathbf{R}_{d,i}$ we observe that the identity matrix has the effect of distributing every element of the group $d_i U_{n_i/g_i}$, and the replicating by $\mathbf{1}_{t_{d,i} \times 1}$ fills up the empty space left in \mathbf{C}_{n_i} when $g_i \neq 1$.

By using the formulation of Corollary 4 we will be able to distribute the results of matrix-vector multiplications with the smaller (nested block) circulant matrices \mathbf{B}_d to the final result vector of multiplication with the complete matrix.

5 Fast matrix-vector for general n

With the result from Theorem 4 it becomes trivial to show that a fast matrix-vector product with matrices homomorphic to Ξ_n is possible in time $O(n \log(n))$. In the proof of the next theorem we will use two lemmas which are deferred to the end of this section and contain some technical details.

Theorem 5 (Fast matrix-vector for Ξ_n structures). *A matrix-vector product with a matrix Ω_n which is homomorphic to Ξ_n can be done in time $O(n \log(n))$ and requires memory of order $O(n)$.*

Proof. Using Theorem 4 we can partition the matrix $\mathbf{\Omega}_n$ in vertical partitions \mathbf{A}_d which have (interleaved) copies of (nested block) circulant matrices \mathbf{B}_d . Since the matrix $\mathbf{\Omega}_n$ is homomorphic to $\mathbf{\Xi}_n$ the same permutations can be done and we can assume an arbitrary elementwise mapping function φ on $\mathbf{\Xi}_n$ to obtain $\mathbf{\Omega}_n$. For our specific purpose this mapping function is $\varphi(t) = \omega(t/n)$. We thus consider the matrix-vector product

$$\begin{aligned} \mathbf{v} &= \varphi([\mathbf{A}_{d(1)} | \mathbf{A}_{d(2)} | \cdots | \mathbf{A}_{d(\nu)}]) \begin{bmatrix} \mathbf{p}_{d(1)} \\ \mathbf{p}_{d(2)} \\ \vdots \\ \mathbf{p}_{d(\nu)} \end{bmatrix} \\ &= \varphi(\mathbf{A}_{d(1)}) \mathbf{p}_{d(1)} + \varphi(\mathbf{A}_{d(2)}) \mathbf{p}_{d(2)} + \cdots + \varphi(\mathbf{A}_{d(\nu)}) \mathbf{p}_{d(\nu)}, \end{aligned} \quad (6)$$

which can be considered as the sum of ν smaller matrix-vector products, where $\nu = d(n)$, the number of divisors of n . Using Theorem 4 it then suffices to calculate $\varphi(\mathbf{B}_d) \mathbf{p}_d$ instead of $\varphi(\mathbf{A}_d) \mathbf{p}_d$, since \mathbf{A}_d contains only stacked copies (in some order) of the same (nested block) circulant matrix \mathbf{B}_d .

The size of \mathbf{B}_d is $\phi(n/d) \times \phi(n/d)$ and it follows from Lemma 3 (forthcoming) that its matrix-vector product can be done in $O(\kappa(n/d) \phi(n/d) \log(\phi(n/d)))$, where $\kappa(n/d)$ is the number of unique factors of n/d . Summing of the $d(n)$ result vectors can be done in $O(\kappa(n) n)$ (see Lemma 2 following). The total cost for the complete matrix-vector product is then

$$O\left(\sum_{d|n} \kappa(n/d) \phi(n/d) \log(\phi(n/d)) + \kappa(n) n\right) = O(n \log(n)), \quad (7)$$

where we used $\sum_{d|n} \phi(d) = n$ and $\kappa(n)$ bounded by a constant.

The memory needed for this operation is $O(n)$ as will be explained in Lemma 3. \square

In (7) we actually assume $\kappa(n)$ to be bounded by a constant. For practical implementations this will always be the case and will be reasonable, e.g. $\kappa(n) \leq 9$ for all $n \leq 2^{32}$. However, we could as well use a crude bound like $\kappa(n) < \log_2(n)$ (the logarithm in base 2) from which it follows that the total complexity is always less than $O(n \log^2(n))$ (the logarithm to the power 2). This is however a serious overestimate. Also it is known that on the average $\kappa(n) \sim \log(\log(n))$, and for the worst choice, i.e. n a product of distinct primes, it is known that on the average $\kappa(n) \sim \frac{\log(n)}{\log(\log(n))}$.

The second part in the total complexity for multiplication with $\mathbf{\Omega}_n$ is the summing of the $d(n)$ result vectors. We could bound this naively by $d(n) \phi(n)$

and for prime n we have $d(p) = 2$ and $\phi(p) = p - 1$, and as such this cost is negligible compared with the $O(n \log(n))$ for the matrix-vector products. Also for prime powers this works out fine, since then $d(p^k) = k + 1 = O(\log(n))$ and $\phi(p^k) = p^{k-1}(p - 1) = O(n)$. However, for general n this does not look so good. For composite n we could bound $d(n)\phi(n)$ very crudely as

$$d(n)\phi(n) \leq 2\sqrt{n}(n - \sqrt{n}) = 2(n^{3/2} - n),$$

which will then dominate the cost over the $O(n \log(n))$ from the matrix-vector products. Summing like this will give a total complexity of $O(n^{3/2} - n)$ but is still asymptotically better than $O(n^{2-\delta})$ for doing the full matrix-vector product (without transforms) as shown in Section 2. Luckily it is possible to do a better summing job by carefully choosing the order of the divisors.

Lemma 2. *The summing of the $d(n)$ result vectors in (6) of the matrix-vector products $\mathbf{v}_d = \varphi(\mathbf{B}_d)\mathbf{p}_d$ with the (nested block) circulant matrices \mathbf{B}_d can be done in time $O(\kappa(n)n)$ by stepping through the powers of the divisors in lexicographical order.*

Proof. To achieve a good summing order we consider the divisors of n in lexicographical ordering of the powers of their prime components (so the divisors itself appear out of order). For $n = p_1^{k_1}p_2^{k_2}\dots p_r^{k_r}$, with $r = \kappa(n)$, we have divisors

$$d = p_1^{\ell_1}p_2^{\ell_2}\dots p_r^{\ell_r}, \quad \text{where } 0 \leq \ell_i \leq k_i.$$

Observe that two adjoining vertical partitions, one for a divisor $d = p_r^{\ell_r}\dots p_2^{\ell_2}p_1^{\ell_1}$ and the other for a divisor $d' = p_r^{\ell_r}\dots p_2^{\ell_2}p_1^{\ell_1+1}$, need

$$\phi\left(\frac{n}{\gcd(d, d')}\right) = \phi(n/d) = \phi(p_r^{k_r-\ell_r}\dots p_2^{k_2-\ell_2}p_1^{k_1-\ell_1})$$

operations to sum their results. This summing of the adjoining partitions for the first prime factor p_1 must be done for $\ell_1 = 0, \dots, k_1 - 1$ and this for all powers of the other prime factors. If we do this for increasing ℓ_1 then the formula above holds for all intermediate results and gives a partial cost for the first prime factor of

$$\begin{aligned} & \sum_{\ell_r=0}^{k_r} \dots \sum_{\ell_2=0}^{k_2} \sum_{\ell_1=0}^{k_1-1} \phi(p_r^{k_r-\ell_r}\dots p_2^{k_2-\ell_2}p_1^{k_1-\ell_1}) \\ &= \sum_{\ell_r=0}^{k_r} \phi(p_r^{k_r-\ell_r}) \dots \sum_{\ell_2=0}^{k_2} \phi(p_2^{k_2-\ell_2}) \sum_{\ell_1=0}^{k_1-1} \phi(p_1^{k_1-\ell_1}) \\ &= p_r^{k_r}\dots p_2^{k_2}(p_1^{k_1} - 1) = O(n). \end{aligned}$$

We now have result vectors of sizes $\phi(p_r^{k_r-\ell_r} \cdots p_2^{k_2-\ell_2} p_1^{k_1})$, and thus, from now on, the part from the first prime will always count for its full size $\phi(p_1^{k_1})$. Similarly, for the next prime factor we get

$$\begin{aligned} \sum_{\ell_r=0}^{k_r} \cdots \sum_{\ell_2=0}^{k_2-1} \phi(p_r^{k_r-\ell_r} \cdots p_2^{k_2-\ell_2} p_1^{k_1}) &= \sum_{\ell_r=0}^{k_r} \phi(p_r^{k_r-\ell_r}) \cdots \sum_{\ell_2=0}^{k_2-1} \phi(p_2^{k_2-\ell_2}) \phi(p_1^{k_1}) \\ &= p_r^{k_r} \cdots (p_2^{k_2} - 1) \phi(p_1^{k_1}) = O(n). \end{aligned}$$

And this continues up to the final prime factor, which has a cost of

$$\begin{aligned} \sum_{\ell_r=0}^{k_r-1} \phi(p_r^{k_r-\ell_r} \cdots p_2^{k_2} p_1^{k_1}) &= \sum_{\ell_r=0}^{k_r-1} \phi(p_r^{k_r-\ell_r}) \cdots \phi(p_2^{k_2}) \phi(p_1^{k_1}) \\ &= (p_r^{k_r} - 1) \cdots \phi(p_2^{k_2}) \phi(p_1^{k_1}) = O(n). \end{aligned}$$

Since we have $r = \kappa(n)$ levels (the number of unique prime factors) and the cost on each level is $O(n)$, the total complexity is $O(\kappa(n) n)$. \square

The last piece of the puzzle is fast matrix-vector multiplication with nested block circulant matrices of any nesting. We provide a method for such a fast matrix-vector multiplication in the following lemma.

Lemma 3. *A matrix-vector product with a nested block circulant matrix \mathbf{C} of order n (where the blocks are again block circulant, or circulant at the lowest level) can be done in time $O(kn \log(n))$ requiring memory $O(n)$, where k is the number of nested circulant levels.*

Proof. In Corollary 2 it was already shown that a matrix-vector product with a circulant matrix can be done in $O(n \log(n))$ using its eigenvalue decomposition which was given in Theorem 1.

Here we will show that any additional block circulant level can be made block diagonal, and furthermore completely diagonal by the use of a permutation and an additional sequence of FFT's (one per diagonal block).

Assume $\mathbf{C} = \text{circ}(\mathbf{C}^{(0:m-1)})$ is a block circulant matrix with m circulant blocks

of order n (i.e. there are 2 levels):

$$\mathbf{C} = \text{circ}(\mathbf{C}^{(0:m-1)}) = \begin{bmatrix} \mathbf{C}^{(0)} & \mathbf{C}^{(m-1)} & \mathbf{C}^{(m-2)} & \dots & \mathbf{C}^{(1)} \\ \mathbf{C}^{(1)} & \mathbf{C}^{(0)} & \mathbf{C}^{(m-1)} & \ddots & \vdots \\ \mathbf{C}^{(2)} & \mathbf{C}^{(1)} & \ddots & \ddots & \mathbf{C}^{(m-2)} \\ \vdots & \ddots & \ddots & \ddots & \mathbf{C}^{(m-1)} \\ \mathbf{C}^{(m-1)} & \dots & \mathbf{C}^{(2)} & \mathbf{C}^{(1)} & \mathbf{C}^{(0)} \end{bmatrix} \in \mathbb{R}^{nm \times nm},$$

and $\mathbf{C}^{(j)} \in \mathbb{R}^{n \times n}$. Then we can diagonalize the smaller circulant matrices by m n -point Fourier transforms on the first column of \mathbf{C} (i.e. on the first columns of the blocks $\mathbf{C}^{(j)} = \text{circ}(\mathbf{c}^{(j)})$). This gives the start of an analog to the diagonalization in Theorem 1 (by applying this same theorem m times):

$$\mathbf{C} = (\mathbf{I}_m \otimes \mathbf{F}_n^{-1}) \text{circ}(\tilde{\mathbf{C}}^{(0:m-1)}) (\mathbf{I}_m \otimes \mathbf{F}_n),$$

where $\tilde{\mathbf{C}}^{(j)} = \text{diag}(\mathbf{F}_n \mathbf{c}^{(j)}) = \text{diag}(\tilde{\mathbf{c}}^{(j)})$.

We observe that the elements in $\text{circ}(\tilde{\mathbf{C}}^{(0:m-1)})$ are laid out in such a way that we actually have n interleaved circulant matrices of order m , e.g. the first circulant matrix is defined by the first elements of the diagonals $\tilde{\mathbf{c}}^{(j)}$, $j = 0, \dots, m-1$, the second circulant matrix is defined by the second elements, etc. . . . The next step is to exchange the single circ-operation and the m diag-operations with one diag-operation and n circ-operations. This can be done by the following permutation (as can easily be verified by a small example)

$$\text{diag}(\tilde{\mathbf{C}}'^{(0:n-1)}) = \mathbf{P}_\sigma^T \text{circ}(\tilde{\mathbf{C}}^{(0:m-1)}) \mathbf{P}_\sigma,$$

where $\sigma(i) = (i \bmod m)n + \lfloor i/m \rfloor$ and $\tilde{\mathbf{C}}'^{(k)} = \text{circ}(\tilde{\mathbf{c}}_k^{(0:m-1)}) = \text{circ}(\tilde{\mathbf{c}}'^{(k)})$.

The diagonalization of the complete matrix \mathbf{C} can now be completed by applying n m -point Fourier transforms to these n circulant blocks. As such we find that

$$\mathbf{C} = (\mathbf{I}_m \otimes \mathbf{F}_n^{-1}) \mathbf{P}_\sigma (\mathbf{I}_n \otimes \mathbf{F}_m^{-1}) \tilde{\mathbf{C}}' (\mathbf{I}_n \otimes \mathbf{F}_m) \mathbf{P}_\sigma^T (\mathbf{I}_m \otimes \mathbf{F}_n),$$

with $\tilde{\mathbf{C}}' = \text{diag}(\mathbf{F}_m \tilde{\mathbf{c}}'^{(0:n-1)})$.

This can be reformulated in a computationally more efficient way when we consider a matrix $\mathbf{C} \in \mathbb{R}^{n \times m}$ defined as

$$\mathbf{C} = \left[\mathbf{c}^{(0)} \mid \mathbf{c}^{(1)} \mid \dots \mid \mathbf{c}^{(m-1)} \right] \quad \sim \quad \mathbf{C} = \text{circ}(\mathbf{C}^{(0:m-1)}),$$

so that

$$\begin{aligned}\tilde{\mathbf{C}} &= \mathbf{F}_n \mathbf{C} && \sim \text{circ}(\tilde{\mathbf{C}}^{(0:m-1)}), \\ \tilde{\mathbf{C}}' &= \tilde{\mathbf{C}}^T && \sim \text{diag}(\tilde{\mathbf{C}}'^{(0:n-1)}),\end{aligned}$$

and

$$\tilde{\tilde{\mathbf{C}}}' = \mathbf{F}_m \tilde{\mathbf{C}}' \quad \sim \quad \tilde{\tilde{\mathbf{C}}}' = \text{diag}(\mathbf{F}_m \tilde{\mathbf{C}}'^{(0:n-1)}).$$

A matrix-vector product with such a 2-level block circulant matrix has an analog to Corollary 2 as

$$\mathbf{C} \mathbf{x} = (\mathbf{I}_m \otimes \mathbf{F}_n^{-1}) \mathbf{P}_\sigma (\mathbf{I}_n \otimes \mathbf{F}_m^{-1}) \tilde{\tilde{\mathbf{C}}}' (\mathbf{I}_n \otimes \mathbf{F}_m) \mathbf{P}_\sigma^T (\mathbf{I}_m \otimes \mathbf{F}_n) \mathbf{x}.$$

From this follows that a matrix-vector product can be done in a fast way by considering \mathbf{x} as an $n \times m$ matrix, denoted as $\mathbf{x}_{n \times m}$, in column order, so that the product can be calculated efficiently as

$$(\mathbf{C} \mathbf{x})_{n \times m} = \mathbf{F}_n^{-1} (\mathbf{F}_m^{-1} (\tilde{\tilde{\mathbf{C}}}' \odot (\mathbf{F}_m (\mathbf{F}_n \mathbf{x}_{n \times m})^T)))^T,$$

where \odot means elementwise multiplication, e.g. $\text{diag}(\mathbf{d}) \mathbf{x} = \mathbf{d} \odot \mathbf{x}$. This calculation has a preprocessing cost of $O(nm \log(nm))$ and a cost of $O(2nm \log(nm))$ per matrix-vector product (ignoring the constants of the FFT's).

If we now consider such matrices \mathbf{C} to be embedded in another block circulant with ℓ blocks, and perform all the previous steps for each such matrix \mathbf{C} then we arrive at a block circulant with diagonal blocks (at cost $O(\ell nm \log(nm))$). Again we can permute this form to a block diagonal matrix with circulant blocks, perform nm Fourier transforms of size ℓ (at cost $O(nm\ell \log(\ell))$) and we again arrive at a diagonal matrix (at cost $O(nm\ell \log(nm\ell))$).

The matrix-vector product can be handled in exactly the same way as for the 2-level case, with one extra level of FFT's and permutations. The diagonal multiplication is $O(nm\ell)$ and thus the total cost here is as expected $O(nm\ell \log(nm\ell))$.

The total cost for a matrix of size n which has k levels of circulant embeddings thus is $O(kn \log(n))$. The memory needed is the memory needed to store \mathbf{C} and \mathbf{x} as well as their k -fold Fourier transforms, this is $O(n)$. \square

It must be noted that the number of embedded circulant matrices in a \mathbf{B}_d block is the number of distinct prime factors in n/d and thus $k = \kappa(n/d)$ and the size of such a \mathbf{B}_d block is $\phi(n/d)$ (again when $8 \mid n$ we have an exceptional case where we have one extra level due to the isomorphic copy effect).

6 Illustrative examples

We now provide some examples which show the complete track of the previous sections in action. We will start with the trivial case for prime n . Since powers of 2 are an exceptional application we present such an example which also illustrates the concepts for other prime powers. Finally we will consider an example of the more general case.

We present images of the matrix Ξ_n in Figures 1–3. Since in such a matrix we have values from 0 up to $n - 1$, there are n different colors per figure. On each row each color occurs only once, i.e. each row is a permutation of the first row of n colors. The matrices are organized as in Theorem 4 and so there is a vertical partition \mathbf{A}_d for each divisor d of n , grouped as given by Lemma 2. The start of these partitions is marked with an arrow which has a text label to denote which d generates this partition. In each vertical partition we have repetitive blocks \mathbf{B}_d , the repetitions of this block are drawn with faded colors to make clear how this block is distributed (cf. Corollary 4).

6.1 Prime n

Our previous result for prime n from [11] can now compactly be restated as follows. If n equals a prime p , the divisors are simply

$$\text{divisors}(p) = \{1, p\}.$$

We obtain the trivial partition $\mathbb{Z}_p = 1U_p \cup pU_1 = U_p \cup \{0\}$. The index-matrix has thus two very simple vertical partitions \mathbf{A}_1 and \mathbf{A}_p :

$$\begin{aligned} \mathbf{A}_1 &= \mathbf{1}_{1 \times 1} \otimes \mathbf{B}_1, & \mathbf{B}_1 &= [k \cdot z]_{z, k \in U_p}, \\ \mathbf{A}_p &= \mathbf{1}_{p-1 \times 1} \otimes \mathbf{B}_p, & \mathbf{B}_p &= [0], \end{aligned}$$

of which \mathbf{B}_1 is isomorphic to a circulant matrix of size $\phi(n)$.

An example of the structure for $p = 41$ is given in Figure 1. On the left we have drawn the matrix in its natural ordering, while on the right the matrix is drawn in the ordering which allows a fast matrix-vector product. The last partition is the partition for $d = 41$ where the complete column is constant; only the first element in this column is drawn at full color, the rest of the column is faded to denote its redundancy.

6.2 Powers of 2 (and other prime powers)

For powers of a prime the divisors are

$$\text{divisors}(p^k) = \{p^\ell : 0 \leq \ell \leq k\},$$

resulting in circulant matrices \mathbf{B}_d (being block circulant with 2 levels for a power of 2), which have regularly diminishing sizes

$$\begin{aligned} |B_{p^\ell}| &= \phi(p^{k-\ell}) \\ &= \begin{cases} p^{k-\ell-1}(p-1), & \text{if } \ell < k, \\ 1, & \text{otherwise.} \end{cases} \end{aligned}$$

Figure 2 shows Ξ_n for $n = 2^6 = 64$. In this figure it is clearly visible that increasing powers of a prime have the effect of overlapping in a nice way. It is this effect we are using to sum the result vectors. Assume we have calculated the 7 result vectors \mathbf{v}_d for $n = 64$, then the summing order from Lemma 2 gives

$$\begin{aligned} \mathbf{v} = & \mathbf{v}_1 + (\mathbf{v}_2 + (\mathbf{v}_4 + (\mathbf{v}_8 + (\mathbf{v}_{16} + (\mathbf{v}_{32} + \mathbf{v}_{64})))))) \\ & \underbrace{\hspace{10em}}_{+1 \text{ addition}} \\ & \underbrace{\hspace{8em}}_{+2 \text{ additions}} \\ & \underbrace{\hspace{6em}}_{+4 \text{ additions}} \\ & \underbrace{\hspace{4em}}_{+8 \text{ additions}} \\ & \underbrace{\hspace{2em}}_{+16 \text{ additions}} \\ & \underbrace{\hspace{1em}}_{+32 \text{ additions}} \end{aligned}$$

resulting in a total of $1 + 2 + 4 + 8 + 16 + 32 = 64 - 1 = 63$ which can be verified on the figure. Naive summing would give $6 \times 32 = 192$ operations.

Also visible in the figure is the isomorphic copy effect when a power of 2 is involved (mentioned in Corollary 3). This can be used nicely since the kernel function $\omega(x)$ is in most cases symmetric around $1/2$, i.e. $\omega(x) = \omega(1-x)$. As was shown in [11, Theorem 2], for prime n this symmetry has the effect that only half the space of z -candidates has to be searched and only one quarter of the $\mathbf{\Omega}_n$ matrix has to be considered. Similar effects occur for n which are not prime, for a power of 2 this means that the isomorphic copy can be left out, in other words, we get circulant matrices instead of block circulant matrices for free.

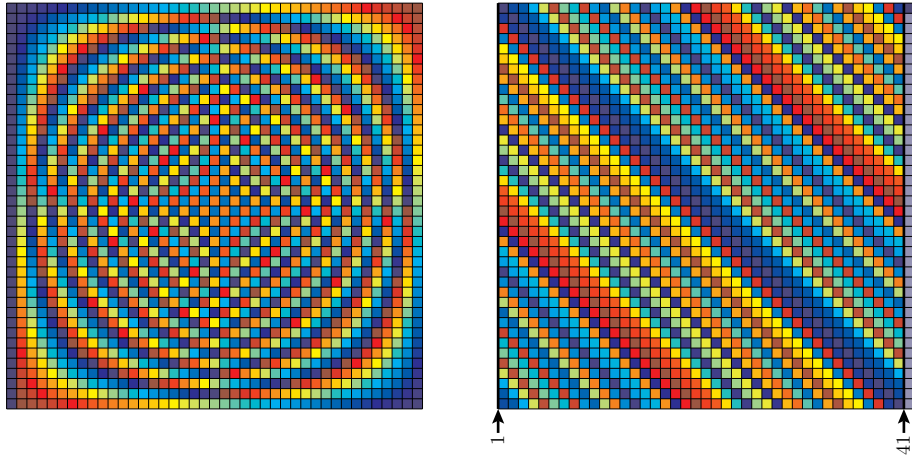


Fig. 1. Example matrix for $n = 41$, left: natural order, right: generator order

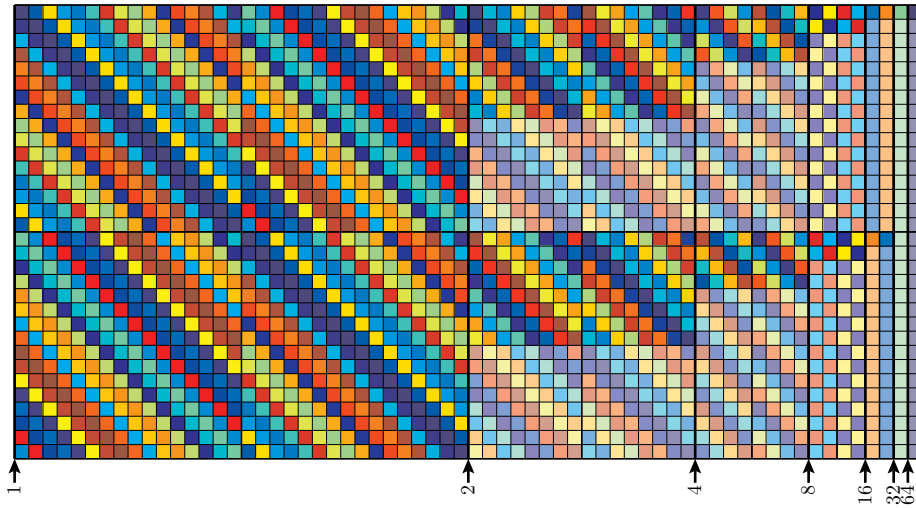


Fig. 2. Example matrix for $n = 2^6 = 64$

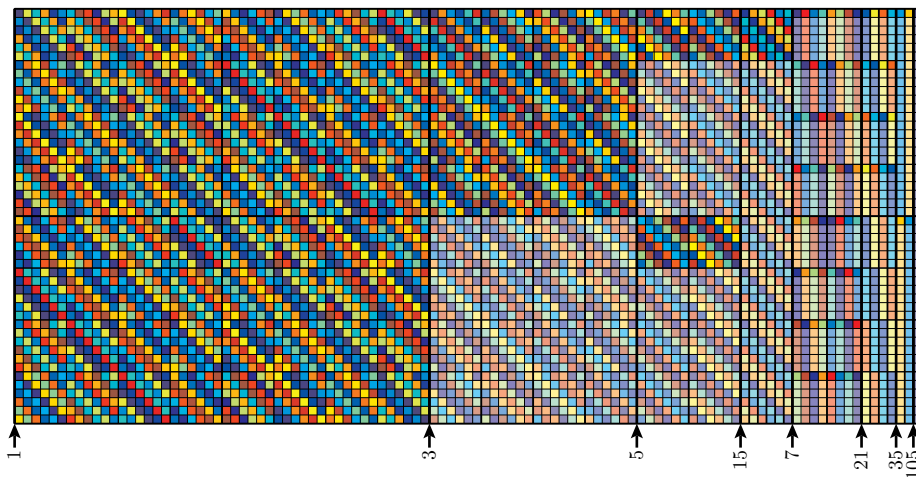


Fig. 3. Example matrix for $n = 3 \times 5 \times 7 = 105$

6.3 For general n

To get a better view of the interleaving of the matrices \mathbf{B}_d as given in Corollary 4 we must of course consider more general n . Figure 3 shows Ξ_n for $n = 3 \times 5 \times 7 = 105$. Clearly visible in the part for $d = 1$ is the nested block circulant structure with 3 levels. At the highest level we see a circulant structure with $\phi(3) = 2$ blocks of size $\phi(5)\phi(7) = 24$, in each such block we see again a circulant structure with $\phi(5) = 4$ blocks of size $\phi(7) = 6$, and these blocks in their turn are circulant matrices of order 6.

The summing as given by Lemma 2 here gives:

$$\mathbf{v} = \underbrace{(\mathbf{v}_1 + \mathbf{v}_3)}_{48 \text{ additions}} + \underbrace{(\mathbf{v}_5 + \mathbf{v}_{15})}_{+12 \text{ additions}} + \underbrace{(\mathbf{v}_7 + \mathbf{v}_{21})}_{+8 \text{ additions}} + \underbrace{(\mathbf{v}_{35} + \mathbf{v}_{105})}_{+2 \text{ additions}}$$

$$\underbrace{\hspace{10em}}_{+48 \text{ additions}} \quad \underbrace{\hspace{10em}}_{+8 \text{ additions}}$$

$$\underbrace{\hspace{20em}}_{+48 \text{ additions}}$$

which makes a total of $(48 + 12 + 8 + 2) + (48 + 8) + (48) = (7 \times 5 \times (3 - 1)) + (7(5 - 1)\phi(3)) + ((7 - 1)\phi(5)\phi(3)) = 174$ additions and can again be verified on the figure. Naive summing would involve $7 \times 48 = 336$ operations.

Finally, we give a smaller textual example for $n = 2 \times 3 \times 5 = 30$:

$$\text{divisors}(30) = \{1, 2, 3, 5, 6, 10, 15, 30\}$$

Or already put into the order of Lemma 2 we get $[1, 2, 3, 6, 5, 10, 15, 30] = [5^0 3^0 2^0, 5^0 3^0 2^1, 5^0 3^1 2^0, 5^0 3^1 2^1, 5^1 3^0 2^0, 5^1 3^0 2^1, 5^1 3^1 2^0, 5^1 3^1 2^1]$.

The biggest block is as always for $d = 1$ and contains here the elements of $U_{30} \simeq U_2 \oplus U_3 \oplus U_5 = \langle 1 \rangle_2 \oplus \langle 2 \rangle_3 \oplus \langle 2 \rangle_5$. With the help of the Chinese remainder theorem we find the correct ordering of the indices for the rows as

$$\begin{array}{ll} (1, 1, 1) \simeq 1, & (1, 2, 1) \simeq 11, \\ (1, 1, 2) \simeq 7, & (1, 2, 2) \simeq 17, \\ (1, 1, 4) \simeq 19, & (1, 2, 4) \simeq 29, \\ (1, 1, 3) \simeq 13, & (1, 2, 3) \simeq 23. \end{array}$$

The ordering of the columns can easily be found by reversing the sequences per prime component except for the first element of each sequence. Using this

generator ordering we can write $\mathbf{A}_1 = \mathbf{B}_1$ as

$$\begin{bmatrix} 1 & 13 & 19 & 7 & 11 & 23 & 29 & 17 \\ 7 & 1 & 13 & 19 & 17 & 11 & 23 & 29 \\ 19 & 7 & 1 & 13 & 29 & 17 & 11 & 23 \\ 13 & 19 & 7 & 1 & 23 & 29 & 17 & 11 \\ 11 & 23 & 29 & 17 & 1 & 13 & 19 & 7 \\ 17 & 11 & 23 & 29 & 7 & 1 & 13 & 19 \\ 29 & 17 & 11 & 23 & 19 & 7 & 1 & 13 \\ 23 & 29 & 17 & 11 & 13 & 19 & 7 & 1 \end{bmatrix}$$

Note that this is a block circulant with 2 levels. We skip the second divisor $d = 2$, which would give us a block as big as the first one since $\phi(2) = 1$. For $d = 3$ we obtain \mathbf{B}_3 as

$$\mathbf{B}_3 = [3 \cdot k \cdot z]_{\substack{z \in U_{10} \\ k \in U_{10}}} \simeq \begin{bmatrix} 3 & 9 & 27 & 21 \\ 21 & 3 & 9 & 27 \\ 27 & 21 & 3 & 9 \\ 9 & 27 & 21 & 3 \end{bmatrix}.$$

We now work out the permutation and replication matrix \mathbf{R}_3 by looking at its tensor product components for each prime factor (for clarity we index with the actual prime instead of with the index of the prime):

$$\mathbf{R}_3 = \otimes_{p=2,3,5} \mathbf{R}_{3,p},$$

where

$$\mathbf{R}_{3,2} = \begin{bmatrix} 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \end{bmatrix} = 1, \quad \mathbf{R}_{3,3} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \end{bmatrix} = \mathbf{1}_{2 \times 1}, \quad \mathbf{R}_{3,5} = \begin{bmatrix} 1 \end{bmatrix} \otimes \mathbf{I}_4 = \mathbf{I}_4,$$

and thus $\mathbf{R}_3 = \mathbf{1}_{2 \times 1} \otimes \mathbf{I}_4$ and we thus get two stacked copies of \mathbf{B}_3 on top of each other. As a final example let's look at $d = 10$. There we have

$$\mathbf{B}_{10} \simeq \begin{bmatrix} 10 & 20 \\ 20 & 10 \end{bmatrix},$$

and we find

$$\mathbf{R}_{10,2} = 1, \quad \mathbf{R}_{10,3} = \mathbf{I}_2, \quad \mathbf{R}_{10,5} = \mathbf{1}_{4 \times 1},$$

so that $\mathbf{R}_{10} = \mathbf{I}_2 \otimes \mathbf{1}_{4 \times 1}$. In other words replicate the first row of \mathbf{B}_{10} 4 times and then replicate the second row 4 times. A full view on the matrix is presented in Figure 4.

1	13	19	7	11	23	29	17	2	26	8	14	22	16	28	4	3	9	27	21	6	18	24	12	5	25	10	20	15	0
7	1	13	19	17	11	23	29	14	2	26	8	4	22	16	28	21	3	9	27	12	6	18	24	5	25	10	20	15	0
19	7	1	13	29	17	11	23	8	14	2	26	28	4	22	16	27	21	3	9	24	12	6	18	5	25	10	20	15	0
13	19	7	1	23	29	17	11	26	8	14	2	16	28	4	22	9	27	21	3	18	24	12	6	5	25	10	20	15	0
11	23	29	17	1	13	19	7	22	16	28	4	2	26	8	14	3	9	27	21	6	18	24	12	25	5	20	10	15	0
17	11	23	29	7	1	13	19	4	22	16	28	14	2	26	8	21	3	9	27	12	6	18	24	25	5	20	10	15	0
29	17	11	23	19	7	1	13	28	4	22	16	8	14	2	26	27	21	3	9	24	12	6	18	25	5	20	10	15	0
23	29	17	11	13	19	7	1	16	28	4	22	26	8	14	2	9	27	21	3	18	24	12	6	25	5	20	10	15	0

Fig. 4. Full view on the structure of \mathfrak{u}_{30} (italic font denotes redundancy)

7 Discussion

In [4] and [5], Dick & Kuo presented an adaptation of the component-by-component algorithm, called ‘Partial search’, by using low composite n having 2, 3, 4 and 5 factors. In their adaptation the worst-case error for each of the factors is calculated separately (by averaging over the components to be optimized) and then the optimal z_i are combined using the Chinese remainder theorem. From our analysis we expect that their calculated errors differ from the true worst-case error probably only in a small amount, since the divisors they left out are large and thus would only give small blocks \mathbf{B}_d .

Also from their papers [4,5], with verification in our previous paper [11], and also from [3], it is known that prime n have lower worst-case errors and thus are preferable over composite n . This clearly lowers the interest in implementing such a general n routine as presented in this paper. However, the prime power case is certainly interesting as this might give the user the opportunity to apply only part of the point set while still keeping a good distribution of the used points.

In our opinion, the main contribution of this paper is the revealing of the structure present in rank-1 lattice rules. We expect that it is possible to get some insights on the effect of combining two existing lattice rules into a new one. Also the ‘natural’ division of the matrix in blocks associated with the divisors of n could be useful. An important property that will probably be of use here is the possibility of using the Chinese remainder theorem to combine units of different groups whenever their n_i are prime to each other.

We mentioned in the examples section that the kernel function $\omega(x)$ is often symmetric. For computational efficiency an implementation should definitely use this fact because, although the algorithm is called fast, there is a huge difference in waiting 10 minutes for circa 10^8 points (a result from [11]) or waiting more than half an hour...

Only preliminary testing has been done for more general n , but from the results in this paper it should be clear that an implementation for prime n will probably be the most efficient (contradicting the previous results from [3–5] due to the fast construction). Especially n which have a large number of unique prime factors will slow the algorithm down because of the more complicated (nested block) circulant matrix-vector calculations involved. An example implementation in Matlab for prime n was given in [12] (as well as a connection with component-by-component construction of polynomial lattice rules).

We conclude that we presented a method to construct rank-1 lattice rules in a weighted, shift-invariant tensor product reproducing kernel Hilbert space

by using the component-by-component algorithm and the intrinsic structure present in this setting. The construction has time complexity $O(sn \log(n))$ for a rank-1 lattice rule with n points in s dimensions, for any n , and needs memory $O(n)$. The time complexity increases when n has more unique prime factors and when n has more divisors, but still is $O(sn \log(n))$.

Acknowledgements

The authors would like to thank the organizers of the Dagstuhl Seminar 04401 titled “Algorithms and Complexity for Continuous Problems” for such a stimulating environment to present this work. We would also like to thank Frances Kuo for renewing our attention to the powers of 2 case and for useful discussions about the content of this paper. This research is part of a project financially supported by the Onderzoeksfonds K.U.Leuven / Research Fund K.U.Leuven.

References

- [1] H. Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer-Verlag, 3rd edition, 1996.
- [2] P. J. Davis. *Circulant Matrices*. Wiley, 1979.
- [3] J. Dick. On the convergence rate of the component-by-component construction of good lattice rules. *J. Complexity*, 20(4):493–522, Aug. 2004.
- [4] J. Dick and F. Y. Kuo. Constructing good lattice rules with millions of points. In H. Niederreiter, editor, *Monte-Carlo and quasi-Monte Carlo Methods - 2002*, pages 181–197. Springer-Verlag, Jan. 2004.
- [5] J. Dick and F. Y. Kuo. Reducing the construction cost of the component-by-component construction of good lattice rules. *Math. Comp.*, 73:1967–1988, 2004.
- [6] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing, Vol. 3*, pages 1381–1384. IEEE, 1998.
- [7] J. A. Gallian. *Contemporary Abstract Algebra*. Houghton Mifflin, 4th edition, 1998.
- [8] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 2nd edition, 1994.

- [9] F. J. Hickernell. Lattice rules: How well do they measure up? In P. Hellekalek and G. Larcher, editors, *Random and Quasi-Random Point Sets*, pages 109–166. Springer-Verlag, 1998.
- [10] F. Y. Kuo. Component-by-component constructions achieve the optimal rate of convergence for multivariate integration in weighted Korobov and Sobolev spaces. *J. Complexity*, 19:301–320, June 2003.
- [11] D. Nuyens and R. Cools. Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. *Math. Comp.* To appear.
- [12] D. Nuyens and R. Cools. Fast component-by-component construction, a reprise for different kernels. In H. Niederreiter and D. Talay, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2004*, pages 371–385. Springer-Verlag. To appear.
- [13] I. H. Sloan, F. Y. Kuo, and S. Joe. Constructing randomly shifted lattice rules in weighted Sobolev spaces. *SIAM J. Numer. Anal.*, 40(5):1650–1665, 2002.
- [14] I. H. Sloan and A. V. Reztsov. Component-by-component construction of good lattice rules. *Math. Comp.*, 71(237):263–273, 2002.
- [15] I. H. Sloan and H. Woźniakowski. When are quasi-Monte Carlo algorithms efficient for high dimensional integrals. *J. Complexity*, 14(1):1–33, Mar. 1998.