Fast Fourier Transform and its applications to integer knapsack problems

Yu. Nesterov[*]

June 7, 2004

Abstract

In this paper we suggest a new efficient technique for solving integer knapsack problems. Our algorithms can be seen as application of Fast Fourier Transform to generating functions of integer polytopes. Using this approach, it is possible to count the number of boolean solutions of a single n-dimensional Diophantine equation $\langle a, x \rangle = b$ in $O(\|a\|_1 \ln \|a_1\| \ln n)$ operations. Another application example is an integer knapsack optimization problem of volume b, which can be solved in $O(\|a\|_1 \ln \|a_1\| \ln n + b \ln^2 n)$ operations of exact real arithmetics. These complexity estimates improve by a factor of n the complexity of the traditional Dynamic Programming technique.

Keywords: Integer programming, knapsack problem, Fast Fourier Transform, Dynamic Programming.

[*]Center for Operations Research and Econometrics (CORE), Catholic University of Louvain (UCL), 34 voie du Roman Pays, 1348 Louvain-la-Neuve, Belgium; e-mail: nesterov@core.ucl.ac.be.

This paper presents research results of the Belgian Program on Interuniversity Poles of Attraction initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. The scientific responsibility is assumed by the author.

1 Introduction

Motivation. Starting from a remarkable paper [2], algebraic methods become more and more popular in Combinatorial Optimization (see [1], [3], [5] and the references therein). However, the majority of the papers address mainly theoretical topics. To our knowledge, up to now no attempt has been made to implement these ideas in an algorithmic form and to compete with existing methods of Combinatorial Optimization. In this paper we show that this can be done for different knapsack-type problems. Moreover, we show that the new methods have better efficiency estimates than the standard Dynamic Programming approach.

In this paper we use an algebraic technique, which is similar to that of [5]. Let us illustrate it on a simple example. Denote by Z_+ the set of non-negative integers. Let $a = (a^{(1)}, \ldots, a^{(n)})^T \in Z_+^n$. Consider the boolean knapsack polytope

$$B_a^e(b) = \{x \in \{0,1\}^n : \langle a, x \rangle = b\}.$$

For a set $Q \subseteq R^n$, denote by $\mathcal{N}(Q)$ the number of integer points in Q ($\mathcal{N}(\emptyset) = 0$); we call this number the *integer volume* of the set Q. Our problem consists of computing the value $\mathcal{N}(B_a^e(b))$ for a given $b \in Z_+$.

Note that in general this computation is NP-hard (since it solves the "problem of stones"). On the other hand, consider the following generating function:

$$f(t) = \prod_{i=1}^{n} (1 + t^{a^{(i)}}), \quad t \in R.$$

We leave justification of the identity

$$f(t) \equiv \sum_{b=0}^{\|a\|_1} \mathcal{N}(B_a^e(b))\, t^b, \quad t \in R, \quad \|a\|_1 = \sum_{i=1}^{n} |a^{(i)}|,$$

as an exercise for the reader. Thus, we need to compute the coefficient for the term t^b of the polynomial $f(t)$. However, in doing that, it is reasonable to compute also all previous coefficients. Note that the direct computation of all these coefficients can take up to

$$O(n \, \|a\|_1)$$

arithmetic operations. The same complexity can be achieved by employing for above computation the standard Dynamic Programming approach (see, for example, Section II.6 [6]). But it is easy to see that the above problem can be solved in a better way.

Indeed, it is well known that the coefficients of the product of two polynomials of degree d can be computed by Fast Fourier Transform (FFT) in $O((d+1)\ln(d+2))$ arithmetic operations. Using this fact, we can make the following observation:

The coefficients of the product of n polynomials of degree d_i, $i = 1, \ldots, n$, can be computed by FFT in

$$O(d \;\ln d \;\ln n)$$

arithmetic operations, where $d = \sum\limits_{i=1}^{n} d_i$.

(For the sake of completeness we prove this statement as Lemma 2 in Section 2.) Using this result for the above polynomial $f(t)$ we get a complexity bound of the order of

$$O(\|a\|_1 \cdot \ln \|a\|_1 \cdot \ln n)$$

arithmetic operations. To our knowledge, for knapsack-type problems, this bound is the first one with such a weak dependence on the dimension of the space of variables. We will see later that similar complexity bounds can be obtained also for other knapsack-type problems including optimization problems.

Contents. The paper is organized as follows. We start from description of an efficient procedure for computing the integer volumes of knapsack polytopes (Section 2). The complexity of all the algorithms presented in this section and later on depends only in a logarithmic way on the number of items. In Section 3 we introduce the notion of characteristic functions of polytopes and discuss efficient algorithms for computing the values of these functions and their derivatives. Characteristic functions provide a bridge to knapsack optimization problems, which are discussed in the end of this section. Using our approach, an n-dimensional integer knapsack optimization problem of volume b can be solved in $O(\|a\|_1 \cdot \ln \|a\|_1 \cdot \ln n \; + \; b \cdot \ln^2 n)$ operations of exact real arithmetics.

Since all algorithms of this paper are based on FFT, we collect the necessary results in Appendix. Most of these facts can be found in classical monographs (e.g. [4]). However, usually FFT-type algorithms are given in an implicit form, from which it is difficult to see their actual performance. In our presentation of FFT-theory everything is explicit (the algorithms, the complexity results). Moreover, using this description, the majority of FFT-type methods can be implemented quite straightforwardly.

Notation. In what follows we denote by

$$\langle x, y \rangle = \sum_i x^{(i)} y^{(i)}$$

the standard inner product in the corresponding vector space. The dimension of the column vectors x and y is always clear from the context. For given $a, u \in Z^n_+$, denote by

$$B^u_a(b) = \left\{ x \in \prod_{i=1}^{n} \{0, \ldots, u^{(i)}\} : \; \langle a, x \rangle = b \right\}$$

the *bounded* knapsack polytope. Thus, $B^\infty_a(b)$ stands for a knapsack polytope with no upper bounds on the variables. For a polynomial $f(t)$, we denote its degree by $D(f)$. Notation e is used for the vector of all ones.

Acknowledgements. The author would like to thank Didier Henrion and Jean-Bernard Lasserre for interesting discussions, and Laurence Wolsey for careful reading and suggestions on the improvement of presentation of the results.

2 Computing the volume of knapsack polytopes

Consider a finite parametric family of discrete sets $\widehat{\mathcal{R}} \equiv \{\mathcal{R}(b)\}_{b \in Z_+}$ (we assume that $\mathcal{R}(b) = \emptyset$ for all b large enough). The *generating function* of this family is defined as follows:

$$f_{\widehat{\mathcal{R}}}(t) = \sum_{b=0}^{\infty} \mathcal{N}(\mathcal{R}(b)) \cdot t^b, \quad t \in R.$$

Let us fix now some a and u from Z_+^n. We can define the following parametric family of bounded knapsack polytopes $\mathcal{B}_a^u = \{B_a^u(b)\}_{b \in Z_+}$. Its generating function is then

$$f_{\mathcal{B}_a^u}(t) = \sum_{b=0}^{\infty} \mathcal{N}(B_a^u(b)) \cdot t^b, \quad t \in R. \tag{2.1}$$

Since u is finite, this function is a polynomial of degree $\langle a, u \rangle$. It appears that the generating function of the family $\mathcal{B}_a^u$ admits a very compact representation.

Lemma 1

$$f_{\mathcal{B}_a^u}(t) = \prod_{i=1}^{n} \left(\sum_{k=0}^{u^{(i)}} t^{k a^{(i)}} \right). \tag{2.2}$$

Proof:
We prove identity (2.2) by induction in dimension n. For $n = 1$, the zero-dimensional knapsack polytopes will have non-zero volume only for

$$b = 0,\, a^{(1)},\, 2a^{(1)},\, \dots,\, u^{(1)} a^{(1)}.$$

Since each of the corresponding volumes is equal to one, identity (2.2) follows.

Assume that (2.2) is proved for some $n \geq 1$. Let us prove it for dimension $n+1$. Let $a, u \in Z_+^n$. Denote

$$a_+ = (a, a^{(n+1)})^T \in Z_+^{n+1}, \quad u_+ = (u, u^{(n+1)})^T \in Z_+^{n+1}.$$

Note that for any $b \in Z_+$ we have

$$\mathcal{N}(B_{a_+}^{u_+}(b)) = \sum_{k=0}^{u^{(n+1)}} \mathcal{N}(B_u^a(b - k a^{(n+1)})).$$

Therefore, in view of the inductive assumption, we have

$$\begin{aligned} f_{\mathcal{B}_{a_+}^{u_+}}(t) &= \sum_{b=0}^{\infty} \mathcal{N}(B_{a_+}^{u_+}(b)) \cdot t^b \\ &= \sum_{b=0}^{\infty} \left(\sum_{k=0}^{u^{(n+1)}} \mathcal{N}(B_u^a(b - k a^{(n+1)})) \right) \cdot t^b \\ &= \sum_{b=0}^{\infty} \mathcal{N}(B_u^a(b)) \sum_{k=0}^{u^{(n+1)}} t^{b + k a^{(n+1)}} \\ &= f_{\mathcal{B}_a^u}(t) \cdot \left(\sum_{k=0}^{u^{(n+1)}} t^{k a^{(n+1)}} \right). \end{aligned}$$

□

Before we present the main result on the complexity of computation of the generating function, let us show how we can use FFT in order to compute efficiently the product of several polynomials.

Lemma 2 *Let polynomial $f(t)$ be represented as a product of several polynomials:*

$$f(t) = \prod_{i=1}^{n} p_i(t).$$

Then its coefficients can be computed by FFT in

$$O(D(f)\ \ln D(f)\ \ln n)$$

arithmetic operations.

Proof:
Let us increase the number of factors of $f(t)$ up to $\bar{n} = 2^m$,

$$\tfrac{1}{2}\bar{n} \le n \le \bar{n},$$

by adding the "virtual" unit factors $p_i(t) \equiv 1$, $n < i \le \bar{n}$, if necessary. Note that this does not change the degree of $f(t)$. Moreover, in the computations below we drop all multiplications involving these added polynomials; we need them only for a convenient description of the order of operations with non-trivial factors. Note that, by construction,

$$m \le 1 + \log_2 n. \tag{2.3}$$

Let us compute the product $f(t)$ in m stages. At the beginning of each stage we have $n_k = 2^{m-k}$ polynomials,

$$p_{k,i}(t), \quad i = 1,\ldots,n_k,\ k = 0,\ldots,m.$$

The polynomials of the next stage are obtained as a product of the neighbors implemented by FFT:

$$p_{k+1,i}(t) = p_{k,2i-1}(t)p_{k,2i}(t),\ i = 1,\ldots,n_{k+1}.$$

Note that the sum of degrees of all polynomials at each stage is equal to $D(f)$. On the other hand, the complexity of multiplication of two polynomials of degree d by FFT does not exceed $C \cdot (1+d)\log_2(2+d)$ arithmetic operations, where C is an absolute constant (see Lemma 4). Therefore, the total complexity of each stage can be estimated from above as following:

$$C \sum_{i=1}^{n_{k+1}} (1 + D(p_{k,2i-1}))\log_2(2 + D(p_{k,2i-1})))$$

$$\le C \sum_{i=1}^{n_k} (1 + D(p_{k,i}))\log_2(2 + D(p_{k,i})))$$

$$\le C \max_{\lambda \in R^{n_k}} \left\{ \sum_{i=1}^{n_k} (1+\lambda^{(i)})\ln(2+\lambda^{(i)}) :\ \langle e, \lambda\rangle = D(f),\ \lambda^{(i)} \ge -1,\ i = 1,\ldots,n_k \right\}$$

$$\le C \cdot ((1 + D(f))\ln(2 + D(f)) + n_k \ln 2.)$$

Note that $n_k \le \bar{n} \le 2n \le 2D(f)$. It remains to use (2.3). □

Thus, in view of representation (2.2), we can compute the volumes of bounded knapsack polytopes in a very efficient way. The following statement is a direct consequence of representation (2.2) and Lemma 2.

Theorem 1 *All $\langle a,u\rangle$ coefficients of the polynomial $f_{\mathcal{B}_a^u}(t)$ can be computed by FFT in*

$$O(\langle a,u\rangle \ \ln\langle a,u\rangle \ \ln n)$$

arithmetic operations. □

Consider now the generating function of the unconstrained knapsack polytopes:

$$f_{\mathcal{B}_a^\infty}(t) = \sum_{b=0}^{\infty} \mathcal{N}(B_a^\infty(b)) \cdot t^b. \tag{2.4}$$

In accordance with Lemma 1, we can represent this function in the following form:

$$f_{\mathcal{B}_a^\infty}(t) \equiv \prod_{i=1}^{n} \frac{1}{1-t^{a^{(i)}}}\ , \quad |t| < 1. \tag{2.5}$$

This representation makes the computation of the coefficients of the generating function much easier.

Theorem 2 *The coefficients of the polynomial $g(t) = \prod\limits_{i=1}^{n}(1-t^{a^{(i)}})$ can be computed by FFT in*

$$O(\|a\|_1 \ \ln\|a\|_1 \ \ln n) \tag{2.6}$$

arithmetic operations. Using these coefficients, the first $b+1$ coefficients of the generating function $f_{\mathcal{B}_a^\infty}(t)$ can be computed in

$$O(b \ \min\{\ln^2 b, \ln^2 n\})$$

arithmetic operations.

Proof:
These statements follow from Lemma 2, and Lemmas 4, 7 and 8 in the Appendix. □

3 Characteristic functions of knapsack polytopes

Let us fix a cost vector $c \in R^n$. Then for a given finite set of points $\mathcal{R} \subset R^n$ we can introduce the *characteristic function* of this set:

$$g_{\mathcal{R}}(c) = \sum_{x\in\mathcal{R}} e^{\langle c,x\rangle},$$

(compare with [7]). If $\mathcal{R} = \emptyset$, we set $g_{\mathcal{R}}(c) \equiv 0$. Note that for $\mathcal{R} = \mathcal{R}_1 \bigcup \mathcal{R}_2$ we have

$$g_{\mathcal{R}}(c) = g_{\mathcal{R}_1}(c) + g_{\mathcal{R}_2}(c).$$

Let us define also the *potential function* of the set $\mathcal{R}$:

$$\psi_{\mathcal{R}}(c) = \ln g_{\mathcal{R}}(c).$$

There is an important relation between the potential function and the *support function* of the set $\mathcal{R}$:

$$\xi_{\mathcal{R}}(c) \equiv \max_{x \in \mathcal{R}} \langle c, x \rangle \le \psi_{\mathcal{R}}(c) \le \xi_{\mathcal{R}}(c) + \ln \mathcal{N}(\mathcal{R}).$$

Therefore, the potential function can approximate the support function with an arbitrarily high accuracy:

$$\xi_{\mathcal{R}}(c) \le \mu \psi_{\mathcal{R}}(c/\mu) \le \xi_{\mathcal{R}}(c) + \mu \ln \mathcal{N}(\mathcal{R}), \quad \mu > 0. \qquad (3.1)$$

Further, for a parametric family of discrete sets $\widehat{\mathcal{R}} \equiv \{\mathcal{R}(b)\}_{b \in Z_+}$, we can define an *augmented* generating function:

$$F_{\widehat{\mathcal{R}}}(c,t) = \sum_{b=0}^{\infty} g_{\mathcal{R}(b)}(c) \cdot t^b, \quad t \in R.$$

Note that $F_{\widehat{\mathcal{R}}}(0,t) \equiv f_{\widehat{\mathcal{R}}}(t)$.

In particular, for the parametric family of bounded knapsack polytopes

$$\mathcal{B}_a^u = \{B_a^u(b)\}_{b \in Z_+},$$

the augmented generating function looks as follows:

$$F_{\mathcal{B}_a^u}(c,t) = \sum_{b=0}^{\infty} g_{B_a^u(b)}(c) \cdot t^b \equiv \sum_{b=0}^{\infty} \exp(\psi_{B_a^u(b)}(c)) \cdot t^b, \quad t \in R.$$

It appears that for the augmented generating function there still exists a simple expression.

Theorem 3

$$F_{\mathcal{B}_a^u}(c,t) = \prod_{i=1}^{n} \left(\sum_{k=0}^{u^{(i)}} e^{kc^{(i)}} t^{ka^{(i)}} \right). \qquad (3.2)$$

Proof:
We prove representation (3.2) by induction on n. For $n = 1$, the knapsack polytopes $B_a^u(b)$ are non-empty only for

$$b = ka, \quad k = 0, \dots, u.$$

Clearly, $B_a^u(ka) \equiv \{k\}$, $0 \le k \le u$, and (3.2) follows.

Assume now that the representation (3.2) is valid for some $n \ge 1$. Let us prove it for dimension $n+1$. Let $a, u \in Z_+^n$ and $c \in R^n$. Denote

$$a_+ = (a, a^{(n+1)})^T \in Z_+^{n+1}, \quad u_+ = (u, u^{(n+1)})^T \in Z_+^{n+1}, \quad c_+ = (c, c^{(n+1)})^T.$$

Note that for any $b \in Z_+$ we have

$$B_{a_+}^{u_+}(b) = \bigcup_{k=0}^{u^{(n+1)}} \{(x,k) \in Z^n \times Z : \; x \in B_u^a(b - ka^{(n+1)})\}.$$

Therefore

$$g_{B_{a_+}^{u_+}(b)}(c_+) = \sum_{k=0}^{u^{(n+1)}} g_{B_u^a(b-ka^{(n+1)})}(c) \cdot e^{kc^{(n+1)}}.$$

Thus,

$$\begin{aligned} F_{B_{a_+}^{u_+}}(c_+, t) &= \sum_{b=0}^{\infty} g_{B_{a_+}^{u_+}(b)}(c_+) \cdot t^b \\ &= \sum_{b=-\infty}^{\infty} \left(\sum_{k=0}^{u^{(n+1)}} g_{B_u^a(b-ka^{(n+1)})}(c) \cdot e^{kc^{(n+1)}} \right) \cdot t^b \end{aligned}$$

Let us change the order of summation in the last sum. Denote

$$b' = b - ka^{(n+1)}, \quad k' = k.$$

Note that for the lattice $Z^2 \equiv \{(b,k)\}$, this is a unitary change of basis. Moreover,

$$b = b' + k'a^{(n+1)}, \quad k = k'.$$

Therefore, in view of our inductive assumption, we conclude that

$$\begin{aligned} F_{B_{a_+}^{u_+}}(c, t) &= \sum_{b'} \sum_{k'=0}^{u^{(n+1)}} g_{B_u^a(b')}(c) \cdot e^{k'c^{(n+1)}} \cdot t^{b'+k'a^{(n+1)}} \\ &= \left(\sum_{b'} g_{B_u^a(b')}(c) \cdot t^{b'} \right) \cdot \left(\sum_{k'=0}^{u^{(n+1)}} e^{k'c^{(n+1)}} t^{k'a^{(n+1)}} \right) \\ &= F_{B_a^u}(c,t) \cdot \left(\sum_{k'=0}^{u^{(n+1)}} e^{k'c^{(n+1)}} t^{k'a^{(n+1)}} \right). \end{aligned}$$

□

Corollary 1 *The augmented generating function for the unconstrained knapsack polytope has the following form:*

$$F_{\mathcal{B}_a^\infty}(c,t) = \left[\prod_{i=1}^n (1 - e^{c^{(i)}} t^{a^{(i)}}) \right]^{-1}, \quad |t| < \min_{1 \le i \le n} e^{-c^{(i)}/a^{(i)}}. \tag{3.3}$$

□

Let us show how we can use the augmented characteristic functions in order to solve integer knapsack optimization problems. Consider the following problem:

$$\text{Find } f^* = \max_{x \in Z^n_+} \{ \langle c, x \rangle : \ \langle a, x \rangle = b \}, \tag{3.4}$$

where all coefficient are integers. In other words, we need to find

$$f^* = \xi_{B_a^\infty(b)}(c).$$

Since f^* is an integer value, it is enough to find its approximation with absolute accuracy less than one. Note that

$$\mathcal{N}(B_a^\infty(b)) \le \prod_{i=1}^n \left(1 + \frac{b}{a^{(i)}} \right) \le (1+b)^n.$$

Thus, if we take $\mu < \frac{1}{n} \ln(1+b)$, then in view of (3.1) we have

$$-1 + \mu \psi_{B_a^\infty(b)}(c/\mu) < f^* \le \mu \psi_{B_a^\infty(b)}(c/\mu).$$

Let us estimate the complexity of finding the coefficient

$$g_{B_a^\infty(b)}(c/\mu) = \exp\{ \psi_{B_a^\infty(b)}(c/\mu) \}.$$

This can be done in two steps:

$$\begin{array}{ll} \textbf{1.} & \text{Compute the coefficients of the polynomial } f(t) = \prod\limits_{i=1}^n (1 - e^{c^{(i)}/\mu} \cdot t^{a^{(i)}}). \\ \textbf{2.} & \text{Compute the first } b+1 \text{ coefficients of the rational function } g(t) = \frac{1}{f(t)}. \end{array} \tag{3.5}$$

In view of Lemma 2 and Lemma 4, the first step of the scheme takes $O(\|a\|_1 \ \ln \|a\|_1 \ \ln n)$ arithmetic operations. Further, in accordance with Lemmas 7 and 8, the second step takes at most $O(b \ \ln^2 n)$ operations. Thus, we have proved the following result.

Theorem 4 *The optimal value of problem (3.4) can be found by (3.5) in*

$$O(\|a\|_1 \cdot \ln \|a\|_1 \cdot \ln n \ + \ b \cdot \ln^2 n)$$

operations of exact real arithmetic.

References

[1] A.I.Barvinok and J.E.Pommersheim. An algorithmic theory of lattice points in polyhedra. In: “New perspectives in algebraic combinatorics”, *MSRI Publications*, 38 (1999), 91 - 147.

[2] M.Brion and M.Vergne. Residue formulae, vector partition functions and lattice points in rational polytopes. *J. Amer. Math, Soc.*, 10 (1997), 797 - 833.

[3] B.Chen. Lattice points, Dedekind sums and Ehrhart polynomials of lattice polyhedra. *Discrete Comput. Geom.* 28 (2002), 175 - 199.

[4] D. Knuth. *The Art of Computer Programming*, vol 2. Addison-Wesley, Reading, MA, 1973.

[5] J.B. Lasserre. Generating functions and duality for integer programs. To be published in *Discrete Optimization.*

[6] G.L. Nemhauser and L.A.Wolsey. *Integer and Combinatorial Optimization.* John Willey & Sons, 1988.

[7] Yu.Nesterov. Characteristic function of directed graphs and applications to stochastic equilibrium problems. CORE Discussion Paper # 2003/13, CORE, February 2003.

4 Appendix: Efficiency of Fast Fourier Transform

4.1 Basic representation

Recall that $j = \sqrt{-1} \equiv \exp(\frac{\pi}{2} j)$. We identify a complex number $z \in C$ with a point on two-dimensional real plane R^2:

$$z \equiv (\Re(z), \Im(z)) \quad \Leftrightarrow \quad z = \Re(z) + j\Im(z), \quad \bar{z} = \Re(z) - j\Im(z).$$

In estimating the complexity of the algorithmic schemes below, we always assume that the complex numbers are stored in the above coordinate form. Thus, the complex multiplication

$$z = z_1 \cdot z_2, \quad z_i \equiv (x_i, y_i) \in C,\ i = 1, 2,$$

defined by

$$\begin{aligned} \Re(z) &= x_1 x_2 - y_1 y_2 \\ \Im(z) &= x_1 y_2 + x_2 y_1 \\ &= (x_1 + y_1)(x_2 + y_2) - x_1 x_2 - y_1 y_2, \end{aligned} \tag{4.1}$$

needs three real multiplications and five real additions. In some situations, the value z_2 can be seen as a part of constant data. Then we do not need to count the addition $x_2 + y_2$; hence (4.1) takes only four additions. Note that such a reduced counting is always valid for algorithms computing *linear* functions of variables.

The majority of FFT-based algorithms can be derived from the spectral properties of one of the main combinatorial objects, the permutation matrix

$$P_n = \begin{pmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ & & \dots & & \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix} \in R^{n \times n}.$$

Note that $\det(\lambda I_n - P_n) = \lambda^n + (-1)(-1)^{n+1}(-1)^{n-1} = \lambda^n - 1$. Thus, P_n has the following eigenvalues:

$$\begin{aligned} \lambda_k(P_n) &= \lambda_n^k, \quad k = 0, \dots, n-1, \\ \lambda_n &= \exp(\tfrac{2\pi}{n}) = \cos(\tfrac{2\pi}{n}) + j \sin(\tfrac{2\pi}{n}). \end{aligned}$$

For $z \in C$ denote by $\pi_n(z)$ its vector of powers:

$$\pi(z) = (1, z, \dots, z^{n-1})^T \in C^n.$$

Then

$$\begin{aligned} P_n \cdot \pi(\lambda_n^{-k}) &= (\lambda_n^{-k(n-1)}, 1, \lambda_n^{-k}, \dots, \lambda_n^{-k(n-2)})^T \\ &= (\lambda_n^k, 1, \lambda_n^{-k}, \dots, \lambda_n^{k-k(n-1)})^T = \lambda_n^k \cdot \pi(\lambda_n^{-k}). \end{aligned}$$

Defining for $u, v \in C^n$ the *complex* inner product

$$\langle u, v\rangle_C = \sum_{i=i}^{n} \bar{u}^{(i)} v^{(i)},$$

we can see that

$$\langle \pi_n(\lambda_n^k), \pi_n(\lambda_n^m)\rangle_C = \sum_{i=0}^{n-1} \lambda_n^{(k-m)i} = \begin{cases} 0, & k \neq m, \\ n, & k = m. \end{cases}$$

For a complex matrix U, denote $U^* = \bar{U}^T$. Defining now

$$U_n = \frac{1}{\sqrt{n}}(\pi_n(1), \pi_n(\lambda_n^{-1}), \ldots, \pi_n(\lambda_n^{1-n})),$$

we can see that

$$P_n U_n = U_n \mathrm{diag}\,(\pi_n(\lambda_n)), \quad U_n^* U_n = I_n.$$

Hence, $U_n^{-1} = U_n^*$ and we get the following representation:

$$P_n = U_n \mathrm{diag}\,(\pi_n(\lambda_n)) U_n^*. \tag{4.2}$$

4.2 Multiplying matrices U_n and U_n^* by vectors

In FFT-based algorithms the key element is fast multiplication of the matrices U_n and U_n^* by a vector. Let us show how this can be done.

Let us fix a vector $p = (p^{(0)}, \ldots, p^{(n-1)})^T \in R^n$. Consider the vector

$$v = U_n^* p = \frac{1}{\sqrt{n}} \begin{pmatrix} \pi_n(1)^T p \\ \pi_n(\lambda_n)^T p \\ \ldots \\ \pi_n(\lambda_n^{n-1})^T p \end{pmatrix}.$$

Introducing the notation $p(z) = \sum\limits_{i=0}^{n-1} p^{(i)} z^i$, we can see that

$$v^{(i)} = p(\lambda_n^i)/\sqrt{n}, \quad i = 0, \ldots, n-1.$$

Denote by Λ_n the spectrum of the matrix P_n:

$$\Lambda_n = \{1, \lambda_n, \ldots, \lambda_n^{n-1}\}, \quad |\Lambda_n| = n.$$

Thus, our problem consists of computing all points from the set $p(\Lambda_n)$.

In order to do this efficiently, we use the following observation. Assume that $n = 2k$. Then the value of the polynomial p at any point z can be represented as follows:

$$\begin{aligned} p(z) &= \sum_{i=0}^{2k-1} p^{(i)} z^i = \sum_{i=0}^{k-1} p^{(2i)} z^{2i} + \sum_{i=0}^{k-1} p^{(2i+1)} z^{2i+1} \\ &= p_0(z^2) + z \cdot p_1(z^2), \\ p_0 &= (p^{(0)}, p^{(2)}, \ldots, p^{(2k-2)})^T \in R^k, \\ p_1 &= (p^{(1)}, p^{(3)}, \ldots, p^{(2k-1)})^T \in R^k. \end{aligned} \tag{4.3}$$

But for any $z \in \Lambda_n \equiv \Lambda_{2k}$ we have

$$z^2 = (\lambda_{2k}^i)^2 = (\lambda_{2k}^2)^i = \lambda_k^i \in \Lambda_k, \quad |\Lambda_k| = k.$$

Thus, in order to compute the values of the polynomial $p \in R^n$ in n points of the spectrum Λ_n we need to compute the values of two polynomials $p_0, p_1 \in R^k$ in $k = \frac{n}{2}$ points of the spectrum Λ_k and perform n additional complex multiplications. If $n = 2^m$, then this recursion can be repeated down to the unit dimension.

For a given $p \in C^N$ with $N = 2^m$, let us present an explicit algorithm for computing the vector

$$p(\Lambda_N) \equiv \sqrt{N}\, U_N^* p.$$

This algorithm generates m vectors $v_k \in C^N$, $k = 1, \dots, m$. Each of these vectors is used in order to store the values of $n_k = 2^{m-k}$ polynomials

$$p_{k,i}(z), \quad p_{k,i} \in C^{2^k}, \quad i = 1, \dots, 2^{m-k},$$

at the points of the spectrum Λ_{2^k}. The meaning of the entries of the vector v_k is as follows:

$$\begin{array}{rl} v_k = (& p_{k,1}(1), \dots, p_{k,n_k}(1), \\ & p_{k,1}(\lambda_{2^k}), \dots, p_{k,n_k}(\lambda_{2^k}), \\ & \dots, \\ & p_{k,1}(\lambda_{2^k}^{2^k-1}), \dots, p_{k,n_k}(\lambda_{2^k}^{2^k-1}) \quad). \end{array} \tag{4.4}$$

Note that in order to inverse the recursion (4.3), we need to compute the values

$$p_{k,i}(z) + z^{1/2} p_{k,l}(z), \quad p_{k,i}(z) - z^{1/2} p_{k,l}(z).$$

The structure (4.4) of the vector v_k ensures that these values are stored in the cells shifted one from another by $\hat{n} \equiv \frac{1}{2}N$. In the scheme below, notation $v[i]$ is used for ith cell of the vector v.

```
1. for $i = 1$ to $\hat{n}$ do { $v_1[i] = p^{(i-1)} + p^{(\hat{n}+i-1)}$, $v_1[\hat{n}+i] = p^{(i-1)} - p^{(\hat{n}+i-1)}$. }

2. for $k = 1$ to $m - 1$ do
   for $l = 1$ to $2^k$ do for $i = 1$ to $\frac{n_k}{2}$ do
   {
       $v_{k+1}\left[\frac{(l-1)n_k}{2} + i\right] = v_k[(l-1)n_k + i] + \lambda_{2^{k+1}}^{l-1} v_k[(l-1)n_k + \frac{n_k}{2} + i]$,   (4.5)
       $v_{k+1}\left[\hat{n} + \frac{(l-1)n_k}{2} + i\right] = v_k[(l-1)n_k + i] - \lambda_{2^{k+1}}^{l-1} v_k[(l-1)n_k + \frac{n_k}{2} + i]$.
   }
```

It is easy to implement the above scheme in such a way that the number of complex multiplications does not exceed

$$\sum_{k=1}^{m-1} 2^k \frac{n_k}{2} = \frac{m-1}{2} N = \frac{N}{2} \log_2 \frac{N}{2}. \tag{4.6}$$

In view of (4.1), one complex multiplication can be implemented using three real ones. Therefore, the above estimate leads to the following statement.

Lemma 3 *For integer $n \geq 2$, define $N = 2^m$ such that*

$$\tfrac{1}{2}N < n \leq N.$$

For $p \in C^n$ denote by $p_N \in C^N$ its extension by zero elements. Then the computation of the vector $U_N^* p_N$ by algorithm (4.5) needs at most

$$n \log_2 n \tag{4.7}$$

complex multiplications. In real arithmetic, this corresponds to

$$3n \log_2 n \tag{4.8}$$

real multiplications.

Let us look now at the problem of multiplying the matrix U_N by a vector. Note that

$$\sqrt{N}(U_N p)^{(k)} = \sum_{i=0}^{N-1} \lambda_N^{-(k-1)i} p^{(i)} = p(\lambda_N^{1-k}) = p(\lambda_N^{N+1-k}), \quad k = 1, \dots, N.$$

Thus, the product $U_N p$ can be obtained from the vector $U_N^* p$ by an appropriate permutation of the entries:

$$U_N p = Z_n U_N^* p, \quad Z_N = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & & 0 & 1 \\ 0 & 0 & 0 & & 1 & 0 \\ & \dots & & & & \dots \\ 0 & 0 & 1 & & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \end{pmatrix} \in R^{N \times N}.$$

Hence, its computational complexity is given also by (4.7).

4.3 Multiplication of two polynomials

Let us show that the spectral factorization (4.2) can be used to find the coefficients of the product of two polynomials with complex coefficients. Consider two polynomials $p, q \in C^n$. We need to compute the coefficients of the product

$$g(z) = p(z)q(z), \quad z \in C^n.$$

Note that $g \in C^{2n-1}$. Let us choose an arbitrary $N \geq 2n - 1$. Consider the following $N \times N$-matrix:

$$T_N(p) = \begin{pmatrix} p^{(0)} & 0 & 0 & \dots & p^{(2)} & p^{(1)} \\ p^{(1)} & p^{(0)} & 0 & & p^{(3)} & p^{(2)} \\ p^{(2)} & p^{(1)} & p^{(0)} & \dots & p^{(4)} & p^{(3)} \\ & \dots & & & \dots & \\ p^{(n-1)} & p^{(n-2)} & p^{(n-3)} & & 0 & 0 \\ 0 & p^{(n-1)} & p^{(n-2)} & & 0 & 0 \\ & \dots & & & & \dots \\ 0 & \dots & p^{(n-1)} & \dots & p^{(1)} & p^{(0)} \end{pmatrix} = \sum_{i=0}^{n-1} p^{(i)} P_N^i. \tag{4.9}$$

Denoting by a_N the natural extensions by zeros of vector a up to dimension N, we can see that

$$g_N = T_N(p) q_N.$$

Note that in view of representation (4.2),

$$\begin{aligned} T_N(p) &= \sum_{i=0}^{n-1} p^{(i)}\, U_N \,\mathrm{diag}\,(\pi_N(\lambda_N))^i\, U_N^* = U_N \left(\sum_{i=0}^{n-1} p^{(i)} \mathrm{diag}\,(\pi_N(\lambda_N))^i \right) U_N^* \\ &= U_N \,\mathrm{diag} \left(\sum_{i=0}^{n-1} p^{(i)} \pi_N(\lambda_N^i) \right) U_N^*. \end{aligned} \tag{4.10}$$

Denote $v = \sum_{i=0}^{n-1} p^{(i)} \pi_N(\lambda_N^i)$. It is clear that $v = U_N^* p_N$. Therefore, we can apply the following strategy for finding the coefficients of polynomial $g(z)$:

$$\begin{array}{ll} \mathbf{1}. & \text{Choose } N = 2^m \text{ such that } \tfrac{1}{2}N \le 2n-1 \le N. \\ \mathbf{2}. & \text{Compute vectors } v = U_N^* p_N \text{ and } w = U_N^* q_N \text{ by (4.5)}. \\ \mathbf{3}. & \text{Compute vector } u^{(i)} = v^{(i)} w^{(i)},\ i = 1, \dots, N. \\ \mathbf{4}. & \text{Compute } g_N = U_N u = Z_N U_N^* u \text{ by (4.5)}. \end{array} \tag{4.11}$$

A straightforward application of the estimate (4.6) leads to the following bound.

Lemma 4 *The complexity of computation of the vector g_N by the algorithm (4.11) does not exceed*

$$n \cdot (10 + 6 \log_2 n) \tag{4.12}$$

complex multiplications. However, if $n = 2^{m-1}$, then the algorithm needs

$$n \cdot (2 + 3 \log_2 n) \tag{4.13}$$

complex multiplications.

Proof:
Indeed, in view of (4.6), Step 2 takes $N \log_2 \frac{N}{2}$ complex multiplications, Step 3 takes N, and Step 4 takes $\frac{N}{2} \log_2 \frac{N}{2}$ multiplications. This gives

$$N(1 + \tfrac{3}{2} \log_2 \tfrac{N}{2})$$

complex multiplications. By the choice of N in (4.11), we always have $N \le 4n$. However, if $n = 2^{m-1}$, then $n = \frac{1}{2}N$. □

Note that the algorithm (4.11) can be used also for multiplying the polynomials with real coefficients. However, in this case the application of the complex machinery (4.5) looks quite artificial. Let us show that for real polynomials there exist a more efficient scheme.

Consider two real polynomials $p, q \in R^n$ with $n = 2k$. AS in (4.3), we can represent them as follows:

$$p(t) = p_0(t^2) + tp_1(t^2), \quad q(t) = q_0(t^2) + tq_1(t^2), \quad t \in R.$$

Let us form the coefficients $\hat{p} = p_0 + jp_1 \in C^k$ and $\hat{q} = q_0 + jq_1 \in C^k$. Then

$$\begin{aligned} p(t)q(t) &= p_0(t^2)q_0(t^2) + t \cdot (p_0(t^2)q_1(t^2) + p_1(t^2)q_0(t^2)) + t^2 p_1(t^2)q_1(t^2) \\ &= [p_0(t^2)q_0(t^2) - p_1(t^2)q_1(t^2)] + t \cdot [p_0(t^2)q_1(t^2) + p_1(t^2)q_0(t^2)] \\ &\quad +(1+t^2)p_1(t^2)q_1(t^2) \\ &= \Re(\hat{p}(t^2)\hat{q}(t^2)) + t \cdot \Im(\hat{p}(t^2)\hat{q}(t^2)) + (1+t^2)p_1(t^2)q_1(t^2) \\ &= \Re(\hat{p} \cdot \hat{q})(t^2) + t \cdot \Im(\hat{p} \cdot \hat{q})(t^2) + (1+t^2)p_1(t^2)q_1(t^2), \end{aligned}$$

where $\hat{p} \cdot \hat{q}$ denotes the complex coefficients of the polynomial $\hat{p}(z)\hat{q}(z)$. From this representation we get the following recurrence:

The coefficients of the product of two real polynomials of degree $n-1$ with $n = 2k$ can be found from the coefficients of the product of two complex polynomials of degree $k-1$ and the coefficients of the product of two real polynomials of degree $k-1$.

(4.14)

Let us estimate the complexity of this strategy. Assume we need to multiply two real polynomials $p, q \in R^n$. Let us choose m:

$$2^{m-1} < n \le 2^m \equiv N_1.$$

In accordance with (4.14), we need to multiply two complex polynomials of dimension $n_1 = \frac{1}{2}N_1$ and two real polynomials of the same dimension. Since n_1 is a power of two, in view of (4.13), the complexity of the first stage is equal to

$$M_1 = n_1 \cdot (2 + 3\log_2 n_1).$$

In the second stage, we need to multiply two real polynomials of dimension $N_2 = n_1 = \frac{1}{2}N_1$, etc. Thus, we have

$$N_k = 2^{m-k+1}, \quad n_k = 2^{m-k}, \quad M_k = n_k \cdot (2 + 3\log_2 n_k), \quad k = 1, \dots m.$$

Note that

$$\begin{aligned} \sum_{k=1}^{m} n_k \cdot (2 + 3\log_2 n_k) &= \sum_{k=1}^{m} 2^{m-k} \cdot (2 + 3(m-k)) \\ &= \sum_{k=0}^{m-1} 2^k \cdot (2 + 3k) \le 4 + (3m-4)2^m. \end{aligned}$$

Since $2^m < 2n$, we come to the following statement.

Lemma 5 *Two real polynomials of size $n \geq 4$ can be multiplied by the rule (4.14) in*

$$6n \log_2 n \tag{4.15}$$

complex multiplications. In real arithmetic, this corresponds to $18n \log_2 n$ multiplications.

Note that the estimate $18n \log_2 n$ becomes smaller than n^2 for $n > 125$.

4.4 Multiplication of a Toeplits matrix by a vector

Let $n = 2k$. For $p = (p^{(0)}, \ldots, p^{(n-1)})^T \in C^n$, consider the following $k \times k$ Toeplits matrix:

$$B_k(p) = \begin{pmatrix} p^{(k)} & p^{(k-1)} & \ldots & p^{(2)} & p^{(1)} \\ p^{(k+1)} & p^{(k)} & & p^{(3)} & p^{(2)} \\ & \ldots & & & \ldots \\ p^{(n-2)} & p^{(n-3)} & & p^{(k)} & p^{(k-1)} \\ p^{(n-1)} & p^{(n-2)} & \ldots & p^{(k+1)} & p^{(k)} \end{pmatrix}.$$

We need to compute the product $B_k(p)q$ for some $q \in C^k$.

Let us choose an arbitrary $N \geq n$. Note that the matrix $B_k(p)$ forms a block of the matrix $T_N(p)$ as its right-upper $k \times k$-corner. Therefore, the entries of the vector $B_k(p)q$ coincide with the first k components of the vector $g_N = T_N(p)q_N$, where

$$q_N = (0, \ldots, 0, q)^T \in C^N.$$

Using representation (4.10) of the matrix $T_N(p)$, we can justify the following algorithm.

1. Choose $N = 2^m$ such that $\frac{1}{2}N < n \leq N$.

2. Compute vectors $v = U_N^* p_N$ and $w = U_N^* q_N$ by (4.5).

3. Compute vector $u^{(i)} = v^{(i)} w^{(i)}$, $i = 1, \ldots, N$. (4.16)

4. Compute $g_N = U_N u = Z_N U_N^* u$ by (4.5).

5. Select the first k entries of g_N as components of vector $B_k(p)q$.

Lemma 6 *The computational complexity of computing the vector $B_k(p)q$ by algorithm (4.16) does not exceed*

$$n \cdot (2 + 3 \log_2 n) \tag{4.17}$$

complex multiplications. However, if $n = 2^m$, then the algorithm needs

$$k \cdot (2 + 3 \log_2 k) \tag{4.18}$$

complex multiplications.

Proof:
Indeed, as for the scheme (4.11), we need $N(1 + \frac{3}{2} \log_2 \frac{N}{2})$ complex multiplications. By the choice of N in (4.16), we always have $N \leq 2n$. However, if $n = 2^m$, then $N = n$. □

4.5 Coefficients of a rational function

Assume that the polynomial $p(z)$ is given by its coefficients $p \in C^n$. Our goal is to compute the first $b+1$ coefficients of the rational function

$$g(z) = \frac{1}{p(z)} = \sum_{i=0}^{\infty} g^{(i)} z^{(i)}.$$

Clearly, the sequence g satisfies the following infinite linear system:

$$\left(\begin{array}{cccc} p^{(0)} & 0 & 0 & \dots \\ p^{(1)} & p^{(0)} & 0 & \\ p^{(2)} & p^{(1)} & p^{(0)} & \\ \dots & & & \dots \\ p^{(n-1)} & p^{(n-2)} & p^{(n-3)} & \\ 0 & p^{(n-1)} & p^{(n-2)} & \\ 0 & 0 & p^{(n-1)} & \\ \dots & & & \dots \end{array} \right) \left(\begin{array}{c} g^{(0)} \\ g^{(1)} \\ g^{(2)} \\ \dots \end{array} \right) = \left(\begin{array}{c} 1 \\ 0 \\ 0 \\ \dots \end{array} \right)$$

However, the first $b+1$ coefficients of g can be found from a truncated sysem. Indeed, let us choose $N \geq b+1$. Consider the following N-dimensional system of linear equations:

$$\left(\begin{array}{cccccc} p^{(0)} & 0 & 0 & \dots & 0 & 0 \\ p^{(1)} & p^{(0)} & 0 & & 0 & 0 \\ p^{(2)} & p^{(1)} & p^{(0)} & & 0 & 0 \\ & \dots & & & \dots & \\ p^{(n-1)} & p^{(n-2)} & p^{(n-3)} & & & \\ 0 & p^{(n-1)} & p^{(n-2)} & & & \\ & \dots & & & \dots & \\ 0 & 0 & p^{(n-1)} & \dots & p^{(1)} & p^{(0)} \end{array} \right) \left(\begin{array}{c} g^{(0)} \\ g^{(1)} \\ g^{(2)} \\ \dots \\ g^{(N-1)} \end{array} \right) = \left(\begin{array}{c} 1 \\ 0 \\ 0 \\ \dots \\ 0 \end{array} \right) \in R^N. \qquad (4.19)$$

Clearly, its solution gives us the first N coefficients of the function $g(z)$.

Denote the low-triangular matrix of the system (4.19) by $L_N(p)$. We allow the size N of the matrix to be smaller than n. Thus, we are interested in a fast algorithm for solving the system

$$L_N(p) z = w. \qquad (4.20)$$

For even N, the structure of the matrix $L_N(p)$ is as follows:

$$L_N(p) = \left(\begin{array}{cc} L_{N/2}(p) & 0 \\ B_{N/2}(p_N) & L_{N/2}(p) \end{array} \right),$$

where p_N is an extension of p up to dimension N by zeros. The product of matrix $L_N^{-1}(p)$ and vector $w = (u, v) \in R^N$ can be computed as follows:

$$z = L_N^{-1}(p) \cdot w = \left(\begin{array}{c} x \\ y \end{array} \right), \quad x = L_{N/2}^{-1}(p) \cdot u, \quad y = L_{N/2}^{-1}(p) \cdot (v - B_{N/2}(p) x).$$

Thus, in order to solve (4.20), it is necessary to implement three operations:

$$\begin{aligned} &\mathbf{1.}\quad \text{Solve the system } L_{N/2}(p)\cdot x = u.\\ &\mathbf{2.}\quad \text{Compute the residual } \Delta = B_{N/2}(p)\cdot x. \qquad (4.21)\\ &\mathbf{3.}\quad \text{Solve the system } L_{N/2}(p)\cdot y = v-\Delta. \end{aligned}$$

Hence, we can see that the system (4.20) can be solved by a recursive procedure. Let us estimate its efficiency.

Assume that $N = 2^m$. Denote by M_N the complexity of finding a solution to (4.20) by a recursive procedure based on (4.21). Denote by C_k the complexity of multiplication of matrix $B_k(p)$ by a vector. Then, assuming that $M_1 = 1$, we can see from (4.21) that

$$\begin{aligned} M_N &= C_{N/2} + 2M_{N/2} = C_{N/2} + 2(C_{N/4} + 2M_{N/4})\\ &= C_{N/2} + 2C_{N/4} + 4M_{N/4} \;=\; 2^m + \sum_{i=1}^{m} 2^{i-1} C_{N\cdot 2^{-i}}. \end{aligned}$$

In view of (4.18), C_k does not exceed $k\cdot(2+3\log_2 k)$. Hence,

$$\begin{aligned} M_N &\le N + \sum_{i=1}^{m} 2^{i-1}(N\cdot 2^{-i})(2+3\log_2(N\cdot 2^{-i}))\\ &= N + \tfrac{1}{2}N\sum_{i=1}^{m}(2+3(m-i)) = N\cdot\left(1+\tfrac{m(3m+1)}{4}\right). \end{aligned}$$

Since we choose $N = 2^m$ such that $\frac{1}{2}N \le b+1 \le N$, we come to the following conclusion.

Lemma 7 *The first $b+1$ coefficients of the function $g(z)$ can be computed in*

$$\tfrac{b+1}{2}\cdot\left(3\log_2^2(b+1) + 7\log_2(b+1) + 8\right) \tag{4.22}$$

complex multiplications.

However, note that our estimate is exact only if b is not too big as compared with n. If $b >> n$, then on the top level the recursive procedure deals with very sparse matrices C_k. Indeed, if $k \ge n$, then all elements of C_k are zeros except the upper-right $n\times n$-corner filled by the matrix $B_n(p_{2n})$. Therefore, for $k \ge n$ we can bound C_k as follows:

$$C_k \le n\cdot(2+3\log_2 n).$$

Let us write down the corresponding upper bound for M_N assuming that $n = 2^l$.

$$\begin{aligned} M_N &\le N + \sum_{i=1}^{m-l} 2^{i-1} n\cdot(2+3l) + \sum_{i=m-l+1}^{m} 2^{i-1}(N\cdot 2^{-i})(2+3\log_2(N\cdot 2^{-i}))\\ &\le N + N\cdot(2+3l) + N\cdot l\cdot\tfrac{3l+1}{4} = N\cdot\tfrac{(3l+4)(l+3)}{4}. \end{aligned}$$

Thus, we have proved the following result.

Lemma 8 *If $n = 2^l < \frac{1}{2}(b+1)$, then the first $b+1$ coefficients of the function $g(z)$ can be computed in*

$$\tfrac{b+1}{2}\cdot(3\log_2 n + 4)(\log_2 n + 3) \tag{4.23}$$

complex multiplications.