LLL Algorithm and the Optimal Finite Wordlength FIR Design

Dušan M. Kodek

***Abstract*—In practical finite-impulse-response (FIR) digital filter applications, it is often necessary to represent the filter coefficients with a finite number of bits. The finite wordlength restriction increases the filter deviation. This increase can be reduced substantially if the optimal finite wordlength coefficients are used. The time needed to compute these coefficients is greatly reduced with the help of a lower bound on the deviation increase. Derivation of an improved lower bound that uses the well-known LLL algorithm is presented in this correspondence.**

***Index Terms*—Finite wordlength, FIR digital filters, LLL algorithm, minimax approximation.**

I. Introduction

In many practical applications, it is not possible to use the optimal FIR digital filter coefficients that were computed by some "infinite precision" method and are typically 32-bit floating-point numbers. If we wish to use a fixed-point DSP processor, which is almost always cheaper and/or faster than a floating-point one, we would like the filter coefficients represented with the number of bits b that is as small as possible. Rounding of the infinite precision coefficients to their nearest b-bit representations gives a suboptimal filter that can be much worse than the optimal filter. More than a 30-dB difference was observed in some cases. Papers on the practical aspects of finite wordlength finite-impulse-response (FIR) filter design typically use one of the two types of coefficient constraints: signed b-bit integers [1]–[8] or sums of a limited number of signed power-of-two terms [9]–[11]. The integer coefficients can be used, for example, on a fixed-point digital signal processing (DSP) processor or a field-programmable gate array (FPGA), whereas the sums of power-of-two allow a multiplierless implementation.

Optimal coefficients require a computationally demanding solution of the discrete variable approximation problem. The computing time was often too long for practical use but this is quite different now. Faster computers are only part of the reason for the increased speed. Just as important are the theoretical results that were derived for the finite wordlength minimax approximation problem [4]–[7], [12]. The lower bound for the increase of minimax approximation error that is caused by the finite wordlength restriction reduces the number of subproblems which must be solved to get the optimal solution. An early version of the bound was given in [12] and later improved in [13]–[16]. This correspondence presents an improvement of the bound in [16] that is obtained with the help of the well-known LLL algorithm.

II. The Finite Wordlength Design Problem

Let us start with the infinite precision design problem. For reasons of simplicity, we limit our attention to filters with real-valued coefficients although the ideas presented can also be applied to complex-valued coefficients [17]. The frequency response $H^*(\omega)$ of a length N optimal infinite precision (i.e., filter coefficients can be any real number) linear phase FIR digital filter is equal to

$$H^*(\omega) = \sum_{k=0}^{N-1} h^*(k) e^{-j\omega k} = e^{j\left(L\frac{\pi}{2} - \frac{N-1}{2}\omega\right)} Q(\omega) P^*(\omega) \quad (1)$$

where $L = 0$ or 1 and

$$P^*(\omega) = \sum_{k=0}^{n} a_k^* \cos k\omega. \quad (2)$$

Depending on N and filter symmetry, there are exactly four types of FIR filters and four real functions $Q(\omega)$. The degree n of the cosine polynomial is related to the filter length N and there are formulas that relate the optimal coefficients $h^*(k)$ and a_k^*. Function $Q(\omega)$ is irrelevant from the point of view of the approximation problem, and we will therefore use $Q(\omega) = 1$. To find $P^*(\omega)$, one must solve the following minimax approximation problem:

$$\min_{P(\omega)} \max_{a \le \omega \le b} |W(\omega)(D(\omega) - P(\omega))|. \quad (3)$$

The real function $D(\omega)$ is the desired frequency response, the weighting function $W(\omega)$ is by definition real and positive, and the interval $[a, b]$ is a subset of the interval $[0, \pi]$. The standard approach to solving (3) is to use the Remez algorithm in a way that was first described by Parks and McClellan [18].

The problem's complexity increases dramatically when the finite wordlength constraint is introduced. The optimal solution is no longer unique and the approximation error does not go towards zero when $n \to \infty$. In this correspondence we will, without loss of generality, make the finite wordlength constraint equal to requesting that the filter coefficients $h(k)$ are b-bit integers from the set $I_b = \{-2^{b-1}, \ldots, -1, 0, 1, \ldots, 2^{b-1}\}$. Constraining $h(k)$ to the set I_b requires a redefinition or scaling of the original infinite precision approximation problem. This is necessary to bring the coefficients $h(k)$ within the range of numbers in I_b and can be done with the help of a scaling factor s. Let us assume that s is known and denote as $D_u(\omega)$, $W_u(\omega)$, and $P_u(\omega)$ the original (unscaled) problem. The approximation problem that gives the optimal b-bit approximation error E_{min} can be written as

$$\begin{aligned} E_{min} &= \min_{P_u(\omega)} \max_{a \le \omega \le b} \left| \frac{W_u(\omega)}{s} (sD_u(\omega) - sP_u(\omega)) \right| \\ &= \min_{P(\omega)} \max_{a \le \omega \le b} |W(\omega)(D(\omega) - P(\omega))| \end{aligned} \quad (4)$$

where the finite wordlength polynomial $P(\omega)$ equals

$$P(\omega) = sP_u(\omega) = \sum_{k=0}^{n} a_k \cos k\omega. \quad (5)$$

Functions $D(\omega) = sD_u(\omega)$ and $W(\omega) = W_u(\omega)/s$ are the scaled input functions $D_u(\omega)$ and $W_u(\omega)$. Since all $h(k)$ must be in I_b, we must have $a_k \in I_b$ for $k = 1, 2, \ldots, n$ and $a_0 \in I_b/2$, where $I_b/2$ denotes the set I_b in which all elements are divided by 2. This follows directly from type 1 FIR filters equations

$$\begin{aligned} h(n) &= 2a_0, \quad n = (N-1)/2 \\ h(n-k) &= a_k, \quad k = 1, 2, \ldots, n. \end{aligned} \quad (6)$$

Note that a_0 is a special case. The scaling factor s can be interpreted as the filter gain. The choice of s is not trivial and is described in [15] and [16]. In this correspondence, we assume that s is a known constant.

Manuscript received June 30, 2011; revised September 21, 2011; accepted November 15, 2011. Date of publication December 02, 2011; date of current version February 10, 2012. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ljubisa Stankovic.

The author is with the University of Ljubljana, Faculty of Computer and Information Science, Tržaška 25, 1000 Ljubljana, Slovenia (e-mail: duke@fri.uni-lj.si).

Digital Object Identifier 10.1109/TSP.2011.2177974

III. Lower Bound Basics of Minimax Approximation

The new LLL based lower bound is derived from the one in [16] an its basics are given here. Properties of the optimal infinite precision polynomial $P^*(\omega)$ follow from the Chebyshev equioscillation theorem [19] that gives the conditions for the optimal minimax approximation of degree n. There are at least $n+2$ extremal points in $[a, b]$ at which the approximation error achieves its maximum. Let $\omega_i, a \le \omega_0 < \omega_1 < \cdots < \omega_{n+1} \le b$, be these extremal points. Then

$$W(\omega_i)\left(D(\omega_i) - \sum_{k=0}^{n} a_k^* \cos k\omega_i\right) = (-1)^i d^*, \quad i = 0, 1, \ldots, n+1 \tag{7}$$

where $|d^*|$ is the optimal approximation error. No such property exists for b-bit $P(\omega)$. Approximation error $e(\omega)$ is simply

$$W(\omega)\left(D(\omega) - \sum_{k=0}^{n} a_k \cos k\omega\right) = e(\omega). \tag{8}$$

Since $P^*(\omega)$ is unique, the approximation error increases if $P(\omega) \neq P^*(\omega)$. There is $\epsilon = \max_{a\le\omega\le b} |e(\omega)| - |d^*|$, where $\epsilon > 0$.

The problem we wish to address can be stated as follows: What is the minimum ϵ given the best possible coefficients $a_k \in I_b$? Let us denote it as δ and define it formally as

$$\delta = \min_{b-\text{bit } P(\omega)} \epsilon = \min_{b-\text{bit } P(\omega)} \max_{a\le\omega\le b} |e(\omega)| - |d^*|. \tag{9}$$

Using the approach developed in [12]–[14], we wish to express δ as a function of differences Δa_k

$$\begin{aligned} \Delta a_0 &= 2a_0^* - 2a_0, \quad 2a_0 \in I_b \\ \Delta a_k &= a_k^* - a_k, \quad a_k \in I_b, k = 1, 2, \ldots, n \end{aligned} \tag{10}$$

where Δa_0 is again a special case. Combining (7) and (8) gives

$$e(\omega_i) = \frac{W(\omega_0)}{2}\Delta a_0 + \sum_{k=1}^{n} W(\omega_i)\cos k\omega_i \Delta a_k + (-1)^i \operatorname{sign}(d^*)|d^*|. \tag{11}$$

These equations can be viewed as a system of $n+2$ equations with $n+2$ unknowns. The system's matrix is identical to the one in (7) and is always invertible. The inverse can be written as

$$\begin{aligned} \Delta a_k &= \sum_{i=0}^{n+1} g_{ki} e(\omega_i), \quad k = 0, 1, \ldots, n, \\ |d^*| &= \sum_{i=0}^{n+1} g_{n+1i} e(\omega_i) \end{aligned} \tag{12}$$

where g_{ki} are the elements of the inverted matrix. This matrix plays a central role in the lower bound derivation and is written explicitly

$$\mathbf{G} = [g_{ki}], \quad k, i = 0, 1, \ldots, n+1. \tag{13}$$

Note that $\mathbf{G}$ depends only on $\operatorname{sign}(d^*)$ and on extremal points ω_i. The matrix elements g_{n+1i} are all nonzero, and their signs alternate, which is of great importance for our purpose. It was shown in [16] that the special properties of $\mathbf{G}$ lead to the lower bound

$$\delta \ge \max_{0\le k\le n} \min(\Delta a_{k+}/f_{pk}, \Delta a_{k-}/f_{mk}) \tag{14}$$

where Δa_{k+} is the smallest positive difference Δa_k from (10) and Δa_{k-} the smallest negative difference. The positive number f_{pk} and the negative f_{mk} are equal to

$$f_{pk} = \max_{0\le i\le n+1}\left(-\frac{g_{ki}}{g_{n+1i}}\right), \quad f_{mk} = \min_{0\le i\le n+1}\left(-\frac{g_{ki}}{g_{n+1i}}\right) \tag{15}$$

for $k = 0, 1, \ldots, n$. This lower bound uses only one of the $n+1$ differences Δa_k, namely the one that gives the maximum in (14). An improved lower bound can be obtained if two are used. Let us select any two of the $n+1$ (12) and denote the corresponding indices as j and ℓ. Equation j is multiplied by a factor γ and added to equation ℓ, which gives

$$\epsilon_{jl} = \max((\Delta a_\ell + \gamma \Delta a_j)/f_{pj\ell}, (\Delta a_\ell + \gamma \Delta a_j)/f_{mj\ell}) \tag{16}$$

where

$$\begin{aligned} f_{pj\ell} &= \max_{0\le i\le n+1}\left(-\frac{g_{\ell i} + \gamma g_{ji}}{g_{n+1i}}\right) \\ f_{mj\ell} &= \min_{0\le i\le n+1}\left(-\frac{g_{\ell i} + \gamma g_{ji}}{g_{n+1i}}\right). \end{aligned} \tag{17}$$

For a given pair $\Delta a_\ell, \Delta a_j$, it is possible to compute γ that gives the highest possible $\epsilon_{j\ell}$. The algorithm that computes γ is described in [16]. The lower bound is found by trying all feasible pairs $\Delta a_\ell, \Delta a_j$ and selecting the lowest ϵ_{jl}.

There are $n(n+1)/2$ possible pairs j, ℓ, which is far too many. In our implementation, only the row $j = n$ is paired with rows in a subset V. The reasons for making special row n follow from the role of the highest degree coefficient a_n. The subset V contains the row that gives the maximum in (14) and two additional rows with the lowest $\max(|f_{pk}|, |f_{mk}|)$ from (15). This choice reduces the search space to at most three pairs and greatly decreases the time needed to compute the lower bound with a negligible lower bound reduction. The lower bound is computed separately for positive and negative Δa_n

$$\begin{aligned} \delta_+ &\ge \max_{n,\ell\in V}\left[\min_{\Delta a_n \ge 0, \Delta a_\ell} \epsilon_{n\ell}\right] \\ \delta_- &\ge \max_{n,\ell\in V}\left[\min_{\Delta a_n < 0, \Delta a_\ell} \epsilon_{n\ell}\right] \end{aligned} \tag{18}$$

which is useful for its application in the branch-and-bound method.

Extension of this idea to three or more Δa_k is much more complicated and leads to excessive computing times. The LLL algorithm provides a more promising approach.

IV. Using the LLL Algorithm to Improve the Lower Bound

The LLL algorithm [20] is based on the lattice theory, which is a powerful concept with applications in many domains of science. It is possible to formulate the finite wordlength design problem (4) within the framework of lattice theory as the "closest vector problem" or CVP. Although the CVP is NP-hard [21], this does not necessarily apply to (4). Our problem is a special case of CVP, and there seems to be no proof that it is NP-hard even if this seems most likely. Complexity and lattice theory are beyond the scope of this correspondence. We will use only those simple properties of the LLL reduced lattice basis that are needed to improve the lower bound developed in the previous section.

The LLL algorithm is often referred to as an integer Gram–Schmidt procedure. Let $\mathbf{X}$ be an arbitrary $\ell \times \ell$ square matrix. The LLL reduced basis spanned by the rows of $\mathbf{X}$ is given by

$$\mathbf{X_L} = \mathbf{UX} \tag{19}$$

where $\mathbf{U}$ is an integer matrix and $\mathbf{X_L}$ the LLL reduced basis. The LLL algorithm computes $\mathbf{U}$ and $\mathbf{X_L}$ in polynomial time. Note that, in (19), the rows represent lattice vectors. This differs from the usual approach where vectors are represented by columns. Row vectors are more suitable for our purpose. This is a minor complication and the LLL algorithm can easily be adjusted to accommodate this change.

Many versions of the LLL algorithm exist in the literature. Since matrix $\mathbf{G}$ is well behaved, we used a simple floating-point version described in [22]. The standard value of its parameter $\omega = 0.75$ was changed to 0.60. This change increased the speed of the LLL algorithm with negligible effect on the lower bound. The LLL reduced matrix $\mathbf{X_L}$ has several remarkable properties that make it useful in many areas. The property that is important for our purposes is the fact that the row vectors of $\mathbf{X_L}$ are short in the Euclidean sense. More precisely, the length of row k vector is defined as its L_2 norm

$$\|x_{Lk}\|_2 = \left(x_{Lk0}^2 + x_{Lk1}^2 + \cdots + x_{Lk\ell-1}^2\right)^{1/2} \tag{20}$$

where x_{Lki} are the elements of matrix $\mathbf{X_L}$. In particular, the LLL algorithm gives $\mathbf{X_L}$ in which the first row vector is the shortest and the other row vectors are also short. For our lower bound, we would like to have

$$\max_{0\le i\le n+1}\left(\left|\frac{g_{ki}}{g_{n+1i}}\right|\right), \quad k = 0, 1, \ldots, n \tag{21}$$

as small as possible since this gives the highest lower bound (16). The terms (21) correspond to the elements of the matrix $\mathbf{G}$ in which the first $n+1$ rows are divided by the corresponding elements of the last row. It follows from (21) that we are looking for vectors that are short in the minimax sense. Instead of short vectors with length defined by L_2 norm (20), we need $\mathbf{X_L}$ with short vectors where the length is defined by L_∞ norm

$$\|x_{Lk}\|_\infty = \max_{0\le i\le \ell-1}(|x_{Lki}|). \tag{22}$$

Although the LLL algorithm only gives L_2 short vectors, they are still useful because the norms are related by $\|x_{Lk}\|_\infty \le \|x_{Lk}\|_2 \le \sqrt{\ell}\|x_{Lk}\|_\infty$. Short L_2 vectors are therefore also rather short L_∞ vectors, which means that the LLL algorithm will probably improve the lower bound. To confirm this, let us first write (12) in the matrix form

$$\mathbf{\Delta a} = \mathbf{G e}. \tag{23}$$

Vector $\mathbf{\Delta a}$ consists of $\Delta a_k, k = 0, 1, \ldots, n$, and $|d^*|$, whereas $\mathbf{e}$ is the approximation error $e(\omega_i), i = 0, 1, \ldots, n+1$. We define the diagonal $n+2 \times n+2$ matrix $\mathbf{C}$ that has as diagonal elements the last row of $\mathbf{G}$

$$c_{ii} = g_{n+1i}, \quad c_{ki} = 0, \quad k \ne i. \tag{24}$$

Using $\mathbf{C}$ (23) can be written as

$$\mathbf{\Delta a} = \mathbf{G C^{-1} C e} \tag{25}$$

where $\mathbf{GC^{-1}}$ is the matrix $\mathbf{G}$ in which all rows are divided by the corresponding elements of the last row. Applying the LLL algorithm on $\mathbf{GC^{-1}}$ gives

$$\mathbf{G_L = U G C^{-1}} \tag{26}$$

where $\mathbf{U}$ is the integer matrix. Our version of the LLL algorithm is slightly modified: It does not use the last row of $\mathbf{GC^{-1}}$, which means that this row is the same in $\mathbf{G_L}$ and in $\mathbf{GC^{-1}}$. This important modification is needed because our lower bound needs the property of matrix elements g_{n+1i} preserved. It makes the elements of the last row and column of integer matrix $\mathbf{U}$ all zero, except for the diagonal element u_{n+1n+1}, which equals 1. Using (26), we can rewrite (25) as

$$\mathbf{U\Delta a = G_L C e}. \tag{27}$$

We define $\mathbf{\Delta a_L = U \Delta a}$ and (27) can be written in the final form

$$\mathbf{\Delta a_L = G_L C e}. \tag{28}$$

The elements of $\mathbf{\Delta a_L}$ follow from (10) as

$$\Delta a_{Lk} = \sum_{i=0}^{n} u_{ki}\Delta a_i = \left(u_{k0}2a_0^* + \sum_{i=1}^{n} u_{ki}a_i^*\right) - \left(u_{k0}2a_0 + \sum_{i=1}^{n} u_{ki}a_i\right) \tag{29}$$

for $k = 0, 1, \ldots, n$. Note that the sum upper limit is n and not $n+1$ because of zero elements in the last column of $\mathbf{U}$. Since u_{ki} are integers and since $2a_0 \in I_b$ and $a_i \in I_b, i = 1, 2, \ldots, n$, the second term in (29) is an integer. This property is very important in lower bound derivation and, of course, derives from the fact that the LLL algorithm produces an integer $\mathbf{U}$. Equation (28) now plays the role of (12) in the previous section. It can be written explicitly as

$$\begin{aligned}\Delta a_{Lk} &= \sum_{i=0}^{n+1} g_{LCki}e(\omega_i), \quad k = 0, 1, \ldots, n,\\ |d^*| &= \sum_{i=0}^{n+1} g_{LCn+1i}e(\omega_i)\end{aligned} \tag{30}$$

where g_{LCki} are the elements of matrix $\mathbf{G_L C}$. Observe that g_{LCki} are not needed for the lower bound computation. It follows from (15) and (17) that g_{LCki}/g_{LCn+1i} are needed and these are equal to the elements of matrix $\mathbf{G_L}$

$$g_{Lki} = \frac{g_{LCki}}{g_{LCn+1i}}, \quad k = 0, 1, \ldots, n. \tag{31}$$

The effect of $\mathbf{U}$ on the original (12) can be described as an integer-multiplied addition of its rows. As mentioned previously, the LLL algorithm ensures that this addition produces short vectors and therefore a better lower bound. The derivation of the LLL based lower bound is simply the repetition of steps (14)–(16) with g_{LCki} instead of g_{ki}. Equations (15) become

$$f_{Lpk} = \max_{0\le i\le n+1}(-g_{Lki}), \quad f_{Lmk} = \min_{0\le i\le n+1}(-g_{Lki}) \tag{32}$$

for $k = 0, 1, \ldots, n$. Similarly, (17) becomes

$$\begin{aligned}f_{Lpj\ell} &= \max_{0\le i\le n+1}(-g_{L\ell i} - \gamma g_{Lji})\\ f_{Lmj\ell} &= \min_{0\le i\le n+1}(-g_{L\ell i} - \gamma g_{Lji}).\end{aligned} \tag{33}$$

The role of a_k^* is now taken by the coefficients a_{Lk}^*

$$a_{Lk}^* = u_{k0}2a_0^* + \sum_{i=1}^{n} u_{ki}a_i^*, \quad k = 0, 1, \ldots, n. \tag{34}$$

For any set of coefficients a_{Lk}^*, we see from (29) that there exist integers, not necessarily from I_b, that are the nearest upper and lower neighbors of a_{Lk}^*. Coefficient a_{L0}^* is no longer a special case because

it is now a linear combination of many a_k^*. The smallest positive difference is denoted Δa_{Lk+}, whereas the smallest negative difference is Δa_{Lk-}. The simple lower bound (14) now equals

$$\delta_L \geq \max_{0\leq k\leq n} \min(\Delta a_{Lk+}/f_{Lpk}, \Delta a_{Lk-}/f_{Lmk}). \tag{35}$$

Similarly, $\epsilon_{Lj\ell}$ is computed as in (16):

$$\max((\Delta a_{L\ell} + \gamma \Delta a_{Lj})/f_{Lpj\ell}, (\Delta a_{L\ell} + \gamma \Delta a_{Lj})/f_{Lmj\ell}) \tag{36}$$

and the lower bound equals

$$\delta_L \geq \max_{j,\ell\in V} \left[\min_{\Delta a_{Lj}, \Delta a_{L\ell}} \epsilon_{Lj\ell} \right]. \tag{37}$$

Note that there are differences between (37) and (18). The row n does not plays a special role anymore because it follows from (29) that Δa_{Ln} is a function of several coefficients a_k and not only of a_n as before. The row j is now the row that gives the maximum in (35) and is paired with rows in the subset V, which now contains three rows with the lowest $\max(|f_{Lpk}|, |f_{Lmk}|)$ in (32).

Since the LLL algorithm gives L_2 and not L_∞ short vectors, it can sometimes occur that δ_L is lower than δ_+ and/or δ_- from (18). In our implementation, we therefore compute

$$\delta_{L+} = \max(\delta_L, \delta_+), \quad \delta_{L-} = \max(\delta_L, \delta_-). \tag{38}$$

This gives a somewhat better lower bound than δ_L and is also justified computationally because it costs very little to compute δ_+ and δ_-. Computing matrices $\mathbf{G}$, $\mathbf{G_L}$, and $\mathbf{U}$ is the most demanding part of lower bound computation.

V. Application of the Lower Bound

The branch-and-bound method is by far the most widely used tool for solving NP-hard combinatorial optimization problems. It is based on the idea of intelligent enumeration which splits the initial problem into many subsets [23]. Branch-and-bound is not one specific algorithm, but rather a very wide class. For each specific problem type, it has to be filled out with details without which our lower bound derivation would not be complete. A good starting finite wordlength solution is needed prior to the start. Rounding the infinite precision coefficients a_k^* is one possibility. We use a much better approach that is called "telescopic rounding" [24] and almost always gives a better starting solution. Its approximation error E_{up} is the starting upper bound for the optimal solution $E_{\min}$.

The search consists of node selection. Constraints are added to the selected node, thereby creating subproblems. In our implementation, we use the Remez algorithm for subproblem solving. This means that, for each node, the new constraint must always be applied to the highest degree coefficient a_k that is not yet from I_b. Or in other words, the constraints begin with a_n and continue with $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$. Any other choice of coefficient constraints violates the Haar condition [19], and both the Remez algorithm and our lower bound do not work.

Two subproblems are created from the infinite precision solution by making the coefficient a_n equal to

$$a_n^{(1)} = \lfloor a_n^* \rfloor, \quad a_n^{(2)} = \lfloor a_n^* \rfloor + 1, \quad a_n^{(1)}, a_n^{(2)} \in I_b \tag{39}$$

where $\lfloor a_n^* \rfloor$ denotes the nearest lower integer to a_n^*, and the superscripts (1) and (2) denote the node number. These two subproblems are of degree $n-1$, and the effect of fixed $a_n^{(1)}$ is taken into account by using $D^{(1)}(\omega) = D(\omega) - a_n^{(1)} \cos n\omega$, instead of $D(\omega)$ (for node (2) $D^{(2)}(\omega) = D(\omega) - a_n^{(2)} \cos n\omega$). Since both subproblems are treated in the same manner, we will give a description for node (1) only. The subproblem is solved using the Remez algorithm giving a set of infinite precision coefficients $a_k^{*(1)}, k = 0, 1, \ldots, n-1$ and the corresponding approximation errors $|d^{*(1)}|$. The matrix $\mathbf{G}$ is computed as described in (11)–(13), and the LLL algorithm is applied producing matrices $\mathbf{U}$ and $\mathbf{G_L}$ as described in (23)–(28). The LLL coefficients $a_{Lk}^{*(1)}, k = 0, 1, \ldots, n-1$ are computed using (34), and the following three lower bounds are computed.

1) Positive $\Delta a_{n-1}^{(1)}$ are used and the lower bound δ_{L+} is computed using (38). It is denoted $\delta_{L+}^{(1)}$ and will be applied later to subproblems that are created by constraining $a_{n-1}^{*(1)}$ to lower integers. It follows from (10) that such constraints always give $\Delta a_{n-1}^{(1)} \geq 0$.
2) Negative $\Delta a_{n-1}^{(1)}$ are used and the lower bound δ_{L-} is computed using (38). It is denoted $\delta_{L-}^{(1)}$ and will be applied to subproblems that are created by constraining $a_{n-1}^{*(1)}$ to higher integers.
3) This lower bound is used for subproblems that are created by replacing $a_n^{(1)}$ from (39) with $a_n^{(1)} - 1$ and, if necessary, also with $a_n^{(1)} - 2$, $a_n^{(1)} - 3$, and so on until all feasible values of reduced $a_n^{(1)}$ are explored (correspondingly, $a_n^{(2)}$ is replaced with $a_n^{(2)} + 1$ and, if necessary, also with $a_n^{(1)} + 2$, $a_n^{(1)} + 3$, and so on). For these subproblems, it is possible to compute the lower bound using the already computed matrices $\mathbf{G}$, $\mathbf{U}$, and $\mathbf{G_L}$. To see how this is done, let us replace $D^{(1)}(\omega)$ with $D^{(1)}(\omega) + \cos n\omega$ (correspondingly, $D^{(2)}(\omega)$ with $D^{(2)}(\omega) - \cos n\omega$). The approximation error $|d_x^{*(1)}|$ and the coefficients $a_{kx}^{*(1)}$ can be computed simply as

$$a_{kx}^{*(1)} = a_k^{*(1)} + \sum_{i=0}^{n} g_{ki} W(\omega_i) \cos n\omega_i \tag{40}$$

$$\left| d_x^{*(1)} \right| = |d^{*(1)}| + \sum_{i=0}^{n} g_{ni} W(\omega_i) \cos n\omega_i. \tag{41}$$

Again, coefficient $k = 0$ is a special case with $2a_{0x}^{*(1)}$ and $2a_0^{*(1)}$ in (40). Lower bound $\delta_v^{(1)} = \min(\delta_+, \delta_-)$ is computed from (19) using coefficients $a_{kx}^{*(1)}$. Note that there are no restrictions on the sign of $\Delta a_{n-1x}^{(1)}$ which is why $\min(\delta_+, \delta_-)$ must be used. Next, the LLL coefficients $a_{Lkx}^{*(1)}, k = 0, 1, \ldots, n-1$, are computed from $a_{kx}^{*(1)}$ using (34) and the LLL lower bound $\delta_{Lv}^{(1)}$ is computed with (37). Both bounds are combined into

$$\delta_{Lt}^{(1)} = \max\left(\delta_{Lv}^{(1)}, \delta_v^{(1)}\right). \tag{42}$$

Since $\delta_{Lt}^{(1)}$ is computed relative to $|d_x^{*(1)}|$, and we need a lower bound relative to $|d^{*(1)}|$, the following formula is used:

$$\delta_{Lx}^{(1)} = \left| d_x^{*(1)} \right| - \left| d^{*(1)} \right| + \delta_{Lt}^{(1)} \tag{43}$$

where $\delta_{Lx}^{(1)}$ is our third lower bound. This bound, however, is useful only if $|d_x^{*(1)}| - |d^{*(1)}| \geq 0$. It must be set to zero otherwise since $|d_x^{*(1)}| - |d^{*(1)}|$ can become even more negative for $a_n^{(1)} - 2, a_n^{(1)} - 3, \ldots$.

Having all three bounds, we now store the subproblem (1) as a node in the tree. Information stored includes $a_n^{(1)}$, which is a fixed integer, plus $|d^{*(1)}|$, $a_{n-1}^{*(1)}$, $\delta_{L+}^{(1)}$, $\delta_{L-}^{(1)}$, and $\delta_{Lx}^{(1)}$. Information about the direction of $a_n^{(1)}$ creation relative to a_n^* must be stored as well. From (39), we see that $a_n^{(1)}$ is lower than a_n^*, which is denoted as negative direction -1. All of the above is repeated for subproblem (2), which is also stored as a node in the tree. Since the subproblem (2) was created by (39), its direction is positive $+1$.

Creation of the first two nodes represents a preparation for the branch-and-bound search. From here on, all nodes are handled in a slightly different manner. Three, not two, new subproblems are always created from the selected node, and the lower bounds are used to

TABLE I
RESULTS OF THE LOWER BOUND EFFECTIVENESS

Filter	Number of subproblems			Computing time		
	No bound	Old bound	LLL bound	No bound	Old bound	LLL bound
A35/8	1991	786	655	0.08	0.05	0.05
A45/9	6023	2384	1919	0.41	0.23	0.23
A55/10	202727	78490	67126	17.10	10.86	8.91
B35/9	7082	2839	1667	0.31	0.14	0.11
B45/10	82283	32783	18735	4.96	2.45	1.67
B55/11	146405	58145	40361	13.53	6.57	5.40
C35/8	5024	1861	1549	0.16	0.10	0.09
C45/8	48074	18058	15304	2.34	1.17	1.26
C55/9	4636823	1802217	1476585	462.68	207.70	187.64
D35/9	35324	13419	7614	1.09	0.55	0.39
D45/9	245975	94377	53032	12.48	5.97	4.24
D55/10	2736803	1093386	665419	266.03	125.80	88.56
E35/8	3836	1466	1273	0.14	0.08	0.09
E45/9	15107	5746	4944	0.86	0.45	0.45
E55/10	80369	30787	27388	6.79	3.34	3.49
Total	8253846	3236744	2383571	788.96	365.28	302.58

decide which subproblems are worth solving. Assume that the node containing subproblem (1) is the selected node. Three subproblems are created with the following constraints:

$$a_n^{(3)} = a_n^{(1)} - 1, \quad a_n^{(3)} \in I_b \tag{44}$$

$$a_{n-1}^{(4)} = \left\lfloor a_{n-1}^{*(1)} \right\rfloor, \quad a_{n-1}^{(5)} = \left\lfloor a_{n-1}^{*(1)} \right\rfloor + 1, \quad a_{n-1}^{(4)}, a_{n-1}^{(5)} \in I_b. \tag{45}$$

Note that -1 in (44) is used because the stored direction of node (1) is -1 (for node (2) there would be $+1$). Subproblem (44) is solved and stored as a new node in the tree only if there is $|d^{*(1)}| + \delta_{Lx}^{(1)} < E_{\mathrm{up}}$. Obviously, if this is not true this subproblem cannot lead to a solution that is better than E_{up} and is therefore discarded. Since minimax approximation is a convex optimization problem, this also applies to all subproblems with constraints $a_n^{(1)} - 2$, $a_n^{(1)} - 3, \ldots$. Similarly, the subproblems (45) need to be solved only if the conditions $|d^{*(1)}| + \delta_{L+}^{(1)} < E_{\mathrm{up}}$ and $|d^{*(1)}| + \delta_{L-}^{(1)} < E_{\mathrm{up}}$ hold.

The lower bounds $\delta_{Lx}^{(1)}$, $\delta_{L+}^{(1)}$, and $\delta_{L-}^{(1)}$ eliminate a large number of subproblems without the need to solve them first. A combination of depth-first and lowest lower bound strategy was used for node selection. Several other strategies were also tried, and the experience shows that they have little effect on the number of subproblems that must be solved. In most cases, the optimal solution is found within the first 10% to 20% of subproblems. This is followed by a long tail of remaining subproblems, which is independent of selection strategy.

VI. RESULTS

Fifteen filters with five different sets of frequency-domain specifications, denoted A through E, were used for testing. The frequency specifications are identical to those that were used in [7] and [16] and are given there. We denote by A35/8 the filter design problem for specification A, length $N = 35$ ($n = 18$ independent coefficients), and $b = 8$ bits (sign included); similarly for A45/9, B35/9, and so on. Constant scaling factor $s = 2^{b-2}$ was used in all design cases.

As expected, the LLL lower bound was found to be better than the old lower bound given by (18). Table I shows a summary of the results, comparing the number of branch-and-bound subproblems that must be solved when lower bound is not used, when the old lower bound is used, and when the LLL lower bound is used. The corresponding computing times in seconds on a 2.67 GHz Intel Core i7 processor are also given. All computations were done using 64-bit arithmetic and only one processor core was used for easier comparison with the older results. It is well known that the branch-and-bound method is easily adaptable to multi core implementation. The computing times were reduced by a factor of almost three when four cores were used.

The results show that the LLL lower bound consistently gives the lowest number of subproblems. The average reduction is about 35%. Because of the time needed by the LLL algorithm, our implementation gives only an average 20% reduction in computing time. Note that the computing times depend on the machine and/or programmer skills, which makes the reduction of subproblems more relevant.

The savings are not dramatic. However, there is space for considerable additional improvements. An obvious way is to use a more efficient realization of the LLL algorithm. A more promising approach is to use a version of the LLL algorithm that uses the L_∞ norm. This would reduce the number of subproblems and lead to more significant savings. Whether such versions of the LLL algorithms can be made fast enough is still an open question. The method presented in this correspondence can be considered a new first step towards a better solution of the finite wordlength Chebyshev approximation problem.

A comment on the filter length is also in order here. For all finite wordlength filters, there exists a maximal filter length $N_{\max}$ and the corresponding number of coefficients $n_{\max}$. If the number of coefficients n is increased beyond $n_{\max}$, the additional coefficients are all zero. For example, filters A45/9 and C45/8 both have $N_{\max} = 43$ ($n_{\max} = 22$). Obviously, these filters do not get any better by using $N > 43$. The computing times, however, increase considerably. No quick and simple method that finds $N_{\max}$ is known, and this remains an interesting open problem.

REFERENCES

[1] D. M. Kodek, "An algorithm for the design of optimal finite wordlength FIR digital filters," in *Proc. Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Denver, CO, Apr. 9–11, 1980, vol. I, pp. 73–76.

[2] D. M. Kodek, "Design of optimal finite word-length FIR digital filters using integer programming techniques," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, pp. 304–308, Jun. 1980.

[3] Y. C. Lim, S. R. Parker, and A. G. Constantinides, "Finite word length FIR filter design using integer programming over a discrete coefficient space," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 30, pp. 661–664, Aug. 1982.

[4] D. M. Kodek and K. Steiglitz, "Comparison of optimal and local search methods for designing finite word-length FIR digital filters," in *Proc. Conf. Inf. Sci. Syst.*, Baltimore, MD, Mar. 28–30, 1979, pp. 1–4.

[5] D. M. Kodek and K. Steiglitz, "A theoretical performance bound on the performance of direct-form finite wordlength FIR digital filters," in *Proc. Conf. Inf. Sci. Syst.*, Princeton, NJ, Mar. 26–28, 1980, pp. 369–371.

[6] D. M. Kodek and K. Steiglitz, "Filter-length word-length tradeoffs in FIR digital filter design," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 28, pp. 739–744, Dec. 1980.

[7] D. M. Kodek and K. Steiglitz, "Comparison of optimal and local search methods for designing finite wordlength FIR digital filters," *IEEE Trans. Circuits Syst.*, vol. 28, pp. 28–32, Jan. 1982.

[8] S. Shantal and S. Y. Kulkarni, "High speed and low power FPGA implementation of FIR filter for DSP applications," *Eur. J. Sci. Res.*, vol. 31, pp. 19–28, May 2009.

[9] Y. C. Lim and S. R. Parker, "FIR filter design over a discrete power-of-two coefficient space," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, pp. 583–591, Jun. 1983.

[10] C. L. Chen and A. N. Wilson, "A Trellis search algorithm for the design of FIR filters with signed power-of-two coefficients," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 29–39, Jan. 1999.

[11] Y. C. Lim, R. Yang, D. Li, and J. Song, "Signed power-of-two term allocation scheme for the design of digital filters," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 577–584, May 1999.

[12] W. P. Niedringhaus, K. Steiglitz, and D. M. Kodek, "An easily computed performance bound for finite wordlength direct-form FIR digital filters," *IEEE Trans. Circuits Syst.*, vol. 29, pp. 191–193, Mar. 1982.

[13] D. M. Kodek, "A lower bound for the increase of finite wordlength minimax approximation error," in *Proc. Electr. Comput. Sci. Conf. (ERK)*, Portorož, Slovenia, Sep. 28–30, 1992, vol. B, pp. 3–6.
[14] D. M. Kodek, "Limits of finite wordlength FIR digital filter design," in *Proc. Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Munich, Germany, Apr. 21–24, 1997, vol. III, pp. 2149–2152.
[15] D. M. Kodek, "Design of optimal finite wordlength FIR digital filters," in *Proc. Eur. Conf. Circuit Theory Des. (ECCTD)*, Stresa, Italy, Aug. 29–31, 1999, vol. I, pp. 401–404.
[16] D. M. Kodek, "Performance limit of finite word-length FIR digital filters," *IEEE Trans. Signal Process.*, vol. 53, pp. 2462–2469, Jul. 2005.
[17] L. J. Karam and J. H. McClellan, "Complex Chebyshev approximation for FIR filter design," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 207–216, Mar. 1995.
[18] T. W. Parks and J. H. McClellan, "A program for the design of linear phase finite impulse response filters," *IEEE Trans. Audio Electroacoust.*, vol. 20, pp. 195–199, Aug. 1972.
[19] M. J. D. Powell, *Approximation Theory and Methods*. Cambridge, U.K.: Cambridge Univ. Press, 1981, pp. 97–99.
[20] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Math. Ann.*, vol. 261, pp. 515–534, 1982.
[21] D. Micciancio and S. Goldwasser, *Complexity of Lattice Problems: A Cryptographic Perspective*. Boston, MA: Kluwer Academic, 2002, pp. 45–68.
[22] N. P. Smart, *The Algorithmic Resolution of Diophantine Equations*. Cambridge, U.K.: Cambridge Univ. Press, 1998, pp. 59–76.
[23] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*. Englewood Cliffs, NJ: Prentice-Hall, 1982, pp. 433–444.
[24] D. M. Kodek and M. Krisper, "Telescopic rounding for suboptimal finite wordlength FIR digital filter design," *Digit. Signal Process.*, vol. 15, pp. 522–535, Nov. 2005.

Reconstruction of Uniformly Sampled Sequence From Nonuniformly Sampled Transient Sequence Using Symmetric Extension

Sung-Won Park, Wei-Da Hao, and Chung S. Leung

***Abstract*—In this correspondence, reconstruction of a uniformly sampled sequence from a nonuniformly sampled transient sequence using symmetric extension is described. First, a relationship between the discrete Fourier transform (DFT) of a uniformly sampled sequence and the DFT of a nonuniformly sampled sequence is obtained. From the relationship, the formula to reconstruct the DFT of a uniformly sampled sequence from the DFT of a nonuniformly sampled sequence is derived when the nonuniform sampling ratios are known. Second, a symmetric extension of the nonuniformly sampled sequence is described to avoid discontinuity that adds high-frequency content in the DFT. Finally, experimental results are presented.**

***Index Terms*—Discrete Fourier transform (DFT), nonuniform sampling, symmetric extension, uniform sampling.**

I. Introduction

In many applications, it is desired to reconstruct a uniformly sampled sequence from a nonuniformly sampled sequence. Early works on recurrent nonuniform sampling are available in the literature [1]–[7]. Recurrent nonuniform sampling means that a continuous-time signal is sampled nonuniformly with a periodic pattern. Recurrent nonuniform sampling problems occur in sampling of a high frequency signal using a very high-speed waveform digitizing system with interleaved A/D converters [2], [3]. Early works on recurrent nonuniform sampling considered sampling of relatively long signals. For example, the number of samples considered in [4] was 512 and the number of samples considered in [5] was 2048. Recurrent nonuniform sampling has also been applied to image enhancement where resolution of an image is increased beyond the number of pixels available in the camera by using multiple aliased copies of unknown relative sampling offsets [6]. On the other hand, recurrent nonuniform sampling described in [2] and [4] has been extended to two-dimensional signals [7]. There are two main differences between [6] and [7]. There is a constraint in the nonuniform sampling ratios in [7] and there is no such a constraint in [6]. Nonuniform sampling ratios are estimated from multiple copies in [6] by iteration, but they are estimated by using a prescribed sinusoidal signal in [7] without iteration.

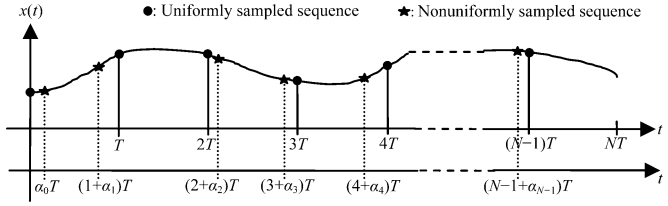

Fig. 1. Uniform and nonuniform sampling. T is the sampling interval for uniform sampling, α_n is the nonuniform sampling ratio, and N is the number of samples.

Reconstruction of uniformly sampled sequence from nonuniformly sampled sequence without any periodic pattern is described in this correspondence. This reconstruction technique is useful when one is interested in reconstructing a short transient sequence. A short sequence, whose length is only 20, is considered in this correspondence.

The correspondence is organized as follows. In Section II, a relationship between the DFT of a uniformly sampled sequence and the DFT of a nonuniformly sampled sequence is obtained. From the relationship, the formula to construct the DFT of a uniformly sampled sequence from the DFT of a nonuniformly sampled sequence is derived when the nonuniform sampling ratios are known. In Section III, symmetric extension of a sequence to avoid a discontinuity that unduly adds high frequency content in the DFT is explained. In addition, results of reconstruction experiments using no extension and symmetric extension are presented. Finally, a conclusion is made in Section IV.

II. Relationship Between Uniform Sampling and Nonuniform Sampling

Suppose a continuous-time signal, $x(t)$, is sampled uniformly at $t = 0, T, 2T, \ldots, (N-1)T$ where T is the sampling interval (see Fig. 1).

The DFT of the uniformly sampled sequence, $x(n)$, for $n = 0, 1, 2, \ldots, N-1$, is given by

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn} \text{ for } k = 0, 1, 2, \ldots, N-1 \qquad (1)$$

where $x(n) = x(nT)$ for all n. The IDFT is given by

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn} \text{ for } n = 0, 1, 2, \ldots, N-1. \qquad (2)$$

Manuscript received August 18, 2011; revised October 26, 2011; accepted November 17, 2011. Date of publication December 14, 2011; date of current version February 10, 2012. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. XiangGen Xia.

The authors are with the Department of Electrical Engineering and Computer Science, Texas A&M University-Kingsville, Kingsville, TX 78363 USA (e-mail: park@tamuk.edu; wei-da.hao@tamuk.edu; chung.leung@tamuk.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2011.2177834