

A New Algorithm for Designing FIR-Filters with Small Word Length

Nitin Ahuja¹, Ulrich Heute², Christian Richter¹, and Anand Srivastav¹

¹ Mathematisches Seminar
Christian-Albrechts-Universität zu Kiel
Ludewig-Meyn-Str. 4, 24098 Kiel, Germany.
{nia,chr,asr}@numerik.uni-kiel.de

² Institute for Circuits and System Theory
Technische Fakultät
Christian-Albrechts-Universität zu Kiel, Kiel, Germany.
uh@tf.uni-kiel.de

Abstract. We present a new algorithm for designing FIR-filters with small word-lengths. The core of our algorithm is a potential function taken from the specific area of randomized algorithms used in combinatorial optimization. We formulate the problem of finding quantized, integral filter coefficients such that the deviation from the desired objective function is minimum as an integer program. We then allow the coefficients in this integer program to take real values (LP-relaxation) and round these real coefficients to obtain integral coefficients with the help of the new potential function. The experimental results show that compared to previous known algorithms our algorithm gives a better solution while taking less time.

1 Introduction

As its name suggests, a filter is an operator which acts on an input signal consisting of multiple frequencies and blocks certain frequencies while allowing certain other frequencies to pass through as output. Our main concern in this article is designing a specific kind of digital or discrete filter. For digital filters the input signal is not a continuous analog signal but a discrete time signal. A discrete time signal is a finite or infinite sequence of numbers $\{x(k)\}$ (or $\{x_k\}$). One can imagine these numbers to be the result of sampling a continuous signal with sampling period T . From now on in this article discrete time signal(s) will be referred to as signal(s) unless specified otherwise. Signals are expressed differently in three different domains. In time domain a signal is a finite or infinite vector $x = (\dots, x(-1), x(0), x(1), \dots)$, in frequency domain a signal x is written as

$$X(e^{j\omega}) = \sum_k x(k)e^{-j\omega k} \quad \text{or} \quad X(\omega) = \sum_k x(k)e^{-i\omega k},$$

and in z -domain a signal x is expressed as $X(z) = \sum_k x(k)z^{-k}$.

A digital filter is a combination of matrices called *delays* and *advances*. A delay S acts on a signal $x = (\dots, x(-1), x(0), x(1), x(2), \dots)$ and causes a delay or shift of one unit, *i.e.* $Sx = (\dots, x(-2), x(-1), x(0), x(1), \dots)$. An advance is an inverted delay, *i.e.* S^{-1} , and it does just the opposite of a delay. As mentioned before, a digital filter H can be expressed as

$$H = \sum_k h(k)S^k$$

in time domain and as

$$H(e^{j\omega}) = \sum_k h(k)e^{-j\omega k}$$

in frequency domain. The filter is completely determined by its coefficients $h(k)$. A filter with $h(k) = 0 \forall k < 0$ is called causal filter. When $-m_1 \leq k \leq m_2$, $m_1, m_2 \in \mathbb{N}$, then filter H is called FIR (Finite Impulse Response) filter. For FIR filters the output signal (or impulse response) in time domain is given by the following formula

$$y(k) = \sum_{i=-m_1}^{m_2} h(i)x(k-i),$$

for $-m_1 \leq k \leq m_2$. In frequency domain the output signal is expressed as $Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega})$.

The ultimate aim is to assign integral values to filter coefficients $h(k)$ such that the resulting filter blocks certain pre-specified frequencies and allows other frequencies to pass through to the output. In this article we design lowpass FIR filters with positive symmetry and small filter length. This means we assume that $m_1 = m_2 = m \in \mathbb{N}$ and $h(-k) = h(k)$ for all $k \in \{-m, \dots, -1, 1, \dots, m\}$. Note that filter length for such filters is $2m+1$. A lowpass filter allows frequencies in passband $[0, \omega_p]$ to pass through while blocking the frequencies in stopband $[\omega_s, \pi]$. Between ω_p and ω_s is the transition band. For ideal lowpass filters $\omega_p = \omega_s$. In other words, for a lowpass filter

$$H(e^{j\omega}) = \begin{cases} 1 & \text{if } 0 \leq |\omega| \leq \omega_p \\ 0 & \text{if } \omega_s \leq |\omega| \leq \pi. \end{cases} \quad (1)$$

Filter coefficients which produce this response as in (1) can be obtained as follows:

$$h(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega})e^{j\omega k} d\omega, \quad k \in \{-\infty, \dots, -1, 0, 1, \dots, \infty\}. \quad (2)$$

But this method gives us infinite filter coefficients and such a filter is not realizable in practice. This fact stresses the importance of designing a lowpass FIR filter with $2m+1$ (actually m) coefficients. So in effect the aim is to approximately achieve the response (1) by designing a filter with finite filter length of $2m+1$. One way of doing this is to consider the coefficients $h(-m), \dots, h(m)$

obtained from (2) and discard the rest of them. The problem is that in doing so we get a filter response which is far worse than the response in (1).

There are different criteria to decide how far does a filter with finite filter length deviate from the ideal behaviour defined in equation (1). For this purpose one has to define a suitable error function [2]. Usually, for filter design problems one tries to minimize the maximum deviation of the behaviour of the filter designed from the ideal behaviour (min-max criteria). We also use this criteria which will be defined properly in Section 2. For this purpose, the most commonly used methods or algorithms are the McClellan-Parks algorithm (MP or Remez method [2, 4]) and Linear Programming based methods [1]. We give a small comparison of these two methods in the third section.

In order to design efficient and fast filters that work in practice the filter coefficients should have short word lengths (number of bits). Therefore after computing real filter coefficients the next step is to bring down the word length of these coefficients such that the behaviour (output or response) of this filter does not deteriorate substantially. This is achieved by scaling up the filter coefficients, rounding them to an integer and then scaling them down. In [2] and [4] after computing the real filter coefficients efficiently, naive rounding is used to round the real values to integral ones. Their main concern is computing the filter coefficients and not the scaling up-rounding-scaling down process for decreasing the word length. Qi [1], in his thesis, uses linear programming to obtain real filter coefficients and then applies a branch&bound algorithm to obtain the optimum integral solution. The running time of this algorithm is exponential in the number of coefficients involved and thus this method is not suitable for designing filters with relatively large filter lengths.

The structure of this article is as follows. In Section 2 we formulate the FIR filter design problem as a mixed integer linear program (MILP). We also give the LP-version of this problem and define an error function which will often serve as the objective function that has to be minimized. In Section 3 we give a comparison of Matlab and CPLEX procedures by applying them to the same test filters. The aim is to show that solving MILP directly does not help much. The integral filter coefficients are obtained in Section 4. Section 5 includes certain experimental data for comparing the techniques of Section 4. In Section 6 we apply some more optimization techniques to improve the solution obtained and we end with some open questions and comments in Section 7.

2 Linear Programming Formulation

In this section we cast the filter design problem as a linear program [1]. This formulation also leads to defining certain error functions used in evaluating the quality of our filter. These error functions will also play an important role later

in our rounding strategy.

We consider the normalised frequency in interval $[0, \pi]$. As already mentioned in (1) the optimal or desired filter response (or behaviour) $D(\omega)$ is:

$$D(\omega) = \begin{cases} 1 & : \omega \in [0, \omega_p] ; \\ 0 & : \omega \in [\omega_s, \pi] . \end{cases} \quad (3)$$

The objective is to design a filter with filter response $H(\omega)$ as close to $D(\omega)$ as possible in the min-max sense. Let us denote the tolerance in passband and stopband by δ_p and δ_s respectively. This means the response of our filter can deviate from the optimal response by at most δ_p in passband and at most δ_s in stopband. In other words:

$$|H(\omega) - D(\omega)| \leq \begin{cases} \delta_p & \text{for } \omega \in [0, \omega_p] ; \\ \delta_s & \text{for } \omega \in [\omega_s, \pi] . \end{cases}$$

δ_p and δ_s can be combined with the help of a weight function $R(\omega)$ to give a total error tolerance δ :

$$|H(\omega) - D(\omega)| \leq R(\omega)\delta .$$

To get a hold on the continuous frequency bands we discretize the entire frequency range $[0, \pi]$ into a grid (or support) of $N + 1$ frequencies ω_i for $i = 0, \dots, N$. Thus, we get $N + 1$ inequalities $|H(\omega_i) - D(\omega_i)| \leq R(\omega_i)\delta$. Since we want to have linear constraints this set of inequalities yields $2(N + 1)$ constraints

$$H(\omega_i) - R(\omega_i)\delta \leq D(\omega_i) \quad \text{and} \quad -H(\omega_i) - R(\omega_i)\delta \leq -D(\omega_i) , \quad (4)$$

for $i = 0, \dots, N$. The number of support points is approximately eight times the filter length.

Recall that our objective is to first find the real filter coefficients $h(k)$, $-m \leq k \leq m$ such that our filter closely imitates the optimal filter. In order to achieve this we take the Z -transform of the filter coefficients, $h(k)$, in time domain with normalised frequency ωT and get

$$\tilde{H}(e^{j\omega T}) = \sum_{k=-m}^m h(k)e^{-j\omega T k} . \quad (5)$$

Due to the positive symmetry of filter coefficients $h(k)$ the real part of the Z -transform above comes out to be

$$H(\omega T) = h(0) + \sum_{k=1}^m 2h(k) \cos(\omega T k) .$$

This real part is now used to approximate the optimal filter response (3) of a lowpass filter. Just like in [1], at discrete support points ω_i , $i = 0, \dots, N$ the real part becomes

$$H(\omega_i) = h(0) + \sum_{k=1}^m 2h(k) \cos\left(\frac{\pi}{N}ik\right).$$

On substituting $H(\omega_i)$ in inequalities (4) we get:

$$\begin{aligned} h(0) + \sum_{k=1}^m 2h(k) \cos\left(\frac{\pi}{N}ik\right) - R(\omega_i)\delta &\leq D(\omega_i), \\ -h(0) - \sum_{k=1}^m 2h(k) \cos\left(\frac{\pi}{N}ik\right) - R(\omega_i)\delta &\leq -D(\omega_i). \end{aligned}$$

It is now easy to cast the problem as a MILP (Mixed Integer Linear Program) as follows

MILP Filter design:

Minimize δ such that:

$$\begin{aligned} Ax &\leq b, \text{ where } x = (h(0), h(1), \dots, h(m), -\delta)^T, \\ h(k) &\in \mathbb{Z} \quad \forall k = 0, \dots, m, \text{ and } \delta \in \mathbb{R} \end{aligned}$$

Here $b = (D_0, \dots, D_N, -D_0, \dots, -D_N)^T$,

$$A = \begin{pmatrix} 1/s & \frac{2}{s} & \dots & \frac{2}{s} & R_0 \\ 1/s & \frac{2}{s} \cos \frac{\pi}{N} 1 & \dots & \frac{2}{s} \cos \frac{\pi}{N} m & R_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1/s & \frac{2}{s} \cos \frac{\pi}{N} N & \dots & \frac{2}{s} \cos \frac{\pi}{N} Nm & R_N \\ -1/s & -\frac{2}{s} & \dots & -\frac{2}{s} & R_0 \\ -1/s & -\frac{2}{s} \cos \frac{\pi}{N} 1 & \dots & -\frac{2}{s} \cos \frac{\pi}{N} m & R_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -1/s & -\frac{2}{s} \cos \frac{\pi}{N} N & \dots & -\frac{2}{s} \cos \frac{\pi}{N} Nm & R_N \end{pmatrix}_{2(N+1) \times (m+2)} \quad (6)$$

and for $i = 0, \dots, N$, D_i , R_i are $D(\omega_i)$ and $R(\omega_i)$ respectively. The parameter s is a constant scaling factor which will be of use later while scaling the coefficients and applying the rounding technique.

It is hard to solve a MILP quickly and efficiently but a linear program (LP) can be solved relatively quickly, for instance by using the interior-point method of Karmarkar or an efficiently implemented simplex algorithm. This encourages us to drop the integral constraints in our MILP, solve the resulting LP quickly, and then round the LP-solution quickly to get an integral solution. So, after relaxing the integral constraints in the MILP we get the following linear program:

LP Filter design:

Minimize δ^* such that

$$Ay \leq b, \quad y = (g(0), g(1), \dots, g(m), -\delta^*)^T, \quad \text{and } y \in \mathbb{R}^{m+2}$$

Thus, for given values of filter length, passband, stopband and weight function $R(\omega)$ we can solve the LP to minimize δ^* . As a result we get real coefficients with values between -1 and 1 .

After optimally solving the LP, the next step is to scale up and round the obtained real filter coefficients. But if, after scaling them up, we change their values then we may end up violating many inequality constraints of our linear program. This means there is a risk of deviating from the optimal response D_i by more than δ . So, we define a slightly modified error tolerance. Let $A = (a_{q,k+1})$ and $R'_q = a_{q,m+2}$ where $1 \leq q \leq 2(N+1)$ and $0 \leq k \leq m$. For a vector $h = (h(0), \dots, h(m)) \in \mathbb{Z}^{m+1}$ we define the error tolerance as

$$\delta(h) := \max_{q \in \{1, \dots, 2N+2\}} \frac{\sum_{k=0}^m a_{q,k+1} h(k) - b_q}{R'_q}. \quad (7)$$

It is clear that with this error tolerance all inequality constraints of our MILP are satisfied.

The disadvantage of the error (tolerance) function defined in (7) is that just one gravely violated constraint can substantially increase its value although the values of the corresponding filter coefficients might lead to a better solution later or the violated constraint might be reparable by slightly changing the values of certain other filter coefficients. On the other hand the error function increases slightly even when many constraints are violated, albeit slightly. In the context of local search algorithms which we intend to apply after our rounding procedure these anomalies of the error function are not desirable. It makes sense to use an error function which takes into account the number of violated constraints. We will define such an error function later.

Our algorithm is as follows:

1. Find the optimal solution $y = (g, -\delta^*) \in \mathbb{R}^{m+2}$ of the LP Filter design.
2. Round the LP solution $g = (g(0), \dots, g(m))$ to an integral one with the help of the potential function (described later).
3. Lastly, use (7) to find the maximum deviation of the response of our filter from the optimal response.

In Section 4.3 we will describe a modification of this algorithm.

Filter length ($2M+1$)	Matlab [Runtime(sec); Error(-dB)]	CPLEX [Runtime(sec); Error(-dB)]
21	0 ; 54	0 ; 55
41	0 ; 58	0 ; 59
61	0 ; 64	1 ; 64
81	0 ; 71	2 ; 71
101	0 ; 78	4 ; 78
121	0 ; 85	8 ; 85
141	0 ; 92	20 ; 91
161	1 ; 99	38 ; 96
181	1 ; 107	98 ; 97
201	1 ; 113	124 ; 98

Table 1. Comparison of running time and error tolerance obtained

3 Calculating Real Filter Coefficients

The software package MATLAB contains a convenient tool (function) for calculating filter coefficients. This tool or function uses the McClellan-Parks algorithm for calculating the coefficients and can be found in the *Signal Processing Toolbox*. For a given filter design problem filter coefficients can also be calculated by using the linear programming based approach. We used the software package CPLEX to calculate the coefficients. This package is specially designed to solve linear (or integer linear) programs and it is not a specialized filter design tool. The input and output commands were all written in the programming language C.

Table 1 shows a small comparison of both methods. As test (optimal) filter we took an (anticausal) lowpass filter with passband $[0, 0.25\pi]$, stopband $[0.3\pi, \pi]$ and transition band $(0.25\pi, 0.3\pi)$. The weight function was of the order of 500 : 1 which means we allowed the response of our filter to deviate from the optimal response 500 times more in passband than in stopband. Table 1 shows the damping in stopband and the running time needed to design a filter with the corresponding filter length. Here damping is measured in decibels (dB) and is related to the error tolerance δ as follows: $\delta[\text{dB}] = 20 \log_{10} \delta$. We ran both algorithms on a Sun Ultra Sparc 5 machine.

It seems that in comparison to the CPLEX routine MATLAB function works better for computing filter coefficients of a simple lowpass filter. On the other hand linear programming approach enables us to easily add some more constraints on filter design if the need arises. For filter lengths of 140 onwards one can clearly see (Table 1) the effect of large numerical problems, which have to be solved by CPLEX, in the form of rapidly increasing running time. For larger filter lengths the running time of CPLEX method increases substantially coupled with a very slight reduction in stopband error. A similar phenomenon occurs with the Matlab routine for filter lengths of more than 460. Another interesting observation is that although Matlab routine takes less time, for filter lengths of

120 onwards error values given by CPLEX method are better (comparatively less).

4 Obtaining Integral Filter Coefficients

4.1 Scaling

Let $(g(0), \dots, g(m)) \in \mathbb{R}^{m+1}$ and $\delta^* \in \mathbb{R}^+$ be the optimal solution of the LP filter design. The next step is to round the obtained coefficients $g(k)$ to get filter coefficients of a pre-specified word length such that the response of our filter does not deteriorate substantially. With the help of scaling factor s we will then up-scale the coefficients, round them and down-scale them again to achieve the pre-specified word length.

CPLEX has an option of applying a branch-and-bound algorithm after solving a linear program to get the optimal integral solution. Using this option and through proper scaling we could really find the optimal integral solution with specified word lengths but only for filters with very short filter length. For filter lengths of 30 onwards the running time of this combined procedure starts increasing substantially and the quality of the solutions begins to deteriorate. This motivates us to look for approximate filter coefficients, thereby taking less time, rather than the optimal ones.

To fix the scaling factor s we solve the LP with $s = 1$. Let us denote the optimal solution of this LP by $g_0(k)$ where $k = 0, \dots, m$. Also, let w be the given word length (total number of bits to be used). The scaling factor is chosen in such a manner that the largest among the absolute values $|g_0(k)|$ of filter coefficients uses up the complete word length:

$$s = \frac{2^{w-1} - 1}{\max_{k \in \{0, \dots, m\}} |g_0(k)|} . \quad (8)$$

Using this scaling factor we scale-up the solution $g_0(k)$, *i.e.*, we set $g(k) = sg_0(k)$ for all $k = 0, \dots, m$. Note that the largest filter coefficient, in the absolute sense, occupies the complete word length and it is integral. This coefficient is now set and it will not be modified any more to preserve the word length constraint.

4.2 Randomized Rounding

Usually the scaled up real coefficients $g(k)$ are naively rounded to integral coefficients $h(k)$ as follows:

$$h(k) = \begin{cases} \lfloor g(k) + 0.5 \rfloor & \text{if } g(k) \geq 0 ; \\ \lceil g(k) - 0.5 \rceil & \text{if } g(k) < 0 , \end{cases}$$

for $k = 0, \dots, m$. We use randomized rounding to round real coefficients $g(k)$. This is done as follows:

Algorithm Randomized Rounding:

1. Input: scaled up, real coefficients $g(k) \in \mathbb{R}$
2. Let $p_k := g(k) - \lfloor g(k) \rfloor \in [0, 1], \forall k = 0, \dots, m$
3. Independently for each $k \in \{0, \dots, m\}$ set

$$h(k) := \begin{cases} \lfloor g(k) \rfloor & \text{with probability } 1 - p_k \\ \lceil g(k) \rceil & \text{with probability } p_k \end{cases} \quad (9)$$

4. Output: integral coefficients $h(k) \in \mathbb{Z}$

Both rounding techniques take $O(m)$ time.

The following theorem shows that the integral coefficients $h(k)$ obtained after randomized rounding satisfy the inequality constraints in an expected sense. Let $v = (g(0), \dots, g(m), -\delta^*)^T \in \mathbb{R}^{m+1} \times \mathbb{R}^+$ and let $x = (h(0), \dots, h(m))^T \in \mathbb{Z}^{m+1}$ be the vector of coefficients to be obtained after randomized rounding. Each $h(k)$ is a random variable as defined in (9). Also let $z = (x, -\delta^*)$

Theorem 1. $\mathbb{E}(Az) \leq b$.

Proof. From the definition of $h(k)$ in (9) we have:

$$\begin{aligned} \mathbb{E}(h(k)) &= \lfloor g(k) \rfloor (1 - p_k) + \lceil g(k) \rceil p_k \\ &= \lfloor g(k) \rfloor (\lceil g(k) \rceil - g(k)) + \lceil g(k) \rceil (g(k) - \lfloor g(k) \rfloor) \\ &= g(k) , \end{aligned}$$

for all $k = 0, \dots, m$. Now, by linearity of expectation

$$\begin{aligned} \mathbb{E}((Az)_q) &= \mathbb{E}\left(\sum_{k=0}^m a_{q,k} h(k) - R'_q \delta^*\right) \\ &= \sum_{k=0}^m a_{q,k} \mathbb{E}(h(k)) - R'_q \delta^* \\ &= \sum_{k=0}^m a_{q,k} g(k) - R'_q \delta^* = (Av)_q \leq b_q , \end{aligned}$$

for all $q = 1, \dots, 2(N+1)$. This proves the theorem. \square

So, after randomized rounding we can expect good filter coefficients that satisfy all inequality constraints. But since the rounding is random, we might end up violating some constraints which in turn will give us a large value of the error tolerance function $\delta(h)$. The following theorem gives an estimate of how bad can the error tolerance function become. Before proceeding to state and prove the theorem let us set

$$\lambda = \sqrt{\frac{m+1}{2} \ln(8(N+1))} .$$

Theorem 2. *With a probability of at least $\frac{1}{2}$ the following are true*

1. $\sum_{k=0}^m a_{q,k}h(k) - R'_q\delta^* \leq b_q + \lambda$ for all $q = 1, \dots, 2(N+1)$, and
2. $\delta(h) = \delta^* + O(\sqrt{m \ln n})$.

Proof (1.) We bound the probability of the following event: after randomized rounding there is at least one inequality constraint such that

$$\sum_{k=0}^m a_{q,k}h(k) - R'_q\delta^* > b_q + \lambda .$$

We do this as follows:

$$\begin{aligned} & \Pr \left[\exists q \in \{1, \dots, 2(N+1)\} : \sum_{k=0}^m a_{q,k}h(k) - R'_q\delta^* > b_q + \lambda \right] \\ & \leq \sum_{q=1}^{2N+2} \Pr \left[\sum_{k=0}^m a_{q,k}h(k) - R'_q\delta^* > b_q + \lambda \right] \\ & \leq \sum_{q=1}^{2N+2} \Pr \left[\sum_{k=0}^m a_{q,k}h(k) - R'_q\delta^* - \left(\sum_{k=0}^m a_{q,k}g(k) - R'_q\delta^* \right) > \lambda \right] \end{aligned} \quad (10)$$

$$\leq \sum_{q=1}^{2N+2} \Pr \left[\left| \sum_{k=0}^m a_{q,k}h(k) - \mathbb{E} \left(\sum_{k=0}^m a_{q,k}g(k) \right) \right| > \lambda \right] \quad (11)$$

$$\leq \sum_{q=1}^{2N+2} \left(2 \exp \left(-\frac{2\lambda^2}{m+1} \right) \right) \quad (12)$$

$$= (4N+4) \exp \left(\ln \left(\frac{1}{8N+8} \right) \right) \quad (13)$$

$$= 0.5 .$$

Hence, the first statement follows. Here (10) and (11) follow from the details given in proof of Theorem 1 and also from the fact that lesser deviations of random variables are more likely. Inequality (12) comes from a corollary (see [6]) of a large deviation bound of Hoeffding and (13) is obtained by substituting the value of λ fixed before.

(2.) Since we have proven that with probability at least 0.5

$$\sum_{k=0}^m a_{q,k}h_k - R'_q \left(\delta^* + \frac{\lambda}{R'_q} \right) \leq b_q ,$$

for all $q = 1, \dots, 2(N+1)$. This gives us a new bound for $\delta(h)$,

$$\delta(h) = \delta^* + \max_{1 \leq q \leq 2N+2} \frac{\lambda}{R'_q} = \delta^* + O(\sqrt{m \ln N}) \quad (14)$$

because R'_q is a pre-specified constant. \square

This bound seems pretty large and it might mislead one into thinking that randomized rounding produces large errors. But, this is a theoretical upper bound and our experimental experience showed us that in practice this error is not large although error tolerance values obtained by naive rounding are comparatively better.

One can see λ as the extent to which a constraint can be violated after randomized rounding. So, we would like λ to be as small as possible. Exactly this is what we attempt to do in the next section.

4.3 Potential Function and Improved Rounding

In Section 2 we defined an error tolerance function $\delta(h)$ (see equation (7)) for $h = (h(0), \dots, h(m)) \in \mathbb{Z}^{m+1}$. The minimum value of this function is $\delta(g) = \delta^*$ where $g = (g(0), \dots, g(m)) \in \mathbb{R}^{m+1}$ is the optimal solution of LP:filter design. In this section we define a new kind of error tolerance function which also takes into consideration the number of violated constraints during the rounding process. We call it potential function.

Again, let $A = (a_{q,k+1})$ and $R'_q = a_{q,m+2}$ where $1 \leq q \leq 2(N+1)$ and $0 \leq k \leq m$. Let δ^* be the value of the optimal solution g of LP and let h be a vector of rounded filter coefficients. Set

$$S_q(h) = \sum_{k=0}^m a_{q,k} h(k) - R'_q \delta^* - b_q, \quad (15)$$

for all $1 \leq q \leq 2(N+1)$. It is clear that the q -th constraint is violated if and only if $S_q(h) > 0$ and similarly the q -th constraint is satisfied if and only if $S_q(h) \leq 0$. Now, if we sum up $S_q(h)$ for all $2N+2$ constraints then we get a potential function which gives us a rough idea of how many constraints are yet to be satisfied. In fact, in $O(Nm)$ time we can find out which constraints are yet to be satisfied and sum up the corresponding $S_q(h)$. For this purpose let $I \subseteq \{1, \dots, 2(N+1)\}$ be the set of indices of violated constraints. We are now ready to define our potential function:

$$S(h) = \sum_{q \in I} S_q(h). \quad (16)$$

This potential function takes into account not only the number of violated constraints but also the magnitude of violation of each of these constraints. In other words, this function is different than the modified error tolerance $\delta(h)$ defined in Section 2 in the sense that here the error tolerance δ^* remains fixed and our main interest is to reduce the number of constraints violated while performing randomized rounding.

We will now use the potential function to improve the randomized rounding procedure. In this process real filter coefficients will be rounded up or down to integers sequentially depending on the potential function $S(h)$ as follows:

Algorithm: Improved Rounding

1. Input: real optimal filter coefficients $g(i) \in \mathbb{R}$, matrix A , vector b and error tolerance δ^* corresponding to g .

2. Set $l := 0$

3. Set

$$h_1 := (h(0), \dots, h(l-1), \lceil g(l) \rceil, g(l+1), \dots, g(m))$$

$$h_2 := (h(0), \dots, h(l-1), \lfloor g(l) \rfloor, g(l+1), \dots, g(m))$$

4. Set

$$h(l) := \begin{cases} \lceil g(l) \rceil & \text{if } S(h_1) \leq S(h_2) \\ \lfloor g(l) \rfloor & \text{if } S(h_1) > S(h_2) \end{cases} \quad (17)$$

5. $l := l + 1$. If $l \leq m$ then go to 3.

6. Output: Integral coefficients $h(k) \in \mathbb{Z}, k \in \{0, \dots, m\}$

This rounding technique can be extended further. In the new version we consider four new possibilities, namely $\lceil g(l) \rceil$, $\lfloor g(l) \rfloor$, $\lceil g(l) \rceil + 1$ and $\lfloor g(l) \rfloor - 1$, instead of just two. We now round the real coefficient depending on which of these options yields the lowest value of $S(h)$. But one has to be careful here and prevent the rounded integral coefficients from violating the pre-specified word length constraint.

5 Comparing Rounding Algorithms

In this section we compare different algorithms for designing FIR filters. We ran the algorithms on five test filters used in standard practical applications. We are given the filter length (fl), passband (pb), stopband (sb), word length (wl) and the weight function (wf). As we mentioned before, the value r of the weight function means we allow the response of our filter to deviate from the optimal response in passband r times more than in stopband. We obtained integral filter coefficients and the corresponding error $\delta(h)$ by applying naive rounding (nr), randomized rounding (rr) and improved rounding (ir). For randomized rounding we applied or ran the algorithm several times and the corresponding error tolerance values are the means of several runs. For a filter with filter length 23 we were able to obtain the integral optimum (io) with the help of CPLEX integer programming procedure. The results are shown in Table 2.

As a result we obtained lower error values after improved rounding as compared to naive rounding. The improved rounding algorithm took at most 30 seconds for all five test filters. The improved rounding procedure has an added advantage because not only does it provide integral filter coefficients which can be used as starting points for subsequent optimization tools like local search, etc., the potential function (16) can also be used as objective function in subsequent optimization efforts.

t. filter→	MIP	Clutter1	Clutter2	Prod	Cap
fl	23	51	63	107	87
pb ω_p	$1/2\pi$	$1/8\pi$	$1/2\pi$	$1/4\pi$	$1/5\pi$
sb ω_s	$26/50\pi$	$1/4\pi$	$3/5\pi$	$1/3\pi$	$1/4\pi$
wf	10x	500x	500x	100x	100x
wl (bits)	10	16	16	12	12
$\delta(h)$ LP	0.044058	0.000070	0.000045	0.000015	0.000605
$\delta(h)$ IP (opt.)	0.044417	-	-	-	-
$\delta(h)$ nr	0.047059	0.000109	0.000125	0.000450	0.001114
$\delta(h)$ rr	~ 0.049	~ 0.00012	~ 0.00017	~ 0.00063	~ 0.0012
$\delta(h)$ ir	0.045527	0.000101	0.000089	0.000449	0.000889

Table 2. Comparing different rounding algorithms

6 Improving the Filter some more

In this section we outline four post-rounding optimization techniques. These techniques are divided in two groups, namely local and global search techniques or algorithms. The error function which we try to minimize here is $\delta(h)$ where h is the rounded vector of filter coefficients. In local search algorithms we change a set of coefficients only if it reduces the value of the error function where as in global search algorithms a slight increase in the value of the error function is also allowed. In the local search algorithms at each step we change the value of exactly one coefficient by one unit and observe its effect on $\delta(h)$. In global search algorithms at each step a coefficient is chosen at random and its value is randomly increased or decreased by one unit. At the end of this section we present a comparison of these four post-rounding optimization techniques applied to different test filters.

Fast Local Search (FLS). We are given scaled up, rounded coefficients $h(k)$, $k \in \{0, \dots, m\}$, the matrix A and the right hand side vector b of the LP discussed in Section 2.

1. Set $j = 1$, $h_j := h$ and calculate $\delta(h_j)$.
2. Set $h_j^k := (h_j(0), \dots, h_j(k-1), h_j(k) + 1, h_j(k+1), \dots, h_j(m))$ for all $k \in \{0, \dots, m\}$.

3. Set $h_j^{k+M+1} := (h_j(0), \dots, h_j(k-1), h_j(k) - 1, h_j(k+1), \dots, h_j(m))$ for all $k \in \{0, \dots, m\}$.
4. Calculate $\delta(h_j^i)$ for all $i \in \{0, \dots, 2m+1\}$.
5. Set $\delta_{min} = \min_{i \in \{0, \dots, 2m+1\}} \delta(h_j^i)$. Let i_{min} be the corresponding index.
6. If $\delta_{min} \geq \delta(h_j)$ go to 8, else continue.
7. Set $h_{j+1} := h_j^{i_{min}}$, $\delta(h_{j+1}) = \delta_{min}$ and $j := j + 1$. Go back to 2.
8. Output h_j .

As expected, this algorithm did not take much time for all test filters. The following algorithm took much longer but delivered better coefficients.

Complete Local Search (CLS). In each iteration of fast local search we took the best candidate (vector with the lowest error value) and tried to improve it further. This often led our algorithm to a local minima. In complete local search we follow all variations of coefficients which lead to a reduced error. Again we are given scaled up, rounded coefficients $h = (h(0), \dots, h(m))$, the matrix A and the right hand side vector b of the LP.

1. Set $j = 1$, $h_j = h$ and $h_{best} = h$.
2. Set $h_j^k := (h_j(0), \dots, h_j(k-1), h_j(k) + 1, h_j(k+1), \dots, h_j(m))$ for all $k \in \{0, \dots, m\}$.
3. Set $h_j^{k+M+1} := (h_j(0), \dots, h_j(k-1), h_j(k) - 1, h_j(k+1), \dots, h_j(m))$ for all $k \in \{0, \dots, m\}$.
4. Calculate $\delta(h_j^i)$ for all $i \in \{0, \dots, 2m+1\}$.
5. Set $\delta_{min} = \min_{i \in \{0, \dots, 2m+1\}} \delta(h_j^i)$. Let i_{min} be the corresponding index.
6. If $\delta_{min} \geq \delta(h_j)$ then terminate, else proceed further.
7. For every $i \in \{0, \dots, 2m+1\}$ with $\delta(h_j^i) < \delta(h_j)$ invoke the whole procedure with h_j^i .
8. Output the best vector.

Running time of this algorithm increases exponentially with linear increase in filter length so a complete execution of this algorithm is feasible just for very small filter lengths. But one can always stop the execution of this algorithm after some time and use the current best vector of filter coefficients as input to other post-rounding optimization methods.

Simulated Annealing (SA). This is a standard post-rounding search algorithm. This algorithm not only follows vectors with less error but also, with certain probability, vectors with more error value than the previous candidate. this is done to avoid getting stuck in a local minima. Parameter β_j controls the probability of accepting a bad vector h as a possible candidate for future search. This probability should decrease after each iteration, therefore, we set $\beta_j = j$. The search is aborted after J steps.

1. Input: scaled up, rounded coefficients $h = (h(0), \dots, h(m))$, the matrix A , the right hand side vector b of the LP and the maximum number of iterations allowed J .

2. Set $j = 1$, $h_j = h$.
3. Choose a number $i \in \{0, \dots, 2m + 1\}$ according to the uniform distribution.
4. If $i \leq m$ set $k = i$ and
 $h'_j := (h_j(0), \dots, h_j(k-1), h_j(k) + 1, h_j(k+1), \dots, h_j(m))$. Otherwise set
 $k = i \bmod m + 1$ and
 $h'_j = (h_j(0), \dots, h_j(k-1), h_j(k) - 1, h_j(k+1), \dots, h_j(m))$.
5. **If** $\delta(h'_j) < \delta(h_j)$ **then** set $h_{j+1} = h'_j$ and $j = j + 1$, further if $j \leq J$ then go to 3 else go to 9. **Else** continue.
6. Choose a random number R uniformly from $[0, 1]$.
7. If $R < e^{-\beta_j(\delta(h'_j) - \delta(h_j))}$ then set $h_{j+1} = h'_j$ otherwise set $h_{j+1} = h_j$. Set $j = j + 1$.
8. If $j \leq J$ go back to 3.
9. Output h_j .

Running time of simulated annealing algorithms is pretty high and the quality of solutions we obtained for different test filters was not better than what we obtained with fast local search (see Table 3). The next technique improves on simulated annealing and delivers better solutions.

T.filter→	MIP	Clutter1	Clutter2	Prod	Cap
fl	23	51	63	107	87
pb ω_p	$1/2\pi$	$1/8\pi$	$1/2\pi$	$1/4\pi$	$1/5\pi$
sb ω_s	$26/50\pi$	$1/4\pi$	$3/5\pi$	$1/3\pi$	$1/4\pi$
wt. func.	10x	500x	500x	100x	100x
wl (bits)	10	14	12	14	12
$\delta(h)$ LP	0.044058	0.000070	0.000045	0.000015	0.000605
$\delta(h)$ ir	0.045527	0.000101	0.000449	0.000203	0.002609
$\delta(h)$ FLS	0.045527	0.000090	0.000320	0.000152	0.001660
run. time	0 sec	2 sec	3 sec	8 sec	6 sec
$\delta(h)$ CLS	0.045527	0.000087	0.000232	≤ 0.000124	≤ 0.001433
run. time	0 sec	18 sec	64 sec	> 48 h	> 48 h
$\delta(h)$ SA	0.045527	0.000101	0.000298	0.000170	0.001652
run. time	4 min	18 min	26 min	66 min	47 min
$\delta(h)$ Comb.	0.045527	0.000084	0.000252	0.000135	0.001482
run. time	12 min	52 min	81 min	199 min	137 min

Table 3. Comparing post-rounding optimization algorithms

Combined Simulated Annealing and Fast Local Search (Comb.).

We modify the simulated annealing algorithm as follows: every time we accept a bad vector instead of an improved vector we perform a fast local search and save the results. For the same number of iterations J the combination of SA and FLS gave better results as compared to simple SA algorithm. Infact the quality of results given by the combined method starts improving in the initial iterations itself. The results of several experiments are shown in Table 3.

7 Open Questions

We feel that by fine-tuning the parameters of evolutionary search algorithms one can obtain better results.

References

1. H. Qi: "Entwurf von FIR-Filtern mit extremen Wortlängen-Beschränkungen", Aachen: Shaker, 1996
2. S.K. Mitra: "Digital signal processing: a computer-based approach", Boston: McGraw-Hill/Irwin, 2001
3. J.D. Broesch: "Digitale Signal-Verarbeitung", Aachen: Elektor-Verl., 1999
4. J.H. McClellan, T.W. Parks: "A program for the design of linear phase finite impulse response digital filters", IEEE Trans. Audio Electroacoust., Vol. 20, 1972
5. P.J.M van Laarhoven, E.H.L Aarts: "Simulated annealing: theory and applications", Dordrecht: Kluwer, 1992
6. N. Alon, J.H.Spencer: "The probabilistic method", New York: Wiley, 2000
7. P. Raghavan, C.D. Thompson: "Randomized rounding: a technique for provably good algorithms and algorithmic proofs", in "Combinatorica" Vol 7, Berlin: Springer, 1987
8. A. Srivastav, P. Stangier: "Algorithmic Chernoff-Hoeffding inequalities in integer programming", in "Random Structures & Algorithms" Vol 8 No 1, New York: Wiley, 1996
9. A. Srivastav: "Derandomization in combinatorial optimization", in "Handbook of Randomization", Chapter 18, Vol II, pp. 731-842, S.Rajasekaran, M. Pardalos, J.H. Reif, J. Rolin (Eds.), Dordrecht: Kluwer, 2001