SYNTHESIS OF LOW-SENSITIVITY DIGITAL FILTERS USING GENETIC PROGRAMMING WITH AUTOMATICALLY DEFINED FUNCTIONS

Kazuyoshi Uesaka and Masayuki Kawamata

Graduate School of Engineering, Tohoku University
Aoba-yama 05, Sendai 980-8579, JAPAN
Phone:+81-22-217-7095, Fax:+81-22-263-9169
Email: uesaka@tk.ecei.tohoku.ac.jp, kawamata@ecei.tohoku.ac.jp

ABSTRACT
This paper proposes a synthesis method for low coefficient sensitivity IIR digital filter structures using Genetic Programming with Automatically Defined Functions (GP-ADF). In this paper, digital filter structures are represented as S-expressions with subroutines. Since the representation is suitable for GP, it is easy to generate syntactically valid S-expressions and perform the genetic operations. In numerical examples, we use the fitness measure including the magnitude sensitivity, and demonstrate that the proposed method can synthesize efficiently very low coefficient sensitivity filter structures.

1. INTRODUCTION

The synthesis of a digital filter is the process of converting the transfer function into a digital filter structure. In the implementation of digital filters in special-purpose hardware or in software on a general-purpose computer, the filter structure corresponds to a hardware configuration or a specification of an algorithm. There are an unlimited number of equivalent filter structures realizing a given transfer function [1], and they may have different characteristics such as coefficient sensitivity, round-off noise, power consumption, and execution speed. Thus it is of interest to choose the filter structure that provides satisfactory performance, depending on the type of digital filter being implemented.

Genetic Programming (GP) has been applied to the synthesis of low coefficient sensitivity digital filters [2, 3]. The GP-based synthesis can search low coefficient sensitivity digital filter structures by representing filter structures as S-expressions and by using the coefficient sensitivity as the fitness measure. However the GP-based synthesis in [3] uses the complicated way of representing the filter structures. For this reason, for example, crossover requires the post-process to generate the syntactically valid S-expressions.

In this paper, we propose a simple way of representing the digital filter structures using the S-expressions with subroutines. The new representation is suitable for GP and does not require the post-process of crossover.

2. GENETIC PROGRAMMING WITH AUTOMATICALLY DEFINED FUNCTIONS

2.1. Genetic Programming

Genetic Programming is a global optimization technique based on the rules of natural selection and natural genetics [4, 5]. GP is an iterative procedure which maintains a constant size population of computer programs representing candidate solutions to a problem. In GP, the computer programs are typically encoded as S-expressions which are the syntactic forms in Lisp programming language [6]. GP has been applied to solve the wide range of problems, e.g. automatic programming generation and synthesis of analog circuits.

Figure 1 shows the basic structure of GP. GP starts with the generation of an initial population which consists of a fixed number of computer programs (Figure 1: line 4). The initialization procedure creates the computer programs randomly. The next step, evaluation procedure, performs all the programs in the population and gives the fitness values to them (Figure 1: line 5). On the basis of the given fitness values, selection operation copies computer programs according to their fitness values, and thus the individual program with a higher fitness value has a higher probability of giving one or more offspring programs in the next generation (Figure 1: line 8). The number of computer programs after selection must be the same as that of computer programs before selection. After selection, new individuals are generated by applying genetic operations.
procedure Genetic Programming;
begin
  generation := 0;
  initialization();
  evaluation();
  repeat
    generation := generation + 1;
    selection();
    genetic_operations();
    evaluation();
  until generation > N_{gen}
end.

Figure 1: Basic Structure of GP.

Figure 2: Example of the S-expression with two subroutines.

to the population (Figure 1: line 9). The genetic operations consist of crossover, mutation, and inversion. These operations are applied to the computer programs in the population probabilistically. Again, evaluation procedure performs the computer programs changed by the genetic operation, and then gives the new fitness values to them (Figure 1: line 10).

The cycle of selection, genetic operations, and fitness evaluation is referred to as the generation, and it is repeated until a predefined number of generation $N_{gen}$ is reached. The program with the highest fitness in the final population is considered as the desired program, i.e. the solution to the problem.

2.2. GP with Automatically Defined Functions

GP with Automatically Defined Functions (GP-ADF) can search the S-expressions with subroutines and solve more complex problems than those solved by plain GP [7]. In GP-ADF, the S-expression consists of one result-producing branch and one or more function-defining branches. Figure 2 shows an example of the S-expression with two subroutines at the tree level. This S-expression consists of one result-producing branch and two function-defining branches joined by the connective PROGN function. The first function-defined branch is rooted at ADF0 and the second is ADF1. The function PROGN sequentially evaluates its arguments of sub S-expressions and returns the value of its final argument.

Table 1: Primitive functions for the S-expression.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>multiplier</td>
</tr>
<tr>
<td>$a$</td>
<td>two-input adder</td>
</tr>
<tr>
<td>$d$</td>
<td>unit delay</td>
</tr>
<tr>
<td>$x$</td>
<td>input of the filter</td>
</tr>
<tr>
<td>$y$</td>
<td>output of the filter</td>
</tr>
<tr>
<td>$b_k$</td>
<td>$k^{th}$ node in the filter</td>
</tr>
</tbody>
</table>

Table 2: Program architecture for representing filter structures.

| Function-name set for the function-defining branches | $b_0, b_1, \cdots, b_{N_b-1}$ |
| Argument set for the function-defining branches | $x, b_0, b_1, \cdots, b_{N_b-1}$ |
| Function set for the function-defining branches | $m, a, d$ |
| Label for the result-producing branch | $y$ |
| Argument set for the result-producing branch | $x, b_0, b_1, \cdots, b_{N_b-1}$ |
| Function set for the result-producing branch | $m, a, d$ |

3. SYNTHESIS OF DIGITAL FILTERS USING GP-ADF

3.1. Representation

The realization of a filter structure requires three basic elements: adders, multipliers, and unit delays. In this paper, the S-expression representing the filter structure is also composed of these functions. Tables 1 and 2 summarize the program architecture for representing the filter structures using the S-expression with subroutines. This representation is suitable for GP compared with the representation used in the other GP-based synthesis [3] because the function set and argument set are clearly defined. The S-expression consists of one result-producing branch and $N_b$ function-producing branches. The result-producing branch is rooted at the output function $y$. The function-defining branches are rooted at the node functions $\{b_0, b_1, \cdots, b_{N_b-1}\}$, where $N_b$ is called a branch size. The branch size limits the number of nodes having one or more output branches in the filter structure synthesized. The
bodies of the function-defining branches and the result-
producing branch are constructed from three primitive
functions representing the multipliers, adders, and unit
delays, whose names are \( m \), \( a \), and \( d \), respectively. The
multiplier and unit delay functions have only one ar-
gument; the adder function has two arguments. The
arguments to the S-expression are the filter input \( x \) or
the automatically defined functions \( \{b_0, b_1, \cdots, b_{N-1}\} \).

Consider a digital filter in Figure 3(a). The S-
expression of the digital filter is represented as
\[
\text{(progn (b0 (a (x))}
\text{ (m (b1)))))
\]
\[
\text{(b1 (d (b0))))}
\text{(y (a (m (b0)))}
\text{(m (b1)))))},
\]
which has two function-defined branches and is iden-
tified with the rooted-tree shown in Figure 3(b). The
function \( b_1 \), for example, represents the sub-circuit be-
tween node \( b_0 \) and node \( b_1 \), and the value of the func-
tion \( b_1 \) is the same as the output signal of the node
\( b_1 \). The function \( b_1 \) can be interpreted as follows:
The argument to the delay function \( d \) is the value of
the function-defining branch \( b_0 \), and the value of the
function-defining branch \( b_1 \) is the value of the delay
function \( d \).

3.2. Genetic Operations

Genetic operations can alter the architecture of the
S-expressions, and consequently change the topology
of filter structures. This paper uses three genetic oper-
ations: crossover, mutation, and inversion. They are
applied to each individual with probabilities,
\( P_{\text{cross}} \) (probability of crossover), \( P_{\text{mut}} \) (probability of muta-
tion), and \( P_{\text{inv}} \) (probability of inversion), respectively.

**Crossover:** Crossover creates new child S-expressions
by exchanging sub S-expressions between two par-
ent S-expressions. The crossover points are se-
lected randomly. The two parents are shown at
the top of Figure 4, and the child S-expressions
after crossover are shown at the bottom of Fig-
ure 4.

**Mutation:** Mutation creates a new S-expression by
replacing an existing argument symbol with the
other possible symbol. The argument symbol re-
placed is selected randomly.

**Inversion:** Inversion creates a new S-expression by ex-
changing two sub S-expressions on only one S-
expression. The sub S-expressions exchanged are
selected randomly.

3.3. Fitness Measure

The fitness measure for the synthesis of filter struc-
tures is concerned with the characteristics of the de-
sired filter structure. In this paper, the fitness measure
is defined as the sum of two evaluation values as follows:
\[
f = f_{\text{sens}} + f_{\text{pre}},
\]
where \( f_{\text{sens}} \) is the evaluation value given by the coef-
ficient sensitivity of the filter structures. On the other
hand, the evaluation value \( f_{\text{pre}} \) is given by the realiz-
ability of the desired transfer function.

The value \( f_{\text{sens}} \) is defined as
\[
f_{\text{sens}} = S_{\text{max}} - S,
\]
where \( S \) is the magnitude sensitivity [3] of the filter
structure and the value \( S_{\text{max}} \) is the maximum of the
Table 3: Control parameters for GP.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>$N_{size}$</td>
</tr>
<tr>
<td>Generations</td>
<td>$N_{gen}$</td>
</tr>
<tr>
<td>Probability of crossover</td>
<td>$P_{cross}$</td>
</tr>
<tr>
<td>Probability of mutation</td>
<td>$P_{mut}$</td>
</tr>
<tr>
<td>Probability of inversion</td>
<td>$P_{inv}$</td>
</tr>
</tbody>
</table>

The coefficient sensitivity values of all the filter structures in the population.

The value $f_{pre}$ is defined as the sum of three evaluation values:

$$f_{pre} = f_{size} + f_{phys} + f_{tf}.$$  \hspace{1cm} (3)

The value $f_{size}$, concerned with the number of functions in the S-expressions, is defined as

$$f_{size} = \begin{cases} 10, & \text{if } N_m \leq 2n + 1, \ N_a \leq 4n, \\ 0, & \text{otherwise}, \end{cases}$$  \hspace{1cm} (4)

where $N_m$, $N_a$, and $N_d$ are the numbers of multipliers, adders, and unit delays in the S-expression, respectively, and $n$ is the order of the transfer function to be synthesized.

The value $f_{phys}$ is concerned with the physical realizability of the filter structures. Since the physically realizable filter structures contain no delay-free loops, we define the value $f_{phys}$ as

$$f_{phys} = \begin{cases} 10, & \text{if the filter structure is physically realizable}, \\ 0, & \text{otherwise}. \end{cases}$$  \hspace{1cm} (5)

The value $f_{tf}$, concerned with the realizability of a given transfer function, is defined as

$$f_{tf} = \begin{cases} 10, & \text{if the S-expression realizes the transfer function } H(z), \\ 0, & \text{otherwise}. \end{cases}$$  \hspace{1cm} (6)

Note that all the multiplier coefficients in the S-expressions are regarded as unknown numbers. Thus, the coefficients of the multipliers should be calculated by the coefficient comparison between the transfer functions $H(z)$ and $H'(z)$, where $H'(z)$ is the transfer function obtained from the S-expression [3].

4. DESIGN EXAMPLES

4.1. Parameter Settings

To demonstrate the performance of the proposed synthesis method, we consider the synthesis of low sensitivity filter structures realizing second-order and fourth-order low-pass filters. The GP parameters used in the experiment are shown in Table 3. Note that these parameters are determined through several preliminary examination to obtain the desired performance of results. In the selection, the elitist and tournament selections [5] are used. Three best-of-generation individuals are selected by the elitist selection and the rest is selected by the tournament selection, where the tournament size is 10. The GP-based synthesis software has been developed in Common Lisp and run on a 400-MHz Intel Pentium II processor.

4.2. Example 1: Second-order Low-pass filter

First, we consider a second-order low-pass filter with

Figure 7: Low-sensitive filter structure of the fourth-order low-pass filter (8) synthesized by GP-ADF (Form B).

The following transfer function:

\[ H(z) = \frac{0.098244 - 0.195065z^{-1} + 0.098244z^{-2}}{1 - 1.957184z^{-1} + 0.9586936z^{-2}}. \]  

(7)

This transfer function has poles close to the unit circle, and such a transfer function is known to be highly sensitive to coefficient quantization.

Figure 5 (Form A) shows the filter structure synthesized by GP-ADF. The run for synthesis has taken about 2.05 hours and the branch size \( N_b \) is 12. The magnitude responses of Form A, the balanced state-space realization [8], and the direct form II obtained with 10-bit coefficient accuracy are shown in Figure 6 along with the ideal response. The balanced realization has the minimum coefficient sensitivity in some sensitivity measure [8]. This figure shows that Form A and the balanced realization are clearly much more tolerant to coefficient quantization than the direct form II.

Table 4 compares the coefficient sensitivity of Form A with three other forms: the balanced realization, the low-sensitive structure proposed by Diniz and Antoniou [9], and the direct form II. Table 4 also compares the computational complexity with respect to the number of multipliers, two-input adders, and branches of these filters. The computational complexity roughly provides an indication of its cost of implementation [1]. The number of multipliers includes only multipliers with non-integer coefficients, and the number of filter branches are counted on the assumption that all the adders have two inputs. From Table 4, it is observed that Form A has much lower sensitivity \( S \) than that of the other filters. Although the coefficient sensitivity of the balanced realization is a little higher than that of Form A, it requires four more multipliers.

4.3. Example 2: Fourth-order low-pass filter

The next example is the synthesis of the low-sensitive filter structures realizing a fourth-order low-pass filter with the following transfer function:

\[ H(z) = \frac{\sum_{k=0}^{4} b_k z^{-k}}{1 + \sum_{k=1}^{4} a_k z^{-k}}, \]  

(8)

where, \( b_0 = 0.094347, b_1 = -0.362946, b_2 = 0.537579, b_3 = -0.362946, b_4 = 0.094347, a_1 = -3.792229, a_2 = 5.425692, a_3 = -3.469858, a_4 = 0.836799. \)

Figure 7 (Form B) shows the low-sensitive filter structure synthesized by GP-ADF. The run for synthesis has taken about 9.63 hours and the branch size \( N_b \) is 12. The magnitude responses of Form B, the balanced state-space realization, and the parallel form composed of second-order direct-form II subsystems (parallel form II) obtained with 10-bit coefficient accuracy are shown in Figure 8 along with the ideal response.

Table 5 compares the coefficient sensitivity of Form B with three other forms: the balanced realization, parallel form II, and direct form II. The comparison of
Table 4: Comparison of the sensitivity and computational complexity of various realizations of the transfer function (7).

<table>
<thead>
<tr>
<th></th>
<th>$S$ multipliers</th>
<th>adders</th>
<th>branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form A (Figure 5)</td>
<td>28.3150</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Balanced realization [8]</td>
<td>75.8174</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Diniz &amp; Antoniou [9]</td>
<td>1317.3267</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Direct form II</td>
<td>3313.8269</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5: Comparison of the sensitivity and computational complexity of various realizations of the transfer function (8).

<table>
<thead>
<tr>
<th></th>
<th>$S$ multipliers</th>
<th>adders</th>
<th>branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form B (Figure 7)</td>
<td>206.7186</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>Parallel form II</td>
<td>584.4802</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Direct form II</td>
<td>93854.9569</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

the coefficient sensitivity and the computational complexity of these forms is shown in Table 5. From Table 5, Form B and the balanced realization have much lower coefficient sensitivity compared with the traditional parallel form II and direct form II. Moreover, the advantage of Form B compared with the balanced realization is that Form B requires only 9 multipliers although the balanced realization requires 25 multipliers.

5. CONCLUSIONS

We have proposed the synthesis method of the low coefficient sensitivity digital filter structures using Genetic Programming with Automatically Defined Functions (GP-ADF). In this method, digital filter structures are represented as the S-expressions with subroutines. Since this representation is suitable for GP, we can easily apply GP to the synthesis of digital filter structures. We have also defined the fitness measure including the value concerned with the coefficient sensitivity of the filter structures implemented. Two numerical examples have shown that GP-ADF can synthesize the very low magnitude sensitivity filter structures efficiently.

REFERENCES