SYNTHESIS OF LOW-SENSITIVITY SECOND-ORDER 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
uesaka@mk.ecei.tohoku.ac.jp, kawamata@ecei.tohoku.ac.jp

ABSTRACT
This paper proposes a synthesis method for low coefficient sensitivity second-order 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. It is easy to generate syntactically valid S-expressions and perform the genetic operations because the representation is suitable for GP. In a numerical example, 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 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, 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]. The GP-based synthesis can search low coefficient sensitivity digital filter structures by representing filter structures as S-expressions. The disadvantage of the GP-based synthesis in [2] is that this method 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.

We restrict ourselves to the synthesis of second-order IIR structures with the transfer function:

\[ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \]  

which can be used in cascade or in parallel for the realization of high-order transfer functions of order more than two.

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 [3, 4]. 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 [5]. 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 to the population (Figure 1: line 9). The genetic operations consist of crossover, mutation, and inversion. These operations are ap-
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.

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 [6]. 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 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, ..., b_7 |
| Argument set for the function-defining branches | x, b_0, b_1, ..., b_7 |
| 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, ..., b_7 |
| 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 [2] because the function set and argument set are clearly defined. The S-expression consists of one result-producing branch and eight 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, ..., b_7\}. The number of the function-defining branches 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 argument; the adder function has two arguments.
The arguments to the S-expression are the filter input \( x \) or the automatically defined functions \( \{b_0, b_1, \cdots, b_7\} \).

Consider a digital filter in Figure 3(a). The S-expression of the digital filter is represented as

\[
(progn \ (b_0 \ (a \ (x)) \ ((m \ (b_1)))) \ (b_1 \ (d \ (b_0))) \ (y \ (a \ (m \ (b_0)) \ ((m \ (b_1))))),
\]

which has two function-defined branches and is identified with the rooted-tree shown in Figure 3(b). The function \( b_1 \) represents the sub-circuit between node \( b_0 \) and node \( b_1 \), and the value of the function \( b_1 \) 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

This paper uses three genetic operations: crossover, inversion, and mutation. They are applied to each individual with probabilities, \( P_{cross} \) (probability of crossover), \( P_{mut} \) (probability of mutation), and \( P_{inv} \) (probability of inversion), respectively.

**Crossover:** Crossover creates new S-expressions by exchanging sub-S-expressions between two S-expressions. The sub-S-expressions exchanged are selected randomly.

**Inversion:** Inversion creates a new S-expression by exchanging two sub-S-expressions in one S-expression. The sub-S-expressions exchanged are selected randomly.

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

### 3.3. Fitness Measure

The fitness measure for the synthesis of filter structures is concerned with the characteristics of the desired filter structure. In this paper, the fitness measure is defined as the sum of two evaluation values as follows:

\[
f = f_{sens} + f_{pre},
\]

where, \( f_{sens} \) is the evaluation value given by the coefficient sensitivity of the filter structures. On the other hand, the evaluation value \( f_{pre} \) is given by the realizability of the desired transfer function.

The value \( f_{sens} \) is defined as

\[
f_{sens} = S_{max} - S,
\]

where \( S \) is the magnitude sensitivity [2] of the filter structure and the value \( S_{max} \) is the maximum of 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:

\[
f_{pre} = f_{size} + f_{phys} + f_{tf}.
\]

The \( f_{size} \), concerned with the number of functions in the S-expressions, is defined as

\[
f_{size} = \begin{cases} 
10, & \text{if } N_m \leq 5, N_a \leq 8, \text{ and } N_d \leq 4, \\
0, & \text{otherwise} 
\end{cases}
\]

where \( N_m, N_a, \) and \( N_d \) are the number of \( m, a, \) and \( d \) in the S-expression, respectively.

The value \( f_{phys} \) is concerned with the physical realizability of the filter structures. In this paper, we regard the filter structures without delay-free loops as the physically realizable filter structures. The value \( f_{phys} \) is defined as

\[
f_{phys} = \begin{cases} 
10, & \text{if the filter structure is physically realizable}, \\
0, & \text{otherwise} 
\end{cases}
\]

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

\[
f_{tf} = \begin{cases} 
10, & \text{if the S-expression realizes the transfer function } H(z), \\
0, & \text{otherwise} 
\end{cases}
\]

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 function \( H(z) \) and \( H'(z) \) of the S-expression [2], where \( H'(z) \) is the transfer function given by the S-expression.
4. DESIGN EXAMPLE

To demonstrate the performance of the proposed synthesis method, we consider the synthesis of low coefficient sensitivity digital filter structures realizing the following transfer function:

\[ H(z) = \frac{0.098244 - 0.195065z^{-1} + 0.098244z^{-2}}{1 - 1.957184z^{-1} + 0.958936z^{-2}}. \]  (8)

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

Figure 4 (Form A) shows the filter structure synthesized by GP-ADF. The magnitude sensitivity \( S \) of Form A is 15.370, while the coefficient sensitivity of the direct-form II realizing (8) is 2393.883, and that of the filter structure shown in [2] is 22.451. The magnitude responses of Form A and the direct form II obtained with 8-bit coefficient accuracies are shown in Figure 5 along with the ideal response. This figure shows that Form A is clearly much more tolerant to coefficient quantization than the direct form II.

The GP parameters used in the experiment are shown in Table 3. Note that these parameters are selected by experiments. The run for the synthesis has taken 22.1 minutes on a 400-MHz Intel Pentium II Processor.

5. CONCLUSIONS

We have proposed the synthesis method of the second-order low coefficient sensitivity digital filter structures using Genetic Programming with Automatically Defined Functions. 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. The numerical example has shown that GP-ADF can synthesize the very low magnitude sensitivity filter structure efficiently.

6. REFERENCES


