Calculation of Selection Probabilities of Stack Filters through BDD

Milena Stanković, Suzana Stojković, Jaakko Astola,
and Karen Egiazarian

Abstract: This paper presents a procedure for calculation of selection probabilities of stack filters using Binary decision diagrams (BDDs) to represent the positive Boolean function that defines the stack filter. The procedure is derived by a modification of the spectral method for calculation of selection probabilities of stack filters. The usage of BDDs, instead of vectors overcomes the exponential complexity of the corresponding spectral method and extends application of the spectral method to stack filters with large windows widths.

Keywords: Stack filter, ordered binary decision diagram, rank selection probabilities, sample selection probabilities, joint selection probabilities, partial Boolean differences, symmetric transform.

1 Introduction

Stack filters constitute an important class of nonlinear filters based on positive Boolean functions and have been successfully used in signal and image processing [1, 2]. Statistical properties of stack filters have been studied in terms of output distribution and rank selection probabilities (RSP), sample selection probabilities (SSP), and joint selection probabilities [3, 4, 5, 6, 7]. RSPs and SSP are probabilities that the output equals a sample with the certain rank and certain time index in the filter window, respectively. The output distribution of a stack filter can be expressed in terms of its RSPs. On the other hand, SSPs are important for examining...
the detail preservation properties of the given stack filter. Joint selection probabilities are combination of both rank and sample selection probabilities. Efficient spectral algorithms for the computation of the selection probabilities of stack filters are discussed in [3]. Their implementation is based on FFT-like fast algorithms, what is not convenient in case when large window, corresponding to positive Boolean function with large number of variables, is used.

Binary decision diagrams (BDDs) are data structures for representation and manipulation of Boolean functions especially convenient in the case of large functions [8, 9]. In this paper, we propose a modification of the spectral algorithm for calculation of joint selection probabilities in [3] and its implementation through BDD representing the positive Boolean function used as the base for stack filtering. This allows the computation of selection probabilities for large window widths, where the application of the existing methods is restricted for their exponential complexity.

2 Stack Filters

A window-width \( n \) stack filter is based on an \( n \)-variable monotone (positive) Boolean function \( f(x_1, x_2, \ldots, x_n) \) Since positive Boolean functions have no negated literals in their minimal disjunctive normal forms, the operations of disjunction and conjunctions (AND and OR) can be replaced by the MAX and MIN operations in the multi-level domain.

Let \( X(i), i = 1, 2, \ldots, L \), be a real valued discrete time signal. For \( k = 1, 2, \ldots, L - n \), denote

\[
X(k) = [X(k + 1), X(k + 2), \ldots, X(k + n)],
\]

that corresponds to a window of the length \( n \) starting at the index \( k + 1 \). Let \( f(x_1, x_2, \ldots, x_n) \) be a positive Boolean function of \( n \) variables, whose minimum sum of products form (which is unique for positive Boolean functions) is:

\[
f(x_1, x_2, \ldots, x_n) = \bigvee_{m=1}^{r} \left[ x_{j(m,1)} \wedge x_{j(m,2)} \wedge \cdots \wedge x_{j(m,p_m)} \right], \quad (1)
\]

where \( j(m, 1) < j(m, 2) < \ldots < j(m, p_m) \) for \( m = 1, 2, \ldots, r \). From (1), we determine the corresponding maximum function or, equivalently, a continuous logic function

\[
\sum_{f}(X) = \sum_{f}(X_1, X_2, \ldots, X_n) = \max_{m=1}^{r} \left\{ \min_{l=1}^{p_m} \{ X_{j(m,l)} \} \right\}
\]

2 Stack Filters

A window-width \( n \) stack filter is based on an \( n \)-variable monotone (positive) Boolean function \( f(x_1, x_2, \ldots, x_n) \) Since positive Boolean functions have no negated literals in their minimal disjunctive normal forms, the operations of disjunction and conjunctions (AND and OR) can be replaced by the MAX and MIN operations in the multi-level domain.

Let \( X(i), i = 1, 2, \ldots, L \), be a real valued discrete time signal. For \( k = 1, 2, \ldots, L - n \), denote

\[
X(k) = [X(k + 1), X(k + 2), \ldots, X(k + n)],
\]

that corresponds to a window of the length \( n \) starting at the index \( k + 1 \). Let \( f(x_1, x_2, \ldots, x_n) \) be a positive Boolean function of \( n \) variables, whose minimum sum of products form (which is unique for positive Boolean functions) is:

\[
f(x_1, x_2, \ldots, x_n) = \bigvee_{m=1}^{r} \left[ x_{j(m,1)} \wedge x_{j(m,2)} \wedge \cdots \wedge x_{j(m,p_m)} \right], \quad (1)
\]

where \( j(m, 1) < j(m, 2) < \ldots < j(m, p_m) \) for \( m = 1, 2, \ldots, r \). From (1), we determine the corresponding maximum function or, equivalently, a continuous logic function

\[
\sum_{f}(X) = \sum_{f}(X_1, X_2, \ldots, X_n) = \max_{m=1}^{r} \left\{ \min_{l=1}^{p_m} \{ X_{j(m,l)} \} \right\}
\]
The continuous stack filter, [11] corresponding to the positive Boolean function $f$ given by (1) is defined by specifying its output $Y(k)$ for the input signal $X(k)$:

$$Y(k) = \sum_f [X(k)], \text{ for } k = 1, 2, \ldots, L - n + 1.$$ 

**Example 1** The Boolean function $f(x_1, x_2, x_3, x_4, x_5) = x_1 x_2 x_3 \lor x_2 x_3 x_4 \lor x_3 x_4 x_5)$ defines the stack filter:

$$\sum_f (X_1, X_2, X_3, X_4, X_5) = \max\{\min\{X_1, X_2, X_3\}, \min\{X_2, X_3, X_4\}, \min\{X_3, X_4, X_5\}\}.$$ 

The output of a stack filter is always one of the input samples. It is possible to determine the probability that the $i$-th smallest sample $X_{i(j)}$ or the $j$-th sample $X_j$ is the output of the filter. These probabilities are called rank and sample selection probabilities, respectively [10]. Selection probabilities give a very intuitive way of understanding the role of different samples in the filter window. They provide a general view of how important different samples are with respect to each other for a given filter.

**Definition 1** Rank selection probability of the stack filter $\sum_f (X_1, X_2, \ldots, X_n)$ is a vector $r = [r_1, r_2, \ldots, r_n]$, where $r_i$ is the selection probability of the smallest $i$-th (i-th rank) input $X_{i(i)}$:

$$r_i = P(Y = X(i)), \text{ for } i = 1, 2, \ldots, n.$$ 

**Definition 2** Sample selection probability of the stack filter $\sum_f (X_1, X_2, \ldots, X_n)$ is a vector $s = [s_1, s_2, \ldots, s_n]$, where $s_i$ is the selection probability of the $i$-th input $X_i$:

$$s_i = P(Y = X_i), \text{ for } i = 1, 2, \ldots, n.$$ 

**Definition 3** Joint selection probability of the stack filter $\sum_f (X_1, X_2, \ldots, X_n)$ is the matrix $p = [p_{ij}]$ whose element $p_{ij}$ is defined as:

$$p_{ij} = P(Y = X_{i(j)} = X_j); \text{ for } i, j = 1, 2, \ldots, n.$$ 

Sample selection probability and rank selection probability vectors can be computed by summing the values in columns and rows in the joint selection probability matrix:

$$r_j = \sum_{i=1}^n p_{ij} \text{ and } s_i = \sum_{j=1}^n p_{ij}.$$
3 Spectral Method for Calculation of Selection Probability Vectors of Stack Filter

The joint selection probability matrix of the stack filters, based on an $n$-variable positive Boolean function $f$, can be computed by using the following relation [3]:

$$ P = \Omega_n S_n F' $$

(2)

where: $\Omega_n$ is a diagonal matrix with diagonal entries $\omega_i = \left(n \binom{n-1}{i-1}\right)^{-1}$. $S_n$ is the Symmetric transform matrix defined by the following recursion:

$$ S_n = \begin{bmatrix} 0_{n-1} & S_{n-1} \\ S_{n-1} & 0_{n-1} \end{bmatrix}, \quad S_0 = [1], $$

(3)

where: $0_n$ is the zero vector of length $2^n$, and $F'$ is the matrix of the partial Boolean differences with respect to all variables in $f$. Thus, columns of $F'$ are partial Boolean differences $f'_j$, defined as:

$$ f'_j = \frac{\partial f}{\partial x_j} = f(x_j = 0) \oplus f(x_j = 1), \text{ for } j \in [1,n]. $$

(4)

By using (2), the computation of the joint selection probability matrix can be performed in $n(3 \cdot 2^{n-1} - n)$ additions, $2^n$ multiplications, and $2^n$ comparisons, [3].

**Example 2** Consider the stack filter from Example 1. Since, the truth-vector of $f$ is $F = [00000001000000110000000100001111]^T$, the truth-vectors of partial Boolean differences of $f$ with respect to variables $x_1, x_2, \ldots, x_3$ are:

$$ F' = \begin{bmatrix} F'_1 \\ F'_2 \\ F'_3 \\ F'_4 \\ F'_5 \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T $$

(5)

From (2), it follows:
\[ P = Q S F^T \]

\[
\begin{bmatrix}
\frac{1}{3} & 0 & 0 & 0 & 0 \\
0 & \frac{1}{20} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{30} & 0 & 0 \\
0 & 0 & 0 & \frac{1}{20} & 0 \\
0 & 0 & 0 & 0 & \frac{1}{3} \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

and

\[ s = \begin{bmatrix}
\frac{1}{12} & \frac{1}{6} & \frac{1}{2} & \frac{1}{6} & \frac{1}{12} \\
\end{bmatrix}^T, \]
\[ r = \begin{bmatrix}
\frac{1}{3} & \frac{1}{2} & \frac{3}{10} & 0 & 0 \\
\end{bmatrix}^T. \]

\section{BDDs}

For representation of Boolean function a Binary decision tree (BDT), as graphical representation of its truth table, can be used [11].

\textbf{Example 3} Fig. 1 shows BDT for the positive Boolean function \( f(x_1, x_2, x_3) = x_1 x_2 \lor x_1 x_3 \).

In the BDT for a given \( f \), each non-terminal node is labeled by a variable \( x_i \) in \( f \) and has two edges directed to two children nodes. The 0-edge and the 1-edge represent the cases when the values 0 and 1 are assigned to the variable \( x_i \).
respectively. Terminal nodes show values of the function $f$ at the points defined by $n$-tuples determined as products of labels at the edges.

**Definition 4** The path $p_{ij}$, connecting node $x_i$ (at the level $i$) and node $x_j$ (at the level $j$), for $j > i$, is defined as the product of labels at the edges, i.e., $p_{ij} = p_i p_{i+1} \cdots p_{j-1}$, where $p_k, k \in [0, 1]$ are the labels at the edges from the node $x_i$ to the node $x_j$.

A BDT consists of sub-trees, where each sub-tree is rooted at a node at lower levels. Each sub-tree is defined by the path from the root node of the BDT to the their root nodes.

**Definition 5** $p_{ij}$-subtree in the BDT is the tree with the root in the node $x_j$ and connected with the node $x_i$ with the path $p_{ij}$, $(j > i)$.

**Example 4** In Fig. 2, the 00-subtree and 1-subtree in the BDT from Example 3 are shown.

As it can be seen from Fig. 1, in a BDT, there may be some redundant nodes and isomorphic sub-trees. By elimination of the redundant nodes and isomorphic sub-trees in the BDT we get a directed graph (DAG) or Binary Decision Diagram (BDD). A BDD contains smaller number of nodes than the corresponding BDT. If we put the restriction that variables occur in the same order on all paths from the root node to the terminal nodes, and that each variable occurs at most once on each path, then the BDD is called the Reduced Ordered Binary Decision Diagram (ROBDD) [11].
Example 5 BDD in Fig. 3 is the ROBDD for the variable ordering \(x_1, x_2, x_3\), derived from the BDT in Fig.1.

In [11], it is shown that the ROBDD is a canonical form for a Boolean function. That is, for any Boolean function, there is exactly one ROBDD representation. In this paper, only ROBDDs are considered and for briefness we will refer to these graphs as BDDs.

5 Calculation of the Selection Probabilities of Stack Filter through BDD

From (2) it follows that for implementation of the spectral method for calculation of selection probabilities through BDD it is necessary to provide procedures for calculation of:

1. Partial Boolean differences with respect to all variables,

2. Symmetric transform (multiplication with the matrix \(S_n\)), and

3. Multiplication with the matrix \(\Omega_n\).
5.1 Procedure for calculation of Boolean differences through BDD

The partial Boolean difference with respect to variable \( x_j \), i.e. \( j \)-th column vector \( F'_j \) of the matrix \( F' \), can be computed as

\[
F'_j = \left( E_n^{(j)} F \right)_{\text{mod2}}
\]

where

\[
E_n^{(j)} = I_{j-1} \otimes \begin{bmatrix} 1 & 1 \end{bmatrix} \otimes I_{n-j}
\]

(5)

\( I_k \) is \( (2^k \times 2^k) \) identity matrix, and \( \otimes \) denotes the Kronecker product of matrices.

From (5), we deduce the following procedure for calculation of Boolean difference with respect to the input variable \( X_k \).

Procedure for calculation of partial Boolean difference with respect to the \( k \)-th variable

Step 1 Given BDD for \( f \).

Step 2 At each node at the level \( k \) perform the \( \text{mod2} \) additions of the 0-subtree and 1-subtree that generates a new sub-tree connected to the node by the edge labeled with \( f'_k \). This transformation is expressed as:

\[
p f'_k \text{-subtree} = p0 - \text{subtree} \oplus p1 - \text{subtree},
\]

for all paths \( p = p_1p_2\cdots p_{k-1} \), for \( p_i \in \{0, 1\} \) and \( i = 1, 2, \cdots, n \) and \( k = 2, 3, \cdots, n \) and \( ft^{(k)} = 0 - \text{subtree} \oplus 1 - \text{subtree} \), for \( k = 1 \).

Fig. 4 illustrates the calculations performed at the nodes to determine the corresponding partial Boolean differences.

Procedure for calculation of the partial Boolean differences with respect to all variables

Step 1 Given BDD for \( f \).

Step 2 Traverse BDD for \( f \) top-down by starting from the root node, and at each level perform the procedure for calculation of the corresponding partial difference.
Fig. 4. Mod2 addition of the sub-trees at the level $k$.

This procedure transforms the BDD for $f$ into a BDD representing at the same time $f$ and its partial differences with respect to all input variables. We denote this BDD by BDDdF (BDD of Boolean differences).

**Example 6** Fig. 5 shows the BDDdF for switching functions for $n = 3$.

![Diagram of BDDdF for n = 3](image)

Note that in the BDDdF, there are two types of nodes, ones with two edges and others with three edges. In the BDDdF, the $p_1p_2\cdots p_n\text{ - subtree}$, where $p_i \in \{0, 1\}$, for $i = 1, 2, \cdots, n$, is the starting BDD for the function $f$. The $0\text{ - subtree}$ and $1\text{ - subtree}$ of the root node including only the $f'_k\text{ - subtrees}$ at the level $k$, e.g., $p_1p_2\cdots p_{k-1}f'_k\text{ - subtree}$, where $p_i \in \{0, 1\}$, for $i = 1, 2, \cdots, k-1$, are the BDDs of the corresponding partial differences. The $f'_1\text{ - subtree}$ of the root node is the BDD of the $f'_{k-1}$ difference.

**Example 7** Fig. 6 shows the BDDdF for the function $f(x_1, x_2, x_3)$, in Example 3,
thus, this figure shows BDD for $f$ and BDDs for the partial Boolean differences with respect to all variables.

![Diagram](image)

Fig. 6. BDDdF for function $f(x_1, x_2, x_3) = x_1x_2 \lor x_1x_3$.

The $k$-th Boolean difference can be determined by traversing the corresponding sub-trees in the BDDdF. That can be performed by the function `ReadDifference` given in the pseudo code in the Appendix.

5.2 Procedure for calculation of the symmetric transform through BDD

The recursive structure of elementary symmetric $S_n$ Boolean matrix permits definition of a FFT-like algorithm for calculation the symmetric spectrum [9].

The elementary symmetric Boolean matrix $S_n$ defined in (2) can be factorized as:

$$S_n = \prod_{i=1}^{n-1} C_i,$$

where

$$C_i = \begin{cases} I_2 & \text{for } j \neq n-i \\ \begin{bmatrix} I_{i+1} & 0_{i+1} \\ 0_{i+1} & I_{i+1} \end{bmatrix} & \text{for } j = n-i \end{cases}$$

In this relation $I_k$ is the $(k \times k)$ identity matrix.

**Example 8** For $n = 3$, the elementary symmetric transform matrix may be factorized as:

$$S_3 = C_1 \cdot C_2$$
where

\[ C_1 = I_2 \otimes \begin{bmatrix} I_2 & 0_2 \\ 0_2 & I_2 \end{bmatrix} \quad \text{and} \quad C_2 = \begin{bmatrix} I_3 & 0_3 \\ 0_3 & I_3 \end{bmatrix}. \]

Fig. 7 shows the FFT-like algorithm for symmetric transform for \( n = 3 \), derived from this factorization of \( S_3 \).

If a function \( f \) is given by a BDD, the BDD for the symmetric spectrum for \( f \) is determined by the following procedure [6]:

Step 1 At each node at the level \( (n-1) \) perform calculation shown in Fig. 8.

Step 2 At each node at the level \( k \), \( (\text{for} \ k = n-2 \ \text{to} \ \text{bystep} \ -1) \) perform calculation shown in Fig. 9.

This procedure transforms the BDD for \( f \) into the BDD for the symmetric spectrum for \( f \). It should be noted that BDDs for the symmetric spectra have a characteristic structure (shape) as shown in Fig. 10.
Example 9 Consider the function $f(x_1, x_2, x_3)$ in Example 3. Fig. 11 shows the BDD for this function, with non-terminal nodes denoted by $Q_i, i = 1, \ldots, q_i$, where $q_i$ is the number of nodes in the BDD.

We calculate the symmetric spectrum for $f$ through the BDD in Fig 11, by applying the operations from Fig. 8 and Fig. 9.

Step 1 First, we perform the calculation in the node $Q_2$ at the level 2. The resulting BDD is shown in Fig. 12.

Step 2 Next, we perform the calculation from Fig. 9 in the node $Q_1$ at the level 1.
Calculation of Selection Probabilities of Stack Filters through BDD

5.3 Procedure for multiplication with $\Omega$

Since $\Omega_n$ is a diagonal matrix, the multiplication of $S F'$ by $\Omega_n$ reduces to the multiplication of the constant nodes in the BDD for the symmetric spectrum with the corresponding non-zero values in $\Omega_n$.

Therefore, the procedure of multiplication with $\Omega_n$ consists of a single step.

Step 1 Multiply the value in the terminal node connected with the node at the level

The resulting BDD is shown in Fig. 13.
434  

M. Stanković, S. Stojković, J. Astola, and K. Egiazarin:

$k$ by the value $ω_k$ where:

$$ω_{k+1} = ω_k \cdot \frac{i}{n-i}, \text{ for } i = 1, 2, \ldots, n-1, \text{ and } ω_1 = \frac{1}{n}.$$ 

5.4 Procedure for building the BDD for selection probabilities

Finally we can define the whole procedure for calculating the selection probabilities of a Stack filter defined by a positive Boolean function $f(x_1, x_2, \ldots, x_n)$.

![BDD for the function $f$ from Example 8.](image)

**Fig. 14.** BDD for the function $f$ from Example 8.

Step 1 Generate BDD for $f(x_1, x_2, \ldots, x_n)$.

Step 2 Transform the BDD from Step 1. into BDDdF by the procedure for calculating the partial Boolean differences with respect to all input variables.

Step 3 Perform the procedure for the symmetric transform through the BDDs of the Boolean differences in the BDDdF generated in Step 2.

Step 4 Perform the procedure for multiplication with $Q_n$ through the BDDs of the Boolean differences in the BDDdF generated in Step 3.
Calculation of Selection Probabilities of Stack Filters through BDD

(Fig. 15. BDDdF for function from Example 10.

(Note: After this step the values of joint probabilities will be values of the terminal nodes in the BDDs of the Boolean differences in the resulting BDDdF.)

Step 5 Calculate the \( r_k \) \((k = 1, 2, \ldots, n)\) values of rank probability vector by summing the values in all terminal nodes in the \( pf_k \)-subtree, where \( p = p_1 p_2 \ldots p_k \), and \( p_i \in \{0, 1\} \).

Step 6 Calculate the BDD of simple selection probability vector by summing all the sub-trees \( pf_k \)-subtree, where \( p = p_1 p_2 \ldots p_k \), and \( p_i \in \{0, 1\} \).

Example 10 Consider selection probabilities of the stack filter defined by the function \( f(x_1, x_2, x_3, x_4, x_5) = x_1 x_2 x_3 \lor x_2 x_3 x_4 \lor x_3 x_4 x_5 \) in Example 1.

Step 1 BDD for this function is shown in Fig 14.

Step 2 By the procedure for calculating partial Boolean differences with respect to all variables, this BDD is transformed into BDDdF shown in Fig. 15.

Step 3 By the procedure for calculating the symmetric spectrum through the BDDs of the Boolean differences in the BDDdF from Fig. 15 we get the BDDs
Fig. 16. BDDdF for the function $f$ from Example 10 with BDDs for the symmetric spectra for all Boolean differences.

representing the symmetric spectra of the Boolean differences of $f$. In the Fig 16, these BDDs are shown separately, while in the Fig. 17, they are shown as sub-trees in the BDDdF.

**Step 4** Finally, after multiplication with the matrix $\mathbf{W}$, we get the BDDdF shown in Fig. 18.

For reading the values for the joint, rank, and a simple selection probability vectors we define the functions ReadJointSelectionProbabilities, ReadRankSelectionProbabilities and ReadSampleSelectionProbabilities, that are given in pseudo codes in the Appendix.

### 6 Experimental Results

For an experimental validation of our method five groups of randomly generated positive Boolean functions were used. In Table 1, the data for these groups of mcnc benchmark functions, [12] are presented:
Calculation of Selection Probabilities of Stack Filters through BDD

Fig. 17. BDD for the symmetric spectra for partial Boolean differences with respect to all variables.

Fig. 18. BDDdF representing selection probabilities for the function from Example 10.

\( n \) - Number of input variables,
\( N \) - Number of tested functions in the group,
\( N_{\text{BDD}} \) - Average number of non-terminal nodes in the starting BDD,
\( N_{\text{BDDdF}} \) - Average number of non-terminal nodes in the BDDdF,
\( N_{\text{final}} \) - Average number of non-terminal nodes in the resulting BDDdF after all transformations,
\( N_{\text{opp}} \) - Average number of operations in our method,
\( N_{\text{oppS}} \) - Number of operation in the spectral method.
Table 1. Complexity of calculation of selection probabilities through BDD for some groups of random generated functions

<table>
<thead>
<tr>
<th>n</th>
<th>N</th>
<th>N_BDD</th>
<th>N_BDDdF</th>
<th>N_final</th>
<th>Nopp</th>
<th>NoppS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3.6</td>
<td>5.1</td>
<td>10.6</td>
<td>14.7</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>11.5</td>
<td>17.7</td>
<td>32.7</td>
<td>92.1</td>
<td>279</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>26.7</td>
<td>42.7</td>
<td>76.4</td>
<td>212.3</td>
<td>668</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>44.6</td>
<td>78.3</td>
<td>122.3</td>
<td>511.4</td>
<td>1551</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>79.3</td>
<td>101.5</td>
<td>181.6</td>
<td>1358.2</td>
<td>7855</td>
</tr>
</tbody>
</table>

Calculation time in all cases was less then 0.1 milliseconds. Calculation is performed on a 133MHz Pentium PC with 32MB of RAM.

7 Conclusion

We propose a procedure for calculating selection probabilities of stack filters through ordered binary decision diagrams. This procedure is a modification of the spectral method in [3], and the implementation using Binary decision diagrams extends the application of the method to functions with large window widths.

References


Calculation of Selection Probabilities of Stack Filters through BDD


Appendix

This appendix contains functions derived to determine values of Boolean differences and the selection probabilities of stack filters through BDDs.

Data structure for DD node representing

```
struct TNode
{
    float value;
    char level;
    unsigned id;
    struct TNode *left, *right, *dif;
}
```

Function for reading Boolean difference through BDDDF

```
ReadDifference(TNode *root, int pret_level, int k)
{
    if (pret_level<k && root->level>k)
        for (i=0; i<2*root->level-pret_level-2; i++)
            printf("0, ");
    else
        for (i=0; i<2*root->level-pret_level-1; i++)
        {
            if (root->level==Nvar+1)
                printf("1, ");
            elseif (root->level==k)
                ReadDifference(root->level,root->dif,k);
```
else
{
    ReadDifference(root->level,root->left,k);
    ReadDifference(root->level,root->right,k);
}
return;
}

Function for reading joint selection probabilities

ReadJointSelectionProbabilities(TNode *root, float **J)
{
    for (i=0;i<Nvar;i++)
    {
        k=0;
        readJoint(root,LEFT,i,k,J);
    }
}
readJoint( TNode *root, short subtree, short row, short& k, float **J )
{
    if ( root->level == Nvar+1 )
    {
        if ( subtree == RIGHT )
            J[row][k++] = root->value;
        else
            for (i=k; i<Nvar; i++)
                j[row][i]=root->value;
    }
    else
    {
        if ( root->level == row )
            readJoint( root->dif,LEFT,row,k,J );
        else
        {
            readJoint( root->right,RIGHT,row,k,J );
            if ( subtree == LEFT )
                readJoint( root->left, LEFT, row, k, J );
        }
    }
}
Function for reading sample selection probabilities

ReadSampleSelectionProbabilities(TNode *root, float *S)
{
    for (i=0; i<Nvar; i++)
    {
        S[i]=0;
        k=0;
        readSample(root, LEFT, i, k, S[i]);
    }
}

readSample( TNode *root, short subtree, short i, short & k, float & Si )
{
    if ( root->level == Nvar+1 )
    {
        if ( subtree == RIGHT )
        {
            Si += root->value;
            k++;
        }
        else
        {
            for (j=k; j<Nvar; j++)
            {
                Si += root->value;
            }
        }
    }
    else
    {
        if ( root->level == i )
            readSample( root->dif, LEFT, i, k, Si );
        else
        {
            readSample( root->right, RIGHT, i, k, Si );
            if ( subtree == LEFT )
                readSample( root->left, LEFT, i, k, Si );
        }
    }
}
Function for reading of rank selection probabilities

ReadRankSelectionProbabilities(TNode *root, float *R)
{
    for (i=0;i<Nvar;i++) R[i]=0;
    for (i=0;i<Nvar;i++)
    {
        k=0;
        readRank( root, LEFT, i, k, R );
    }
}

readRank( TNode *root, short subtree, short i, short &k, float* R )
{
    if ( root->level == Nvar+1 )
    {
        if ( subtree == RIGHT )
            R[k++] += root->value;
        else
            for (j=k; j<Nvar; j++ )
                R[j] += root->value;
    }
    else
    {
        if ( root->level == i )
            readRank( root->dif, LEFT, i, k, R );
        else
        {
            readRank( root->right, RIGHT, i, k, R );
            if ( subtree == LEFT )
                readRank( root->left, LEFT, i, k, R );
        }
    }
}