# Orthogonal Block Building
# Using Ordered Lists of Ternary Vectors

Bernd Steinbach, Christina Dorotska

Freiberg University of Mining and Technology, Institute of Computer Science,
D-09596 Freiberg, Germany,
e-mail: {steinb, dorotsk}@informatik.tu-freiberg.de

**Abstract.**
In this paper we investigate the possibility of faster calculation of operations on Boolean functions. We use the representation of a function as an ordered list of ternary or Boolean vectors and propose a faster algorithm that is based on ordering of vectors. We sort the vectors in lists using the number of ones and strokes to create classes and subclasses. This model is used to speed up the minimization of orthogonal vector lists, an algorithm known as block building.
Our algorithm is compared with three other algorithms, and finally it is shown by means of experimental results that our algorithm with ordering of vectors require fewer comparisons and can find more pairs of vectors, which can build a block.

## 1. Introduction

Recently, two fundamental approaches are used for representation of Boolean functions. In contrast to decision diagrams (DD's) representation, in this paper we will focus on the data structures List of Boolean Vectors (BVL) and List of Ternary Vectors (TVL) [1-8].
A BVL is a very simple data structure to represent a Boolean function – we only include all these Boolean vectors in the list, for which the function has the value ONE. The maximal number of Boolean vectors in a BVL is bounded by $2^k$, where $k$ is the number of variables. In order to reduce this large number of vectors, TVL was proposed.
A Ternary Vector (TV) consists of elements ZERO, ONE and STROKE. A STROKE (that means a DASH) is allowed to replace a ZERO or a ONE. Vice versa, if two Boolean vectors are different only in one place, they can be represented as one ternary vector containing a STROKE in this place. If one ternary vector includes $s$ elements STROKE, this vector represents $2^s$ Boolean vectors. Therefore, the data structure TVL reduces the exponential expansion of a List of Binary Vectors by an exponential reduction of the number of ternary vectors. Minimization of the number of ternary vectors leads at the same time to the reduction in the required memory [4].
If a TVL contains two vectors, that are different only in one column with combination "0/1", they can be combined into one ternary vector that contains a STROKE in the corresponding column. Using this operation, called "orthogonal block building" [5], the need of the memory can be reduced further. Smaller TVL can be calculated in a shorter time, but not only the size of the TVL has influence to the speed of their calculation. Therefore we have developed an ordered model of classes and subclasses of ternary vectors without changing the total number of ternary vectors. This method reduces the number of comparisons, needed to perform operations on Boolean functions [8]. In this paper, we suggest an algorithm for block building, which uses this ordered model, and compare it with three other block building algorithms – two without vector ordering, and one with ordering according to the numbers of strokes.
For a better understanding, we review some basic knowledge about TVL's in section 2. In order to create an ordering model, we study some properties of the Boolean space in section 3. We generalize this model in section 4. In section 5, we compare the four orthogonal block building algorithms. Experimental results are discussed in section 6. Finally, we summarize our paper and give some ideas for future work in section 7.

## 2. Preliminaries

A TVL may represent all Boolean vectors of a BVL by a smaller number of ternary vectors. The set of Boolean vectors collected in one ternary vector can be reconstructed if the strokes in the ternary vector are substituted by all possible patterns of zero and one elements. One ternary vector, which contains only stroke elements, describes all $2^k$ Boolean vectors of the Boolean space.

Two different Boolean vectors of the same length are called orthogonal. Two ternary vectors have the property of orthogonality, if there exists no Boolean vector included in both ternary vectors. This property is satisfied, if these ternary vectors have a 0/1 combination in at least one column. The ternary vectors $t_1$ = "01-" and $t_2$ = "11-" for example, have a 0/1 combination in the first column. Vector $t_1$ includes the binary vectors {"010", "011"} and $t_2$ includes {"110", "111"}. There is no Boolean vector included in both sets. Therefore, $t_1$ is orthogonal with respect to $t_2$.
The number of rows in a TVL can be minimized. If two orthogonal ternary vectors (two lines) of a TVL differ in exactly one column, in which the combination 0/1 is present, both TV's can be combined into one TV. For example, the ternary vectors "0-0" and "1-0" have the 0/1 combination in the first column and can be merged into the ternary vector "- - 0".
Each ternary element $\boldsymbol{t_{ij}}$ is encoded by two bits – $\boldsymbol{a_{ij}}$ and $\boldsymbol{b_{ij}}$ (see Table 1). Each ternary vector $\boldsymbol{t_i} = \{ \boldsymbol{t_{ij}}, j=1, \dots, k \}$, consist also of two parts - $\boldsymbol{a_i} = \{ \boldsymbol{a_{ij}}, j=1, \dots, k \}$ and $\boldsymbol{b_i} = \{ \boldsymbol{b_{ij}}, j=1, \dots, k \}$, where $k$ is the number of variables. Both parts of a ternary vector are stored in two machine words and placed in the memory one after another, if $k$ is smaller than or equal to the width of a machine word of the computer. If the number of Boolean variables is larger than a machine word and we need an even number of machine words, represented by the variable *typ,* to store one ternary vector, $a$ and $b$ parts will consist of *typ*/2 components $A_m$ corresponding to $B_m$, stored as shown in Figure 1.

Table 1. Encoding of ternary elements

| Ternary element $\boldsymbol{t_{ij}}$ | $\boldsymbol{a_{ij}}$ | $\boldsymbol{b_{ij}}$ |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 0 | 1 |
| - | 0 | 0 |

Using this encoding of the ternary elements, the following Lemma can be formulated:

**Lemma 1.** Two ternary vectors can be merged into one block if their $b$-parts are equal and their $a$-parts are different only in one place.

**Proof**. We consider two ternary vectors that are different in exactly one column, in which the combination 0/1 is present. Using the definition of block building, only such vectors can create a block. Assume their $b$-parts are not equal. It means that a combination 0/- or 1/- (see Table 1) must occur at least in one column. If $a$-parts were different in more than one place, then both vectors would have more than one difference. ■

Most of the operations needed to check whether two ternary vectors are different in exactly one place can be calculated for $n$ variables in parallel, if $n$ is smaller than or equal to the width of a machine word. If the number of Boolean variables is larger than a machine word, we have to repeat these operations at most *typ*/2 times (see Figure 1).

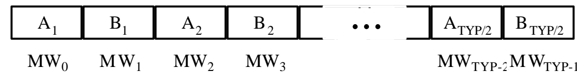


Figure 1. Storage structure of a ternary vector.

# 3. Properties of the Boolean Space

A Boolean space $\boldsymbol{B^k}$ may be represented by the set of all $k$-digit Boolean vectors.

$$B^k = \{ \boldsymbol{x} \mid \boldsymbol{x} = (x_1, x_2, \dots, x_k), x_i \in \{0,1\}, i = 1,2,\dots,k \} \quad (1)$$

Each variable can, independently of all other variables, take exactly one value from the set {0, 1}. Therefore, the number of all vectors of $\boldsymbol{B^k}$ is $2^k$. For two Boolean variables, the operation "smaller than or equal to" is defined as shown in Table 2.
Using this operation we can define a half order relation over the Boolean vectors of $\boldsymbol{B^k}$. For Boolean vectors $\boldsymbol{x} = (x_1, x_2, \dots, x_k)$ and $\boldsymbol{y} = (y_1, y_2, \dots, y_k)$ is true

$$\boldsymbol{x} \leq \boldsymbol{y} \Leftrightarrow x_i \leq y_i, \forall i = 1, \dots, k \quad (2)$$

Figure 2 shows this half ordering relation for the sets of Boolean vectors of $\boldsymbol{B}^2$ and $\boldsymbol{B}^3$. For a better overview the reflexive and transitive edges are not included in the graphs. Note that Boolean spaces include pairs of vectors, which are not connected by an edge. Thus we cannot use this relation for our ordering task.

The first model to order all Boolean vectors may be the set of decimal equivalents of the Boolean vectors; {“00”, “01”, “10”, “11”} $\Rightarrow$ {0, 1, 2, 3}.

We can order the Boolean vectors using the decimal equivalents. But it is not possible to generalize this model for ternary vectors, because they can define overlapping Boolean subspaces. For example the ternary vector “-01” represents the decimal equivalents {1, 5} and the ternary vector “0-0” represents the decimal equivalents {0, 2}. There does not exist an order between these sets of decimal equivalents.

Another model for ordering uses the number of ones in the Boolean vectors. All Boolean vectors with the same number of ones belong to the same *class*. Each class is labeled by the number of ones. Figure 3 shows these 5 classes for $\boldsymbol{B}^4$.

Using $k$+1 classes $C_{o_b}$, where the index $o_b$ is the number of ones in the class $C_{o_b}$, Boolean space may be represented by:

$$\boldsymbol{B}^k = \{ C_{o_b}, o_b \in \{0, 1, 2, \dots, k\} \} \qquad (3)$$

$$C_{o_b} = \{ \boldsymbol{x} / \boldsymbol{x} = (x_1, x_2, \dots, x_k) \text{ and } x_{j1}, x_{j2}, \dots, x_{jo_b} = 1, \text{ all other } x_i = 0 \} \qquad (4)$$

Table 2. Smaller than or equal to operation for Boolean variables

| $x_i$ | $y_i$ | $x_i \le y_i$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

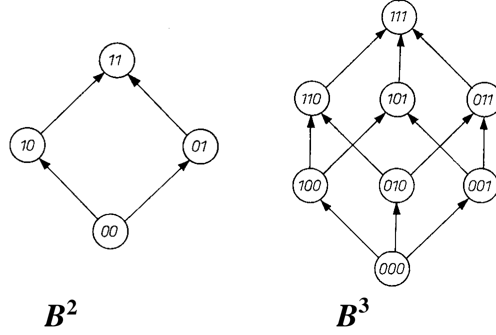


Figure 2. Half ordered sets $\boldsymbol{B}^2$ and $\boldsymbol{B}^3$

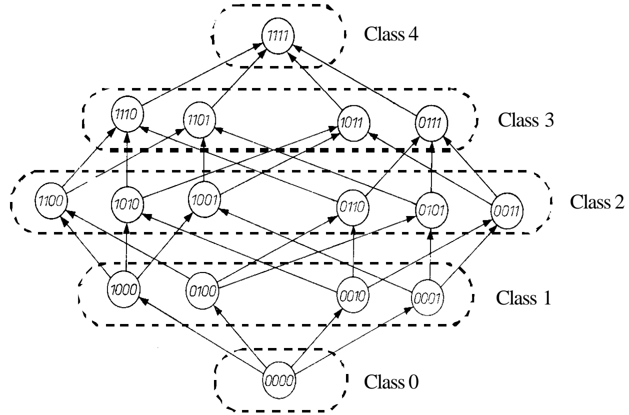


Figure 3. Classes of Boolean vectors in $\boldsymbol{B}^4$

# 4. Ordered Lists of Ternary Vectors

## 4.1. Basic Idea

We will generalize the above method for TVL. The number of ones #$o$ in a ternary vector can also be used for their classification.

$$C_o = \{ \boldsymbol{t} / \boldsymbol{t} = (t_1, t_2, \dots, t_k) \text{ and } t_{j1}, t_{j2}, \dots, t_{jo} = 1, \text{ all other } t_i \in \{0, -\} \} \qquad (5)$$

Each ternary vector represents a set of Boolean vectors. If the ternary vector includes #$s$ strokes, the number of ones #$o_b$ in the Boolean vectors of this set is defined by

$$\#o \le \#o_b \le (\#o + \#s) \ . \qquad (6)$$

Because the upper bound depends of the number of strokes #$s$, we can use this number to create (#$s$+1) subclasses $SC_{os}$.

$$C_o = \{ SC_{os}, s \in \{0, 1, 2, \dots, k\text{-}o\} \}, \qquad o \in \{0, 1, 2, \dots, k\} \qquad (7)$$

$$SC_{os} = \{ \boldsymbol{t} / \boldsymbol{t} = (t_1, t_2, \dots, t_k) \text{ and } t_{j1}, t_{j2}, \dots, t_{jo} = 1,\ t_{m1}, t_{m2}, \dots, t_{ms} = \text{‘}-\text{’}, \text{ all other } t_i = 0 \} \qquad (8)$$

This classification into classes and subclasses may be used to reduce the number of necessary comparisons to find all pairs of vectors that can be combined into one ternary vector.

As mentioned above, two ternary vectors, which have combination 0/1 in exactly the same place, can be merged into one block. Such a pair of ternary vectors must be taken from two special subclasses.

**Lemma 2.** In an ordered model of classes and subclasses of ternary vectors only vectors from neighbor classes that belong to the subclasses with the same number of strokes have the property of producing a block.

**Proof**. We consider two ternary vectors from subclasses $SC_{o1s1}$ and $SC_{o2s2}$ that have the property to build a block. Inasmuch as they are different only with combination 0/1, they must have the same number of strokes, so $s1 = s2$. Since they have only one difference, the number of ones of one vector differs from the number of ones of other vector only by one, so $| o1\text{-}o2 | = 1$. ■

## 4.2. Ordering of vectors

In Figure 4, we show an unordered TVL with the numbers of ones (#*o*) and the numbers of strokes (#*s*) in the first two columns. In Figure 5, the same TVL is sorted by QuickSort algorithm.
We decide on the order in two levels. On the first level we order from lower to higher number of ones #*o* into classes. On the second level we order the vectors of each class from the higher to the lower number of strokes #*s* into subclasses. More information on the TV-ordering can be found in [8]. As a result, we receive an ordered list of ternary vectors that can be represented as the series of classes and subclasses. The vectors, which belong to the same subclass, are stored as separate TVL's (Sub_TVL). In our example the ordered TVL consist of 6 classes and 12 subclasses.

| #*o* | #*s* | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | 0 | 1 | 0 | - | - |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 3 | - | - | 0 | 0 | 0 | - |
| 3 | 1 | 1 | 1 | 0 | 0 | 1 | - |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | - | 0 | 0 | 1 | 0 | 0 |
| 4 | 2 | 1 | 1 | 1 | - | - | 1 |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 1 | 1 | 1 | - | 1 |
| 3 | 1 | 0 | 1 | - | 0 | 1 | 1 |
| 2 | 1 | 0 | - | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 1 | 0 | 1 | - |
| 4 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 3 | 1 | - | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Figure 4. Unordered TVL with number of ones and strokes for each vector

| #*o* | #*s* | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ |
|---|---|---|---|---|---|---|---|
| 0 | 3 | - | - | 0 | 0 | 0 | - |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | 1 | - | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| | | 1 | 0 | 0 | 0 | 1 | 0 |
| | | 0 | 0 | 0 | 1 | 0 | 1 |
| | 1 | 0 | 0 | 1 | 0 | 1 | - |
| | | 0 | - | 1 | 0 | 0 | 1 |
| | 2 | 1 | 0 | 1 | 0 | - | - |
| 3 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| | | 1 | 0 | 0 | 1 | 1 | 0 |
| | | 0 | 0 | 0 | 1 | 1 | 1 |
| | 1 | - | 0 | 1 | 1 | 0 | 1 |
| | | 1 | 1 | 0 | 0 | 1 | - |
| | | 0 | 1 | - | 0 | 1 | 1 |
| 4 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| | 1 | 0 | 1 | 1 | 1 | - | 1 |
| | 2 | 1 | 1 | 1 | - | - | 1 |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |



Figure 5. Ordered TVL as series of classes or subclasses

## 4.3. Data Structure

For the fast access to the classes and subclasses we create a triangular matrix structure outside of the ordered TVL. The number of rows and columns of this matrix is bounded by $k$+1 rows and columns, where $k$ is the number of variables. The elements of this array are pointers on the Sub_TVL's. The index of a row corresponds to the number of ones in Sub_TVL of this row. Thus each row of the matrix corresponds to one class. The column index of a column corresponds to the number of strokes in each Sub_TVL. If some Sub_TVL's does not exist, NULL is stored in the appropriate array item.
The whole matrix structure of our example from Figure 4, where a TVL is represented as 12 subclasses, is shown in Figure 6.
Similarly, this structure can also represent a BVL.

number of strokes

| number of ones | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | NULL | NULL | NULL | 1 | NULL | NULL | NULL |
| 1 | 3 | 2 | NULL | NULL | NULL | NULL | |
| 2 | 6 | 5 | 4 | NULL | NULL | | |
| 3 | 8 | 7 | NULL | NULL | | | |
| 4 | 11 | 10 | 9 | | | | |
| 5 | 12 | NULL | | | | | |
| 6 | NULL | | | | | | |

Figure 6. Matrix structure

The matrix for BVL will have pointers to Sub_BVL's only in the first column, because the Boolean vectors contain no strokes.

# 5. Algorithms

We consider an orthogonal TVL or BVL. All next algorithms are developed for TVL, but they are true for BVL too. We check whether two ternary vectors *v1* and *v2* can build a block using the OBB (*v1*, *v2*) algorithm. As mentioned in Lemma 1, we compare at first *b*-parts of vectors that must be equal. If the *b*-parts are equal, the *a*-parts will be compared, and should be different only in one column. If both conditions are fulfilled, the block building between *v1* and *v2* is possible and the *new_vect* will be composed. This vector contains a stroke in the column, where *v*1 and *v*2 had the 0/1 combination. Figure 7 shows the block scheme of this algorithm.
It is necessary to find all pairs of ternary vectors, which make block building possible. We have to take into account that all new vectors can again build a new block. We consider four different ways to check all pairs of such vectors.

**Method 1.**
Each vector *v1* from *tvl* is compared with all next standing vectors *v2*. If *v1* and *v2* can create a block, the algorithm OBB (*v1*, *v2*) returns a *new_vect*. Otherwise give the algorithm NULL back and *v1* will be compared with the next vector *v2*. If *new_vect* exist, it is saved at the end of *tvl*, while the original ternary vectors *v1* and *v2* are deleted. Vector *v1* is allocated to the next vector from *tvl*, that will be compared with all next standing vectors *v2* until new block will be build or *v2* got at the last vector in *tvl*. If *v1* is assigned to the last vector in *tvl* and during this cycle of algorithm has built at least one new block, the algorithm will be repeated one more time. If no more block building is possible, the algorithm stops (see Figure 8).

**Method 2.**
This algorithm was proposed in [2, 5, 7 ] and is part of the XBOOLE system.
It uses two lists of ternary vectors: original *tvl* and *help_tvl*, containing vectors that cannot create one more block.

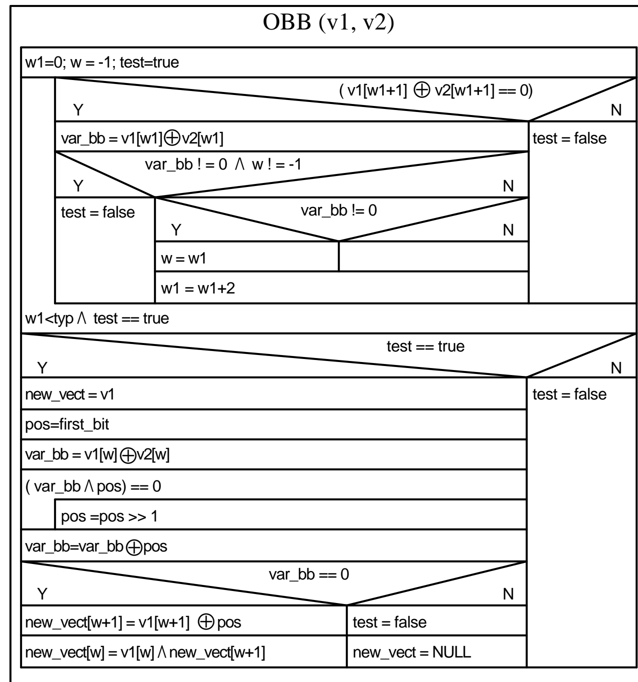

Figure 7. Test of block building between two vectors *v1* and *v2* and building a *new_vect* algorithm

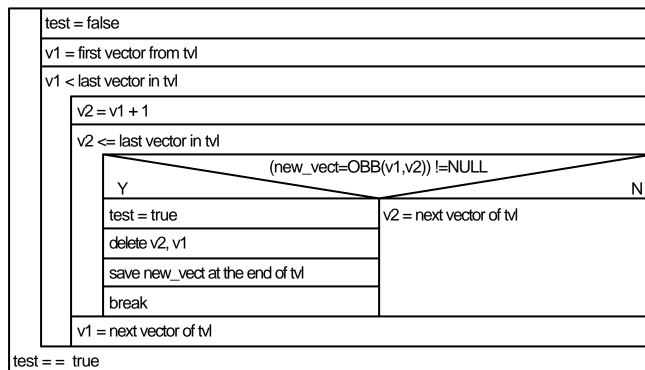

Figure 8. Block building-algorithm using method 1.

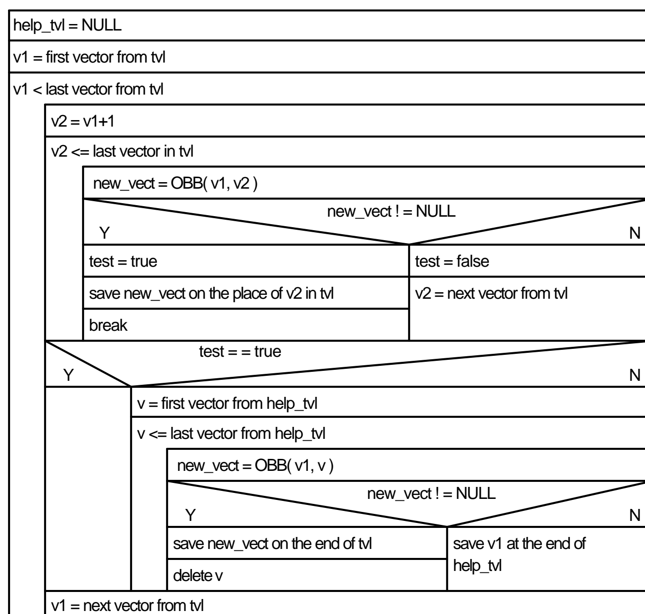

Figure 9. Block building-algorithm using method 2.

At first, each vector *v1* from *tvl* is compared with all the next standing vectors *v2*. As in the previous algorithm, it is checked whether *v1* and *v2* can create a block using algorithm OBB(*v1*, *v2*). If block building was possible, *new_vect* is saved in the place of *v2*, and vector *v1* is allocated to the next vector from *tvl*. Note that all ternary vectors located in the *tvl* before *v1* are throw away when the algorithm terminates. If *v1* cannot be combined with any vectors *v2* from *tvl*, it will be compared with all vectors from *help_tvl*. If again no block building was possible, *v1* is saved on the end of *help_tvl*. If *v1* had build a block with a vector *v* from *help_tvl*, *new_vect* is saved on the end of *tvl*. Vector *v* from *help_tvl* is deleted. The algorithm is repeated, until *v1* is not the last vector in *tvl*. In the end, *help_tvl* only vectors, which cannot build a block. The block scheme to algorithm 2 is show in Figure 9.

**Method 3.**

This algorithm (see Figure 10) was suggested in [3]. In offered paper we propose classification approach represent a TVL as a set of classes $C_o$,

$$C_o = \bigcup_s SC_{os} \Big|_{o=\text{const}} , \qquad (9)$$

where each class comprises more subclasses and the feature for class is their number of ones $o$ (see (7), (8), Figure 6). In contrast to this classification in algorithm from [3] each TVL is merged in another way, and represented as a set of *σ-sets* $S_s$

$$S_s = \bigcup_o SC_{os} \Big|_{s=\text{const}} . \qquad (10)$$

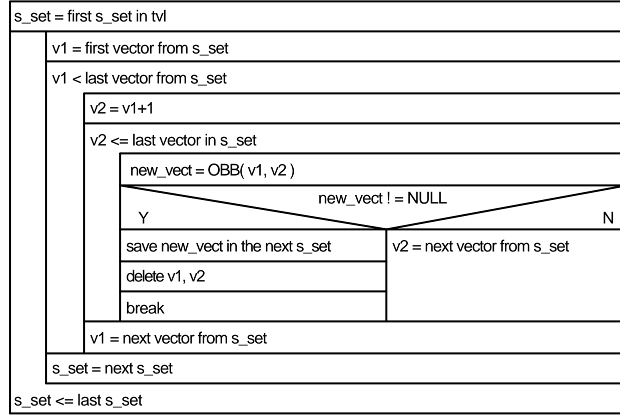


Figure 10. Block building-algorithm using method 3.

where σ-set is characterized by the number of strokes *s*. Using the matrix structure in Figure 6, we see that subclasses from the same column belong to one σ-set. For example, subclasses 3, 6, 8, 11, and 12 belong to σ-set with number of strokes 0. Using the Lemma 2, only vectors from the same σ-set can build a block. So it is necessary to check only vectors from the same σ-set in order to find possible block buildings. The vectors from TVL are sorted according to the number of strokes and are built maximal $k$+1 σ-sets, where $k$ is the number of variables in TVL.

For every σ-set, like method 1, each vector *v1* is compared with all next standing vectors *v2* using algorithm OBB(*v1*, *v2*). If block building was possible, *new_vect* contains one more stroke and is saved in corresponding σ-set. If such σ-set does not exist, it is created. The original vectors *v1* and *v2* are deleted. The vector *v1* is allocated to next vector from active σ-set. If *v1* and *v2* cannot be combined, *v1* will be checked with next vector *v2*. If vector *v2* gets the last vector from active σ-set, vector *v1* is allocated to the next vector from active σ-set and check-loop is repeated. If vector *v1* reaches the last vector from active σ-set, algorithm is repeated for the next σ-set (see Figure 10).

**Method 4.**

In Figure 11 we show our new algorithm, which takes into account the ordered data structure, described in section 4. According to Lemma 2, only ternary vectors from Sub_TVL belonging to the neighboring classes and having the same numbers of strokes can be combined into a block. If we use the matrix structure from Figure 6, only vectors from Sub_TVL that belong to the neighboring

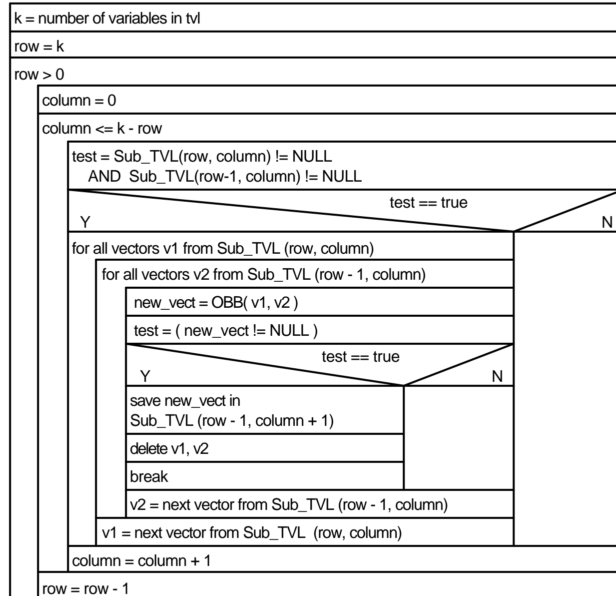


Figure 11. Block building-algorithm using method 4.

rows of the same column have the property for block building.

Our block building algorithm starts with Sub_TVL(*row, column*) where *row* is the last row in matrix and *column* is the first column in this row. We check whether this Sub_TVL and the corresponding Sub_TVL(*row*- 1, *column*) from the neighboring row are not equal NULL. If the subclasses are not NULL, the algorithm runs OBB(*v1*, *v2*) for each vector *v1* from Sub_TVL( *row, column*) and each vector *v2* from Sub_TVL(*row* -1, *column*). In case of successful block building, the resulting vector *new_vect* contains the same number of ones and one more stroke, compared to vectors in Sub_TVL(*row* -1, *column*). So it has to be added to Sub_TVL(*row* - 1, *column*+1). If such Sub_TVL does not exist, it will be created.

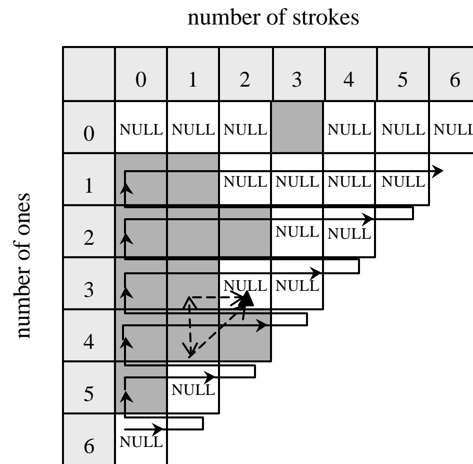


Figure 12. Direction of selection of subclasses by method 4 of block building

The algorithm iterates through the internal loop over Sub_TVL(*i, j*) by increasing the column number *j* and through the external loop over Sub_TVL by decreasing the row number *i* (see Figure 12). It is easy to see that all the new vectors will be considered in the next loop of the block building algorithm.

For example, subclass (5, 0) is compared with subclass (4, 0), (4, 0) with (3, 0), and (4, 1) with (3, 1). If the last comparison led to block building, a new vector belonging to Sub_Tvl (3, 2) is created. In the next iteration of the internal loop, the new vector will be checked with Sub_TVL (4, 2). The algorithm terminates when it reaches the first row in matrix or the first class in the TVL, respectively.

# 6. Results

To compare the efficiency of the four different versions of block building algorithms, we have run each of them for TVL's of 15 Boolean variables with different numbers of vectors and with various percentages of strokes and have calculated the numbers of necessary comparisons between pairs of vectors to perform block building. The results are shown in Figure 13.

In Table 3, we summarize the results. Column #1 gives the number of comparisons using method 1. It is the most elementary method so we use #1 as the basis of all percentages. Column #2 is the number of comparisons, if we use variant 2 of the block building algorithm, which is employed in system XBOOLE. Column #3 is the number of comparisons, if the TVL is ordered according to the number of strokes, form the $k$+1 σ-sets of ternary vectors and use the algorithm 3 to realize the block building. Column #4 is the most interesting case and describes the number of comparisons, if the TVL is ordered by classes and

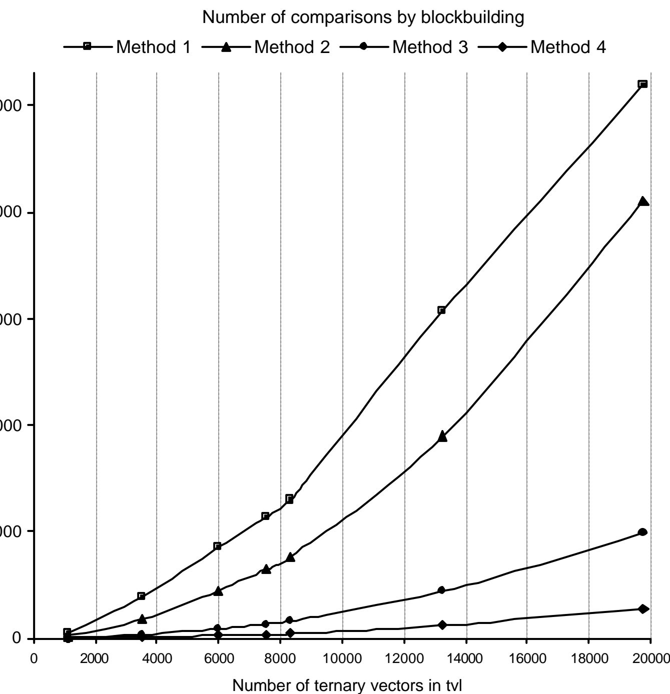


Figure 13. Number of comparisons between two vectors using different block building algorithms.

subclasses and processed by the new faster algorithm 4. Value ρ is the number of binary vectors in TVL divided by $2^k$, where $k$ is the number of variables. The percentages are defined as follows:

P1 = (#2 / #1) × 100%
P2 = (#3 / #1) × 100%
P3 = (#4 / #1) × 100%

Value P1 shows the trade-off, if instead of algorithm 1 the optimized algorithm 2 is used.
Percentages P2 and P3 show that ordered TVL needs fewer comparisons. The most interesting result is P3, which corresponds to algorithm 4. In case of ordering by the number of strokes and ones in the TVL, our new block building algorithm shows the best performance.
The number of resulting vectors is not constant for the four different algorithms (see Table 4). The result depends on the starting position in the list of ternary vectors and on the direction of comparison. In our example given in Figure 4, vector "100010" can create a block with three other vectors: "100011", "100110", "000010". After block building, we will have the following vectors: "10001-", "100-10", "-00010". Comparison shows that the first vector can be combined only with one other vector, "11001-". As a result, a new vector "1-001-" is created and the total number of vectors is decreased by two. If the vectors were considered in a different order, the last vector would not be created, and the total number of vectors would only be reduced by one. Thus the position of vectors that can be combined to one block, is very important, because all algorithms perform the first possible block building. So if ordering creates a favorable sequence of vectors, the algorithm with ordering is better. Each of our algorithms works better for some of the TVL's (underlined in Table 4).

Table 3. The performance of different block building algorithms

| Number of ternary vectors | ρ | #1 | #2 | #3 | #4 | P1 % | P2 % | P3 % |
|---|---|---|---|---|---|---|---|---|
| 1134 | 0,196 | 2329578 | 1030866 | 136071 | 36321 | 44,25 | 5,84 | 1,56 |
| 3542 | 0,375 | 19446628 | 9079892 | 1580582 | 430468 | 46,69 | 8,13 | 2,21 |
| 5989 | 0,557 | 42423221 | 22226876 | 4420192 | 1190972 | 52,39 | 10,42 | 2,81 |
| 7536 | 0.836 | 56843345 | 32095272 | 6624524 | 1692232 | 56,46 | 11,65 | 2,98 |
| 8325 | 0,879 | 64832378 | 37936553 | 7993770 | 2091618 | 58,51 | 12,33 | 3,23 |
| 13270 | 0,999 | 153715654 | 94958824 | 21924275 | 5861539 | 61,78 | 14,26 | 3,81 |
| 19477 | 0,991 | 257899999 | 202531498 | 48292306 | 13646295 | 78,53 | 18,73 | 5,29 |
| 19594 | 0,995 | 255188105 | 203099382 | 48523606 | 13810574 | 79,59 | 19,01 | 5,41 |
| 19760 | 0,995 | 259729177 | 205273460 | 49127749 | 13859861 | 79,03 | 18,91 | 5,34 |

Table 4. The number of the resulting ternary vectors

| Number of ternary vectors | ρ | #1 | #2 | #3 | #4 |
|---|---|---|---|---|---|
| 1134 | 0,196 | 935 | 930 | 937 | <u>929</u> |
| 3542 | 0,375 | 2670 | 2653 | <u>2664</u> | 2670 |
| 5989 | 0,557 | 3868 | <u>3826</u> | 3876 | 3835 |
| 7536 | 0.836 | 4367 | 4356 | 4418 | <u>4334</u> |
| 8325 | 0,879 | 4639 | 4649 | 4771 | <u>4622</u> |
| 13270 | 0,999 | 7089 | 7173 | 7306 | <u>7028</u> |
| 19477 | 0,991 | 10111 | 10506 | 10499 | <u>10066</u> |
| 19594 | 0,995 | 10046 | 10423 | 10459 | <u>9992</u> |
| 19760 | 0,995 | 10143 | 10529 | 10634 | <u>10025</u> |

# 7. Conclusion and Future Work

We have developed a faster algorithm for block building in orthogonal TVLs using ordering of ternary vectors into classes and subclasses. Classes are defined by the numbers of ones in a vector and subclasses by the number of strokes. First, our experimental results show that the approach that is based on sorting TV's into classes and subclasses needs significantly less comparisons than approaches without sorting or with sorting into σ-set only. Second, the order of comparisons in our algorithm guarantees that the new vector is stored in a subclass that will be considered for minimization in the same sweep of the algorithm. This leads to a more efficient minimization of Boolean functions.

Our algorithm is better in terms of the number of comparisons, because only ternary vectors from fitting subclasses are compared. Additionally our algorithm is better in terms of the size of the result TVL, because all new developed vectors are useful for the next block building steps.
In the future work, we will generalize the new algorithm to perform reshaping of vectors in the list without increasing their number, to facilitate new block building opportunities. Besides we will adapt the new approach based on the orthogonal ordered lists of ternary vectors to other Boolean operations.

# 8. References


[1] Bochmann, D.; Posthoff, C.: Binäre dynamische Systeme. Akademie-Verlag, Berlin, 1981

[2] Bochmann, D.; Steinbach, B.: Logikentwurf mit XBOOLE. Verlag Technik Berlin, 1991.

[3] Hesse, K.: Ein Beitrag zur Lösung hochdimensionaler Boolescher Probleme mit parallelen Algorithmen. Dissertationsschrift, Chemnitz, Technische Universität 1998

[4] Kempe, G., Steinbach, B.: Vergleich der Darstellungen einer Booleschen Funktion als TVL und ROBDD. Tagungsunterlagen des Workshops "Boolesche Probleme", Freiberg, 1994, S. 25 -32

[5] Posthoff, Ch.; Steinbach, B.: Binäre Gleichungen - Algorithmen und Programme. Wissenschaftliche Schriftenreihe der Technischen Hochschule, 1/1979, Karl-Marx-Stadt, 1979

[6] Steinbach, B.; Le T. Q.: Tools für den Logikentwurf. Wissenschaftliche Schriftenreihe der Technischen Universität 9/1988, Karl-Marx-Stadt 1988

[7] Steinbach, B.: „XBOOLE - A Toolbox for Modelling, Simulation, and Analysis of Large Digital Systems. System Analysis and Modelling Simulation”, Gordon & Breach Science Publishers, 9(1992), Number 4, pp. 297 – 312

[8] Steinbach B., Dorotska K. Fast Boolean calculations using ordered lists of ternary vectors // Proceedings of the Third International Conference on Computer-Aided Design of Discrete Devices (CAD DD'99), Volume 1, pp. 20 - 27, pp. 20 - 27, November 1999, Minsk, Belarus