On the Sensitivity of Boolean and Multiple-Valued Symmetric Functions

Jon T. Butler

Department of Electrical
and Computer Engineering
Naval Postgraduate School
Monterey, CA U.S.A. 93943-5121

Tsutomu Sasao

Department of Computer Science
and Electronics
Meiji University
Kawasaki, Kanagawa 214-8571 JAPAN

Abstract

An n-variable Boolean logic function $f(\vec{x})$ is sensitive to x_i if there is at least one assignment of values to $\vec{x} - \{x_i\}$ such that f changes when x_i changes. We investigate the sensitivity of Boolean logic functions experimentally. For example, we show the use of a reconfigurable computer in computing the sensitivity of n-variable Boolean functions with up through $n = 5$ variables. For $n = 5$, this computation is 193 times faster than a single Xeon microprocessor and 1.8 times faster than a cluster computer with 256 processors. We also examine sensitivity in multiple-valued logic functions.

Index Terms: Sensitivity, Boolean function complexity, symmetric functions, reconfigurable computer, P equals NP.

1 Introduction

The question "Does P equal NP?" has been the focus of much research since it was posed by Cook [4] in 1971. It has been suggested that the answer will likely come from a better understanding of Boolean functions. Indeed, Sipser [10] says "Many researchers consider circuit complexity to be the most viable direction for resolving the P versus NP question". There is much research on the complexity of Boolean functions in the realization of specific classes of functions, including symmetric and monotone functions. Complexity measures include the number of products in minimum sum-of-products expressions, length of the shortest path in a decision diagram (also called query length), and sensitivity. We say that the i-th bit of an input vector is **sensitive** if complementing the bit complements the function. We say that an n-variable function $f(\vec{x})$ is **sensitive** to x_i if there exists at least one assignment of values to $\vec{x} - \{x_i\}$ (also written as $\vec{x} \setminus \{x_i\}$) such that f changes when x_i changes. The **sensitivity** of the input vector is the number of bits for which it is sensitive. The **sensitivity** of the function is the maximum of the sensitivities among its input vectors.

Readers familiar with the concept of Boolean differences may recognize a resemblance to sensitivity. In spite of this resemblance, there has been little, if any, cross research. One goal of this paper is to introduce logic designers to this interesting topic.

Cook, Dwork, and Reischuk [5] suggest sensitivity as a complexity measure of Boolean functions CREW PRAM (Concurrent Read Exclusive Write Programmable Random Access Machine), a model for multi-processors, each processor with a RAM, and a common tape. Nisan [7] extended this concept to block sensitivity and showed that CREW PRAM complexity and block sensitivity are polynomially related; i.e. the time complexity of one can be bounded above by a polynomial of the time complexity of the other. Similarly, block sensitivity has been shown to be polynomially related to other Boolean function complexity measures, such as query length complexity. However, no one had, up to that point, shown that sensitivity is polynomially related to other complexity measures. This left open the question of whether there exists a function with exponential sensitivity. In 2019, Huang [6] answered this question by proving that sensitivity and block sensitivity are polynomially related. In 2017, Amano [1] computed the number of NPN classes [9] of functions with sensitivity 3 for n-variable functions for $n \geq 3$. A discussion of research in sensitivity up to 2005 appears in Chakraborty [3].

2 Background

Definition 2.1. *Let $f\{0,1\}^n \rightarrow \{0,1\}$ be a Boolean logic function, and let $\vec{x} \in \{0,1\}^n$ be an n-bit input vector. Input vector $\vec{y}$ is a* **neighbor** *of $\vec{x}$ if $\vec{y}$ differs by exactly one bit from $\vec{x}$. $s(f,\vec{x})$, the* **sensitivity** *of $\vec{x}$, is the number of neighbors $\vec{y}$ of $\vec{x}$, such that $f(\vec{x}) \neq f(\vec{y})$.*

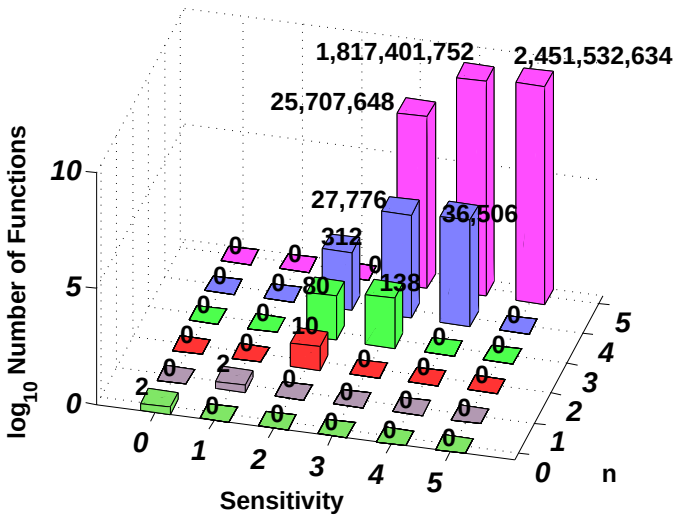


Figure 1. Distribution of Non-degenerate Boolean Logic Functions According to Sensitivity and Number of Variables n.

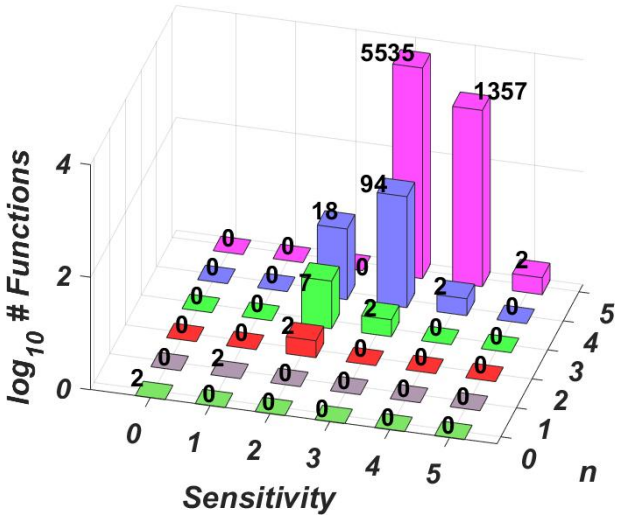


Figure 2. Distribution of Non-degenerate Monotone Boolean Logic Functions According to Sensitivity and Number of Variables n.

Example 2.1. *Note that* $0 \leq s(f,\vec{x}) \leq n$. *For example, if* f *is the AND function on* $n \geq 2$ *variables, then for* $\vec{x} = 11\ldots1$, *all bits are sensitive, and* $s(f,\vec{x}) = n$. *On the other hand, for* $\vec{x} = 00\ldots0$, $s(f,\vec{x}) = 0$. *In an example by Ramsey [8],* f *is the totally symmetric 8-variable function that is 1 iff 4 or 5 variables are 1. For* $\vec{x} = 11100000$, $s(f,\vec{x}) = 5$, *since changing any of the five 0's to a 1 changes* f, *and changing any 1 to 0 leaves* f *unchanged.*

Definition 2.2. *The* **sensitivity** *of a function* f, $s(f)$, *is the maximum of* $s(f,\vec{x})$ *across all choices of* $\vec{x}$. *The* **sensitivity** $s(F)$ *of a set of functions* F *is the minimum of* $s(f)$ *over all* $f \in F$.

Example 2.2. $s(f) = n$, *where* f *is the* n*-variable AND function, since* $\vec{x} = 11\ldots1$ *has* $s(f,\vec{x}) = n$, *the maximum. In the case of Ramsey's function* f, $s(f) = 6$, *since* $s(f,\vec{x})$ *is maximum when* $\vec{x} = 11111100$.

The term "critical complexity" has been used instead of "sensitivity" (e.g. Wegener [12]), but "sensitivity" seems to be more commonly used now.

A Verilog program was written to enumerate n-variable functions for $2 \leq n \leq 5$ on an SRC-6 reconfigurable computer (viz. Butler [2]) according to their sensitivity. It was compiled by Synplify Pro® and run on a 100 MHz Xilinx-II FPGA. We chose to do the computation on an FPGA because other computations have performed well in similar scientific investigations. For example, to count the functions with various sensitivities, a reconfigurable computer uses an FPGA to implement a hardware counter for each sensitivity. Because the implementation is so efficient compared to a software program, a 100 MHz FPGA performs much faster than a single processor. This same computation was done on an Intel Pentium Xeon 2.8 GHz single CPU microprocessor from a program written in C.

3 Experimental Results

Fig. 1 shows the result. It should be noted that the functions depend on *all* variables (are *non-degenerate functions*). So, for example, while there are $2^{2^2} = 16$ functions on two or fewer variables, Fig. 1 shows only 10, which are the 10 that depend on both variables. Similarly, Fig. 2 shows the distribution of *monotone* Boolean logic functions according to their sensitivity. Again, all functions in the distribution depend on *all* n variables. There are considerably fewer functions with this restriction. Note that there are exactly two functions with largest sensitivity n. These are the AND and OR function on all variables.

Tables 1 and 2 show the computation times. Two times are shown, the total computation time of all functions and the average computation times per function. As n increases, this approaches 10.0 ns for functions computed by the SRC-6. This reflects the fact that for large circuits on the SRC-6, we can still achieve a sensitivity computation of one per clock period, where the clock period of 10 ns corresponds to a 100 MHz clock speed. Table 2 shows how the SRC-6 compares to a cluster machine of AMD processors when 1, 4, 16, 64, and 256 processors are used. These times are *walltime*s and are affected by the running of other jobs

Table 1. Computation Times of Sensitivity by SRC-6 and a Xeon Microprocessor

	All Functions		Each Function		
n	SRC-6	Xeon	SRC-6	Xeon	Speedup
2	970 ns	2120 ns	60.6 ns	133.8 ns	2×
3	3,430 ns	110,000 ns	13.4 ns	429.7 ns	32×
4	656,330 ns	62,100,000 ns	10.0 ns	947.6 ns	95×
5	42.95 sec	8267.7 sec	10.0 ns	1925.0 ns	193×

Table 2. Computation Time for Sensitivity of a Function Using the SRC-6 Reconfigurable Computer and an AMD Cluster Machine (Times are the Average Per Function Over All n-Variable Functions)

# of Variables	Reconfig. Computer	AMD Cluster Machine Number of Processors				
n	SRC-6	1	4	16	64	256
2	60.6 ns	188.4 ns	60.9 ns	33.5 ns	*	*
3	13.4 ns	433.1 ns	138.0 ns	50.1 ns	12.4 ns	3.3 ns
4	10.0 ns	1,028.1 ns	347.0 ns	129.1 ns	35.5 ns	9.1 ns
5	10.0 ns	2,463.3 ns	842.9 ns	317.6 ns	70.9 ns	18.4 ns

* This corresponds to more processors than there are n-variable functions.

on the system. We show the time required by the slowest processor, which is not as robust as the average processor time. So, one must view these times as approximate. However, scaling is evident. For example, if we fix the value of n, the number of variables, then there is a reduction to roughly 25% of the computation time as the number of processors increases by a factor of 4. That is, the computation is *embarrassingly parallel*. Specifically, computation of the sensitivity of any function is independent of the sensitivity computation of any other function. Therefore, the computation can be divided evenly so that the workload is perfectly balanced across all processors, provided that there are a power of 2 number of processors.

Also evident in the data of Table 2 is the fact that the computation time per function for the AMD cluster machine approximately doubles as n is increased by 1. Specifically, the sensitivity is computed by computing the sensitivity of *each* assignment of values to the variables, taking the maximum value. As n increases by 1, the number of assignments doubles. Therefore, the work doubles. It is interesting that the SRC-6 computation times remains constant (at 10 ns per function) as n increases by 1. The reason for this is that, each increase by 1 simply doubles the hardware needed to do the computation, and not the time. We are using only a small part of the FPGA, and the logic resources used are much less than the maximum available. If the maximum resources are reached, then the circuit would have to be configured so that the hardware operates serially on the assignments.

4 Multiple-Valued Symmetric Functions

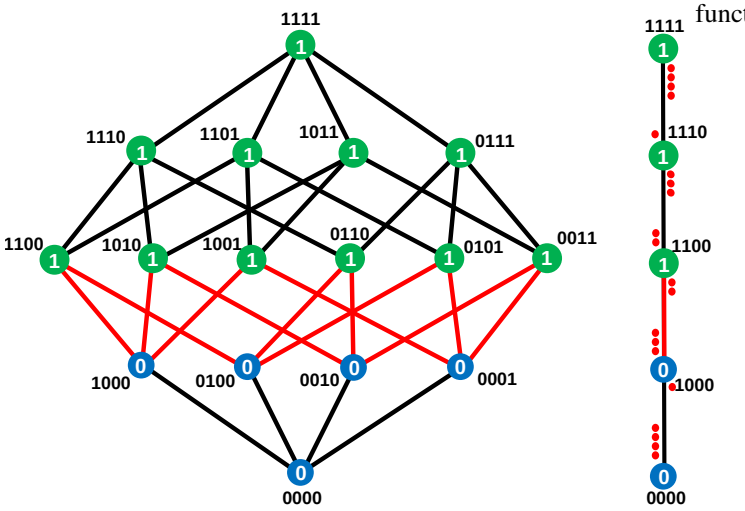
Definition 4.3. *Let $f\{0,1,\ldots,r-1\}^n \rightarrow \{0,1,\ldots r-1\}$ be a multiple-valued logic function, and let $\vec{x} \in \{0,1,\ldots,r-1\}^n$ be an n-digit input vector. Input vector $\vec{y}$ is a* **neighbor** *of $\vec{x}$ if $\vec{y}$ differs at exactly one digit from $\vec{x}$. $s(f,\vec{x})$, the* **sensitivity** *of $\vec{x}$, is the number of neighbors $\vec{y}$ of $\vec{x}$, such that $f(\vec{x}) \neq f(\vec{y})$. In tallying the sensitivity of $\vec{x}$, a digit contributes at most 1 even when the difference is more than 1. For example, if the function value of 210 differs from that of 110 and 010, the leftmost digit contributes 1 to the sensitivity of 210, not 2. Stated formally, the sensitivity of multiple-valued function f is $\max_{\vec{x}\in\{0,1,\ldots,r-1\}^n} s(f,\vec{x})$.*

Functions of interest in the complexity theory community include the symmetric functions, Nisan[7] and Wegener[12].

Definition 4.4. *A* **symmetric function** *is unchanged by any permutation of the variable values.*

Consider Fig. 3a. This shows a 4-variable symmetric Boolean function that is 1 iff two or more variables are 1. Here, each node corresponds to a single input vector. For example, the node labeled '0011' represents $x_1x_2x_3x_4 = 0011$. This node is assigned '1', which means that the function is 1 when $x_1x_2x_3x_4 = 0011$. This is shown by a green circle with a white 1 inside. The edges represent input vectors that are neighbors of each other. That is, 0011 is a neighbor of 0111, 1011, 0010, and 0001. Thus, 0011 is sensitive to two of the four variables, x_3 and x_4, because two

nodes, 0010 and 0001 map to 0, which is the complement of 0011's function value 1. The other two nodes for which 0011 is a neighbor, 1011 and 0111, map to the same function value as 0011, and so 0011 is not sensitive to a change in x_1 or x_2.



a) Lattice Representation

b) Compact

Figure 3. Lattice Representation of a Boolean Symmetric Function and its Compact Form.

Fig. 3b shows a compact version of Fig. 3a. In Fig. 3b, one node represents multiple nodes in Fig. 3a. For example, because Fig. 3a represents a symmetric function, the six nodes in Fig. 3a, 1100, 1010, 1001, 0110, 0101, and 0011 must map to the same function value; 1 in this example.

In Fig. 3b, all of these nodes are represented by one node, labeled 1100. As in Fig. 3a, the number in the circle of a node specifies the function value mapped to by the input vector(s) of that node.

The edges in Figs. 3a and 3b specify neighbors; the nodes at two ends of a edge are a distance 1 from each other and designate input vectors that are neighbors. The small red dots specify how the edges interconnect between nodes. For example, in Fig. 3b, two red dots extend down from the node 1100 to 1000, while three red dots extend up from 1000 to 1100. This specifies that node 1100 in Fig. 3a is a neighbor of two nodes with one fewer 1. Similarly, the three red dots extending up from node 1000 in Fig. 3b specify that each node with one 1 in Fig. 3a is a neighbor of three nodes each with two 1's in Fig. 3a.

From this information, we can determine the sensitivity of the function shown in Fig. 3b. First, note that the only edges that specify the sensitivity are between the 1100 and 1000 levels. All other edges connect two nodes that map to the same function value. So, complementing a variable value leaves the function unchanged. The maximum of the sensitivity between the 1100 and 1000 levels is 3, and so the function's sensitivity is 3. Note that a red edge connects two input vectors that map to different function values. A black edge connects two input vectors that map to the same function value.

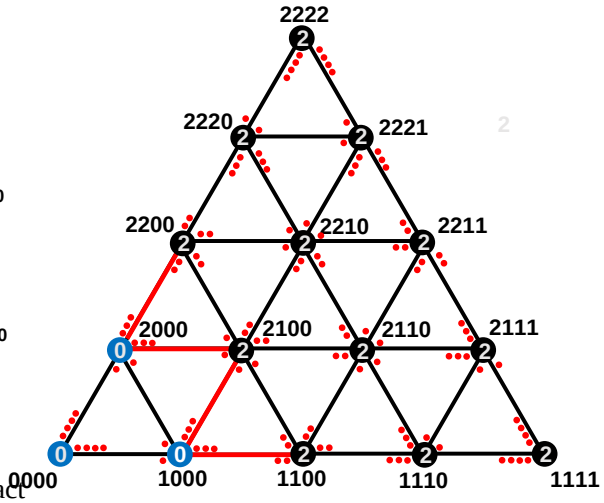


Figure 4. Lattice Representation of a 3-Valued 4-Variable Symmetric Function in Its Compact Form.

Fig. 4 shows a 3-valued 4-variable symmetric function using the compact representation. That is, each node represents all arrangements of the logic values shown. For example, **2200** corresponds to the six nodes, 2200, 2020, 2002, 0220, 0202, and 0022 in the full lattice representation of this function. Since the function is symmetric, $f(2200) = f(2020) = f(2002) = f(0220) = f(0202) = f(0022)$. Since one node, not six, are needed to represent its function value, it follows that the compact version of the lattice structure requires fewer nodes. Indeed, this structure requires 15 nodes. That is, each node in the compact structure represents one of $n = 4$ ways to choose $r = 3$ values with repetition. This is $\binom{n+r-1}{n} = \binom{4+3-1}{4} = 15$.

In this case, the edges, nodes, and red dots serve the same purpose as they do in Fig. 3b. A significant reduction in complexity has been achieved. For example, instead of $3^4 = 81$ nodes, only 15 nodes are needed to represent a 3-valued 4-variable *symmetric* function. The example function maps three nodes to 0 (shown as a blue circle) and 12 nodes to 2 (shown as black circles). No node maps to 1 (to remind the reader that this is a multiple-valued function).

Table 3 shows the distribution of sensitivity values for *all* 3-valued 4-variable *symmetric* functions. This was com-

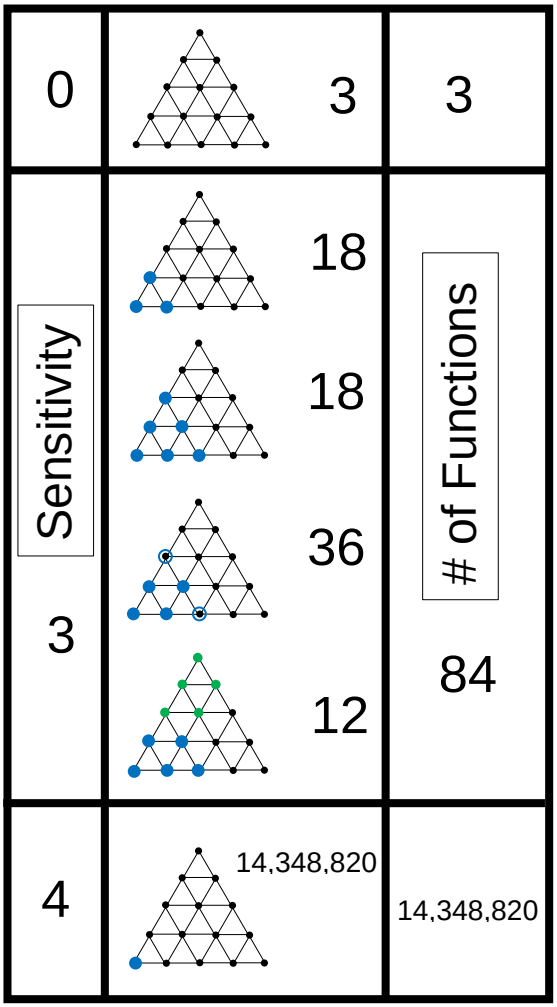


Table 3. Distribution of Sensitivities of 3-Valued 4-Variable Symmetric Functions

puted by a MATLAB program that enumerated all functions. Included are constant functions, in which all input vectors map to the same value, either 0, 1, or 2. Strictly speaking, these are not 4-variable functions. However, all other $3^{15} - 3$ functions are dependent on all four variables. Among these functions, the *smallest* sensitivity is 3. Thus, the sensitivity of 4-variable 3-valued functions is 3. One of these functions is shown in Fig. 4.

We count functions according to sensitivity, as follows. Consider first the function shown in Fig. 4. It produces two of a possible three function values. One is 0, which is produced by the three input vectors, **0000**, **1000**, and **2000**. The other is 2, which is produced by all of the other 12 input vectors. Each edge connects input vectors that differ in one (of four) input values. For example, the edge between **2100** and **1000** differs between a 2 in the former and a 0 in the latter. The three red dots on this edge specify that, in the underlying lattice, the single 2 of **2100** can replace one of three 0's in **1000**. Because the two input vectors map to different function values, they contribute three to this function's sensitivity. Similarly, **1000** and **1100** contribute 3 to the function's sensitivity. However, the combined sensitivity is 3, effectively the OR of the two sensitivities, since the **2** of **2100** or the leftmost **1** of **1100** can be replaced by a **0** of **1000** (This can be better seen by viewing **2100** as **1200**). From these observations, we can conclude that input vector **1000** has sensitivity 3. Similarly, we can conclude that **2000** also has sensitivity 3. Input vector **0000** has sensitivity 0 since all (both) edges incident to it are also incident to an input vector that maps to the same function value as **0000**. A similar argument concludes that **2100** has sensitivity 2. All other input vectors are adjacent to input vectors that map to the same function value, and therefore have sensitivity 0. Thus, among all input vectors that maximum sensitivity is 3. Thus, this function has sensitivity 3.

This is only one case among $3^{15} = 14,348,907$. By rotational symmetry, we can specify other functions whose sensitivity is 3. Specifically, the *small* triangle, **0000**, **1000**, and **2000** in the case of the example above, can occur in three ways. Its function value can be chosen in three ways. And, the remaining 12 function values can be chosen in two ways. Thus, there are a total of $3 \times 3 \times 2 = 18$ ways the ensemble can be chosen.

Because triangles of the type described above do not overlap, it is tempting to believe that two or three triangles similar to **0000**, **1000**, and **2000** can occur in the same function. For example, suppose a small triangle such as **1111**, **1110**, and **2111** exists, in addition to **0000**, **1000**, and **2000**. Then, **1100** is adjacent to two nodes, **1000** and **1110**, that map to function values that are different from the function value to which **1110** maps. Indeed, the two red dots on edges from **1100** to **1000** and **1110** indicate that a total contribution of 4 to the sensitivity of **1100**. Thus, the sensitivity of such a function is 4. Fig. 5 shows the lattice representation of the neighborhood around **1100**. It can be seen that all four logic values of **1100** are sensitive. That is, changing any one of these values changes the function value. Thus, the sensitivity of this function is 4. Therefore, we cannot count such functions towards the tally of those that have sensitivity 3.

We have referred to the above assignment of function values to input vectors as a "small" triangle. We shall refer to a $3 \times 3 \times 3$ version as a "large" triangle. The third row of Table 3 shows this case. Here, overlap among large triangles precludes more than one large triangle per function. For example, in the case of large triangle **0000**, **1000**, **1100**, **2000**, **2100** and **2200**, input vectors **2220** and **1110** have sensitivity 3, while all others have a lower sensitivity. By similar arguments to those used in small triangles, the func-

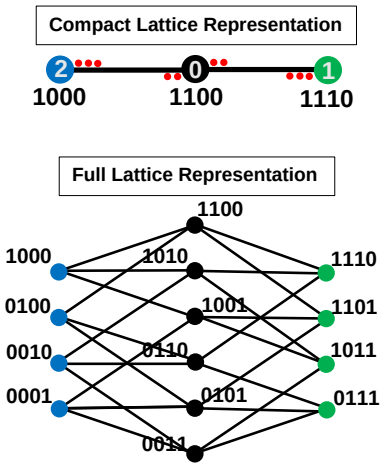


Figure 5. Comparison of Full to Compact Lattice Representation

tion's sensitivity is also 3, and there are 18 such triangles, just as in the case of small triangles.

The fourth row of Table 3 shows a "large" triangle with two special vertices indicated by a black dot enclosed by a blue circle, which corresponds to either a blue vertex on one end or a black vertex on the other. By an analysis similar to that applied to the third row, a function with such a triangle has sensitivity 3. There are 36 such triangles.

The fifth row has the same partial triangles as shown in the fourth row and represent still another way to form a triangle with sensitivity 3.

The last row in Table 3 consists of all remaining triangles, which is represented by a single input vector. Such a vector is surrounded by input vectors all of which map to a different function value.

5 Concluding Remarks

The study of sensitivity in Boolean logic functions has been restricted to researchers in complexity theory. We believe that sensitivity in Boolean and multiple-valued logic functions merits a closer examination by researchers in logic design. We show that a reconfigurable computer is useful in the study of sensitivity. We achieve a speed-up of 193 times faster than a single Xeon microprocessor. We show a cluster computer with 256 processors is also useful. Additionally, we examine the computation of sensitivity in multiple-valued symmetric functions, showing that a compact representation is useful in understanding sensitivity in such functions.

Acknowledgments

This research is partly supported by The Japan Society for the Promotion of Science (JSPS) Grant in Aid for Scientific Research. We thank three referees. Referee 1 provided many comments.

References

[1] K. Amano, "Enumeration of Boolean functions of sensitivity three and inheritance of nondegeneracy," *Proc. of IEEE International Symposium on Information Theory, 2017*, pp. 251–255. June 2017, Aachen, Germany.

[2] J. T. Butler, "Bent function discovery by reconfigurable computer," *Proceedings of the 9th International Workshop on Boolean Problems*, Freiberg, Germany, Sept. 16–17, 2010, pp. 1–12.

[3] S. Chakraborty, "Sensitivity, block sensitivity and certificate complexity functions," Masters Thesis, Univ. of Chicago, Feb. 2005, https://people.cs.uchicago.edu/~sourav/papers/mastersthesis.pdf.

[4] S. Cook, "The complexity of theorem proving procedures," *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, 1971.

[5] S. Cook, C. Dwork and R. Reischuk, "Upper and lower time bounds for parallel random access machines without simultaneous writes," *SIAM J.Comput.* 15 (1986), no. 1, 87–97.

[6] H. Huang, "Induced subgraphs of hypercubes and a proof of the sensitivity conjecture," *Ann. of Math. (2)*, 190(3):949-–955, 2019.

[7] N. Nisan, "CREW PRAMs and decision trees," *SIAM J. Comput.* 20 (1991), no. 6, 999–1070.

[8] D. Ramsey, "The sensitivity conjecture and the complexity of binary functions," *http://www.math.uchicago.edu/~may/VIGRE/VIGRE 2011/REUPapers/RamseyD.pdf*, Aug. 6, 2011.

[9] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999, p. 107.

[10] M. Sipser, "The history and status of the P versus NP question," *24th Annual ACM STC*, Victoria, BC, Canada, 1992, pp. 603–618.

[11] G. Turán, "The critical complexity of graph properties," *Inform. Process. Lett.*, Vol. 18, pp. 151–153, 1984.

[12] I. Wegener, "The critical complexity of all (monotone) Boolean functions and monotone graph properties," *Information and Control*, 67 (1985), pp. 212–222.