Original Research Paper

On a Proof of Inequality of PvsNP

Angelo Raffaele Meo

Department of Computer Science, Politecnico di Totino, Italy

Email: angelo.meo@polito.it

Article history
Received: 23-06-2022
Revised: 03-09-2022
Accepted: 26-09-2022

Abstract: This study is a new version of a previous paper. Its purpose is to simplify some sections of the old version and, above all, to present the proofs of some theorems which had been omitted for the sake of brevity. The analysis discussed in this study and its previous version is based on a well-known NP-complete problem which is called the "satisfiability problem" or "SAT". From SAT a new NP-complete problem, called "core function", derives; this problem is described by a Boolean function of the number of the clauses of SAT. In this study, a new proof is presented according to which the number of gates of the minimal implementation of core function increases with *n* exponentially. Since the synthesis of the core function is an NP-complete problem, this result can be considered as the proof of the theorem which states that the class *P* of all the decision problems which can be solved in polynomial time does not coincide with the class *NP* of the problems for which an answer can be verified in polynomial time.

Keywords: P-NP Question, Complexity, Boolean Functions, Satisfiability, Polynomial or Exponential Increase, Core Function

Introduction

A paper devoted to the proof of the theorem according to which P and NP do not coincide was presented to the Journal of Computer Science on September 2020 and published (Meo, 2021). According to the Journal of Computer Science at the end of August 2022 more than 2200 readers had viewed that paper and more than 600 readers had downloaded it.

Some readers have asked some questions concerning a few theorems whose proofs had been omitted in that paper for the sake of brevity. To prove these theorems is the main purpose of this new version of that paper.

The proof of inequality on the question PvsNP which had been presented in the previous paper and which will be completed in this study is based on the following steps:

1. A new Boolean function called "core function" is derived from the well-known SAT function. The core function is equivalent to SAT according to the known definition of NP-completeness
2. The main properties of the core function are presented and discussed
3. It is shown that the number of gates necessary to implement core function increases exponentially with the size of the problem

At present, no reader of my papers has found any mistake in the three steps of that proof. Future work might concern some mistakes which will be discovered. For example, if it will be proved that core function is not NP-complete, another function will be presented and discussed.

The second line of future research might concern the direct synthesis of SAT function or some other function equivalent to SAT.

Definitions

A brief description of the definitions and properties well-known among the scientists of modern computational complexity theory is presented in this section.

P denotes the class of all the decision problems which can be solved in polynomial time.

NP denotes the class of all the decision problems *f* satisfying the property that the function check (*f*) analyzing a witness of the decision problem is polynomial time decidable.

"*P* = *NP*?", or, in other terms, "Is *P* a proper subset of *NP*?", is one of the most important open questions in modern computational complexity theory.

A decision problem *C* in *NP* is NP-complete if it is in *NP* and if every other problem *L* in *NP* is reducible to it, in the sense that there is a polynomial time algorithm that transforms instances of *L* into instances of *C* producing the same output values.

The importance of NP-completeness derives from the fact that, if we find a polynomial time algorithm for just one NP-complete problem, then we can construct polynomial time algorithms for all the problems in NP and, conversely, if any single NP-complete problem does not have a polynomial time algorithm than no NP-complete problem has a polynomial time solution.

The analysis discussed in this study will be based on the following well-known NP-complete problem which is called the "satisfiability problem or SAT".

Given a Boolean expression containing only the names of variables (some of which may be complemented), the operators AND, OR, and NOT and parentheses, is there an assignment of TRUE or FALSE values to the variables which makes the entire expression TRUE.?

It is well known that the problem remains NP-complete also when all the expressions are written in "conjunctive normal form" with 3 variables per clause (problem 3SAT). In this case, the analyzed expressions will be of the type:

$$\begin{aligned} F = &\left(x_{11} OR\, x_{12} OR\, x_{13}\right) and \\ &\left(x_{21} OR\, x_{22} OR\, x_{23}\right) and \\ &\ldots\ldots\ldots\ldots \\ &\left(x_{t1} OR\, x_{t2} OR\, x_{t3}\right) \end{aligned} \tag{1}$$

where:

t	=	The number of clauses or triplets
each x_{ij}	=	A variable in complemented or uncomplemented form

Each variable may appear multiple times in that expression.

Usually, the deterministic Turin machine is assumed as the computational model. In this study, the analysis will be developed concerning a family $\{C_n\}$ of Boolean circuits, where C_n has n binary inputs and it produces the same binary output as the corresponding Turing machine.

The equivalence between a deterministic Turing machine M processing some input x belonging to $\{0,1\}^n$ and an n-input Boolean circuit C_n is well known. It is also known that the number of gates, or AND, OR, NOT operators, appearing in circuit C_n, is polynomial in the running time of the corresponding Turing machine.

The synthesis of the state of the art of question PvsNP can be found in (Fortnow, 2009; Cook, 1997; Mulmuley and Sohoni, 2001).

Materials and Methods

The Core Function

The Boolean circuit implementing the function described by Eq. (1) will be called C_t or C_n. Indeed, the number t of triplets appearing in Eq. (1) plays the role of symbol n used in the standard complexity theory. In the following analysis, we shall use the symbol t when it is necessary to remember the number of triplets and n in the other cases.

To simplify the analysis, circuit C_n will be decomposed into two processing layers called the "compatibility layer" and "core layer".

Compatibility Layer

A variable j of triplet i will be defined as "compatible" with variable k of triplet h when and only when, either:

- The sign s_{ij} of the former variable is equal to the sign s_{hk} of the latter variable, or
- The name $<n_{ij1}\ n_{ij2}\ \ldots\ n_{ijm}>$ of the former variable is different from the name $<n_{hk1}\ n_{hk2}\ \ldots n_{hkm}>$ of the latter variable

From that definition it follows that two "not compatible" variables have different signs and the same name; therefore, their AND is identically FALSE.

The compatibility layer is composed of 3·t·(3·t-3)/2 identical cells, one for each pair of variables belonging to different triplets.

The inputs of a cell will be the sign s_{ij} and the binary code $<n_{ij1}\ n_{ij2}\ \ldots n_{ijm}>$ of variable j of triplet i and the sign s_{hk} and the binary code $<n_{hk1}\ n_{hk2}\ \ldots n_{hkm}>$ of variable k of triplet h. The output of the same cell $c(i, j; h, k)$ will be TRUE when, and only when, the two variables are compatible between themselves.

Therefore, the function implemented by a cell may be written as follows (by using the symbols ∗, + and! for representing AND, OR, and NOT operators, respectively):

$$\begin{aligned} c(i,j;h,k) = &\ s_{ij} * s_{hk} +\ !s_{ij} * !s_{hk} + \\ &+ n_{ij1} * !n_{hk1} + !n_{ij1} * n_{hk1} + \\ &+ n_{ij2} * !n_{hk2} + !n_{ij2} * n_{hk2} + \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ &+ n_{ijm} * !n_{hkm} + !n_{ijm} * n_{hkm} \end{aligned} \tag{2}$$

Variable $c(i, j; h, k)$ will be called a "compatibility variable" or simply a "compatibility".

Core Layer

The Boolean function implemented by the core layer will be called the "core function" of order t, where t is the number of triplets. It will be denoted with the symbol $CF(t)$ or $CF(n)$. The core layer processes only the 9·t·(t-1)/2 compatibility variables $c(i, j; h, k)$ and produce the global result of the computation. The core function can be determined by proceeding as follows.

Consider one selection of variables appearing in Eq. (1), one and only one for each triplet, for all the triplets:

$$<1i_1>, <2i_2>, \ldots, <ti_t> \tag{3}$$

with, i_1, i_2, …., $i_t \in \{1, 2, 3\}$ be the indexes <number of triplets, number of variables in the triplet> of the selected variables. They will be called "characteristic indexes". Let

Π^k be the product of all the compatibility variables relative to the *k-th* of selections (3):

$$\Pi^k = c(1,i_1;2,i_2) * c(1,i_1;3,i_3) * ...... * c(t-1,i_{t-1};t,i_t) \qquad (4)$$

The core function can be defined as the sum:

$$\sum_k \prod^k \qquad (5)$$

of the products (4) relative to all the selections (3).

For example, in the case of *CF* (3), the core function can be defined as follows:

$$\begin{aligned} CF(3) &= c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1) \\ &+ c(1,1;2,1) * c(1,1;3,2) * c(2,1;3,2) \\ &+ c(1,1;2,1) * c(1,1;3,3) * c(2,1;3,3) \\ &+ c(1,1;2,2) * c(1,1;3,1) * c(2,2;3,1) \\ &+ ...(other 22 products)... \\ &+ c(1,3;2,3) * c(1,3;3,3) * c(2,3;3,3) \end{aligned} \qquad (6)$$

It is easy to prove that there is an assignment of values TRUE or FALSE to variables appearing in Eq. (1) which make the value of (1) equal to TRUE when and only when, the core function takes the value TRUE.

Notice that the processing work of a cell increases as a polynomial function $P(t)$ of the number of the variables since the increment of the length of the code of the name is logarithmic. Therefore, the total processing work of the compatibility layer increases as $9 \cdot t \cdot (t-1) \cdot P(t)$ where $9 \cdot t \cdot (t-1)/2$ is the total number of the compatibility cells.

Besides, the problem solved by the core layer is clearly in *NP*, because it is easy to verify a witness solution. It follows that, since the compatibility layer polynomially reduces an NP-complete problem (3SAT) to the problem solved by the core layer, the core function describes a new NP-complete problem.

Some properties of core function have been discussed in (Meo, 2008).

A Theorem of Boolean Monotonic Functions

Let $f(x_1, x_2, ..., x_h)$ be an isotonic Boolean function, that is a Boolean function that can be implemented with only AND and OR gates, applied to uncomplemented literals $x_1, x_2, \ldots, x_h$. It was believed that the minimum cost implementation of $f(x_1, x_2, \ldots, x_h)$ always contains only OR and AND gates, but A. Razborov proved that there are isotonic functions whose minimum cost implementation contains also NOT gates (Razborov, 1985).

However, there is an upper bound on the comparison of the costs of the minimum cost implementations with and without NOT gates. It is specified by the following theorem.

Theorem 4.1

Let I_{min} be one of the minimum cost implementations of the isotonic Boolean function $f(x_1, x_2,...,x_h)$, the cost being defined as the total number of AND, OR, or NOT gates. Let C_{min} be the cost of I_{min}.

There exists always an implementation *J* of *f* containing only and or gates (in addition, if necessary, to the NOT operators producing input variables! x_1,! x_2, ...,! x_h) such that:

$$cost(J) <= 2 \cdot C_{min} + h$$

where, *h* is the number of variables.

The proof of this theorem can be found in Chapter 4 of (Meo, 2021).

This theorem will be used to simplify the analysis of core function circuits.

Properties of Core Function

It is easy to prove the following properties of the core function.

Property 1

A function defined by Eq. (5) isotone.

Property 2

Any product defined by Eq. (4) is a prime implicant of core function (that is, a Product of Compatibilities ("PoC") which implies core function and no other term of it).

Property 3

Since the different selections of each of the variables defined by Eq. (3) are 3, the number of prime implicants of the core function is equal to 3^t. Each of these prime implicants is essential (that is, it does not imply a sum of other prime implicants) and it is the product of $t \cdot (t-1)/2$ compatibilities.

Products of Compatibilities

In the next sections, reference will be made to the following definitions.

Definition of Spurious Compatibilities Pair

A pair of compatibility variables $\{c(h,k;l,m), c(p,q;r,s)\}$ is defined as a spurious pair if:

$$\begin{aligned} &(h = p \; and \; k \neq q) \\ or &(h = r \; and \; k \neq s) \\ or &(1 = p \; and \; m \neq q) \\ or &(1 = r \; and \; m \neq s) \end{aligned}$$

For example, the pair $\{c(1,1;2,1), c(1,2;3,1)\}$ is spurious since the triplet 1 is associated with two different indexes of variables (1 and 2).

Definition of Spurious Products of Compatibilities

A spurious Product of Compatibilities (spurious PoC) is a product of compatibility variables containing the elements of one or more than one spurious pair.

For example, the PoC:

$$c(1,1;2,1)*c(1,2;3,1)*c(2,1;3,1)$$

is a spurious PoC since it contains the elements of the spurious pair:

$$\{c(1,1;2,1),c(1,2;3,1)\}$$

Definition of Impure Products of Compatibilities

A PoC containing one or more complemented variables will be defined as an impure PoC. In particular, a term *T* of *CF* (that is, a PoC implying *CF*) which contains one or more complemented variables, will be defined as an impure term of *CF*. A product of compatibilities that is neither spurious nor impure will be defined as a pure product of compatibilities.

Definition of Mark

Consider a pure product of compatibilities satisfying the property that all the indexes of triplet {1,2,…,*t*} appear at least once in some variable. The product of the variables of such a subset will be defined as a "mark" or "pure mark" of the prime implicant of which it contains a subset of compatibilities.

For example, in the case of *CF* (4), the PoC:

$$M=c(1,a;2,b)*c(1,a;3,c)*c(1,a;4,d) \quad (7)$$

where the indexes of the triplet are elements of the set {1,2,3,4} and *a*, *b*, *c*, *d* are elements of {1,2,3} is a mark of the prime implicant:

$$\begin{aligned}P=c(1,a;2,b)*c(1,a;3,c)*c(1,a;4,d)\\ *c(2,b;3,c)*c(2,b;4,d)*c(3,c;4,d)\end{aligned} \quad (8)$$

since all the indexes of triplet appear at least once in Eq. (7).

Definition of Spurious Mark

A spurious PoC in which all the indexes of triplet appear at least once will be called a “spurious mark”. Notice that a spurious mark may be the mark of more than one prime implicant. For example, in the case of *CF* (3):

$$c(1,1;2,1)*c(1,1;3,1)*c(1,1;2,2)$$

is a spurious mark of both the prime implicants:

$$c(1,1;2,1)*c(1,1;3,1)*c(2,1;3,1)$$

and:

$$c(1,1;2,2)*c(1,1;3,1)*c(2,2;3,1)$$

An impure PoC containing a (possibly spurious) mark will be defined as a (possibly spurious) impure mark.

Definition of Extended Prime Implicant

A term *T* of core function, that is, an implicant of core function (a product of literals implying core function), contains all the uncomplemented literals of a prime implicant. Therefore, it may be defined as an “extended prime implicant” (only) to remember that it contains all the compatibilities of a prime implicant.

It may be a spurious extended prime implicant or an impure extended prime implicant or both a spurious and impure extended prime implicant.

Notice that an extended prime implicant can be viewed as a (possibly spurious or impure) mark.

Definition of Remainder

A PoC which is neither a (possibly spurious or impure) mark nor an (extended) prime implicant will be called a remainder". Also, a remainder may be pure (if for any triplet index there is only one index of variable in that triplet) or spurious or impure.

A pure remainder *R* may be implied by more than one prime implicant. For example, in the case of *CF* (3), *R* = *c*(2,1;3,1) is a remainder which is implied by the following prime implicants:

$$\begin{aligned}P1&=c(1,1;2,1)*c(1,1;3,1)*c(2,1;3,1)\\ P2&=c(1,2;2,1)*c(1,2;3,1)*c(2,1;3,1)\\ P3&=c(1,3;2,1)*c(1,3;3,1)*c(2,1;3,1)\end{aligned} \quad (9)$$

On the definitions of mark and remainder, the following property is based.

Property 4

Let P_1 and P_2 be two PoCs such that P_1*P_2 is equal to a prime implicant *P* of the core function. Either P_1 or P_2 is a mark of *P*.

The External Core Function

Let I_j be a prime implicant of *CF*(*n*). The external core function relative to I_j, *ECF*(*n*, I_j), is defined as the sum of all the minterms of *CF*(*n*) which imply I_j and no other prime implicant I_k of *CF*(*n*) with $k \neq j$. (Remember that a minterm of a Boolean function *F* is a product of all the variables of *F*, some complimented and some others uncomplemented, implying *F*).

Of course:

$$ECF(n,I_j)=I_j*P_{k\neq j}(!I_k) \quad (10)$$

where all the prime implicants of core function are involved and! I_k denotes the complement of I_k (i.e., NOT I_k).

The global external core function of order n, or *ECF*(*n*), will be defined as the sum of *ECF*(*n*, I_j)'s relative to all the prime implicants I_j of *CF*(*n*):

$$ECF\left(n\right)=\sum_{j}ECF\left(n,I_{j}\right) \tag{11}$$

The importance of external core function derives from the following theorems.

The proofs of these theorems can be found in (Meo, 2022; 2008), except Theorem 7.5 which has been presented in Appendix 1 of this study.

Theorem 7.1

Let *T* be a term (or extended prime implicant) of *CF*(*n*). It may be the product of all the compatibilities of a prime implicant I_j of *CF*(*n*) and other compatibilities, that is:

$$T=I_{j}*X$$

where *X* is a possibly empty PoC. *T* can also be written as *T* = *T*(I_j).

All the minterms of *T*(I_j) contained in *ECF*(*n*) are minterms of *ECF*(*n*, I_j).

Theorem 7.2

Let *T* be a term of *CF*(*n*) implying two or more than two prime implicants of *CF*(*n*):

$$T=T\left(I_{j},I_{k}\right)$$

The number of minterms of *T*(I_j, I_k) belonging to *ECF*(*n*) is equal to 0.

Theorem 7.3

Let *T* = *T*(I_j) = I_j**X* be a term of *CF*(*n*) which is spurious for a single not complemented compatibility *X*.

If *NMT*(*F*) denotes the number of minterms of Boolean function *F*, the number of minterms of I_j**X* contained in *ECF*(*n*, I_j) is:

$$NMT\left(I_{j}*X*ECF\left(n,I_{j}\right)\right)<=\left(1/2\right).NMT\left(ECE\left(n,I_{j}\right)\right) \tag{12}$$

However, for large values of *n*, as shown by the data of Appendix 1:

$$NMT\left(I_{j}*X*ECF\left(n,I_{j}\right)\right)\approx\left(1/2\right).NMT\left(ECE\left(n,I_{j}\right)\right)^{\sim}$$

By proceeding in the same way, it is possible to generalize the preceding Theorem 7.3 as follows.

Theorem 7.4

Let:

$$I_{j}*X_{1}*X_{2}*\ldots X_{m}$$

be a spurious term characterized by *m* spurious not complemented compatibilities.

The number of its minterms contained in *ECF*(*n*, I_j) is:

$$\begin{aligned}&NMT\left(I_{j}*X_{1}*X_{2}*,\ldots*X_{m}*ECF\left(n,I_{j}\right)\right)\\&<=\left(1/\left(2^{m}\right)\right).NMT\left(ECE\left(n,I_{j}\right)\right)\end{aligned}$$

However, for large values of *n*, as shown by the data of Appendix 1:

$$\begin{aligned}&NMT\left(I_{j}*X_{1}*X_{2}*,\ldots*X_{m}*ECF\left(n,I_{j}\right)\right)\\&\approx\left(1/\left(2^{m}\right)\right).NMT\left(ECE\left(n,I_{j}\right)\right)\end{aligned} \tag{13}$$

Theorem 7.5

Let *T* = *T*(I_j) be an impure term of *CF*(*n*) characterized by a single impure variable (!*X*):

$$T=I_{j}*\left(!X\right)$$

For large values of n, the number of minterms of *ECF*(*n*, I_j) contained in *T* is:

$$NMT\left(I_{j}*\left(!X\right)*ECF\left(n,I_{j}\right)\right)\approx\left(1/2\right).NMT\left(ECE\left(n,I_{j}\right)\right) \tag{14}$$

The proof of this theorem can be found in Appendix 1.

Theorem 7.6

Let *T* = *T*(I_j) be an impure term of *CF*(*n*) characterized by *m* impure variables:

$$T=I_{j}*\left(!X_{1}\right)*\left(!X_{2}\right)*\ldots\left(!X_{m}\right)$$

For large values of *n*, the number of minterms of *ECF*(*n*,I_j) contained in *T* is:

$$NMT\left(T*ECF\left(n,I_{j}\right)\right)\approx\left(\left(1/2\right)^{m}\right)\cdot NMT\left(ECF\left(n,I_{j}\right)\right) \tag{15}$$

This theorem is an obvious extension of Theorem 7.5.

Notice that *NMT* (*ECF* (*n*, I_j)) = *NMT* (*ECF* (*n*, I_k)) for any *j* and *k*. It will be called *NMT*1(*n*).

Results and Discussion

The Value of a Node

Let *U* be a node of the network implementing core function and let *F*(*U*) be the Boolean function of compatibilities *c*(*i*, *j*, *h*, *k*) implemented by *U*. Since the subnetwork having U as its input does not contain any NOT gate, we can write:

$$CF = F(U) * x_1 + F(U) * x_2 + \ldots + y_1 + y_2 + \ldots \quad (16)$$

where, x_1, x_2, …, y_1, y_2, …, are products of variables of core function, that is, products of compatibilities. Notice also that every $F(U)*x_i$ and every y_j must be an extended prime implicant of the core function. As we shall see in some examples, generally a single product of compatibilities is sufficient to implement the core function according to the following equation:

$$CF = F(U) * x + y_1 + y_2 + \ldots \quad (17)$$

where, x is single compatibility.
x_1, x_2, …, y_1, y_2, …, or x will be called "completion code".

More than one solution of Eq. (16) and (17) can produce the value of the core function. However, we are looking for a solution characterized by the following property: The total number of minterms of the external core functions $ECF(n, I_j)$ of the prime implicants produced by $F(U) * x_1 + F(U) * x_2 + \ldots$ s or by $F(U) * x$ takes the maximum value. By definition, this maximum value will be considered as the value $val(U)$ of the node U or the value $val(F(U))$ of the Boolean function implemented by U, on the condition that no x_i (or x) is a mark, since, otherwise, the contribution of the subnetwork having U as its input might be considered as more important than the contribution of U.

The values of x_1, x_2, …, or x which appear in the best solution of Eq. (16) and (17) will be called "optimal completion code".

It is easy to prove that the value of a pure remainder and the value of a Boolean function which can be described as a sum of remainders are always equal to 0. On the contrary, the value of a pure mark can be considered equal to $NMT1(n)$ while the value of an impure mark can be considered equal to $NMT1(n)\cdot 2^{-m}$, where m is the number of spurious or complemented compatibilities. Besides, the value of a Boolean function which is equal to a sum of marks is always less than or equal to the sum of the values of the considered marks.

For example, as we shall discuss in the following sum of remainders of CF (4):

$$\begin{aligned} &c(1,1;2,1)*c(1,1;3,1)*c(2,1;3,1)+c(1,2;2,1)\\ &*c(1,2;3,1)*c(2,1;3,1)+c(1,3;2,1)*c(1,3;3,1)*c(2,1;3,1)\end{aligned}$$

has a value equal to 0, while the following sum of marks:

$$\begin{aligned} &c(1,1;2,1)*c(1,1;3,1)*c(1,1;4,1)+c(1,2;2,1)\\ &*c(1,2;3,1)*c(1,2;4,1)+c(1,3;2,1)*c(1,3;3,1)*c(1,3;4,1)\end{aligned}$$

has a value equal to $3 \cdot NMT1(4)$.

The Value of an OR Gate

An OR gate characterized by n inputs can be implemented as a sum of two inputs OR gates. Therefore, we can restrict our attention to two inputs OR gates.

The value of an OR gate having node A and node B as its inputs and node U as its output can be defined as:

$$val(A\mathrm{OR}B) = val(U) - \left(val(A) + val(B)\right)$$

On the statements discussed it is easy to prove the following simple rules for evaluating the values of functions $F(U)$, $F(A)$, and $F(B)$, under the hypothesis that these three functions are written as the sums of their prime implicants:

1. The value of a remainder is equal to 0
2. The value of a sum of remainders is equal to 0
3. The value of a pure mark is equal to $NMT1(n)$. The value of an impure mark M is $NMT1(n)\cdot(1/2^m)$ where, m is the number of spurious or impure compatibilities contained in M
4. The value of a sum of marks is always equal to or less than the sum of the values of the prime implicants of core function which imply those marks
5. Notice that, theoretically, a mark might derive from the Boolean sum of two or more than two remainders. For example, the mark of CF (4) $m = c(1,1;4,1) * c(2,1;4,1) * c(3,1;4,1)$ might derive from the sum of the two remainders $r_1 = c(1,1;4,1) * c(2,1;4,1) * !c(1,1;3,2)$ and $r_2 = c(3,1;4,1) * c(1,1;3,2)$. Let remainders r_1 and r_2 be two of the inputs of the OR gate producing mark m and let U be the output of this OR gate. Since the circuit-producing CF does not contain NOT circuits, the value of the circuit-producing CF can be written as follows:

$$\begin{aligned} CF &= U*x_1 + U*x_2 + \ldots + y_1 + y_2 + \ldots\\ &= r_1*x_1 + r_2*x_1 + r_1*x_2 + r_2*x_2 + \ldots + y_1 + y_2 + \ldots\end{aligned}$$

Since r_1 and i_2 are remainders, every x_i must be a mark. Besides, either there is a y_k equal to the prime implicant $I(m)$ deriving from mark m or one of the products $r_i * x_j$ is equal to $I(m)$ and, therefore, the mark of $I(m)$ is produced outside the considered OR operation. It follows that the production of a mark as the sum of two remainders cannot be used to generate new prime implicants.

From these rules, it is easy to prove that $val(U)$ is never larger than $val(A) + val(B)$ and, therefore, the value of an OR gate can always be considered equal to 0.

The Value of an AND GATE the Most Powerful and Gate

As in the case of OR gates, an n inputs AND gate can be implemented as the product of two inputs AND gates. Therefore, we can restrict our attention to two inputs AND gates.

The value of an AND gate having *A* and *B* as its inputs and *U* as its output can be defined as:

$$val(A\ AND\ B) = val(U) - (val(A) + val(B))$$

Since we are interested in identifying the most powerful AND gate, we shall assume that both *F*(*A*) and *F*(*B*) are sums of remainders so that both *val*(*A*) and (*B*) are equal to 0. Therefore, the value of the considered gate will be always equal to the value of output *U*.

The most powerful AND gate can be identified by proceeding as follows:

1. Let $A = (a_1 + a_2 + a_3 + \dots)$ and $B = (b_1 + b_2 + b_3 + \dots)$, where all the a_i and b_j are remainders. A product $a_i * b_j$ can produce more than one mark, but a product $a_i * b_j * x$ cannot produce more than one prime implicant because the product of two prime implicants has a value equal to 0. To produce a pure prime implicant, the product $a_i * b_j$ must produce a pure mark
2. If a_1 is a remainder, at least one of the t indexes of triplet does not appear in the list of triplet indexes of a_1 because, otherwise, it $a_1$would be a mark. Let it be *i*'
   For the same reason, at least another triplet index does not appear in the list of triplet indexes of b_1. Let it be *j*'
   By example:

$$\begin{aligned} &a_1 = c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1) \\ &b_1 = c(1,1;4,1) * c(2,1;4,1) \\ &m_1 = a_1 * b_1 \end{aligned} \tag{18}$$

   Triplet index 4 is missing in a_1; triplet index 3 is missing in b_1. So that $a_1 * b_1 * x$ is a prime implicant of *CF* (4), *x* must be equal to *c*(3,1;4,1)
3. Eq. (18) is the example of two remainders whose product is a mark without spurious or impure variables. The value of that mark is *NMT*1(4)
   According to Theorems 7.3, 7.4, 7.5, 7.6, a mark containing a spurious or impure compatibility has a value equal to (1/2)·*NMT*1(*n*) while a mark containing *m* spurious or ·impure compatibilities has a value equal to $(1/2^m)\cdot NMT1(n)$
4. Assume that $a_2 * b_1$ is equal to a new mark as:

$$m_2 = c(1,2;2,1) * c(1,2;3,1) * c(2,1;4,1) \tag{19}$$

   We can start by assuming $a_2 = c(1,2;2,1) * c(1,2;3,1) * c(2,1;3,1)$
   Since the optimal completion code x must be equal to *c*(3,1;4,1) and a_2 cannot contain all the three compatibilities involving <1,2>, the value of b_1 must be corrected by adding *c*(1,2;4,1) to b_1:
   $b_1' = b_1 * c(1,2;4,1)$

   Therefore:

$$\begin{aligned} val(a_1 * b_1) &= (1/2) \cdot NMT1\ (4) \\ val(a_2 * b_1) &= (1/2) \cdot NMT1\ (4) \end{aligned}$$

   No increment of the total value has been obtained by introducing a new mark
5. To implement the new mark m_2 without reducing the value of m_1 it is necessary to introduce a new remainder $b_2 = c(1,2;4,1) * c(2,1;4,1)$ so that $m_2 = a_2 * b_2$

   However, the products $a_1 * b_2$ and $a_2 * b_1$ are not marks. Therefore, it is necessary to introduce a correction like the following one:

$$\begin{aligned} &a_1 = c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1) \\ &b_1 = c(1,1;4,1) * c(2,1;4,1) * c(1,2;4,1) \\ &a_2 = c(1,2;2,1) * c(1,2;3,1) * c(2,1;3,1) \\ &b_1 = c(1,2;4,1) * c(2,1;4,1) * c(1,1;4,1) \end{aligned}$$

   which is characterized by a total value equal to (1/2 + 1/2) * *NMT*1(4)

   It is easy to prove that the best solution is the following:

$$\begin{aligned} &a_1 = c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1) \\ &b_1 = c(1,1;4,1) * c(2,1;4,1) * c(1,1;2,1) \\ &a_2 = c(1,2;2,1) * c(1,2;3,1) * c(2,1,3,1) * !c(1,1;2,1) \quad (20) \\ &b_2 = c(1,2;4,1) * c(2,1;4,1) * c(1,2;2,1) * !c(1,1;2,1) \end{aligned}$$

   where:
   val ($m_1 = a_1 * b_1$) = *NMT*1(4)
   val ($m_2 = a_2 * b_2$) = *NMT*1(4) · (1/2)

6. The two pairs of remainders appearing in $(a_1 + a_2) * (b_1 + b_2)$ can produce four different marks. Appendix 2 shows the best implementation. Its total value is (1/2) * *NMT*1(4), but the value of the four marks decreases very quickly with *n*. Therefore, there is no point in continuing this line

7. By following the same line of reasoning which has made it possible to prove that Eq. (20) is the best solution for implementing two marks, it is easy to prove that the best solution for implementing three marks is the following one:

$$\begin{aligned}
a_1 &= c(1,1;2,1)*c(1,1;3,1)*c(2,1;3,1)\\
b_1 &= c(1,1;4,1)*c(2,1;4,1)*c(1,1;2,1)\\
a_2 &= c(1,2;2,1)*c(1,2;3,1)*c(2,1,3,1)*!c(1,1;2,1)\\
b_2 &= c(1,2;4,1)*c(2,1;4,1)*c(1,2;2,1)*!c(1,1;2,1)\\
a_3 &= c(1,3;2,1)*c(1,3;3,1)*c(2,1;3,1)*!c(1,1;2,1)*!c(1,2;2,1)\\
b_3 &= c(1,3;4,1)*c(2,1;4,1)*c(1,3;2,1)*!c(1,1;2,1)*!c(1,2;2,1)
\end{aligned} \tag{21}$$

The value of this solution is:

$$(1+(1/2)+(1/4))\cdot NMT1(4)$$

8. Appendix 3 shows the best solution for implementing the marks of all the nine prime implicants of *CF* (4) compatible with the conditions that the variables <3,2>, <3,3>, <4,2>, <4,3> do not appear in that product and the completion code *x* takes the value *c*(3,1;4,1)
   The value of the gate implementing those imarks is:

$$(1+(1/2)+(1/4))\cdot(1+(1/4)+(1/16))\cdot NMT1(4) \tag{22}$$

   which is slightly less than the:

$$(1+(1/2)\ +(1/4))^2\cdot NMT1(4) \tag{23}$$

   Equation (22) and (23) can be generalized according to the following equations which show the value of the best gate implementing the marks of $3^{(n\text{-}2)}$ prime implicants:

$$(1+(1/2)+(1/4))^{n\text{-}3}\cdot(1+(1/4)+(1/16))\cdot NMT(n) \tag{24}$$

   which is slightly less than:

$$(1+(1/2)+(1/4))^{n.2}\cdot NMT1(n) \tag{25}$$

   To prove that the solution proposed in Appendix 3 is the best consider three marks that are different for the value of one and only one triplet index. For example, the three marks $m_7 = a_7 * b_7$, $m_8 = a_8 * b_8$, and $m_9 = a_9 * b_9$, which have been defined in Appendix 3, are different only for the values in triplet index 1
   In order that $a_7 * b_8 = 0$ and $a_8 * b_7 = 0$, both a_8 and b_8 must contain compatibility !*c*(1,1;2,3). Therefore, the value of mark m_8 will be multiplied by 1 / 2o
   In order that $a_7 * b_9 = a_9 * b_7 = a_8 * b_9 = a_9 * b_8 = 0$, both a_9 and b_9 must contain !*c*(1,1;2,3) *!*c*(1,2;2,3)
   No other solution makes it possible to reduce the values of m_8 and m_9 by a smaller value
   It is easy to verify on the data of Appendix 3 that all the triplets $\{m_i, m_j, m_k\}$ satisfying that property has received the same type of corrections and only those corrections have been applied
   Therefore, we can state that the solution proposed in this study leads to the best solution and that the maximum value of an AND gate of the type above specified is slightly less than $(1 + (1/2) + (1/4))^{n\text{-}2} \cdot NMT1(n)$

9. So far all the new marks contained only <3,1> and <4,1> in the compatibilities involving variables of triplet 3 or 4. This condition can be removed to try to increase the value of the considered AND gate
   For example, as shown in Appendix 3, we can add nine new remainders $a_{10} \ldots a_{18}$ to $a_1 \ldots a_9$ and $b_{10} \ldots b_{18}$ to $b_1 \ldots b_9$, where the new remainders are obtained by replacing all the appearances of <4,1> with <4,2>. Thus nine new marks and nine new prime implicants will be generated but the value of the considered gate will not be doubled. Indeed, the optimal completion code *x*, which was *c*(3,1;4,1> becomes *c*(3,1;4,1) * *c*(3,1;4,2) and the value of all the marks will be multiplied by (1/2)

10. The lists $(a_1 + a_2 + \ldots + a_9)$ and $(b_1 + b_2 + \ldots + b_{9u})$ can be updated as follows:

$$\begin{aligned}
&(a_1+\ldots+a_{10}+\ldots+a_{19+}+\ldots+a_{28}+\ldots+a_{37}\\
&+\ \ldots+a_{46}+\ldots+a_{55}+\ldots+a_{64}+\ldots+a_{73}+\ldots)\\
&(b_1+\ldots+b_{10}+\ldots+b_{19}+\ldots+b_{28}+\ldots+b_{37}+\ldots\\
&+b_{46}+\ldots+b_{55}+\ldots+b_{64}+\ldots+b_{73}+\ldots)
\end{aligned}$$

   where also the completion code should be updated:

$$\begin{aligned}
x = {}& c(3,1;4,1)*c(3,1;4,2)*c(3,1;4,3)\\
&*c(3,2;4,1)*c(3,2;4,2)*c(3,2;4,3)*\\
&c(3,3;4,1)*c(3,3;4,2)*c(3,3;4,3)
\end{aligned}$$

   Thus, all 81 prime implicants of *CF* (4) will be generated, but their total value will increase very slowly
   It is very hard to identify the most powerful AND gate in the implementation of *CF*(*n*). However, since the number of elementary products $(a_1 + a_2 + \ldots) * (b_1 + b_2 + \ldots)$ is 9 and it has been proved that each of these products has the value shown by Eq. (24) and (25), it is obvious that the value of the most powerful AND gate is smaller than:

$$valmax(n) = \ 9\cdot(1+(1/2)+(1/4))^{n-2}\cdot NMT1(n) \tag{26}$$

Conclusion

Since the number of minterms of *ECF*(*n*) contained in *CF*(*n*) is equal to $3^{n\cdot}$ *NMT*1(*n*) and the value of a gate, that is the number of new minterms produced by a gate, is less than:

$$valmax(n) = 9\cdot(1+(1/2)+(1/4))^{n-2}\cdot NMT1(n)$$

the number of gates necessary to implement *CF*(*n*) is larger than $3^n/(9\cdot((1+1/2+1/4)^{(n\text{-}2)}))$ and, therefore, it increases exponentially with *n*.

Since the synthesis of core function *CF*(*n*) is an NP-complete problem, this result is equivalent to proving that *P* and *NP* do not coincide.

Acknowledgment

The Academy of Science of Turin who published a preliminary version of this study.

Funding Information

The authors have not received any financial support or funding to report.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Cook, S. (1997) “The P versus NP problem,” in J. Carlson, A. Jaffe, & A. Wiles (eds.), *The Millennium Prize Problem*, pp. 88–104, *Providence*: *American Mathematical Society*. https://www.cs.toronto.edu/~toni/Courses/Complexity2015/handouts/cook-clay.pdf

Fortnow, L. (2009). The status of the P versus NP problem. *Communications of the ACM, 52*(9), 78-86. https://dl.acm.org/doi/fullHtml/10.1145/1562164.1562186

Meo, A. R. (2021). “On the P versus NP question: A new proof of inequality”, *Journal of Computer Science*, 17/5, 511-524, 2021.

Meo, A. R. (2018). “On the P vs NP question: A proof of inequality”, *arXiv*:1802.05484

Meo, A. R. (2008). Some theorems concerning the core function. In *Concurrency, Graphs, and Models* (pp. 778-796). Springer, Berlin, Heidelberg. https://link.springer.com/chapter/10.1007/978-3-540-68679-8_48

Meo, A. R. (2022). On the P versus NP question.

Mulmuley, K. D., & Sohoni, M. (2001). Geometric complexity theory I: An approach to the P vs. NP and related problems. *SIAM Journal on Computing*, *31*(2), 496-526. https://doi.org/10.1137/S009753970038715X

Razborov, A. A. (1985). Lower bounds for the monotone complexity of some Boolean functions. In *Soviet Math. Dokl.* (Vol. 31, pp. 354-357).

Appendix 1

In order to prove Eq. (14) consider the following example relative to the external core function *ECF* (4,I_j) where, I_j is the prime implicant defined by the mark *c*(1,1;2,1) * *c*(3,1;4,1) and !*X* = !*c*(1,1;2,2).

ECF (4, I) =
c(1,1;2,1) * *c*(1,1;3,1) * *c*(1,1;4,1) * *c*(2,1;3,1) * *c*(2,1;4,1) * *c*(3,1;4,1)*
(!*c*(1,1;2,1) + !*c*(1,1;3,1) + !i(1,1;4,2) + !*c*(2,1;3,1) + !*c*(2,1;4,2) + !*c*(3,1;4,2))*
(!*c*(1,1;2,1) + !*c*(1,1;3,1) + !*c*(1,1;4,3) + !*c*(2,1;3,1) +!*c*(2,1;4,3) + !*c*(3,1;4,3))*
(!*c*(1,1;2,1) +!*c*(1,1;3,2) + !*c*(1,1;4,1) + !*c*(2,1;3,2) + !*c*(2,1;4,1) + !*c*(3,2;4,1))*
. . . .

(!*c*(1,1;2,2) +!*c*(1,1;3,1) + !*c*(1,1;4,1) + !i(2,2;3,1) + !*c*(2,2;4.1) + !*c*(3,1;4,1))*
(!*c*(1,1;2,2) + !*c*(1,1;3,1) + !*c*(1,1;4,2)+ !*c*(2,2;3,1) + !*c*(2,2;4,2) + !*c*(3,1;4,2))*
. . . .

(!*c*(1,1;2,3) + !*c*(1,1;3,1) + !*c*(1,1;4,1) + !*c*(2,3;3,1) + !*c*(2,3;4,1) + !*c*(3,1;4,1)*
(!*c*(1,1;2,3) + !*c*(1,1;3,1) + !*c*(1,1;4,2) + !*c*(2,3;3,1) + !*c*(2,3;4,2) + !*c*(3,1;4,2))*
. . . .

(!*c*(1,2;2,1) + !*c*(1,2;3,1) + !*c*(1,2;4,1) + !*c*(2,1;3,1) + !*c*(2,1;4,1) + !*c*(3,1;4,1))*
(!*c*(1,2;2,1) + !*c*(1,2;3,1) + !*c*(1,2;4,2) + !*c*(2,1;3,1) + !*c*(2,1;4,2) + !*c*(3,1;4,2))*
. . . .

(!*c*(1,2;2,2) +!*c*(1,2;3,1) + !*c*(1,2;4,1) + !*c*(2,2;3,1) + !*c*(2,2;4,1) + !*c*(3,1;4,1))*
. . . .

(!*c*(1,2;2,3) + !*c*(1,2;3,1) + !*c*(1,2;4,1) + !*c*(2,3;3,1) + !*c*(2,3;4,1) + !*c*(3,1;4,1))*
. . . .

(!*c*(1,3;2,1) + !*c*(1,3;3,1) + !*c*(1,3;4,1) + !*c*(2,1;3,1) + !*c*(2,1;4,1) + !*c*(3,1;4,1))*
. . . .

(!*c*(1,3;2,2) + !*c*(1,3;3,1) + !*c*(1,3;4,1) + !*c*(2,2;3,1) + !*c*(2,2;4,1) + !*c*(3,1;4,1))*
. . . .

(!*c*(1,3;2,3) + !*c*(1,3;3,1) + !*c*(1,3;4,1) + !*c*(2,3;3,1) + !*c*(2,3;4,1) + !*c*(3,1;4,1))*
. . . .

This table is characterized by 73 rows and 6 columns and produces 6·73 products of compatibility. A minority of these products contain compatibility! *c*(1,1;2,2) while the other terms-the majority-do not contain that compatibility.

The multiplication by !c(1,1;2,2) deletes half of the minterms contained in the majority of terms and leaves the other minterms unchanged.

Since the number of columns increases as n^2 is apparent that the ratio of the number of unchanged minterms divided by the total number of minterms decreases very quickly with the number n of variables.

Appendix 2

Consider the product ($a_1 + a_2$) * ($b_1 + b_2$) relative to *CF* (4) where:

a_1 = c(1,1;2,1)* c(1,1;3,1)* c(1,1;3,2)
a_2 = c(1,1;2,2)* c(1,1;3,2)* c(1,1;3,1)
b_1 = c(2,1;3,1)* c(2,1;4,1)* c(3,1;4,1)* c(2,2;3,1)* c(2,2;4,1)
b_2 = c(2,2;3,2)* c(2,2;4,1)* c(3,2;4,1)* c(2,1;3,2)* c(2,1;4,1)
with x = c[1,1] * c[4,1]

The following four marks of *CF* (4) are generated:

m_1 = a_1 * b_1 involving variables ([1,1], [2,1], [3,1], [4,1])
m_2 = a_1 * b_2 involving variables ([1,1], [2,1], [3,2], [4,1])
m_3 = a_2 * b_1 involving variables ([1,1], [2,2], [3,1], [4,1])
m_4 = a_2 * b_2 involving variables ([1,1], [2,2], [3,2], [4,1])

It is easy to verify that:

$$val(m_1) = val(m_2) = val(m_3) = val(m_4) = (1/8) \cdot NMT1(4)$$

Therefore, the total value of the considered product is (1/2) · *NMT*1(4)

Now consider the following product ($a_1 + a_2$) * ($b_1 + b_2$) relative to *CF* (5), where:

a_1 = c(1,1;2,1) * c(1,1;3,1) * c(1,1;5,1) * c(1,1;3,2)
a_2 = c(1,1;2,2) * c(1,1;3,2) * c(1,1;5,1) * c(1,1;3,1)
b_1 = c(2,1;3,1) * c(2,1;4,1) * c(2,1;5,1) * c(3,1;4,1)* c(3,1;5,1) * c(4,1;5,1) * c(2,2;3,1) * c(2,2;4,1) * c(2,2;5,1)
b_2 = c(2,2;3,2) * c(2,2;4,1) * c(2,2;5,1)* c(3,2;4,1) * c(3,2;5,1) * c(4,1;5,1) * c(2,1;3,2) * c(2,1;4,1) * c(2,1;5,1)

It is easy to verify that:

m_1 = a_1 * b_1
m_2 = a_1 * b_2
m_3 = a_2 * b_1
m_4 = a_2 * b_2

are four marks implying four different prime implicants of *CF* (5) and that:

$$val(m_1) = val(m_2) = val(m_3) = val(m_4) = (1/16) \cdot NMT1(5)$$

In more general terms, the product ($a_1 + a_2$) * ($b_1 + b_2$) can produce four marks implying four different prime implicants of *CF*(n), but the value of these marks decreases very quickly with n.

Appendix 3

Consider the following example relative to *CF* (4):

$$U = A * B = (a_1 + a_2 + \ldots + a_9) * (b_1 + b_2 + \ldots + b_9) \tag{27}$$

where:
a_1 = c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1)
b_1 = c(1,1;4,1) *c(2,1;4,1) * c(1,1;2,1)

a_2 = c(1,2;2,1) * c(1,2;3,1) * c(2,1,3,1) * !c(1,1;2,1)
b_2 = c(1,2;4,1) * c(2,1;4,1) * c(1,2;2,1) * !c(1,1;2,1)

a_3 = c(1,3;2,1) * c(1,3;3,1) * c(2,1;3,1) * !c(1,1;2,1) * !c(1,2;2,1)
b_3 = c(1,3;4,1) *c(2,1;4,1) * c(1,3;2,1) * !c(1,1;2,1) * !c(1,2;2,1)

a_4 = c(1,1;2,2) * c(1,1;3,1) * c(2,2;3,1) * !c(2,1;4,1)
b_4 = c(1,1;4,1) * c(2,2;4,1) * c(1,1;2,2) * !c(2,1;3,1)

a_5 = c(1,2;2,2) * c(1,2;3,1) * c(2,2;3,1) * !c(1,1;2,2) * !c(2,1;4,1)
b_5 = c(1,2;4,1) * c(2,2;4,1) * c(1,2;2,2) *!c(1,1;2,2) * !c(2,1;3,1)

a_6 = c(1,3;2,2) * c(1,3;3,1) * c(2,2;3,1) * !c(1,1;2,2) * !c(1,2;2,2) * !c(2,1;4,1)
b_6 = c(1,3;4,1) * c(2,2;4,1) * c(1,3;2,2) * !c(1,1;2,2) * !c(1,2;2,2) * !c(2,1;3,1)

a_7 = c(1,1;2,3) * c(1,1;3,1) * c(2,3;3,1) * !c(2,1;4,1) *!c(2,2;4,1)
b_7 = c(1,1;4,1) * c(2,3;4,1) * c(1,1;2,3) * !c(2,1;3,1) *!c(2,2;3,1)

a_8 = c(1,2;2,3) * c(1,2;3,1) *c(2,3;3,1) * !c(1,1;2,3) * !c(2,1;4,1) *!c(2,2;4,1)
b_8 = c(1,2;4,1) * c(2,3;4,1) * c(1,2;2,3) * !c(1,1;2,3) * !c(2,1;3,1) *!c(2,2;3,1)
a_9 = c(1,3;2,3) * c(1,3;3,1) * c(2,3;3,1) * !c(1,1;2,3)* !c(1,2;2,3) * !c(2,1;4,1) *!c(2,2;4,1)
b_9 = c(1,3;4,1) * c(2,3;4,1) * c(1,3;2,3) * !c(1,1;2,3) * !c(1,2;2,3) * !c(2,1;3,1) *!c(2,2;3,1)

The product specified by A*B, multiplied by the optimal completion code x = c(3,1;4,1), produces nine marks and nine prime implicants, whose total value is:

$$\left(1+(1/2)+(1/4)\right)\cdot\left(1+(1/4)+(1/16)\right)$$

It is easy to verify by extending this example to *CF*(n) that the value of an AND gate performing the product ($a_1 + a_2 + \ldots$) * ($b_1 + b_2 + \ldots$) (with constant <3,1> and <4,1>) is:

$$val(n) = (1+(1/2)+(1/4))^{n-3} \cdot (1+(1/4)+(1/16)) \cdot NMT1(n)$$

which is slightly less than the:

$$(1+(1/2)+(1/4))^{n-2} \cdot NMT1(n) \quad (28)$$

In order to prove that the solution proposed in this study is characterized by the maximum value of the gate performing the product $A*B$, analyze in detail the preceding example.

First, consider the product (a_1+a_2+a_3) * (b_1+b_2+b_3).

The product of compatibilities a_2 can be obtained from a_1 and b_2 can be obtained from b_1, by replacing variable <1,1> with variable <1,2>. In order that a_1 * b_2 = 0 and a_2 * b_1 = 0 both a_2 and b_2 must contain! c(1,1;2.1) .

Similarly, a_3 can be obtained from a_1 and b_3 can be obtained from b_1 by replacing variable <1,1> with variable <1, 3>. In order that a_1 * b_3 = 0, a_3* b_1 = 0, a_2 * b_3 = 0 and a$_3$ * b_2 = 0, both a_3 and b_3 must contain !c(1,1;2,1) * !c(1,2;2,1).

Then consider the product (a_4+a_5+a_6) * (b_4+b_5+b_6). In this case, triplet index a$_5$ can be obtained from a_4 and b_5 can be obtained from b_4 by replacing variable <1,1> with variable <1,2>. In order that a_4 * b_5 = 0 and a_5 * b_4 = 0, both a_5 and b_5 must contain !c(1,1;2,2). Similarly, in order that a_4 * b_6 = 0, a_6 * b_4 = 0, a_5 * b_6 = 0 and a_6 * b_5 = 0, both a_6 and b_6 must contain !c(1,1;2,2) * !c(1,2;2,2).

For similar reasons, both a_8 and b_8 must contain !c(1,1;2,3) while a_9 and b_9 contain !c(1,1;2,3) * !c(1,2;2,3).

As the second step of analysis considers the product (a_1+a_4+a_7) * (b_1+b_4+b_7).

The product of compatibilities a_4 can be obtained from a_1 and b_4 can be obtained from b_1, by replacing variable <2,1> with variable <2,2>. In order that a_1 * b_4 = 0, b_4 must contain! c(2,1;3,1); in order that a_4 * b_1 = 0, a_4 must contain !c(2,1;4,1). Therefore:

$$val(a_4 * b_4) = (1/4) \cdot NMT1(4)$$

In order that a_1 * b_7 = 0 and a_7 * b_1 = 0, b_7 must contain !c(2,1;3,1) * !c(2,2;3,1) and a_7 must contain !c(2,1;4,1) *!c(2,2;4,1).

Therefore:

$$val(a_7 * b_7) = (1/16) \cdot NMT1(4)$$

In the same way, all the complemented compatibilities appearing in Eq. (27) can be easily justified.

From all the data appearing in Eq. (27), it follows that the total value of the marks produced by the product (a_1+a_2+…+a_9) * (b_1+b_2+…+b_9) is equal to:

$$(1+(1/2)+(1/4)) \cdot (1+(1/4)+(1/16) \quad (29)$$

slightly less than:

$$(1+(1/2)+(1/4))^2 \quad (30)$$

which becomes Eq. (28) for n>4.

Consider again the product (a_1+a_2+a_3) * (b_1+b_2+b_3).

By replacing variable <1,1> with <1,2> in all the compatibilities of a_1 and b_1 we obtain a_2 and b_2 respectively, while by replacing <1,1> of a_1 and b_1 with <1,3> we obtain a_3 and b_3. It is apparent that the multiplication of both a_2 and b_2 by !c(1,1;2,1) and the multiplication of both a_3 and b_3 by !c(1,1;2,1) * !c(1,2;2,1) are the best solutions from the viewpoint of the values of the new marks produced by (a_1+a_2+a_3) * (b_1+b_2+b_3).

The same considerations hold exactly for the products a_2 * b_2 and a_3 * b_3.

Now consider the product (a_1+a_4+a_7) * (b_1+b_4+b_7).

By replacing <2,1> with (2,2) in all the compatibilities of a_1 and b_1 we obtain a_4 and b_4, in the first step, and by replacing <2,1> with (2,3) in all the compatibilities of a_1 and b_1 we obtain a_7 and b_7, in a second step. It is apparent that the multiplications of a_4 by !c(2,1;4,1), b_4 by !c(2,1;3,1), a_7 by !c(2,1;4,1) * !c(2,2;4,1), b_7 by !c(2,1;3,1) * 1c(2,2;3,1) are the best solutions for the new marks.

The same conclusions hold also for the products:

$$(a_2 + a_5 + a_8) * (b_2 + b_5 + b_8) and (a_3 + a_6 + a_9) * (b_3 + b_6 + b_9)$$

Notice that only the complemented variables are absolutely necessary in order that $a_i * b_j$ = 0 for all i<>j that appear in the list of values of a_i and b_j. This is equivalent to proving that Eq. (29) is the total value of the "best" product $A*B$ producing all the marks and all the prime implicants of core function with some exceptions. Indeed, the prime implicants containing variables characterized by triplet indexes equal to 3 or 4 but different from those appearing in the completion code x (in our example: <3,2>, <3,3>, <4,2>, <4,3>) do not appear in the list of prime implicants which have been generated.

For example, we can extend the list (a_1 +a_2 + …+a_9) with (a_{10} + a_{11} + … + a_{18}) and the list (b_1 + b_2 +… +b_9) with (b_{10} + b_{11} + … +b_{18})
where:

a_{10} = c(1,1;2,1) * c(1,1;3,1) * c(2,1;3,1)
b_{10} = c(1,1;4,2) *c(2,1;4,2) * c(1,1;2,1)

a_{11} = c(1,2;2,1) * c(1,2;3,1) * c(2,1,3,1) * !c(1,1;2,1)
b_{11} = c(1,2;4,2) * c(2,1;4,2) * c(1,2;2,1) * !c(1,1;2,1)

a_{12} = c(1,3;2,1) * c(1,3;3,1) * c(2,1;3,1) * !c(1,1;2,1) * !c(1,2;2,1)

b_{12} = c(1,3;4,2) *c(2,1;4,2) * c(1,3;2,1) * !c(1,1;2,1) * !c(1,2;2,1)

a_{13} = c(1,1;2,2) * c(1,1;3,1) * c(2,2;3,1) * !c(2,1;4,2)
b_{13} = c(1,1;4,2) * c(2,2;4,2) * c(1,1;2,2) * !c(2,1;3,1)

a_{14} = c(1,2;2,2) * c(1,2;3,1) * c(2,2;3,1) * !c(1,1;2,2) * !c(2,1;4,2)
b_{14} = c(1,2;4,2) * c(2,2;4,2) * c(1,2;2,2) *!c(1,1;2,2) * !c(2,1;3,1)

a_{15} = c(1,3;2,2) * c(1,3;3,1) * c(2,2;3,1) * !c(1,1;2,2) * !c(1,2;2,2) * !c(2,1;4,2)
b_{15} = c(1,3;4,2) * c(2,2;4,2) * c(1,3;2,2) * !c(1,1;2,2) * !c(1,2;2,2) * !c(2,1;3,1)

a_{16} = c(1,1;2,3) * c(1,1;3,1) * c(2,3;3,1) * !c(2,1;4,2) *!c(2,2;4,2)
b_{16} = c(1,1;4,2) * c(2,3;4,2) * c(1,1;2,3) * !c(2,1;3,1) *!c(2,2;3,1)

a_{17} = c(1,2;2,3) * c(1,2;3,1) *c(2,3;3,1) * !c(1,1;2,3) * !c(2,1;4,2) *!c(2,2;4,2)
b_{17} = c(1,2;4,2) * c(2,3;4,2) * c(1,2;2,3) * !c(1,1;2,3) * !c(2,1;3,1) *!c(2,2;3,1)

a_{18} = c(1,3;2,3) * c(1,3;3,1) * c(2,3;3,1) * !c(1,1;2,3)* !c(1,2;2,3) * !c(2,1;4,2) *!c(2,2;4,2)
b_{18} = c(1,3;4,2) * c(2,3;4,2) * c(1,3;2,3) * !c(1,1;2,3) * !c(1,2;2,3) * !c(2,1;3,1) *!c(2,2;3,1)

This increment of the lists (a_1 + a_2 +…) and (b_1 + b_2 +…) can be updated as follows:

$$\begin{aligned}&\left(a_1 + \ldots + a_{10} + \ldots + a_{19} + \ldots + a_{28} + \ldots + a_{37} + \ldots\right.\\&\left.+\, a_{46} + \ldots + a_{55} + \ldots + a_{67} + \ldots + a_{73} + \ldots\right)\\&\left(b_1 + \ldots + b_{10} + \ldots + b_{19} + \ldots + b_{28} + \ldots + b_{37} + \ldots\right.\\&\left.+\, b_{46} + \ldots + b_{55} + \ldots + b_{67} + \ldots + b_{73} + \ldots\right)\end{aligned}$$

where also the completion code should be updated:

$$\begin{aligned}&x = c(3,1;4,1) * c(3,1;4,2) * c(3,1;4,3) * c(3,2;4,1)\\&* c(3,2;4,2) * c(3,2;4,3) * c(3,3;4,1) * c(3,3;4,2) * c(3,3;4,3)\end{aligned}$$

Thus, the 81 prime implicants of *CF* (4) will be generated, but the values of many of them will be very small because all the products a_1 * b_j must be equal to 0. Besides, the multiplication of every mark by the completion code x will dramatically reduce the value of the corresponding prime implicants.