Hypercube Graph Decomposition for Boolean Simplification: An Optimization of Business Process Verification

Mohamed NAOUM[1], Outman EL HICHAMI[2],
Mohammed AL ACHHAB[3], Badr eddine EL MOHAJIR[4]
New Technology Trends Team, Science and Technology Center for Doctoral Studies,
Abdelmalek Essaâdi University,
Tetouan, Morocco

***Abstract*—This paper deals with the optimization of business processes (BP) verification by simplifying their equivalent algebraic expressions. Actual approaches of business processes verification use formal methods such as automated theorem proving and model checking to verify the accuracy of the business process design. Those processes are abstracted to mathematical models in order to make the verification task possible. However, the structure of those mathematical models is usually a Boolean expression of the business process variables and gateways. Thus leading to a combinatorial explosion when the number of literals is above a certain threshold. This work aims at optimizing the verification task by managing the problem size. A novel algorithm of Boolean simplification is proposed. It uses hypercube graph decomposition to find the minimal equivalent formula of a business process model given in its disjunctive normal form (DNF). Moreover, the optimization method is totally automated and can be applied to any business process having the same formula due to the independence of the Boolean simplification rules from the studied processes. This new approach has been numerically validated by comparing its performance against the state of the art method Quine-McCluskey (QM) through the optimization of several processes with various types of branching.**

Keywords*—Business process verification; minimal disjunctive normal form; Boolean reduction; hypercube graph; Karnaugh map; Quine-McCluskey

I. Introduction

Business processes are key assets of any organization or information system [1], [2]. They are the communication interface and the medium of exchange between the organization stakeholders [3].

BP describe the core business and govern the operation of a system. **Business Process Model and Notation (BPMN)** is the wide used standard for modeling BP in view of its simplicity and usability [4], [5]. Nevertheless, BP may contain structural flaws [5] due to poor design or human errors. Hence, the verification task is a crucial step between the modeling and the execution phases of any BP. The complexity of real-life BP and the use of automated modeling tools often lead to complex models called "spaghetti" process models [6], [7] where manual verification is difficult to perform [8]. Therefore, automated formal methods are used instead. Automatic verification includes: **Model Checking (MC)** [5], [9] and **Automated Theorem Proving (ATP)** [10], [11].

The MC approach uses software called model checker to exhaustively check whether an abstraction equivalent structure of the BP satisfies some properties expressed in temporal logics. **Simple Promela INterpreter (SPIN)** is a widely used model checker that verifies if a model writen in a C-like modeling language called **Process Meta LAnguage(Promela)**, meets properties expressed as **Linear Temporal Logic** (LTL) formulas [12], [13], [14]. Although this method has the advantage of indicating the counter example violating the checked propriety, it suffers from the state explosion problem [12] since its complexity is too high and the number of states grows exponentially.

The ATP (or automated deduction) is a subfield of mathematical logic dealing with automatic (or semi-automatic) proving of mathematical theorems. The computer programs allowing this task are called theorem provers [15].

First-order theorem proving is one of the most mature subfields of ATP thanks to its expressivity that allows the specification of arbitrary problems [16]. However, some statements are undecidable [17] in the theory used to describe the model. thereby, current research [18], [17], [19] deal with the challenge of finding subclasses of first-order logic(FOL) that are suitable and decidable in the mapping of such models.

Higher order logics are more expressive and can map wider range of problems than FOL, but theorem proving for these logics is not as developed as in the FOL[20].

Regardless the used approach to verify a BP, its logical structure is deducted as a propositional logic formula written in Disjunctive Normal Form (DNF) [2], [7]. The DNF can be reduced to a minimal form in order for the manipulation and practical implementation to become more efficient. Thus, an optimization of the PB verification is achieved.

Since the simplification of Boolean expressions is extensively used in the analysis and design of algorithms and logical circuits, several methods were developed to perform this task:

– The **algebraic manipulation** of the Boolean expressions aims at finding an equivalent expression by applying the laws of Boolean algebra. However, for such methods, there is no fixed algorithm to be used to minimize a given expression. Thus, choosing which

Boolean theorems to apply is left to the expert's ability.

– The **Karnaugh map** which is a pictorial and straightforward method [21]. First, a grid of the truth table of the function to minimize has to be drawn. The minterms of this grid have to be arranged in Gray code which makes each pair of adjacent cells different only by the value of one variable.
The problem is then converted into finding rectangular groups of adjacent cells containing ones, these groups should have an area that is a power of two (i.e., 1, 2, 4, 8 ...). Consequently, unwanted variables are eliminated. This method is easy to understand, however it is a manual process which is not practical when dealing with more than six variables [22].

– The **tabulation method** (also known as **Quine McCluskey** algorithm) [23] is a useful minimization algorithm when dealing with more than 4 variables. It has a tabular form that makes it easy to implement in computer programs. It consists of finding all prime implicants of the function to minimize, and then tries to find the necessary ones that cover the function. Although this method is more practical than the previous ones, it is impaired by the redundancy during the search of prime implicants. Moreover, the application of Petrick's method [24] in a second phase is required to define essential prime implicants and resolve the cyclic covering problem.

This article introduces a novel technique to optimize the verification of a BP by simplifying its equivalent logical formula written in the **Disjunctive Normal Form (DNF)**. This new simplification algorithm searches for the largest **hypercubes** of lower dimensions (called *elements*) that are enough to cover all vertices in a partial cube graph mapping of the BP. A minimal equivalent DNF is then expressed as a disjunction of the necessary hypercube abstractions in this *elements* coverage.

The rest of this paper is structured as follows: Section II describes how the BP is modeled in BPMN. Section III presents the main Boolean algebra simplification rules as well as the hypercube properties that are used in the developed algorithm. Section IV explains in details the simplification algorithm and goes throw the used speedup tweaks. Our findings are presented and discussed in Section V. Finally, a conclusion is given.

II. Business Process Modeling and Notation

The most used business process modeling standard is Business Process Model and Notation (BPMN). It is a specification of the Object Management Group (OMG) [25]. The modeling is done by interconnecting standard graphical symbols grouped in five categories:

The **Swimlanes** and **Artifacts** categories are used to group objects into lanes and to provide additional descriptions. The **Data elements** category is used to describe the flow of the data through the process.

The main role of the three categories above is to increase readability of the model without effecting its execution. Therefore the whole BP flow can be described with the remaining two categories: **Flow Objects** and **Connecting Objects** [25].

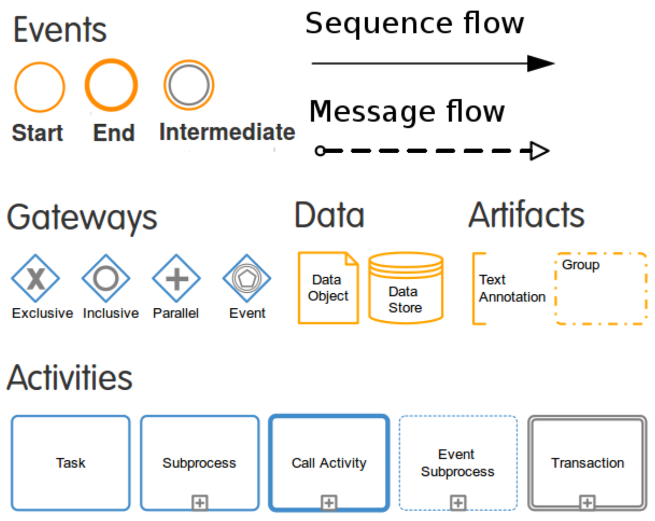

Fig. 1. Main flow objects and sections flows of BPMN 2.0.

The BPMN 2.0 specifies three Flow Objects: 1) Events, 2) Activities and 3) Gateways (see Fig. 1). These elements are connected using **Connecting Objects** especially *Sequence flows*.

The *Event* elements indicate the various incidents that can occur during the process execution. Three main type of events can be distinguished according to their trigger time: 1) Start Events, 2) End Events, and 3) Intermediate Events. They indicate the beginning or the end of a process or simply any event that may arise in-between.

The *Activity* elements are used to indicate any performed task in a process. Depending on the level of abstraction, an Activity may be compound or atomic.

The *Sequence flows* are the arcs connecting related events and activities. They define the chronological order of the elements within a process. If the activation of a sequence flow depends on some condition, then a Boolean variable is defined above it. Thus the immediate successor element is activated only if this condition is considered to be true.

The *Gateway* elements are used to indicate any divergence or convergence in a Sequence Flow. Depending of their behavior, the five types of Gateways are: Exclusive, Inclusive, Parallel, Event-Based, and Complex. They determine the branching, forking, merging, and joining of paths.

The graph composed of Flow objects and their Sequence Flows connections describes the eventual executions of a BP. Each path of the graph going from the start to the end events indicates a single execution scenario. As an example, Fig. 2 shows a simplified payment/delivery BP.

Once the modeling of the BP is done, the designer must choose which verification method to apply. The structure of the BP model is then extracted as a mathematical expression that depends on the used gateways and the sequence flows branching. The next section will present the necessary elements

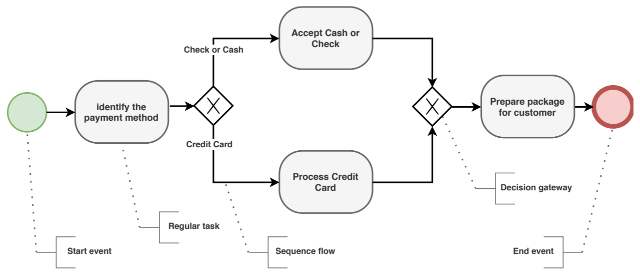

Fig. 2. An Example of a Simple payment/delivery BP.

used to map the logical structure of a BP and the main rules used to simplify its equivalent formula.

III. Binary Representation and Reduction Rules

A. Definitions

1) Boolean variable: A Boolean variable is a variable that takes only one of the logical values: either 1 (meaning $True$) or 0 (meaning $False$). The complement of a variable A is denoted $\overline{A}$ and has the opposite value of A. A literal is either the logic variable A or its complement $\overline{A}$.

2) Minterm: A Minterm is a product (conjunction) of all the variable literals. For instance, for three Boolean variables A, B, and C the expressions $AB\overline{C}$, $A.B.\overline{C}$, and $A \wedge B \wedge \overline{C}$ denote the same minterm. It means that C has the value 0 and both A and B have the value 1. By assigning a power of 2 to each variable of a minterm $V_{n-1}...V_2V_1V_0$ composed of n variables V_i, the shorthand notation is m_d where d denotes the decimal value of the binary expression $V_{n-1}...V_2V_1V_0)_2$. For example, m_6 is the short hand notation of $AB\overline{C}$ because $110)_2 = 6$.

3) Disjunctive Normal Form (DNF): A logical formula is considered to be in Disjunctive Normal Form (DNF) if and only if it is a disjunction (sum) of one or more conjunctions (products) of one or more literals [26]. A DNF formula is in full disjunctive normal form if each of its variables appears exactly once in every conjunction (minterm). The only propositional operators in DNF are and (denoted with . or $\wedge$), or (denoted with + or $\vee$), and not (denoted with $\neg A$ or $\overline{A}$). The not operator can only be used as part of a literal, which means that it can only precede a propositional variable. The following formula of three variables A, B, and C is in DNF:

$$f = \overline{A}\ B\ C\ +\ A\ \overline{B}\ \overline{C}\ +\ A\ B\ \overline{C}\ +\ A\ B\ C \qquad (1)$$

It can be written in shorthand notation as follow:

$$f = m_3 + m_4 + m_6 + m_7 \qquad (2)$$

B. Boolean Algebra

1) Boolean algebra identities: In Boolean algebra, there are four basic identities for addition (logical or) and four for multiplication (logical and) that holds true for all possible values of a Boolean statement variables. Table I gives a summary of those identities:

2) Boolean algebra properties: In Boolean algebra, there are three basic properties: commutative, associative, and distributive. Table II gives a summary of those properties:

TABLE I. Boolean algebraic identities

Addition	$A \vee 0 = A$	$A \vee 1 = 1$	$A \vee A = A$	$A \vee \overline{A} = 1$
multiplication	$A \wedge 0 = 0$	$A \wedge 1 = A$	$A \wedge A = A$	$A \wedge \overline{A} = 0$

TABLE II. Boolean algebra properties

Addition ($\vee$)	Multiplication ($\wedge$)
$A \vee B = B \vee A$	$A \wedge B = B \wedge A$
$A \vee (B \vee C) = (A \vee B) \vee C$	$A \wedge (B \wedge C) = (A \wedge B) \wedge C$
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$	

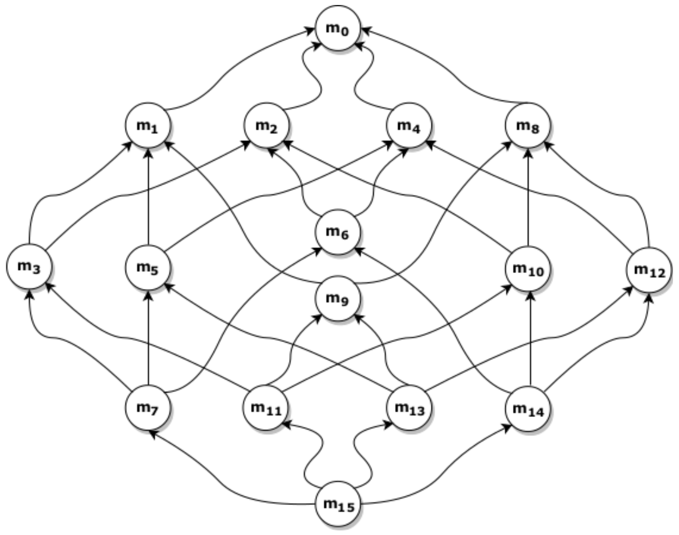

Fig. 3. Hasse diagram of the hypercube graph Q_4

3) Boolean simplification rules: By using the identities and properties of Boolean algebra, a Boolean statement can be simplified by reducing the number of literals using the following rules:

$$ABC + \overline{A}BC = BC \qquad (3)$$

$$A + AB = A \qquad (4)$$

$$A + \overline{A}B = A + B \qquad (5)$$

$$(A + B)(A + C) = A + BC \qquad (6)$$

C. The Hypercube Graph Representation

A Boolean statement of n variables can be written in DNF with at most 2^n minterms of n literals. By creating a vertex for each minterm m_i and linking each two vertices when their binary representations differ in a single digit (the Hamming distance of their minterms is one), a hypercube graph (noted n-cube or Q_n) is created[27]. Fig. 3 gives a flat representation of the hypercube graph Q_4.

A hypercube graph of n vertices can be viewed as the disjoint union of two hypercubes Q_{n-1} if an edge is added from each vertex/minterm in one copy of Q_{n-1} to the corresponding minterm/vertex of the other copy. As shown in Fig. 4, the joining edges form a perfect matching between the blue and black vertices.

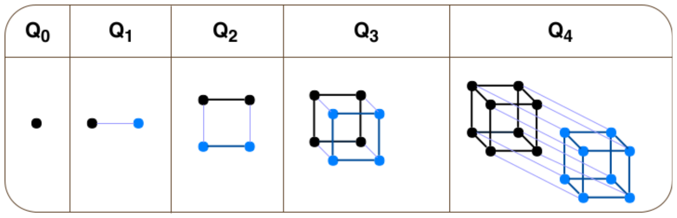

Fig. 4. Construction of hypercube Q_n from two Q_{n-1} hypercubes

In fact, every hypercube Q_n of $n > 0$ is composed of *elements*, or n-cubes of a lower dimension, on the (n-1)-dimensional surface on the parent hypercube. The smallest *elements* are the vertices (points). There is 2^n of them.

In general, the number of m-cubes on the boundary of a given n-cube is $E_{m,n} = 2^{n-m}\binom{n}{m}$ where $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ is the binomial coefficient.

A partial cube is an isometric subgraph of a hypercube. The distance between any two vertices in the subgraph is the same as the distance between those vertices in the hypercube.

Lemma III.1 *Let Q_n be a hypercube graph with $n > 0$ minterms m_i where $i \in [0, 2^n[$. Let f be a DNF formula given by the disjunction of all Q_n minterms. Then n variables of f can be simplified. The abstracted equivalent formula is easily obtained by identifying the common literals between the minterm with maximum shorthand notation value (denoted m_{max}) and the one with the minimum shorthand notation value (denoted m_{min}). This abstraction is chosen to be called: abstraction m_{max} with filter m_{min}.*

Proof: For instance, if $n = 1$ then Q_1 is composed of two minterms m_0 and m_1 of one variable v_0. By applying the identity $v_0 + \overline{v_0} = 1$, an abstraction of the variable v_0 is given (abstraction m_1 with the filter m_0).

If $n = 2$ then Q_2 is composed of four minterms $\{m_0, m_1, m_2, m_3\}$ each one is composed of two variables v_0 and v_1. By applying the same identity to two opposite sides of Q_2 an abstraction of the variables v_0 and v_1 is given (the abstraction m_3 with the filter m_0). In fact:

$$f = m_0 + m_1 + m_2 + m_3 = \overline{v_1}.\overline{v_0} \vee \overline{v_1}.v_0 \vee v_1.\overline{v_0} \vee v_1.v_0$$
$$f = \overline{v_1}.(\overline{v_0} \vee v_0) \vee v_1.(\overline{v_0} \vee v_0) = \overline{v_1} \vee v_1 = 1$$

Let us assume that the lemma III.1 is correct for any $n > 0$. Let $Q1_n$ and $Q2_n$ be two hypercubes that their disjoint union form the hypercube Q_{n+1}. Each minterm $m_x = m_{\overline{V_n}V_{n-1}...V_2V_1V_0)_2}$ in $Q1_n$ forms a perfect matching with another minterm $m_y = m_{V_nV_{n-1}...V_2V_1V_0)_2}$ in $Q2_n$. m_x and m_y can be abstracted to m_x because they differ by the value of a single variable v_n. In fact:

$$f = m_x + m_y = \overline{V_n}V_{n-1}...V_2V_1V_0 \vee V_nV_{n-1}...V_2V_1V_0$$
$$f = (\overline{V_n} \vee V_n)V_{n-1}...V_2V_1V_0 = V_{n-1}...V_2V_1V_0 = m_x$$

which gives an abstraction of the variable V_n. As a result, the hypercube Q_{n+1} gives an abstraction of $n+1$ variables: n variables with the hypercube $Q1_n$ plus that of V_n. ■

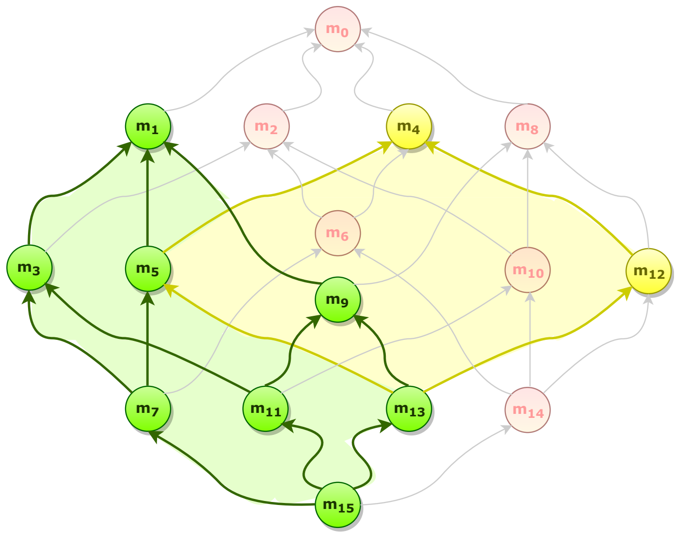

Fig. 5. Reduction of a full DNF of 4 variables to hypercubes Q_2 and Q_3

In the next section, an explanation of how the lemma III.1 can be used as a key stone to perform the simplication of any formula written in DNF is given.

IV. Simplification Algorithm

In order to simplify a Boolean expression written in DNF, its expression is represented as a partial cube PQ_n of the hypercube graph Q_n, with n the number of variables in the DNF formula. The developed algorithm consists in finding the largest *elements* (hypercubes) Q_m, with $m \leq n$, so that their disjoint union covers all vertices of the partial cube PQ_n. The fewer is the number of necessary hypercubes Q_m, the more abstract is the equivalent formula. As an example, the following DNF formula can be considered:

$$\begin{aligned} f(A,B,C,D) = {} & \overline{A}\ \overline{B}\ \overline{C}D + \overline{A}\ \overline{B}CD + \overline{A}B\overline{C}\ \overline{D} + \\ & \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}\ \overline{C}D + A\overline{B}CD + \\ & AB\overline{C}\ \overline{D} + AB\overline{C}D + ABCD \end{aligned} \quad (7)$$

This formula is represented as a partial cube PQ_4 with vertices m_1, m_3, m_4, m_5, m_7, m_9, m_{11}, m_{12}, m_{13}, and m_{15}. Fig. 5 shows that the vertices of PQ_4 (green and yellow vertices) can be covered with the disjoint union of two hypercubes Q_3 and Q_2.

Using lemma III.1, three variables A, B, and C can be reduced with the hypercube Q_3 composed of vertices $\{m_1, m_3, m_5, m_7, m_9, m_{11}, m_{13}, m_{15}\}$. Thus Q_3 is reduced to m_{15} with the filter m_1 which is equivalent to the expression D since it is the only variable that remains with the same value in all minterms of Q_3 (we have $m_{max} = m_{15} = m_{111\underline{\mathbf{1}})_2}$ and $m_{min} = m_1 = m_{000\underline{\mathbf{1}})_2}$ the abstraction is $---\underline{\mathbf{1}})_2$).

The hypercube Q_2, composed of $\{m_4, m_5, m_{12}, m_{13}\}$, gives an abstraction of tow variables A and D. Thus Q_2 is reduced to m_{13} with the filter m_4 which is equivalent to the expression $B\overline{C}$ (we have $m_{max} = m_{13} = m_{1\underline{\mathbf{10}}1)_2}$ and $m_{min} = m_4 = m_{0\underline{\mathbf{10}}0)_2}$ the abstraction is $-\underline{\mathbf{10}}-)_2$).

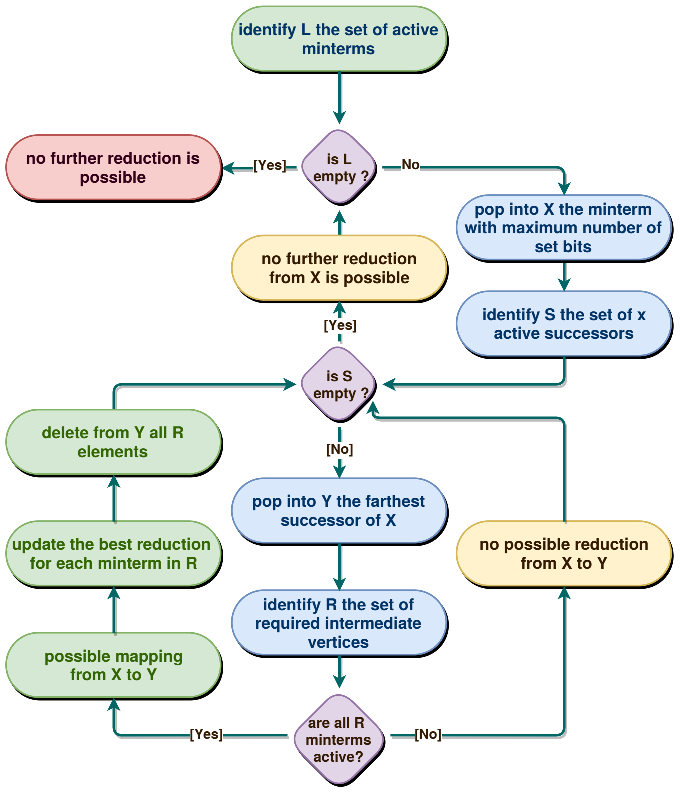
identify L the set of active minterms
is L empty ?
[Yes]
No
no further reduction is possible
pop into X the minterm with maximum number of set bits
no further reduction from X is possible
identify S the set of x active successors
[Yes]
is S empty ?
[No]
delete from Y all R elements
pop into Y the farthest successor of X
update the best reduction for each minterm in R
no possible reduction from X to Y
identify R the set of required intermediate vertices
possible mapping from X to Y
[Yes]
are all R minterms active?
[No]

Fig. 6. Organogram of the proposed Boolean reduction algorithm

Finally, the disjunction of this two abstractions gives the minimal formula :

$$(A, B, C, D) = D + B\overline{C}. \quad (8)$$

If a vertex of the partial cube is covered by multiple hypercubes, the largest one has to be considered. That way, each vertex is surely covered with the most abstract expression.

A simplified version of the reduction algorithm is summarized in Fig. 6. The algorithm starts with identifying the vertices of the partial cube PQ that maps all the minterms of the formula to minimize. Then, it tries to find, for each vertex m_i of the PQ, the largest hypercube (or hypercubes if there are many with the same size) that contains m_i. Finally, the algorithm gives priority to external vertices then holds only the necessary hypercubes to cover them all. The abstraction given by those hypercubes is the minimal equivalent expression of the DNF to minimize.

In the next section, the performance of the proposed algorithm will be compared with the Quine-McCluskey method (QM).

All abstractions were performed using a Python implementation of the developed algorithm. They were then compared to the an optimized Python implementation of the standard Quine-McCluskey algorithm (This implementation is included in the digital electronics simulation library **BinPy**).

The experiments were carried out on a conventional laptop computer equipped with an $Intel\ i5$ processor and $8GB$ of RAM.
For each dimension n, with $n \leq 4$, all formulas were tested since there is only 65812 possible ones ($2^{2^1} + 2^{2^2} + 2^{2^3} + 2^{2^4} = 65812$). For $n > 4$, the formulas to minimize were chosen up to $x = 2^{24}$.

For each test, the running times, for both methods, were recorded starting from the feeding of the formula to minimize until the reception of the minimal equivalent DNF. The integer representing the input formula is then incremented for the next test. Since the execution time can vary significantly depending on the input size, we choose to plot the relative percent difference of the two algorithms runtimes. Each scatter in Fig. 7 represents the result of one test that is given by the formula :

$$100 * \frac{QM's\ runtime\ -\ Our\ algorithm's\ runtime}{minimum\ of\ both\ runtimes}$$

A blue scatter indicates a result in favor of the proposed algorithm while a red scatter indicates a result in favor of the QM algorithm.

The plot was generated using the python data visualization library **Seaborn** based on matplotlib.

V. Results and Discussion

From Fig. 7 we can conclude that our algorithm has better performances than the QM Method since it has better results in 89.40% cases of the 2^{24} conducted tests. Moreover, the proposed algorithm is over 400% faster in 1380450 cases while the QM method is over 400% faster only in 746 cases. Also this percent difference can reach over 2000% in 3829 cases in favor of the developed algorithm and in no case in favor of the QM method.

One advantage of the new algorithm introduced in this work, is that it follows a top-down approach: it searches first for the largest hypercube that covers a minterm which means that the algorithm does not waste time on smaller hypercubes with less abstraction. In the counterpart, the Quine-McCluskey algorithm follows a down-top approach: it tries to find all prime implicants of size 2 then size 4 and so on, which means that it wastes time on multiple partial prime implicants before reaching the optimum formula.

A second advantage of the developed algorithm is that, unlike for the tabular method, there is no need to use the Pitrick's method to solve the problem of cyclic covering. It is simply solved by holding first the coverage of the external vertices of the decomposition hypercubes.

Finally, another advantage is the use of binary operations that are directly supported by the microprocessor; it applies a simple binary and/or filters to find the successors of a given vertex or to store the previous found coverage. For instance, if there are six variables then there are $2^6 = 64$ minterms, instead of using a loop of 64 iterations, a single microprocessor operation can be used to filter the active minterms in the partial cube.

VI. Conclusion

Business Processes are indubitable tools for the modern business planning, but those models can include structural flaws that are hard to detect with manual verification, which gives extreme importance to automatic verification. Formal

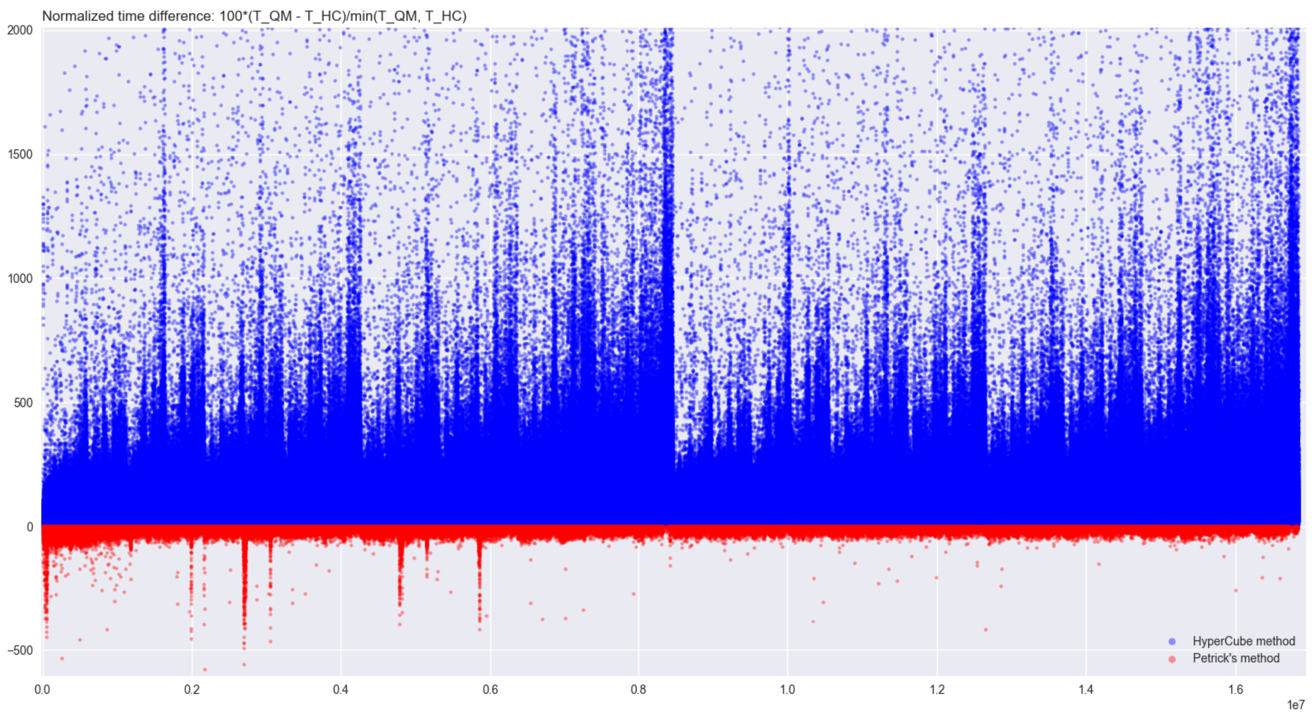

Fig. 7. Relative percent difference of the two algorithms' runtimes.

methods verification algorithms suffer from the high complexity since the problem they try to solve is NP-hard, hence the necessity to reduce the problem size by minimizing the number of literals.

In this paper, a novel technique of business processes simplification has been presented. A simplification tool that performs literals reduction using hypercube decomposition has been built. Moreover, the simplification algorithm was entirely automated which makes the optimization task accessible to the regular BP designers. Promising subject of research can be explored in further depth, such as how machine learning algorithms could be used to accelerate the simplification algorithm, how the algorithm can be modified to reduce the spatial complexity, and finally, the possibility of adapting the algorithm, view its characteristics, for quantum computing.

References

[1] R. Heinrich, P. Merkle, J. Henss, and B. Paech, "Integrating business process simulation and information system simulation for performance prediction," *Softw Syst Mod*, vol. 16, no. 1, pp. 257–277, 2017.

[2] D. Batory, "Feature models, grammars, and propositional formulas," in *International Conference on Software Product Lines*. Springer, 2005, pp. 7–20.

[3] J. Stark, "Product lifecycle management," in *Product Lifecycle Management*. Springer, 2015, vol. 1, pp. 1–29.

[4] H. Völzer, "An overview of bpmn 2.0 and its potential use," in *International Workshop on Business Process Modeling Notation*. Springer, 2010, pp. 14–15.

[5] W. M. P. Van Der Aalst, M. L. Rosa, and F. M. Santoro, "Business process management - don't forget to improve the process!" *Bus Inform Syst Eng*, vol. 58, no. 1, pp. 1–6, 2016.

[6] V. Gruhn and R. Laue, "Complexity metrics for business process models," in *9th international conference on business information systems (BIS 2006)*, vol. 85. Citeseer, 2006, pp. 1–12.

[7] K. Batoulis, A. Meyer, E. Bazhenova, G. Decker, and M. Weske, "Extracting decision logic from process models," in *International Conference on Advanced Information Systems Engineering*. Springer, 2015, pp. 349–366.

[8] A. Förster, G. Engels, T. Schattkowsky, and R. V. D. Straeten, "Verification of business process quality constraints based on visual process patterns," in *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007, June 5-8, 2007, Shanghai, China*. IEEE Computer Society, 2007, pp. 197–208.

[9] A. Elgammal, O. Turetken, W.-J. van den Heuvel, and M. Papazoglou, "Formalizing and appling compliance patterns for business process compliance," *Softw Syst Model*, vol. 15, no. 1, pp. 119–146, 2016.

[10] X. Tan, Y. Gu, and J. X. Huang, "An ontological account of flow-control components in bpmn process models," *Big Data Inf Anal*, vol. 2, no. 2, pp. 177–189, 2017.

[11] S. Mallek, N. Daclin, V. Chapurlat, and B. Vallespir, "Enabling model checking for collaborative process analysis: from bpmn to 'network of timed automata'," *Entrep Inf Syst - UK*, vol. 9, no. 3, pp. 279–299, 2015.

[12] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Progress on the state explosion problem in model checking," in *Informatics*. Springer, 2001, pp. 176–194.

[13] Y. Li, A. Deutsch, and V. Vianu, "A spin-based verifier for artifact systems," *Comput Res Rep*, vol. abs/1705.09427, 2017.

[14] C. Wolter, P. Miseldine, and C. Meinel, "Verification of business process entailment constraints using spin," in *international symposium on engineering secure software and systems*. Springer, 2009, pp. 1–15.

[15] L. C. Paulson, *Isabelle: A generic theorem prover*, ser. Lecture Notes in Computer Science. Springer Science & Business Media, 1994, vol. 828.

[16] G. Buday, "Logic in computer science: Modelling and reasoning about systems by huth michael and ryan mark, isbn 0 521 54310 x." *J Funct Program*, vol. 18, no. 3, pp. 421–422, 2008.

[17] S. Halfon, P. Schnoebelen, and G. Zetzsche, "Decidability, complexity, and expressiveness of first-order logic over the subword ordering," in *Logic in Computer Science (LICS), 2017 32nd Annual ACM/IEEE Symposium on*. IEEE, 2017, pp. 1–12.

[18] M. Elberfeld, M. Grohe, and T. Tantau, “Where first-order and monadic second-order logic coincide,” *ACM Trans Comput Logic*, vol. 17, no. 4, p. 25, 2016.

[19] M. Lamotte-Schubert, “Automatic authorization analysis,” Ph.D. dissertation, Saarland University, Saarbrücken, Germany, 2015.

[20] A. Gawanmeh and A. Alomari, “Challenges in formal methods for testing and verification of cloud computing systems,” *Scalable Comput Pract Exp*, vol. 16, no. 3, pp. 321–332, 2015.

[21] M. Karnaugh, “The map method for synthesis of combinational logic circuits,” *T Am Inst Elec Eng 1*, vol. 72, no. 5, pp. 593–599, 11 1953.

[22] T. K. Jain, D. S. Kushwaha, and A. K. Misra, “Optimization of the quine-mccluskey method for the minimization of the boolean expressions,” in *Fourth International Conference on Autonomic and Autonomous Systems, ICAS 2008, 16-21 March 2008, Gosier, Guadeloupe*. IEEE Computer Society, 2008, pp. 165–168.

[23] W. V. Quine, “The problem of simplifying truth functions,” *Am Math Mon*, vol. 59, no. 8, pp. 521–531, 1952.

[24] S. R. Petrick, “A direct determination of the irredundant forms of a boolean function from the set of prime implicants,” *AFCRC-TR-56*, vol. 10, p. 110, 1956.

[25] J. Mendling and M. Weidlich, Eds., *Business Process Model and Notation - 4th International Workshop, BPMN 2012, Vienna, Austria, September 12-13, 2012. Proceedings*, ser. Lecture Notes in Business Information Processing, vol. 125. Springer, 2012.

[26] J. Cohen, “Review of ”introduction to lattices and order by b. a. davey and h. a. priestley”, cambridge university press,” *ACM SIGACT News*, vol. 38, no. 1, pp. 17–23, 2007.

[27] Y. Saad and M. H. Schultz, “Topological properties of hypercubes,” *IEEE T Comput*, vol. 37, no. 7, pp. 867–872, 1988.