**EPH - International Journal of Biological & Pharmaceutical Science**
ISSN (Online): 2208-2166
Volume 01 Issue 1-January -2015

DOI:https://doi.org/10.53555/eijbps.v1i1.3

# APROACHING TO HOSOYA INDEX

**Juan Manuel Dato Ruiz***
**Graduate Application Development*

***Corresponding Author:-***
*E-mail:- jumadaru@gmail.com*

**Abstract:-**
*Chemistry must not be an Experimental Science, its nature is engineering. For that reason, some methods look for working in a bureau better than in laboratory. Constructing models to get the conversion from Theory and Practice complete is one of the most logic projects everyone is very confident for doing. So, in the other side, Informatics and Engineering of Computers must open gateways to make possible the Management of the Molecular Design easily. But experts of computers know what kind of barriers exists, the strongest for Graph Theory is counting cases which assert something expected: those problems are defined and usually counted in Sharp-P class. This document try to approach technology to any required petition from that expects more from computers.*

**Keywords:-***SAT, sharp*

Copyright 2015 EIJBPS
Distributed under Creative Commons CC-BY 4.0 OPEN ACCESS

**INTRODUCTION**
Architects construct buildings, but they always begin with a project designed in a bureau. Engineers are mastered by the Architect and generate a budget in hours, materials and workers. With a very realistic project, the result will generate a minimum margin of error from the final prize and the initial. In Informatics, for example, even not having a way to know at all if the final Design will be accepted or rejected in the middle of the project, the time of finalization is budgetable too. Time we need to investigate a concept to Design a project is able to get a budget in Engineering. So that property is interesting for Chemistry and the Molecular Design.

Molecular Design works with different ways of comparing the molecules and their empirical properties. To get this, every molecule could be associated by a structure of datas which properties could be associated to the molecule again. One of the most horrible index to compute is the Hosoya Index, which requires the solving of a problem called match-sharp in Graph Theory.

The problem of matching in a Graph **G** (*V*, *E*), with *V* verteces and *E* edges, is the problem of constructing a subset of *E* where every element has no vertex in common. So, the match-sharp problem of a Graph is knowing the number of sets that asserts that property. Even solving the match easily, match-sharp is considered very complicated uder a function of time expressable in O ($k \cdot n^k$) where *n* is the number of edges of the graph. If the function of time were expressable in that way, we could say match-sharp is in P, so every transformation of structure of data in P which origin were match-sharp could be solved in P. And those things means that there exists a budgetable machine able to solve the problem in reality. In this document, an interesting tool will be showed to get closer results.

**1. Choicing operator**
I define **choicing operator** (**1.1**) as a Boolean function whose result is True iff <u>one and only one</u> of its members is true. When we have a product of choicing operator I will call it a **choicing formula**
**(1.2)**.

$$(A_1 \mid A_2 \mid \dots \mid A_n) \quad (1.1)$$

The choicing formula **(1.2)** is divided in clausules. And clausules are divided in literals. Each literal is an atomic formula that can be negated.

$$(A1,1 \mid A1,2 \mid \dots \mid A1,n[1])\cdot(A2,1 \mid A2,2 \mid \dots \mid A2,n[2])\cdot\ldots\cdot(Am,1 \mid Am,2 \mid \dots \mid Am,n[m]) \quad (1.2)$$

This operator and the operator **and** are enough to represent every Boolean formula and the **choicing formula** is very well conditionated to manage the number of satisfied cases of the formula, as I will explain later.

When **choicing operator** works with only two operands, then it is equivalent with **XOR** operator. And moreover, if we have a **choicing formula** like in **(1.2)**, we can transformate it in a **choicing-3 formula** like in **(1.3)**, considering the max number of operators must be 3 in every clausule.

$(A_{1,1} \mid A_{1,2} \mid A_{1,3})\cdot(A_{2,1} \mid A_{2,2} \mid A_{2,3})\cdot\ldots\cdot(A_{m,1} \mid A_{m,2} \mid A_{m,3})$(1.3)That equivalence is very easy to obtain avoiding a combinatorial explosion using the fórmula **(1.4)**.

$$(A_1 \mid A_2 \mid \dots \mid A_n) = (A_1 \mid A_2 \mid X_1)\cdot(X_1 \mid A_3 \mid X_2)\cdot\ldots\cdot(X_{n-2} \mid A_{n-1} \mid A_n) \quad (1.4)$$

**1 2. Boolean algebra with choicing formulas**
If you want to represent every formula in Boolean algebra, it is needed simply a theorem. That theorem is described in **(2.1)** and there is where we can find a relationship between logic operators **AND** and **XOR** with choicing formulas up to three literals.

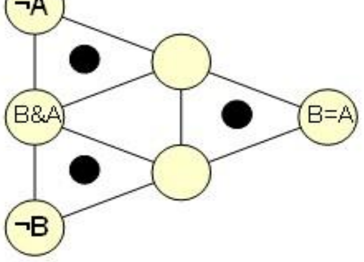


**Figure 1: Theorem (2.1)**

$$\mathbf{(\neg A \mid Z_1 \mid A_{\text{AND}} B)\cdot(\neg B \mid Z_2 \mid A_{\text{AND}} B)\cdot(Z_1 \mid Z_2 \mid \neg A_{\text{XOR}} B) = 1} \quad \mathbf{(2.1)}$$

In article *Relationship between P and NP* [1], you can find how every formula can be easily converted as a product of clausules like (A = B $_{\text{AND}}$ C) where A, B and C are literals: atomic boolean formulas which can be negated.

From that consideration, it is possible to construct great structures to define different kind of asserts, like in Figure 2.

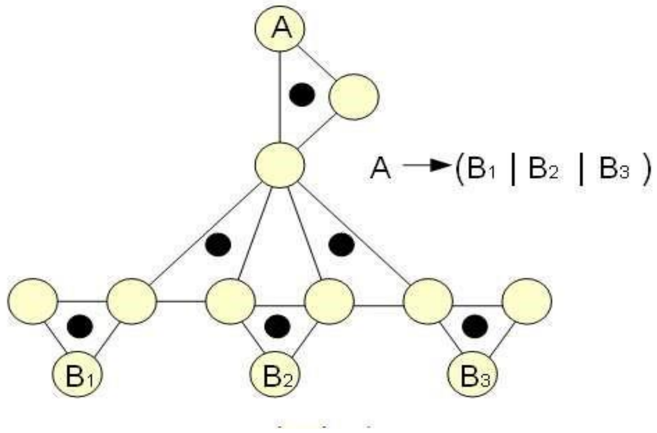


**Figure 2: How to get a condicionated choicing formula**

**3. Architecture of data of choicing. A possible polynomial implementation.**

The structure of data presented is a set of tables with a sort of triangular organization. In the way that every pair of clausules in the formula gets its own table. Matrix will contain rows and columns as literals in their clausules. So for every pair (I, J) of clausules en the formula with I < J, it is defined **table (I, J)** as the matrix in the structure which contains columns as literals in clausule I and rows as literals in clausule J. In this document **table (I, J)** = **table (J, I)** will be considered, even there could be reached good implementation where it could be otherwise. The matrix **table (0, 1)** it is called main table.

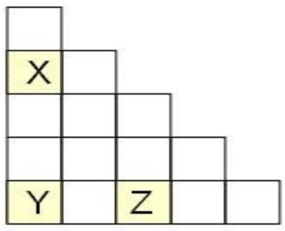
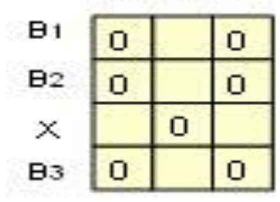


**Figure 3: The structure and three matrices for three clausules**

**3.1 Initialitation**

Every table is constructed crossing the literals, so in the cell *i*, *j* the *i*-th literal in first clausule will be crossed with the *j*-th literal in second clausule to apply this formula:

**a)** if both literals are exactly the same, the result is **forced 1 in *i, j***
**b)** if both literals are exactly the opposite, the result is **forced 0 in *i, j***
**c)** In the rest of the cases, pass.



| | A1 | ¬X | A3 |
|---|---|---|---|
| B1 | 0 | | 0 |
| B2 | 0 | | 0 |
| X | | 0 | |
| B3 | 0 | | 0 |

**Figure 4: Forced 0 in i, j**

So the algorithm **forced 1 in *i, j*** means that:
**1.** The cell *i*, *j* in this table must be 1 in the *initiation* step.
**2.** Every cell *k*, *j* and *i*, *h* where $k \neq i$ and $h \neq j$, must be 0.

And the algorithm **forced 0 in *i, j*** means that:
**1.** The cell *i*, *j* in this table must be 0 from *initiation* step.
**2.** Every cell *k*, *j* and *i*, *h* where $k \neq i$ and $h \neq j$, must be 1.

After executing those algorithms in every table, if any column or file in any table is full of 0's or in a step there was necesary to put 1 and 0 in the same cell, then the final result is **0 cases**. And the algorithm ends with all matrices nulled.

## 3.2 Scrape Out and Polish

In the next steps it is needed to operate with the matrices to make disapear incoherent cases. To get this we will reach three tables from three clausules and a matrix operation will be used. So, to update the **table (I, J)**, every clausule K is taken to catch **table (I, K)** and **table (J, K)**. After operating those matrices for every K, **table (I, J)** is updated, so it is needed to choose another pair (I, J). If we reach all tables in an order and, then, we repeat the same process in the inverted order, we will get the structure perfectly coherent with the formula.

The first operation of reaching all tables in an order is called **Scrape Out** and the second process is called to **Polish**. After Scraping Out we will get the result of satisfiability of the formula: if the last matrix updated is nulled (no ones in cells) then the formula is not satisfiable; elsewhere it is satisfiable.

The step of Polishing is decided to get every matrix (or interested) will be coherent with the formula: in the way if there is a one in a cell that will mean that there is almost one case satisfiable after putting literals pointed in their clausules as True.

All the steps commented will be explained slower in the next paragraphs.
Firstly, for every three tables defined by three clausules I, J, K with I < J < K their matrices are **table (I, J)**, **table (I, K)** and **table(J, K)** which are able to represent the coordinates (i, j, k) as only one possible case where in clausule I is activated the i-th literal, in clausule J is activated the j-th literal and in clausule K is activated the k-th literal. And considering that to activate a negated literal is to make the variable False and to activate a positive literal is to make it True, it is set that:

1) **table(I, J)** is called Top, and its cell is in column i, row j
2) **table(I, K)** is called Base, and its cell is in column i, row k
3) **table(J, K)** is called Edge, and its cell is in column j, row k

Cells are denoted by upcase letters A, B, C..., with the information of the clausules of the original formula and the value in its matrix, which will be in this document always a Boolean.
Now it is needed to define the impulse function δ **(3.1)**.

$$(x) = \left\{\begin{matrix} 0 if x = 0 \\ 1 if x \neq 0 \end{matrix}\right\} \tag{3.1}$$

From this point it is possible to understand the next definition, which is the most important to understand the reason everything works as is expected. It is defined a **Joined Correspondence** of cells A, B y C by the relationship R(A, B, C) under the showed condition **(3.2)**.

$$R(A, B, C) \text{ iff } \delta(A) + \delta(B) + \delta(C) \neq 2 \tag{3.2}$$

That points to the theorem **(3.3)** which ensures we can expect the condition above in every three coordinates.

For every ordered clausules $\mathbf{I}_1$, $\mathbf{I}_2$, ..., $\mathbf{I}_n$ in a choicing formula, if we choose three clausules and from every one literal *i*, *j*, *k* respectively, then their three cells of Coordinates (*i*, *j*, *k*) satisfy a Joined Correspondence (3.3)

From **(3.2)** to update tables is possible, that is because this Principle is Transitive; that is: if the Principle happens in R(A, B, C) and R(A, B, D) then is easy to demonstrate that R(B, C, D) will happen too.

After the update of all the tables the Phase of Scrape Out will be ended, and in the last matrix every Joined Correspondences of the formula will be reached. So in that matrix we will study if formula is satisfiable.
One operation of matrices to satisfy the **(3.2)** is denoted by δ (·*·) in **(3.4)**. Es decir, it will aply the AND operator for boolean matrices, coordinate to coordinate.
For X, Y, Z matrices,

$$X = \delta(Y * Z) \quad \text{iff} \quad X_{i,j} = \delta(Y_{i,j} \cdot Z_{i,j}) \tag{3.4}$$

Graphicaly, we can begin with clausules (I, J, K) to recognize matrices X, Y, Z in Top, Base and Edge with the matrices **table(I, J)**, **table(I, K)** y **table(J, K)**.

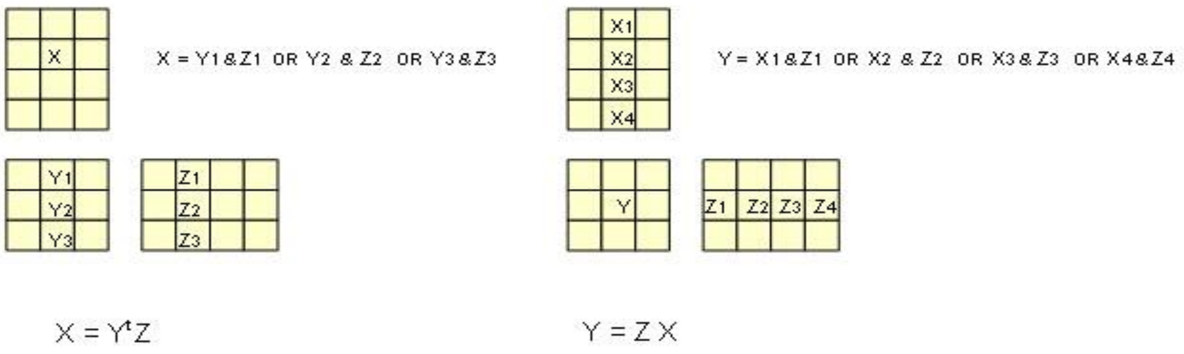

X
X = Y1&Z1 OR Y2 & Z2 OR Y3&Z3
Y1
Y2
Y3
Z1
Z2
Z3
X = Y^t Z
X1
X2
X3
X4
Y = X1&Z1 OR X2 & Z2 OR X3&Z3 OR X4&Z4
Y
Z1 Z2 Z3 Z4
Y = Z X

**Figure 5: Examples of the tables with a matrix operation**

Demonstration of the theorems is in the algebra beyond and in the real results in a coded class in Python. Considering the code of observations, it is easy to find an error if the code is wrong, so it must be considered with the fact this actually works.

The next three formulas in **(3.5)**, **(3.6)** and **(3.7)** are used for X, Y, Z matrices Top, Edge and Base to get the updated matrix.

$$X_{n+1} = \delta( X_n * (Z_n^t \cdot Y_n) ) \quad (3.5)$$

$$Yn+1 = \delta( Yn * (Zn \cdot Xn) ) \quad (3.6)$$

$$Zn+1 = \delta( Zn * (Yn \cdot Xnt) ) \quad (3.7)$$

So the number of steps after Scrape Out Phase is over $o(n^3)$ and under $O(n^3)$ where n is the number of clausules in the Choicing Formula. The workspace needed is under $O(n^2)$, so it is demonstrated those formulas finished in P.
In Polish Phase it is possible to update only n tables to get all the cases, so complexity in that phase will be the same or lower.

## 4. Conclusions

This document was presented to show how the language of informatics is not very well developed to offer good codes for an interesting kind of problems. Those problems are where explicit sets are used to model the questions, in special the Theory of Graphs.
This document was intended to get an approximation to the Hosoya Index, that is solving the sharp problem of matching a Graph, but at last it only could show all the cases represented one by one to get counted in slowly.
It could be considered a very strong and a good step to continue investigating.

## 5. References

[1]. J.M. Dato, Relationship between P and NP. The ultimate definition, *American Open Computer Science Journal.* **2015**, Vol 2, No. *1*.
[2]. Haruo Hosoya, Topological index. A newly proposed quantity characterizing the topological nature of structural somers of saturated hydrocarbons, *Bulletin of the Chemical Society of Japan* **1971**, *44* (9):2332-2339.