AN OPTIMIZATION OF THE PROCEDURE FOR THE CALCULATION OF HOSOYA'S INDEX

Michael I. TROFIMOV
Laboratory of Computer Chemistry, N.D. Zelinsky Institute of Organic Chemistry, USSR Academy of Sciences, Leninsky Prospect 47, Moscow 117913, USSR

Received 5 October 1990

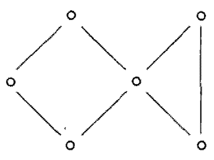
Abstract

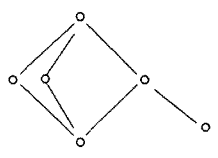
An optimized procedure for the calculation of Hosoya's topological index is presented. The procedure has been implemented in PASCAL. The described procedure could be simply modified for calculations of different modifications of Hosoya's index and Merrifield–Simmons' index. As an example, four useful modifications of Hosoya's index are defined.

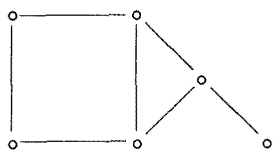
1. Introduction

The well-known topological index Z introduced by Hosoya in 1971 [1] is defined as the total number of edge-matchings of a graph plus 1.

Today, Z is a well-tried index and has considerable applications in chemistry [2,3]. Unfortunately Z, as well as other topological indices, is not free of shortcomings. For example, it was found [4] that Z loses discrimination applicability for the three cases shown below:


1


2


3

However, this defect can be eliminated by a proper modification of Z (see appendix). A grave limitation for practical usage of Z makes its calculation difficult: "A non-exponential algorithm for the exact calculation of Z is not known" [5]. In many cases, a simple effective algorithm for approximate calculation of Z [5] is a satisfactory solution. In other cases, such as described in the appendix, a modifications backtrack algorithm [6] should be applied.

In this research, an optimization of the implementation of this algorithm was made by using a comparative analysis of the data structures and different recursive and iterative versions.

It is interesting to note that in spite of the widely distributed bias against recursion [17], the recursive version seems to be the most effective one. The implementation and test were carried out on an IBM System 360 computer and a PASCAL 8000/2.0 compiler.

2. The data structure

The global type RIB_SET_TYP is the set of edges ("ribs") defined* as

SET OF RIB_INT_TYP;

where the range RIB_INT_TYP =

1 . . maximum_number_of_ribs_of_molecular_graph;

(it is necessary that the base type of this set is not more than the allowed range for using the compiler).

The global variables:

- List of disconnected edges is

  N_ADJ_RIB_LIST : **ARRAY** [RIB_INT_TYP] **OF** RIB_SET_TYP;

  Each element *i* of this array is a set of edges; the latter are not incident with edge *i*.
- Variables of the **INTEGER** type:

  RIB_NUMBER is the number of edges in the graph,

  COUNT is the counter of matchings.

3. The algorithm implementation

The program body:

Step 1: Assign the value 1 to the variable COUNT.

Step 2: Count the number of edges of the given graph and assign this value to the RIB_NUMBER variable.

Step 3: For each edge i ($i \in [1 \,..\, \text{RIB\_NUMBER}]$), enumerate all disconnected edges and load them into the N_ADJ_RIB_LIST array.
(Such an enumeration may be trivially organized by loop examination of the adjacency list of the graph. This step takes $\leq m^2$ time, where m is the number of edges in the graph.)

*We use the notation of the PASCAL programming language. The key words of this language are printed with **bold letters.**

Step 4: For all $j \in [1 \,..\, \mathrm{RIB\_NUMBER}]$, call procedure MAIN (see below) with the following actual value parameters:

- value j (current edge) will be passed to formal parameter i;
- $[j \,..\, \mathrm{RIB\_NUMBER}]$ (current set of edges) will be passed to S0.

Step 5: End of program.

Procedure MAIN:

Step 1: Each call of procedure MAIN exactly corresponds to one matching, so the counter of matchings COUNT every time must be incremented by 1.

Step 2: Form set S1 for a further search by including in this set all edges disconnected from edge i and consisting of the current set S0.

Step 3: If the following conditions

$$\begin{cases} S1 \neq \varnothing \\ S1 \neq S0 \\ \text{current edge } i \text{ is not the last} \end{cases}$$

are true, then for each edge j existing in S1 when $i > j$ (this condition eliminates repetitions) continue the search of matchings into set S1 by recursive calling of procedure MAIN.

Step 4: End of procedure MAIN.

The listing of the procedure is presented in table 1.

Table 1

Procedure MAIN

```
PROCEDURE MAIN (i : RIB_INT_TYP; S0 : RIB_SET_TYP);
 VAR
    j  : RIB_INT_TYP;
    S1 : RIB_SET_TYP;
 BEGIN
   S1 := N_ADJ_RIB_LIST [i] * S0;
   COUNT := COUNT + 1;
   IF (S1 <> [ ]) & (S1 <> S0) & (i < RIB_NUMBER)
    THEN
      FORALL j IN S1 DO
       IF j > i
         THEN MAIN (j, S1);
 END;  {________________________MAIN________}
```

4. Estimation of efficiency and conclusive remarks

The total number of callings is the total number of matchings, since each calling of procedure MAIN corresponds to one matching. The latter is estimated as an exponential function of the number of edges m [5]. So, this algorithm uses time:

$$O[a\exp(bm)].$$

For a series of different graphs containing 9–30 edges, measuring of the program was accomplished. This program had compiled from a source text without tests of the Run-Time system of the PASCAL compiler (V-, T-, P-options). As a result of this experiment, the following empirical relation for the estimation of the time of calculation was obtained*:

$$t = 3.2\exp(0.4m) \times 10^{-3}\ \text{sec.}$$

Results of the research of optimization of the reviewed algorithm indicated that the algorithm is a critical one for the choice of the kind of implementation; thus, the differences in speed of calculation for different versions reached about 20%.

For comparison, some programs following the directions of [8] were implemented. However, all of them indicated worse results (about 40% slower). Unfortunately, the **FORALL** statement used in procedure MAIN (table 1) is not defined for the PASCAL standard. Therefore, for other compilers, e.g. Turbo PASCAL (Borland Int.) of the popular IBM personal computer, this statement must be changed as follows:

```
FOR j := i TO RIB_NUMBER DO
    IF j IN S1 THEN
```

The maximum number of elements in a set permissible for different compilers is also different. In this case, the efficiency of implementation of a set by the compiler will be a key influence upon the efficiency of the program [9]. However, at any rate one should proceed from the following: the test on set membership is faster than the calculation of the equivalent Boolean expression.

It is evident that the procedure given here may be implemented for the calculation of modifications of Hosoya's index as well as for other popular topological indices, such as that of Merrifield–Simmons [10].

*For ES 1022 (the old Russian version of the IBM 360 computer).

Appendix [11]

DEFINITION 1

Let R_i be an index of edge i of a graph. Each R_i is equal to zero at the beginning. Let us generate all matchings $\mathcal{M}_j$ (where j is the number of generation: $j = 1, 2, \ldots$) of edges of the graph. Also, if $\mathcal{M}_j$ is a new matching and $i \in \mathcal{M}_j$, then we increment R_i by 1 ($\mathcal{M}_j$ is new if $j = 1$ or $\mathcal{M}_j \neq \mathcal{M}_1, \ldots, \mathcal{M}_j \neq \mathcal{M}_{j-1}$).

DEFINITION 2

Define the new local topological index T_i of vertex i as follows:

$$T_i = \sum_j \frac{1}{R_j},$$

where j's are the edges incident to vertex i.

DEFINITION 3

The new total index X is given by

$$X = \sum_{j=1}^{m} \frac{1}{R_j},$$

where m is the number of edges in the graph.

DEFINITION 4

Define the new total index Y as a real value, whose integer part is Z and whose fractions part is $X/10^{n+1}$, where n is the order of magnitude of X.

Whereas graphs **1–3** have the same $Z = 20$, the indices X and Y for them are different ($X = 1.6428, 1.6095, 1.6261$, $Y = 20.16428, 20.16095, 20.16261$) and thus have discrimination applicability. Apparently, other analogous modifications of Z are possible*.

Acknowledgements

The author would like to thank Academician of AS USSR Nikolai S. Zefirov for encouragement and for stimulating this research, and Serge S. Tratch for amiably placing his implementation of the FORTRAN program for Hosoya's index calculation at the author's disposal.

*For example, R_i is the Z value for the subgraph of graph G obtained by deleting edge i together with all the adjacent edges.

References

[1] H. Hosoya, Bull. Chem. Soc. Japan 44(1971)2332.

[2] A.T. Balaban, I. Motoc, D. Bonchev and O. Mekenyan, Top. Curr. Chem. 114(1983)38.

[3] I. Gutman and O. Polanski, *Mathematical Concepts in Organic Chemistry* (Akademie-Verlag, Berlin, 1987).

[4] D. Bonchev, O. Mekenyan and N. Trinajstić, J. Comp. Chem. 2(1981)127.

[5] I. Gutman, Z. Marković and S. Marković, Chem. Phys. Lett. 134(1987)139.

[6] S.S. Tratch and N.S. Zefirov, *Proc. 8th USSR Conf. on Application of Computers in Molecular Spectroscopy and Chemical Studies* (Novosibirsk, 1989), p. 210.

[7] M.I. Trofimov, J. Pascal, Ada & Modula-2, 9, 3 (1990)6.

[8] H. Hosoya, J. Chem. Doc. 12(1972)181.

[9] M.I. Trofimov, *PASCAL vs. PASCAL: the Slow Sets* (in press).

[10] R.E. Merrifield and H.E. Simmons, Theor. Chim. Acta 55(1980)55.

[11] M. Trofimov, *Proc. 6th USSR Conf. on Mathematical Methods in Chemistry* (Novocherkassk, 1989), p. 32.