General Method for the Computation of Matching Polynomials of Graphs

M. M. Balakrishnarajan and P. Venuvanalingam*

Department of Chemistry, Bharathidasan University, Tiruchirappalli 620 024, India

Received January 5, 1993®

A novel algorithm based on search is developed for computation of matching polynomials of graphs, viewing the edges as ordered pairs. The acyclic Sach's subgraphs of matching polynomials are generated by disjoint-set-union operation. The algorithm is recursive and very general and hence is more compact and has a wide range of applications. The present approach uses artificial intelligence techinques tactfully for partially offsetting combinatorial explosion. This algorithm is more advantageous in terms of space complexity, and its time complexity is exponential as would normally be the case due to nondeterministic polynomial (NP) complete nature of the problem.

INTRODUCTION

Graph theory and combinatorics have become effective tools in chemistry and chemical physics, and the use of mathematical techniques to handle chemical problems has been the subject of numerous investigations in recent years.[1-4] One of the ways in which chemical information is derived is through graph theoretical invariants, by correlating them directly with physicochemical properties of systems. Hence, there is a constant search for apt graph theoretical invariants, and this has resulted in a sudden proliferation of various topological invariants, namely, indices and graph polynomials.[5-8] Computation of graph polynomials has been attracting a lot of interest, of late, and newer computer algorithms have been designed in which time and space requirements are optimized. Recently there have been reports on the vectorization of codes, taking full advantage of the present day technology, leading to faster working programs.[9,10] In this direction, we make a new attempt of using artificial intelligence (AI) techniques toward computing graph polynomials. We take up, in the present work, computation of matching polynomials because there is so far no general algorithm for computing these. The present approach can be extended to other graph invariants.[11]

Matching polynomial $M(G;x)$, which is also referred to as acyclic or reference polynomial, is defined as

$$M(G;x) = \sum_{k=1}^{N/2} (-1)^k p(G,k) x^{N-2k}$$

where N is the number of vertices of the graph G, k denotes the number of disjoint K_2 (edges) taken at a time, and $p(G;k)$ is the number of ways of placing k numbers of K_2 graphs on G.[12-16] Matching polynomials find interesting applications in various disciplines.[2] Topological resonance energy (TRE) of conjugated molecules, which is used for predicting aromaticity, could be obtained by using matching polynomials.[2] The evaluation of a grand canonical partition function of a lattice gas through exact lattice statistics methods and the partition function of a system of interacting ferromagnets (Ising problem) could be obtained from matching polynomials. Matching polynomials have also found applications in thermodynamics of adsorption of diatomics on metal surfaces, in chemical documentation, and in estimating several physicochemical properties of chemical compounds. Because of its vast application potential,[17-24] there is considerable interest in computing matching polynomials. Very recently, Balasubramanian had computed matching polynomials of fullerene cages.[25] Randić and co-workers have well documented the earlier work done on matching polynomial in their recent paper.[24] A cursory glance at the literature reveals the fact that only a few algorithms are reported,[26-28] and still a direct method for the computation of matching polynomials for an arbitrary graph is not available. It is interesting to note that, characteristic polynomials of graphs, which are closely related to matching polynomials, pose no serious difficulty for computation.[29-31] This is due to the reason that characteristic polynomials arise as an intermediate step in a matrix eigenvalue problem, and hence the existing linear algebraic procedures are well adapted for use in generating this. But, computation of matching polynomials is not that straightforward for nontree graphs as they cannot be fit into matrix operationalization directly. In the present work we develop an algorithm for computing the matching polynomial, which is direct and applicable to any arbitrary graph. Before describing our algorithm, a brief discussion on the existing methods of computation of matching polynomials is made in chronological order.

Matching polynomials can be hand-computed for any graph, but the process becomes laborious even for medium-sized graphs. Hosaya and Okhami[32] proposed a recurrence operator method which is applicable only to special polyhex graphs. Gutman and Hosaya[17] have derived analytical expressions to matching polynomials of special graphs such as complete and complete-bipartite graphs and circuit graphs. Mohar and Trinajstić[26] have developed an algorithm to compute matching polynomials of any arbitrary graph; this involves recurrence reduction of a graph into chains, and matching polynomials are computed from the characteristic polynomial of the resulting chains. Obviously, this method involves large-scale pruning.

Several attempts have been made to modify the problem of computation of matching polynomials, into a problem of matrix algebra. Ramaraj and Balasubramanian[27] have developed a program which also depends on recursive reduction of a given graph G as outlined by Gutman and Hosaya.[17] In this algorithm, the given graph is reduced into trees, and characteristic polynomials of these trees are computed using the Le verrier-Fadeev-Frame method.[30] Then, from the characteristic polynomials of these trees matching polynomial of the main graph is determined. This method differs from the earlier one[26] in that this stops the reduction at tree stage and uses the Le verrier-Fadeev-Frame method for computing characteristic polynomials of trees. Therefore, this method, though general, suffers from the same limitation but to a

* Author to whom correspondence should be addressed.

® Abstract published in *Advance ACS Abstracts*, May 15, 1994.

0095-2338/94/1634-1122$04.50/0

lesser degree. For highly complicated clustered graphs and for those containing many cycles, this procedure will be time consuming.

Hosaya and Balasubramanian[28] have proposed an algorithm for the computation of matching polynomials for special classes of molecules, like polycyclic, spiro, and bridged molecules. This procedure involves three steps. At first, it generates, for a given graph, a set of edge-weighted, directed graphs by a special procedure, and in the second step, it uses the Le verrier–Fadeev–Frame method to compute characteristic polynomials of these edge-weighted graphs. Finally, it computes the matching polynomial of the main graph from characteristic polynomials of the edge-weighted graphs thus obtained. This method, though efficient with respect to time, is rather limited, as the authors themselves have expressed, in that there is no single algorithm for obtaining the set of "required" edge-weighted graphs for a general graph. An algebraic method, called the "transfer matrix method" is used by several authors for computing matching polynomials of graphs with periodic properties.[33-35] Very recently, Randić and co-workers[24] have reported an algorithm for obtaining matching polynomials of an arbitrary, cata-condensed unbranched benzenoid molecule which uses this transfer matrix method. In the present work we present an algorithm which computes the coefficients of matching polynomials through direct enumeration of disjoint $p(G;k)$ members of a graph using search. Our algorithm is adaptable to any arbitrary graph, and in this respect, it differs from the existing algorithms.

COMPUTATION OF MATCHING POLYNOMIAL

The entire algorithm is based on search, which is an artificial intelligence technique and is normally resorted to when no direct methods are available.[36] Search provides the framework into which heuristic techniques and generalized knowledge about the system can be embedded to overcome the combinatorial explosion. In order to generate intelligent action, a *physical symbol system*[37] is defined. This consists of a set of entities called *symbols* (edges) which are physical patterns that can occur as components of an *expression* (symbol structure) which is the required acyclic Sach's subgraph. This symbol structure is composed of a number of instances of symbols related by the property of disjointness. At any instant of time, the system will contain a collection of these symbol structures and produces through time an evolving collection of symbol structures. Besides these structures, the system also contains a collection of processes and rules that operate on expressions and symbols to generate, to reproduce, and finally to destroy other expressions. During this process, the expressions corresponding to various lengths (subgraphs) are enumerated. Though the computation of a matching polynomial is apparently a non-AI problem, AI techniques are successfully utilized here to design an efficient method.

ALGORITHM DESCRIPTION

$G(V,E)$ denotes a finite graphs with no loop and multiple edges, where V represents the set of all vertices and E represents the set of all edges. If $x,y \in V$, we say that x adj y is true, iff there exists an edge between x and y.[38] The connectivity information of G is given by labeling all the vertices in V by positive integers and listing its edges. The logical data base required for the *inference engine*[36] is constructed during the process of operationalization. This is done by viewing the edges as a set of ordered pairs. Here an ordered pair $\langle x,y \rangle$ is an entity uniquely determined by two vertices x and y in a specified order. The set theoretic representation of $\langle x,y \rangle$ is

$$\{\{x\}, \{x,y\}\}$$

i.e., the two member set of which member $\{x,y\}$ is the unordered pair involved and the other $\{x\}$ represents the first coordinate.[38] "y" is referred to as the *accomplice* of x. We define a binary relation ρ as

$$\{\langle x,y \rangle | x,y \in V, (x \text{ adj } y) \wedge (x < y)\}$$

Thus, a stage is set for viewing the graph as a relation defined by the set of ordered pairs. It is to be noted, that ρ is defined for unique representation of edges and nothing connected to directed graphs. The domain of ρ, symbolized by D_ρ is

$$D_\rho = \{x \in V | \exists y \in V, \langle x,y \rangle \in \rho\}$$

and a range of ρ, R_ρ, is

$$R_\rho = \{y \in V | \exists x \in V, \langle x,y \rangle \in \rho\}$$

Thus a systematic database comprising a set of ordered pairs is constructed.

In order to facilitate easier access to the database, further structuralization is done by introducing another relation ζ, which divides ρ into disjoint subsets. It is defined as the set of ordered pairs having the identical first coordinate. Since ζ is symmetric, reflexive, and transitive, it is an equivalence relation and hence divides ρ into m equivalence classes (partitions), where m is the cardinality of D_ρ. The ith partition of k by ζ is denoted as ζ_i (k modulo ζ_i) and is called the quotient set of ρ by ζ. It is given as

$$\zeta_i = \{\langle x,y \rangle | \quad \langle x,y \rangle \in \rho, \in i \in D_\rho, x = i\}$$

Now the subgraphs of matching polynomials are generated by performing disjoing-set-union[40] between the set of unordered vertices which are members of $\langle x,y \rangle$ lying in different partitions.

The production rule for generating subgraphs is

$$(S \cap \{x,y\} = \varnothing) \rightarrow S = S \bigcup \{x,y\}$$

where the set S comprises vertices that are involved in the subgraph. S is initialized to null (m). The left-hand side of the above rule determines the applicability of the rule, and the right-hand side describes the operation to be performed if the rule is applied. For a subgraph S, subgraph S' is a prefix of S if there is an edge $\{x,y\}$ such that

$$(S' \cap \{x,y\} = \varnothing) \wedge (S = S' \bigcup \{x,y]\})$$

Here S is called a suffix to S'. The entire set of subgraphs of the matching polynomial falls in a tree domain D[41] (Figure 1), which is a nonempty subset of subgraphs satisfying the following conditions.

$$\forall S \in D, \quad S' \in D$$

$$\forall S,S' \in D, \quad (S \cap S') = \varnothing \rightarrow (S \cap S') \in D$$

The subgraphs of D are called nodes of the tree. The ramification $\deg(S)$ of a node S is the cardinality of the set

$$\{\langle x,y \rangle \in \rho | S \cap \{x,y\} = \varnothing\}$$

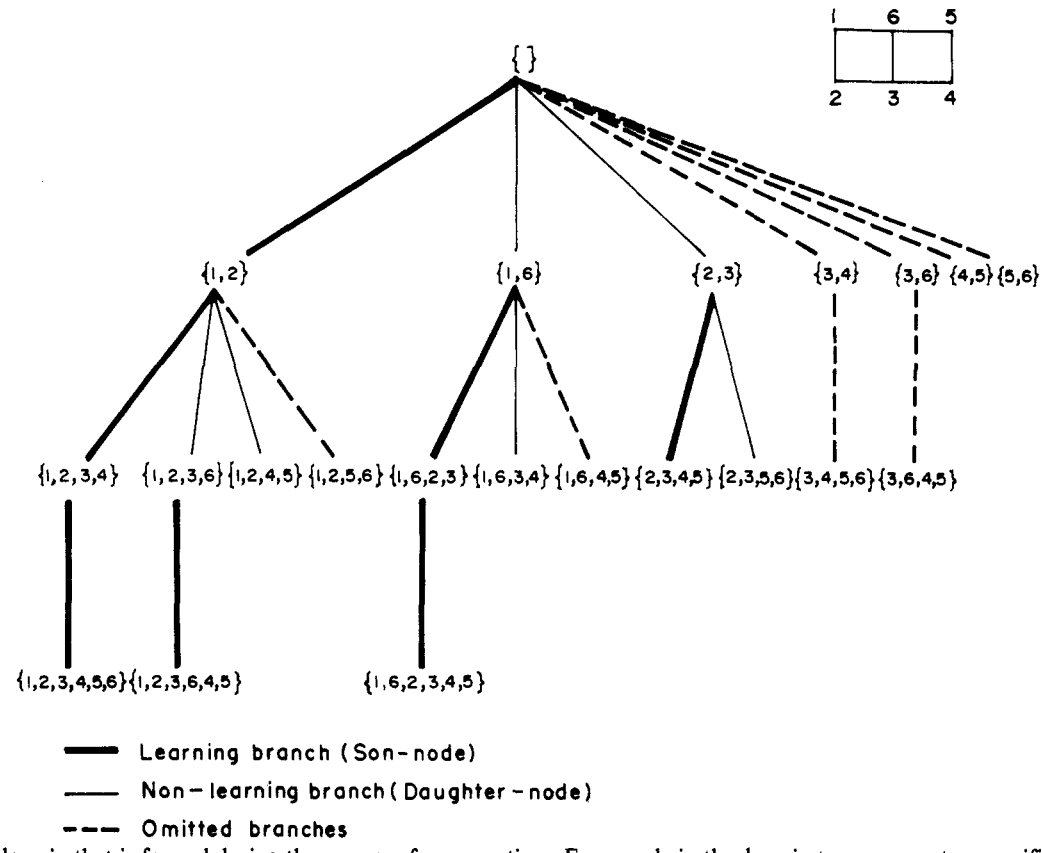

Figure 1. Tree domain that is formed during the process of enumeration. Every node in the domain tree represents a specific disjoint edge set corresponding to an acrylic Sach's subgraph S_n. The number of nodes at a specified depth k is $p(G;k)$.

A node of ramification 0 is called as leaf. A path with source u and target v is a sequence of nodes $S_0, S_1, S_2, ..., S_n$, such that $S_0 = u$ and $S_n = v$, and for all j, $1 \leq j \leq n$, $S_j = S_{j-1} \cup i_j$ for some $i_j \in \rho$. A branch is a path from the root to a leaf. The depth of a node S in D is equal to the number of nodes (length) of the path from the root to S. If the depth of a leaf is $N/2$, then it represents a Kékule structure of the graph.

In order to compute the matching polynomial, one has to traverse the whole tree described above to count the number of nodes at various depths. Their number directly yields the $p(G;k)$ numbers. The matching polynomial and Z-counting polynomials can thus be computed. The search strategy chosen is depth-first in order to minimize the memory space requirements.[40,42,43] Generation of the tree and enumeration of its nodes are done simultaneously.

The set type variable *GRAPH* contains the vertices that are involved in the subgraph. Initially GRAPH is empty. Then, an edge ⟨*position, accomplice*⟩, obeying the left-hand side of the production rule, is selected from position. Then, the process referred in the right-hand side of the production rule is performed to get a *son-node*. The corresponding coefficient of the matching polynomial is increased by one. Then, the procedure is called recursively until GRAPH becomes a leaf. After the recursive call is returned, position and accomplice are removed from GRAPH. From the rest of E, edges containing either position or accomplice which satisfy the right-hand side of the production rule are selected. The above procedure is repeated for all of these edges. These nodes are termed *daughter–nodes*. Hence, at any instant of time the system will be comprised of a node GRAPH corresponding to some depth. The branches corresponding to the rest of the disjoint edges (i.e. edges that are disjoint to GRAPH but not containing position or accomplice) have the same operator sequence with the branches descending from the *son-node*. Hence, a learning procedure which chuncks the number of subgraphs arising from the branches of the *son-node* are developed and are added. Thus, branches descending from edges other than those containing position, accomplice are completely omitted. Hence, traversing a branch, which has a similar operating sequence with some branches traversed earlier, never occurs, thereby decreasing the time complexity of the system. This type of learning is popularly known as *rote learning*.[44]

The algorithm for the above method is listed in Table 1. It can be easily understood by the search tree of a sample graph given in Figure 1. Here, the thick lines represent learning paths leading to son-nodes, and dotted lines represent the paths that are avoided by using the knowledge learned earlier. Normal lines lead to daughter-nodes. The search starts by first choosing edge ⟨1,2⟩. The knowledge gained by this *son-node* is utilized to avoid traversing the paths passing through ⟨3,4⟩, ⟨3,6⟩, ⟨4,5⟩, and ⟨5,6⟩. Similarly the knowledge gained by the *son-node* ⟨1,2,3,4⟩ is used for avoiding the path leading to ⟨1,2,5,6⟩. The significance of this learning is that the number of omitted paths grows exponentially with the complexity of the graph. Hence, it effectively retards the rate of combinatorial growth. Such learning is desirable as the time complexity of this problem is exponential.

Since there are no restrictions in the algorithms as to the number of vertices of a graph, the algorithm is applicable for any arbitrary graph with any number of vertices. The time factor for larger graphs can be reduced by introducing heuristics, which exploits domain-specific knowledge, to improve the efficiency of the search process. The use of heuristics can be well explained for the case of complete or complete-bipartite graphs. For such a graph G, the edges in the same partition are equivalent in the sense that the corresponding subgraphs $(G–e)$ are all isomorphic. Hence it is sufficient to know the RESULT of a single edge, which is

Table 1. Algorithm for the Matching Polynomials of Graphs

```
PROCEDURE TRAVERSE(     GRAPH   : NODE
                      POSITION  : POINTER of PARTITIONS
                      VAR DEPTH : DEPTH of the NODE
                      VAR RESULT : ARRAY of  p(G;k) numbers)
VAR
   I,J              : COUNTERS
   RESULT-SON,RESULT-DAUGHTER, RESULT-LEARN : same type as RESULT
   LEAF             : BOOLEAN

   BEGIN
     Initialize  DEPTH, RESULT-SON,
     RESULT-DAUGHTER,RESULT-LEARN,RESULT  to  0
     IF EXISTS J ∈ D_ρ   SUCH THAT NOT(J IN GRAPH) AND
        <J,I> ∈ ζ_position , NOT(I IN GRAPH) THEN
        BEGIN
          LEAF = FALSE
          ACCOMPLICE = J
          POSITION = J
        ENDIF;
     IF NOT(LEAF) THEN
       BEGIN
         SON-NODE = GRAPH ⋃ { POSITION, ACCOMPLICE }
         TRAVERSE(SON-NODE, Next POSITION, DEPTH-1 , RESULT-SON)
         Chunk Information regarding the Number Of Subgraphs
         In  SON-NODE to  RESULT  LEARN
         FOR ALL II ∈ D_ρ    SUCH THAT II >= POSITION DO
           BEGIN
             FOR ALL <II,JJ> ∈ ζ_II   SUCH THAT
            ((II =   ACCOMPLICE   OR    II    =POSITION)
              XOR (JJ = POSITION OR JJ = ACCOMPLICE))
              AND ({ II, JJ } ∩ GRAPH) = Ø DO
               BEGIN
                 DAUGHTER-NODE  =  GRAPH ⋃ { I, J }
                   TRAVERSE(DAUGHTER-NODE, Next POSITION, DEPTH-2,
                            RESULT-DAUGHTER)
               ENDFOR
           ENDFOR
       ENDIF;
      RESULT = RESULT_SON + RESULT_DAUGHTER + RESULT_LEARN
END
```

the same for all edges in the partition. This heuristic, when introduced, drastically reduces the CPU time requirements, as it avoids traversing several paths in the search tree. Similar heuristic functions can be designed for different systems, but it needs in-depth inquiry into the nature of its tree domain.

The search time can also be reduced by selective creation and utilization of useful knowledge. For graphs containing many even-membered rings, the time complexity can be improved by creating knowledge bases containing the nodes of a particular depth and their RESULT, since the same node may be generated again. For example, in Figure 1, the node [1,2,3,6] is generated twice, for the first time by the combination ⟨1,2⟩ and ⟨3,6⟩ and then as ⟨1,6⟩ and ⟨2,3⟩. This occurs because of the four-membered ring present in the graph. In such cases of multiple occurrences, the RESULT saved for the first time can be used later to avoid certain paths in the search tree. This technique actually reduces the time complexity but at the expense of memory space.

The very obvious advantage of the above algorithm is that it forms a skeleton for search in which domain-specific information can be incorporated. It is very generalized so that applicability is not restricted to any special graph. It is this latter aspect which gives a leverage, in the range of applications, over the existing algorithms. Another important aspect of this algorithm is its dynamic learning nature during enumeration. This AI technique partially offsets combina-

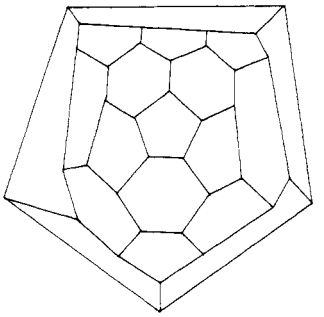

Figure 2. Schelegal diagram for a C_{36} fullerene.

Table 2. Matching Polynomial of C_{36} Fullerene Cage

k	coefficient	k	coefficient
[1]	1	[21]	203 942 024
[3]	−53	[23]	−228 641 142
[5]	1 274	[25]	181 920 513
[7]	−18 394	[27]	−98 879 291
[9]	178 075	[29]	34 680 820
[11]	−1 222 197	[31]	−7 183 884
[13]	6 131 604	[33]	756 374
[15]	−22 852 106	[35]	−30 186
[17]	63 674 647	[37]	204
[19]	−132 457 899		

torial explosion tactfully. The algorithm is recursive, making it more compact and readable. The distinguishing feature of the present algorithm is that it can be altered to provide a pictorial mapping of various subgraphs contributing to the matching polynomial if desired. The only disadvantage is its time complexity. For a graph with N vertices and ramification D, the number of nodes required to be traversed is $D^{N/2}$ for the worst case. This can be easily proven by induction. Here, ramification is the average cardinalities of ζ. Since the problem of computing the matching polynomial itself falls in NP class,[40] it is justifiable that *a priori* time complexity[45] is exponential. In terms of memory, our algorithm was found to be very effective. Our algorithm has been implemented in Pascal and is found to be very efficient in computing matching polynomials for chemical graphs, even in a personal computer without any additional hardware supports like expanded memory, coprocessor, etc. For instance, the CPU time required for computing the matching polynomial of a hypothetical C_{36} fullerene, shown in Figure 2, is about 15 min on a PC-2AT6 with a clock speed of 16 MHz.

The reported CPU time of the earlier code[27] for a fullerene of the same complexity is reported to be nearly the same on a comparatively powerful IBM-3090 system.[25] The matching polynomial of this fullerene (Figure 2) is listed in Table 2. This search skeleton can also be effectively modified to compute other related invariants.[11]

ACKNOWLEDGMENT

The financial assistance from UGC, New Delhi, in the form of JRF to M.M.B is gratefully acknowledged.

REFERENCES AND NOTES

(1) Balaban, A. T. *Chemical Applications of Graph Theory*; Academic Press: New York, 1976.
(2) Trinajstić, N. *Chemical Graph Theory*; CRC Press: Boca Raton, FL, 1983; Vols I & II.
(3) King, R. B., Ed. *Chemical Applications of Topology and Graph Theory*; Elsevier: Amsterdam, 1983.
(4) Balasubramanian, K. *Chem. Rev.* **1985**, *85*, 599.
(5) Knop, J. V.; Trinajstić, N. *Int. J. Quantum Chem. Symp.* **1980**, *14*, 503.
(6) Barysz, M.; Plavšić, D.; Trinajstić, N. *MATCH* **1986**, *19*, 89.
(7) Balaban, A. T.; Motoc, I.; Bonchev, D.; Mekenyan, O. *Topics Curr. Chem.* **1983**, *114*, 21.
(8) Rouvray, D. H. *J. Comput. Chem.* **1987**, *8*, 470.
(9) Balasubramanian, K. *J. Comput. Chem.* **1991**, *12*, 248.
(10) Venuvanalingam, P.; Thangavel, P. *J. Comput. Chem.* **1991**, *12*, 779.
(11) Balakrishnarajan, M. M.; Venuvanalingam, P. Unpublished results.
(12) Hosaya, H. *Bull. Chem. Soc. Jpn.* **1971**, *44*, 2332.
(13) Gutman, I.; Milun, M.; Trinajstić, N. *MATCH* **1975**, *1*, 171.
(14) Aihara, J. *J. Am. Chem. Soc.* **1976**, *97*, 2750.
(15) Gutman, I.; Milun, M.; Trinajstić, N. *J. Am. Chem. Soc.* **1977**, *98*, 1692.
(16) Farrell, E. J. *J. Combinatorial Theory (B)* **1979**, *27*, 75.
(17) Gutman, I.; Hosaya, H. *Theor. Chim. Acta* **1978**, *40*, 279.
(18) Gutman, I. *MATCH* **1979**, *6*, 75.
(19) Gutman, I. *J. Chem. Soc., Faraday Trans. 2* **1983**, *79*, 33.
(20) Godsil, C. D.; Gutman, I. *Croat. Chem. Acta* **1981**, *54*, 53.
(21) Groavac, A.; Polansky, O. E. *MATCH* **1987**, *21*, 33.
(22) Groavac, A.; Polansky, O. E. *MATCH* **1987**, *21*, 47.
(23) Groavac, A.; Polansky, O. E. *MATCH* **1987**, *21*, 81.
(24) Randić, M.; Hosaya, H.; Polansky, O. E. *J. Comput. Chem.* **1989**, *10*, 683.
(25) Balasubramanian, K. *Chem. Phys. Lett.* **1993**, *201*, 306.
(26) Mohar, B.; Trinajstić, N. *J. Comput. Chem.* **1982**, *3*, 28.
(27) Ramaraj, R.; Balasubramanian, K. *J. Comput. Chem.* **1985**, *6*, 122.
(28) Hosaya, H.; Balasubramanian, K. *J. Comput. Chem.* **1989**, *10*, 698.
(29) Trinajstić, N.; Klein, D. J.; Randić, M. *Int. J. Quantum Chem. Symp.* **1986**, *20*, 699.
(30) Balasubramanian, K. *J. Comput. Chem.* **1984**, *5*, 387.
(31) Křivka, P.; Jeričević, Ž.; Trinajstić, N. *Int. J. Quantum Chem. Symp.* **1986**, *19*, 129.
(32) Hosaya, H.; Ohkami, N. *J. Comput. Chem.* **1983**, *4*, 585.
(33) Lieb, E. H. *J. Math. Phys.* **1967**, *8*, 2339.
(34) Graovac, A.; Polansky, O. E.; Tyutyulkov, N. N. *Croat. Chem. Acta* **1983**, *56*, 352.
(35) Gutman, I.; Farrell, E. J.; Wahid, S. A. *J. Comb. Inf. Syst. Sci.* **1983**, *8*, 159.
(36) Rich, E.; Knight, K. *Artificial Intelligence*; McGraw-Hill: New York, 1991.
(37) Newell, A.; Simon, H. A. *Commun. ACM* **1976**, *19*, 113.
(38) Harary, F. *Graph Theory*; Addison-Wesley: New York, 1969; p 9.
(39) Stoll, R. R. *Set Theory and Logic*; American Book Co.: New York, 1967; p 24.
(40) Aho, A. V.; Hopcroft, J. E.; Ullman, J. D. *Data Structures and Algorithms*; Addison-Wesley: New York, 1985.
(41) Gorn, S. *Systems and Computer Science*; University of Toronto Press: Toronto, Canada, 1965.
(42) Hopcroft, J.; Tarjan, R. *Commun. ACM* **1973**, *16*, 372.
(43) Tarjan, R. *SIAM J. Comput.* **1972**, *1*, 146.
(44) Samuel, A. L. In *Computers and Thought*; Feigenbaum, E. A., Feldman, J., Eds.; McGraw-Hill: New York, 1963; p 144.
(45) Wah, B. W.; Ramamoorthy, C. V. In *Handbook of Software Engineering*: Vick, C. R., Ramamoorthy, C. V., Eds.; Van Nostrand Reinhold Co.: New York, 1986; p 24.