Cyclical Conjunction: An Efficient Operator for the Extraction of Cycles from a Graph

Gonzalo Cerruela García, Irene Luque Ruiz,* and Miguel Angel Gómez-Nieto

Department of Computing and Numerical Analysis, University of Córdoba, Campus Universitario de Rabanales, Edificio C2, Planta-3, E-14071 Córdoba, Spain

Received April 30, 2002

The structural characteristics of a molecule, namely size, bond type and number, cycles, shape, and functional groups, will largely determine its physicochemical properties and biological activity. Extraction of data such as the complete structural information of a molecule's ring system is complex (NP-complete) and has traditionally involved high computational cost. The present study proposes a new operator for the extraction of cycles from a graph. Based on an initial cycle set, the operator employs a reduced number of operations in an iterative process of error-free cycle extraction, hence greatly reducing computational cost. Algorithm efficiency has been enhanced by designing new data structures suited to cycle storage, useful not only for interactive solutions but also for applications managing large volumes of information such as descriptor calculation, QSPR/QSAR, matching, clustering, screening, and filtering. Validation was performed by applying the algorithm to a test suite of chemical compounds of varying complexity.

1. INTRODUCTION

Over the last few decades, many new descriptors or topological indices have been proposed for the quantification of the molecular structure of chemical compounds, in an attempt to produce a mathematical model that would represent the relationship between molecular structures and a great number of their physicochemical properties.[1]

Recent studies have pointed to the need for multivariant models, where several descriptors are employed to describe the properties of a compound; poor correlations have been obtained when the nature of the compound is highly varied or when the number of descriptors is low.[2−4]

While results may so far have been disappointing, the scientific community has expressed great interest in the following:

• New descriptors to allow the representation of the structural properties of diverse compound families.

• The application of existing descriptors, discarding those proving to be unsuitable. While the number of reported topological indices may be high, many of these are inter-related, meaning that rather than improve results, their combined use in a model may lead to a substantial increase in complexity and computational cost.

• The proposal of new algorithms to permit extraction of these descriptors at reasonable computational cost. Since certain descriptors would, however, be computationally costly to derive, they will simply not appear in solutions demanding low response times.

The extraction of topological descriptors for the characterization of ring or cycle systems is of major importance, since a compound's cycle system will define its shape, properties, and bioactivity, and such descriptors are widely used in measuring aspects such as complexity, similarity, and cyclicity in the synthesis of new products (e.g. drugs) as well as in the search for compounds in large databases.[5−7]

The present study proposes a novel algorithm for cycle extraction, with two main objectives:

• To provide a simple solution, in both conceptual and computing terms, for the extraction of full cycle system information at low computational cost.

• To employ suitable data structures, again at extremely low computational cost, for the extraction of other invariants or descriptors related to a ring system—as detailed in the bibliography—in addition to proposing new descriptors.

The paper begins with a brief overview of previously proposed algorithms, to continue in Section 3 with a description of the cyclical conjunction operator and its properties. Section 4 then presents an implementation of the algorithm along with experimental results, and finally there is discussion of results and of future studies.

2. BACKGROUND

The use of graphs is widespread for the abstract representation of information involved in a problem, which can then be solved by applying analysis techniques. Where the graph is intricate and contains cycles, extracting these cycles poses a complex problem normally requiring a great deal of computing power.

In chemistry, graphs are commonly used to represent the molecular structure of compounds. By analyzing such graphs, a huge amount of information can be obtained for use in areas such as compound and fragment identification, the calculation of topographic descriptors, complexity and similarity, the estimation of physical and chemical properties, and assistance in synthetic processes.

For over three decades, researchers have endeavored to find efficient methods and algorithms for the detection of cycles in a graph,[8−17] yet despite this effort, not many of the proposed algorithms are suitable for use in all types of graphs

* Corresponding author phone: +34-957-212082; fax: +34-957218630; e-mail: gcerruela@uco.es; ma1lurui@uco.es; mangel@uco.es.

and in most cases computational cost is prohibitive, especially as graph complexity rises.

Bearing this in mind, the authors present a new operator—cyclical conjunction—which is capable of detecting all the cycles present in a graph. Extraction is performed iteratively, starting from an initial set of cycles and consistently generating a valid set of cycles, at low computational cost.

To unify the different terminologies and concepts of graph theory used in the bibliography, the following notation has been adopted herein: N is the number of nodes or vertices in a graph; E is the number of edges in a graph; and η is the cyclomatic complexity, *Frerejacque* number or nullity, obtainable thus

$$\eta = N - E + 1 \qquad (1)$$

this enables us to predict that the number of cycles present in a graph will lie within the interval $[\eta, 2^{\eta}\text{-}1]$.

The algorithms described in the bibliography may be grouped under two fairly different objectives:

1. The extraction of a basic set of cycles by which the graph may be characterized. This approach has resulted in sets such as the *Smallest Set of Smallest Rings* (*SSSR*), *Essential Set of Essential Rings* (*ESER*), and *Set of Fundamental Bases*.

2. The detection of all cycles present in a graph, in some cases working from a basic set of cycles.

2.1. The Basic Set of Cycles in a Graph. Analysis of the literature reveals that depending on the end to which it will be put, the basic set of cycles may be defined in different ways.

2.1.1. Smallest Set of Smallest Rings (SSSR). The *SSSR* is a group of cycles that is presented as a solution for the search for a subset of cycles which will characterize the graph structure and allow the extraction of all cycles from a graph, at optimal performance. In certain complex graphs, however, where there are many symmetrically equivalent cycles, the *SSSR* is unsuitable, since it is not unique.

Plotking[16] based the *SSSR* on all the cycles of a graph with fewer than nine nodes, which he called $\mathscr{R}$*-rings*. Bersohn[17] described an *SSSR* formed by cycles of up to six nodes using a path-tracing algorithm, for which enhancements have been proposed.[18,19] In these algorithms, as in others[20] of the same type, computational cost is dependent on the number of cycles found, and they are not efficient for all graphs.

Elsewhere,[21,22] the cyclical structure of a graph is converted into a spanning tree; path tracing is successively applied to the different spanning trees generated by treating each node of the graph as a root, but this incurs high computational costs.

Fan[23] developed a method employing a connection table to walk through the graph seeking cycles with nodes of connectivity 2, subsequently collapsing the graph. The same author claims that the collapsing process is critical given that the removal of a connectivity 2 node could produce a different, incorrect *SSSR*. To this end, another function is introduced to determine which node of the cycle may be eliminated. These shortcomings were overcome by two enhancements introduced by Figueras.[24]

2.1.2. Essential Set of Essential Rings (ESER). The *SSSR* has the disadvantage that in many cases it is not unique. To overcome this, Fujita[25] proposed an extension of the *SSSR* concept, the *Essential Set of Essential Rings* (*ESER*). The cycles contained in a structure are classified as essential and nonessential. Nonessential cycles are defined by three categories: *Tied-Rings*, *Multi-tied-Rings*, and *Dependent Rings*.

An *ESER* will be formed by any cycle which is not *tied*, *multitied*, or *dependent* and will always contain the elements of an *SSSR* along with, in some instances, a set of added cycles.

2.1.3. Set of Fundamental Bases. The set of fundamental bases generates a set of cycles which is not unique to each graph and is used mainly to calculate other cycle sets.

Welch[26] used an incidence matrix whose columns are reordered and split, thus generating a spanning tree; by including new edges the cycles forming the fundamental set are obtained. Gotlieb,[27] and later Gibbs,[28] proposed modifications to this algorithm which lowered memory requirements but resulted in a performance hit, although this was remedied to some extent by the algorithm presented by Paton[29] and by Wipke.[30]

Gasteiger[22] calculated the fundamental set of cycles in order to find the *SSSR*. To do so, they used a spanning tree built from an initial, arbitrarily selected atom, storing and ordering the removed edges according to their distance from the starting node; a path-tracing technique iteratively discovers the cycles and terminates on reaching a limit referred to as ring system complexity, for whose calculation a formula was described. Sorkau[31] proposed certain refinements for the above algorithm, such as collapsing the graph and renaming the vertices to facilitate the finding of the shortest path between the root node and its neighbors.

2.2. The Set of All Cycles in a Graph. In addition to calculating the fundamental bases, Welch[26] combined these to produce new cycles using a *summation* operation. At this stage, any union of disjoint cycles is ignored meaning that numerous cycles are lost. Gibbs[28] modified the last stage to find all of the cycles, separating unions from disjoint cycles. Although this increases the processing time of the algorithm it does, however, result in correct determination of all cycles.

Tiernan[32] presented an algorithm that uses directed instead of undirected graph to represent molecular structure. The directional nature of a graph makes path tracing easier to implement, calculating all cycles at the same time. The conversion of this algorithm to use undirected graphs makes it less efficient than others.

Beiss[33] also used path tracing to search directed graphs. Numerous studies have been made of the application of directed graphs to electrical networks using the adjacency matrix; while it may be easy to find all the circuits, additional computing power is required to detect cycles within those circuits. Danielson[34] employed a variable adjacency matrix which included an edge identifier. Elsewhere, similar algorithms have been proposed.[35−38]

Corey[39] presented an algorithm based on a connection table which detected all circuits, their size, and their relationship with each other (*isolated*, *spiro*, *fused*, and *bridge*). Each cycle is represented by a vector and is obtained by walking the spanning tree. Processing begins at the first atom in the connection table and randomly traces a route, nodes are labeled as *selection points*, and if the next node is the starting point the system backtracks to the previous selection point.

When all possible paths from the starting atom have been covered, a check is made to see if all the vertices have been included in the cycle set, and should one have been missed, path tracing is repeated.

Balaban[40] proposed collapsing the initial graph (homeomorphic reduction), with the intention of simplifying the cycles. The algorithm generates all possible combinations of the edges in the adjacency matrix and each combination is checked to see whether it is a cycle or the union of disjoint cycles, but this approach requires excessive computing power.

Hanser[41] presented an algorithm which worked with an additional graph referred to as *P-Type*. This graph is reduced by erasing nodes and concatenating edges until the cycles are obtained. The basic problem here is once again the rise in computational demands incurred by the increase in node connectivity.

3. CYCLICAL CONJUNCTION

Cyclical conjunction, an operation we shall represent by the $\oplus$ symbol, is a mathematical operation that acts on two cycles of a graph G in order to extract a new cycle from the graph. Thus, starting with a small set of cycles belonging to a graph G, with a cardinality equal to the cyclomatic complexity, every cycle in the graph will be detected by performing the cyclical conjunction operation on pairs of cycles, successively.

For a graph G, cyclical conjunction of two different cycles R and T—each defined by a set of nodes, and where in G there are no duplicated elements and order is important—produces a new cycle C belonging to the graph G; this cycle must fulfill the following conditions:

1. If $R \equiv \langle\varnothing\rangle$ or $T \equiv \langle\varnothing\rangle$, then $C \equiv \langle\varnothing\rangle$.
2. If $R \cap T \equiv \langle\varnothing\rangle$, then $C \equiv \langle\varnothing\rangle$.
3. If $\bar{R} = 1$ or $\bar{T} = 1$, then $\bar{C} = 1$, and $\bar{G}$ now describes the number of nodes in the graph G.
4. If $\overline{R \cap T} = 1$, then $\bar{C} = 1$, meaning that R and T cycles are connected by a single node.
5. If $R \equiv T$, then $C \equiv \langle a,z\rangle$, where a and z are two consecutive nodes in the R or T cycles.
6. Otherwise C is a cycle belonging to G, such that $C \subseteq \{R + T\}$.

3.1. Properties of Cyclical Conjunction. The cyclical conjunction operation satisfies the following properties:

1. *Commutative*: $R \oplus T = T \oplus R$.

2. *Parental*: cyclical conjunction of any cycle C with a parent or ancestor cycle (one which has been involved in cyclical conjunction to obtain C) will not give rise to a new cycle of the graph G. (If $C = R \oplus T$, then: $R = T \oplus C$ and $T = R \oplus C$, with R and T being designated as ancestor cycles of C.)

3. *Semiassociative*: $R \oplus (T \oplus S) \equiv (R \oplus T) \oplus S$, wherever there is a set $Z \equiv \langle z_1,z_2,...,z_n\rangle | Z \subseteq R \cap T$, $Z \subseteq R \cap S$, $Z \subseteq T \cap S$.

3.2. Procedure for the Application of Cyclical Conjunction. Cyclical conjunction may be implemented through the following three-stage procedure:

First Stage: Perform Intersection between the R and T Cycles. Intersection is performed between the R and T cycles, the properties of the operator are checked, and, according to the result, we move on to the next stage. Thus

1. If $R \cap T \equiv \langle\varnothing\rangle$, then $C \equiv \langle\varnothing\rangle$. The cycles have no common elements and the process is terminated.

2. If $R \cap T \equiv \langle r_n\rangle \equiv \langle t_m\rangle$, then $C \equiv \langle r_n\rangle \equiv \langle t_m\rangle$. The cycles are joined by a common node $r_n \equiv t_m$, and the process is terminated.

3. If $R \cap T \equiv R \equiv T$, then $C \equiv \langle r_i,r_j\rangle \equiv \langle t_m,t_n\rangle$, where $r_i = t_m, r_j = t_n$, so the process is halted.

4. If $R \cap T \equiv Z$, we then move on to the second stage.

Second Stage: Reorder the Sets R and T according to the Order of Z.

5. Reorder R such that the elements common to Z are the last in the set: $R \equiv \langle r_1,r_2,...,r_k,z_1,z_2,...z_n\rangle$, $\forall\ (z_1,z_2,...z_n) \in R$.

6. Reorder T such that the elements common to Z are the first in the set and will be found in the same relative arrangement as in R: $T \equiv \langle z_1,z_2,...,z_n,t_L,t_{l+1},...,t_m\rangle$, where $\forall\ (z_1,z_2,...z_n) \in T$.

Third Stage: Perform the Cyclical Conjunction of R and T.

7. Initialize the set $C \equiv \langle\varnothing\rangle$.

8. Append to C the elements of R up to and including the first element of Z. $C \equiv \langle r_1,r_2,...,r_n,z_1\rangle$.

9. Add to C the elements of T in reverse order up to the last element of Z (inclusive). $C \equiv \langle r_1,r_2,...,r_n,z_1,t_n,t_{m-1},...,t_l,z_n\rangle$. The resulting set is a cycle in the structure of the graph.

4. CALCULATION OF CYCLES USING CYCLICAL CONJUNCTION

Graph cycle extraction involves the repeated application of cyclical conjunction to the set of known cycles, the process being repeated until no new cycles are obtained or until reaching the theoretical maximum number of cycles in the graph (2^{η}-1)—this limit is seldom reached.

4.1. Preprocessing (Obtaining the Initial Set of Cycles). The process begins by obtaining an initial set of cycles (***ICL***), applying cyclical conjunction to all possible pairs in order to arrive at a new list of cycles (***NCL***).

The initial set of cycles is extracted by an algorithm developed by the present authors.[42] It is based on the adjacency matrix representing the graph ordered by node connectivity and by means of a zigzag process a spanning tree is built, whose main feature is its shallowness but great breadth. Then, by choosing unvisited edges, a set of cycles is obtained whose cardinality is equal to or greater than the cyclomatic complexity.

The algorithm avoids the use of path tracing and produces excellent results, with a computational cost of N (number of nodes), at worst.

4.2. Processing (Extracting All Cycles from the Graph). Once the initial cycles have been obtained, we can calculate all cycles in the graph by applying cyclical conjunction to this set (***ICL***) and to the new sets of cycles generated by this function. The process will terminate when all the required operations have been performed on the sets of known cycles.

Chart 1 shows the pseudocode for this operation. The algorithm works as follows:

1. The data structures used by the algorithm are initialized. Initial cycles obtained[42] are stored in the *Initial Cycles List* (***ICL***).

Chart 1. Algorithm for the Calculation of All Cycles, Using Cyclical Conjunction

```
Cyclical Conjunction (Sequential Algorithm).
      /* Prepare the list of previous cycles for the first
      iteration */
   assignlist(ICL, PCL);
      /* Initialize the list of new cycles */
   NCL = NULL;
      /* Add the first saga to the TCL */
   insertlist(ICL, TCL);
      /* Initialize terminator */
   end = TRUE;
      /* Begin iteration loop */
   While (end)
      /* Walk all cycles of the ICL */
      For i = 0 to ICL.total_cycles
      /* Walk the cycles from the last saga obtained (PCL) */
         For j = 0 to PCL.total_cycles
      /* Check if operation will be effective */
            If property(ICL.cycle[i], PCL.cycle[j]) = FALSE
      /* Perform cyclical conjunction */
               cycle = cyclicalconjunction (ICL.cycle[i], PCL.cycle[j]);
      /* Check cycle and its presence in the NCL and TCL */
               If is_cycle(cycle)
                  If search(NCL, cycle) = FALSE
      /* If it is a new cycle, insert in new list (NCL) */
                     If search(TCL, cycle) = FALSE
                        insert(NCL, cycle);
                     EndIf
                  EndIf
               EndIf
            EndIf
         EndFor
      EndFor
      /* On ending iteration, insert NCL into TCL, then initialize
      variables for next iteration.
      If no new cycles are created or end of total cycles is
      reached, terminate the process */
      If (NCL != NULL)
         insertlist(NCL, TCL);
         assignlist(NCL, PCL);
         NCL = null;
      EndIf
      Else
         end = FALSE;
      EndElse
   EndWhile.
End CyclicalConjunction (Sequential Algorithm).
```

Chart 2. Information Held in the ***TCL***, ***CSL***, and ***NSL*** Data Structures

```
Define Total Cycles List (TCL)
      /* List containing each group of cycles obtained from the
      cycles extracted from the graph */
   infosaga:   bitsarray,
      /* bitmap storing information on cycles taking part in the
      cyclical conjunction operation to obtain the current cycle */
   cycles:     integer,
      /* number of cycles forming the saga */
   firstcycle: *CSL,
      /* pointer to the first cycle on the TCL */
   nextsaga:   *TCL,
      /* pointer to the previous cycle on the TCL */
End of Total Cycles List.


Define Cycles Structure List (CSL)
      /* List containing each and every cycle */
   occur:      integer,
      /* position of the structure in the occurrence of the TCL */
   infocycle:  bitsarray,
      /* information on the cycles producing the corresponding
      cycle through a prior cyclical conjunction operation */
   edges:      bitsarray,
      /* edges occurring in the cycle */
   direction:  boolean,
      /* direction walked in the list of nodes */
   nextcycle:  *CSL,
      /* pointer to the next cycle on the TCL (saga) */
   firstnode:  *NSL,
      /* pointer to the first node on the list */
End of Cycles Structure List.


Define Nodes Structure List (NSL)
      /* List containing each element of the graph that occurs in
      the cycles */
   idnode:     integer,
      /* identifier for the node */
   nextnode:   *NSL,
      /* pointer to the next node */
   priornode:  *NSL,
      /* pointer to the previous node */
End of Nodes Structure List.
```

2. The set of initial cycles (***ICL***) is copied straight into the set of all the cycles in the graph—the *Total Cycles List* (***TCL***).

3. The set of initial cycles is copied directly into the *Previous Cycles List* (***PCL***), which stores the cycles generated during the previous step of the iteration.

4. Cyclical conjunction is performed on cycle pairs from the ***ICL*** and ***PCL*** lists, for all pairs which satisfy the properties of this operator, and the following:

(a) First, the parental property is applied, to determine whether it makes sense to perform conjunction, thus avoiding a great number of operations and tests. The commutative property is applied directly due to the structure of the algorithm.

(b) We test to see if the intersection of the cycles might produce a new cycle, and if this is not the case, the operation is rejected. This step determines the nodes in common between the two cycles.

(c) Cycles are rearranged and stored in the order established by the second stage of the algorithm (steps 5 and 6).

(d) Cyclical conjunction is carried out on the cycles, enacting the third stage of the algorithm.

(e) A check is made to see if the new cycle is present in the *New Cycles List* (***NCL***), a list that stores the cycles generated during each iterative pass of the algorithm.

(f) A check is then made to see if the new cycle is present in the total cycles list (***TCL***).

(g) The cycle is stored in the new cycles list (***NCL***).

5. When the iteration step has concluded, the ***NCL*** is inserted into the ***TCL***, the ***PCL*** is initialized to the ***NCL*** and the ***NCL*** is cleared, ready for the next pass of the iteration.

6. The iterative process is halted when conjunction of the lists yields no further new cycles, or when the ***TCL*** has a cardinality equal to 2^{η}-1.

4.3. Data Structures Used by the Algorithm. The number of cyclical conjunction operations will clearly rise with graph complexity, in line with the number of cycles contained in the graph. While the properties of the cyclical conjunction operator described above in Section 3 contribute greatly to the reduction of the number of operations, there is still a real need for data structures that will keep computational cost as low as possible.

Three main data structures have been employed to represent the information managed by the algorithm, as described in Section 4.2.

*4.3.1. Nodes Structure List (**NSL**).* The nodes of the graph that form each cycle are contained in a double-linked list. This list enables the algorithm to walk the nodes of a cycle in any order, though obviously consecutively. Chart 2 and Figure 1 contain textual and graphical information detailing this data structure.

*4.3.2. Cycles Structure List (**CSL**).* Cycles are represented by a data structure that stores their nodes, using pointers to the corresponding ***NSL***. For each *saga* (iteration of the algorithm), detected cycles are linked by an instance of the ***CSL***, a simply linked list as shown in Figure 1. Chart 2 shows this data structure, with the following points worthy of note:

• The cycles for each *saga* (each element of the ***TCL***) are identified by their arrangement in the ***CSL*** (the *occur* attribute). Identification of both the *saga* and the cycle within it permits the unambiguous perception of the step of the iterative process in which it was detected. This information, together with that provided by the *infocycle* attribute, allows us moreover to identify the cycles involved in the cyclical

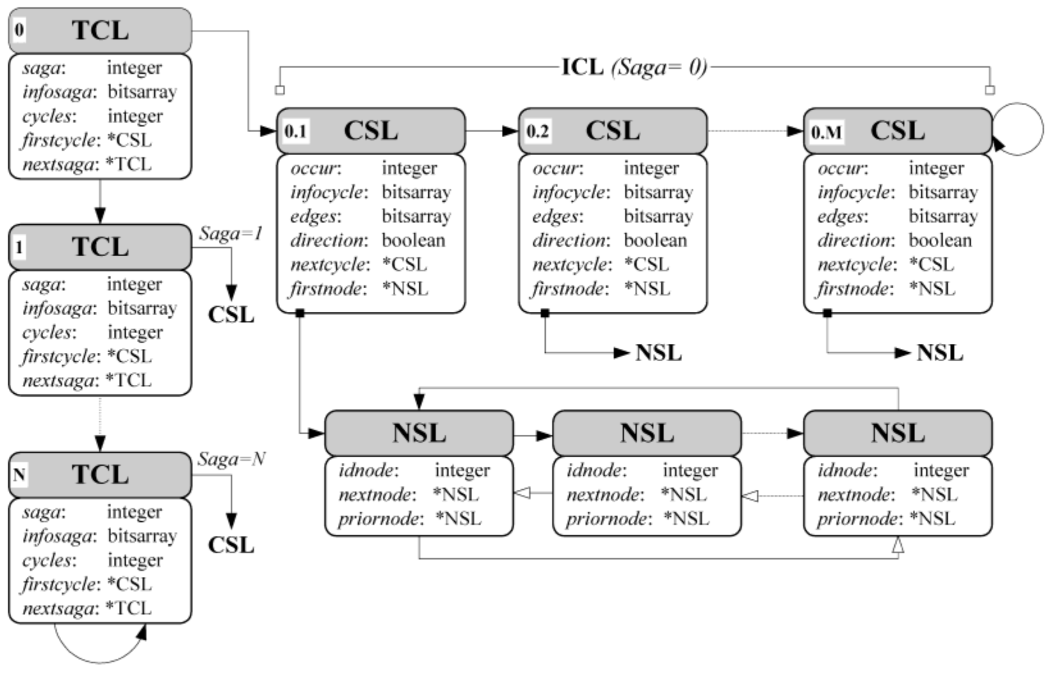


Figure 1. Graphic representation of the data structures used by the algorithm.

conjunction operation that generated the new cycle, which in turn permits the effective application of the parental property.

Where *saga* = *0*, corresponding to the ***ICL*** (see Figure 1), this attribute is assigned a value supplied by the expression cycle.*info* = *0 OR* 2^{occur}, where *occur* is the value of the occurrence of the cycle in the *saga* (its ordinal arrangement for an occurrence of the ***TCL***).

For succeeding *sagas* (elements of the ***TCL***), the attribute is calculated from the values of the *infocycle* attributes of the cycles involved in the cyclical conjunction operation to obtain the corresponding value, i.e., cycleC.*infocycle*=cycleR.*infocycle* AND cycleT.*infocycle*.

• The *direction* attribute is used to examine the cycle nodes in a different order. Employed in conjunction with the pointers to the first and last nodes of the cycle (*firstnode* and *lastnode*), it allows the cycle to be represented by any sequence of its nodes.

These attributes, together with the *edge* attribute, boost the performance of the first and second stages of the algorithm, since the cycles taking part are arranged in a suitable sequence of nodes by simply modifying the attributes' values, thus obviating any further ordering process.

*4.3.3. Total Cycles List (**TCL**).* This is a complex store of information about every cycle extracted from the graph. As seen in Figure 1, it is a simply linked list where each element points to a ***CSL*** containing information about the cycles discovered in each iteration of the algorithm (the above-mentioned *sagas*).

Chart 2 shows the structure of the elements of this list, where

• The *infosaga* attribute is used to apply the parental property, bearing information about which *sagas* intervened in the generation, through the cyclical conjunction operations, to generate the current *saga*.

• The arrangement of cycles in the ***TCL*** (and the *saga* attribute) allows their grouping for repeated iterations of the algorithm, while providing their unique identification, preventing duplication of operations and ensuring correct application of the commutative property.

4.3.4. Other Temporary Structures. While running the proposed algorithm, use is made of other temporary structures such as the ***ICL***, ***NCL***, and ***PCL***. In fact, these are simply pointers to a *saga*, an element of the ***TCL***, hence their memory requirements are minimal.

5. EXPERIMENTAL RESULTS

The proposed algorithm has been implemented in the C language and tested using a Silicon Origin 2000 computer across a wide range of graphs. Results are summarized in Table 1. The series of graphs tested included those examined in other studies cited in the references section.

Table 1 details graph information (number of nodes, edges, and cyclomatic complexity), results for algorithms from cited studies, and results for the algorithm herein proposed.

Analysis of experimental results reveals the following:

• In every case our algorithm perceives the total number of cycles in the problem graph. This number coincides with data published elsewhere and is almost always much lower than the maximum possible number of cycles (2^{η}-1).

• There is absolutely no incidence of nonexistent cycles being generated, thus validating the cyclical conjunction operator for the extraction of cycles from a graph.

• Computing time per cyclic conjunction operation is not constant; this effect is mainly due to the computational cost of the second stage of the algorithm, where the determining factor is the number of nodes involved.

• The properties of the cyclical conjunction operator allow substantial reduction of the number of operations required, as detailed in Table 1 by the reduction factor *R*. If the properties of the operator were not applied, as shown in the *property* function in Chart 1, the number of operations required would be

Table 1. Experimental Results for the Algorithm[a]

	characteristics			published data			results						
graphs	nodes	edges	η	ref	cycles	operations	$\overline{ICL}$	$\overline{TCL}$	O_T	O_P	time (s)	R	E
G-03x03	3	3	1	36-3	1	2	1	1	0	0	19.4×10^{-6}		1.00
G-10x11a	10	11	2	37-3	3		2	3	3	1	78.6×10^{-6}	67	1.00
G-10x11b	10	11	2	37-4	3		2	3	3	1	66.6×10^{-6}	67	1.00
G-08x10	8	10	3	6-7	4		3	4	6	1	120.2×10^{-6}	83	1.00
G-16x18	16	18	3	38-d			3	6	12	4	247.4×10^{-6}	67	0.75
G-05x07	5	7	3	6-6	6		3	6	12	4	225.0×10^{-6}	67	0.75
G-04x06	4	6	3	36-4	7	19	3	7	15	4	285.8×10^{-6}	73	1.00
G-06x08	6	8	3	6-5	7		3	7	15	4	284.2×10^{-6}	73	1.00
G-20x22	20	22	3	37-5	6		3	6	12	3	574.0×10^{-6}	75	1.00
G-20x25	20	25	6	38-e			6	13	57	10	1116.6×10^{-6}	82	0.70
G-16x19a	16	19	4	6-1			4	14	46	19	1265.2×10^{-6}	59	0.53
G-16x19b	16	19	4	6-3	14		4	14	46	16	1320.6×10^{-6}	65	0.63
G-12x15	12	15	4	38-c			4	10	31	10	1537.8×10^{-6}	68	0.60
G-17x20	17	20	4	37-4	10		4	10	31	9	1835.6×10^{-6}	71	0.67
G-12x16	12	16	5	6-4	28		5	28	125	45	4285.8×10^{-6}	64	0.51
G-05x10	5	10	6	36-5	37	186	6	37	201	60	5059.6×10^{-6}	70	0.52
G-40x49	40	49	10	37-9	30		10	30	245	30	7040.2×10^{-6}	88	0.67
G-26x31	26	31	6	37-7	21		6	21	106	22	7402.6×10^{-6}	79	0.68
G-10x15a	10	15	6	38-a			6	47	261	95	10.0×10^{-3}	64	0.43
G-18x27	18	27	10	38-g			10	48	425	69	13.6×10^{-3}	84	0.55
G-12x18	12	18	7	38-f			7	39	245	54	14.3×10^{-3}	78	0.59
G-10x15b	10	15	6	6-8	57		6	57	321	105	14.0×10^{-3}	67	0.49
G-14x20	14	20	7	18-18			7	64	421	126	20.9×10^{-3}	70	0.45
G-34x42	34	42	9	16-26			9	78	657	157	31.8×10^{-3}	76	0.44
G-24x30a	24	30	7	37-8	94		7	94	630	231	31.9×10^{-3}	63	0.38
G-24x30b	24	30	7	16-24			7	94	630	231	32.6×10^{-3}	63	0.38
G-06x15	6	15	10	36-6	197	2719	10	197	1915	420	65.2×10^{-3}	78	0.45
G-25x35	25	35	11	19-12			11	708	7722	1819	600.8×10^{-3}	76	0.38
G-07x21	7	21	15	36-7	1172	61047	15	1172	17460	2820	891.4×10^{-3}	84	0.41
G-08x28	8	28	21	36-8	8018	2136588	21	8018	168147	20370	29.6	88	0.39
G-09x30	9	30	22	*b*			22	9975	219197	26566	58.7	88	0.37
G-60x76	60	76	17	*b*			17	24635	418642	87771	303.2	79	0.28
G-09x33	9	33	25	*b*			25	27588	689375	68974	421.0	90	0.40
G-13x33	13	33	21	*b*			21	33843	710472	75359	631.2	89	0.45
G-09x36	9	36	28	*b*			28	62814	1758386	164136	3064.8	91	0.38

[a] η: cyclical conjunction, $\overline{ICL}$: initial cycles number, $\overline{TCL}$: total cycles number, *Ref.*: bibliography reference-identification of the graph in the article, O_T: theoretical operations, O_P: real operations performed, R: reduction factor, E: effectivity. [b] Complex graphs included in the test suite.

$$O_T = \sum_{i=1}^{i=n} \overline{ICL} x \overline{PCLi} \qquad (2)$$

where O_T is the number of theoretical operations, n is the number of *sagas*, $\overline{ICL}$ is the cardinality of the initial cycles set, and $\overline{PCL}$ is the cardinality of the set of cycles extracted by the previous iteration of the algorithm.

The reduction factor R (see Table 1), which shows the effectivity of applying the operator properties, may be defined thus

$$R = 1 - \frac{O_P}{O_T} x100 \qquad (3)$$

where O_P is the number of operations performed on each graph (as described in Chart 1; this is how often the *cyclicalconjunction* function is executed). Table 1 shows how the reduction factor R increases with graph complexity (the number of nodes present).

• Algorithm effectivity (E) is defined by the formula

$$E = \frac{\overline{TCL} - \overline{ICL}}{O_P} \qquad (4)$$

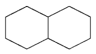
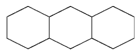
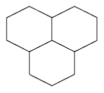
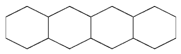
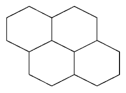
where O_P is the number of cyclical conjunction operations performed. In Table 1 it may be seen that E decreases with graph complexity, depending on factors such as the total cycles number, which cycles have already been detected in the initial cycles set, and the topology of the graph.

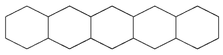
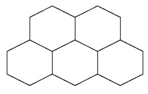
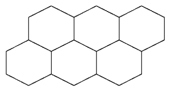
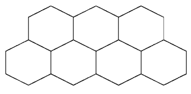
• Although a few unnecessary cyclical conjunction operations might be carried out (those which fail to generate new cycles), the number of operations is still much lower than in algorithms described elsewhere (this information is detailed in a study by Hanser[41]).

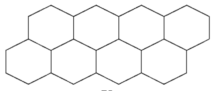
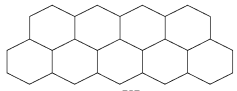
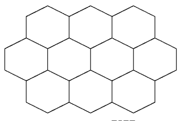
• While calculation time cannot be directly compared with that of other cited algorithms, since this depends on computing resources employed, it may be seen that graphs with fewer than 1000 cycles take under 1 s to analyze (in the order of milliseconds for graphs of medium complexity and microseconds for simple graphs), and only where graphs are highly complex, with say tens of thousands of cycles, is computing time relatively high. Many factors intervene here, such as the number of operations to be performed, the size of the data structures used to store the cycles, and graph topology.

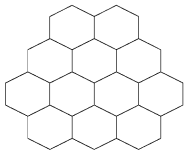
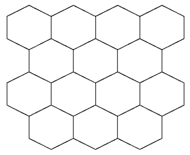
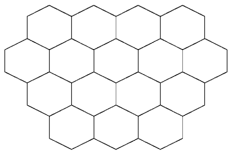
• Given the complex nature of fused rings, particularly those found in benzenoid systems,[44] we felt this would be a stern test of the algorithm's performance. As may be seen in Table 2, the algorithm proved highly efficient even for such complex systems; the number of real operations (O_P) was well below the theoretical limit (O_T), with successful extraction of all cycles at moderate computing cost.

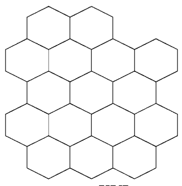
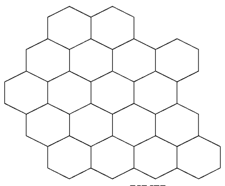
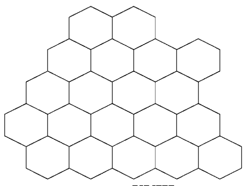
Table 2. Summary of Results Obtained from Applying the Proposed Algorithm to a Set of Benzenoid Systems

	I	**II**	**III**	**IV**	**V**
					
Nodes, edges, η	10, 11, **2**	14, 16, **3**	13, 15, **3**	18, 21, **4**	16, 19, **4**
$\overline{\text{ICL}}$, $\overline{\text{TCL}}$	2, **3**	3, **6**	3, **7**	4, **10**	4, **14**
O_T, O_P,	3, **1**	13, **3**	15, **4**	31, **9**	46, **19**
Time (sec.)	**$82\ 10^{-6}$**	**$214\ 10^{-6}$**	**$335\ 10^{-6}$**	**$556\ 10^{-6}$**	**$997\ 10^{-6}$**

	VI	**VII**	**VIII**	**IX**
				
Nodes, edges, η	22, 26, **5**	19, 23, **5**	22, 27 **6**	25, 31, **7**
$\overline{\text{ICL}}$, $\overline{\text{TCL}}$	5, **15**	5, **26**	6, **46**	7, **79**
O_T, O_P,	61, **15**	115, **41**	255, **92**	525, **162**
Time (sec.)	**$1539\ 10^{-6}$**	**$3523\ 10^{-6}$**	**$9705\ 10^{-6}$**	**$26.3\ 10^{-3}$**

	X	**XI**	**XII**
			
Nodes, edges, η	28, 35, **8**	31, 39, **9**	32, 41, **10**
$\overline{\text{ICL}}$, $\overline{\text{TCL}}$	8, **133**	9, **220**	10, **552**
O_T, O_P,	1028, **292**	1935, **481**	5465, **1820**
Time (sec.)	**$65.3\ 10^{-3}$**	**$246.5\ 10^{-3}$**	**$479.0\ 10^{-3}$**

	XIII	**XIV**	**XV**
			
Nodes, edges, η	37, 48, **12**	43, 56, **14**	47, 62, **16**
$\overline{\text{ICL}}$, $\overline{\text{TCL}}$	12, **1880**	14, **5738**	16, **21292**
O_T, O_P,	22482, **6942**	80227, **24802**	340536, **83446**
Time (sec.)	**3.15**	**20.4**	**231.6**

	XVI	**XVII**	**XVIII**
			
Nodes, edges, η	48, 63, **16**	53, 70, **18**	58, 77, **20**
$\overline{\text{ICL}}$, $\overline{\text{TCL}}$	16, **19457**	18, **64799**	20, **215625**
O_T, O_P,	311176, **92853**	1166211, **304385**	4312290, **985023**
Time (sec.)	**204.5**	**2721**	**42475**

DISCUSSION

Information concerning the number, size, and composition of a compound's rings is widely used, alongside other topological descriptors, to tackle many issues in chemistry, such as the following:

• Deriving the relationship between structure and physicochemical properties of compounds (e.g. shape[45] is intimately associated with molecular electric and steric properties, which are implicated in structure−activity relationships of many kinds). Work in progress on QSPR/QSAR applications employs both topological and activity descriptors to this end.[3,46−49]

• Information clustering and screening criteria for searching huge databases of compounds (e.g. in the design of drugs containing a relevant number of rings).[50−55]

• The development of many other solutions for measuring similarity/diversity or molecular complexity, substructure searches in complex structures, etc.[56−57] used both in the aforementioned applications and in the design and synthesis of new compounds.

This paper has presented an algorithm, based on an algebraic operator named cyclical conjunction, which may be employed for the extraction of all the cycles from a graph. These are detected using a set of initial cycles and an iterative process to perform cyclical conjunction on new rings detected in the foregoing iteration.

The algorithm has been tested on a large number of graphs (several of which have previously been studied elsewhere), and all existing cycles were successfully detected. There was no instance of the generation of nonexistent cycles, and where comparison was possible with results reported elsewhere, the proposed algorithm would appear to operate more efficiently.

Implementation of the cyclical conjunction operator properties afforded a reduction of 60−90% of the number of operations required to extract all cycles, greatly speeding up the algorithm, although this obviously raised computational cost a little.

Tests showed that calculation time for the extraction of graph cycles is dependent on factors such as total number

Table 3. Summary of Results Obtained from Applying the Proposed Algorithm to a Database of Compounds of Varying Complexity (See Ref 43)

compound	nodes	edges	η	$\overline{\text{ICL}}$	$\overline{\text{TCL}}$	O_P	time (s)
diacylsinuatol	49	53	5	5	6	1	0.000058
robinin	52	57	6	6	7	1	0.000070
amentoflavone hexaacetate	58	63	6	6	8	2	0.000118
browniine perchlorate	36	41	6	6	9	5	0.000374
furostane-3,22,26-triol	77	85	9	9	10	2	0.000174
bufotoxin	59	63	5	5	11	9	0.000639
glycyrrhetinic ammoniated	57	63	7	7	12	8	0.000834
7-xylosyltaxol C	71	77	7	7	13	9	0.001116
ginsenoside Rb1 (sanchinoside E1; gypenoside III)	80	87	8	8	14	6	0.001079
mogroside V	89	97	9	9	15	8	0.001134
adiantifoline	55	61	7	7	16	13	0.001976
mogroside III	67	73	7	8	17	10	0.001641
kashtacin	73	80	8	8	18	13	0.002229
cyclocanthoside E	57	63	7	7	19	21	0.002615
furostane-2,3,22,26-tetrol	94	103	10	10	20	13	0.002624
digitonin	85	95	11	11	21	13	0.002875
azadirachtin	53	60	8	8	22	22	0.002983
saikosaponin A	56	63	8	8	23	25	0.002688
alpha-solanine	64	72	9	9	24	19	0.005040
19-hydroxyl baccatin III	44	48	5	6	25	37	0.003216
dasiant A	47	54	8	8	26	33	0.004876
haplocine	28	33	6	6	27	37	0.003452
hopeaphenol	70	81	12	12	28	24	0.005326
conodurine	53	61	9	10	29	29	0.005576
solasonine (solasodamine)	66	74	9	9	30	47	0.005843
peraksine (vomifoline)	23	28	6	6	31	43	0.006284
jujuboside A	87	97	11	12	33	28	0.008235
cycloorbicoside G	57	65	9	9	34	47	0.006377
pseudaconitine	49	55	7	7	35	47	0.009145
beta-D-glucopyranosyl (1→6) erycordine	63	70	8	9	36	54	0.008040
anemarsaponin A	52	59	8	8	37	49	0.008987
astragaloside VII	68	76	9	9	38	51	0.008331
askendoside F	66	73	8	8	39	74	0.008180
harrisonin	37	42	6	7	40	65	0.007432
ajmaline-digitoxigenin	58	68	11	11	44	65	0.011401
patrinozid D	103	114	12	12	45	60	0.013575
elsinochrome A (phycarone)	42	47	6	6	47	106	0.011340
cyclosieversioside B	63	70	8	9	48	82	0.015149
cyclosiversioside B (cyclosieversioside B)	62	69	8	9	50	73	0.015644
stevioside (eupatorin; rebaudin)	58	64	7	8	54	99	0.013690
eldeline (deltaline, delphelatine)	37	43	7	7	59	101	0.017892
delcorine	37	43	7	8	60	104	0.016953
liriodendrin	54	59	6	9	63	86	0.029649
brucine (10,11-dimethoxystrychnine)	34	40	7	7	67	140	0.021645
2,11,13,14 O-tetraacetate tangutisine	42	48	7	7	69	109	0.021102
thalibrunine	51	57	7	7	72	180	0.027898
alpha-viniferin	57	66	10	10	76	153	0.037357
ajmaline-gitoxigenin	87	103	17	17	77	122	0.035356
glycyrrhizic acid, derivative of	116	125	10	11	84	201	0.043177
peracetyl ginsenoside Rb1	125	132	8	10	92	153	0.044175
hypaconitine (deoxymesaconitine; japaconitine B1)	46	52	7	8	93	188	0.036703
hypaconitine (many other names)	47	53	7	8	93	188	0.037615
villalstonine (alkaloid B)	53	63	11	11	103	155	0.077586
tenuipine	47	54	8	8	106	310	0.054134
isochondodendrine (isobebeerine; isendryl; isodendril)	46	52	7	8	123	288	0.066797
3,15-di-O-acetylmesaconitine	52	58	7	10	126	190	0.065944
mesaconitine (japaconitine A, japaconitine B)	47	53	7	10	127	184	0.076345
tiliacorine	45	52	8	8	145	409	0.068508
hexamethylviniferin	63	72	10	12	255	833	0.198386

of cycles and graph topology and complexity, in addition to the nature of the initial cycles detected at the preprocessing phase—factors that are either unknown or excessively costly to determine a priori.

The data structure allows real-time storage of all cycle information. For example, from the compounds shown in Table 2, the following (and more) information was obtained: (II) has 3 cycles of 6 nodes, 2 of 10, and 1 of 12; (III) has 3 cycles of 6 nodes, 2 of 10, and 1 of 12; (IV) has 4 cycles of 6 nodes, 3 of 10, 2 of 14, and 1 of 18; and (V) has 4 cycles of 6 nodes, 5 of 10, 2 of 12, and 3 of 14.

This information is extremely useful for the applications mentioned above—(II) could be found in (IV), but not in (III), nor in (V), though it could also be found in a substructure of (II); (III) was found in (V), but not in (IV), though it was also found in a substructure of (III); while (II) and (III) are formed by 3 benzenes, and (IV) and (V) by 4 benzenes, their shapes (perimeters) are different, since the largest ring in (II) has 14 nodes, whereas in (III) it has 12, and the biggest cycle in (IV) has 18 nodes, compared to (V) which has 14. Furthermore, the data may be used in the consideration of new descriptors related to the ring system; new descriptors are currently undergoing evaluation, and results will be presented in a future paper.

Table 3 summarizes the application of our algorithm to a database[43] of over 3000 compounds; it may be seen that complete molecular cycle system information is extracted at low computational cost, even in the case of complex molecules.

Although the computational cost of our algorithm is generally modest, it will clearly increase for extremely complex graphs, suggesting the need for parallel preprocessing techniques in a practical implementation of the proposed algorithm; this aspect is the object of a future study.[58]

ACKNOWLEDGMENT

The authors would like to thank Andy de Laet, PhD, Project Manager SPECS and BioSPECS B.V., for providing us with the database used in the validation of the proposed algorithm.

REFERENCES AND NOTES

(1) Devillers, J.; Balaban, A. T. *Topological Indices and related Descriptors in QSAR and QSPR*; Gordon and Breach, Eds.; Amsterdam, The Netherlands, 1999.

(2) Gutman, I.; Strada, E. Topological indexes based on the line graph of the molecular graph. *J. Chem. Inf. Comput. Sci.* **1996**, *36*(3), 541−543.

(3) Cao, C.; Yuan, H. Topological indices Based on Vertex, Distance, and Rings: On the Boiling Points of Paraffins and Cycloalkanes. *J. Chem. Inf. Comput. Sci.* **2001**, *41*(4), 867−877.

(4) Randic, M. On Characterization of Cyclic Structures. *J. Chem. Inf. Comput. Sci.* **1997**, *37*, 1063−1071.

(5) Lipkus, A. H. Exploring Chemical Rings in a Simple Topological-Descriptor Space. *J. Chem. Inf. Comput. Sci.* **2001**, *41*(2), 430−438.

(6) Rücker, G.; Rücker, C. Walk Counts, Labyrinthicity, and Complexity of Acyclic and Cyclic Graphs and Molecules. *J. Chem. Inf. Comput. Sci.* **2000**, *40*(1), 99−106.

(7) Pisanski, T.; Plavsic, D.; Randic, M. On Numerical Characterization of Cyclicity. *J. Chem. Inf. Comput. Sci.* **2000**, *40*, 520−523.

(8) Chua, L. O.; Chen, L. K. On optimally sparce cycle and coboundary basis for a linear graph. *IEEE Trans. Circuit Theory* **1973**, *20*, 495−503.

(9) Knuth, D. E. *The Art of Computer Programing, 1*; Addison-Wesley: 1968; pp 363−370.

(10) Corey, E. J.; Petersson, G. A. An algorithm for machine perception of synthetically significant rings in complex cyclic organic structures. *J. Am. Chem. Soc.* **1972**, *94*, 460−465.

(11) Mateti, P.; Deo, N. On algorithms for enumerating all circuits of a graph. *SIAM J. Comput.* **1976**, *5*, 90−99.

(12) Randic, M. Random walks and their diagnostic value for characterization of atomic environment. *J. Comput. Chem.* **1980**, *1*(4), 386-399.

(13) Downs G. M.; Gillet V. J.; Holliday J. D.; Lynch, M. F. Theoretical aspects of ring perception and development of the extended set of smallest rings concept. *American Chemical Society* **1988**, *29*(3), 187−206.

(14) Balasubramanian, K.; Subhash, B. Characterization of isospectral graphs using graph invariants and derived orthogonal parameters. *J. Chem. Inf. Comput. Sci.* **1998**, *38*(3), 367−373.

(15) Basak, S.; Balaban, T.; Grunwald, G.; Brian, D. Topological indices: Their nature and mutual relatedness. *American Chemical Society* **2000**, *40*(4), 891−1150.

(16) Plotkin, M. Mathematical basis of ring-finding algorithms in CIDS. *J. Chem. Doc.* **1970**, *11*(1), 60−63.

(17) Bersohn, M. An algorithm for finding the synthetically important rings of a molecule. *Theory Graphs, American Mathematical Society Colloquium Publications* **1973**, *38*, 1239−1241.

(18) Esack, A. A procedure for rapid recognition of the rings of a molecule. *J. Chem Soc.* **1975**, 1.

(19) Schmidt, B.; Fleischhauer, J. A. Fortran iv program for finding the SSSR of a graph. *J. Chem. Inf. Comput. Sci.* **1984**, 18.

(20) Zamora, A. An algorithm for finding the smallest set of smallest rings. *J. Chem. Inf. Comput. Sci.* **1976**, *16*(1), 40−43.

(21) Roos-Kozel, B. L.; Jorgensen, W. L. Computer-assisted mechanistic evaluation of organic reactions. 2. perception of rings, aromaticity, and tautomers. *American Chemical Society* **1981**, *21*(2), 101−111.

(22) Gasteiger, J.; Jochum, C. An algorithm for the perception of synthetically important rings. *J. Chem. Inf. Comput. Sci.* **1979**, *19*(1), 43−48.

(23) Fan, T. F.; Doucet, J.; Barbu, A. Ring perception. a new algorithm for directly finding the smallest rings from a connection table. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 657−662.

(24) Figueras, J. Ring perception using breadth-first search. *J. Chem. Inf. Comput. Sci.* **1996**, *36*, 986−991.

(25) Shindaku Fujita, A new algorithm for selection of synthetically important rings. *J. Chem Inf. Comput. Sci.* **1988**, *28*, 22−26.

(26) Welch, J. T.; A mechanical analysis of the cyclic structure of undirected linear graphs. *J. ACM* **1966**, *13*(2), 205−210.

(27) Gotlieb, C. C.; Corneil, D. G. Algorithms for finding a fundamental set of cycles for an undirected linear graph. *Commun. ACM* **1967**, *10*(12), 780−783.

(28) Gibbs, E. A cyclic generation algorithm for finite undirected linear graphs. *J. Assoc. Comput. Machinery* **1969**, *16*(4), 564−568.

(29) Paton, K. An algorithm for finding a fundamental set of cycles of a graph. *Commun. ACM* **1969**, *12*(9), 514−518.

(30) Wipke, W.; Dyott, M. Use of ring assemblies in a ring perception algorithm. *J. Chem. Inf. Comput. Sci.* **1975**, *15*(3), 140−147.

(31) Sorkau, Ringerkennung in chemischen strukturen mit dem computer. *Wiss. Z. Technol. Hochsch. Leuna-Meuseburg* **1985**, 27.

(32) Tiernan, J. An efficient search algorithm to find the elementary circuits of a graph. *Commun. ACM* **1970**, *13*, 722−726.

(33) Von Beiss, G.; Jänicke, W.; Meisller, H. G. Ein Algorithmus zur bestimmung aller elementarkreise eines gerichteten graphen. *Wiss. Z. Hachsch Leuma-Mersebug*. **1971**, *19*, 103−111.

(34) Danielson, G. H. On finding the simple paths and circuit in a graph. *IEEE Trans. Circuit Theory* **1968**, *15*, 294−295.

(35) Yau, S. S. Generation of all Hamiltonian circuits, catchs, and centers of a graph, and related problems. *IEEE Trans. Circuit Theory* **1967**, *14*, 79−81.

(36) Mateti, P.; Deo, N. On algorithms for enumerating all circuits of a graph. *SIAM J. Comput.* **1976**, *5*, 90−99.

(37) Tarjan, R. E. Enumeration of the elementary circuits of a directed grap. *SIAM J. Comput.* **1973**, *2*, 211−216.

(38) Bertziss, A. T.; *Data Structures: Theory and Practice*; Academic Press: New York, 1971.

(39) Corey, E. J.; Wipke, W. T.; Cramer, R. D.; Howe, W. J. Techniques for perception by a computer of synthetically significant structural features in complex molecules. *J. Am. Chem. Soc.* **1972**, *94*, 431−439.

(40) Balaban, A.; Filip, P.; Balaban, T. Computer program for finding all possible cycles in graphs. *J. Comput. Chem.* **1985**, *6*(4), 316−329.

(41) Hanser, T.; Jauffret, P.; Kaufmann, G. A new algorithm for exhaustive ring perception in a molecular graph. *J. Chem. Inf. Comput, Sci*. **1996**, *36*, 1146−1152.

(42) Cerruela García, G.; Luque Ruiz, I.; Gómez-Nieto, M. A. Un algoritmo en zigzag para la obtención de un conjunto de ciclos característicos de un grafo; Internal Report UCO-ISCBD-JCVG93-00; University of Córdoba: 2000.

(43) de Laet, A. SPECS and BioSPECS B.V., Fleminglaan, 16 NL-2289 CP Rijswijk, The Netherlands. http://www.specs.net.

(44) Deza, M.; Fowler, P. W.; Grishukhim, V. Allowed boundary sequences for fused polycyclic and related algorithm problems. *J. Chem. Inf. Comput. Sci*. **2001**, *41*(2), 300−308.

(45) Hansch, C.; Leo, A. *Exploring QSAR. Fundamentals and Applications in Chemistry and Biology*; ACS Professional Reference Book; American Chemical Society: Washington, DC, 1995.

(46) *MACCS Keys*; MDL Information Systems, Inc.: 14600 Catalina Street, San Leandro, CA 94577.

(47) Rücker, G.; Rücker, C. Topogical Indices, Boiling Points and Cycloalkanes. *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 788−802.

(48) Ravi, M.; Hopfinger, A. J.; Hormann, R. E.; Dinan, L. 4D-QSAR Analysis of a set of ecdysteroids and a comparison to CoMFA modeling. *J. Chem. Inf. Comput. Sci.* **2001**, *41*(6), 1587−1604.

(49) Venkatarangan, M. P.; Hopfinger, A. J. Prediction of ligand−receptor binding free energy by 4d-QSAR analysis: Application to a ser of glucose analogue inhibiyors of glycogen phosphorylase. *J. Chem. Inf. Comput. Sci.* **1999**, *39*(6), 1141−1150.

(50) McGregor, M. J.; Pallai, P. V. Clustering of large databases of compounds: using MDL keys as structural descriptors. *J. Chem. Inf. Comput. Sci.* **1997**, *37*, 443−448.

(51) Xue, L.; Bajorath, J. Accurate partitioning of compounds belonging to diverse activity classes. *J. Chem. Inf. Comput. Sci*. **2002**, *42*(3), 757−764.
(52) Dury, L.; Latour, T.; Leherte, L.; Barberis, F.; Vercauteren, D. A new graph descriptor for molecules containing cycles. Application as screening criterion for searching molecular structures within large databases of organic compounds. *J. Chem. Inf. Comput. Sci*. **2001**, *41*(6), 1437−1445.
(53) de Laet, A.; Hehenkamp, J. J.; Wife, R. L. Finding drug candidates in virtual and lost/emerging chemistry. *J. Heterocycl. Chem*. **2000**, *37*, 669−674.
(54) Lipkus, A. H. Mining a Large database fro Peptidomimetic Ring Structures Using Topological Index. *J. Chem. Inf. Comput. Sci.* **1999**, *39*(3), 582−586.
(55) Lipkus, A. H. Exploring chemical rings in a simple topological-descriptor space. *J. Chem. Inf. Comput. Sci*. **2001**, *41*(2), 430−438.
(56) Flower, D. R. On the properties of bit string-based measures of chemical similarity. *J. Chem. Inf. Comput. Sci*. **1998**, *38*, 379−386.
(57) Godden, J.; Xae, L.; Bajorath, J. Combinatorial preferences affect molecular similarity/diversity calculations using binary fingerprints and Tanimoto coefficients. *J. Chem. Inf. Comput. Sci*. **2000**, *40*, 163−166.
(58) Cerruela García, G.; Luque Ruiz, I.; Gómez-Nieto, M. A. Parallel algorithms for graph cycle extraction using the cyclical conjunction operator (accepted for publication to *J. Chem. Inf. Comput. Sci*.)

CI020373E