Isomorphism, Automorphism Partitioning, and Canonical Labeling Can Be Solved in Polynomial-Time for Molecular Graphs

Jean-Loup Faulon

Computational Materials Science Department, Sandia National Laboratories, Albuquerque, New Mexico 87185-1111

Received November 3, 1997

The graph isomorphism problem belongs to the class of NP problems, and has been conjectured intractable, although probably not NP-complete. However, in the context of chemistry, because molecules are a restricted class of graphs, the problem of graph isomorphism can be solved efficiently (i.e., in polynomial-time). This paper presents the theoretical results that for all molecules, the problems of isomorphism, automorphism partitioning, and canonical labeling are polynomial-time problems. Simple polynomial-time algorithms are also given for planar molecular graphs and used for automorphism partitioning of paraffins, polycyclic aromatic hydrocarbons (PAHs), fullerenes, and nanotubes.

INTRODUCTION

The graph isomorphism problem consists of deciding whether two given graphs are isomorphic; that is, whether there is a one-to-one mapping (a permutation) from the vertices of one graph to the vertices of the second graph, such that the edge connections are respected. An isomorphic mapping of the vertices onto themselves is called an automorphism. The set of all automorphisms of a given graph is the automorphism group of the graph. The automorphism group contains information regarding the topological symmetry of the graph. In particular, the orbits of an automorphism group identify symmetrical vertices. The canonical labeling problem consists of finding a unique labeling of the vertices of a given graph, such as all isomorphic graphs have the same canonical labels. Examples of canonical representations are graphs that maximize (or minimize) their adjacency matrix while preserving the graph connectivity. Two graphs with the same canonical representation are isomorphic.

Although most articles related to graph isomorphism have been published in the computer science literature, the computation of the orbits of automorphism groups using partitioning techniques has received most attention in chemistry. Nonetheless, the two problems have been shown to be computationally equivalent.[1] The canonical labeling problem has been studied both in chemistry and computer science. It would appear that the canonical labeling problem is closely related to the isomorphism testing problem; the latter can be performed at least as fast as the former, and for many published algorithms, isomorphism tests either include a procedure for canonization or else have an analogue for that problem.[2] However, some studies suggest that the two problems are not equivalent and that canonical labeling is computationally harder than isomorphism.[3]

As already mentioned, automorphism partitioning and canonical labeling are of significant interest in chemistry. Both problems have practical applications in (1) molecular topological symmetry perception for chemical information and quantum mechanics calculations, (2) computer-assisted structure elucidation, (3) NMR spectra simulation, and (4) database storage and retrieval, including determination of maximum common substructure. Since 1965,[4] many automorphism partitioning and canonical labeling methods have been proposed in the context of chemical computation. The general characteristic of these methods is the use of graph invariants to perform an initial vertex partitioning. Three techniques have been proposed to compute vertex invariants: (1) calculation of extended vertex connectivities through an iterative process, (2) computation of the higher powers of the graph adjacency matrix, and (3) determination of the adjacency matrix eigenvalues. All these techniques can be implemented efficiently (i.e., in polynomial-time) and have led to the development of fast algorithms. For most published algorithms, the initial vertex partitioning is followed by an exhaustive generation of all labelings. The computational complexity of the exhaustive generation can be reduced using the fact that two vertices with different invariants belong to different equivalent classes; hence, exhaustive labeling generation is performed only for vertices with the same invariant. Nonetheless, because all vertices may have the same invariant, the upper bound of the time-complexity for the exhaustive labeling generation scales exponentially with the number of vertices.

Whereas vertices with different invariants belong to different equivalent classes, the reverse is not necessarily true. As a matter of fact, isospectral points are vertices with the same invariant that belong to different classes. For some authors, this fact was overlooked, and the erroneous claim that automorphism partitioning could be solved solely by computing vertex invariants has been made more than once. These claims have initiated much debate in the literature.[5] Although the invariant approach may not be totally successful in the sense that the proposed methods work in all cases, it has been shown to behave well on averaged.[6,7] Indeed, for a given random graph there is a high probability that its vertices can be correctly partitioned using graph invariants.[8] Hence, probabilistic efficient algorithms could eventually be designed for automorphism partitioning; however, up to date, all the rigorous algorithms developed in the context of chemistry do not achieve polynomial-time scaling.

One has to be cautious when claiming efficiency for isomorphism, automorphism partitioning, or canonical labeling. As it has been demonstrated in graph theory, these problems are unlikely to be polynomial-time,[9] and unless specific algorithms are developed for molecular graphs, polynomial scaling cannot be achieved. Nonetheless, automorphism partitioning and canonical labeling algorithms have been claimed to scale polynomially. Liu and Klein[10] have proposed an O(N^3) canonical labeling algorithm. The algorithm is based on the computation of the eigenvalues of the graph adjacency matrix, and is *a priori* applicable to all types of graphs. A careful reading of the paper reveals that the proposed algorithm is not guaranteed to succeed, especially for highly symmetrical graphs. The reason computation of eigenvalues and eigenvectors is not sufficient for automorphism partitioning can be found in the Collatz and Sinogowitz article.[11] In a recent paper,[12] an algorithm for automorphism partitioning is proposed where vertices are partitioned according to their subspanning trees. Although the computational complexity of the algorithm is not evaluated by the authors, it can easily be shown that the algorithm is efficient because the problem of automorphism partitioning is reduced to a tree isomorphism problem. The algorithm makes use of two types of subspanning trees: a *maximum overlapping spanning tree* (MOST) involving for any level all the atoms present at a given topological distance, and a *minimum not overlapping spanning tree* (MINST) defined with the constraint that when an atom is used in a layer, it cannot be present again in the next layer. The algorithm is based on the following statement "two equivalent atoms must have identical subspanning trees (MOST and MINST)", which the authors claim to be necessary and sufficient for automorphism partitioning. Although the authors sketched a proof of the aforementioned claim, the statement is necessary but cannot be sufficient because it reduces automorphism partitioning to tree isomorphism and consequently makes graph automorphism a polynomial-time problem. As compelling evidence that automorphism partitioning cannot be reduced to tree isomorphism, Figure 1 gives a counter example where two nonequivalent vertices have identical subspanning trees. As with Liu and Klein's method, this latter technique succeeds in most instances but fails in specific cases.

Broadbelt *et al.*[13] have proposed a canonical labeling technique based on Hopcroft-Tarjan[14] planar graph isomorphism algorithm. Broadbelt *et al.'s* technique is rigorous and achieves polynomial-time scaling because Hopcroft-Tarjan algorithm scales as O(NlogN).

However, the technique is limited to graphs that are either trees, biconnected but not triconnected graphs (i.e., monocyclic compounds), or a combination of both. The canonical code is obtained using Hopcroft-Tarjan's technique for tree isomorphism, which consists of sorting the decomposition tree of the graph. The code for a given biconnected component is the lexicographically minimum code generated when taking all the possible directions of traversal of the perimeter of the component (i.e., clockwise or counterclockwise). The authors attempt to process polycyclic components by appending the codes of the interior atoms to the code of

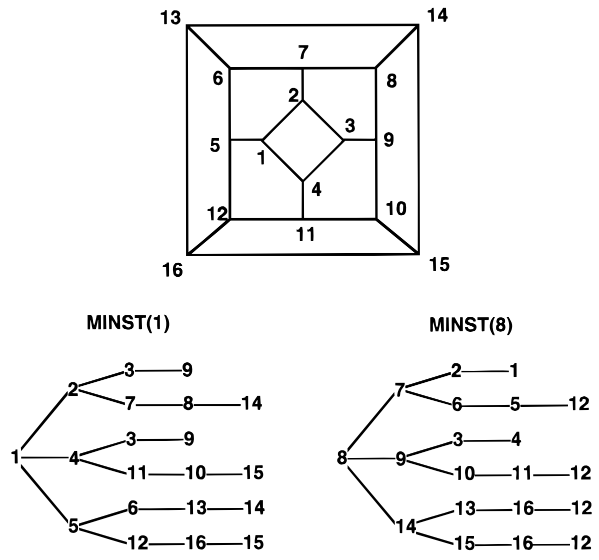
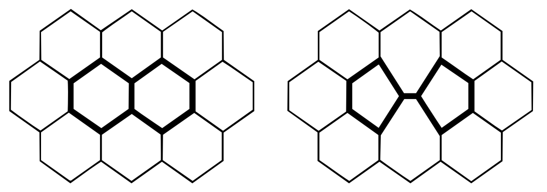


Figure 1. Subspanning trees and automorphism partitioning. The minimum not overlapping spanning trees (MINST) are constructed as described in ref 12. Although vertices 1 and 8 are not automorphic they have the same MINST. Vertices 1 and 8 also have the same maximum overlapping spanning tree (MOST) because all vertices of the graph have the same degree. Thus, the two vertices are considered equivalent by the algorithm outlined in ref 12. In fact, all the vertices of Figure 1 have the same MINST and MOST because vertices 2, 3, 4, 13, 14, 15, and 16 are automorphic to vertex 1, and vertices 6, 7, 9, 10, 11, and 12 are automorphic to vertex 8. According to the description given in their article, Fan et al.'s automorphism partitioning algorithm should output only one class of vertices, even if the calculations of MINST and MOST are repeated through an iterative process.



CC(H)C(H)CC(H)C(H)CC(H)C(H)CC(H)C(H)CC(H)C(H)CC(H)C(H)CC(H)C(H)CC(H)C(H)C:C:C:C:C:C:C:C:C:C:

Figure 2. Canonical code for polycyclic hydrocarbons. It is assumed that each vertex is a trivalent atom. Hydrogen atoms are not drawn. The canonical code is computed as described in ref 13. The code of interior atoms is added to the code of perimeters atoms. Although the two graphs are nonisomorphic, they have the same code.

the perimeter atoms. As shown in Figure 2, this procedure fails for polycyclic components with the same number of atoms in their interiors. Hence, polyaromatic hydrocarbons cannot be properly processed, and the algorithm fails with fullerenes because these compounds do not contain perimeter atoms.

Independently of chemical applications, the graph isomorphism problem has been studied in computer science. Important results have been obtained that remain mostly ignored by the chemistry community. As already mentioned, up to date, no efficient algorithm exists for the graph isomorphism problem. Hence, it is natural to ask whether such an algorithm exists at all or whether, on the contrary, the problem is intractable. The theory of NP-completeness classifies intractable problems. The graph isomorphism problem lies in the class NP, an therefore the question arises whether it is in P (i.e., can be solved in polynomial-time) or whether it is NP-complete (which would prove that the problem is intractable). It has been shown[15] that if $P \neq NP$, then there exist problems that are neither in P nor NP-complete. There is evidence[9] that graph isomorphism is a candidate for such an intermediate problem in NP. From an algorithmic point of view, the fact that graph isomorphism lies outside P is bad news because it cannot be solved in polynomial-time no matter whether it is NP-complete or not. The good news is that graph isomorphism can be solved efficiently for restricted classes of graphs. For instance, it is well known that isomorphism of trees[14] and planar graphs[16] can be performed with linear-time algorithms. An efficient graph isomorphism algorithm has also been derived for partial *k*-trees.[17] Partial *k*-trees are graphs with a treewidth less than *k*. Graph isomorphism[18,19] and canonical labeling[20] have been proven to be polynomial-time problems for graphs of bounded valence. Graphs of bounded valence are graphs where the degree of each vertex is bounded by a constant. Finally, a linear-time algorithm has been given for graphs of bounded average genus.[21] The genus of a graph, g_k is the number of embeddings in the topological surface S_k, where S_k is a torus with *k* holes (S_0 is a sphere). The average genus of a graph is the value $\sum k g_k / \sum g_k$.

The question is whether or not we can make use of the aforementioned results in the context of chemistry.

ISOMORPHISM, AUTOMORPHISM PARTITIONING, AND CANONICAL LABELING ARE POLYNOMIAL-TIME PROBLEMS FOR MOLECULAR GRAPHS

Most molecular graphs are planar: DNA, RNA, proteins, paraffins, olefins, polyaromatic hydrocarbons, fullerenes, and dendrimers have planar representations. However, there exist molecules, such as inorganic frameworks (zeolites) and cross-linked polymer networks (epoxies), for which no planar representation can be drawn. Therefore, the polynomial-time algorithms for isomorphism of planar graphs are not applicable to all molecules. Some molecular graphs may not be planar, but all molecules are bounded valence graphs, because the number of bonds formed by any given atom is always limited. Hence, the computational complexity results for isomorphism and canonical labeling of bounded valence graphs should hold true for all molecules. Yet, a technical difficulty arises because these results were obtained for simple graphs (graphs without double or triple bonds) in which the vertices do not correspond to chemical elements. In fact, this difficulty exists when any algorithm developed in the context of graph theory is applied to chemical compounds. To derive a polynomial time algorithm for molecular graphs, we first need to transform molecules into simple graphs.

Before proceeding further, it is necessary to characterize a molecule using graph theory terminology. A molecular graph is a multigraph without loops (an atom is not bonded to itself), which is colored by the elements of the periodic table. In other words, with a molecular graph, vertices have colors and bonds have orders, whereas with a simple graph, every vertex has the same color and every bond has an order equal to 1. As shown in Figure 3, bond order and vertex color have direct implications on isomorphism. To transform molecules into simple graphs for isomorphism purposes, one has to find a systematic way of removing colors and bond

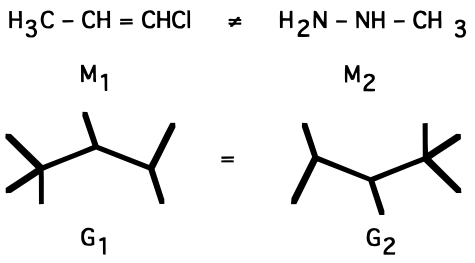


Figure 3. Molecules are colored multigraphs. Molecular graphs M_1 and M_2 are nonisomorphic because of the mutiplicities of the bonds and the color of the vertices. However, graphs G_1 and G_2 are isomorphic.

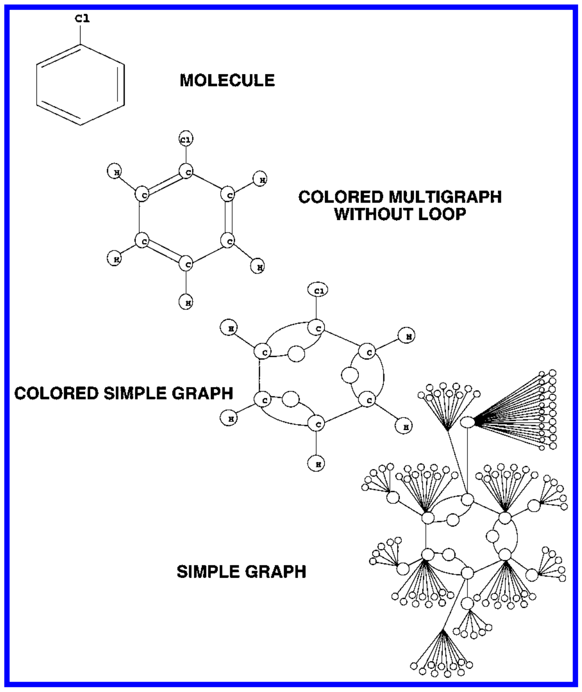


Figure 4. From molecular graphs to simple graphs.

orders while guaranteeing a one-to-one mapping between the molecular graph and the transformed graph.

Let $G_M = (A,B)$ be a molecular graph, and $G_S = (V,E)$ its corresponding simple graph. Let $Z_0 \geq MAX(2, d)$ be a constant equal to or greater than *2* and the maximum valence *d*, of the atom of G_M. An acceptable value for most organic compounds is $Z_0 = 4$. Let $Z(a)$ be the color of atom *a*, that is, the atomic number of *a* in the periodic table, and let order-(*b*) be the order of bond *b*. To remove the colors of a molecular graph, dummy vertices are attached to each atom. The number of dummy vertices attached to a given atom *a* is equal to $Z(a) + Z_0$. Consequently, the degree of all transformed atoms is at least $Z(a) + Z_0 \geq 5$ (if $Z_0 = 4$), and the degree of any dummy vertex is equal to 1. It is important to note that two vertices corresponding to different elements will have different degrees in the transformed graph. Double bonds are removed by adding a dummy vertex between the corresponding atoms. Triple bonds are removed by adding two dummy vertices. Note that the dummy vertices used to remove the double and triple bonds have a degree equal to 2. The transformation procedure is illustrated in Figure 4 and summarized in Scheme 1.

The vertices created during the transformation just presented need to be labeled for computer storage purposes. The following labeling system was adopted. Each of the *N*

Scheme 1

```
Input: set A of atoms, set B of bonds
Output: set V of vertices, set E of edges
V = ∅,  E = ∅
For each atom a in A do
      add vertex v = (a,0,0) to V
   For i = 1 to Z(a) + Z0  do
      add vertex vi = (a,0,i) to V
      add edge [v,vi] to E
   done
done
For each bond b = [a1,a2] in B do
   let v1 be the vertex labeled (a1,0,0)
   let v2 be the vertex labeled (a2,0,0)
   d = order(b)
   while (d ≥ 2) do
         add vertex v = (a1,a2,0) to V
         add edges [v1,v] and [v,v2] to E
         d = d - 1
   done
   add edge [v1,v2] to E
done
```

Scheme 2

```
Input: set V of vertices, set E of edges
Output: set A of atoms, set B of bonds
A = ∅,  B = ∅
For each vertex v in V do
   if (deg(v) > Z0) then
      add v to A
      color(v) = deg(v) - Z0
   endif
done
For each edge e = [v1,v2] in E do
   if (deg(v1) > Z0 and (deg(v2) > Z0) then
      /* comment: v1 and v2 belong to A */
      add b = [v1,v2] to B
      order(b) = 1
   endif
done
For each edge e = [v1,v2] in E do
   if (deg(v1) > Z0 and deg(v2) = 2) then
      /* comment: only v1 belongs to A */
      a1 = v1, a2 = Γ(v2) - v1
      order([a1,a2]) = order([a1,a2]) + 0.5
   else if (v2 belongs to A and deg(v1) = 2) then
      (comment: only v2 belongs to A)
      a1 = v2, a2 = Γ(v1) - v2
      order([a1,a2]) = order([a1,a2]) + 0.5
   endif
done
```

atoms of the molecular graph is arbitrarily labeled with an integer a, $1 \leqslant a \leqslant N$. Each vertex of G_S corresponding to an atom a is labeled $(a, 0, 0)$. Each dummy vertex added when removing atom colors is labeled $(a, 0, k)$ where a is the corresponding adjacent atom, and $k = 1, ..., Z(a) + Z_0$. Each dummy vertex added when removing a multiple bond is labeled (a_1,a_2,k) where a_1 and a_2 are the atoms adjacent to the multiple bond and $k = 1, 2$.

The inverse transformation (Scheme 2), which consists of generating a molecular graph from a given simple graph, relies on the fact that all dummy atoms have a degree equal to 1 or 2. Hence, the removal of all vertices having a degree equal to, or less than 2 leads to the corresponding molecular graph. In the inverse transformation, $\Gamma(v)$ is the set of all vertices attached to vertex v.

Next, we prove the one-to-one mapping between molecular graphs and simple graphs obtained using Schemes 1 and 2 by showing that two molecular graphs $G_{M1} = (A_1, B_1)$ and $G_{M2} = (A_2, B_2)$ are isomorphic ($G_{M1} \cong G_{M2}$) if and only if their corresponding simple graphs $G_{S1} = (V_1, E_1)$ and $G_{S2} = (V_2, E_2)$ are isomorphic ($G_{S1} \cong G_{S2}$). For simple graphs we use the standard isomorphism definition: $G_{S1} \cong G_{S2}$ if and only if there is a mapping π from V_1 to V_2, such that: $E_2 = \pi(E_1)$. Molecular graph isomorphism is redefined as follows: $G_{M1} \cong G_{M2}$ if and only if a mapping π from A_1 to A_2 can be found, such that the three following conditions are verified: (i) $B_2 = \pi(B_1)$; (ii) for all atoms a_1 in A_1 and a_2 in A_2 if $a_2 = \pi(a_1)$ then $Z(a_2) = Z(a_1)$; and (iii) for all bonds b_1 in B_1 and b_2 in B_2 if $b_2 = \pi(b_1)$ then order $(b_2) =$ order(b_1).

Proposition I: if $G_{M1} \cong G_{M2}$ then $G_{S1} \cong G_{S2}$. *Proof.* Let us recall that the vertices of a transformed molecular graph are labeled with the triplets (a_i, a_j, k) where a_i is an atom, a_j is either 0 or an atom, and $k = 0, 1$, or 2. Let π be an isomorphism function between G_{M1} and G_{M2}. We construct the function π^* from V_1 to V_2 such that: $\pi^*(a_i, a_j, k) = (\pi(a_i), \pi(a_j), k)$ with $\pi(0) = 0$. Evidently, if $B_2 = \pi(B_1)$ then $E_2 = \pi^*(E_1)$, and π^* is an isomorphism between G_{S1} and G_{S2}.

Proposition II: if $G_{S1} \cong G_{S2}$ then $G_{M1} \cong G_{M2}$. *Proof.* Let π^* be an isomorphism function such that $E_2 = \pi^*(E_1)$. The set of edges E of any transformed graph can be partitioned into three sets, E^1 the set of edges attached to vertices of degree one, E^2 the set of edges attached to vertices of degree two, and E^{zo} the set of edges attached to two vertices of degrees greater than or equal to Z_0. E^{zo} is the only set related to the bonds of the corresponding molecular graph. If $E_2 = \pi^*(E_1)$ then $E^{zo}{}_2 = \pi^*(E^{zo}{}_1)$, and consequently $B_2 = \pi^*(B_1)$.

Let v_1 and v_2 be two vertices of V_1 and V_2 such that $v_2 = \pi^*(v_1)$. We assume that v_1 and v_2 have degrees equal to or greater than Z_0. Because $E_2 = \pi^*(E_1)$, then $\Gamma_1(v_2) = \pi^*(\Gamma_1(v_1))$, where $\Gamma_1(v_1)$ is the set of vertices of degree 1 attached to $v_1\Gamma_1(v_2)$ is the set of vertices of degree 1 attached to v_2. Consequently, $|\Gamma_1(v_2)| = |\pi^*(\Gamma_1(v_1))| = |\Gamma_1(v_1)|$ and $Z(v_1) = |\Gamma_1(v_1)| - Z_0 = Z(v_2)$.

Let $[v_{11},v_{12}]$ and $[v_{21},v_{22}]$ be two bonds of $E^{zo}{}_1$ and $E^{zo}{}_2$ such that $[v_{21},v_{22}] = \pi^*([v_{11},v_{12}])$. We note by $\Gamma_2(v)$ the set of vertices of degree 2 attached to v. Because $[v_{21},v_{22}] = \pi^*([v_{11},v_{12}])$, we have $\Gamma_2(v_{21}) \cap \Gamma_2(v_{22}) = \pi^*(\Gamma_2(v_{11}) \cap \Gamma_2(v_{12}))$, and consequently $|\Gamma_2(v_{21}) \cap \Gamma_2(v_{22})| = |\Gamma_2(v_{11}) \cap \Gamma_2(v_{12})|$. Because $|\Gamma_2(v_{11}) \cap \Gamma_2(v_{12})| + 1$ and $|\Gamma_2(v_{21}) \cap \Gamma_2(v_{22})| + 1$ are the orders of the bonds corresponding to $[v_{11},v_{12}]$ and $[v_{21},v_{22}]$, these bonds have the same order.

We have proven the three conditions required for molecular graph isomorphism.

The computational complexity of Schemes 1 and 2 is trivial to evaluate. In both schemes the vertices and the bonds are visited at most twice, therefore Schemes 1 and 2 have a linear-time complexity. The number of vertices added in Scheme 1 or removed in Scheme 2 is bounded by $(Z_{max} + Z_0)\ N_A + N_2 + 2N_3$, where N_A is the number of atoms, N_2 is the number of double bonds, N_3 the number of triple bonds, and Z_{max} is the maximum atomic number of the tested molecule. Note that the total number of vertices in a transformed molecular graph is lower than $N_V = (Z_{max} + Z_0 + 1)\ N_A + N_2 + 2N_3$, and is therefore linearly proportional to the number of atoms (i.e., $N_V = O(N_A)$). The number of vertices added in the transformed graph ($N_V - N_A$) can be reduced by substituting each atomic number Z, by an integer Z′, whose maximum value is the number of atom types in the studied molecule. For instance, if the studied molecule contains hydrogen, carbon, and chlorine, one may choose $Z' = 1$ for all hydrogen atoms, $Z' = 2$ for all carbon atoms, and $Z' = 3$ for all chlorine atoms. Finally, it is interesting to note that Schemes 1 and 2 neither add nor remove cycles. Hence, if the input graph is a tree, the graph output by either scheme will also be a tree. The statement is also true for the planarity of graphs.

The consequence of the aforementioned schemes and propositions is that isomorphism, automorphism partitioning, and canonical labeling can be solved in polynomial-time for molecular graphs. To test isomorphism, Scheme 1 is applied on the two-tested molecular graphs with the time complexity $O(N_V) = O(N_A)$. Planarity is tested on the transformed graphs, this can be done with a running time of $O(N_V)$.[22] If the two graphs are planar, a planar drawing can be obtained with the time complexity $O(N_V)$ using the Hopcroft-Tarjan algorithm.[23] Equally fast, the isomorphism test is then performed using the Hopcroft-Wong algorithm in running time $O(N_V)$.[16] The overall time complexity for testing isomorphism on planar graphs is therefore $O(N_V) = O(N_A)$. When the two tested graphs are nonplanar, Luks and Hoffman[18,19] complexity results give the running time $O(N_V{}^{h(d)})$, where d is the maximum valence of the atoms, and $h(\ )$ is a polynomial function. However, the aforementioned complexity results are theoretical, and practical computer codes still remain to be developed. Following Read and Corneil,[1] automorphism partitioning is equivalent to isomorphism testing; therefore, the expected complexity is $O(N_V)$ for planar molecular graphs and $O(N_V{}^{h(d)})$ for nonplanar graphs. Neither Hopcroft-Tarjan[14] nor Hopcroft-Wong[16] algorithms are directly applicable for automorphism partitioning, so a polynomial-time algorithm is given in the next section for automorphism partitioning of planar molecular graphs. Finally, canonical labeling of molecular graph is also a polynomial-time problem. According to Babai and Luks,[20] canonical labeling of a bounded valence graph has an expected running time of $O(N_V{}^{h(d)})$, where $h(d) = (d - 1) \log (d - 1) + c$, and c is a constant. A canonical labeling algorithm for planar molecular graphs is given in the present paper.

POLYNOMIAL-TIME AUTOMORPHISM PARTITIONING ALGORITHM FOR PLANAR MOLECULAR GRAPHS

In this section, the extended connectivity concept[24] popular in the computational chemistry literature is investigated for planar molecular graphs. Although originally introduced by Morgan[4] for canonical labeling computations, the extended connectivity concept has been used by many authors to derive automorphism partitioning algorithms.[25−31] Extended connectivity algorithms compute graph invariants. As stated by Read and Corneil,[1] the temptation to suppose that from these invariants we can derive some complete graph invariant is strong, but it does not appear to be true for all graphs. Indeed, the extended connectivity technique fails to compute the correct automorphism partitioning in some instances.[5] Nonetheless, as will be seen in this section, the extended connectivity technique may be modified to compute complete graph invariants for planar molecular graphs. Based on this modification, the first polynomial-time algorithm for automorphism of planar molecular graphs is outlined. The correctness and time complexity upper bound of the algorithm are rigorously proven. Additionally, practical time complexity is illustrated for several families of hydrocarbons in the last section of the paper.

Let G_M be the studied molecular graph, $G_S = (V, E)$ its corresponding simple graph derived using Scheme 1, and $D_S = (V, A)$ the digraph corresponding to G_S. Let e be an edge of E and v,w its adjacent vertices; then, D_S is constructed by replacing every edge e by two directed edges (also called arcs) $a = [v,w]$ and $a^r = [w,v]$. If $a = [v,w]$ is an arc, then v is the tail of a, and w is the head of a. Note that the size of A is twice the size of E. The main task of the proposed algorithm is to partition the arcs of D_S using graph invariants. Edge and vertex partitionings are computed in a post process. Let $\lambda(v)$ be the invariant attached to vertex v, $\lambda(e)$ the invariant of an edge e in E, and $\lambda(a)$ the invariant corresponding to an arc a in A. We use the notation $\lambda(A)$ for the set of nonidentical invariants of all arcs of A. Following Morgan,[4] arc partitioning is achieved by taking into account the extended connectivity (i.e., two arcs have the same invariants if their neighbors have the same invariants). This definition being recursive, the algorithm proceeds until the total number of arc invariants $|\lambda(A)|$ remains unchanged. Prior to computing edge invariants, we notice that two edges are automorphic if and only if their respective arcs are automorphic. Hence, edge invariants need to be computed in such a way that two edges have the same invariants if and only if their arcs have identical invariants. Once edges have been partitioned, vertex invariants are computed using the fact that two vertices have the same invariants only if there is a mapping between their two sets of edge invariants. In the following, $\Gamma(v)$ is the connectivity of vertex v (i.e., the set of all edges adjacent to v). $\Gamma(v)$ can be partitioned into two sets, $\Gamma_{in}(v)$ and $\Gamma_{out}(v)$, which correspond respectively to the set of arcs whose heads are v and the set of arcs whose tails are v. Using this notation, $\Gamma_{in}(\text{head}(a)) \cup \Gamma_{out}(\text{head}(a)) \cup \Gamma_{in}(\text{tail}(a)) \cup \Gamma_{out}(\text{tail}(a))$ is the set of all arcs attached to a. Note that a and a^r are included twice in this set because a belongs to $\Gamma_{in}(\text{head}(a))$ and $\Gamma_{out}(\text{tail}(a))$, and a^r belongs to $\Gamma_{out}(\text{head}(a))$ and $\Gamma_{in}(\text{tail}(a))$. The main algorithm is given in Scheme 3.

In the remaining portion of this section, the initialization function (init-arc-invariant) and computation functions (compute-arc-invariant, compute-edge-invariant, and compute-vertex-invariant) will be first detailed for trees, then extended for cyclic planar graphs, and finally generalized for all planar graphs.

Trees. For trees, the initial arc invariant is computed from the pair of degrees of the head and tail of the arc (see Scheme 4).

In Scheme 4, as well as in all the following schemes, the function MAP performs a one-to-one mapping between an n-tuple ($n = 2$ in Scheme 4) and the set of positive integers. An example of such a function is:

$$\text{MAP}(x_1, x_2, ..., x_n) = x_1 + x_2\,M + ... + x_n\,M^{n-1}$$

where M is defined such that for all n, $x_n < M$. Because the total number of invariants is bounded by the total number of arcs, $M = |A| + 1$. More sophisticated MAP functions can be constructed, where an integer varying from 1 to M is assigned to each different n-tuple. This procedure, however, necessitates maintainenance of a sorted list of n-tuples. Note that the MAP function used by Morgan in its original article[4]

Scheme 3

```
partition(V,E)
/* V is a set of vertices
   E is a set of edges
   local variables:
   - set of arcs A
   - sets of invariants λ and λ'
   - integers N and N'
         */

   compute the set of arcs A from E
   For each arc a in A do λ(a) = init-arc-invariant(a)

   N = |λ(A)|, N' = -1
   while (N ≠ N') do
     N' = N, λ'=λ
     For each arc a in A do
         λ(a) = compute-arc-invariant(a,{Γ_out(head(a))-a^r},λ')
     done
     N = |λ(A)|
   done

   For each edge e in E do λ(e) = compute-edge-invariant(e, λ)
   For each vertex in V do λ(v) = compute-vertex-invariant(v,Γ(v),λ)

end
```

Scheme 4

```
init-arc-invariant-tree(a)
  /* a is an arc
     */
  invariant = MAP(deg(head(a)),deg(tail(a)))
  return(invariant)
end
```

does not guarantee a one-to-one mapping between *n*-tuples and integers; Morgan's MAP function consists of summing the coordinates of the *n*-tuples. The initialization procedure can easily be modified and enables one to run Scheme 3 directly on molecular graphs rather than transformed graphs. The advantage of this modification is to avoid having to use Scheme 1, and in turn to decrease the number of vertices, edges, and arcs processed by Scheme 3. The modification consists of computing the initial invariant from the pair of degrees *and* the atomic numbers of the head and tail of the arc, *and* from the bond order of the arc.

After initialization, the invariant of each arc is updated iteratively by sorting the invariants of its neighbors and applying the MAP function on the resulting sorted *n*-tuples. Once arc invariant calculations have converged, edge invariants are computed using the MAP function with the invariants of the corresponding arcs. Finally, vertex invariants are computed using the MAP function with the sorted *n*-tuples of edge invariants adjacent to the studied vertices. Scheme 5 gives the details of the computation functions for trees.

Although several algorithms based on extended connectivity have been developed in the context of chemistry,[4,25−31] to the best of the author's knowledge none of them have been mathematically proven to be correct. The purpose of the next proposition and corollary is to prove the correctness of the partition algorithm applied to trees, and in turn, to demonstrate that extended connectivity is a valid concept when applied to noncyclic molecular graphs.

Proposition III: upon termination of Scheme 3 applied to trees, two arcs *a* and *b* are automorphic if and only if $\lambda(a) = \lambda(b)$ and $\lambda(a^r) = \lambda(b^r)$. *Proof.* Two arcs *a* and *b*

Scheme 5

```
compute-arc-invariant-tree(a, Γ, λ)
  /* a is an arc
     Γ is the set of arcs whose tail is the head of a
     λ is the set of invariants
     */
  L = λ(Γ)
  sort L in lexicographic order
  invariant = MAP(λ(a),L)
  return(invariant)
end

compute-edge-invariant-tree(e, λ)
  /* e is an edge
     λ is the set of invariants
     */
  v1, v2 = vertices adjacent to e
  a12 = [v1,v2],     a21 = [v2,v1]
  λ12 = λ(a12),      λ21 = λ(a21)
  invariant = MAP(MIN(λ12, λ21),MAX(λ12, λ21))
  return(invariant)
end

compute-vertex-invariant-tree(v, Γ, λ)
  /* v is a vertex
     Γ is the set of edges adjacent to v
     λ is the set of invariants
     */
  L = λ(Γ)
  sort L in lexicographic order
  invariant = MAP(L)
  return(invariant)
end
```

are automorphic if and only if (iff) (i) they have the same initial invariant, (ii) the subtree attached to the head of *a* (T(head(*a*))) is isomorphic to the subtree attached to the head of *b* (T(head(*b*))), and (iii) the subtree attached to the tail of *a* (T(tail(*a*))) is isomorphic to the subtree attached to the tail of *b* (T(tail(*b*))). We have to prove that these three conditions are verified iff $\lambda(a) = \lambda(b)$ and $\lambda(a^r) = \lambda(b^r)$.

•Let us first prove that if $\lambda(a) = \lambda(b)$ and $\lambda(a^r) = \lambda(b^r)$, then conditions (i), (ii), and (iii) are verified.

•*Condition (i).* At each iteration, invariants are computed using the invariants of the previous iteration. Therefore, upon termination of Scheme 3, if two arcs have the same invariant then they had the same initial invariant.

Scheme 6

```
partition-tree(V,E)
/* V is a set of vertices
   E is a set of edges
   local variables:
   - set of arcs A,L[]
   - sets of invariants λ and λ'
   */

   compute the set of arcs A from E
   compute-arc-layer-tree(A,L[])
   For each arc a in A do λ(a) = init-arc-invariant-tree(a)
   i = 1
   while (L[i] ≠ ∅) do
     For each arc a in L[i] do
       λ(a) = compute-arc-invariant-tree(a,{Γ_out(head(a))-a^r},λ)
     done
     i = i+1
   done

   For each edge e in E do λ(e) = compute-edge-invariant-tree(e, λ)
   For each vertex in V do λ(v) = compute-vertex-invariant-tree(v,Γ(v),

end

compute-arc-layer-tree(A,L[])
/* A and L[] are sets of arcs
   local variables:
   - set of integers (layers) l
   */
   For each arc a in A do l(a) = |A|+1
   L[1] = {arc a in A, such that Γ_out(head(a)) = ∅}
   i = 1
   while (L[i] ≠ ∅) do
     L[i+1] = ∅
     For each arc a in L[i] do l(a) = i
     For each arc a in L[i] do
         For each arc a' in {Γ_in(tail(a))-a^r} such that l(a') > i do
             if each arc a" in {Γ_out(head(a'))-[a']^r}
                verifies l(a") ≤ i then L[i+1] = L[i+1] U a'
             endif
         done
     done
     i = i+1
   done
end
```

•*Condition (ii).* We show by induction that T(head(a)) and T(head(b)) are isomorphic if $\lambda(a) = \lambda(b)$. In the following, the invariants λ_i, $0 \leq i \leq n$ are represented using the n-tuple notation. This representation can be computed by the partition algorithm on removal of all calls to the MAP function. Let the layer of an arc a be the maximum distance between a and all the leaves of the subtree attached to head(a). All arcs a, whose heads are leaves ($\Gamma_{out}(a) = \emptyset$), are located in layer 1. The induction hypothesis is that for two arcs a_i and b_i located in layer i verifying $\lambda_i(a_i) = \lambda_i(b_i)$, the subtrees T(head($a_i$)) and T(head($b_i$)) are isomorphic. The hypothesis is true at layer i = 1, because the subtrees are empty. Let a_{i+1} and b_{i+1} be two arcs located in layer i + 1, verifying $\lambda_{i+1}(a_{i+1}) = \lambda_{i+1}(b_{i+1})$. According to Scheme 5 we have:

$$\lambda_{i+1}(a_{i+1}) = (\lambda_i(a_{i+1}), \lambda_i(\Gamma_{out}(a_{i+1}))) = (\lambda_i(b_{i+1}), \lambda_i(\Gamma_{out}(b_{i+1}))) = \lambda_{i+1}(b_{i+1})$$

Therefore:

$$\lambda_i(\Gamma_{out}(a_{i+1})) = \lambda_i(\Gamma_{out}(b_{i+1}))$$

Let $\Gamma_{out}(a_{i+1}) = (a_i^1, a_i^2, ..., a_i^k)$ and $\Gamma_{out}(b_{i+1}) = (b_i^1, b_i^2, ..., b_i^k)$; then, the previous equation is equivalent to:

$$\lambda_i(a_i^k) = \lambda_i(b_i^k) \text{ for all k}$$

According to the induction hypothesis, if this last equation is verified, then the subtree attached to the head of a_i^k is isomormorphic to the subtree attached to head of b_i^k; that is, an isomorphism π^i can be found, such that for all k, π^i(T(head(a_i^k))) = T(head(b_i^k)). Let π^{i+1} be a function such that $\pi^{i+1}(a^k) = \pi^i(a^k)$ for all arcs a^k in T(head(a_i^k)), and $\pi^{i+1}(a_i^k) = \pi^{i+1}(b_i^k)$. π^{i+1} is an isomorphism between T(head(a_{i+1})) and T(head(b_{i+1})) because π^i is an isomorphism and condition (i) is verified (two arcs are isomorphic if they have the same initial invariant).[32] We have proven the induction hypothesis and consequently condition (ii).

•*Condition (iii).* Because T(tail(a)) = T(head(a^r)) and T(tail(b)) = T(head(b^r)), the same proof by induction when applied to T(tail(a)) and T(tail(b)) leads to the statement that T(tail(a)) and T(tail(b)) are isomorphic if $\lambda(a^r) = \lambda(b^r)$.

•We now prove that if two arcs a and b are isomorphic, then upon termination of Scheme 3, $\lambda(a) = \lambda(b)$ and $\lambda(a^r) = \lambda(b^r)$.

•From condition (i) we derive $\lambda_0(a) = \lambda_0(b)$ and $\lambda_0(a^r) = \lambda_0(b^r)$.

·From condition (ii) we know that the subtrees T(head(a)) and T(head(b)) are isomorphic; therefore, $\Gamma_{out}(a)$ is isomorphic to $\Gamma_{out}(b)$. This relationship, according to the computation of arc invariants, leads to $\lambda_1(a) = (\lambda_0(a), \lambda_0(\Gamma_{out}(a))) = (\lambda_0(b), \lambda_0(\Gamma_{out}(b))) = \lambda_1(b)$. We also find that $\lambda_1(\Gamma_{out}(a)) = \lambda_1(\Gamma_{out}(b))$, because $\Gamma_{out}(\Gamma_{out}(a))$ is isomorphic to $\Gamma_{out}(\Gamma_{out}(b))$. Hence, $\lambda_2(a) = (\lambda_1(a), \lambda_1(\Gamma_{out}(a))) = (\lambda_1(b), \lambda_1(\Gamma_{out}(b))) = \lambda_2(b)$. This process can be repeated until all nodes of the subtrees have been visited; if n is the corresponding iteration number we have $\lambda_n(a) = (\lambda_{n-1}(a), \lambda_{n-1}(\Gamma_{out}(a))) = (\lambda_{n-1}(b), \lambda_{n-1}(\Gamma_{out}(b))) = \lambda_n(b)$.

·Using the same arguments with condition (iii) we derive $\lambda(a^r) = \lambda(b^r)$.

Corollary I: upon termination of Scheme 3 applied to trees, two edges or two vertices are automorphic if they have the same invariant. *Proof.* Two edges g and h are automorphic iff their respective arcs are automorphic. Following the compute-edge-invariant-tree procedure, g and h have the same invariant iff their respective arcs have identical invariants. Because we have shown that two arcs have the same invariant iff they are automorphic, g and h are automorphic iff they have the same invariant. Two vertices v and w are automorphic iff $\Gamma(v)$ and $\Gamma(w)$ are automorphic. We have shown that edges are automorphic iff they have the same invariant, hence two vertices are automorphic if a mapping can be found between the n-tuples $\lambda(\Gamma(v))$ and $\lambda(\Gamma(w))$. The n-tuples are sorted in lexicographic order in compute-vertex-invariant-tree; thus, a mapping is found when $\lambda(v) = \text{MAP}(\lambda_{sorted}(\Gamma(v))) = \text{MAP}(\lambda_{sorted}(\Gamma(w))) = \lambda(w)$. Consequently, two vertices are automorphic iff they have the same invariant.

Next, we evaluate the computational complexity of the partition algorithm applied to trees. The set of arcs constructed in Scheme 3 can be computed in a time proportional to the number of edges N-1, where N is the number of vertices. The size of λ(A) is computed by sorting the invariants in lexicographic order; this can be achieved in time O(N) if radix or bucket sorting algorithms are used.[22] Computing the MAP function is performed with a constant number of steps, because all n-tuples have a maximum size equal to the number of arcs attached to any given vertex. Initialization (Scheme 4) and computation (Scheme 5) functions have a constant computational cost because the n-tuples manipulated have a maximum size and the MAP function has a constant computational cost. As outlined in the proof of proposition III, after iteration of n is completed, the automorphism partitioning is solved for all the arcs located at a distance $\leq n$ from the leaves of the tree. The total number of iterations required for convergence is therefore the maximum distance between two arcs, which is bounded by the total number of arcs, 2N-2. Thus, for noncyclic molecular graphs, the upper bound time complexity of Scheme 3, and more generally the time complexity of all Morgan type algorithms, is O(N^2).

Scheme 3 can be optimized by noticing that at iteration n, invariants need to be computed only for the arcs belonging to the nth layer of the tree. From the proof of proposition III, it can be seen that it is not necessary to compute the invariants of arcs belonging to layers lower than n because for these arcs $\lambda_n = \lambda_{n-1}$. Furthermore, there is no need to compute arc invariants for layer n+1 and above, before completing invariant calculations at layer n because invariants at layer n+1 are computed from the invariants of layer n. The optimized algorithm (Scheme 6) consists of first computing the layer of each arc and then computing invariants layer by layer. The layer calculations are based on the following. All arcs a whose heads are leaves are in layer 1. Any given arc a', belongs to layer i+l, if all its predecessors (arcs $a'' \neq [a']^r$ that are attached to the head of a') are in layer i or below.

The proofs of Proposition III and Corollary I are directly applicable to Scheme 6, which guarantees the correctness of the partition-tree algorithm. Evaluating the computational complexity of Scheme 6 is straightforward. As with Scheme 3, the construction of A requires N − 1 steps, and initialization and computing procedures have a constant computational cost. Note that in compute-arc-layer-tree the loop

$$<<\text{For each arc } a' \text{ in } \{\Gamma_{in}(\text{tail}(a)) - a^r\}>>$$

has a constant computational cost because the total number of arcs attached to a is bounded (molecular graphs are bounded valence graphs). In compute-arc-layer-tree as well as in the main algorithm, each arc is visited only once, because any given arc belongs only to one layer. The total number of arcs processed is 2N − 2, and the overall time complexity of Scheme 6 is O(N).

Cyclic Planar Graphs. Cyclic planar graphs are biconnected planar graphs. A graph is biconnected if for each triple of distinct vertices v, w, and x there is a path from v to w such that x is not on the path. In other words, a graph is biconnected when there are at least two paths between two given vertices. Trees are not biconnected. Biconnected graphs include triconnected graphs, where there are at least three paths between two given vertices. Biconnected graphs that are not triconnected are called n-gons and these graphs consist of a cycle with n edges. In chemistry, examples of cyclic planar graphs are cycloalkanes (n-gons), fullerenes (triconnected), and polycyclic aromatic hydrocarbons (PAHs; combination of biconnected and triconnected).

At this point of the discussion, it is important to note that the partition algorithm applied to trees (Schemes 3−5) handles n-gons correctly. According to Proposition III, two arcs are automorphic when they have the same initial invariant and when the subtrees attached to the heads and tails of the arcs are isomorphic. In the case of an n-gon, the two subtrees attached to the head and tail of a given arc are two linear chains containing all the arcs of the n-gon ordered clockwise and counterclockwise. According to Broadbelt et al.,[17] the clockwise/counterclockwise ordering is necessary and sufficient to compute the canonical labeling of an n-gon, and consequently, to partition the arcs of an n-gon (two arcs having the same label; i.e., the same invariant, are automorphic). Because we have an algorithm (Schemes 3−5) to process biconnected graphs that are not triconnected, the remainder of the discussion concerns triconnected graphs. Cyclic planar graphs that are composed of both biconnected and triconnected components will be investigated later when generalizing the partition algorithm to all planar molecular graphs.

The algorithm for triconnected graphs relies on the fact that any triconnected graph has only two planar representations, an arbitrary planar representation and its mirror

Scheme 7

```
init-arc-invariant-cyclic(a)
  /* a is an arc
     */
  v1 = head(a), v2 = tail(a)
  inv+ = MAP(deg(v1),deg(v2),deg($F_R$(a)),deg($F_L$(a)))
  inv- = MAP(deg(v1),deg(v2),deg($F_L$(a)),deg($F_R$(a)))
  invariant = (inv+,inv-)
  return(invariant)
end
```

image.[33] Several embedding algorithms[23,34] can be used to compute planar representations, one of which scales linearly with the number of vertices.[23] A planar representation can also be obtained by minimizing the potential energy of the studied molecular graph. It is unlikely that after being energy minimized, a planar molecular structure will include two bonds crossing in the same plane. This latter solution has the advantage of being practical for chemical applications; however, it is not rigorous because it may rely on force field approximations to compute the energy.

The purpose of the embedding procedure is to assign planar coordinates to each vertex in such a way that there is no crossing between edges. Using these coordinates, all edges can then be ordered clockwise or counterclockwise with respect to their adjacent vertices. Note that if the edges are ordered clockwise in an arbitrary planar representation, the same edges are ordered counterclockwise in the mirror representation. In a given planar representation, every arc *a* belongs to two planar faces, a right face F_R (*a*) and a left face F_L (*a*). Let deg(F_R(*a*)) and deg(F_L(*a*)) be the number of edges of the right and left faces of *a*. We define two invariants (λ^+, λ^-) associated with each arc *a*, and these invariants are computed from the quadruples [deg(head(*a*)), deg(tail(*a*)), deg(F_R(*a*)), deg(F_L(*a*))] and [deg(head(*a*)), deg(tail(*a*)), deg(F_L(*a*)), deg(F_R(*a*))]. If *a* is an arc in a given planar representation and *a*′ is the same arc in the mirror representation, then $\lambda^+(a) = \lambda^-(a')$. This observation is crucial when computing automorphism partitioning. For every planar graph with the symmetry σ, one possible automorphism is the reflection operation. Because having the same initial invariant is a necessary condition for automorphism, there is a need to compute two invariants for each arc; λ^+, that is, the invariant in the given planar representation, and λ^-, the invariant in the mirror representation. The initialization procedure is outlined in Scheme 7.

Note that if Scheme 3 is run with molecular graphs without using Scheme 1, atomic numbers and bond orders must be added to the *n*-tuples of the MAP function. Cyclic computation procedures (Scheme 8) differ from the procedures used for trees; the two invariants (λ^+, λ^-) are updated for each arc from the invariants of the adjacent arcs in the right and left faces. Let *a* be an arc; then, we write R(*a*) the arc that immediately follows *a* in the right face, and L(*a*) the arc that follows *a* in the left face. The operators R and L are illustrated in Figure 5a. Edge invariants are computed by sorting in lexicographic order the invariants of the corresponding arcs. Finally, vertex invariants are calculated by ordering adjacent edges clockwise and counterclockwise.

To prove the correctness of the partition algorithm applied to triconnected planar graphs, some terminology needs to be introduced. Let $\lambda_0(a)$ be the initial invariant of arc *a*, and let $\lambda_n(a)$ be the invariant of *a* after *n* iterations of the partition algorithm. Two arcs *a*, *b* are *similar* if they have

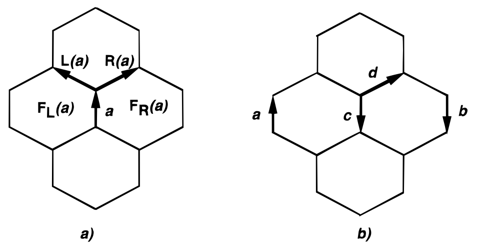


Figure 5. (a) Operators R and L. (b) Similar and indistinguishable arcs. Arcs *a* and *b* are similar, all their adjacent vertices have a degree of 2, their right faces have 6 vertices, and their left faces have 14 vertices (external face). Arcs *a* and *b* are indistinguishable. Any string of operators *R* and/or L applied to *a* and *b* leads to two similar arcs. Arcs *c* and *d* are similar because all their adjacent vertices have a degree of 2 and all their adjacent faces contain 6 vertices. However, arcs *c* and *d* are not indistinguishable because arcs R(*c*) and R(*d*) are not similar.

Scheme 8

```
compute-arc-invariant-cyclic(a, $\Gamma$, $\lambda$)
  /* a is an arc
     $\Gamma$ is the set of arcs whose tail is the head of a
     $\lambda$ = ($\lambda^+$,$\lambda^-$) is the set of pair-invariants
     */
  inv+ = MAP($\lambda^+$(a), $\lambda^+$(R(a)), $\lambda^+$(L(a)))
  inv- = MAP($\lambda^-$(a), $\lambda^-$(L(a)), $\lambda^-$(R(a)))
  invariant = (inv+,inv-)
  return(invariant)
end
compute-edge-invariant-cyclic(e, $\lambda$)
  /* e is an edge
     $\lambda$ = ($\lambda^+$,$\lambda^-$) is the set of pair-invariants
     */
  v1, v2 = vertices adjacent to e
  a12 = [v1,v2],    a21 = [v2,v1]
  L = ($\lambda^+$(a12), $\lambda^-$(a12), $\lambda^+$(a21), $\lambda^-$(a21))
  sort L in lexicographic order
  invariant = MAP(L);
  return(invariant)
end
compute-vertex-invariant-cyclic(v, $\Gamma$, $\lambda$)
  /* v is a vertex
     $\Gamma$ is the set of edges adjacent to v
     $\lambda$ is a set of invariants
     */
  L = $\lambda$($\Gamma$)
  L+ = L sorted turning counterclockwise with respect to v
  L- = L sorted turning clockwise with respect to v
  L = MIN (L+,L-)
  invariant = MAP(L)
  return(invariant)
end
```

the same initial invariant; that is, if one of the following two conditions is verified: $\lambda^+_0(a) = \lambda^+_0(b)$ or $\lambda^+_0(a) = \lambda^-_0(b)$. Note that if the first condition is true, then $\lambda^-_0(a) = \lambda^-_0(b)$, whereas if the second condition is true, then $\lambda^-_0(a) = \lambda^+_0(b)$.

Let X^+ be a series of operators R and/or L, and let X^- be the inverse series. For example, if $X^+ =$ RLRL, then $X^- =$ LRLR. Two arcs *a* and *b* are *indistiguishable*, if for all $X^{+/-}$, $X^{+/-}(a)$ is similar to $X^{+/-}(b)$. More precisely, *a* and *b* are indistiguishable if and only if one of the following two conditions is verified: $\lambda^+_0(X^+(a)) = \lambda^+_0(X^+(b))$ or $\lambda^+_0(X^+(a)) = \lambda^-_0(X^-(b))$. Examples of similar and indistinguishable arcs are given in Figure 5b. For any given planar graph with only two planar representations, two arcs are isomorphic (or automorphic) iff they are indistinguishable. This result is due to Hopcroft-Tarjan.[14] To prove the correctness of our partition algorithm for triconnected planar graphs, we therefore need to prove the following proposition.

Proposition IV: upon termination of Scheme 3 applied to triconnected planar graphs two arcs are indistinguish-

able iff they have the same invariant. *Proof.* Let *a* and *b* be two arcs. We will show that $\lambda_n(a) = \lambda_n(b)$ if an only if *a* and *b* are indistinguishable. Let X^+ be an arbitrary chain and operators R and/or L, and let X^- be the inverse chain. According to Scheme 6, $\lambda_n(a) = \lambda_n(b)$ iff one of the following conditions is true:

$$(1)\ \lambda^+{}_n(a) = \lambda^+{}_n(b) \qquad (2)\ \lambda^+{}_n(a) = \lambda^-{}_n(b)$$

Let us assume that condition (1) is true. Following Scheme 8, condition (1) is true iff the next three conditions are true:

$$(3)\ \lambda^+{}_{n-1}(a) = \lambda^+{}_{n-1}(b)$$
$$(4)\ \lambda^+{}_{n-1}(\mathrm{R}(a)) = \lambda^+{}_{n-1}(\mathrm{R}(b))$$
$$(5)\ \lambda^+{}_{n-1}(\mathrm{L}(a)) = \lambda^+{}_{n-1}(\mathrm{L}(b))$$

Again, following Scheme 8, condition (3) is true iff the next three conditions are true:

$$(6)\ \lambda^+{}_{n-2}(a) = \lambda^+{}_{n-2}(b)$$
$$(7)\ \lambda^+{}_{n-2}(\mathrm{R}(a)) = \lambda^+{}_{n-2}(\mathrm{R}(b))$$
$$(8)\ \lambda^+{}_{n-2}(\mathrm{L}(a)) = \lambda^+{}_{n-2}(\mathrm{L}(b))$$

For condition (4) we obtain:

$$(9)\ \lambda^+{}_{n-2}(\mathrm{R}(a)) = \lambda^+{}_{n-2}(\mathrm{R}(b))$$
$$(10)\ \lambda^+{}_{n-2}(\mathrm{RR}(a)) = \lambda^+{}_{n-2}(\mathrm{RR}(b))$$
$$(11)\ \lambda^+{}_{n-2}(\mathrm{LR}(a)) = \lambda^+{}_{n-2}(\mathrm{LR}(b))$$

For condition (5) we derive:

$$(12)\ \lambda^+{}_{n-2}(\mathrm{L}(a)) = \lambda^+{}_{n-2}(\mathrm{L}(b))$$
$$(13)\ \lambda^+{}_{n-2}(\mathrm{RL}(a)) = \lambda^+{}_{n-2}(\mathrm{RL}(b))$$
$$(14)\ \lambda^+{}_{n-2}(\mathrm{LL}(a)) = \lambda^+{}_{n-2}(\mathrm{LL}(b))$$

By iterating the same process it is easy to show that condition (1) is true iff:

$$(15)\ \lambda^+{}_0(X^+(a)) = \lambda^+{}_0(X^+(b))$$

for all X^+ having a size $\le n$.
Using the same procedure, condition (2) is true iff:

$$(16)\ \lambda^+{}_0(X^+(a)) = \lambda^-{}_0(X^-(b))$$

for all $X^{+/-}$ having a size $\le n$. Because condition 15 and 16 define indistinguishable edges, we have proven that two edges have the same invariant iff they are indistinguishable.

Corollary II: upon termination of Scheme 3 applied to triconnected planar graphs, two edges or two vertices are automorphic iff they have the same invariant. *Proof.* From Corollary I, we know that two edges *g* and *h* are automorphic iff their respective arcs (g_{12}, g_{21}, h_{12}, h_{21}) have identical invariants. Because the automorphism function may be a reflection operation, the two edges *g* and *h* are automorphic iff $(\lambda^+(g_{12}),\lambda^+(g_{21}),\lambda^-(g_{12}),\lambda^-(g_{21})) = (\lambda^+(h_{12}),\lambda^+(h_{21}),\lambda^-(h_{12}),\lambda^-(h_{21}))$ when sorted in lexicographic order. Such a test is performed by the function compute-edge-invariant-cyclic in Scheme 8. From Corollary I, we know that two vertices v and w are automorphic iff $\lambda(v) = \mathrm{MAP}(\lambda_{\mathrm{sorted}}(\Gamma(v))) = \mathrm{MAP}(\lambda_{\mathrm{sorted}}(\Gamma(w))) = \lambda(w)$. In the present case, the set of edge invariants must be sorted in respect to the planar representation. If the two sets of invariants are identical but the corresponding edges do not appear in the same order when turning clockwise or counterclockwise around the vertices, the two vertices are not automorphic. The clockwise/counterclockwise ordering is introduced in compute-vertex-invariant-cyclic in Scheme 8.

To evaluate the computational complexity of Scheme 3 applied to triconnected planar graph, we first assume that a planar representation has been constructed prior to running the algorithm. As already mentioned, such a representation can be computed in time O(N), where N is the number of vertices.[23] We also assume that the degree of the faces is known prior to running Scheme 3. Computing face degrees can be achieved in time O(N). As with trees, initialization and computing procedures have constant computational costs. According to Proposition IV, Scheme 3 halts when for every edge all paths X have been computed. A given path may comprise all arcs, thus the number of iterations is bounded by the total number of arcs ($<$6N for planar graphs). Because every arc is visited at each iteration, the upper bound time complexity is O(N^2).

Planar Graphs. The generalization of Scheme 3 to all planar molecular graphs is straightforward. All planar graphs can be decomposed into mono-, bi-, and triconnected components. Algorithms for planar graphs decomposition can be found in Kucera.[22] We have shown that Schemes 4 and 5 handle trees and *n*-gons, whereas Schemes 7 and 8 process triconnected graphs. A monoconnected component that is not bi- or triconnected is a tree. Biconnected components that are not triconnected are attached to the graph through one or two articulation points. If the component is attached via one articulation point, the component is an *n*-gon, and if the component is attached via two articulation points, the component is a linear chain (i.e., a tree). We call *tree-components* all monoconnected components that are not triconnected (this includes *n*-gons). We call *cyclic-components* all the other components; that is, all the triconnected components of the graph. The tree- and cyclic-components can be derived directly from the mono-, bi-, and triconnected component graph decompositions given in Kucera.[22]

The generalized partition algorithm consists of decomposing the given molecular graphs or transformed graph into tree- and cyclic-components and then to use Schemes 4−5 for the tree-components, and Schemes 7−8 for the cyclic-components. In the generalized algorithm, invariants are represented by a triplet $(\lambda^0, \lambda^+, \lambda^-)$, where λ^0 is the invariant corresponding to the tree-components, and (λ^+, λ^-) are the invariants of the cyclic-components. The initial invariant values are $(\lambda^0, 0, 0)$ if the corresponding arcs belong to tree-components, and $(0, \lambda^+, \lambda^-)$ for arcs that belong to cyclic-components. When the invariants are updated at each iteration of Scheme 3, all of their three components may have non-null values, depending on which component their neighbors belong. The initialization procedure is detailed in Scheme 9.

For any given arc *a*, if *a* belongs to a cyclic-component, then Scheme 8 is applied with the adjacent arcs that belong to the same cyclic-component. For the remaining adjacent

Scheme 9

```
init-arc-invariant(a)
  /* a is an arc
     */
  inv0 = inv+ = inv- = 0
  if a belongs to a triconnected component then
     (inv+,inv-) = init-arc-invariant-cyclic(a)
  else
     inv0 = init-arc-invariant-tree(a)
  endif
  invariant = (inv0,inv+,inv-)
  return(invariant)
end
```

arcs, Scheme 5 is applied because these arcs are free to rotate around *a*. If *a* belongs to a tree-component, Scheme 5 is applied. Because tree-components may change the invariants of the cyclic-component to which they are attached, the cyclic invariants of an arc (inv+,inv−) are computed by taking into account the tree invariant of that arc. For the same reason, the tree invariant of an arc (inv0) is computed by mapping the invariant returned by compute-invariant-tree with the pair of cyclic invariants of the same arc. Vertices can be articulation points, and therefore may belong to several cyclic- and tree-components. If a given vertex is an articulation point, Scheme 8 is applied independently for each cyclic-component, whereas the tree-components are treated using Scheme 5. The final invariant is obtained by sorting all resulting invariants because the cyclic- and tree-components are not ordered with respect to each other but are free to rotate around the articulation points. The algorithms are given in Scheme 10.

To maintain a constant computational cost for the initialization and computation functions just presented, tree- and cyclic-components must be determined and labeled prior to running the partition algorithm. Linear-time algorithms to compute the components of a graph are given in Kucera.[22] If these algorithms are used, the overall computational complexity of Scheme 3 is the same than for trees and cyclic graphs; that is, O(N^2).

POLYNOMIAL-TIME CANONICAL LABELING ALGORITHM FOR PLANAR MOLECULAR GRAPHS

Upon termination of Scheme 3, arcs, edges, and vertices are labeled by integers. We have shown these integers to be complete graph invariants when the studied graph is planar. All invariants are computed in a systematic manner from the degree of the vertices and faces and a given MAP function. Thus, two graphs leading to the same series of invariants are isomorphic. Consequently, the canonical labeling of any given planar molecular planar graph is obtained by partitioning the arcs using Scheme 3, and sorting the arcs with respect to their invariants. The time complexity of the canonical labeling procedure is dominated by the time complexity of Scheme 3; that is, O(N^2). Note that if Scheme 6 is applied instead of Scheme 3 for trees, then the time complexity of canonical labeling is dominated by the sorting procedure; that is, O(N) if radix or bucket sorts are used, and O(NlogN) otherwise. The canonical labeling procedure can be applied directly to planar molecular graphs without using Scheme 1 if the initial invariants are computed using face degrees, atom degrees, atomic numbers, and bond orders.

Scheme 10

```
compute-arc-invariant(a, $\Gamma$, $\lambda$)
  /* a is an arc
     $\Gamma$ is the set of arcs whose tail is the head of a
     $\lambda$ = ($\lambda^0$, $\lambda^+$, $\lambda^-$) is the set of triple-invariants
     */

  inv0 = inv+ = inv- = 0
  let c be the cyclic (triconnected) component of a
  partition $\Gamma$ in $\Gamma_c$ U ($\Gamma - \Gamma_c$)
  if ($\Gamma_c \neq \emptyset$) then
     (inv+,inv-) = compute-arc-invariant-cyclic(a, $\Gamma_c$, $\lambda^+$, $\lambda^-$)
     inv+ = MAP(inv+, $\lambda^0$(a))
     inv- = MAP(inv-, $\lambda^0$(a))
  endif
  invc = (MIN($\lambda^+$(a),$\lambda^-$(a)),MAX($\lambda^+$(a),$\lambda^-$(a))
  inv0 = MAP(compute-arc-invariant-tree(a, $\Gamma - \Gamma_c$, $\lambda^0$),invc)
  invariant = (inv0,inv+,inv-)
  return(invariant)
end

compute-edge-invariant(e, $\lambda$)
  /* e is an edge
     $\lambda$ = ($\lambda^0$, $\lambda^+$, $\lambda^-$) is the set of triple-invariants
     */
  inv0 = compute-edge-invariant-tree(e, $\lambda^0$)

  invc = compute-edge-invariant-cyclic(e, $\lambda^+$, $\lambda^-$)
  invariant = MAP(inv0,invc)
  return(invariant)
end
compute-vertex-invariant(v, $\Gamma$, $\lambda$)
  /* v is a vertex
     $\Gamma$ is the set of edges adjacent to v
     $\lambda$ is the set of invariants
     */
  partition $\Gamma$ in $\Gamma_{c1}$ U $\Gamma_{c2}$  U...  U $\Gamma_{ck}$
    where $c_1$, $c_2$,..$c_3$ are the cyclic and tree-components of $\Gamma$
  For i = 1 to k do
    if $c_i$ is a cyclic-component then
       $inv_i$ = compute-vertex-invariant-cyclic(v, $\Gamma_{ci}$, $\lambda$)
    else
       $inv_i$ = compute-vertex-invariant-tree(v, $\Gamma_{ci}$, $\lambda$)
    endif
  done
  L = ($inv_1$,$inv_2$,...,$inv_k$)
  sort L in lexicographic order
  invariant = MAP($\lambda$(v),L)
  return(invariant)
end
```

APPLICATION OF THE AUTOMORPHISM PARTITIONING ALGORITHM

In this section, the computational time complexity of Schemes 3 and 6 are probed for branched paraffins, perifused PAHs, fullerenes, and nanotubes containing a number of carbon atoms up to 4600. The branched paraffins were constructed using a randomized isomer generator, and the other structures were constructed manually and energy minimized using the MSI/Cerius2 modeling package. All structures constructed were planar. Schemes 3 and 6 were run directly on the molecular graphs using the appropriate invariant initialization procedures. All runs were performed on an O2/R5000 SGI platform with 64Mbytes RAM. Complexity results are presented in Figures 6−9. For branched paraffins, Scheme 3 exhibits a time complexity bounded by O(NlogN), whereas for perifused PAHs and fullerenes, Scheme 3 appears to scale linearly. In both cases the “experimental” computational complexity is lower than the theoretical complexity O(N^2). This result was in fact expected because Scheme 3 requires N steps to converge only in specific cases. As a general rule, the more compact the structure, the faster the convergence; consequently, linear structures will require the highest number of steps. Such a requirement is fulfilled with nanotubes, and Scheme 3 scales O(N^2) in Figure 9. In agreement with theoretical results,

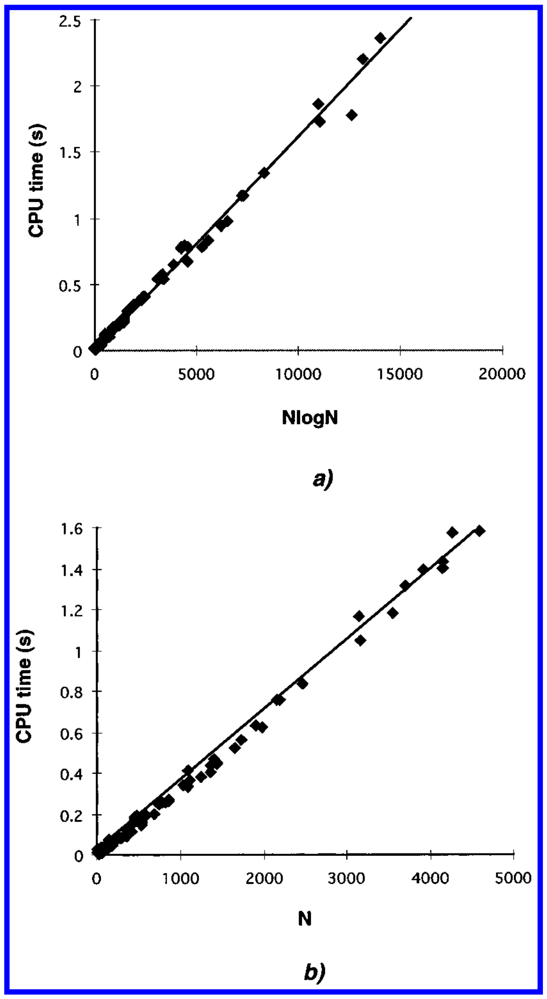


Figure 6. Time complexity results for branched paraffins: (a) using Scheme 3; (b) using Scheme 6.

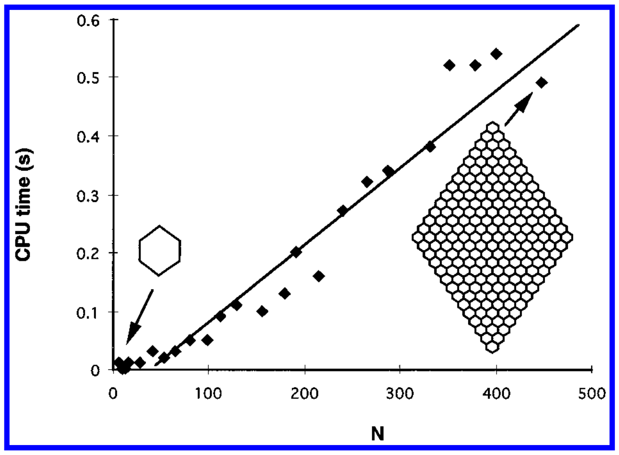


Figure 7. Time complexity results of Scheme 3 for perifused PAHs.

Scheme 6 shows a linear time complexity for branched paraffins. Note that the time complexity cannot be sublinear because it takes at least N steps to load any molecular structure before running Scheme 3 or Scheme 6.

It is worth noticing that the CPU times given in this paper are smaller than previously reported CPU times obtained using other partitioning algorithms on similar molecular structures. More importantly, the structures probed are several orders of magnitude larger than previously studied structures. These results, of course, may be attributed to

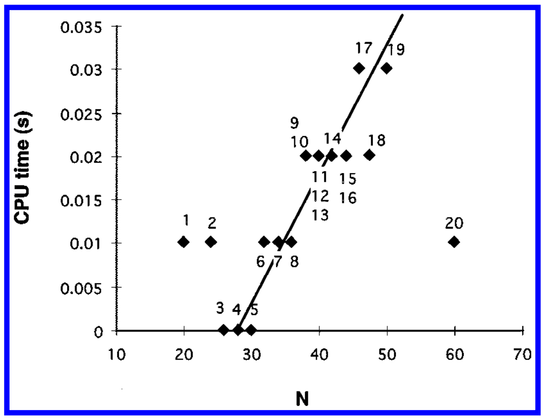


Figure 8. Time complexity results of Scheme 3 for fullerenes. The numbers indicated are the fullerenes structures drawn in Figure 5 of Laidboeur et al.[31] Scheme 3 found the same number of classes for edges and vertices as Laidboeur et al.[31] Carbon numbers and symmetries are: 1 = $C_{20}(I_h)$, 2 = $C_{24}(D_{6d})$, 3 = $C_{26}(D_{3h})$, 4 = $C_{28}(T_d)$, 5 = $C_{30}(D_{5h})$, 6 = $C_{32}(D_3)$, 7 = $C_{34}(C_{3v})$, 8 = $C_{36}(D_{6h})$, 9 = $C_{38}(D_{3h})$, 10 = $C_{38}(C_{3v})$, 11 = $C_{40}(T_d)$, 12 = $C_{40}(C_{3v})$, 13 = $C_{40}(D_{5d})$, 14 = $C_{42}(D_3)$, 15 = $C_{44}(T)$, 16 = $C_{44}(D_{3h})$, 17 = $C_{46}(C_3)$, 18 = $C_{48}(D_3)$, 19 = $C_{50}(D_{5h})$, 20 = $C_{60}(I_{5h})$.

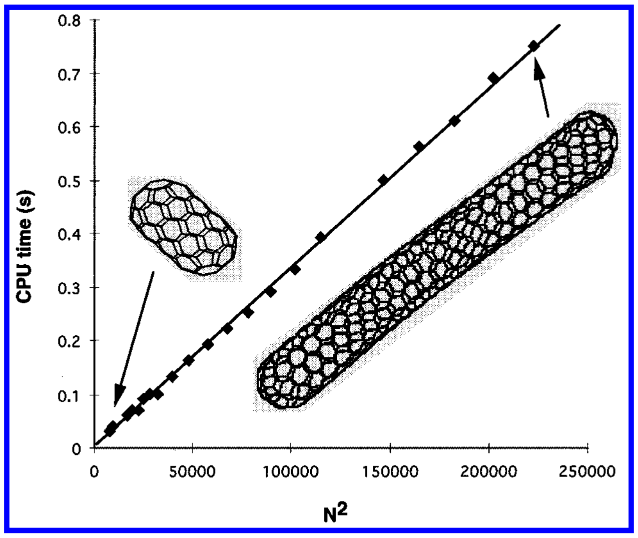


Figure 9. Time complexity results of Scheme 3 for nanotubes.

advances in hardware technology, but the main factor is the use of polynomial time algorithms.

CONCLUSION

The problems of isomorphism, automorphism partitioning, and canonical labeling have been shown to be polynomial-time for all molecular graphs. This result is important for chemical information studies, and in particular for structure generation, because all structure generators are making use of an isomorphism or a canonical labeling routine. A transformation algorithm between molecular graphs and simple graphs has been outlined. This transformation enables one to use all the polynomial-time isomorphism and canonical labeling algorithms developed in the context of graph theory for molecules. Morgan's extended connectivity technique has been tested for automorphism partitioning of planar graphs. Morgan's technique was rigorously proven to be correct for noncyclic graphs, with a time complexity scaling $O(N^2)$, where N is the number of vertices. A linear-time optimized algorithm was also given. A modification of Morgan's technique was introduced to handle cyclic planar graphs. With this modification, extended connectivity is now computed with respect to the planar drawings of the graph. The time complexity of the modified Morgan algorithm remains $O(N^2)$.

The automorphism partitioning algorithms described are the first to be rigorously proven polynomial-time for molecular graphs. The consequences of the polynomial-time scaling is that the molecular structures that can be processed by the proposed algorithms are several order of magnitudes larger than previously reported. Furthermore, the present study demonstrates that the two-step procedure, which consists of first computing invariants and then refining the partitioning with an exhaustive generation of all labelings, is not necessary for planar graphs because the invariants computed are complete graphs invariants. However, not every technique to compute invariants leads to complete graph invariants. Although the extended connectivity concept has been shown in this paper to be valid, the eigenvalue technique is not correct because some planar graphs admit isospectral points. Finally, one may wonder if the extended connectivity technique can be applied to nonplanar graphs? The extended connectivity concept seems to generalize well to nonplanar rigid graphs. With rigid graphs such as zeolites, the faces of the graph can be ordered in a unique fashion clockwise or counterclockwise around the edges. The invariant of a given arc is thus computed considering the invariants of adjacent arcs not only in the right and left faces, but in all faces attached to the arc. Whether or not the extended connectivity concept may be applied to nonplanar rigid graph is an open problem.

ACKNOWLEDGMENT

I thank Leslie A. Golberg for pointing out Luks, Babai, and Hoffman algorithms, and Robert T. Wilcox for collecting the data plotted in Figures 6−9. I am also pleased to acknowledge the funding provided by the Math. Information and Computer Science program of the U.S. Department of Energy under contract DE-AC04-76DP00789.

REFERENCES AND NOTES

(1) Read, R. C.; Corneil, D. G. The Graph Isomorphism Disease. *J. Graph. Theory* **1977**, *1*, 339−363.
(2) Miller, G. Graph isomorphism, general remarks. *J. Comput. Syst. Sci.* **1979**, *18*, 128−142.
(3) Babai, L.; Kucera, L. Canonical labelling of graphs in linear average time. In *Foundations of Computer Science*, Proceedings of the 20th IEEE Symposium, 1979; pp 39−46.
(4) Morgan, H. L. Generation of a unique machine description for chemical structures - A technique developed at chemical abstracts services. *J. Chem. Doc.* **1965**, *5*, 107−113.
(5) Carhart, R. E. Erroneous claims concerning the perception of topological symmetry. *J. Chem. Inf. Comput. Sci.* **1978**, *18*, 108−110.
(6) Rücker, G.; Rücker, C. Computer perception of constitutional (topological) symmetry: TOPSYM, A fast algorithm for partitioning atoms and pairwise relations among atoms into equivalent classes. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 187−191.
(7) Rücker, G.; Rücker, C. On using the adjacency matrix power method for perception of symmetry and for isomorphism testing of highly intricate graphs. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 123−126.
(8) Babai, L.; Erdös, P.; Selkow, S. M. Random graph isomorphism. *SIAM J. Computing* **1980**, *9*, 628−635.
(9) Köbler, J.; Schöning, U.; Torán, J. *The Graph Isomorphism Problem. Its Structural Complexity*; Birkhauser: Boston, 1993.
(10) Liu, X.; Klein, D. J. The graph isomorphism problem. *J. Comput. Chem.* **1991**, *12*, 1243−1251.
(11) Collatz. L.; Sinogowitz, U. Spektren endlichen Graphen. *Abh. Math. Sem. Univ. Hamburg* **1957**, *21*, 63−77.
(12) Fan, B. T.; Barbu, A.; Panaye, A.; Doucet, J.-P. Detection of constitutionally equivalent sites from a connection table. *J. Chem. Inf. Comput. Sci.* **1996**, *36*, 654−659.
(13) Broadbelt, L. J.; Stark, S. M.; Klein, M. T. Computer generated reaction modelling: decomposition and encoding algorithms for determining species uniqueness. *Comput. Chem. Eng.* **1995**, *20*, 113−129.
(14) Hopcroft, J. E.; Tarjan, R. E. Isomorphism of planar graphs. In *Complexity of Computer Computation*; Miller, R. E., Thatcher, J. W. Ed.; Plenum: New York, 1972; pp 131−152.
(15) Ladner, R. E. On the structure of polynomial-time reducibilities. *J. Assoc. Comput. Mach.* **1975**, *22*, 155−171.
(16) Hopcroft, J. E.; Wong, J. K. Linear time algorithm for isomorphism of planar graphs. *Proc. 6th ACM Symp. Theory Comput.* **1974**, 172−184.
(17) Bodlaender, H. L. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms* **1990**, *11*, 631−643.
(18) Luks, E. M. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Sys. Sci.* **1982**, *25*, 42−65.
(19) Hoffmann, C. M. *Group-Theoretic Algorithms and Graph Isomorphism*; Springer-Verlag: Berlin, 1982.
(20) Babai, L.; Luks, E. M. Canonical labelling of graphs. *Proc. 15th ACM Symp. Theory Comp.* **1983**, 171−183.
(21) Chen, J. A linear-time algorithm for isomorphism of graphs of bounded average genus. *SIAM J. Disc. Math.* **1994**, *7*, 614−631.
(22) Kucera, L. *Combinatorial Algorithms;* Adam Hilger: Bristol, 1989.
(23) Hopcroft, J. E.; Tarjan, R. E. Efficient planarity testing. *J. Assoc. Comput. Mach.* **1974**, *22*, 155−171.
(24) Razinger, M. Extended connectivity in chemical graphs. *Theor. Chim. Acta* **1982**, *61*, 581−586.
(25) Shelley, C. A.; Munk, M. E. Computer perception of topological symmetry. *J. Chem. Inf. Comput. Sci.* **1977**, *17*, 110−113.
(26) Shelley, C. A.; Munk, M. E. An approach to the assignment of canonical tables and topological symmetry perception. *J. Chem. Inf. Comput. Sci.* **1979**, *19*, 247−250.
(27) Wipke, W. T.; Dyott, T. M. Stereochemically unique naming algorithm. *J. Am. Chem. Soc.* **1974**, *96*, 4834−4842.
(28) Moreau, G. A. Topological code for molecular structures. A modified Morgan algorithm. *Nouv. J. Chim.* **1980**, *4*, 17−22.
(29) Balaban, A. T.; Mekenyan, O. Bonchev, D. Unique description of chemical structures based on hierarchically ordered extended connectivities (HOC procedures). I. Algorithms for finding graph orbits and canonical numbering of atoms. *J. Comput. Chem.* **1985**, *6*, 538−551.
(30) Lui, X.; Balasubramanian, K.; Munk, M. E. Computational techniques for vertex partitioning of graphs. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 187−191.
(31) Laidboeur, T.; Cabrol-Bass, D.; Ivanciuc, O. Determination of topological equivalent classes of atoms and bonds in C20-C60 fullerenes using a new prolog coding program. *J. Chem. Inf. Comput. Sci.* **1996**, *36*, 811−821.
(32) Condition (i) is verified because $\lambda_i(a_i^k) = \lambda_i(b_i^k)$ implies $\lambda_0(a_i^k) = \lambda_0(b_i^k)$.
(33) Whitney, H. A set of topological invaraints for graphs. *Am. J. Math.* **1933**, *55*, 321−325.
(34) Auslander, L.; Parter, S. V. On embedding graphs into the plane. *J. Math. Mech.* **1961**, *10*, 517−523.

CI9702914