Finding a Kekulé Structure in a Benzenoid System in Linear Time

Pierre Hansen,*,[§] Brigitte Jaumard,[†] Horst Sachs,[‡] and Maolin Zheng[§]

GERAD and Ecole des Hautes Etudes Commerciales, 5255 avenue Decelles,
Montréal, H3T 1V6, Canada, GERAD and Ecole Polytechnique de Montréal, H3T 1V6, Canada,
Institute of Mathematics, Technical University of Ilmenau, POB 327, D-98684, Germany

Received November 17, 1994[⊗]

Two algorithms with a linear time complexity are proposed for recognizing a benzenoid system from its adjacency list (or packed adjacency matrix), which leads to a plane representation of it with some vertical edges, and for finding a Kekulé structure in a benzenoid system.

1. INTRODUCTION

A benzenoid hydrocarbon *BH* is often represented by its molecular graph, i.e., a *benzenoid system H*.[1-5] A benzenoid system *H* is a finite connected subgraph of the infinite hexagonal lattice without cut vertices or nonhexagonal finite faces. The carbon atoms of *BH* correspond to the vertices of *H*, and the bonds between the carbon atoms correspond to the edges of *H*. The hydrogen atoms of *BH* together with their incident bonds are omitted in this representation.

Given a benzenoid system *H*, it is then natural to ask whether or not it corresponds to a benzenoid hydrocarbon *BH*. The answer appears to depend on the existence of a Kekulé structure or perfect matching of *H*. A *Kekulé structure* of *H* is a set of disjoint edges covering all vertices of *H*. Such edges correspond to double bonds of *BH*. Despite many attempts, benzenoid hydrocarbons corresponding to benzenoid systems without Kekulé structure have never been synthesized, even as transient species.[2] So it appears that a necessary condition for a benzenoid system *H* to be the molecular graph of a benzenoid hydrocarbon is that *H* has a Kekulé structure. Conversely, numerous benzenoid hydrocarbons corresponding to benzenoid systems with Kekulé structures have been synthesized and no *a priori* reason seems to exist for being unable to do so whenever this condition holds, at least in principle.

Several algorithms have been proposed to assign a Kekulé structure to a benzenoid system or show that it has none.[6-13] These algorithms are based on quite different ideas: checking the cut condition on the leftmost lowest vertex (see section 3), exploitation of the correspondence between peak–valley path systems and Kekulé structures, use of network flows, detection of convex pairs of edges, construction of a cut tree, etc. Main criteria to compare such algorithms are their simplicity (or ease of application) and the time they require on a computer, i.e., their computational complexity. Only the algorithm given in ref 13 has been proven to have a linear time complexity (which is the best possible up to a constant factor).

In this paper, we first give in section 2 a linear algorithm for recognizing a benzenoid system, a coronoid system (which has precisely one finite nonhexagonal face), or a multiple coronoid system (which has several such faces) from its adjacency list and obtaining a plane representation of it. Then we describe in section 3 an implementation of the algorithm of ref 8, which uses the cut tree concept of ref 13, and show that it has a linear time complexity. In the last section, we discuss the use of this algorithm to detect fixed bonds in benzenoid systems.

2. AN ALGORITHM FOR RECOGNIZING GENERALIZED CORONOID SYSTEMS

A *generalized coronoid system S* is a finite two-connected plane subgraph of the infinite hexagonal lattice such that no two nonhexagonal faces (finite or infinite) have an edge in common. (This property implies that every edge belongs to at least one hexagon.) If *S* has precisely *p* nonhexagonal finite faces then *S* is called a *p*-coronoid, and *p* is called its *porosity*. A 0-coronoid is a benzenoid, a 1-coronoid is a coronoid.

2.1. Algorithm. Let *G* be a finite graph without loops or multiple edges. In what follows we describe an algorithm called RGC (recognize generalized coronoid) which in linear time recognizes whether or not *G* represents a generalized coronoid system *S* and, if so, determines its porosity; further, RGC yields the coordinates of an embedding of *S* into the plane.

The input data for *G* are the number *n* of vertices, the degree sequence *D*, and the adjacency list *A* of vertices, where *D* is an array of length *n*, *D*[*i*] is the valency of vertex *i*, and *A* is an array of length $\sum_{j=1}^{n} D[j]$ such that

$$A[j] \text{ with } \sum_{k=1}^{i-1} D[k] < j \le \sum_{k=1}^{i} D[k]$$

are the neighbors of *i*.

The output, if *G* is a generalized coronoid, is *n*, *h* (the number of its hexagons), *p*, *D*, an $n \times 4$ matrix *B* which defines a plane representation of *G* and a sequence *W* of integer vectors (*x*,*y*) ranked in lexicographic order; *B*[*i*][1] = 1(= −1) indicates that the nonvertical edge(s) incident with vertex *i* are above (below) *i*; *B*[*i*][2] is the left neighbor of *i*, *B*[*i*][4] is the right neighbor of *i*, and *B*[*i*][3] is the neighbor of *i* which is on the same vertical line as *i*; if any of *B*[*i*][*j*] (*j* = 2, 3, 4) is 0, then *i* has no neighbor at that relative position. An integer vector (*x*,*y*) is associated with a vertex *v* and represents *v*'s position in the plane, in terms of horizontal and vertical levels. An $n \times 2$ matrix *C* is also produced: (*C*[*i*][1], *C*[*i*][2]) is the geometric position of vertex *i* in the plane in an embedding of *G*.

[†] GERAD and Ecole Polytechnique de Montréal.
[‡] Technical University of Ilmenau.
[§] GERAD and Ecole des Hautes Etudes Commerciales.
[⊗] Abstract published in *Advance ACS Abstracts*, April 1, 1995.

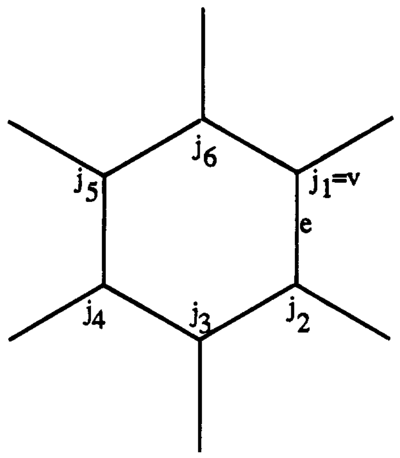


Figure 1. $\mathcal{C}$ in the embedding step of algorithm RGC.

The algorithm works in the same way as one embeds a benzenoid system into the hexagonal lattice. Also the following facts are used in the algorithm: (i) the position of a hexagon is determined by the positions of any three consecutive vertices of it; (ii) the position of an edge e incident to vertex v is determined by the positions of the other two edges which are incident to v (we assume here v has valency 3).

Algorithm RGC. Initial Step: Let the input graph be G. Select a vertex i. Set $B[i][1] = 1$. Set $B[i][j]$ ($j = 2, 3, D[i] + 1$) to be the neighbors of i (note that $D[i] \leq 3$); C[i]-[1] = 0; C[i][2] = 0;

$$(C[B[i][2]][1], C[B[i][2]][2]) = (-\cos 30°, \sin 30°)$$

$$(C[B[i][3]][1], C[B[i][3]][2]) = (0, -1)$$

if B[i][4] $\neq$ 0 then

$$(C[B[i][4]][1], C[B[i][4]][2]) = (\cos 30°, \sin 30°)$$

All edges are unmarked. Set $Q[1] = i$ (where Q is a stack). Mark i.

Preliminary Step. If there is a vertex whose valency is 0 or greater than 3, go to Output (b).

Embedding Step. If Q is empty, go to Checking Step. Otherwise, let v be the top element of Q. Repeat: if all the edges adjacent to v are marked then remove v from Q and set V to be the top element of Q. If Q is not empty then let e be an unmarked edge incident to v, otherwise go to Checking Step.

(1) Find all circuits with length less than or equal to 6 which contain e. If there is such a circuit with length less than 6 or the number of circuits found is 0 or greater than 2, go to Output (b).

(2) For each circuit $\mathcal{C} = (j_1 j_2 j_3 j_4 j_5 j_6)$ found above, without loss of generality, let $j_1 = v$ and $e = (j_1, j_2)$. Note that all neighbors of $v = j_1$ have been embedded in the plane (i.e., their geometric positions in the plane represented by $(C[i][1], C[i][2])$ are known). Thus the geometric positions of j_1, j_2, and j_6 are known. Again without loss of generality, let e be vertical and edge $(j_1 j_6)$ be on the left of and above e (see Figure 1).

(a) Assign a temporary position to ecah of j_3, j_4, and j_5 as follows

$$j_3\text{:}\quad (C[j_6][1], C[j_2][2] - \sin 30°)$$

$$j_4\text{:}\quad (C[j_2][1] - 2\cos 30°, C[j_2][2])$$

$$j_5\text{:}\quad (C[j_1][1] - 2\cos 30°, C[j_1][2])$$

If j_k ($k = 3,4,5$) is marked and its temporary position is different from its assigned geometric position, go to Output (b); else

$$B[j_1][1] = 1,\quad B[j_1][2] = j_6,\quad B[j_1][3] = j_2$$

$$B[j_2][1] = -1,\quad B[j_2][2] = j_3,\quad B[j_2][3] = j_1$$

$$B[j_3][1] = 1,\quad B[j_3][2] = j_4,\quad B[j_3][4] = j_2$$

$$B[j_4][1] = -1,\quad B[j_4][3] = j_5, B[j_4][4] = j_3$$

$$B[j_5][1] = 1,\quad B[j_5][3] = j_4,\quad B[j_5][4] = j_6$$

$$B[j_6][1] = -1,\quad B[j_6][2] = j_5,\quad B[j_6][4] = j_1$$

(b) Determine a temporary position for each vertex i which is adjacent to some vertex j_k and is not contained in $\mathcal{C}$ in a similar way as in (a) by the positions of j_k and $B[j_k][r]$ ($r = 1,2,3,4$). If i is marked and its temporary position is different from $(C[i][1], C[i][2])$, go to Output (b); otherwise set $(C[i]$-$[1], C[i][2])$ equal to the temporary position.

(c) Mark all unmarked vertices and unmarked edges of $\mathcal{C}$.

(3) Remove v from Q. For each vertex i of the circuits found above, if the number of marked edges incident to i is less than $D[i]$ add i to Q. Repeat the Embedding Step.

Checking Step. **(1)** Assign an integer vector (x_i, y_i) to each vertex i in the following way (where (x_i, y_i) indicates the relative position of i in the plane):

(i) set $(x_1, y_1) = (0,0)$; add vertex 1 to L (where L is a stack) and mark vertex 1 as scanned.

(ii) Repeat the following until L is empty: let v be the top element of L. Remove v from L. For each unscanned neighbor i of v, set

$$x_i = \begin{cases} x_v - 1 & \text{if } i \text{ is on the left of } v \\ x_v + 1 & \text{if } i \text{ is the right of } v \\ x_v & \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} y_v - 1 & \text{if } i \text{ is below } v \\ y_v + 1 & \text{if } i \text{ is above } v \end{cases}$$

Add i to L and mark i as scanned.

(2) Rank the integer vectors assigned above in lexicographic order. Let W be the obtained ordering. If there are two vectors assigned in (1) which are equal, go to Output (b). Otherwise, G is a plane graph (C gives an embedding of G in the hexagonal lattice).

(3) (Calculate the number h of hexagons of G.) Set all edges of G to be unmarked. Let e be an unmarked edge. Let $h = 0$. Repeat: {find the hexagons containing e; let j be the number of hexagons found which do not contain any marked edge; set h to be $h + j$; mark e} until there is no unmarked edge.

(4) By Euler's formula $F = E - V + 2$ (where F, V, and E are the numbers of faces, vertices, and edges of G, respectively), G is a generalized coronoid with $p = F - 1 - h$. Go to Output (a); otherwise go to Output (b).

Output. **(a)** Return n, D, h, W, p, and B.

(b) *G* is not a generalized coronoid system.

2.2. Correctness. We give the main points of the proof below and omit the details. The initial step correctly embeds a vertex and its neighbors into the hexagonal lattice. Each iteration of Embedding Step (2) embeds a hexagon and the vertices adjacent to it into the hexagonal lattice if this is possible. If Embedding Step ends at (1), 2(a) or 2(b), then clearly the input is not a generalized coronoid system. Otherwise, when Embedding Step ends the input graph is embedded into the hexagonal lattice. Checking Step will detect if there are two vertices overlapping in the embedding. If there are none, then the input graph is a generalized coronoid system; otherwise it is not. It is easy to see: if the input graph is a generalized coronoid system, the algorithm will end at Output (a). We summarize the above as the following theorem.

Theorem 1. *Algorithm RGC determines whether an input graph G is a generalized coronoid system and, if so, yields its porosity.*

2.3. Complexity. The above algorithm has a linear time complexity. This is because of the following:

Initial Step takes constant time. Preliminary Step takes $O(n)$ time. Each iteration of the Embedding Step needs a constant time. Thus the total time of the Embedding Step is $O(n)$. The time needed in the Checking Step (1) is $O(n)$ since each vertex is scanned once. To assign the integer vectors associated to the vertices also requires $O(n)$ time. To calculate *W* and determine if there are any two integer vectors equal can be done in $O(n)$ time as follows:

Let *s* and *m* be the smallest numbers among x_i ($i = 1,2,...,n$) and among y_i ($i = 1,2,...,n$), respectively. Set (x_i,y_i) to be $(x_i - (s - 1), y_i - (m - 1))$ ($i = 1,...,n$). Then all x_i and y_i are greater than 0 and less than $n + 1$; rank the vectors (x_i,y_i) ($i = 1,2,...,n$) in lexicographic order in the following way:

Assume that there are *n* buckets $B_1,B_2,...,B_n$ and *n* queues $Q_1,Q_2,...,Q_n$; put (x_i,y_i) into B_{y_i}; for *j* from 1 to *n*, add to Q_{x_i} those (x_i,y_i) which belong to B_j (note that $y_i = j$); then rank the vectors in Q_i before those in Q_{i+1} ($i = 1,...,n-1$); a lexicographic order *W* of these vectors is obtained; following this order, we can determine if there are two equal integer vectors.

Each iteration of Checking Step (3) takes constant time. There are at most $3n/2$ such iterations. So the total time spent on Checking Step (3) is $O(n)$. Checking Step (4) takes constant time. The time complexity of the algorithm is $O(n)$. Thus we have the following theorem.

Theorem 2. *Algorithm RGC has a linear time complexity in* the *size of the input.*

3. ALGORITHM FOR FINDING A KEKULE STRUCTURE

Throughout the symbol *H* denotes a fixed benzenoid system. Let *B* and *W* be the same as in the output of RGC. As in the complexity analysis of RGC, we can assume that the components of the vectors in *W* are nonnegative. We also assume that the vertices of *H* are labeled in the same way as in RGC. See Figure 2 for an illustration of the assigned integer vectors. The integer vectors assigned to the vertices of *H* indicate the relative positions of the vertices. For instance, let *u* and *v* be two vertices and (x_u,y_u), (x_v,y_v) their vectors. If $x_u < x_v$ and $y_u < y_v$ then *u* is on the left of and below *v*, etc.

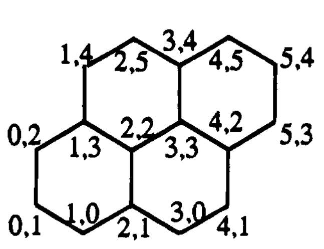


Figure 2. Vectors assigned to the vertices of *H*.

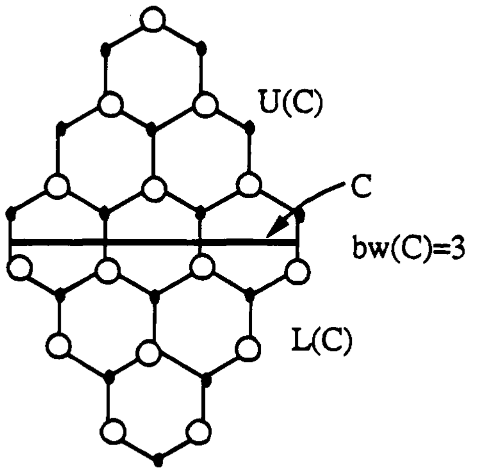


Figure 3. *C*, *U*(*C*), *L*(*C*), and *bw*(*C*).

Since *H* is a bipartite graph, we assume that the vertices of *H* are colored with two colors, white and black, such that any two adjacent vertices have different colors. Without loss of generality, let the highest vertices of *H* have color white. Thus the lowest vertices of *H* have color black. For any subgraph *G* of *H*, let *b*(*G*) and *w*(*G*) denote the number of black vertices and the number of white vertices of *G*, respectively.

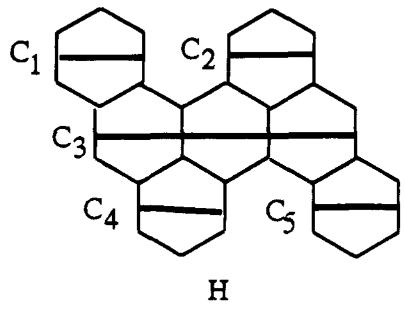
An edge of *H* is called a *boundary* edge if it belongs to only one hexagon of *H*, otherwise it is called an *internal* edge. All boundary edges form the *boundary* of *H*. A *cut C* of *H* is a horizontal line segment such that the ends of *C* are the midpoints of two vertical boundary edges of *H* and *C* is covered entirely by hexagons of *H* (see Figure 3). Removing all the edges of *H* crossed by *C* results in two connected subgraphs of *H*. The one above (below) *C*, denoted by *U*(*C*) (*L*(*C*)), is called the *upper* (*lower*) *bank* of *C*. Let $bw(C) = b(U(C)) - w(U(C))$. Note that the upper end vertices of the edges crossed by *C* are black. If *H* has a Kekulé structure then *bw*(*C*) is nonnegative.

3.1. Algorithm. In the algorithm given below, edges of *H* are colored red and blue progressively such that the red ones are disjoint. Whenever an edge is colored red, its end vertices are marked. When the algorithm stops, if all the vertices of *H* are marked, then a Kekulé structure of *H* has been found; otherwise, *H* has no Kekulé structures.

First apply Algorithm RGC to *H* and obtain *B* and *W*. Let *R* be the list of vertices of *H* ranked according to *W*. Note that in *R*, the vertices of *H* are ordered according to their positions on the plane from left to right. Then compute *bw*(*C*) for all cuts *C* of *H*.

Now we show how to obtain *R* and the set of numbers *bw*(*C*) for all cuts *C* of *H* in $O(n)$ time. Note that *R* can be obtained directly from *W*. Since the time complexity of RGC is linear, to obtain *R* and *B* requires $O(n)$ time.

Define the *cut tree T* of *H* (see Figure 4) as follows: the vertex set of *T* is the set of cuts of *H*; two vertices *C* and *C′*



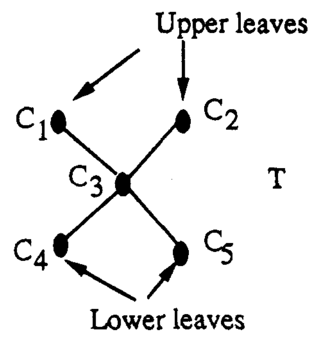


Figure 4. *H* and its cut tree *T*.

of *T* are adjacent if and only if the hexagons of *H* intersected by *C* and the hexagons of *H* intersected by *C'* have some edges in common. The vertices of *T* and the cuts of *H* are not distinguished below. The *upper* (*lower*) *neighbors* of a vertex *C* of *T* are those cuts which are adjacent to *C* and lie on the upper (lower) bank of *C*. For each vertex of *T*, the list of its upper neighbors and the list of its lower neighbors are calculated (and thus *T* is obtained). In ref 13, it is shown that the so defined graph *T* is a tree, and the lists of upper neighbors and the lists of lower neighbors of the cuts of *H* can be obtained in $O(n)$ time. For the sake of completeness, we next describe the procedure for obtaining these lists. For each cut *C*, we use the leftmost vertical edge crossed by *C* to represent *C*. Travel clockwise around the boundary of *H* once and mark the vertical edges whenever traveling them upward (traveling is made by using the representation, *B*, of *H*). Then all cuts together with their representing edges are found. The total time for this is clearly $O(n)$.

Let *H'* be the subgraph of *H* obtained by deleting all vertical edges of *H* but not their end vertices from *H*. Then *H'* is the union of disjoint paths. For each such path *P*, let $bw(P) = b(P) - w(P)$. A cut of *H* and a path of *H'* are said to be *incident* if some of the end vertices of the vertical edges crossed by the cut belong to the path. For a cut *C*, let P^u_c and P^l_c be the two paths of *H'* incident to *C* such that P^u_c is in the upper bank of *C* and P^l_c is in the lower bank of *C*. Associate with each cut *C* a pair $d(C) = (\mathrm{bw}(P^u_c), bw(P^l_c))$.

Travel path *P* of *H'* from the left to the right and let *C'* and/or *C''* be the currently met cuts (if any) incident to *P* in the traveling such that *C'* is above *P* and *C''* is below *P* (*C'* and *C''* can be determined by the marked vertical edges). Then put *C'* (*C''*) into the list of upper (lower) neighbors of *C''* (*C'*). Counting black and white vertices, $bw(P)$ is found when the traveling of *P* is completed. Assign $bw(P)$ to the first (second) coordinate of $d(C)$ of cut *C* if *C* is adjacent to *P* and below (above) *P*. The time spent on the above is $O(|P|)$ in total where $|P|$ is the length of *P*, i.e., the number of edges of *P*. So the total time to build the cut tree *T* (i.e., find the lists of lower and upper neighbors) and to assign the pair $(bw(P^u_c), bw(P^l_c))$ to each cut *C* of *H* is $O(\Sigma_P|P|) = O(n)$. Let $UN(f)(LN(f))$ be the set of upper (lower) neighbors of vertex *f* of *T*.

A leaf (a vertex of degree 1) *f* of a subtree *T'* of *T* is said to be an *upper* (*lower*) leaf if all the vertices of *T'* adjacent to *f* are below (above) it. Now we are ready to state the procedure to calculate $bw(C)$ for all cuts of *H*.

Set the initial value of T^* to be *T*. Let *F* be the set of leaves of T^*. Choose a leaf *f* from *F* (which is also a cut of *G*). If *f* is an upper (lower) leaf of T^*, set $bw(f) = bw(P^u_f) + \Sigma_{C\in UN(f)}\, bw(C)$ $(= -bw(P^l_f) + \Sigma_{C\in LN(f)}\, bw(C))$. If the vertex adjacent to *f* in T^* has degree 1 or 2, then add it to *F*. Delete *f* from T^* and *F*. Repeat the above procedure until *F* is empty. Note that when T^* has only one vertex, then this vertex is considered as either an upper leaf or a lower leaf of T^*. It can easily be checked that $bw(C)$ so obtained is exactly the one defined before.

The time spent in each iteration of the above procedure is $O(d_f)$ where d_f is the number of cuts adjacent to *f*. Since the number of vertices of *T* is less than *n*, $\Sigma_f d_f \le 2(n - 1)$. The total time to calculate $bw(C)$ for all cuts is $O(n)$.

We summarize the above as follows:

Lemma 1. *R and the set of numbers bw(C), where C runs through all cuts C of H, can be obtained in O(n) time.*

Algorithm PM. (Perfect matching.)

Input. A benzenoid system *G*.

Preliminary Calculation. Compute *B*, *R*, and the numbers $bw(C)$. If the number of black vertices is not equal to the number of white vertices or there is a cut *C* with $bw(C) < 0$, go to Output (b). Set $g(C)$ to be $bw(C)$ for each cut *C*.

Main Step. If there are no unmarked vertices, go to Output (a). Otherwise, let *v* be the first unmarked vertex in *R* (which is a leftmost one among the unmarked vertices).

(a) If there is no uncolored edge incident to *v*, go to Output (b).

(b) If *v* is incident to an uncolored vertical edge *e*, then: if the cut *C* which crosses *e* has $g(C) > 0$, set $g(C)$ to be $g(C) - 1$, color *e* red and the edges adjacent to *e* blue and mark the end vertices of *e*; otherwise color *e* blue.

(c) If *v* is not incident to an uncolored vertical edge, then color the edge *e* incident to *v* (since *v* is a leftmost unmarked vertex, it can be easily shown that *e* is a pendant edge) red and the edges adjacent to *e* blue and mark the end vertices of *e*.

Iterate the Main step.

Output. **(a)** Return the list of red edges.

(b) *H* has no perfect matching.

The algorithm is applied to an example of ref 15 in Figure 5.

3.2. Correctness of the Algorithm. The correctness of PM is proven in ref 8; for completeness we give a short proof here. A connected finite subgraph of the infinite hexagonal lattice is said to be a *generalized benzenoid system* if all of its finite faces are hexagons. The definition of Kekulé structure is extended to generalized benzenoid systems in a straightforward way as well as the definitions of coloring of vertices, cut, upper and lower banks (note, however, that a cut can possibly intersect a single edge in this case).

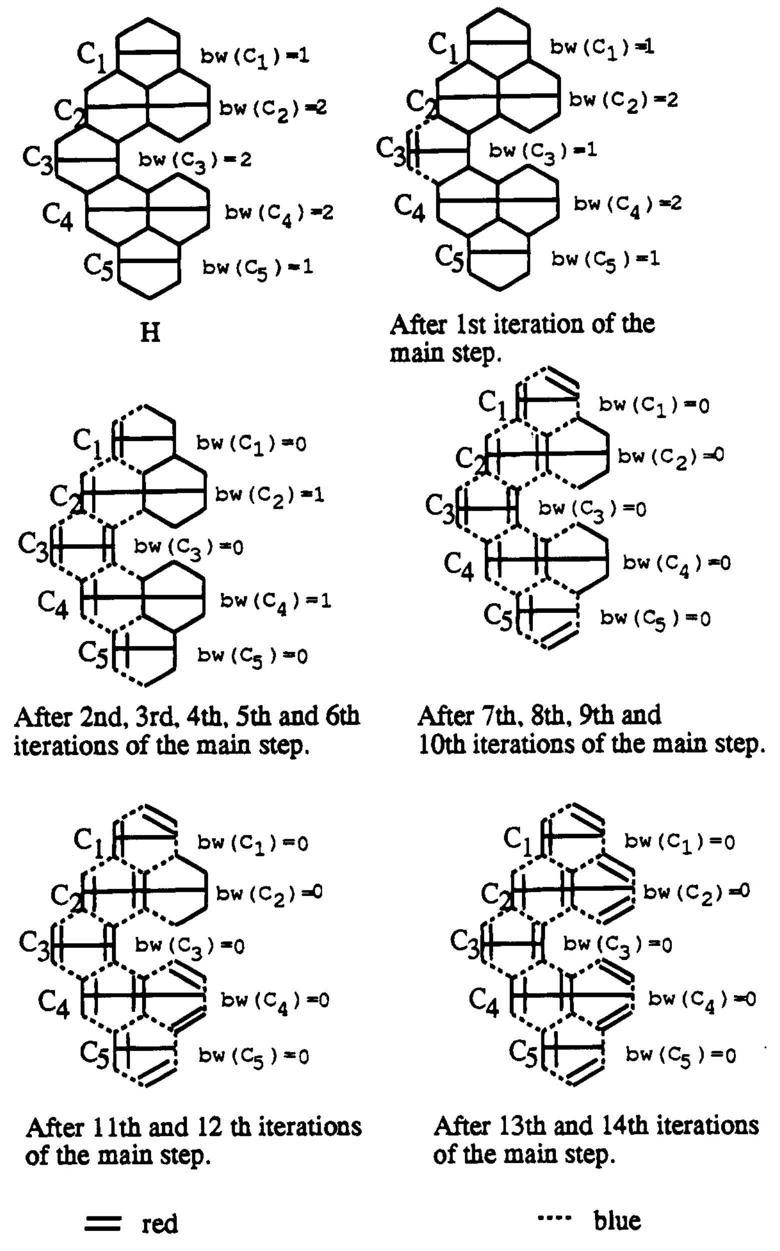

H
After 1st iteration of the main step.
After 2nd, 3rd, 4th, 5th and 6th iterations of the main step.
After 7th, 8th, 9th and 10th iterations of the main step.
After 11th and 12 th iterations of the main step.
After 13th and 14th iterations of the main step.
= red
···· blue

Figure 5. Illustration of algorithm PM.

Lemma 2. *Let G be a generalized benzenoid system with a Kekulé structure and v a leftmost vertex of G.*

(1) Assume that there is a vertical edge e of G incident to v. Let C be the cut of G crossing e. If bw(C) is 0 then e does not belong to any Kekulé structure of G. Otherwise e belongs to some Kekulé structure of G.

(2) Assume that there is no vertical edge incident to v. Then the edge e of G incident to v belongs to all Kekulé structures of G.

Proof. Statement (2) is clearly true since e is a pendant edge. We prove the truth of statement (1). Since the upper end vertices of the vertical edges crossed by C are all black, and since the edges of any Kekulé structure M represent a matching between the black and the white vertices, $bw(C)$ is equal to the number of edges of M which connect a black vertex from $U(C)$ to a white vertex from $L(C)$, i.e., we have

$$(*)\quad bw(C) = |\text{set of edges of } M \text{ crossed by } C| \geq 0, \quad \text{for every Kekulé structure } M \text{ of } G$$

Thus, in particular, $bw(C) = 0$ implies that the set of edges of M crossed by C is empty. Therefore if $bw(C) = 0$, then the edge e cannot belong to any Kekulé structure of G. Assume $bw(C) > 0$ and let M_0 be a fixed Kekulé structure of G. Because of $(*)$, the set of edges of M_0 corssed by C is not empty; let e' be the leftmost edge in M_0 intersected by C. If $e' = e$, then there is nothing to prove. Otherwise, let H' be the hexagonal chain containing all hexagons between e and e' intersected by C. Note that e and e' are the leftmost and the rightmost vertical edges of H'. Since v is a leftmost vertex of G, e is a leftmost edge of G, and the end vertices of e have valency 2 in G. Thus the leftmost nonvertical edges e_1 and e_2 of H' belong to M_0. Since e' is in M_0 and all other vertical edges of H' are not, the boundary edges of H' are alternatingly in M_0 and not in M_0 (see Figure 6). By exchanging the edges of M_0 with the edges not in M_0 along the boundary of H', a Kekulé structure M' of G is obtained. Moreover, e belongs to M'. The proof is completed.

Let the input benzenoid system of PM be H and the subgraph of H consisting of the uncolored edges and the unmarked vertices be G_i before the ith iteration of the main step. Then the following are true:

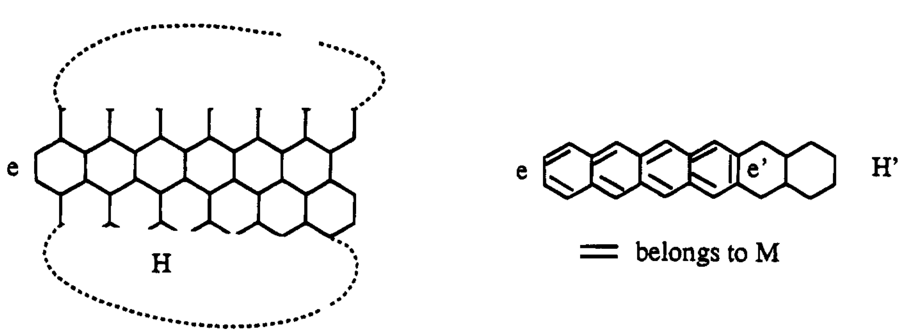


Figure 6. Illustration of H and H'.

(1) The unmarked vertex chosen in the ith iteration of the main step is one of the leftmost vertices of G_i (due to the fact that in R the vertices of H are ranked from left to right).

(2) Each component (i.e., each maximal connected subgraph) of G_i is a generalized benzenoid system. This can be easily proved by induction on the number of iterations of the main step.

(3) For a cut C of H, if it crosses some vertical edges of a connected component G_i^j of G_i, then the part C' of C which is completely covered by hexagons and edges of G_i^j is a cut of G_i^j. Moreover, C' and C have the same right end. Before the ith iteration of the main step, $g(C)$ is equal to $bw(C')$.

This can be shown as follows. Let H' be the hexagonal chain consisting of all hexagons of H intersected by C. Since PM always colors the edges and marks the vertices of H' from left to right, C' is a cut of G_i^j. By the way of updating $g(C)$ in PM, we can show by induction on the number of iterations of the main step that $g(C) = bw(C')$.

(4) If H has a Kekulé structure, then G_i has a Kekulé structure.

Proof of (4). When $i = 0$, $H = G_i$. The conclusion of (4) is true. Assume when $i < k$, the conclusion of (4) is true. Now consider $i = k$. If G_i becomes empty, then G_i has a Kekulé structure (by convention). By the induction hypothesis, G_{i-1} has a Kekulé structure. Let v be the vertex chosen in the $(i-1)$th iteration of the main step. Also assume that v belongs to a connected component G_{i-1}^j of G_{i-1}. Since G_{i-1} has a Kekulé structure, each of its connected components has a Kekulé structure, too. Thus (a) of the main step does not happen in the $(i-1)$th iteration (otherwise, G_{i-1} has an isolated vertex). If (c) of the main step happens in the $(i-1)$th iteration, then v is a pendant vertex, and the edge e of G_{i-1}^i incident to v belongs to all Kekulé structures of G_{i-1}. Thus after e has been colored red, G_i must have a Kekulé structure. If (b) happens in the $(i-1)$th iteration of the main step, let e' be the vertical edge incident to v which belongs to G_{i-1}^j. Let C and C' be the cuts of H and G_{i-1}^j such that both intersect e'. By the conclusion of (3), $g(C) = bw(C') > 0$. By Lemma 2, after the $(i-1)$th iteration, the subgraph consisting of the unmarked vertices and the uncolored edges of H, i.e., G_i, has a Kekulé structure.

By the conclusion of (4), we have the following Theorem.

Theorem 3. *Algorithm PM finds a Kekulé structure of H if there is one.*

3.3. Complexity of the Algorithm. By Theorem 2 and Lemma 1, B, R and $bw(C)$ for all cuts of H can be obtained in $O(n)$ time. Note that R is searched once in the whole procedure and each of (a), (b), and (c) takes constant time in each iteration of the main step, and there are at most $n/2$ such iterations. Thus the computational complexity of algorithm PM is $O(n)$.

Theorem 4. *Algorithm PM has a time complexity linear in the size of the input benzenoid system.*

4. APPLICATION OF PM

In this section we shall show that and how PM can be used for finding all fixed bonds of a benzenoid system in linear time. A brief comparison of PM with the linear algorithm given in ref 13 is also made.

An edge of H is called a *fixed bond* if it belongs to all Kekulé structures of H or to none. In the former case it is a fixed double bond and in the latter case a fixed single bond.

For a benzenoid system H, the vertices of H without higher (lower) neighbors are called *peaks* (*valleys*). A set of disjoint monotone paths is called a *perfect path system* of H if these paths issue from the peaks, end at the valleys of H, and cover all peaks and valleys.

Gordon and Davison[17] made the following observation: there is a one to one correspondence between Kekulé structures and perfect path systems of H. A rigorous proof of this fact was given by Sachs.[6] The correspondence is as follows: for a given Kekulé structure M, delete all vertical edges in M together with their end vertices and all nonvertical edges not in M from H, then a perfect path system is obtained. Conversely, for a given perfect path system P, choose the nonvertical edges in and the vertical edges not in these paths of P, then a Kekulé structure is found. In refs 12 and 14 it is shown that the matching between peaks and valleys induced by the perfect path systems P of H is unique (i.e., independent of the choice of P).

Let $\mathcal{R}$ ($\mathcal{L}$) be the set of monotone paths such that each path $R_{p,v}$ ($L_{p,v}$) in $\mathcal{R}$ ($\mathcal{L}$) with ends p and v is the rightmost (leftmost) monotone path from p to v which belongs to at least one perfect path system of H. In ref 16, it is shown that $\mathcal{R}$ and $\mathcal{L}$ are perfect path systems of H called the rightmost perfect path system and the leftmost perfect path system, respectively. Let $R_{p,v} \oplus L_{p,v}$ denote the set of edges belonging to $R_{p,v}$ or $L_{p,v}$ but not to both. Let RL be the subgraph of H consisteing of all edges (with their end points) in the union of the sets $R_{p,v} \oplus L_{p,v}$. The RL is the union of disjoint plane graphs which (as can easily be shown) have no cut edges (a cut edge is an edge whose removal disconnects the component to which it belongs). Let $\mathcal{C}$ be the set of boundaries of the connected components of RL. Then $\mathcal{C}$ consists of disjoint circuits.

In ref 16, it is proved that an edge of H is not fixed if and only if it belongs to, or is in the interior of, some circuit C of $\mathcal{C}$. A linear algorithm to find all fixed bonds to H is proposed in ref 16. This algorithm takes $\mathcal{R}$, $\mathcal{L}$ and H as the input. It is shown that the linear algorithm given in ref 13 can be used to find $\mathscr{R}$ and $\mathscr{L}$ in $O(n)$ time.

Instead of using the linear algorithm proposed in ref 13 (there denoted by PH), we can apply PM to find $\mathscr{R}$ and $\mathscr{L}$ in $O(n)$ time. It can be shown that the Kekulé structure of H found by PM corresponds to $\mathscr{R}$. If we rank all the vertices according to the reverse order of W to obtain the sequence R (which is used in PM), then PM will find the Kekulé structure which corresponds to $\mathscr{L}$. Thus $\mathscr{R}$ and $\mathscr{L}$ can be obtained in $O(n)$ time.

Finally, let us mention that the ideas underlying algorithms PM and PH are different. PH finds a perfect path system of H first if there is one and then gives the corresponding Kekulé structure. Also PH is easier to implement by hand, whereas PM is simpler for computer.

REFERENCES AND NOTES

(1) Cyvin, S. J.; Gutman, I. *Kekulé Structures in Benzenoid Hydrocarbons*; Springer: Berlin, 1988.

(2) Gutman, I.; Cyvin, S. J. *Introduction to the Theory of Benzenoid Hydrocarbons*; Springer: Berlin, 1989.

(3) Trinajstić, N. *Chemical Graph Theory*, 2nd edition; CRC Press: Boca Raton, FL, 1992.

(4) Dias, J. R. *Handbook of Polycyclic Hydrocarbons, Part A: Benzenoid hydrocarbons*; Elsevier: Amsterdam, 1987.

(5) Dias, J. R. *Handbook of Polycyclic Hydrocarbons, Part B: Benzenoid hydrocarbons*; Elsevier: Amsterdam, 1988.

(6) Sachs, H. Perfect Matchings in Hexagonal Systems. *Combinatorica* **1984**, *4*, 89–99.

(7) Zhang, F.; Chen, R.; Guo, X. Perfect Matchings in Hexagonal Systems. *Graphs Combinatorics* **1985**, *1*, 383–386.

(8) Sachs, H.; Zernitz, H. An O (n log n)-Algorithm for finding a Perfect matching in a hexagonal System. *Wyższa Szkola Inż. Zielona Gora, Zeszyty Naukowe* **1983**, or **84**, *75*, 101–108.

(9) He, W.; He, W. On Kekulé Structure and P–V Path Method. In *Studies in Physical and Theoretical Chemistry*; King, R. B., Rouvray, D. H., Eds.; 1987; Vol. 51, pp 476–483.

(10) Sheng, R. Identification of The Kekulé Structures of A Hexagonal System. *Chem. Phys. Lett.* **1987**, *142*, 196–199.

(11) Guo, X.; Zhang, F. Recognizing Kekuléan Benzenoid Systems by C–P–V Path Elimination. *J. Math. Chem.* **1990**, *5*, 157–170.

(12) Hansen, P.; Zheng, M. A Revised Peeling Algorithm for Determining if a Benzenoid System is Kekuléan. *J. Mol. Struct. (Theochem)* **1991**, *235*, 393–309.

(13) Hansen, P.; Zheng, M. A Linear Algorithm For Perfect Matching in Hexagonal Systems. *Dis. Math.* **1993**, *122*, 179–196.

(14) Cameron, K.; Sachs, H. *Research Report*; Bonn University, 1984. Also: Monotone Path Systems in Simple Regions. *Combinatorica* **1994**, *14*, 1–21.

(15) Gutman, I.; Cyvin, S. J. On Recognizing Kekuléan Benzenoid Systems. *J. Mol. Struct. (Theochem)* **1988**, *164*, 183–188.

(16) Hansen, P.; Zheng, M. A Linear Algorithm for Fixed Bonds in Hexagonal Systems. *J. Mol. Struct. (Theochem)* **1992**, *257*, 75–83.

(17) Gordon, M.; Davison, W. H. T. Theory of Resonance Topology of Fully Aromatic Hydrocarbons. *J. Chem. Phys.* **1952**, *20*, 428–453.

CI940126D