Coding Chemical Trees with the Centered *N*-tuple Code

Pierre Hansen,*,† Brigitte Jaumard,‡ Catherine Lebatteux,§ and Maolin Zheng†

GERAD and École des Hautes Études Commerciales, Montréal, Québec, Canada, GERAD and École Polytechnique of Montréal, Montréal, Québec, Canada, and Institut National des Télécommunications, France

Received September 29, 1993®

A new coding algorithm (called NTUPLE) for computing the N-tuple code (due to Knop *et al.*) for chemical trees is proposed. The algorithm NTUPLE has a time complexity in $O(N)$, where N is the number of vertices of the tree instance, on a RAM machine and a time complexity in $O(N^2)$ on a machine with words of bounded length. A varient of the N-tuple code, called the centered N-tuple code, is also introduced, and a coding algorithm (called CENTERTUPLE) to compute it is provided. The algorithm CENTERTUPLE has a time complexity in $O(N)$ on a RAM machine and in $O(N \log N)$ on a machine with words of bounded length. C implementations of both coding algorithms are included.

1. INTRODUCTION

A molecular code is a sequence of symbols representing a given molecule.[1,2] Names of molecules according to the IUPAC rules[3-5] can therefore be viewed as molecular codes. However, there are other codes using more restricted alphabets and with proven desirable properties. Numerous codes have been proposed.[6-27] A digital code, i.e., a sequence of digits, is particularly desirable for computer handling of structural diagrams of molecules. Indeed, for such codes there is no necessity to have delimiters (e.g., blanks) between successive symbols. As pointed out in ref 1, molecular codes play the role of a translator of a structure to a code and of a code to a structure. Through their corresponding codes, chemical structural diagrams may be stored, displayed, or printed by a computer. Molecular codes have found many different uses, such as numbering of atoms in molecules,[28] discernment of symmetry in a structure,[29] enumeration and graphical representation of isomers,[1,2,30-36] and their systematic analysis in structure–property and structure–activity studies.[37-40]

Read[41] lists some desirable properties for a molecular code such as being linear, unique, well-defined, brief, simple, easily comprehensible to chemists, and efficiently encodable and decodable. Not many molecular codes have all of these properties. The N-tuple code for chemical trees, due to Knop *et al.*[1,2,10] is one of the few which enjoy all of them. For a tree with N vertices it consists of N nonnegative integers smaller than N. Moreover, for chemical trees with N vertices, which have a maximum degree bounded by a small integer (i.e., usually ≤ 4), it consists of N digits. Before giving the definition of the N-tuple code, we first recall that a sequence (or string) of integers (or digits) $S = s_1 s_2 ... s_l$ is *lexicographically larger* than a sequence of integers (or digits) $S' = s'_1 s'_2 ... s'_{l'}$ (i.e., $S \geq S'$) if (i) there exists an integer j, $1 \leq j \leq \min\{l,l'\}$, such that $s_i = s'_i$ for all $i = 1,2,...,j-1$ and $s_j > s'_j$ or (ii) $l > l'$ and $s_i = s'_i$ for all $i = 1,2,...,l'$. This is similar to the ordering of words in dictionaries. A set $S_1, S_2, ..., S_d$ of d finite sequences of integers (or digits) is in *lexicographic order* (respectively *reverse lexicographic order*) if $S_1 \leq S_2 \leq ... \leq S_d$ (respectively $S_1 \geq S_2 \geq ... \geq S_d$). A sequence (or string) S obtained by concatenation of d finite sequences $S_1, S_2, ..., S_d$ of integers (or digits), i.e., $S = S_1 S_2 ... S_d$, is *lexicographically maximum* if for any permutation σ of the indices $S \geq S_{\sigma(1)} S_{\sigma(2)} ... S_{\sigma(d)}$. Recall that a rooted tree T_r[42] is a tree T in which one vertex (r) is indicted as the root.

The N-tuple code (or N-tuple for short) of a rooted tree T_r is recursively defined as follows (see refs 1, 2, and 10): the N-tuple of a pendant vertex is 0; let d be the degree of the root r of T_r; let $v_1, v_2, ..., v_d$ be the adjacent vertices of r; delete from T_r the root r together with its adjacent edges; compute the N-tuples of the subtrees rooted at $v_1, v_2, ..., v_d$; concatenate them in such a way as to obtain a lexicographically maximum sequence S; the N-tuple of T_r is the concatenation of d and S. For an unrooted tree T, all vertices of maximum degree are chosen successively as roots; the N-tuples of the resulting rooted trees are computed. The N-tuple code of T is defined to be the lexicographically maximum N-tuple among these. This code has been successfully used for generating isomeric acyclic structures[1,10] (alkanes, alkenes, alkynes, alkanols, etc.). It is also a building block of Randić's compact code[14,15,22] for coding molecular graphs with cycles. In ref 10, an algorithm is proposed to generate trees and rooted trees through implicit enumeration of N-tuples. Finding an N-tuple representing a tree requires on average $O(N^2)$ operations on a RAM machine (note that the tree is unknown before its N-tuple is found).

In this paper, we first present an algorithm, called NTUPLE, to compute the N-tuple code for chemical trees. Next we define a variant of this code, the centered N-tuple code (or CN-tuple code for short), which has all of the basic properties of the N-tuple code and can be more efficiently computed. A coding algorithm, called CENTERTUPLE, is provided for this purpose. It is shown that the N-tuple code for chemical trees of Knop *et al.*[1,2,10] can be obtained in $O(N)$ time on a RAM machine and in $O(N^2)$ time on a machine with words of bounded length. The CN-tuple code is obtained in $O(N)$ and in $O(N \log N)$ time in these two cases, respectively. Recall that a RAM machine (e.g., Aho *et al.*[43] and Garey and Johnson[44]) is a theoretical model of a computer with words of unbounded length, unlimited internal memory, and unit time for all basic operations. Both coding algorithms are easy to implement by hand. C implementations for these coding algorithms are given in Charts 1 (NTUPLE) and 2 (CENTERTUPLE).

† GERAD and École des Hautes Études Commerciales.
‡ GERAD and École Polytechnique of Montréal.
§ Institut National des Télécommunications.

Chart 1. C Program for the NTUPLE Code

```
                    /* Computation of the CN-Tuple Code */

      #include <stdio.h>

      struct cellule
      {
        int v;
        struct cellule *succ;
      };

      typedef struct cellule *cel;

      /* g: current tree T'; tree: oriented tree  */
      cel *g, *tree;

      /* list: list of the leaves of T'           */
      cel list=NULL;

      /* degre: array of the degrees              */
      /* leaves: array of the degrees in T'       */
      int *degre, *leaves;

      /* vertex is the number of vertices         */
      /* nbv is the number of vertices in g       */
      int nbv, vertex;

      /* Initialization of a cel c */
      cel init(c,v)
      cel c;
      int v;
      {
        if (c = (cel)malloc(sizeof(struct cellule)))
  {
    c->v = v;
    c->succ = NULL;
  }
        else
  printf("no more memory\n");

        return c;
      }

      /* Input the tree */
      int enter(fp)
      FILE *fp;
      {
        cel c;
        int i,v;

        fscanf(fp,"%d",&vertex);

        if (vertex<3)
  return vertex;
        g = (cel *)calloc(vertex,sizeof(cel));
        tree = (cel *)calloc(vertex,sizeof(cel));
        degre = (int *)calloc(vertex,sizeof(int));
        leaves = (int *)calloc(vertex,sizeof(int));

        for (i=0;i<vertex;i++)
  {
    tree[i] = NULL;
    fscanf(fp,"%d",&v);
    degre[i] = leaves[i] = v;
  }
        for (i=0;i<vertex;i++)
  while (1)
      {
        fscanf(fp,"%d",&v);
        if (v==0)
  break;
        c = init(c,i);
        c->succ = g[v];
        g[v] = c;
        c = init(c,v);
        c->succ = g[i];
        g[i] = c;
      }
        return vertex;
      }

  /* Recursive lexicographic comparison of two codes      */
  /* -1, 0, 1 are respectively returned for               */
  /* code[v]>code[w], code[v]=code[w], code[v]<code[w]    */
  int compare(n,p)
  int n,p;
  {
    int c;
    cel r,s;

    /* comparison of the first digit */
    if (degre[n]==degre[p])
      {
        r = tree[n];
        s = tree[p];
        while(r!=NULL && s!=NULL)
          {
    /* recursive call */
    if ((c=compare(r->v,s->v))!=0)
      return c;
    else
      {
        r = r->succ;
        s = s->succ;
      }
          }
        if (r==NULL)
          if (s==NULL)
    return 0;      /* the codes are equal     */
          else
    return 1;      /* n has the maximum code */
    else
      {
        if (s==NULL)
          return -1;   /* p has the maximum code */
        else
          return c;
      }
          }
        else
          {
    if (degre[n]>degre[p])
      return -1;     /* p has the maximum code */
    else
      return 1;      /* n has the maximum code */
          }
    }

    /* rank the codes in the lexicographic order         */
    /* p represents the vertex whose code is concatenated */
    /* dad is the father of p in the tree                */
    void ajout(p,dad)
    cel p;
    int dad;
    {
      cel q,r;

      r = q = tree[dad];
      while(q!=NULL)
        {
          /* p->v is at least as large as q->v */
          if (compare(p->v,q->v)!=1)
    break;
          r = q;
          q = q->succ;
        }
      p->succ = q;
      if (r==q)
        tree[dad] = p;
      else
        r->succ = p;
    }
    /* Remove a leaf in T' and call ajout */
    void order(s)
    int s;
    {
      int dad;
      cel p,q;

      nbv--;
      dad = g[s]->v;
      if (--leaves[dad]==1)
        {
      g[s]->succ = list;
      list = g[s];
      g[s] = NULL;
    }

  q = g[dad];
  if (q->v==s)
    g[dad] = g[dad]->succ;
  else
    {
      for(p=q;(p->succ)->v!=s;p=p->succ);
      q = p->succ;
      p->succ = (p->succ)->succ;
    }
```

Chart 1 (continued)

```
    ajout(q,dad);
  }

  /* deallocate the memory */
  void freelist(l)
  cel l;
  {
    cel p;

    for (p=l;p!=NULL;p=l)
      {
        l = p->succ;
        free(p);
      }
  }

  /* Display the N-tuple code */
  void affiche(v)
  int v;
  {
     cel p;

     printf("%d",degre[v]-1);
     p = tree[v];
     while (p!=NULL)
       {
         affiche(p->v);
         p = p->succ;
       }
  }

memory()
{
  int i;

  for (i=0;i<vertex;i++)
    {
      freelist(g[i]);
      freelist(tree[i]);
    }
  freelist(list);
  free(g);
  free(tree);
  free(degre);
  free(leaves);
}

 /* Main program */

main(argc, argv)
int argc;
char *argv[];
{
  int v,w,i;
  cel p,r;
  FILE *fp;

  if (argc==1 || (fp=fopen(*++argv,"r"))==NULL)
    {
```

```
     printf("You need to enter the name of an
existing file\n");
       return;
     }

   nbv =  enter(fp);
   fclose(fp);
   if (nbv>=3)

     {
       for (i=0;i<vertex;i++)
 if (degre[i]==1)
   order(i);

       /* there are still leaves to be removed */
       while(nbv>=3)
 {
   p = r = list;
   list = NULL;
   while (p!=NULL)
     {
       order(p->v);
       r = p;
       p = p->succ;
       free(r);
     }
 }

       v = list->v;

       /* T' has two vertices: step 2(c) */
       if (nbv==2)
 {
   w = (list->succ)->v;
   if (compare(v,w)==1)
     {
       i = v;
       v = w;
       w = i;
     }
   /* w has the maximum code */
   ajout(g[v],v);
 }
       /* display the code */
       printf("The code is: ");
       degre[v]++;
       affiche(v);
     }
   else
     {

          printf("\nThe code is: ");
          if (vertex==2)
    printf("10\n");
          else
    printf("0\n");
        }

   }
```

2. COMPUTING THE N-TUPLE CODE

Computing the N-tuple of a tree implies recursive computation of lexicographically maximum N-tuples of subtrees. This is an easy task in view of the following result.

Theorem 1: *Let $v_1,v_2,...,v_d$ be the vertices adjacent to the root r of a tree T. The sequence $S = S_1S_2...S_d$ obtained by concatenation of the d N-tuples of the subtrees rooted at $v_1,v_2,...,v_d$ is lexicographically maximum if and only if $S_1 \geq S_2 \geq ... \geq S_d$.*

Proof: *Necessity*. Consider two sequences S_i and S_j of length l_i and l_j for $i,j \in \{1,2,...,d\}$. Assume without loss of generality that $l_i \leq l_j$. If $S_i = S_j$, any order of S_i and S_j can be considered. Assume now that $S_i \neq S_j$. Let S'_j be the sequence obtained by considering the l_i first digits (on the left) of S_j. If $l_i = l_j$, $S_j = S'_j$; since $S_i \neq S_j$, we have $S_i \neq S'_j$. If $l_i < l_j$, we show by contradiction that $S_i \neq S'_j$. Indeed, assume that $S_i = S'_j$. Let T_i and T_j be the subtrees corresponding to the sequences S_i and S_j, respectively. Applying the recursive definition of the N-tuple code to construct T_j from S_j leads to a subtree T'_j corresponding to S'_j and isomorphic to T_i. The remaining integers of S_j are not used, hence a contradiction: S_j cannot be an N-tuple code. This proves the claim. Assume that $S_i \geq S_j$. Let us consider a first permutation σ of the indices of the sequences such that $\sigma(i) < \sigma(j)$. We define a second permutation σ' such that $\sigma(k) = \sigma'(k)$ for all $k \in \{1,2,...,d\}$ with $k \neq i, j$, $\sigma'(i) = \sigma(j)$, and $\sigma'(j) = \sigma(i)$. It is easy to check that $S_{\sigma(1)}S_{\sigma(2)}...S_{\sigma(d)} \leq S_{\sigma'(1)}S_{\sigma'(2)}...S_{\sigma'(d)}$. Considering all pairs of sequences S_i and S_j shows that if S is lexicographically maximun, then $S_1 \geq S_2 \geq ... \geq S_d$.

Sufficiency. Assume $S_1 \geq S_2 \geq ... \geq S_d$. We only need to show that $S = S_1S_2...S_d \geq S_{\sigma(1)}S_{\sigma(2)}...S_{\sigma(d)}$ for any permutation σ of the indices such that $S_{\sigma(1)} \geq S_{\sigma(2)} \geq ... \geq S_{\sigma(d)}$. These permutations differ only in the positions of identical sequences.

Chart 2. C Program for the CENTERTUPLE Code

```
     /* Computation of the N-tuple Code */


  #include <stdio.h>

  struct cellule
  {
    int elt;
    struct cellule *succ;
  };

  typedef struct cellule *cel;

  /* g contains the orientations not considered yet    */
  /* tree represents the oriented tree                 */
  cel *g, *tree;

  /* list of the vertices of maximum degree            */
  cel list=NULL;

  /* degre is the array of degrees, mx the maximum one  */
  int *degre, mx;

  /* vertex: number of vertices in T                   */
  int vertex;

  /* Initialization of a cel c */
  cel initcel(c,v)
  cel c;
  int v;
  {
    if (c = (cel)malloc(sizeof(struct cellule)))
      {
        c->elt = v;
        c->succ = NULL;
      }
    else
    printf("No more memory\n");
  return c;
}

/* Input the tree */
void enter(fp)
FILE *fp;
 {
   cel c;
   int i,v;

    fscanf(fp,"%d",&vertex);
   g = (cel *)calloc(vertex,sizeof(cel));
   tree = (cel *)calloc(vertex,sizeof(cel));
   degre = (int *)calloc(vertex,sizeof(int));

   for (i=0;i<vertex;i++)
     fscanf(fp,"%d",&degre[i]);

  for (i=0;i<vertex;i++)
    while (1)
      {
fscanf(fp,"%d",&v);
if (v==0)
  break;
c = initcel(c,i);
c->succ = g[v];
g[v] = c;
c = initcel(c,v);
c->succ = g[i];
g[i] = c;
      }
}

/* deallocate the memory */
void freelist(l)
cel l;
{
  cel p;

  for (p=l;p!=NULL;p=l)

    {
      l = p->succ;
      free(p);
    }
}
```

```
  /* Compare the codes for n and p                  */
  /* Return -1, 0,1 respectively for n>p, n=p, n<p */
  /* rootn and rootp represent the fathers         */
  int compare(n,p,rootn,rootp)
  int n,p,rootn,rootp;
  {
    int c;
    cel r,s;
    /* comparison of the first digit */
    if (degre[n]==degre[p])
      {
        r = tree[n];
        s = tree[p];
        while(r!=NULL && s!=NULL)
  {
    if (r->elt!=rootn && s->elt!=rootp)
       /* recursive call */
      if ((c=compare(r->elt,s->elt,n,p))!=0)
        return c;
      else
        {
  r = r->succ;
  s = s->succ;
        }
    else
      {
        if (r->elt==rootn)
  {
    r = r->succ;
    if (s->elt==rootp)
      s = s->succ;
  }
         else if(s->elt==rootp)
   s = s->succ;
      }

  }
        if (r==NULL)
  if (s==NULL)
    return 0;      /* the codes are equal */
  else
    return 1;      /* n has the maximum code */
        else
  {
    if (s==NULL)
      return -1;   /* p has the maximum code */
    else
      return c;
  }
      }
    else
      {
        if (degre[n]>degre[p])
  return -1;     /* p has the maximum code */
        else
  return 1;      /* n has the maximum code */
      }
  }

  /* insertion of p, whose dad is n */
          /* at its rank in the tree        */
  void insertion(n,p)
  int n;
  cel p;
  {
    cel r,q;

    q = NULL;
    r = tree[n];
    while (r!=NULL)
      {
    /* The code of p->elt is at least equal */
    /* to the code of r->elt?               */
        if (compare(p->elt,r->elt,n,n)!=1)
  break;
        q = r;
        r = r->succ;

    }
  p->succ = r;
  if (q!=NULL)
    q->succ = p;
  else
         /* p is the son of n which has the maximum code */
    tree[n] = p;
}
```

Chart 2 (continued)

```
/* rank the edges and               */
/* orient them in the subtree of root n */
void place(n,temp)
int n,temp;
{
  cel p,q;
  int m;

  p = q = g[n];
  while (p!=NULL)    /* n has some sons in the tree */
    {
      m = p->elt;
      if (temp!=m)
{
  /* CODE(m) */
  place(m,n); /*place the son m by recursivity */
  if (q==g[n])
    g[n] = q = g[n]->succ;
  else
    g[n]->succ = q = p->succ;
  /* p is taken from g and inserted in tree */
  /* CODE(n,p)   */
  insertion(n,p);
  p = q;
}
      else
p = q = p->succ;
    }
}

/*Add double orientations and so associate a code to n */
doubl(n)

int n;
{
  cel p;

  p = g[n];
  if (p!=NULL) /* the code is still unknown for p->elt */
    {
      doubl(p->elt);
      insertion(n,p);
      g[n] = NULL;
    }
}

/* Display the N-tuple Code, preorder traversal */
void dispcode(n,temp)
int n, temp;
{
  cel p;

  printf("%d",degre[n]-1);
  p = tree[n];
  while (p!=NULL)
    {
      if (p->elt!=temp)
      dispcode(p->elt,n);
      p = p->succ;
    }
}

memory()
{
  int i;

  for (i=0;i<vertex;i++)
    {
      freelist(g[i]);
      freelist(tree[i]);
    }
  freelist(list);
  free(g);
  free(tree);
  free(degre);
}

        /* Main Procedure */

main(argc, argv)
int argc;
char *argv[];
{
  int root=0, i,j,v;
  cel c, p, q;
  FILE *fp;

  if (argc==1 || (fp=fopen(*++argv,"r"))==NULL)
    {
      printf("You need to enter the name of an
existing file\n");
      return;
    }
  enter(fp);
  fclose(fp);
  mx = degre[0];
  for (i=1;i<vertex;i++)
    if (degre[i]>mx)
      mx = degre[i];
  /* step 1 */
  while (degre[root]!=mx)
    root++;

/* computation of the code of root */
  place(root,-1);

/* computation of the codes for       */
/* the other vertices of maximum degree */
  c = initcel(c,root);
  list = c;
  for (i=++root;i<vertex;i++)
    if (degre[i]==mx)
      {
doubl(i);
c = initcel(c,i);

c->succ = list;
list = c;
      }

/* search for the code which is */
/* the lexicographic maximum    */
  p = list;
  q = p->succ;
  while (q!=NULL)
    {
      if (compare(p->elt,q->elt,-1,-1)==1)
p = q;
      q = q->succ;
    }

/* Print the code */
  degre[p->elt]++;
  printf("The code is: \n");
  dispcode(p->elt,-1);
}
```

Therefore, there is a unique sequence S corresponding to all of them, which is lexicographically maximum.

Observe that theorem 1 does not hold for arbitrary sequences. Consider $S_1 = 101$, $S_2 = 101100$. We have $S_2 \geq S_1$. However, $S_1S_2 \geq S_2S_1$.

We now discuss efficient computation of the N-tuple code. Let T be a tree and e be an edge of T with u and v as its endpoints. The removal of the edge e leads to two rooted subtrees; let ST_u^e be the subtree rooted at u and ST_v^e be the subtree rooted at v. The computation of the N-tuple of T_w rooted at w, for any w belonging to ST_v^e, requires the computation of the N-tuple for ST_u^e (rooted at u). It follows that the N-tuple for ST_u^e can be used many times when the N-tuple code for T is computed if ST_v^e contains many vertices of maximum degree. Recomputing the N-tuple code for ST_u^e is wasteful and should be avoided.

To make use of the above remark, we define a directed graph G as follows. The set of vertices of G is equal to the set of vertices of T. There is an arc (u, v) in G if and only if the edge $\{u, v\}$ belongs to T and the subtree ST_v^e of T contains a vertex of maximum degree in T. Note that there may be two arcs (u, v) and (v, u) in G for a given pair $\{u, v\}$ of vertices, i.e., for an edge of T. We associate the N-tuple of ST_u^e with the arc (u, v) if it exists.

The N-tuple for a tree T_u rooted at a vertex u of maximum degree can be found in the following way: concatenate the sequences associated with the arcs entering u in such a way that the resulting sequence S is lexicographically maximum and add the decrease of u to the left of S. Once the N-tuple code for a rooted tree T_u has been found, the N-tuple code for one more rooted tree T_v with root v can be easily found by computing only the sequences associated to the vertices on the path from u to v. This procedure is iterated, taking only vertices of maximum degree of T as roots and keeping track of all sequences already computed. The N-tuple code for the undirected tree T can be obtained by choosing the lexicographically maximum one among the N-tuples of all rooted trees T_u of T with vertex u of maximum degree.

Let u be a vertex of T (and G) and $e = \{u, v\}$ be an edge incident to it in T. We define the following operations which correspond, respectively, to finding the code associated with a vertex u and with an arc (u, v).

CODE(u): Determine a reverse lexicographic order of all sequences associated with arcs (v, u) entering u. Concatenate them in that order and add the degree of u to the left of the so-obtained sequence.

CODE(u, e): Add the arc (u, v) to the set of arcs of G. Determine a reverse lexicographic order of all sequences associated with the arcs (w, u) entering u, except the arc (v, u) if it exists. Concatenate these sequences in that order and add the degree of u minus 1 to the left of the so-obtained sequence.

The following algorithm gives the orientations defined above to the edges of T, leading to a directed graph G. It also provides the digit sequences associated with each vertex of maximum degree in T as well as the N-tuple code for T. Recall that a pendant vertex is a vertex of degree one.

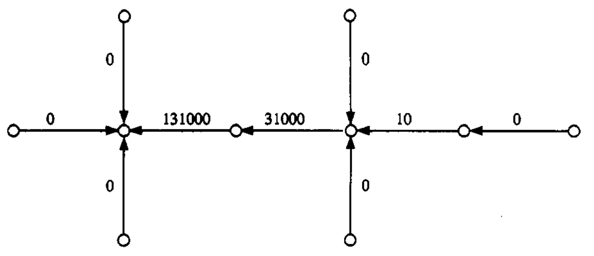
Algorithm NTUPLE. Step 1. Compute the list $\mathcal{L}$ of vertices of maximum degree in T. Choose a vertex w in $\mathcal{L}$ and remove it from $\mathcal{L}$. Let G be the empty graph. Let T' be equal to the input tree T.

Step 2. Repeat the following operations until T' contains only w: 2.1, choose a pendant vertex u of T', distinct from w; 2.2, let $e = \{u, v\}$ be the edge of T' which is incident to u; 2.3, perform CODE(u, e); 2.4, remove e from T'. Perform CODE(w) (the N-tuple for the tree T_w rooted at w is found). Mark w in G.

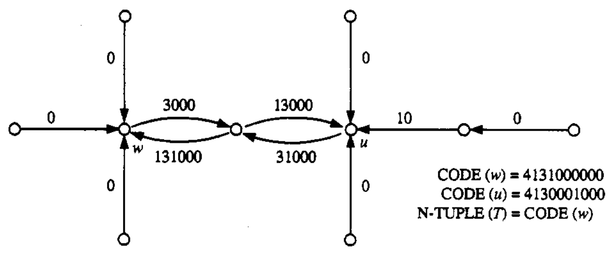
Step 3. Repeat the following operations until $\mathcal{L}$ is empty: 3.1, select a vertex u in $\mathcal{L}$ and remove it from $\mathcal{L}$; 3.2, start with u and iteratively follow the unique arc in G going out of the currently reached vertex until a marked vertex w is reached; 3.3, let $\mathcal{P}$ be the directed path from u to w in G, and set $v = w$; 3.4, repeat the following operations until $v = u$: 3.4.1, let v' be the predecessor of v in $\mathcal{P}$, let $e = \{v, v'\}$, and perform CODE(v, e); 3.4.2, if v' is of maximum degree, perform CODE(v') (the N-tuple for the tree $T_{v'}$ rooted at v' is found); 3.4.3, mark v' in G; 3.4.4, set v to v'.

Step 4. Compare the N-tuple codes found for all rooted trees T_u with u in the initial list $\mathcal{L}$. The N-tuple code of T is the lexicographically maximum one among them.

Figure 1 gives a step-by-step illustration of NTUPLE. By induction, we can show that the sequence assigned to each direction of an oriented edge by the algorithm NTUPLE is



(a) Graph G and digit sequences associated with arcs after Step 2 of algorithm NTUPLE.



(b) Graph G and digit sequences associated with arcs after Step 4 of algorithm NTUPLE.

Figure 1. Step-by-step illustration of algorithm NTUPLE.

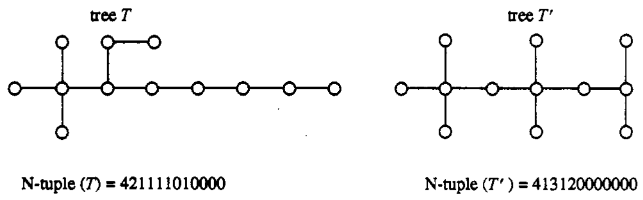


Figure 2. N-tuple codes vs numbers of branchings.

exactly the digit sequence associated with it as described above. Correctness of algorithm NTUPLE follows.

After giving their recursive definition of the N-tuple code, Knop *et al.*[10] do not discuss further how to compute it. As mentioned above, their paper aims at listing chemical trees rather than at finding the N-tuple code of given trees. This last problem is discussed in a later paper by Knop *et al.*[45] In an effort to simplify code derivation, five rules for paper-and-pencil encoding are proposed. However, as Kirby[46] pointed out, one of these rules is not universally applicable. The rules of NTUPLE are precisely defined and can be easily implemented by hand.

In ref 45, it is also claimed that an important property of the N-tuple code is that it orders acyclic structures according to their skeletal branching (branching in a graph is identified through the appearance of branched vertices, which have degrees of three or larger). This property does not seem to true in general. Indeed, consider the two trees T and T' given in Figure 2 along with their N-tuple codes. The N-tuple code of T is lexicographically larger than that of T', but T' has more branchings.

3. CENTERED N-TUPLE CODE

First, we define the *center* (or *centers*) of a tree T algorithmically as follows: delete simultaneously all pendant vertices of T together with their adjacent edges, thus obtaining a smaller tree; repeat this operation until a tree with a single vertex or two vertices (which are adjacent) is obtained; the remaining vertex or vertices are called the center or centers of T. Figure 3 illustrates the above definition. It has long

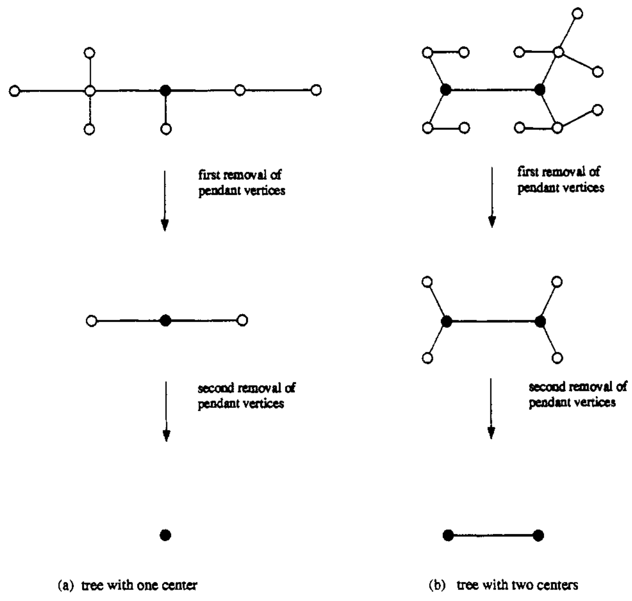

first removal of pendant vertices
first removal of pendant vertices
second removal of pendant vertices
second removal of pendant vertices
(a) tree with one center
(b) tree with two centers

Figure 3. Centers of trees.

been known[42] that the center(s) of a tree T is (are) the middle vertex (vertices) of the longest path of T. If the longest path has an even number of vertices, then T has two centers; otherwise, T has one center.

The *centered N-type* (CN-tuple) code for a tree T is defined as follows: find the N-tuple code for each rooted tree with its root at a center of T; select the lexicographically maximum one among these N-tuples as the CN-tuples code of T.

Since most of the basic properties of the N-tuple code are obtained from the N-tuples of rooted trees,[1] the CN-tuple code has these properties too. We can show also that two trees with the same associated CN-tuple code are isomorphic. This follows from the fact that the CN-tuple code can be used to reconstruct the tree from its center (or from the center used to define it when there are two centers) in a unique way. There is thus a one-to-one mapping between unlabeled trees and CN-tuple codes.

The algorithm NTUPLE described above can easily be modified to give the CN-tuple code. It suffices to compute the center or centers of T at the outset and consider these vertices as root(s) instead of the vertices of maximum degree. However, there are other possibilities. We next describe an algorithm for the CN-tuple that appears to be simpler than the modified version of NTUPLE. It proceeds by coding edges from the pendant vertices toward the center(s).

As algorithm NTUPLE, this algorithm gives the orientation of T, toward the center or centers and both ways between them if there are two, as well as the digit sequence associated with each arc, and the CN-tuple code for T.

Algorithm CENTERTUPLE. Step 1. Set T' to be the input tree T. Let $\mathcal{L}$ be the list of pendant vertices of T. Repeat the following steps until T' has at most two vertices: 1.1, set $\mathcal{L}'$ equal to the empty set (*$\mathcal{L}'$ will be the set of vertices adjacent to the pendant vertices of T'*); 1.2, repeat the following steps until $\mathcal{L}$ is empty: 1.2.1, choose a vertex u in $\mathcal{L}$ and remove it from $\mathcal{L}$; 1.2.2, let $e = \{u, v\}$ be the edge of T' incident to u; 1.2.3, perform CODE(u, e); 1.2.4, add v to $\mathcal{L}'$, if it is not yet there; 1.2.5, remove e from T'; 1.3, set $\mathcal{L}$ equal to $\mathcal{L}'$.

Step 2. 2.1, If T' has a single vertex, then go to step 2.5; 2.2, if T' has two vertices r_1 and r_2, let e be the edge of T joining them; 2.3, perform CODE(r_1, e); 2.4, perform CODE(r_2, e); 2.5, for each vertex r of T', perform CODE(r) and denote the resulting sequence CN(r); 2.6, choose the lexicographically maximum one among the (at most two) sequences CN(r) as the CN-tuple code of T.

These rules are just a realization of the recursive definition of the CN-tuple code. Figure 4 gives a step-by-step illustration of their use. Since to find the CN-tuple code for a tree T implies computing the N-tuples for at most two rooted trees, CENTERTUPLE is easier to implement by hand than NTUPLE.

4. COMPUTATIONAL COMPLEXITY OF NTUPLE AND CENTERTUPLE

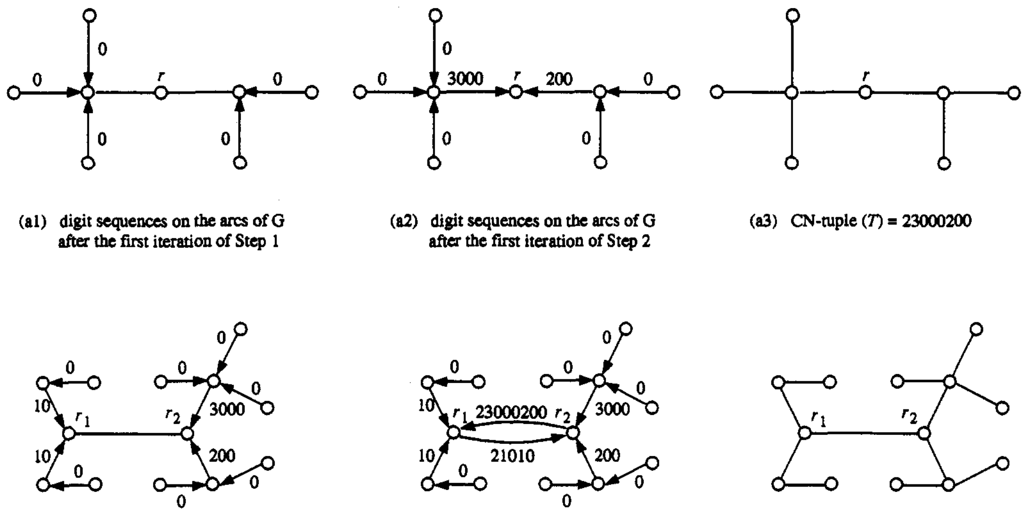
We consider here the computational complexity of algorithms NTUPLE and CENTERTUPLE for chemical trees, i.e., trees with a maximum degree bounded by a small integer.

4.1. Complexity on a RAM Machine. We first consider the RAM machine model (ramdom access machine) which admits words with any length.

Theorem 2: *On a RAM machine, both the N-tuple code and the CN-tuple code for a chemical tree with N vertices can be obtained in $O(N)$ time.*

Proof. We only prove the theorem for the N-tuple code since the proof is similar for the CN-tuple code. Finding vertices of maximum degree takes $O(N)$ time. Since the trees considered can have a maximum degree bounded by a small integer (always less than or equal to nine), we can treat a digit sequence S obtained in NTUPLE as a number $n(S)$. Let the length $l(S)$ of a digit sequence S be the number of digits in S (the length of $n(S)$ is therefore $[\log_{10} S]$). For two digit sequences S and S', S is lexicographically larger than or equal to S' if and only if $n(S) \geq 10^{l(S)-l(S')} \times n(S')$. Suppose S is lexicographically larger than or equal to S'. Then the concatenation of S and S', i.e., the digit sequence SS', is equal to $S \times 10^{l(S')} + S'$ and its length is equal to $l(S) + l(S')$. So the lexicographic comparison as well as the concatenation of two digit sequences S and S' can be done in $O(1)$ time. The time for defining an arc (u, v) of G and computing its digit sequence (i.e., performing CODE(u, e)) in NTUPLE is also $O(1)$ (or a constant). The reason is the following: the ranking of the sequences $S_1,S_2,...,S_d$ (where d is less than or equal to the maximum degree in the trees considered) according to their reverse lexicographic order can be done in $O(d \log d)$ time (where $O(d \log d)$ is the complexity of sorting d numbers). Then to concatenate $S_1,S_2,...,S_d$ requires $d-1$ multiplications and $d-1$ summations. So the total number of operations to obtain the concatenation of $S_1,S_2,...,S_d$ according to their lexicographic order is $O(d \log d) = O(1)$ for d is less than or equal to a constant. Thus, operation CODE(u, e) takes $O(1)$ time. A tree has $N-1$ edges. Thus, G has at most $2N-2$ arcs; i.e., there are at most $2N-2$ direction assignments, i.e., application of CODE(u, e), in NTUPLE. Hence, the overall time complexity for assignment of direction to edges and computation of the sequences associated with the resulting arcs is $O(N)$. Similarly, when direction assignments are completed, the time to obtain an N-tuple for a rooted tree T_r with root at a vertex r of maximum degree is also $O(1)$. In other words, operation CODE(u) takes $O(1)$ time. To find the lexicographically maximum one among the N-tuples associated with all trees rooted at maximum degree vertices requires $b-1$ comparisons, where b is the number of vertices of maximum degree in the tree. As b is less than N, this takes $O(n)$ time. Thus, the total time to find the N-tuple code for the input tree is in $O(N)$.

4.2. Complexity on a Real Machine. We now consider a machine with words of bounded length (which is the case for real world computers). Then the comparison between two



(a1) digit sequences on the arcs of G after the first iteration of Step 1

(a2) digit sequences on the arcs of G after the first iteration of Step 2

(a3) CN-tuple (*T*) = 23000200

(b1) digit sequences on the arcs of G after the second iteration of Step 1

(b2) digit sequences on the arcs of G after Step 2.4

(b3) CN (r_1) = 3230002001010
CN (r_2) = 3210103000200
CN-tuple (*T*) = CN (r_1)

Figure 4. Illustration of algorithm CENTERTUPLE.

digit sequences has to be made digit by digit (or possibly a few digits at a time) in NTUPLE and CENTERTUPLE. This entails the following.

Theorem 3: *On a machine with words of bounded length, the CN-tuple code and the N-tuple code for a chemical tree with N vertices can be obtained in $O(N \log N)$ and $O(N^2)$* time, respectively.

Proof. As shown in the proof of theorem 2, both algorithms require $O(N)$ applications of $\text{CODE}(u, e)$ and $\text{CODE}(u)$. On a real machine, computations being done digit by digit implies these operations will take $O(N)$ time instead of $O(1)$ time on a RAM machine. Hence, both algorithms are in $O(N^2)$. However, a more precise analysis yields a better result for CENTERTUPLE. Assume first the maximum degree $d^{\max}$ of T to be at most three. Then each application of $\text{CODE}(u, e)$ implies the comparison of two digit sequences, say S and S', if u is of degree three in T and none if u is of degree two or one in T. Assume without loss of generality $l(S) \leq l(S')$. Then the resulting sequence SS' (or $S'S$) has a length $l(S) + l(S') \geq 2l(S)$. Define a *good comparison* for a vertex v to be one in which the digit $S(v)$ (equal to the degree of v minus 1) corresponding to this vertex appears in the shortest of two sequences. Then for each v the number of good comparisons involving v is not more than $\log_2 N$, as the length of the sequence involving v doubles at least after each good comparison. The total number of comparisons is equal to the total number of good comparisons and hence to $N \times O(\log N)$, i.e., $O(N \log N)$. Operation $\text{CODE}(u)$ is applied at most twice, once for each center. The result follows. If $d^{\max}$ is greater than three but bounded by nine, consider a reverse lexicographic ranking and concatenation of $d \leq d^{\max} - 1$ sequences $S_1, S_2, ..., S_d$. Define a good comparison for a vertex v to be one in which $S(v)$ does not belong to the longest sequence. When $\text{CODE}(u, e)$ is applied, v will be involved in at most $d - 1$ good comparisons and the resulting sequence will have at least twice the length of that one containing $S(v)$. The total number of good comparisons involving v is thus in $O(d \log N)$, which is equal to $O(\log N)$ as d is bounded by the constant $d^{\max} - 1$–. As the total number of comparisons when $\text{CODE}(u, e)$ is applied cannot exceed the total number of good comparisons for all vertices, it is in $O(N \log N)$. Applying $\text{CODE}(v)$ implies $O(2d^{\max} N)$, which is equal to $O(N)$, comparisons in all. This completes the proof.

5. COMPUTATIONAL COMPARISON

In this section, we discuss and compare two programs in C corresponding to the implementation of algorithms NTUPLE and CENTERTUPLE. Listings of these programs are given in Charts 1 and 2.

In the programs, instead of storing the intermediate sequence on each arc, we rank the incoming arcs to each vertex in reverse lexicographic order of the digit sequences associated with them and keep pointers to the first, second, ..., one of them. When a direction is assigned to an edge (u, v) from u to v, the digit sequence associated to this direction can be obtained as follows: traverse T_u starting at u in a depth-first search manner such that the current searched incoming edge is the first edge in the ranking of the unsearched incoming edges and write down $d_w - 1$, where d_w is the degree of the current visited vertex w in the traversal. After assignment of all possible directions to the input tree, finding the N-tuple for a rooted tree T_u with root u (of maximum degree) can be done as above except that the first number written down is the degree d_w of u instead of $d_w - 1$. Thus, the comparisons between digit sequences in the programs are made digit by digit.

The input file of both programs is as follows:

N (the number of vertices

$d_0, d_1, ..., d_{N-1}$ (the degree sequence of the input tree and the vertices of the tree are labeled $0, 1, 2, ..., N - 1$)

the list of neighbors of vertex 0

the list of neighbors of vertex 1 with labels larger than 1

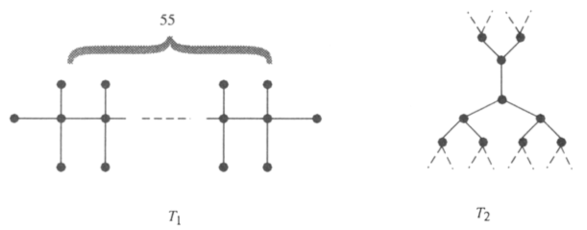
...

the list of neighbors of vertex i with labels larger than i

...

(each list ends with a zero).

In Figure 5, two trees together with their N-tuple codes, CN-tuple codes, and computing times are given. Accurate values for computing times were obtained by computing the codes 200 times and dividing. Computations are made on a SUN SPARC 2. For trees with many vertices of maximum



T_1 has 167 vertices

```
N-tuple code:
4333333333333333333333333333
333333333333333333333333333
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000

Time:  0.19 sec.


CN-tuple code:
4333333333333333333333333333
0000000000000000000000000000
000000000000000000000000000
333333333333333333333333333
0000000000000000000000000000
00000000000000000000000000000

Time:  0.105 sec.
```

T_2 has maximum degree 3 and 94 vertices

```
N-tuple code:
32222222200200220020022200
20022002002222002002200200
222002002200200222002002200
200220020020000

Time:  0.9 sec.


CN-tuple code:
3222200200220020022200200
2200200222002002200200
222002002200200222200200
2200200222002002200200

Time:  0.055 sec.
```

Figure 5. Computational experience on two large trees.

degree, computation of the C*N*-tuple code is clearly faster than computation of the *N*-tuple code.

ACKNOWLEDGMENT

The first and second authors have been supported by FCAR (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) Grant 92EQ1048. The first author has also been supported by an NSERC (Natural Sciences and Engineering Research Council of Canada) grant to HEC and NSERC Grant OGP0105574. The second author has also been supported by NSERC Grant OGP0036426 and a FRSQ (Fonds pour la Recherche en Santé du Québec) fellowship. The fourth author ahs been supported by an international fellowship from NSERC.

REFERENCES AND NOTES

(1) Trinajstić, N.; Nikolić, S.; Knop, J. V.; Müller, W. R.; Szymanski, K. *Computational Chemical Graph Theory*; Ellis Horwood: New York, 1991; pp 36–77.
(2) Nikolić, S.; Knop, J. V.; Müller, W. R.; Szymanski, K.; Trinajstić, N. *Computer Generation of Certain Classes of Molecules*; SKTH: Zagreb, 1985.
(3) *IUPAC—Nomenclature of Inorganic Chemistry*; Pergamon: London, 1971.
(4) *IUPAC—Nomenclature of Organic Chemistry*; Pergamon: London, 1979.
(5) Davidson, S. Compact Numeric Alkane Codes Derived from IUPAC Nomenclature. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 417–422.
(6) Mitchell, A. D. *Chemical Nomenclature*; Arnold: London, 1948.
(7) Wiswesser, W. J. *A Line-Formula Chemical Notation*; Crowell: New York, 1954.
(8) Randić, M. On the Recognition of Identical Graphs Representing Molecular Topology. *J. Chem. Phys.* **1974**, *60*, 3920–3928.
(9) Randić, M. Canonical Numbering of Atoms in a Molecule and Graph Isomorphism. *J. Chem. Inf. Comput. Sci.* **1977**, *17*, 171–180.
(10) Knop, J. V.; Müller, W. R.; Jeričević, Ž.; Trinajstić, N. Computer Enumeration and Generation of Trees and Rooted Trees. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 91–99.
(11) Uchino, M. Algorithms for Unique and Unambiguous Coding and Symmetry Perception of Molecular Structure Diagrams. 5. Unique Coding by the Method of Orbit Graphs. *J. Chem. Inf. Comput. Sci.* **1982**, *22*, 201–206.
(12) Herndon, W. C. Canonical Labeling and Linear Notation for Chemical Graphs. *Stud. Phys. Theor. Chem.* **1983**, *28*, 231–242.
(13) Tošić, R.; Doroslovački, R.; Gutman, I. The Boundary Code. *Match* **1986**, *19*, 219–228.
(14) Randić, M. Compact Molecular Codes. *J. Chem. Inf. Comput. Sci.* **1986**, *26*, 136–148.
(15) Randić, M. Compact Codes. 2. Bicyclic Staturated Hydrocarbons. *Croat. Chim. Acta* **1986**, *59*, 327–343.
(16) Kirby, E. C. Coding and Factorization of Polycyclic Chemical Graphs. *Stud. Phys. Theor. Chem.* **1987**, *51*, 529–536.
(17) Herndon, W. C.; Bertz, S. H. Linear Notations and Molecular Graph Similarity. *J. Comput. Chem.* **1987**, *8*, 367–374.
(18) Herndon, W. C.; Bruce, A. J. Perimeter Codes for Benzenoid Aromatic Hydrocarbons. *Stud. Phys. Theor. Chem.* **1987**, *51*, 491–513.
(19) Klein, D. J.; Herndon, W. C.; Randić, M. On the Classification of Polyhex Polymers. *New J. Chem.* **1988**, *12*, 71–76.
(20) Herndon, W. C.; Bruce, A. J. Linear Notation for Benzenoid Aromatic Hydrocarbons. Molecular Similarity Based on Notations Similarity. *J. Math. Chem.* **1988**, *2*, 155–169.
(21) Kvasnička, V.; Pospíchal, J. Canonical Indexing and Constructive Enumeration of Molecular Graphs. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 99–105.
(22) Nikolić, S.; Trinajstić, N. Compact Molecular Codes for Annulenes, Aza-Annulenes, Annule-Noannulenes, Aza-Annulenaonnulenes, Cyclazines and Aza-Cyclazines. *Croat. Chim. Acta* **1990**, *63*, 155–170.
(23) Kruglyak, Yu. A.; Dochtmanov, M. E. Coding System for Quasipolycyclic Structures. *J. Mol. Struct.* (*THEOCHEM*) **1992**, *258*, 199–208.
(24) Balaban, T. S.; Filip, P. A.; Ivanciuc, O. Computer Generation of Acyclic Graphs Based on Local Vertex Invariants and Topological Indices. Derived Canonical Labeling and Coding of Trees and Alkanes. *J. Math. Chem.* **1992**, *11*, 79–105.
(25) Neville, E. H. The Codifying of Tree-Structure. *Proc. Cambridge Philos. Soc.* **1953**, *165*, 381–385.
(26) Balaban, A. T. Applications of Graph Theory in Chemistry. *J. Chem. Inf. Comput. Sci.* **1985**, *25*, 334–343.
(27) Balaban, A. T. Theoretical Indexes and Their Uses—A New Approach for the Coding Alkanes. *J. Mol. Struct.* (*THEOCHEM*) **1988**, *165*, 243.
(28) Hendrickson, J. B.; Toczko, A. G. Unique Numbering and Cataloging of Molecular Structures. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 171–177.
(29) Randić, M. Resonance Energies of Very Large Benzenoid Hydrocarbons. *Int. J. Quantum Chem.* **1980**, *14*, 549–586.
(30) Trinajstić, N.; Jeričević, Ž.; Knop, J. V.; Müller, W. R.; Szymanski, K. Computer Generation of Isomeric Structures. *Pure Appl. Chem.* **1983**, *55*, 379–390.
(31) Knop, J. V.; Szymanski, K.; Jeričević, Ž.; Trinajstić, N. Computer Enumeration and Generation of Benzenoid Hydrocarbons and Identification of Bay Regions. *J. Comput. Chem.* **1983**, *4*, 23–32.
(32) He, W.; He, W. Generation and Enumeration of Planar Polycyclic Hydrocarbons. *Tetrahedron* **1986**, *42*, 5291–5299.
(33) Brunvoll, J.; Cyvin, S. J.; Cyvin, B. N. Enumeration and Classification of Benzenoid Hydrocarbons. *J. Comput. Chem.* **1987**, *8*, 189–197.
(34) Brunvoll, J.; Cyvin, S. J.; Cyvin, B. N. Enumeration and Classification of Coronoid Hydrocarbons. *J. Chem. Inf. Comput. Sci.* **1987**, *27*, 14–21.
(35) Cioslowski, J. Computer Enumeration of Polyhexes Using Compact Naming Approach. *J. Comput. Chem.* **1987**, *8*, 906–915.
(36) Müller, W. R.; Szymanski, K.; Knop, J. V.; Nikolić, S.; Trinajstić, N. On the Enumeration and Generation of Polyhex Hydrocarbons. *J. Comput. Chem.* **1990**, *11*, 223–235.
(37) Bawden, D. Computerized Chemical Structure-Handling Techniques in Structure-Activity Studies and Molecular Property Prediction. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 14–22.
(38) Dubois, J.-E. In *Chemical Applications of Graph Theory*; Balaban, A. T., Ed.; Academic Press: London, 1976; Chapter 11.
(39) Stuper, A. J.; Brügger, W. E.; Jurs, P. C. *Computer Assisted Studies of Chemical Structure and Biological Function*; Wiley: New York, 1979.
(40) Dubois, J.-E.; Mercier, C.; Panaye, A. Darc Topological System and Computer-Aided Design. *Acta Pharm. Jugosl.* **1986**, *36*, 135–169.
(41) Read, R. C. A New System for Designation of Chemical Compounds. 1. Theoretical Preliminaries and Coding of Acyclic Compounds. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 135–149.
(42) Harary, F. *Graph Theory*; Addison-Wesley: Reading, MA, 1971; 2nd printing.
(43) Aho, A. V.; Hopcroft, J. E.; Ullman, J. D. *The Design and Analysis of Computer Algorithms*; Addison-Wesley: Reading, MA, 1974.
(44) Garey, M. R.; Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; Freeman: New York, 1979.
(45) Knop, J. V.; Müller, W. R.; Szymanski, K.; Trinajstić, N. Computer-Oriented Molecular Codes. In *Computatinal Chemical Graph Theory*; Rouvray, D. H., Ed.; Nova Science Publishers: New York, 1990; pp 9–32.
(46) Kirby, E. C. Coding and Enumeration of Trees That Can Be Laid upon a Hexagon Lattice. *J. Math. Chem.* **1992**, *11*, 187–197.