Chemical Trees Enumeration Algorithms

R. Aringhieri, P. Hansen, and F. Malucelli.

Chemical Trees Enumeration Algorithms.

4OR, 1:67-83, 2003

DOI: 10.1007/s10288-002-0008-9

4OR manuscript No.
(will be inserted by the editor)

Chemical Tree Enumeration Algorithms ⋆

Roberto Aringhieri[1], Pierre Hansen[2], Federico Malucelli[3]

[1] DISMI, Università di Modena e Reggio Emilia, Viale Allegri 13, 42100 Reggio Emilia - Italy
[2] GERARD - HEC, Université de Montréal, 5255 Avenue Decelles, Montréal, H3T 1V6 - Canada
[3] Dipartimento di Elettronica, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano - Italy

Received: date / Revised version: date

Abstract In the chemical community the need for representing chemical structures within a given family and of efficiently enumerating these structures suggested the use of computers and the implementation of fast enumeration algorithms. This paper considers the isomeric acyclic structures focusing on the enumeration of the alkane molecular family. For this family, Trinajstić et al. ([22]) devised an enumeration algorithm which is the most widely known and utilized nowadays. Kvasnička and Pospichal ([16]) have proposed an algorithmic scheme which, from the computational complexity point of view, can prove to be more efficient than the Trinajstić one, nevertheless, this algorithm, to the best of our knowledge, has never been implemented. Indeed an efficient implementation requires the introduction of non trivial data structures and other computational tricks. The main contribution of this paper consists of the definition of the implementation details of Kvasnička-Pospichal's algorithm, in a comparison of Trinajstić's, Kvasnička-Pospichal's and two new algorithms, proposed here, in terms of both computational complexity analysis and running times.

1 Introduction

The enumeration of the alkane molecular family is a problem whose roots go back to the nineteenth century. Alkanes are chemical compounds with identical molecular formulae and weights but differing at least in some physical and/or chemical properties leading to different structures. Alkanes belong

⋆ These algorithms have been developed in parallel when the first two authors are visiting at EPFL in Lausanne

to the general class of isomeric acyclic structures. The term isomerism and the definition of isomers were introduced in 1830 by Berzelius ([6]).

The alkane molecular family is partitioned into *classes of homologous molecules*, that is molecules with the same numbers of carbonium and hydrogen atoms; the n-th class is characterized by the following formula:

$$C_n H_{2n+2}, \quad n = 1, 2, \ldots \tag{1}$$

An alkane molecule is usually represented by indicating the carbonium atoms and their (primary) links, omitting to represent hydrogen atoms (see figure 1), whose placement can be obtained automatically. When n is sufficiently large ($n \geq 4$), to the same formula correspond molecules with different structures. Take as an example butane (C_4H_{10}): there are two possible molecular structures (nbutane and isobutane) as depicted in figure 2.

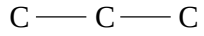

Fig. 1 propane C_3H_8

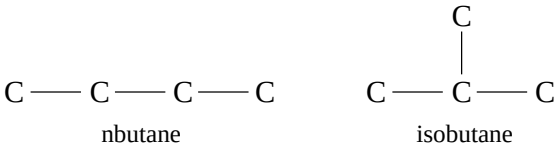

Fig. 2 C_4H_{10} butanes

Alkanes can be represented by acyclic connected graphs whose nodes correspond to carbonium atoms and edges correspond to the primary links. There is a one-one correspondence between elements in the family of alkanes and elements in the family of the connected acyclic graphs (i.e. trees) whose nodes have degree less than or equal to 4, therefore an alkane molecule can be represented by such a degree constrained tree.

At the end of the 19-th century, Cayley was the first to realize the importance of tree mathematical theory in the enumeration of isomeric acyclic structures ([7–9]). He enumerated manually the alkane isomers and alkyl radicals with up to $n = 13$ carbonium atoms, though introducing some counting errors. This enumeration had a considerable impact on chemists of that time.

Subsequently, the development took a turn to computer-oriented methods for the enumeration and generation of isomeric structures: the development of techniques for the generation of graphs (chemical structures) by computing devices has made possible the direct enumeration of isomeric

acyclic structures by way of enumerating algorithms. The pioneering contributions in this area were made by Lederberg and his group ([17,18]) and Corey and his group ([10,11]).

One of the possible applications of chemical tree enumeration algorithms is the investigation of different chemical and physical properties on massive quantities of molecules through the computation of topological indices, e.g. the efficient computation carried out in [2] of the *Hyper-Wiener* index defined in [19].

For these enumeration purposes, the algorithm most widely used by chemists is that devised by Trinajstić et al. in [22]. In 1991, Kvasnička and Pospichal ([16]) proposed an algorithmic scheme which in principle can be more efficient than the other one, depending on its actual implementation. However the details described in [16] are not sufficient to implement a computer code. Here we propose, to our knowledge, the first implementation of the Kvasnička and Pospichal algorithm. Moreover, in this paper we introduce two new algorithms for the generation of the alkanes molecular family based on **Reverse Search** ([3,4]) which is a technique to avoid duplicated trees in the enumeration.

In section 2 we introduce the molecular codes used to encode alkanes, the two algorithms proposed in literature ([22,16]) are described in section 3, and our new algorithms are presented in section 4. The details needed to implement our algorithms as well as that in [16] are also reported. The computational complexity analysis of all algorithms is carried out in section 5. Section 6 reports running times comparison of the algorithms applied to molecules with up to $n = 27$ carbonium atoms. Some conclusions are drawn in section 7.

Hereafter, we denote by n-tree and n-rooted respectively a tree and a rooted tree with n nodes. For the sake of simplicity, we use the expression "tree" or "alkane" as a synonymous with "degree constrained tree".

2 Tree codes

In order to efficiently enumerate degree constrained trees, data must be suitably encoded: the motivation is that a code can be easily handled by computer program. In [20], the authour 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 ([20]). Different numerical codes have been proposed in the literature ([5,12, 15,20–22]).

The codes used in this paper are the N-TUPLE code ([20,22]) and the *centered* N-TUPLE code proposed in [12] hereafter called CN-TUPLE code for short. We define the N-TUPLE code for a rooted tree, then we extend this definition to an unrooted tree. The definition of CN-TUPLE code for a tree is a variant of that for the N-TUPLE.

First we recall that a sequence of integers $C = c_1 c_2 \dots c_l$ is *lexicographically larger* than a sequence $C' = c'_1 c'_2 \dots c'_{l'}$ if i) there exists an integer j,

$1 \leq j \leq \min\{l, l'\}$, such that $c_i = c'_i$ for all $i = 1, \ldots, j-1$ and $c_j > c'_j$ or ii) $l > l'$ and $c_i = c'_i$ for all $i = 1, \ldots, l'$.

The code of a n-rooted tree, rooted in r, T_r is recursively defined as follows: the N-TUPLE of a pendant node is 0; let g be the number of nodes $(v_1, v_2, \ldots, v_g)$ adjacent to r. In order to obtain the N-TUPLE of tree T_r, rooted in r, we delete from T_r the root r together with its incident edges; we compute the N-TUPLES of the subtrees rooted in $v_1, v_2, \ldots, v_g$; then we concatenate these N-TUPLES in such a way as to obtain a lexicographically maximum sequence S. The N-TUPLE code of T_r is the concatenation of g and S.

For an unrooted tree T, all nodes of maximum degree are chosen successively as roots and the corresponding N-TUPLES are computed. The N-TUPLE code of T is defined as being the lexicographically maximum N-TUPLE among these. For example, the N-TUPLE codes for the butanes in figure 2 are 2100 and 3000, respectively.

To obtain the CN-TUPLE code, we first identify the *center* (or the centers) of a tree T. We delete simultaneously all pendant nodes of T together with their incident edges, thus obtaining a smaller tree; we repeat this operation until a tree with a single node (or two adjacent nodes) is obtained which is, by definition, the center (or the centers) of T. The CN-TUPLE code for a tree T is the N-TUPLE code of the rooted tree with the root in the center (or the the lexicographically maximum N-TUPLE of the two rooted trees rooted in the centers).

The computation of the N-TUPLE code and the CN-TUPLE code for a n-tree requires respectively $O(n^2)$ and $O(n \log n)$ time as reported in [12].

3 Previous work

In this section we report two different enumeration algorithms: the algorithm proposed by Trinajstić et al. ([22]), briefly referred to as TNKMS, and that outlined by Kvasnička and Pospichal ([16]), briefly referred to as KP.

The algorithm TNKMS is based on the following idea: a n-tree can be obtained by adding one node (and one edge) to a $(n-1)$-tree in such a way that the degree constraint still holds. We refer to this class of algorithms as *one-to-one enumeration*.

One-to-one enumeration does not guarantee avoiding the generation of multiple copies of the same tree. An example of enumeration is reported in figure 3: each arrow represents how the tree can be derived and the label is the number of different ways in which it can be obtained. The multiple generation of a tree starting from the same tree or from different trees is the crucial point emerging from this class of enumeration algorithms. Therefore the introduction of some rules which guarantee the uniqueness of enumerated trees is required. Note that TNKMS as described in [22] does not introduce any such rules.

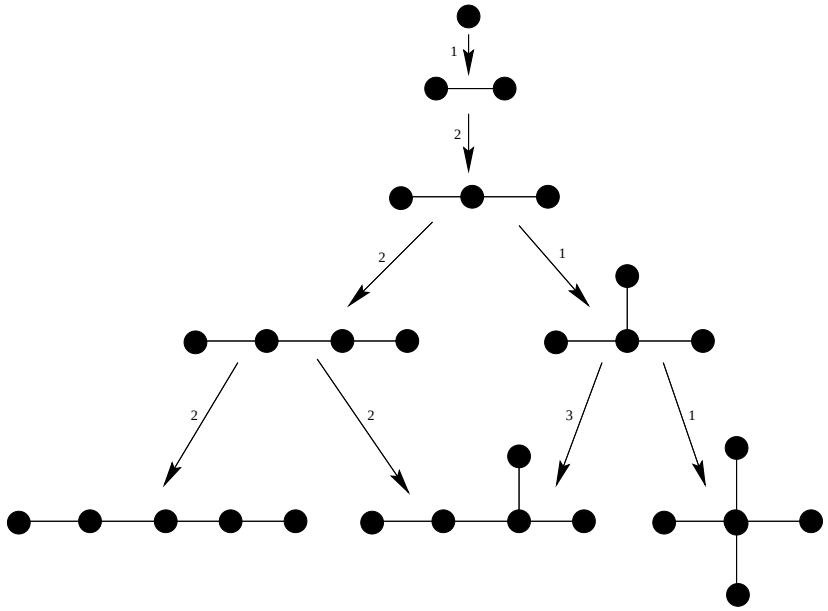

Fig. 3 Example of one-to-one enumeration

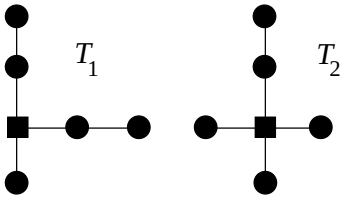

Fig. 4 Example of constructive enumeration (part i)

The algorithm KP, also called *constructive enumeration* algorithm, is based on the following observation: an n-tree can be decomposed into one node and one or more rooted trees; then, each rooted tree can be decomposed in the same way, and so on. Therefore, a n-tree can be enumerated joining one node together with one or more rooted trees in such a way that the sum of nodes in the rooted trees is equal to $n-1$. An example is given in figure 4: T_1 is generated by joining the squared node with two 2-rooted and one 1-rooted trees; T_2 is generated by joining three 1-rooted and one 2-rooted trees.

In [16] a more formal description of this idea is reported. In particular, this description is based on the rules defined by Jordan's theorem ([14]) which guarantee the unique enumeration of trees.

Theorem 1 (Jordan 1869) *Let T be a n-tree, the following three different cases should be separately considered:*

(odd case) For an odd $n = 2k + 1$ there exist a unique node, called the ***centroid****, such that all (two or more) incident subtrees are composed, at most, by k nodes.*

For an even $n = 2k$ there exist either

(even case i) a unique node - centroid - such that all (three or more) incident subtrees are composed of less than k nodes, or

(even case ii) a unique edge, called the **bicentroid***, such that the incident two subtrees are composed exactly of k nodes.*

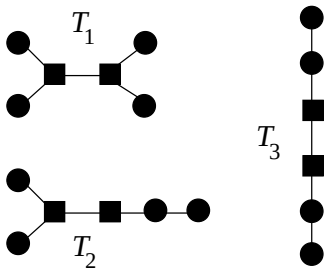

Fig. 5 Example of constructive enumeration (part ii)

Note that to generate a degree constrained tree, we have to take into account that we are not allowed to combine more than four rooted trees. For example, the set of 10-trees can be generated joining one node together with three or four rooted trees in such a way that the sum of nodes is equal to 9 (even case i). Moreover, we can combine two rooted trees with 5 nodes (even case ii).

Jordan's theorem defines a way to compose rooted trees, e.g. one 4-rooted and one 5-rooted for a 9-tree. It is clear that different trees can be obtained by composing different rooted trees with the same cardinality. Figure 5, T_1, T_2 and T_3 shows the trees obtained by combining all the pairs of 3-rooted trees.

Finding the possible combinations of t rooted trees with no more than k nodes is equivalent to finding all the possible combinations of t numbers smaller than or equal to k in such a way that their sum is equal to $n-1$. Note that t has to be greater than or equal to 2 according to Jordan's theorem and less than or equal to 4 according to the degree constraint.

KP generates all n-trees computing all the possible combinations of t rooted trees and, for each combination, combining the rooted trees with the same cardinality in all possible ways. We have found neither computational results for this approach nor any computer code implementing it, therefore we devised our own implementation which required to specify data strucutures and clarify the details missing in [16].

Note that both TNKMS and KP encode the enumerated trees by using the N-TUPLE code.

3.1 Implementation details of KP algorithm

KP algorithm requires computing of all the sets of p-rooted trees, say R_p, for $p = 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor$ to generate the set of n-trees according to the rules defined by Jordan's theorem and to the degree constraint.

As discussed above, a combination of t rooted trees having no more than p nodes to build an n-tree is equivalent to a combination of t integer numbers smaller than or equal to p whose sum is equal to $n-1$. We denote such a combination with

$$(a(x), b(y), c(z), d(w)) \tag{2}$$

which means that x a-rooted trees and y b-rooted trees (and so on) are used to build one n-tree and $ax + by + \ldots = n - 1$.

Moreover, each combination of numbers generating n-tree is subject to the following conditions according to Jordan's theorem and to the degree constraint.

(Condition 1) The combination $(a(x), b(y), c(z))$ yielding a p-rooted tree must be such that:
1. $ax + by + cz = p - 1$;
2. $1 \leq x + y + z \leq 3$, (degree constraint);
3. $a, b, c \in \{1, \ldots, p-1\}$, $x, y, z \in \{0, .., 3\}$;

(Condition 2) The combination $(a(x), b(y), c(z), d(w))$ yielding an n-tree (with n odd) must be such that:
1. $ax + by + cz + dw = n - 1$;
2. $2 \leq x + y + z + w \leq 4$, (degree constraint);
3. $a, b, c, d \in \{1, \ldots, \lfloor n/2 \rfloor\}$, (Jordan's theorem, odd case);
4. $x, y, z, w \in \{0, .., 4\}$;

(Condition 3) The combination $(a(x), b(y), c(z), d(w))$ yielding an n-trees (with n even) must be such that:
1. $ax + by + cz + dw = n - 1$;
2. $3 \leq x + y + z + w \leq 4$, (degree constraint);
3. $a, b, c, d \in \{1, \ldots, n/2\}$, (Jordan's theorem, even case i);
4. $x, y, z, w \in \{0, .., 4\}$

Referring to algorithm KP summarized in figure 6, we first compute all sets (`R[p]`) of rooted trees with up to $\left\lfloor \frac{n}{2} \right\rfloor$ nodes generating all combinations (`a(x),b(y),c(z)`) such that condition 1 holds. Then, the set of n-trees (`T`) is generated: for odd n, `KP` generates all combinations (`a(x),b(y),c(z),d(w)`) such that condition 2 holds and the corresponding n-trees are generated by procedure `Compose`; in the case of even n, the n-trees are generated from all combinations verifying condition 3 and all combinations of two rooted trees with $\frac{n}{2}$ nodes, that is, the case of bicentroid in Jordan's theorem.

```
procedure KP (n);
begin
   R[1..floor(n/2)] : sets of rooted trees; T : set of trees;
   R[1] := {0}; n_of_alk := 0; k := floor( n/2 ); T := emptyset;
   for p = 2 to k do begin               {rooted tree generation}
      R[p] := emptyset;
      for each [a(x),b(y),c(z)] s.t. condition 1 holds do
          R[p] := Union( R[p], Compose( R[a], x, R[b], y, R[c], z, emptyset, 0 ) );
   end;
   if Odd( n )
      then                                {odd n-tree generation}
         for each [a(x),b(y),c(z),d(w)] s.t. condition 2 holds do
             T := Union( T, Compose( R[a], x, R[b], y, R[c], z, R[d], w );
      else begin                          {even n-tree generation}
         for each [a(x),b(y),c(z),d(w)] s.t. condition 3 holds do
             T := Union( T, Compose( R[a], x, R[b], y, R[c], z, R[d], w ) );
         T := Union( T, Compose( R[k], 2, emptyset, 0, emptyset, 0, emptyset, 0 );   {bicentroids n-tree}
      end;
   n_of_alk := CardinalityOf( T );
   output( n_of_alk, elapsed_time, T );
end.
```

Fig. 6 Kvasnička and Pospichal algorithm

4 Two new enumeration algorithms

In this section we present two new enumeration algorithms whose main difference with respect to TNKMS algorithm is the introduction of rules avoiding the multiple generation of trees. The two algorithms, namely AHM ([1]) and HPVK ([13]), have the same structure but use different codes to represent trees: AHM uses the CN-TUPLE code while HPVK uses the N-TUPLE code.

As already discussed in section 3 (see figure 3), the enumeration scheme suggests the use of some rules guaranteeing the uniqueness of the enumerated trees. A first attempt can be carried out by implementing a list of already enumerated trees: a new tree is accepted if and only if it does not belong to the list. The use of a list of codes (ordered or not) requires a large amount of computational resources, in terms of time and memory, as we will show in section 6.

In order to avoid the use of such a list, we introduced two techniques which prevent the generation of multiple copies of the same tree, namely *Reverse Search* (RS) and *Symmetry Detection* (SD).

Let T_o, T_d be two trees such that T_d can be generated adding one pendant node to T_o in such a way that the degree constraint holds. Let the *tree generation function* $G_T : T_o \mapsto T_d$ be the function describing such operation. Moreover, let $D(T_o) = \{T_d : G_T(T_o) = T_d\}$ be the multiset of trees generated from T_o and $O(T_d) = \{T_o : G_T^{-1}(T_d) = T_o\}$ be the multiset of trees from which T_d can be generated. Note that multiple copies of the

same tree may belong to $D(T_o)$ and $O(T_d)$. For example:

$$D(200) = \{2100,\ 2100,\ 3000\}$$
$$O(31000) = \{3000,\ 3000,\ 3000,\ 2100,\ 2100\}.$$

Given a tree and a root r, for any node v let $l_r(v)$ be the distance (i.e. the number of edges) between v and r.

RS has been introduced by Avis and Fukuda ([3, 4]) for efficiently enumerating polyhedra's vertices and it can be considered as a particular search on a graph.

We summarize the basic idea of Reverse Search in the following, referring to [4] for a more formal and detailed description. Let G be a connected graph whose vertices are the objects to be enumerated, and suppose that we have an objective function to be maximized over all vertices of G. A local search algorithm on G is a deterministic procedure which explores the neighborhood of a given vertex of G to find a vertex improving the objective function value. The search is repeated until there exist no better neighboring vertex. For simplicity, suppose that the considered local search avoids cycling and there is only one optimal vertex x^*. Consider now the digraph T having the same vertices of G and the edges (x, x') corresponding to the moves from x to x' performed by the local search algorithm starting from each vertex of G. Since x^* is the unique global optimum, T turns out to be a spanning arborescence having x^* as sink. Thus if we trace backward T from x^* sistematically, e.g. by depth first search, or, in other words, *reversing the local search*, we can enumerate all vertices. Note that RS does not require storing any information about visited vertices.

By applying RS to our problem with the lexicographical order as the objective function, we derive the following *enumeration rule*.

Enumeration Rule 1 (RS) *An enumerated tree $T_d = G_T(T_o)$ is accepted if and only if T_o is the lexicographically maximum among the codes in $O(T_d)$.*

For example, $T_d = 31000$ is accepted if $T_o = 3000$ and it is not accepted if $T_o = 2100$.

Let us now illustrate the Simmetry Detection technique, that is the way to recognize identical subtrees or *symmetries* within a given T_o. Consider T_o as composed by two or more identical subtrees; clearly, it is possible to generate two or more identical T_d by adding a pendant node to any of the identical subtrees of T_o. For example, $T_d = 31000$ can be generated three times from $T_o = 3000$ (see figure 7).

Consider T_o with root r and two subtrees S_1 and S_2 of T_o with roots v and w respectively. S_1 and S_2 are symmetrical if and only if i) the code of S_1 is equal to that of S_2 and ii) $l_r(v) = l_r(w)$. The enumeration rule is:

Enumeration Rule 2 (SD) *Given S_i symmetrical subtrees of T_o with $i = 1, \ldots, q$, $2 \leq q \leq 4$, the trees generated from T_o are those obtained expanding S_q thus skipping $S_1, \ldots, S_{q-1}$.*

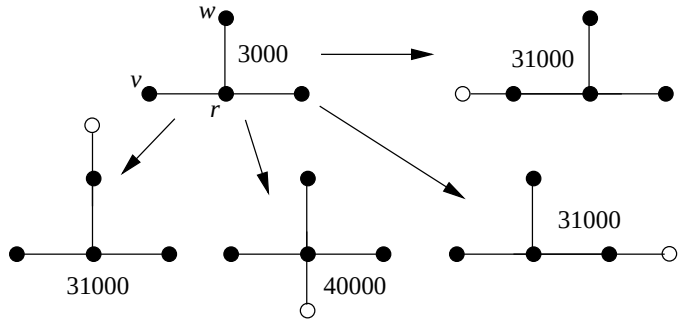

Fig. 7 Symmetries

We observe that the enumeration rule 2 can be applied recursively to all subtrees of S_q and to all non symmetrical subtrees of T_o. Note that the choice of S_q instead of S_1 is only motivated by implementation reasons (see §4.1).

The use of RS and SD techniques guarantees the uniqueness of enumerated trees. Enumeration rule 1 avoids multiple enumeration of trees from different T_o while enumeration rule 2 avoids multiple enumeration of trees from the same T_o.

4.1 Implementation details

We report only the main details concerning the implementation of algorithm AHM. These observations can be extended to HPVK in a straightforward manner considering the fact that this algorithm uses N-TUPLE codes, hence it requires the use of simpler data structures.

In particular AHM makes use of the *code tree* data structure (CTREE for short) which is exemplified in figure 8.

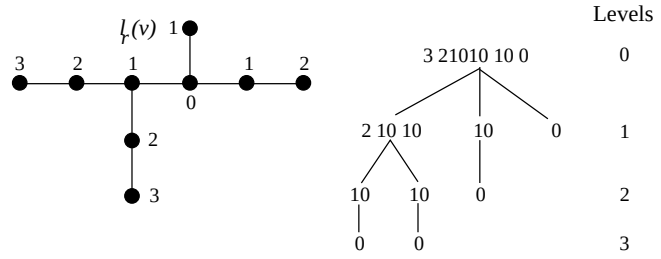

Fig. 8 A tree with the indication of $l_r(v)$ and the corresponding CTREE

CTree is a rooted tree with root in r which is decomposed into levels where a *level* i is the set of nodes v such that $l_r(v) = i$.

Starting from a pendant node whose code is 0, the code of a node v is obtained concatenating the number of subtrees incident in v with the subtree codes in lexicographical order.

CTree drives the code computation of a new enumerated tree and the application of the enumeration rules 1 and 2.

First of all, we observe that the subtree code $C = c_1c_2 \dots c_l$ can be easily manipulated: if a pendant node is added to the root of subtree the new code is $C' = (c_1 + 1)c_2 \dots c_l0$; otherwise, if a pendant node is deleted from the root, the code is $C' = (c_1 - 1)c_2 \dots c_{l-1}$.

Figure 9 shows how the CTree of T_o can be used to compute a new code T_d: first we compute the new code of the right subtree where a pendant node is added (200); then the codes at same level are lexicographically ordered (200, 110) to correctly compose the subtree code at the upper level (2 200 110); these operations (*ordering* and *composing* steps) are repeated until level 0 is reached, that is until the new code T_d is computed.

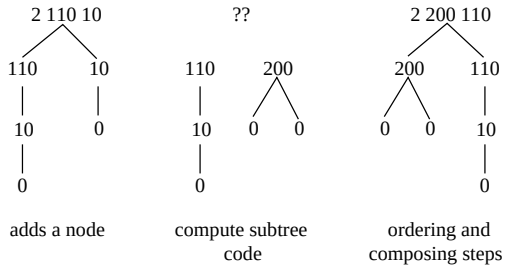

Fig. 9 New code computation

The same procedure can be utilized to compute T_o code from T_d. The new subtree code is obtained by deleting a pendant node, instead of adding it. Then, the ordering and composing steps are repeated in the same way as before until level 0 is reached.

The enumeration rule 2 can be implemented by performing a *breadth first search* (BFS) on CTree : finding a symmetry is equivalent to finding two nodes at the same level with the same code.

The enumeration rule 1 can be implemented by generating all possible T_o codes and checking if the lexicographically largest one corresponds to the code of the current T_o. Note that the enumeration rule 2 can be also used to compute T_o trees.

Procedure `GenSetD` (see figure 10) taking as input a tree T_o, generates the set $D(T_o)$ by applying the enumeration rules 1 and 2. In particular, the CTree corresponding to the input tree is generated and the codes belonging to level 1 are inserted into a queue `Q`. Following the BFS search, at each

iteration a code u is extracted from `Q`; if u is equal to the first code in `Q` then it is skipped (enumeration rule 2); otherwise, if the degree constraint holds (`adj(u)<=3`), the procedure tries to add a pendant node to u and recomputes the code corresponding to the new tree, if the enumeration rule 1 applies.

The pseudo-code of `GenSetD` is depicted in figure 10.

```
procedure GenSetD ( T, D );
begin
   D, O : sets of trees; Q : queue;
   MakeCTreeOf( T ):
   D := emptyset; Q := { v : v = succ(r(T)) };
   while Q is not empty
   do begin
      u := GetFirst( Q ); Q := Q \ { u };
      if ( code(u) is not equal code(next(u)) )      {enum. rule 2}
      then begin
         Union( Q, { v : v = succ(u) } );
         if ( adj(u)<=3 )
         do begin
            t := NewCode( u, l(u) );
            m := ComputeMax( O(t) );
            if T = m then Insert( D, t );            {enum. rule 1}
                     else Discard( t );
         end;
      end;
   end;
   output( D );
end.
```

Fig. 10 Procedure `GenSetD`

The procedure `OneToOneEnumeration` (see figure 11) generates the set of all n-trees by calling procedure `GenSetD`. The pseudo-code reported is amenable to both AHM and HPVK algorithms.

5 Complexity of algorithms

In this section, we compare the computational complexities of AHM, HPVK and KP algorithms. The complexity of TNKMS is not reported since the algorithm structure is not described precisely in the original paper [22].

5.1 KP algorithm

The complexity of KP depends on the number of combinations (2) subject to conditions 1, 2 and 3. We recall that the problem of finding all possible

```
procedure OneToOneEnumeration ( n );
begin
   Trees[1..n] : sets of trees;
   Trees[1] := { 10 }; n_of_trees[1] := 1;
   Trees[2..n] := emptyset; n_of_trees[2..n] := 0;
   node:=2;
   while ( node <= n )
   do begin
      while Trees[node-1] is not empty
      do begin
         t := Get( Trees[node-1] );
         GenSetD( t, Trees[node] );
         n_of_trees[node] := CardinalityOf( Trees[node] );
      end;
      node := node + 1;
   end;
   output( n_of_trees[n], elapsed_time, Trees[n] );
end.
```

Fig. 11 AHM and HPVK algorithms

combinations of t rooted trees with no more than k nodes is equivalent to finding all possible combinations of t numbers less than or equal to k in such a way that their sum equals to $n-1$.

Let C_c, N_c, N_t and N_q be respectively the cost of composing a new code, the number of possible combinations of rooted trees, the number of triplets verifying condition 1 and the number of quadruplets verifying both conditions 2 and 3. The complexity of cycles in `KP` (see figure 6) is given by

$$O(k \times C_c \times N_c \times N_t) \tag{3}$$

for the generation of p-rooted trees, and

$$O(C_c \times N_c \times N_q) \tag{4}$$

for the generation of n-trees. The proof of the following Lemma is straightforward.

Lemma 1 *The number of triplets* $(a(x), b(y), c(z))$ *verifying condition 1 are* $O(n^2)$. *The number of quadruplets* $(a(x), b(y), c(z), d(w))$ *verifying both conditions 2 and 3 are* $O(n^3)$.

Let K_p be the cardinality of p-rooted set. The number of possible combinations of m p-rooted trees ($p = 1, .., \lfloor \frac{n}{2} \rfloor$) is $O(H^m)$ where H is the maximum cardinality of rooted trees sets used to generate the p-rooted set in procedure `Compose`, that is $H = \max\{K_a, K_b, K_c\}$ with $ax + by + cz = p - 1$ according to Condition 1. Note that the function `Compose` in `KP` generates a new code by composing the subtree codes in lexicographical order in linear time. Therefore, C_c is $O(n)$. According to (3) and (4), the rooted tree generation cycle is $O(n^4 K^3_{\lfloor \frac{n}{2} \rfloor})$ while the n-tree generation cycle is $O(n^4 K^4_{\lfloor \frac{n}{2} \rfloor})$.

Moreover, for even n, the generation of bicentroids is $O(nK^2_{\lfloor \frac{n}{2} \rfloor})$. Therefore, we have

Theorem 2 *Algorithm KP is* $O(n^4 K^4_{\lfloor \frac{n}{2} \rfloor})$.

5.2 AHM and HPVK algorithms

AHM and HPVK have the same computational complexity since the only difference between the two algorithms is that HPVK does not use the CTree data structure to drive the implementation of the enumeration rules 1 and 2.

The complexity of AHM depends on the time required to compute the code of a new enumerated tree T_d (procedure `NewCode`) and to compute T_o codes when the algorithm checks the enumeration rule 1 (procedure `ComputeMax`). We show that this computation can be carried out in linear time. As depicted in figure 9, this computation consists of the subtree code computation and the consequent ordering and composing steps.

The code of a modified subtree (i.e. obtained by adding or removing a pendant node) is computed in constant time (see §4.1).

The ordering step consists of checking the lexicographical order of two codes which can be carried out in linear time in the code length d. The worst case arises when from a code we generate a complete CTree with L levels. Let d_j be the length of a subtree code at level $j = 0, .., L$, therefore we have:

$$\begin{aligned} d_0 &= n, \\ d_1 &= \frac{n-1}{4}, \\ d_j &= \frac{d_{j-1}-1}{3}, \quad j = 2, \ldots, L-1, \\ d_L &= 0. \end{aligned}$$

In fact, by definition of CTree , $d_0 = n$ and $d_L = 0$. Since the CTree is complete, the root has 4 subtrees whose code lengths are equal to $\frac{n-1}{4}$. At level $j \geq 2$, the length of the 3 subtree codes is $\frac{d_{j-1}-1}{3}$. The number of codes compared are 2 at level $j = 2, \ldots, L-1$, and 3 at level $j = 1$. Each

comparison requires $O(d_i)$ steps. Hence the overall number of steps is:

$$\begin{aligned} 3d_1 + \sum_{j=2}^{L-1} 2d_j \leq 3\sum_{j=1}^{L-1} d_j &= \left[\frac{n-1}{4} + \frac{\frac{n-1}{4}-1}{3} + \ldots\right] \\ &\leq \left[\frac{n-1}{4} + \frac{n-1}{4\,3} + \frac{n-1}{4\,3^2} + \ldots\right] \\ &\leq 3\sum_{j=1}^{L-1} \frac{n-1}{4\,3^j} = \frac{3}{4}(n-1)\sum_{j=1}^{L-1}\frac{1}{3^j} \\ &\leq \frac{3}{4}(n-1)\sum_{j=1}^{\infty}\frac{1}{3^j} = \frac{3}{4}(n-1)\frac{1}{2} = \frac{3}{8}(n-1) \end{aligned}$$

which is $O(n)$.

Finally, the concatenation of two or more subtree codes is $O(n)$. Therefore, the computation of a new code after adding or deleting a pendant node is $O(n)$.

Procedure `GenSetD` (figure 10) visits once all $n-1$ nodes of CTree in the worst case. For each one, the application of enumeration rule 2 is $O(n)$, the encoding of a new enumerated tree T_d is $O(n)$ and the application of enumeration rule 1 is $O(n^2)$: in fact, the codes of all $n-1$ possible $T_o = G_T(T_d)$ are obtained in $O(n)$. This means that `GenSetD` is $O(n^3)$.

By consequence, the overall complexity of procedure `OneToOneEnumeration` (figure 11) is given by

$$\sum_{i=1}^{n-1} i^3 S_{i-1}.$$

where S_i is the number of i-trees. Since the following inequalities hold

$$\sum_{i=1}^{n} i^3 S_i \leq \sum_{i=1}^{n} i^3 S_n \leq S_n \sum_{i=1}^{n} i^3 = S_n \left(\sum_{i=1}^{n} i\right)^2 = S_n \frac{n^2(n+1)^2}{4} \quad ,$$

we have theorem 3.

Theorem 3 *Algorithms AHM and HPVK are $O(n^4 S_n)$.*

◇

Table 1 Computational complexities of reported algorithms

Enumeration algorithms			
KP	HPVK	AHM	TNKMS
$O(n^4\, K^4_{\lfloor \frac{n}{2} \rfloor})$	$O(n^4\, S_n)$	$O(n^4\, S_n)$	- -

Table 1 summarizes the complexity of algorithms. The reported complexities depend on the number of carbonium atoms n, that is the size of

each generated tree, and the size of the set of trees (rooted tree in the case of KP algorithm) used by the algorithms to generate the required class of alkanes. For example, for $n = 20$, $K_{10} = 879$ while $S_{20} = 251731$ (see table 2). Looking at these results, it is not possible to select *a priori* which is the best algorithm since it is difficult to clearly evaluate any single contribution to the complexity functions. Therefore, running time comparisons are also required.

6 Running time comparisons

This section reports some computational results based on our running time experiences made on Sylicon Graphics Indingo 4000, processor Risc IP20 100 MHz, 48 Mbytes of main memory. AHM, HPVK and KP are implemented in Pascal while TNKMS in Fortran77, as reported in [22].

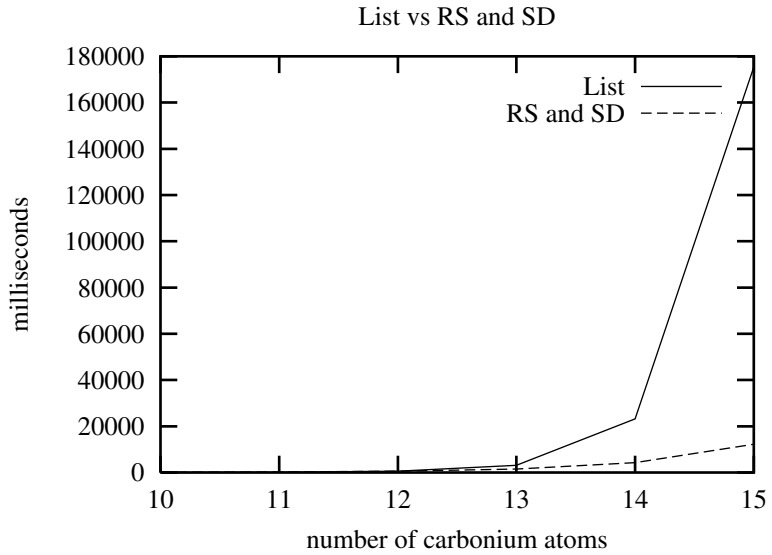

Fig. 12 New code computation

Figure 12 plots the running times performances obtained by comparing the use of Reverse Search and Symmetry Detection techniques and the use of a list of codes in order to avoid multiple generations. The joint use of these techniques outperforms the use of an unordered list. Moreover, they allow to save memory since they do not require storing all generated codes. Therefore, in principle, these techniques make it possible to enumerate classes of alkanes with larger numbers of carbonium atoms.

Table 2 reports the running times of the algorithms presented in the course of the paper. TNKMS appears the slowest algorithm to generate

the alkane molecular family. This might be due to the lack of rules avoiding multiple enumerations of trees. AHM and HPVK have similar running times and their gaps depend on the different computer implementations. KP is surprisingly the best algorithm: with $n = 21$, KP is about 4, 5 times faster than HPVK and AHM and about 1320 times faster than TNKMS.

Table 2 KP, HPVK, AHM and TNKMS running time

n	S_n	**HPVK**	**AHM**	**TNKMS**	**KP**	$\sum_{p=1}^{\lfloor \frac{n}{2} \rfloor} K_p$
16	10359	0”	0”	45”	0”	161
17	24894	2”	2”	2’43”	0”	161
18	60523	6”	7”	9’50”	2”	372
19	148284	14”	16”	35’44”	4”	372
20	366319	37”	42”	2h11’20”	15”	879
21	910726	1’38”	1’52”	8h04’39”	22”	879
22	2278658	4’17”	4’57”	- -	1’34”	2117
23	5731580	11’11”	13’09”	- -	2’25”	2117
24	14490245	29’22”	31’39”	- -	10’18”	5174
25	36797588	1h18’43”	1h24’11”	- -	15’58”	5174
26	93839412	3h33’50”	3h49’33”	- -	1h06’44”	12813
27	240215803	9h09’35”	9h36’55”	- -	1h46’25”	12813

The main reason seems to be the fact that KP builds the alkane molecular family employing only $\lfloor \frac{n}{2} \rfloor$-rooted trees. On the contrary, the other algorithms need to generate all the alkane molecular families up to $n - 1$. In other words, KP generates 879 rooted trees to enumerate all the 366319 20-trees while AHM, HPVK and TNKMS need 251731 trees.

One may observe that contrary to the KP algorithm, AHM, HPVK and TNKMS algorithms give as output not only the set of all trees of a given cardinality n, but they also give as a byproduct all the sets of trees of cardinality m with $m < n$. However, in order to achieve the same result also utilizing algorithm KP we may run it once for each possible value of m. The overall running time would be the sum of running times of each run. It is clear from table 2 that even in this case KP proves to be much faster than the other algorithms. Moreover, an optimized algorithm computes once the rooted trees needed to enumerate all alkane families saving further computational time.

In conclusion, KP appears the best algorithm to enumerate an alkane molecular family with n carbonium atoms.

7 Conclusions

We have presented four different algorithms to enumerate the alkane molecular family C_nH_{2n+2} for any given n.

Reading the alkane enumeration literature, we have noted that remarkable work on this topic has been done by researchers not deeply involved

with algorithmic aspects. They often avoid studing or reporting on the computational complexity and the running times. For these reasons, we reported a detailed complexity study.

One-to-one enumeration requires some rules which avoid duplications. We have shown how Reverse Search and Symmetry Detection techniques are useful to avoid multiple enumeration and to save memory allowing the enumeration of larger families. Moreover, they speed up the running time of the algorithms HPVK and AHM in comparison to TNKMS. Jordan's Theorem guarantees the uniqueness in the constructive enumeration algorithm. As shown in section 6, KP is surprisingly the best algorithm.

Acknowledgments

The authors are indebted to the Editor and to the anonymous referee for their comments and suggestions which improved the presentation.

References

1. Aringhieri R., Metodi dell'Ottimizzazione Combinatoria applicati alla Chimica: L'Enumerazione ed il Calcolo del Numero di Hyper Wiener di Molecole di Alcano. *Tesi di laurea*, **1996**, Dipartimento di Informatica, Università di Pisa.
2. Aringhieri R., Hansen P. and Malucelli F., A Linear Algorithm for The Hyper Wiener Index of Chemical Trees. *J. Chem. Inf. Comput. Sci.*, **2001**, *41(4)*, 958-963.
3. Avis D.; Fukuda K., A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra. *Discrete Comput. Geometry* **1992**, *8*, 295-313.
4. Avis D.; Fukuda K., Reverse Search for enumeration. *Discrete Applied Mathematics* **1996**, *65*, 21-46.
5. Balaban T. S.; Filip P. A.; Ivanciuc O., Computer Generation of Acyclic Graphs based on Local Vertex Invariants and Topological Indices. Derived Canonical Labelling and Coding of trees and Alkanes. *J. Math. Chem.* **1992**, *11*, 79-105.
6. Berzelius J. J., *Ann. Phys. Chem.* **1830**, *19*, 305-.
7. Cayley A., On the Theory of Analytical Forms called Trees. *Philos. Mag.* **1857**, *13*, 172-176.
8. Cayley A., On the Mathematical Theory of Isomers. *Philos. Mag.* **1874**, *47*, 444-446.
9. Cayley A., On the Analytical Figures called Trees in Mathematics and Their Applications to the Theory of Chemical Combinations. *Ber. Dtsch. Chem. Ges.* **1875**, *8*, 1056-1059.
10. Corey E. J.; Wipke W. T., *Science* **1969**, *166*, 178-.
11. Corey E. J., *Quart. Rev. Chem. Soc.* **1971**, *25*, 455-.
12. Hansen P.; Jaumard B.; Labatteux C.; Zeng M., Coding Chemical Trees with the Centered N-Tuple Code. *J. Chem. Inf. Comput. Sci.*, **1994**, *34*, 782-790.

13. Hansen P.; Pitteloud G.; Van der Klink R., Enumeration d'Alcanes par le Code N-Tuple et la Recherche Inversée. *Travail de semestre* 1995, DMA, EPFL, Lausanne.
14. Jordan C., *Reine Angew.Math.* **1869**, 185-.
15. Kvasnička V.; Pospichal J., Canonical Indexing and Constructive Enumeration of Molecular Graphs. *J. Chem. Inf. Comput. Sci.*, **1991**, *56*, 1777-1802.
16. Kvasnička V.; Pospichal J., Constructive Enumeration of Acyclic Molecules. *Collect. Czech. Chem. Commun.* **1991**, *56*, 1777-1802.
17. Lederberg J.; Sutherland J. L.; Buchanan B. G.; Feigenbaum E. A.; Robertson A. V.; Duffield A. M.; Djerassi C. J., *J. Am. Chem. Soc.* **1969**, *91*, 2973-.
18. Lederberg J.; Sutherland J. L.; Buchanan B. G.; Feigenbaum E. A.; Lindsay R. K. , Application of Artificial Intelligence for Organic Chemistry: the DENDRAL project. McGraw-Hill: New York, **1980**.
19. Randić, M.; Guo, X.; Oxley, T.; Krishnapriyan, H.; Naylor, L. Wiener Matrix Invariants. *J. Chem. Inf. Comput. Sci.*, **1994**, *34*, 261-367.
20. Read R. C., The Coding of Various Kind of Unlabelled Trees in *Graph Theory and Computing*; Ed Read R. C., Academic Press: New York, **1972**, Chapter 12, 153-.
21. 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*, 135-182.
22. Trinajstić N.; Nicolić S.; Knop J. V.; Müller W. R.; Szymanski K., Computational Chemical Graph Theory. Ellis Horwood: New York. **1991**.