Enumerating Treelike Chemical Graphs with Given Path Frequency

Hiroki Fujiwara,[†] Jiexun Wang,*[,†] Liang Zhao,*[,†] Hiroshi Nagamochi,*[,†] and Tatsuya Akutsu*[,‡]

Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan, and Bioinformatics Center, Institute for Chemical Research, Kyoto University, Uji, Kyoto 611-0011, Japan

Received October 24, 2007

The enumeration of chemical graphs satisfying given constraints is one of the fundamental problems in chemoinformatics. In this paper, we consider the problem of enumerating (i.e., listing) all treelike chemical graphs from a given path frequency. We propose an exact algorithm for enumerating all solutions to this problem on the basis of the branch-and-bound method. To further improve the efficiency of the enumeration, we introduce a new variant of the compound enumeration problem by adding a specification on the number of multiple bonds to the input and design another exact enumeration algorithm. The experimental results show that our algorithms can efficiently solve instances with larger sizes that are impossible to solve by the previous methods. In particular, we apply the latter algorithm to the enumeration problem of the special treelike chemical structures—alkane isomers. The theoretical and experimental results show that our algorithm works at least as fast as the state-of-the-art algorithms specially designed for generating alkane isomers, however using much less memory space.

1. INTRODUCTION

The enumeration of chemical graphs is one of the fundamental problems in chemoinformatics and has a long history. In the 19th century, Cayley initiated the study on the enumeration of structural isomers of alkanes.[1,2] Since then, the enumeration problem has attracted many mathematicians and chemists.[3–11] In particular, Pólya proposed important theorems on counting the number of isomers.[8,9] Subsequently, computer-oriented methods have been extensively studied.[12–16]

One of the important applications of the enumeration of chemical compounds is structure determination using mass-spectra or NMR spectra. For this purpose, the enumeration of chemical graphs satisfying given constraints is important, and extensive studies have been done.[17,18] Enumeration with constraints has been studied for other purposes including the virtual exploration of the chemical universe[19,20] and the reconstruction of molecular structures from their signatures.[21,22] Efficient enumeration of graphs plays a key role in most of these works. However, many of them are based on simple or complicated heuristics with no guarantee on the computational complexity, and they often require much overhead to avoid the generation of identical structures.

On the other hand, studies on the classification of compounds based on *support vector machines* (SVMs) and other *kernel-based methods* have recently been investigated extensively.[23–26] The general idea behind them is that all chemical compounds are mapped to be *feature vectors* (i.e., vectors with real entries) in a feature space, and then SVMs[27] are employed to design the classification rules. The definition of the feature vectors based on *frequencies of labeled paths*[25,26] or *frequencies of small fragments*[23,24] is more widely used than the others, where chemical properties such as molecular weights and partial charges are sometimes associated and weights/probabilities are sometimes put on paths/fragments of compounds.

In the study of the kernel-based methods, a new problem called the *preimage problem* has been proposed for optimizing objects in an input space by using kernel methods.[28,29] In this problem, a desired object is computed as a point in the feature space, and then the point is mapped back to the input space, where such an object mapped back from a point is called a *preimage* of the point. More formally, let ψ be a mapping from an input space to a feature space, where ψ is not necessarily injective or surjective. Then, the preimage problem is, given an arbitrary point y in the feature space, how to find a preimage x in the input space such that $y = \psi(x)$. Note that the exact preimage of the given point may not exist. In this case, it is natural to find an approximate preimage x^* defined as $x^* = \arg\min_x \mathrm{dist}(y,\psi(x))$, where $\mathrm{dist}(y,z)$ is an appropriately defined distance measure. See Figure 1 for an illustration.

To our knowledge, several studies have been done on the preimage problem.[28,29] However, all of them were based on heuristic or stochastic methods. It is rather recent that theoretical issues on exact algorithms have been studied. Akutsu and Fukagawa[30] formulated the graph preimage problem as the problem of inferring graphs from the numbers of occurrences of vertex-labeled paths with a length of at most K, and they proved that this problem can be solved in polynomial time for the sizes of outputs if the graphs are trees of bounded degree and the lengths of given paths are bounded by a constant, whereas it is NP-hard even for planar graphs with bounded degrees. They also designed a dynamic programming algorithm for inferring the graphs in a restricted

* Corresponding authors. E-mail: wangjx@amp.i.kyoto-u.ac.jp (J.W.), liang@amp.i.kyoto-u.ac.jp (L.Z.), nag@amp.i.kyoto-u.ac.jp (H.N.), takutsu@kuicr.kyoto-u.ac.jp (T.A.).

[†] Graduate School of Informatics.

[‡] Institute for Chemical Research.

10.1021/ci700385a CCC: $40.75 © 2008 American Chemical Society
Published on Web 06/28/2008

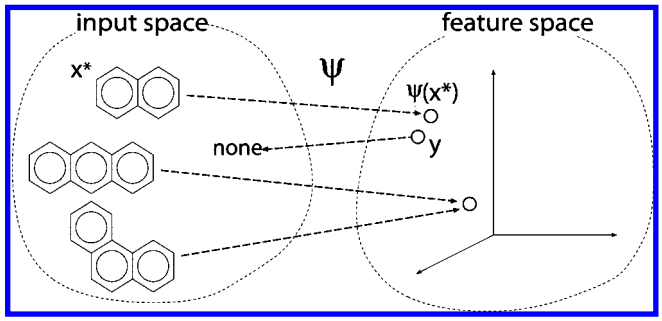

Figure 1. The preimage problem for chemical compounds. Different compounds may be mapped to the same point in the feature space. For a point y in the feature space, we may find an approximate preimage x^* such that dist$(y,\psi(x))$ is minimal if $\psi^{-1}(y)$ does not exist.

class of outerplanar graphs.[31] However, the time complexity of the algorithm is exponential in K and the number of labels. Nagamochi[32] proved that the graph inference problem from the frequencies of paths with lengths of at most $K = 1$ can be efficiently solved by formulating the problem of finding a connected detachment.

In this study, we consider the problem of enumerating (i.e., listing) all treelike chemical graphs from a given feature vector based on frequencies of paths (called *Problem 1*). (In the remainder of the paper, we let "to enumerate" only mean "to list" but not "to count" to avoid confusion.) It is to be noted that the preimage problem for chemical compounds can be regarded as a problem of inferring or enumerating compounds that satisfy the constraint $y = \psi(x)$. As a preceding work, Akutsu and Fukagawa developed two algorithms for this problem.[31] Their computational results showed that their algorithms can output solutions of moderate size (i.e., at most 46 atoms) after treating the benzene as a new atom with six valences.

To deal with compounds of larger size, we propose a new algorithm (called *Algorithm A*) based on the branch-and-bound approach, which systematically enumerates candidate solutions, uses bounds to limit the search space,[33] and has been applied to various kinds of problems in chemoinformatics and bioinformatics.[31,34,35] Specifically, each compound without ring structures is uniquely represented as a labeled multitree satisfying the valence constraint. On the basis of the well-established theory for simple trees, we first develop a coding scheme and canonical representations for multitrees. Then, for Problem 1, we find all rooted multitrees satisfying the given constraints, where the root of each multitree is either a unique vertex or a unique pair of vertices. Algorithm A deals with Problem 1 on the basis of a procedure which starts from an empty graph and recursively generates new rooted multitrees by attaching a new leaf to the current multitree (the branching operation) unless the newly generated multitree violates the given constraints (the bounding operations). The experimental results show that Algorithm A can also deal with compounds with 46 atoms, which have been solved by the algorithms proposed by Akutsu and Fukagawa.[31] But Algorithm A works more efficiently than their algorithms because it requires less information about the compounds.

For further improvement in the efficiency of the enumeration, we formulate a new variant of the compound inference problem (called *Problem 2*) by removing all hydrogen atoms from the compounds (called *H-removal transformation*) and by replacing the multiple edges with a new atom and two new single edges (called *single-bond transformation*). We modify Algorithm A to a new *Algorithm B* for this variant, whose scheme is similar to that of Algorithm A. Our experimental results show that, compared to Algorithm A, Algorithm B runs much faster and can treat some instances of 61 atoms. This is a significant step toward improvement of the efficiency of practical algorithms for the graph preimage problem.

In order to show the performance of our algorithms as enumeration algorithms, we focus on the problem of inferring alkanes from a given path frequency. When the length of the given paths is not larger than one, the inference problem of alkanes is converted to the problem of enumerating all structural isomers for alkanes with a given number of vertices. We compare Algorithm B with the algorithms specialized for the enumeration problem of alkane isomers.[36,37] Both the theoretical and experimental results show that Algorithm B works at least as fast as the state-of-the-art algorithms, although it uses much less memory space.

The rest of this paper is organized as follows. Section 2 gives some preliminaries and formulates the treelike chemical graph enumeration problem (i.e., Problem 1). Section 3 presents a canonical representation for labeled trees. Section 4 designs a branch-and-bound algorithm (i.e., Algorithm A) for Problem 1. Section 5 describes a new formulation of the treelike chemical graph enumeration problem (i.e., Problem 2) by introducing two transformations and proposes Algorithm B, which is based on Algorithm A. Section 6 reports the experimental results of our algorithms, and finally section 7 makes some concluding remarks.

2. PRELIMINARIES AND PROBLEM FORMULATION

A graph is called a *multigraph* if multiple edges (i.e., edges with the same end vertices) are allowed; otherwise, it is called *simple*. In particular, a *multitree* or *treelike* graph is a multigraph with no cycle or self-loop. A *path* P in a graph G is a finite non-null sequence $(v_0, e_1, v_1, e_2, v_2,..., e_s, v_s)$, where the vertices v_i are distinct for $i = 0, 1,..., s$ and the edges e_j are incident to v_{j-1} and v_j for $j = 1, 2,..., s$. Such a path P is also denoted as a sequence of vertices $P = (v_0, v_1,..., v_s)$ if no confusion arises. Let Σ be a set of labels. Define Σ^k as the set of all sequences with k labels in Σ, and $\Sigma^{\leq k} = \cup_{j=1}^{k} \Sigma^j$. Let $F_k(\Sigma) = \{g: \Sigma^{\leq k+1} \rightarrow \mathbf{Z}^+\}$, where $\mathbf{Z}^+$ denotes the set of nonnegative integers.

A multigraph G is called Σ-*labeled* if each vertex v has a label, denoted by $\boldsymbol{l}(v)$, belonging to Σ; moreover, G is called (Σ,val)-labeled if the label of each vertex v has some valence, denoted by $\mathrm{val}(\boldsymbol{l}(v))$, belonging to the set $\mathbf{N}$ of natural numbers. Note that each chemical compound can be viewed as a (Σ,val)-labeled, self-loopless, and connected multigraph, where the label of each vertex represents an atom, the multiple edges represent the multiple chemical bonds between two adjacent atoms, and the degree of (i.e., the number of edges incident to) each vertex equals the valence of the corresponding atom. For a path $P = (v_0, v_1,..., v_s)$, its label sequence is defined as $\boldsymbol{l}(P) = (\boldsymbol{l}(v_0), \boldsymbol{l}(v_1),..., \boldsymbol{l}(v_s))$. For a label sequence t, let occ(t,G) denote the number of paths in G with the label sequence t, where we treat multiple edges with the same end vertices as a single edge when counting

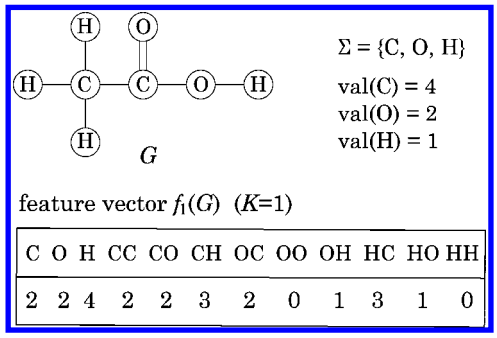

Figure 2. An illustration of a (Σ,val)-labeled multitree G and its feature vector $f_1(G)$.

the number of paths in G. The *feature vector* of G with *level* K is defined by $f_K(G) = (\mathrm{occ}(t,G))_{t \in \Sigma^{\le K+1}}$. Figure 2 illustrates a (Σ,val)-labeled multitree G (representing a chemical compound) and its feature vector with level $K = 1$, where we treat all paths in G as "directed", and hence, we have $\mathrm{occ}(\mathrm{OH},G) = \mathrm{occ}(\mathrm{HO},G) = 1$, and so forth.

In this study, we focus on the problem of enumerating treelike chemical compounds from a given feature vector and formulate it as follows (see Figure 3 for an illustration).

Problem 1. *Given a finite label set* Σ*, an integer* $K \ge 1$*, a feature vector* $g \in F_K(\Sigma)$*, and a valence function val:* $\Sigma \to \mathbf{N}$*, find all (*Σ*,val)-labeled multitrees* T *satisfying* $f_K(T) = g$ *(the feature vector constraint) and* $\deg_T(v) = val(\boldsymbol{l}(v))$ *(the valence constraint) for each vertex* v *in* T*, where* $\deg_T(v)$ *denotes the number of edges incident to the vertex* v*.*

Note that the number of vertices of all solutions for Problem 1 is $n = \sum_{l \in \Sigma} g(\boldsymbol{l})$. Besides, we see that each (Σ,val)-labeled multitree can be obtained from the corresponding (Σ,val)-labeled *simple* tree, since the valence constraint can uniquely determine the multiplicities of edges from leaves (note that the feature vectors of each multitree and its corresponding simple tree are the same). Then, we design a branch-and-bound algorithm satisfying the following:

i. It can enumerate all nonisomorphic (Σ,val)-labeled simple trees with at most n vertices by using a branching operation.

ii. For each (Σ,val)-labeled simple tree with at most n vertices, it can obtain the corresponding multitree by calculating the multiplicities of edges and can decide whether to discard this multitree by using some bounding operations.

The key issue for i is to avoid enumerating isomorphic (Σ,val)-labeled simple trees. To the best of our knowledge, several algorithms have been proposed for the problem of generating all nonisomorphic *unlabeled* simple trees with a given number of vertices.[38–42] Most of these algorithms choose a *unique* vertex or a *unique* pair of adjacent vertices as the *root* for each tree with exact n vertices on the basis of some rule, and then they generate the resulting class of unlabeled *rooted* trees.[38,39,42] We will follow the idea to generate all isomorphic (Σ,val)-labeled simple trees while checking ii for each generated tree.

In the following, section 3 introduces some important definitions for the generation of (Σ,val)-labeled simple trees, and section 4 presents the design of Algorithm A for Problem 1, satisfying the above requirements (i and ii).

3. CANONICAL REPRESENTATION OF LABELED TREES

First, we restrict to the class of (Σ,val)-labeled *rooted* simple trees, where each tree is arbitrarily employed a vertex or a pair of adjacent vertices as its root. We define a sequence to represent a (Σ,val)-labeled rooted tree, and we choose a tree with some property as the canonical representation for all isomorphic (Σ,val)-labeled rooted simple trees. Then, we formally introduce a more special class of (Σ,val)-labeled rooted trees, where each tree has a unique vertex or a unique pair of adjacent vertices as the root on the basis of some rule. We define a relation between two (Σ,val)-labeled rooted trees, where the roots may not be unique. In this section, we simply call (Σ,val)-labeled simple trees "trees", and we call (Σ,val)-labeled rooted simple trees "rooted trees" if no confusion arises.

3.1. Depth Label Sequence. This subsection introduces a mathematical representation for a rooted tree.

We assume that each rooted tree is embedded in the plane, where the children of each nonleaf vertex are ordered from left to right. Let T be a tree with n vertices rooted at a vertex or a pair of adjacent vertices, where the label of each vertex belongs to the label set Σ. Without a loss of generality, we assume that $v_0, v_1, \ldots, v_{n-1}$ are the vertices of T indexed by the depth-first search (DFS) order. That is, we start from the unique root or the left vertex of a pair of roots and name it v_0, and then we traverse other vertices by DFS on the assumption that the children of a vertex are traversed from left to right, and we name them $v_1, \ldots, v_{n-1}$ in turn. For each nonroot vertex v, let $P(v)$ denote the unique path to v starting from the unique root or one vertex of two roots that is closer to v. We define the parent of v to be the vertex adjacent to v on $P(v)$, and we define the ancestors of v to be all vertices except v on $P(v)$. Thus, we say that v is a child of w if w is the parent of v, and we say that v is a descendant of w if w is an ancestor of v. Besides, we let the *depth*, $d(v)$, of a vertex v be the number of edges on $P(v)$, where we set $d(v) = 0$ when v is the root. Then, we define the *depth label sequence* of T to be

$$\mathrm{DL}(T) = [d(v_0), \boldsymbol{l}(v_0), \ldots, d(v_{n-1}), \boldsymbol{l}(v_{n-1})] \qquad (1)$$

where $d(v_i)$ and $\boldsymbol{l}(v_i)$ are the depth and the label of v_i for all $0 \le i \le n-1$, respectively (note that it is always true for $d(v_0) = 0$). See Figure 4 for illustrations. The rightmost path $\mathrm{RP}(T)$ of T is the path $(r_0, r_1, \ldots, r_k)$ such that r_0 is the vertex root or the right vertex of the edge root, r_i is the rightmost child of r_{i-1} for all $1 \le i \le k$, and r_k is the rightmost leaf of T.

Let T_1 and T_2 be two rooted trees with n_1 and n_2 vertices, respectively, and $\mathrm{DL}(T_1) = (x_0, a_0, x_1, a_1, \ldots, x_{n_1-1}, a_{n_1-1})$ and $\mathrm{DL}(T_2) = (y_0, b_0, y_1, b_1, \ldots, y_{n_2-1}, b_{n_2-1})$ be the depth label sequences. Given an arbitrary order for the elements in Σ, we say that $\mathrm{DL}(T_1)$ is *lexicographically larger* than $\mathrm{DL}(T_2)$, denoted by $\mathrm{DL}(T_1) > \mathrm{DL}(T_2)$, if there exists some i ($\in [1, \min\{n_1-1, n_2-1\}]$) such that $x_j = y_j$ and $a_j = b_j$ for all $j = 0, 1, \ldots, i-1$, and either (1) $x_i > y_i$ (possibly y_i may not exist) or (2) $x_i = y_i$ and $a_i > b_i$. In particular, we say that $\mathrm{DL}(T_2)$ is a prefix of but not equal to $\mathrm{DL}(T_1)$, denoted by $\mathrm{DL}(T_1) \supset \mathrm{DL}(T_2)$, if $x_j = y_j$ and $a_j = b_j$ for all $j = 0, 1, \ldots, n_2 - 1 < n_1 - 1$. In addition, we let $\mathrm{DL}(T_1) \ge \mathrm{DL}(T_2)$

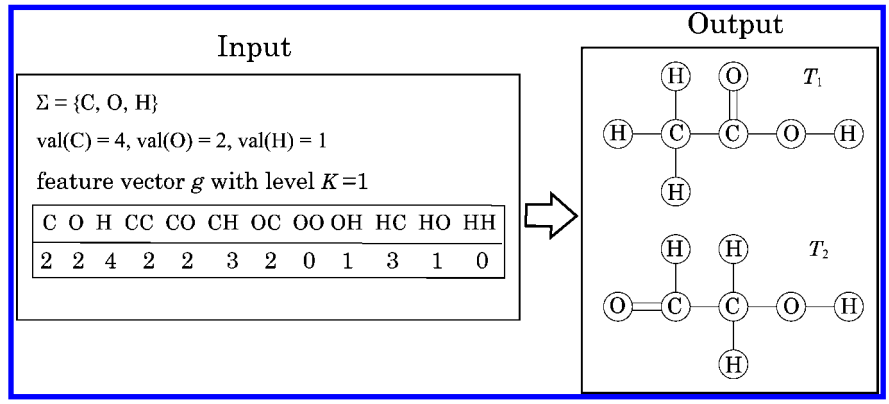

Figure 3. An instance of problem 1 with the feature vector in Figure 2, which admits two different solutions.

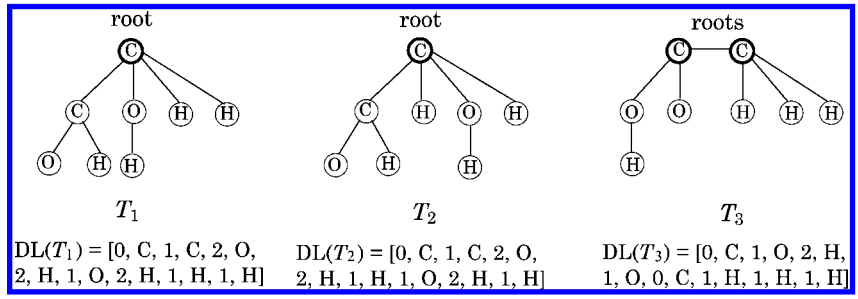

Figure 4. Rooted trees and their depth label sequences ($\Sigma = \{C, O, H\}$). Trees T_1 and T_2 are isomorphic. Assuming C > O > H, we have $DL(T_1) > DL(T_2) > DL(T_3)$. Both T_1 and T_3 are left-heavy (defined in section 3.2), whereas T_2 is not. Both T_1 and T_2 are rooted at unicentroids, whereas T_3 is rooted at a bicentroid (defined in section 3.3).

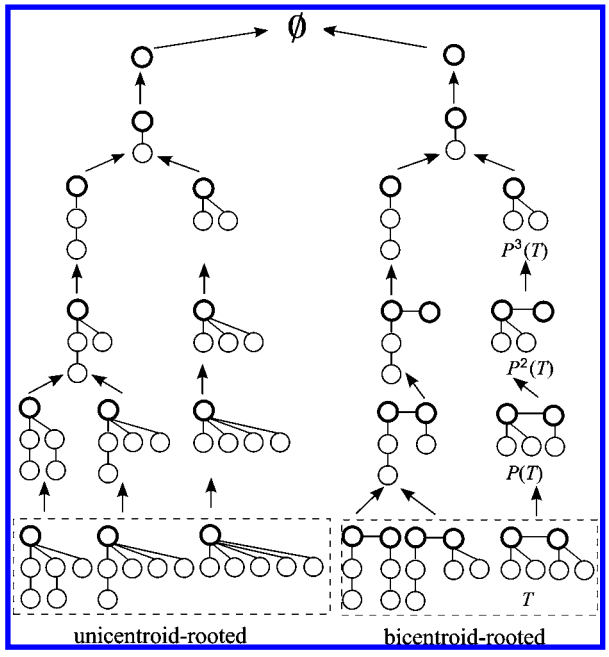

Figure 5. An illustration of the family tree $F(6,1)$. All leaf nodes in $F(6,1)$ are rooted at centroids shown by the thick circuits. Tree T is rooted at a bicentroid, whereas $P(T)$ is not. The ancestor $P^2(T)$ is rooted at a pair of adjacent vertices, whereas $P^3(T)$ is rooted at a vertex.

mean $DL(T_1) > DL(T_2)$ or $DL(T_1) = DL(T_2)$, and we let $DL(T_1) \geq DL(T_2)$ mean $DL(T_1) \supset DL(T_2)$ or $DL(T_1) = DL(T_2)$.

3.2. Left-Heavy Embedding. Recall that an abstract rooted tree may have different plane embeddings (or drawings). For example, trees T_1 and T_2 are isomorphic since they are embeddings for the same tree. We assign the embedding with the largest label sequence as the canonical representation of each rooted tree.

Let T denote an embedding for a rooted tree, and let $\mathbf{T}(v)$ denote the subtree of T consisting of a vertex v and all its descendants. We say that T is *left-heavy* if $i < j$ implies $DL(\mathbf{T}(v_i)) \geq DL(\mathbf{T}(v_j))$ for any two siblings v_i and v_j. By definition, we find the following property for these embeddings.

Lemma 1. *For a tree rooted at a vertex, its embedding with the largest depth-label sequence is left-heavy. On the other hand, for a tree rooted at a pair of adjacent vertices* u and v *from left to right, its embedding with the largest depth-label sequence satisfies that both of the subtrees* $\mathbf{T}(u)$ *and* $\mathbf{T}(v)$ *are left-heavy and* $DL(\mathbf{T}(u)) \geq DL(\mathbf{T}(v))$.

3.3. Family Tree. This subsection formally introduces the choice of a unique vertex or a unique pair of adjacent vertices as the root for each tree, and it defines a relationship between two left-heavy rooted trees, where the roots may not be unique.

The following theorem provides the theoretical rule of the existence of a unique vertex or a unique pair of adjacent vertices for each tree.

Theorem 1 (Jordan's Theorem).[43] *For any simple tree with* n' *vertices, either there exists a unique vertex* v^**, such that any subtree obtained by removing* v^* *contains at most* $\lfloor (n'-1)/2 \rfloor$ *vertices, or there exists a unique pair of adjacent vertices* v_1^* *and* v_2^* *joined by an edge, such that both subtrees obtained by removing the edge incident to* v_1^* *and* v_2^* *contain exactly* $n'/2$ *vertices.*

Such a vertex v^* and a pair of adjacent vertices v_1^* and v_2^* in Theorem 1 are called *unicentroid* and *bicentroid*, respec-

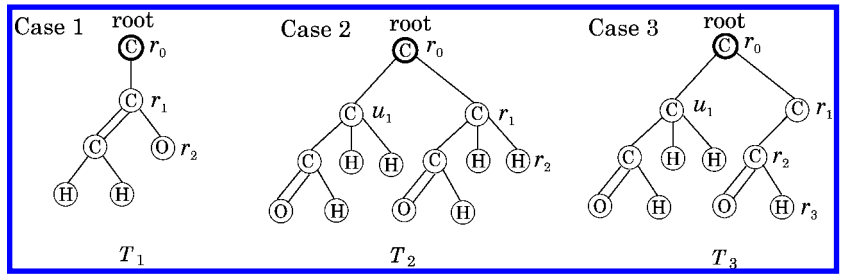

Figure 6. Illustrations of rooted multitrees in different three cases. (1) Tree T_1 belongs to case 1, and its copy depth is $\mathrm{CD}(T_1) = 2$. Thus, we can generate new rooted multitrees by attaching a new leaf p to any vertex of $\{r_0, r_1, r_2\}$ of T_1. (2) T_2 belongs to case 2 and has $\mathrm{CD}(T_2) = 0$. Thus, we can generate new multitrees by attaching p to the vertex r_0 of T_2. (3) T_3 belongs to case 3 and has $\mathrm{CD}(T_3) = 0$. Thus, we can generate new multitrees by attaching p to r_0 or r_1 of T_3.

tively. Both a unicentroid and a bicentroid are classified as a *centroid*. Note that there does not exist a bicentroid for an odd n'. We say that a rooted tree is a *unicentroid-rooted* (respectively, *bicentroid-rooted*) tree if its root is unicentroid (respectively, bicentroid). See Figure 4 for illustrations. Trees T_1 and T_2 are unicentroid-rooted trees, and T_3 is a bicentroid-rooted tree. Generally, we say that a rooted tree is *centroid-rooted* tree if the tree is rooted at a unicentroid or bicentroid. Note that a simple tree uniquely corresponds to a left-heavy centroid-rooted tree.

Now we define a parent−child relation between two left-heavy rooted trees, whose roots may not be centroids. Specifically, given a left-heavy rooted tree T, we define its parent $P(T)$ to be the tree by removing the rightmost leaf from T. By definition, we easily find the following property of the parents.

Lemma 2. *For a given left-heavy rooted tree T, the parent P(T) remains left-heavy.*

Correspondingly, T is called a *child* of $P(T)$. In a similar way, we can define other ancestors of T: $P(P(T)) = P^2(T)$, $P(P(P(T))) = P^3(T)$, and so on. Generally, all ancestors of a left-heavy centroid-rooted tree are still left-heavy but may not be rooted at their centroids. In particular, for a bicentroid-rooted tree with exact n (even) vertices, its ancestor $P^i(T)$ for all $n/2 \le i \le n - 1$ is rooted at only one vertex, which may not be unicentroid. See trees T and its ancestors $P(T)$, $P^2(T)$, and $P^3(T)$ in Figure 5 for illustrations.

Further, we define a special tree structure—*family tree, F(n,m)*—for all centroid-rooted trees with exact n vertices and at most m ($= |\Sigma|$) labels and all their ancestors. In such a tree structure, each node represents a left-heavy rooted tree; in particular, all leaf nodes represent centroid-rooted trees, and all nonleaf nodes represent ancestors of centroid-rooted trees which may not be rooted at their centroids. Figure 5 gives an illustration of the family tree $F(6,1)$ of centroid-rooted trees.

4. ALGORITHM A FOR PROBLEM 1

We have introduced some definitions for simple trees in section 3. Analogously, for the multitree case, we can define a *rooted* multitree to be the one whose root is arbitrarily assigned as a vertex or a pair of adjacent vertices. Note that most of the definitions proposed for simple trees are unrelated to the number of edges, including "depth-label sequence," "rightmost path," and "left-heavy." Because of this, we can easily have the multitree version by replacing "tree(s)" with "multitree(s)" in these definitions. Moreover, we can easily extend Theorem 1 to define "unicentroid," "bicentroid," and "centroid" for multitrees and, hence, define "unicentroid-rooted multitrees," "bicentroid-rooted multitrees," and "centroid-rooted multitrees," respectively.

Thus, for finding all solutions, with exact n vertices, of Problem 1 without repetition, we need to output all nonisomorphic left-heavy centroid-rooted multitrees satisfying the feature vector and the valence constraints. For this, we propose a branch-and-bound algorithm (called *Algorithm A*), which, starting from an empty graph, generates partial left-heavy rooted multitrees consisting of only a vertex. It then recursively generates new left-heavy rooted multitrees by attaching a new leaf to a vertex on the rightmost path of the current multitree with less than n vertices (called the *branching* operation), unless the newly generated multitrees T_c with no larger than n vertices violate the feature vector constraint, that is, $f_K(T_c) \le g$ does not hold, or violate the valence constraint, that is, $\deg_{T_c}(v) \le \mathrm{val}(\boldsymbol{l}(v))$ does not hold for some vertex $v \in T_c$, or unless the derived multitrees with exact n vertices are not rooted at centroids (called the *bounding* operations). In this section, we focus on the class of left-heavy rooted (Σ,val)-labeled multitrees, where the roots may not be the centroids, and simply call them rooted multitrees if no confusion arises.

4.1. Branching Operation. Recall that, for a left-heavy Σ-labeled rooted simple tree, it is easy to find its parent. That is, we just remove the rightmost leaf from this tree. However, it is not obvious to see how to gain all of children by reversing this procedure, that is, how to attach a new leaf to the rightmost path of this tree without violating the left-heavy property.[44] Analogously, we have to consider the efficient generation of all possible new rooted multitrees from a partial rooted multitree without repetition. This subsection will explain how to complete this task.

For ease of explanation, first, we introduce some additional notations and definitions. Let T be a rooted multitree with $n'(< n)$ vertices, and let $\mathrm{RP}(T) = (r_0, r_1, ..., r_k)$ be the rightmost path, where the root is either r_0 or two adjacent vertices u_0 and r_0 from left to right joined by at least one edge. Note that the root may not be the centroid. For each vertex r_i ($1 \le i < k$) with at least one sibling, we always denote u_i as its immediate left sibling. In addition, we say that T is *active at depth* $(i - 1)$ if r_{i-1} has at least two children (the rightmost one is r_i) and $\mathrm{DL}(\mathbf{T}(u_i)) \supseteq \mathrm{DL}(\mathbf{T}(r_i))$ holds. In particular, if T is rooted at the adjacent vertices u_0 and r_0 such that $\mathrm{DL}(\mathbf{T}(u_0)) \supseteq \mathrm{DL}(\mathbf{T}(r_0))$, then we say that T is active at depth -1. We define the *copy depth* of T to be

the minimum depth where T is active and denote it CD(T). If T is not active at any depth, then we let CD(T) = k.

By analogy with the results of Nakano and Uno,[44] we classify all left-heavy rooted multitrees into three cases based on the copy depth (see Figure 6 for illustrations).

Lemma 3. *Let T be a rooted multitree and RP(T) = (r_0, r_1,..., r_k) be its rightmost path. Then, T belongs to one of the following three cases:*

Case 1: CD(T) = k;

Case 2: CD(T) = d($\in$[−1, k)) *and* DL($\mathbf{T}(u_{d+1})$) = DL($\mathbf{T}(r_{d+1})$);

Case 3: CD(T) = d($\in$[−1, k)) *and* DL($\mathbf{T}(u_{d+1})$) $\supset$ DL($\mathbf{T}(r_{d+1})$).

On the basis of lemma 3, we characterize all possible, new, left-heavy multitrees generated from a given multitree as follows.

Lemma 4. *Let T be a left-heavy rooted multitree with n'(< n) vertices and RP(T) = (r_0, r_1,..., r_k) be its rightmost path. Then, all new left-heavy multitrees T_c can be obtained by attaching a new leaf p with label $\boldsymbol{l}(p)$ to a vertex r_i on RP(T) such that*

(1) $0 \le d(r_i) < k$ and $\boldsymbol{l}(p) \le \boldsymbol{l}(r_{i+1})$, or, $d(r_i) = k$ and $\boldsymbol{l}(p) \in \Sigma$ when T belongs to Case 1;

(2) $0 \le d(r_i) \le \max\{0, d\}$ and $\boldsymbol{l}(\mathrm{p}) \le \boldsymbol{l}(r_{i+1})$ when T belongs to Case 2;

(3) otherwise, $0 \le d(r_i) < d(v_{n'-L}) - 1$ and $\boldsymbol{l}(\mathrm{p}) \le \boldsymbol{l}(r_{i+1})$, or, $d(r_i) = d(v_{n'-L}) - 1$ and $\boldsymbol{l}(p) \le \boldsymbol{l}(v_{n'-L})$, where L is the number of vertices of the subtree $T(u_{d+1})$.

Figure 6 provides illustrations. We remark that each T_c in the above lemma is generated by making some (constant) modifications on the given rooted multitree T without using the information of other multitrees.

4.2. Bounding Operations. Let T be the current rooted multitree with n'(< n) vertices, RP(T) = (r_0, r_1,..., r_k) be its rightmost path, and T_c be a new rooted multitree obtained by attaching a new leaf p to a vertex r_i ($0 \le i \le k$) on RP(T).

For each newly generated multitree T_c, we first update its feature vector and calculate the multiplicities of edges on it, and then we determine whether to discard it or continue executing a new branching operation on it on the basis of the following three constraints:

(C1) Any subtree obtained by removing the root or all edges incident to a pair of roots has at most S, where $S = \lfloor (n-1)/2 \rfloor$ if T_c is rooted at a vertex or $S = n/2$ if the given n is even and T_c is rooted at two adjacent vertices;

(C2) $f_K(T_c) \le g$ (the feature vector constraint);

(C3) $\deg_{T_c}(v) \le \mathrm{val}(\boldsymbol{l}(v))$ for all $v \in T_c$ (the valence constraint).

Clearly, if $T_c$violates any constraint mentioned above, then we discard it and stop the further generation of new multitrees from it, since any newly derived multitree from T_c must violate some of these constraints; otherwise, we conduct the branching operation on T_c for the next enumeration. In the following two subsections, since the bounding operation related to C1 is very simple, we will only discuss the other two bounding operations resulting from C2 and C3, called *feature-cut* and *bond-cut*, respectively, and then we will explain how to efficiently update the feature vector of T_c and the multiplicities of edges of T_c from T as well.

4.2.1. Feature-Cut Procedure and Update of Feature Vector. For each rooted multitree, we store its feature vector by using a data structure—*trie* (or *prefix tree*), which is totally different from the data structure of the family tree in section 3.3. Trie is a special node-rooted labeled tree structure with the following two properties: (1) each nonroot node stores a label and an integer, and (2) no sibling nodes share the same label.

More specifically, let $f_K(T)$ denote the feature vector of a given rooted multitree T, Trie($f_K(T)$) denote the trie for $f_K(T)$, and $P = (v_{i_1}, v_{i_2},..., v_{i_t})$ denote a path in T, where $1 \le t \le K + 1$ and $0 \le i_1 \le i_2 \le ... \le i_t \le n - 1$. For any path P in T, there exists a corresponding path $P^* = (\tau_0, \tau_1,..., \tau_t)$ in Trie($f_K(T)$) such that, for all $1 \le j \le t$, the label of τ_j is equal to the label of v_{i_j}, that is, $\boldsymbol{l}(\tau_j) = \boldsymbol{l}(v_{i_j})$, and an integer $w(\tau_t)$ stored on τ_t is equal to the frequency of P in T, that is,

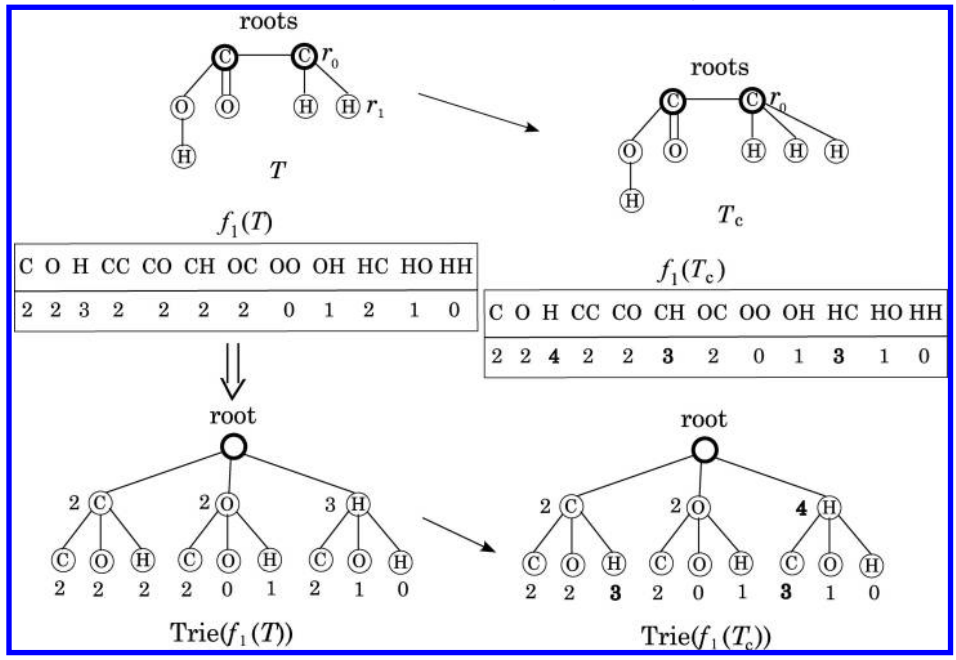

Figure 7. Illustrations of updating Trie($f_1(T_c)$) from Trie($f_1(T)$). The differences between $f_1(T)$ and $f_1(T_c)$ and between Trie($f_1(T)$) and Trie($f_1(T_c)$) are shown by the bold numbers, respectively, where T_c is obtained by attaching a new leaf p with the label H to r_0 in T, and Σ = {C, O, H} with C > O > H.

$w(\tau_t) = \mathrm{occ}(P,T)$, where τ_0 is the root of Trie($f_K(T)$) and has no label. Note that the number of nonroot nodes in Trie($f_K(T)$) is the same as that of all paths in $f_K(T)$. Figure 7 gives an illustration for trie storing a feature vector of a rooted multitree.

Now, we consider how to update Trie($f_K(T_c)$) for a newly generated rooted multitree T_c from the Trie($f_K(T)$) of the given multitree rooted T. Observe that the differences between $f_K(T)$ and $f_K(T_c)$ result from paths with lengths not larger than K, starting from the new attached leaf p in T_c, which are illustrated by the bold numbers in $f_K(T_c)$ in Figure 7. Thus, we obtain Trie($f_K(T_c)$) by modifying Trie($f_K(T)$) on the basis of these differences using the DFS, starting from p with a depth of no more than K. The differences between Trie($f_K(T_c)$) and Trie($f_K(T)$) are shown by the bold numbers in Figure 7. Then, we conduct the *feature cut* on the basis of the feature vector constraint, and we decide whether to discard T_c or to continue applying another bounding operation to T_c, which will be introduced in the next subsection.

4.2.2. Bond-Cut Procedure and Calculation of Edge Multiplicities. Let T be the current rooted multitree, $\mathrm{RP}_T = (r_0, r_1,..., r_k)$ be the rightmost path of T, T_c be the new rooted multitree obtained by attaching a new leaf p to a vertex r_i $(0 \le i \le k)$, and RP(T_c) be the rightmost path of T_c.

We claim that the multiplicities of the edges of T that are not on RP(T) have been determined before T_c is generated (we will explain the claim at the end of this subsection). Then, we easily observe that the differences between T and T_c are their rightmost paths resulting from the new attached leaf. That means that the edges that are not on RP(T_c) have the same multiplicities of the edges that are not on RP(T). Thus, after generating T_c, we only need to determine the multiplicities of a set (E) of edges that are on RP(T) but not on RP(T_c). Note that the edge set E may be the null set. Specifically, the edge set E is the null set if p is attached to the vertex r_k, or it is $\{(r_j, r_{j-1}), j = k - 1, k - 2,..., i + 1\}$ if p is attached to a vertex r_i $(0 \le i \le k - 1)$. Then, we discard T_c if $\deg_{T_c}(v) > \mathrm{val}(\boldsymbol{l}(v))$ holds for some $v \in \mathrm{RP}(T)$. We call the bounding operation related to this valence constraint the *bond cut*. In the following, we only discuss the execution of bond cut on T_c in the scenario in which T belongs to Case 1, since the other two cases are similar.

Denote the multiplicity of an edge (r_j, r_{j-1}) in T_c as $\mathrm{Mul}(r_j, r_{j-1}|T_c)$. The rightmost path RP($T_c$) of T_c is updated by attaching p to the end of RP(T) when a new leaf p is attached to r_k, that is, $\mathrm{RP}(T_c) = (r_0, r_1,..., r_k, p)$. In this case, the multiplicities of all edges on T_c are exactly the same as that on T except for the new edge (p, r_k) with $\mathrm{Mul}(p, r_k|T_c) = 1$. We discard T_c if $\deg_{T_c}(r_k) > \mathrm{val}(\boldsymbol{l}(r_k))$. On the other hand, assume that p is attached to a vertex r_i $(0 \le i < k)$. Then, the rightmost path RP(T_c) is updated by replacing the edge (r_i, p) with the path $(r_i, r_{i+1},..., r_k)$ when p is attached to a vertex r_i $(0 \le i < k)$, that is, $\mathrm{RP}(T_c) = (r_0, r_1,..., r_i, p)$. Then, we can determine the multiplicities of the edges $\{(r_j, r_{j-1}), j = k, k - 1,..., i + 1\}$ on the replaced path $(r_i, r_{i+1},..., r_k)$ and decide whether to discard T_c by using the following steps:

```
1  if val(l(r_k)) < val(l(r_{k-1})) then
2     Discard T_c and stop
3  else
4     Mul(r_k, r_{k-1}|T_c) := val(l(r_k));
      /* Calculate the multiplicities of other edges (r_j, r_{j-1}) for
         k - 1 <= j <= i + 1 in T_c, and check the valence constraint for r_j */
5     j := k - 1
6     while (j <= i + 1)
7        Set Δ := val(l(r_j)) - Mul(r_{j+1}, r_j|T_c) - deg_T(r_j) + 2
8        if Δ < 1 then
9           Discard T_c and stop
10       else
11          Mul(r_j, r_{j-1}|T_c) := Δ, and j := j - 1
12       endif
13    end; /* while */
14    if j = i then
         /* Apply the bond-cut procedure to the vertex r_i in T_c */
15       if deg_{T_c}(r_i) > val(r_i) + 1 then
16          Discard T_c and stop
17       else if deg_{T_c}(r_i) = val(r_i) + 1 then
18          Discard T_c and stop
19       else
20          Conduct the branching operation on T_c
21       endif
22    endif
23 endif.
```

Given an edge (r_j, r_{j-1}) $(k - 1 \le j \le i + 1)$ in T_c, if the multiplicity of this edge is at least 1, then we easily find that $\deg_{T_c}(r_j) = \mathrm{val}(\boldsymbol{l}(r_j))$ holds. Note that we generate all new multitrees from T by attaching a new leaf to a vertex r_i for $i = k, k - 1,..., 0$. When T_c is discarded because it violates one of the conditions in steps 2, 9, and 17, then we easily see that any younger sibling of T_c obtained by attaching a new leaf to vertices $\{r_{i-1}, r_{i-2},..., r_0\}$ in T must violate the same condition with T_c. In this case, we discard T_c and stop generating new children from T. See multitrees T_1 and T_2 in Figure 8(1) for an illustration. However, if T_c satisfies the condition in step 19, then any younger sibling T_d of T_c satisfies $\deg_{T_d}(r_i) = \mathrm{val}(\boldsymbol{l}(r_i))$, and hence T_d does not violate the valence constraint on the vertex r_i. Then, we need to check whether T_d satisfies the feature vector constraint.

Now, we explain the claim mentioned in the second paragraph of this subsection. For each valid multitree T_c, we have already determined the multiplicities of the edges on the path $(r_k, r_{k-1},..., r_i)$ before considering the next enumeration from T_c. That means that, for such a multitree T_c, the multiplicities of the edges that are not on RP(T_c) have already been determined from its ancestors and its elder siblings obtained by attaching a new leaf to the vertices $\{r_{k-1}, r_{k-2},..., r_{i+1}\}$.

4.3. Overview of Algorithm A. In this subsection, we present the outline of the algorithm discussed so far. Starting from an empty graph, Algorithm A enumerates all left-heavy

unicentroid-rooted multitrees that satisfy the feature vector and valence constraints in a recursive way.

Algorithm A

Input: a finite label set Σ, an integer K, a feature vector g with level K, and a valence function val

Output: all left-heavy unicentroid-rooted multitrees satisfying the feature vector constraints and the valence constraints

```
begin
  for all labels l in Σ do
    T := ∅;
    Attach a new leaf with the label l to T;
    Gen(T)
  end
end
```

Procedure Gen (T)

Input: a left-heavy multitree T rooted at some vertex

Output: all left-heavy unicentroid-rooted multitrees satisfying the feature vector constraints and the valence constraints

```
begin
  if the number of vertices in T is n (= Σ_{ℓ∈Σ} g(l)) then
    if T is rooted at the unicentroid and f_K(T) = g  /* bounding */ then
      Apply the bond-cut operation to T  /* bounding */
      if T is valid then
        Output T
      endif
    endif
  else if the number of vertices in T is smaller than n then
    for all vertices r_i in T to which a new leaf p can be attached do
      Attach p to r_i and gain a new multitree T_c  /* branching */
      if all subtrees obtained by removing the root of T_c have at most ⌊(n−1)/2⌋
        vertices /* bounding */ then
        for all labels that are valid for p do
          if f_K(T_c) ≤ g /* bounding */ then
            Apply the bond-cut operation to T_c /* bounding */
            if T_c is valid then
              Gen(T_c)
            endif
          endif
        end
      endif
    end
  endif
end. /* Gen */
```

On the other hand, we can output all feasible left-heavy bicentroid-rooted multitrees with even n vertices after making minor modifications on Algorithm A so as to guarantee that (1) any left-heavy multitree with less than $n/2$ vertices is rooted at a vertex and (2) any left-heavy multitree with more than $n/2$ vertices is rooted at two adjacent vertices, and all subtrees obtained by removing all edges joined by the roots from the multitree have at most $n/2$ vertices.

5. PROBLEM TRANSFORMATION

Observe that a large number of chemical compounds contain a high proportion of hydrogens. For further improvement on the efficiency of the enumeration, we reduce the size of compounds by removing all hydrogen atoms. We call this procedure the *H-removal transformation*. In addition, to simplify the inference of Problem 1, we replace all multiple edges with a new atom and two new simple edges. We call this procedure the *single-bond transformation*. Figure 9 illustrates these two transformations.

More specifically, we assume that, in the class of (Σ,val)-labeled multitrees, there exists a unique label H ($\in\Sigma$) such that val(H) = 1. Then, the H-removal transformation removes all vertices of degree 1. For a given multitree T, we replace multiple edges $\{u,v\}$ by introducing a new vertex w and two new simple edges $\{u,w\}$ and $\{w,v\}$. In addition, we say that the label $\boldsymbol{l}(w)$ of w is the *bond label* and define it to be $\boldsymbol{l}(w) = (\{\boldsymbol{l}(u), \boldsymbol{l}(v)\})$, and we say that the *bond valence* of $\boldsymbol{l}(w)$ is the multiplicity of $\{u,v\}$. Let C_Σ be the set of all such bond labels and $\Sigma^* = \Sigma \cup C_\Sigma$. For a vertex $v \in \Sigma^*$, we say that its *bond degree*, denoted by $\widetilde{deg}_T(v)$, is the number of vertices adjacent to v.

On the basis of the above definitions, we formulate a variant of Problem 1 as follows.

Problem 2. *Given a set of labels Σ^*, an integer $K \geq 1$, a feature vector $g \in F_K(\Sigma^*)$, and a valence function val: $\Sigma \rightarrow \mathbf{N}$, find all Σ^*-labeled simple trees $T^* = (V^*,E^*)$ that satisfy $f_K(T^*) = g$ (the feature vector constraint) and $\widetilde{deg}_{T^*}(v) \leq val(\boldsymbol{l}(v))$ (the new valence constraint) for all $v \in V^*$.*

For solving Problem 2, we have designed another branch-and-bound algorithm (called Algorithm B), where the branching operation is the same as that of Algorithm A but the bounding operation is different. In this algorithm, the bounding operation, called *degree-cut*, is defined on the basis of the new valence constraint. That is, when we generate a new tree T_c^* by attaching a new vertex p with some label to a vertex r_i on the rightmost path of a given tree, we do not discard T_c^* if it satisfies $\widetilde{deg}_{T_c^*}(r_i) \leq \text{val}(\boldsymbol{l}(r_i))$, and we discard T_c^* otherwise.

6. EXPERIMENTAL RESULTS

This section reports the experimental results of our algorithms. In subsection 6.1, we conduct computational experiments for Algorithms A and B. In subsection 6.2, we compare the previously fastest algorithms for enumerating alkane isomers with Algorithm B in terms of theoretical bounds on time and space complexities and running times of implementations.

6.1. Implementations of Algorithms A and B. We have tested our algorithms on a Linux PC with an AMD Sempron 3000+ CPU. We selected several appropriate chemical compounds from the KEGG LIGAND database (http://www.genome.jp/kegg/ligand.html). Since most of these compounds contain benzene rings, we treat the benzene ring as a new atom with six valences. In addition, for Problem 2, as we mentioned in Section 5, all instances for Problem 1 are preprocessed by the H-removal and single-bond transformations.

The results of computational experiments are included in Table 1, where the "entry" shows the entries of all instances under study in the KEGG LIGAND database; the names of

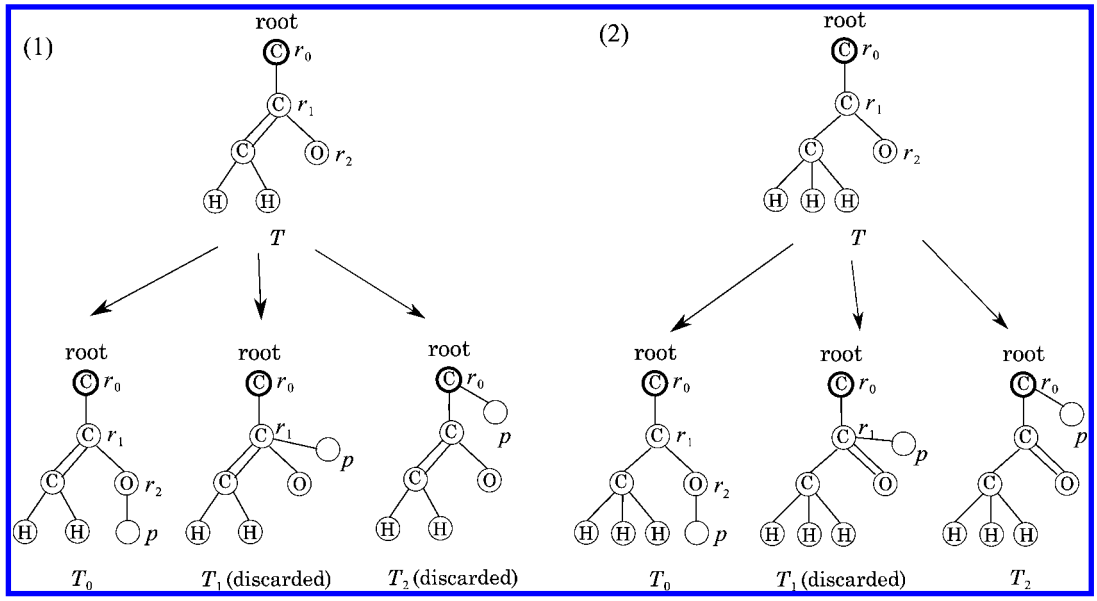

Figure 8. Illustrations of the bond-cut procedure. (1) T_0 is not discarded because there is no valence violation, whereas its younger siblings T_1 and T_2 are discarded due to $\deg_{T_1}(r_1) > \mathrm{val}(\boldsymbol{l}(r_1)) + 1$. (2) T_0 and T_2 are not discarded due to no valence violation, whereas T_1 is discarded due to $\deg_{T_1}(r_1) = \mathrm{val}(\boldsymbol{l}(r_1))+1$.

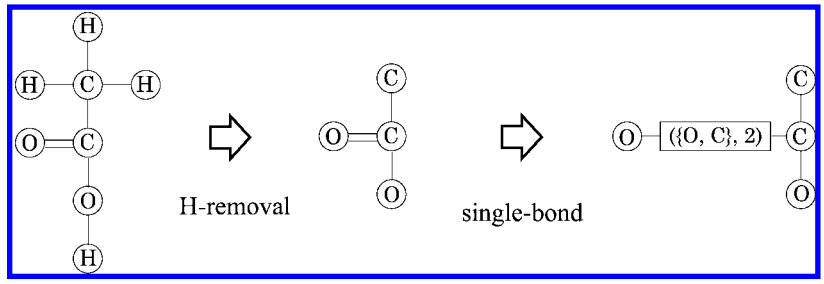

Figure 9. Illustrations of the H-removal and single-bond transformations.

these compounds are listed in the footnote of the table. n_1 and n_2 are the number of the atoms in the instances for Problem 1 and the number of the atoms in the preprocessed instances for Problem 2, respectively. *K*is the level of a feature vector. "c_time" is the CPU time for generating all solutions in seconds. "T.O." means "time over" (we set the time limit to be 1800 seconds). "nnt" is the number of nodes of the family tree that are traversed. "fs" is the number of feasible solutions within the time limit, and "fc_time" is the time when the first feasible solution is found during the execution.

From this table, we see that, in most cases, the bounding operations for both Algorithm A and Algorithm B are performed efficiently for the level of the feature vector, $K > 2$. Intuitively, the larger *K* represents the stricter feature vector constraint, and it results in a lower number of feasible solutions. Our algorithms generate all solutions by visiting the nodes which do not violate the feature vector and valence constraints in the family tree by DFS. Table 1 shows that the number (nnt) of visited nodes in the family tree drastically decreases, especially when *K* increases from one to two or a larger integer. However, it is not always true that a smaller nnt implies a shorter c_time. The reason may be that algorithms require much more time for updating the more complicated feature vectors for each new generation when *K* is larger.

On the other hand, Algorithm B works more efficiently than Algorithm A. Specifically, Algorithm B can solve the enumeration problem of treelike compounds with exact *n* vertices when $n = 61$ for all $2 \le K \le 7$, whereas Algorithm A can solve the case where $n = 46$ for all $1 \le K \le 7$. Moreover, for a moderate-size problem, Algorithm B runs much faster than Algorithm A. One reason is that the proportion of hydrogen atoms of instances under study is high, and thereby the H-removal transformation decreases the sizes of compounds drastically. Then, Algorithm B visits much fewer nodes for generating all solutions in relation to Algorithm A. Another reason is that there are few multiple edges after treating the benzene ring as a new artificial atom, and the overhead due to the single-bond transformation is small.

6.2. Application to Alkane Molecular Series. The alkane molecular family $\{C_nH_{2n+2}, n \ge 1\}$ is one of the most fundamental classes of treelike compounds, where each alkane contains only single bonds (either C−C or C−H bonds). As mentioned in section 1, recent significant developments in computers allows different computer-oriented approaches for generating all alkane isomers.[12,16–36] For example, Aringhieri et al.[36] designed two algorithms, called AHM and HPVK, to generate all alkane isomers for a given integer n (≥ 1), on the basis of the work of Trinajstić et al.,[16] and they compared these algorithms with the algorithm due to Kvasnička and Pospíchal[37] (denoted by KP), from the viewpoint of theoretical bounds on time complexities and running times of implementations. Algorithms AHM and HPVK use different canonical representations for trees. But both algorithms AHM and HPVK are designed on the basis of the same procedure, which recursively generates all nonisomorphic rooted trees

Table 1. Experimental Results of Algorithms A and B[a]

				algorithm A				algorithm B			
entry	formula	n_1/n_2	K	c_time	nnt	fs	fc_time	c_time	nnt	fs	fc_time
C03343	$C_{16}H_{22}O_4$	37/17	1	T.O.	288 904 523	55 09	87.88	281.76	19 266 745	570 773	6.01
			2	9.15	859 84	9	3.26	0.16	5167	9	0.13
			3	10.58	636 078	2	5.81	0.21	4615	2	0.17
			4	10.74	442 816	1	6.22	0.22	4092	1	0.17
			5	12.50	401 439	1	7.36	0.25	3476	1	0.20
			6	7.82	212 685	1	3.89	0.23	2911	1	0.19
			7	6.08	149 044	1	2.57	0.24	2676	1	0.19
C07530	$C_{17}H_{28}N_2O$	43/16	1	T.O.	211 304 44	7 921	12.87	95.04	4 102 957	73 711	0.09
			2	313.83	18,084 598	55	70.82	1.66	43 651	55	0.05
			3	104.21	3 579 381	1	40.68	0.81	16 131	1	0.09
			4	43.67	1 088 774	1	13.97	0.52	8019	1	0.08
			5	22.34	408 376	1	7.09	0.42	5625	1	0.07
			6	21.99	333 876	1	7.37	0.35	4643	1	0.07
			7	24.23	333 876	1	8.15	0.38	4643	1	0.08
C07178	$C_{21}H_{28}N_2O_5$	46/19	1	T.O.	240 462 194	16 389	10.54	1775.00	77 619 751	70 170	1.82
			2	201.25	13 216 617	16	2.32	1.00	21 502	16	0.02
			3	19.16	733 117	2	0.54	0.91	11 956	2	0.02
			4	4.07	115 381	1	0.76	0.30	3154	1	0.11
			5	4.23	96 857	1	0.79	0.24	2144	1	0.08
			6	4.67	93 086	1	0.89	0.26	2089	1	0.09
			7	4.22	74 677	1	0.83	0.28	2089	1	0.10
C03690	$C_{24}H_{38}O_4$	61/25	1	T.O.	262 113 862	0		T.O.	62 818 868	0	
			2	T.O.	129 353 690	0		149.61	2 986 237	1198	0.19
			3	T.O.	92 688 141	0		105.19	1 465 099	8	0.35
			4	T.O.	53 031 082	0		46.21	510 058	4	0.35
			5	T.O.	39 624 064	0		33.65	283 553	2	0.41
			6	T.O.	32 815 752	0		18.98	132 439	1	13.02
			7	T.O.	30 108 057	0		15.40	101 098	1	10.39

[a] Note: (1) C03343, C07530, C07178, and C03690 are the entries of 2-ethylhexyl phthalate, etidocaine, trimethobenzamide, and bis(2-ethylhexyl) phthalate in the KEGG LIGAND database, respectively; (2) n_1 is the number of atoms in an instance preprocessed by replacing each benzene ring with a new atom having six valences, and n_2 is the number in an instance preprocessed by replacing each benzene ring with a new atom having six valences and by the H-removal and single-bond transformations; (3) K is the level of a given feature vector; (4) c_time is the CPU time for enumerating all solutions in seconds; (5) T.O. means "time over" (the time limit is set to be 1800 seconds); (6) nnt is the number of nodes of the family trees that are traversed; (7) fs is the number of all possible solutions within the time limit; and (8) fc_time is the time when the first solution is found.

by attaching a new vertex without violating the degree constraint. To avoid generating duplications, these algorithms exploit some procedure for testing whether a newly generated tree is the canonical one among its different forms. In addition, Algorithm KP is designed on the basis of a procedure which first builds all rooted trees with at most $\lfloor n/2 \rfloor$ vertices and then constructs all rooted trees with exactly n vertices by combining all possible rooted trees with at most $\lfloor n/2 \rfloor$ vertices. Recall that Algorithms A and B proposed in this paper generate all rooted trees in the order of traversing the family tree by modifying the previous rooted tree to obtain the next one. Algorithms A and B need not test whether a tree is canonical and need not store all generated rooted trees.

Here, we focus on applying our algorithms to the problem of generating all alkane isomers with the form C_nH_{2n+2}. Through the H-removal transformation, we formulate the problem to be Problem 2: given $K = 1$ and the label set $\Sigma = \{C\}$, generate all trees consisting of n carbon atoms. Then, we implement Algorithm B to enumerate all trees with exactly n vertices such that the degree of each vertex is at most four. In what follows, we present theoretical bounds on the time and space complexities and the running time of the implementation of Algorithm B.

Upper Bounds on Time Complexities. Let S_i ($i \geq 1$) denote the number of (unrooted) trees with exact i vertices such that the degree of each vertex is at most four. Thus, S_i also implies the number of all alkane isomers with i carbon atoms. Let K_i denote the number of rooted trees with exact i vertices, such that the degree of each vertex is at most four. By definition, we easily observe

$$S_i \leq K_i \leq iS_i, \text{ for } i \geq 1 \tag{2}$$

It is known that the time complexities of Algorithms KP, AHM, and HPVK for generating all alkanes with n carbon atoms are bounded by $O(n^4K^4_{\lfloor n/2 \rfloor})$, $O(n^4S_n)$, and $O(n^4S_n)$, respectively.[36]

To compare these bounds, we use the following asymptotic behavior of S_n obtained by Pólya and Read:[9,45]

$$\lim_{n\to\infty} S_n = \alpha\beta^n n^{-2.5} \tag{3}$$

where $\alpha > 0$ and $2.85 < \beta < 2.86$ are positive constants independent of n. Through eq 3, we easily have the following upper bound:

$$S_n = O(\beta^n n^{-2.5}) \tag{4}$$

Through eqs 2 and 4, the preceding bounds on the time complexities of Algorithms KP, AHM, and HPVK are specified to be $O(\beta^{2n}n^{-2})$, $O(\beta^n n^{1.5})$, and $O(\beta^n n^{1.5})$, respectively.

We now show that the time complexity of Algorithm B is bounded by $O(\beta^n n^{-0.5})$. Recall that Algorithm B generates all rooted trees in the order of DFS applied to the family tree $F(n,1)$ with the label set $\Sigma = \{C\}$, where some of nodes

in $F(n,1)$ will not be generated due to the bounding operation by the degree constraint. Let F' be the subtree of $F(n,1)$ which consists of all nodes visited by Algorithm B. Then, the number of leaf nodes of depth n in F' is S_n.

Lemma 5. *Algorithm B runs in $O(\beta^n n^{-0.5})$ time.*

Proof. Consider the set N_i of all nodes with depth i ($< n$) in F', that is, the set of all rooted trees with exactly i vertices that are generated by Algorithm B. Since no two nodes in F' represent the same rooted tree, the number $|N_i|$ of such nodes is at most K_i. Recall that Algorithm B generates the next rooted tree from the previous one in $O(1)$ time. However, for a node in F', some of its children in $F(n,1)$ may not be generated due to the bounding operation for the degree constraint. Thus, the time delay to generate the next node from the previous one in N_i in F' is $O(i)$. Hence the total time for visiting all nodes in N_i is $O(iK_i)$, which is $O(\beta^i i^{-0.5})$ according to eqs 2 and 4. Therefore, the time complexity of Algorithm B is bounded by

$$O\left(\sum_{1\le i\le n-1} \beta^i i^{-0.5}\right) \tag{5}$$

which is $O(\beta^n n^{-0.5})$, since it holds

$$\begin{aligned}\sum_{1\le i\le n-1} \beta^i i^{-0.5} &\le \sum_{1\le j\le \lceil n/2\rceil} [\beta^j j^{-0.5} + \beta^{(j+\lfloor n/2\rfloor)}(j+\lfloor n/2\rfloor)^{-0.5}] \\ &= \sum_{1\le j\le \lceil n/2\rceil} \beta^{(j+\lfloor n/2\rfloor)}[\beta^{-\lfloor n/2\rfloor} j^{-0.5} + (j+\lfloor n/2\rfloor)^{-0.5}] \\ &\le 2\sum_{1\le j\le \lceil n/2\rceil} \beta^{(j+\lfloor n/2\rfloor)}(j+\lfloor n/2\rfloor)^{-0.5} \\ &< 2\lfloor n/2\rfloor^{-0.5} \sum_{1\le j\le \lceil n/2\rceil} \beta^{(j+\lfloor n/2\rfloor)} = O(\beta^n n^{-0.5})\end{aligned} \tag{6}$$

Upper Bounds on Space Complexities. We now analyze the space complexities of Algorithms B, KP, AHM, and HPVK in terms of "work space", the amount of memory required to generate all outputs, not the amount of memory required to simply store generated outputs.

First, we consider Algorithm B. This algorithm can construct the next rooted tree only from the previous rooted tree without knowing any information on all rooted trees generated so far. Then, Algorithm B requires $O(n)$ space for maintaining the current rooted tree. Thus, the work space of Algorithm B is $O(n)$.

Algorithms AHM and HPVK generate a new rooted tree T_d from the current one T by adding a new leaf according to two rules.[36] The first rule allows the generation of such a tree T_d as a child of T only when the code of the tree T' obtained by removing any leaf from T_d is equal to or less than that of T. For this, the algorithms need $O(n)$ space to maintain the current tree. In addition, the second rule is used to avoid generating the same children from a tree T, which has symmetric subtrees, where the information of symmetries of all subtrees can be maintained in $O(n)$ space. Thus, the workspaces of Algorithms AHM and HPVK are also $O(n)$.

In addition, recall that Algorithm KP needs to store all rooted trees with i ($1 \le i \le \lfloor n/2\rfloor$) vertices before it constructs trees with n vertices by combining all possible choices of rooted trees with at most $\lfloor n/2\rfloor$ vertices. Thus, the workspace of Algorithm KP is $O(nK_{\lfloor n/2\rfloor}) = O(\beta^n n^{-0.5})$.

Table 2 summarizes the bounds of time and space complexities of Algorithms B, KP, AHM, and HPVK. It is clear from Table 1 that Algorithm B is better than any of Algorithms KP, AHM, and HPVK in terms of the worst-case theoretical complexities.

Table 2. Complexities of Algorithms B, KP, AHM, and HPVK

complexity	algorithm B	KP	AHM	HPVK
time	$O(\beta^n n^{-0.5})$	$O(\beta^{2n} n^{-2})$	$O(\beta^n n^{1.5})$	$O(\beta^n n^{1.5})$
work space	$O(n)$	$O(\beta^n n^{-0.5})$	$O(n)$	$O(n)$

Table 3. Running Times of Algorithms B, KP, AHM, and HPVK (in seconds)[a]

n	algorithm B	KP	AHM	HPVK
16	0	0	0	0
17	0	0	2	2
18	0	2	7	6
19	0	4	16	14
20	0	15	42	37
21	0	22	112	98
22	1	92	297	257
23	2	145	789	671
24	6	618	1899	1762
25	20	958	5051	4723
26	40	4004	13 773	12 830
27	101	6385	34 615	32 975
28	262	-	-	-
29	652	-	-	-
30	1713	-	-	-
31	4904	-	-	-

[a] Note: (1) n is the number of carbon atoms of alkanes, and (2) the running times of algorithms KP, AHM, and HPVK for all cases in which $16 \le n \le 27$ in this table are reported by Aringhieri et al., where "-" represents that no results are reported in their work.[36]

Running Times of Implementations. Aringhieri et al. reported the running times of their implementations of Algorithms KP, AHM, and HPVK on a computer with a Risc IP20 100 MHz CPU and 48 MB of memory.[36] Their computational experiments presented in Table 3 show that Algorithm KP runs faster than Algorithms AHM and HPVK. Table 2 also includes the running time of our implementation of Algorithm B, where all experiments are conducted on a Linux PC with a Pentium4 3.00 GHz CPU, which is expected to be roughly 30 times faster than the computer used in the experiments by Aringhieri et al. From the results, we find that Algorithm B runs at least as fast as Algorithm KP and runs much faster than Algorithms AHM and HPVK.

7. CONCLUSIONS

In this study, we have proposed Algorithms A and B based on the branch-and-bound method for enumerating treelike chemical graphs from a given feature vector based on frequencies of paths. Algorithm A is based on the procedure which starts from an empty graph, recursively generates new left-heavy rooted multitrees by applying a branching operation to a given rooted multitree, and checks whether each newly generated multitree satisfies the feature vector and valence constraints. In addition, we have formulated a new variant of the multitree enumeration problem by exploiting the H-removal and single-bond transformations and proposed Algorithm B. The scheme of Algorithm B is similar to that of Algorithm A. The experimental results show that Algorithm B is much more efficient than Algorithm A and the existing algorithms of Akutsu and Fukagawa,[31] especially for the instances with a high proportion of hydrogen atoms

(after treating the benzene ring to a new atom with six valences). Explicitly, the algorithms of Akutsu and Fukagawa can find solutions for the treelike compounds with a size of at most 46 for $K = 4$ (after treating the benzene ring to a new atom). Algorithm A in this paper works more efficiently than their algorithms. It can solve the same size of enumeration problem for compounds (i.e., 46 atoms) with a smaller level ($K = 2, 3$). This is an improvement because fewer levels of feature vectors indicates less information needed to restrict the search space. Moreover, Algorithm B can find all solutions for certain compounds with 61 atoms for all $2 \leq K \leq 7$. This is a significant step toward the improvement of efficiency of practical algorithms for the enumeration problem of chemical compounds.

In particular, to further show the performance of our algorithms, we restricted ourselves to the enumeration problem of alkane isomers. Alkanes belong to a special class of treelike compounds, where each alkane consists of carbon and hydrogen atoms joined by single bonds. We have showed that the problem of generating all alkane isomers with a given number of atoms can be formulated as the enumeration problem with a given feature vector with level $K = 1$, and we have applied Algorithm B to the enumeration problem after removing all hydrogen atoms from compounds. We compared our algorithm with several existing efficient algorithms in terms of theoretical bounds on time and space complexities and experimental implementations. Both the theoretical and experimental results reveal that Algorithm B works at least as efficiently as the fastest one while using much less memory space.

Recall that, in the paper, we preprocessed all compounds with benzene rings by introducing a new atom with six valences, and then we implemented our algorithms. However, the algorithms proposed in this paper may not work for the case in which compounds have more complicated cycle structures than the single benzene ring. Naturally, one may restrict themselves to a more general class of labeled outerplanar graphs, which consist of edges and cycles with or without inner edges such that they are planar graphs and all vertices are on the outer face. Wang et al.[46] focused on the enumeration problem of labeled outerplanar graphs. They designed an efficient branch-and-bound algorithm that can output all outerplanar graphs with at most a given size and at most a given number of labels, where the differences between two consecutive outputs can be generated in constant time. Referring to their work, how to design an efficient algorithm for the enumeration problem of chemical compounds with outerplanar structures is an important future research topic.

Furthermore, the depth label sequences defined in this paper only represent the graphical structures of compounds in the viewpoint of planarity but may lose information of stereochemistry, especially for stereoisomers.[47] Thus, how to find good codes for compounds is no doubt an interesting topic for future research.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for their helpful comments and suggestions. We thank Mr. Yusuke Ishida for implementing our algorithm to the enumeration problem of alkane isomers. This work was supported in part by Grant-in-Aid #19200022 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan.

REFERENCES AND NOTES

(1) Cayley, A. On the Mathematical Theory of Isomers. *Philos. Mag. (1798−1977)* **1874**, *47*, 444–446.
(2) Cayley, A. On the Analytic Forms Called Trees, with Applications to the Theory of Chemical Combinations. *Rep. Br. Assoc. Adv. Sci.* **1875**, *45*, 257–305.
(3) Bytautas, L.; Klein, D. J. Chemical Combinatorics for Alkane-Isomer Enumeration and More. *J. Chem. Inf. Comput. Sci.* **1998**, *38*, 1063–1078.
(4) Henze, H. R.; Blair, C. M. The Number of Isomeric Hydrocarbons of the Methane Series. *J. Am. Chem. Soc.* **1931**, *53*, 3077–3085.
(5) Lederberg, J.; Sutherland, G. L.; Buchanan, A.; Feigenbaum, E. A.; Robertson, A. V.; Duffield, A. M.; Djerassi, C. Applications of Artificial Intelligence for Chemical Inference. I. The Number of Possible Organic Compounds. Acyclic Structures Containing Carbon, Hydrogen, Oxygen, and Nitrogen. *J. Am. Chem. Soc.* **1969**, *91*, 2973–2976.
(6) Lederberg, J. Topology of molecules. In *The Mathematical Sciences*; COSRIMS, Ed.; MIT Press: Cambridge, MA, 1969; pp 37−51.
(7) Lindsay, R. K.; Buchanan, B. G.; Feigenbaum, E. A.; Lederberg, J. *Application of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*; McGraw-Hill Companies, Inc.: New York, 1980.
(8) Pólya, G. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen, un chemische Verbindungen. *Acta Math.* **1937**, *68*, 145–253.
(9) Pólya, G.; Read, R. C. *Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds*; Springer-Verlag: New York, 1987.
(10) Read, R. C. The Enumeration of Acyclic Chemical Compounds. In *Chemical Applications of Graph Theory*; Balaban, A. T., Ed.; Academic Press: New York, 1976; 25−61.
(11) Rouvray, D. H. Isomer Enumeration Methods. *Chem. Soc. Rev.* **1974**, *3*, 355–372.
(12) 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.
(13) Jackson, M. D.; Bieber, T. I. Applications of Degree Distribution. 2. Construction and Enumeration of Isomers in the Alkane Series. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 701–708.
(14) Knop, J. V.; Müeller, W. R.; Jeričević, Ź.; Trinajstić, N. Computer Enumeration and Generation of Trees and Rooted Trees. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 91–99.
(15) Masinter, L. M.; Sridharan, N. S.; Lederberg, J.; Smith, D. H. Applications of Artificial Intelligence for Chemical Inference. XII. Exhaustive Generation of Cyclic and Acyclic Isomers. *J. Am. Chem. Soc.* **1974**, *96*, 7702–7714.
(16) Trinajstić, N.; Nicolić, S.; Knop, J. V.; Müller, W. R.; Szymanski, K. *Computational Chemical Graph Theory: Characterization, Enumeration and Generation of Chemical Structures by Computer Methods*; Simon Schuster/Horwood: Chichester, U.K., 1991.
(17) Buchanan, B. G.; Feigenbaum, E. A. DENDRAL and Meta-DENDRAL - Their Applications Dimension. *Artif. Intell.* **1978**, *11*, 5–24.
(18) Funatsu, K.; Sasaki, S. Recent Advances in the Automated Structure Elucidation System, CHEMICS. Utilization of Two-Dimensional NMR Spectral Information and Development of Peripheral Functions for Examination of Candidates. *J. Chem. Inf. Comput. Sci.* **1996**, *36*, 190–204.
(19) Fink, T.; Reymond, J. L. Virtual Exploration of the C hemical Universe up to 11 Atoms of C, N, O, F: Assembly of 26.4 Million Structures (110.9 Million Stereoisomers) and Analysis for New Ring Systems, Stereochemistry, Physicochemical Properties, Compound Classes, and Drug Discovery. *J. Chem. Inf. Comput. Sci.* **2007**, *47*, 342–353.
(20) Mauser, H.; Stahl, M. Chemical Fragment Spaces for De Novo Design. *J. Chem. Inf. Comput. Sci.* **2007**, *47*, 318–324.
(21) Faulon, J. L.; Churchwell, C. J.; Visco, D. P., Jr. The Signature Molecular Descriptor. 2. Enumerating Molecules from Their Extended Valence Sequences. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 721–734.
(22) Hall, L. H.; Dailey, E. S. Design of Molecules from Quantitative Structure-Activity Relationship Models. 3. Role of Higher Order Path Counts: Path 3. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 598–603.
(23) Byvatov, E.; Fechner, U.; Sadowski, J.; Schneider, G. Comparison of Support Vector Machine and Artificial Neural Network Systems for Drug/Nondrug Classification. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 1882–1889.
(24) Deshpande, M.; Kuramochi, M.; Wale, N.; Karypis, G. Frequent Substructure-Based Approaches for Classifying Chemical Compounds. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 1036–1050.

(25) Kashima, H.; Tsuda, K.; Inokuchi, A. Marginalized Kernels between Labeled Graphs. *Proceedings of 20th International Conference on Machine Learning*, Washington, DC, August 21−24, 2003; Fawcett, T., Mishra, N., Eds.; The AAAI Press: Menlo Park, CA, 2003; pp 321−328.
(26) Mahé, P.; Ueda, N.; Akutsu, T.; Perret, J. L.; Vert, J. P. Graph Kernels for Molecular Structure-Activity Relationship Analysis with Support Vector Machines. *J. Chem. Inf. Model.* **2005**, *45*, 939–951.
(27) Cristianini, N.; Shawe-Taylor, J. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*; Cambridge University Press: New York, 2000.
(28) Bakr, G. H.; Weston, J.; Schölkopf, B. Learning to Find Pre-Images. In *Advances in Neural Information Processing Systems*; Thrun, S., Saul, L., Schölkopf, B., Eds.; MIT Press: Cambridge, MA, 2003; Vol. 16, pp 449−456.
(29) Bakr, G. H.; Zien, A.; Tsuda, K. Learning to Find Graph Pre-Images. *Lect. Notes Comput. Sci.* **2004**, *3175*, 253–261.
(30) Akutsu, T.; Fukagawa, D. Inferring A Graph from Path Frequency. *Lect. Notes Comput. Sci.* **2005**, *3537*, 371–382.
(31) Akutsu, T.; Fukagawa, D. Inferring a Chemical Structure from a Feature Vector Based on Frequency of Labeled Paths and Small Fragments. In *Series on Advances in Bioinformatics and Computational Biology, Proceedings of 5th Asia-Pacific Bioinformatics Conference*, Hong Kong, China, January 15−17, 2007; Sankoff, D., Wang, L., Chin, F., Eds.; Imperial College Press: London, 2007; pp 165−174.
(32) Nagamochi, H. A Detachment Algorithm for Inferring A Graph from Path Frequency. *Lect. Notes Comput. Sci.* **2006**, *4112*, 274–283.
(33) Konc, J.; Janeč, D. An Improved Branch and Bound Algorithm for the Maximum Clique Problem. *MATCH* **2007**, *58*, 569–590.
(34) Konc, J.; Janeč, D. Protein-Protein Binding-Sites Prediction by Protein Surface Structure Conservation. *J. Chem. Inf. Model.* **2007**, *47*, 940–944.
(35) Konc, J.; Janeč, D. A Branch and Bound Algorithm for Matching Protein Structures. *Lect. Notes Comput. Sci.* **2007**, *4432*, 399–406.
(36) Aringhieri, R.; Hansen, P.; Malucelli, F. Chemical Trees Enumeration Algorithms. *4OR: Quart. J. Oper. Res.* **2003**, *1*, 67–83.
(37) Kvasnička, V.; Pospíchal, J. Constructive Enumeration of Acyclic Molecules. Constructive Enumeration of Acyclic Molecules. *Collect. Czech. Chem. Commun.* **1991**, *56*, 1777–1802.
(38) Kozina, A. V. Coding and Generation of Nonisomorphic Trees. *Cybernetics* **1975**, *15*, 645–561.
(39) Liu, J. Lexicographical Generation of Rooted Trees and Trees. *Kexue Tongbao* **1983**, *28*, 448–451.
(40) Nakano, S.; Uno, T. Generating Colored Trees. *Lect. Notes Comput. Sci.* **2005**, *3787*, 249–260.
(41) Tinhofer, G.; Schreck, H. Linear Time Tree Codes. *Computing* **1984**, *33*, 211–225.
(42) Wright, R. A.; Richmond, B.; Odlyzko, A.; Mckay, B. D. Constant Time Generation of Free Trees. *SIAM J. Comput.* **1986**, *15*, 540–548.
(43) Jordan, C. Sur Les Assemblages De Lignes. *J. Reine Angew. Math.* **1869**, *70*, 185–190.
(44) Nakano, S.; Uno, T. Efficient Generation of Rooted Trees. *Technical Report, NII-2003−005E*; National Institute of Informatics: Tokyo, Japan, July 3, 2003; ISSN: 1346-5597.
(45) Knopfmacher, J. Asymptotic Isomer Enumeration in Chemistry. *J. Math. Chem.* **1998**, *24*, 61–69.
(46) Wang, J.; Zhao, L.; Nagamochi, H.; Akutsu, T. An Efficient Algorithm for Generating Colored Outerplanar Graphs. *Lect. Notes Comput. Sci.* **2007**, *4484*, 573–583.
(47) Shungo, K.; Iwata, S.; Uno, T.; Koshino, H.; Satoh, H. Algorithm for Advanced Canonical Coding of Planar Chemical Structures that Considers Stereochemical and Symmetric Information. *J. Chem. Inf. Model.* **2007**, *47*, 1734–1746.

CI700385A