社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－MPS－62 (27)
2006－BIO－7 (27)
2006／12／22

特徴ベクトルに基づく木状の化学分子の列挙アルゴリズム

藤原 大樹 [1], 趙 亮 [1], 永持 仁 [1], 阿久津 達也 [2]

[1] 京都大学大学院 情報学研究科 数理工学専攻
[2] 京都大学 化学研究所 バイオインフォマティクスセンター

概要 化学分子のグラフ構造を与えられた部分構造から推定する問題は，創薬などバイオインフォマティクスの領域における基本的な課題である．本論文は，グラフにおける高々長さ K のパスの出現頻度に基づく特徴ベクトルが与えられたとき，これと同じ特徴ベクトルを有する木構造の化学グラフを全て列挙する問題を考える．また，推定するグラフの候補を制限するために，結合度の指定を入力条件に取り入れた問題の設定も行う．両者の問題に対して分枝限定法に基づく厳密アルゴリズムを提案する．既知の化合物から再推定する計算機実験では，$2 \leq K \leq 7$ のとき，提案アルゴリズムは，後者の問題に対して最大 61 原子の問題例を厳密に解くことができた．

Enumerating Tree-like Chemical Structures from Feature Vector

Hiroki Fujiwara[1], Liang Zhao[1], Hiroshi Nagamochi[1], Tatsuya Akutsu[2]

[1]Department of Applied Mathematics and Physics,
Graduate School of Informatics, Kyoto University
[2]Bioinformatics Center, Institute for Chemical Research, Kyoto University

[1]{hujiwara, liang, nag}@amp.i.kyoto-u.ac.jp,
[2]takutsu@kuicr.kyoto-u.ac.jp

Abstract Inferring chemical structures from a given partial structure is one of the fundamental problems in the field of bioinformatics such as drug design. In this paper, we consider a problem of enumerating all tree-like chemical graphs from a given feature vector that represents occurrences of vertex-labeled paths with length K. We also introduce a variant of the problem whose input contains a condition on the number of multiple bonds. For both problems, we design exact algorithms based on a branch-and-bound method. Our computational experiments reveal that, for $2 \leq K \leq 7$, the algorithm for the latter problem can find all solutions from a feature vector of a known chemical compound with at most 61 atoms.

1 Introduction

Various computational approaches have been proposed for drug design, which is one of the important targets of bioinformatics. Among those, extensive studies have been done for prediction of activities of chemical compounds. Recently, *kernel methods* have been applied to prediction of activities of chemical compounds [6, 8, 9, 11]. In most of these approaches, chemical compounds are mapped to *feature vectors* (i.e., vectors of reals) and then *support vector machines* (SVMs) [7] are employed to learn prediction rules. Feature vectors based on *frequency of labeled paths* [9, 11] or *frequency of small fragments* [6, 8] are widely used in these studies.

Kernel methods have been used mainly for prediction problems so far. However, a new approach was recently proposed for designing and/or optimizing objects using kernel methods [4, 5]. In this approach, a desired object is computed as a point in the feature space using suitable objective function and optimization technique and then the point is mapped back to the input space, where this mapped back object is called a *pre-image*. Let ϕ be a mapping from an input space to a feature space. Then, the problem is, given a point y in the feature space, to find a pre-image x in the input space such that $y = \phi(x)$. It should be noted that ϕ is not necessarily injective or surjective. If there does not exist an exact pre-image, it is desired to compute the approximate pre-image x^* defined by $x^* = \arg\min_x \; dist(y, \phi(x))$ (see Figure 1), where $dist(y, z)$ is an appropriate distance measure. If there exist several or many pre-images, it is desired to enumerate all possible pre-images. The pre-image problem for graphs is very important from a practical viewpoint because it has potential application to drug design [5] by using a suitable objective function reflecting desired properties.

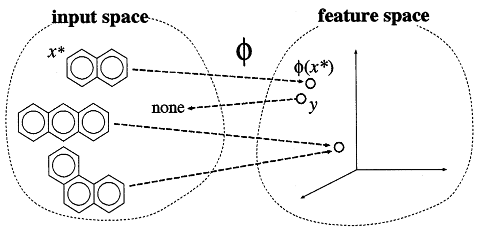


Figure 1: Feature mapping and pre-image problem for chemical compounds.

Several studies have been done on the pre-image problem, see, e.g., [4, 5]. While heuristic and/or stochastic methods were proposed in these studies, exact algorithms have recently been proposed. Akutsu and Fukagawa formalized the graph pre-image problem as a problem of inferring a graph from the numbers of occurrences of vertex-labeled paths [1]. They show this problem is NP-hard even for planar graphs of bounded degree. In [2], these results were further improved. Nagamochi developed a polynomial time algorithm for the case where the lengths of paths are less than 2 [12].

In addition to these theoretical studies, Akutsu and Fukagawa developed a branch-and-bound algorithm for inference of tree-like chemical structures [3]. It works within a few or few-tens of seconds for inference of moderate size chemical compounds with tree-like structures. However, it does not work for larger size chemical compounds. Besides, it does not output approximate pre-images or does not enumerate all possible pre-images. Therefore, further and further development should be done.

In this paper, we consider the problem of enumerating all tree-structured chemical compounds with the given feature vector. We first propose a branch-and-bound based algorithm. Then we exploit the valence of hydrogen to give a new formulation and propose a faster algorithm. The result of computational experiments show that we can treat an instance of 46 atoms (18 excluding hydrogen). This is an important step towards development of practical algorithms for the graph pre-image problem.

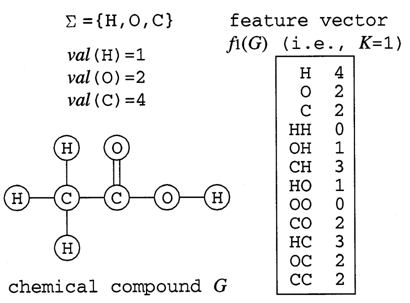
The rest of this paper is organized as follows. Section 2 gives some preliminaries, and formulates a graph inference problem. Section 3 designs a branch-and-bound algorithm for the problem. Section 4 introduces a new formulation by exploiting a condition on the number of multiple bounds as part of the input, and modifies the branch-and-bound algorithm for the new problem. Section 5 reports the results on our computational experiments, and finally Section 6 makes some concluding remarks.

2 Preliminary

2.1 Notations and definitions

Let Σ denote a set of *labels*, where each label stands an chemical element. For example, $\Sigma = \{\mathtt{H}, \mathtt{O}, \mathtt{C}\}$. A function $val : \Sigma \to \mathbb{Z}^+$ is called a *valence function*, where $\mathbb{Z}^+$ denotes the set of nonnegative integers. For example, $val(\mathtt{H}) = 1$, $val(\mathtt{O}) = 2$ and $val(\mathtt{C}) = 4$. A multigraph $G = (V, E)$ with a vertex-label function $\ell : V \to \Sigma$ is called Σ-*labeled*. Then a chemical compound can be viewed as a Σ-labeled loopless and connected multigraph such that each vertex v labeled by $\ell \in \Sigma$ has the degree $val(\ell)$. For a $K \in \mathbb{Z}^+$, define $\Sigma^{\leq K+1} = \cup_{k=1}^{K+1} \Sigma^k$. A function $f : \Sigma^{\leq K+1} \to \mathbb{Z}$ is called a *feature vector of level K over Σ* of a compound graph G if for all label sequences $s = (\ell_1, \ell_2, \ldots, \ell_k)$, $1 \leq k \leq K+1$, $f(s)$ equals to the number of paths in G whose vertices are labeled as ℓ_1, ℓ_2, ..., ℓ_k, and vice versa. The feature vector of a multigraph G of level K is denoted by $f_K(G)$.

Figure 2 illustrates a compound graph G and its feature vector $f_K(G)$ with $K = 1$.

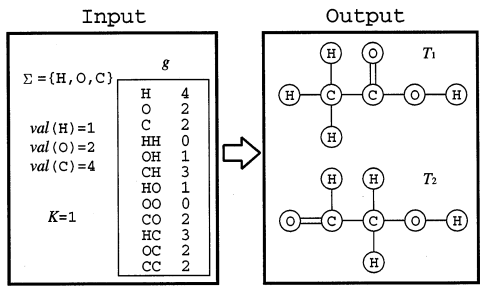


H	4
O	2
C	2
HH	0
OH	1
CH	3
HO	1
OO	0
CO	2
HC	3
OC	2
CC	2

Figure 2: Illustration of chemical compound and its feature vector.

Given a vector $g : \Sigma^{\leq K+1} \to \mathbb{Z}$, we want to find all chemical compounds G such that $f_K(G) = g$. In this paper, we focus on enumerating *tree-like* compounds. A graph is called tree-like or a *multiple tree* if it has no cycle other than a pair of parallel edges. Throughout the paper, we denote the number of vertices by $n = \sum_{\ell \in \Sigma} g(\ell)$.

Problem 1 *Given a finite set Σ, $val : \Sigma \to \mathbb{Z}^+$, $K \in \mathbb{Z}^+$ and $g : \Sigma^{\leq K+1} \to \mathbb{Z}$, find all Σ-labeled multi-trees $T = (V, E)$ such that $f_K(T) = g$ and $deg(v) = val(\ell(v))$ for all $v \in V$, where $deg(v)$ and $\ell(v)$ denote the degree and the label of v, respectively.*



g	
H	4
O	2
C	2
HH	0
OH	1
CH	3
HO	1
OO	0
CO	2
HC	3
OC	2
CC	2

Figure 3: An instance of Problem 1.

Figure 3 shows an instance of Problem 1 and two solutions.

2.2 Tree structure in detail

In order to avoid duplicate enumeration, we need a unique representation for the output trees. For this, we define a *unique "root"* of them and a *unique total order* among them. Firstly, for any (multi-)tree, the next theorem specifies a vertex or an edge as a unique root.

Theorem 1 ([10]) *For any tree of n vertices, exact one of the next two statements must hold.*

1. There exists a unique vertex v^* such that any subtree obtained by removing v^* contains at most $\lceil (n-1)/2 \rceil$ vertices.

2. There exists a unique edge e^* such that each of the two subtrees obtained by removing e^* contains $n/2$ vertices. ■

Such a vertex v^* and an edge e^* are called *unicentroid* and *bicentroid*, respectively. For example, in Figure 3, T_1 has a bicentroid (C-C), whereas T_2 has a unicentroid (the right C). Call unicentroid and bicentroid *centroid*. We will use it as the root of all multi-trees.

Next let us define a total order among rooted multi-trees. First we fix a total order to Σ (any one works). An *ordered tree* is then defined as a rooted tree in which the children vertices of any vertex or the bicentroid (if exists) are in the *descending* order (with respect to the order of Σ) from the left to the right. Let T be an ordered tree, and each vertex v is assigned a label $l(v) \in \Sigma$. Let the vertices of T has a DFS (Depth-First-Search) sequence $v_1, v_2, \ldots, v_n$ (otherwise rename the vertices), where the DFS starts from the root and traverses according to the tree order. Let $d(v)$ denote the *depth* of a vertex v. We define the *depth label sequence of* T as

$$DL(T) = (d(v_1), \ell(v_1), \ldots, d(v_n), \ell(v_n)).$$

Let the order of depth label sequences be lexicographically defined. Denote by $T(v)$ the subtree of T of root v. We say that T is *left heavy* if, for any $j > i$, v_i and v_j are siblings implies

$$DL(T(v_i)) \geq DL(T(v_j)).$$

If the tree T has an edge root e (i.e., a bicentroid), let T_1 and T_2 be the left and the right subtrees beside e, respectively. If T_1 and T_2 are both left heavy and $DL(T_1) \geq DL(T_2)$, then we say T left heavy. Taking the centroid as the root, clearly any feasible tree can be equivalently represented by a unique left-heavy tree.

2.3 Overview of the algorithm

We first give a brief overview of our branch-and-bound algorithms. The details will be described in the following sections.

The algorithm starts by enumerating the centroid (the root vertex or the root edge of the tree), and recursively add a vertex to the current tree (the *branching* operation) as far as none of the constraints (degree and feature vector) is violated (the *bounding* operation). As noted before, we enumerate left heavy trees.

Branch-and-bound algorithm has a search tree, which is called *family tree* in this paper. Denote it by $\mathcal{F}$. $\mathcal{F}$ has a root node (we say "node" to distinguish from "vertex" in the compound trees) who has two subtrees. Leaf nodes on the left are all left heavy trees with a unicentroid, whereas leaf nodes on the right are left heavy trees with a bicentroid. Let T be a node in $\mathcal{F}$. The parent node $P(T)$ of T is the tree obtained by removing the *rightmost* leaf from T. Notice that $P(T)$ remains left heavy.

3 Algorithm A

Leaving the details to the following subsections, we give an outline of the first algorithm. It is a recursive procedure, and the problem can be solved by calling Gen(r) for the root r.

```
procedure Gen($T$)
Input: a left heavy tree $T$.
Output: all feasible solutions below $T$ in $\mathcal{F}$
begin
  if the number of vertices in $T$ is $n$ then
    /* Output a feasible solution */
    if $f_K(T) = g$ then
      Calculate the multiplicities for all edges
      in $T$ and get a multiple tree $T'$.
      if $T'$ is valid then Print $T'$ endif
    endif
    return;
  endif
  /* Else expand (i.e., branch) it */
  for all vertices $w$ in $T$ to which a new leaf $p$
  can be appended do
    Append $p$ to $w$ to get a tree $T_w$;
    Apply bond-cut on $T_w$; /* bounding */
    if $T_w$ is not cut then
      for all labels $\ell$ that are valid for $p$; do
        Label $p$ with $\ell$ to get a tree $T_{w,l}$;
        if $f_K(T_{w,l}) \leq g$ then Gen($T_{w,l}$);
        endif
      done
    endif
  done
end /* of Gen */
```

3.1 Branching operation

In branching, we do not need to consider the multiplicity of edges, because they can be calculated later. In fact, the multiplicity of an edge with a leaf endpoint v is obviously $deg(v) = val(\ell(v))$, and the multiplicity of other edges can be found recursively. The family tree $\mathcal{F}$ is searched by DFS (travel from left to right). This is illustrated in Figure 4.

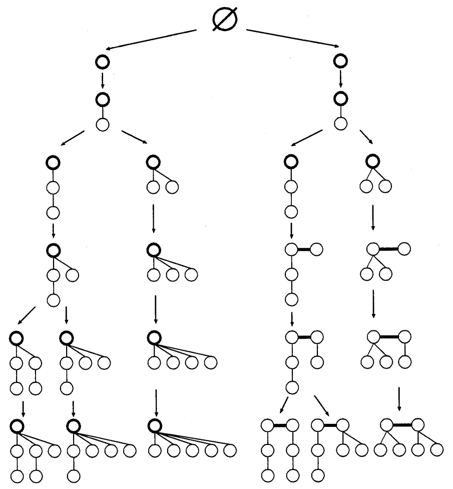


Figure 4: An illustration of DFS in the family tree (we use only one label for simplicity).

We must explain how to enumerate all children of a node T, i.e., to find all left heavy trees by adding a new leaf to T. Let $r_0, r_1, \ldots, r_k$ be the vertices on the rightmost path of T, starting from the root. If the root of T is an edge e, let r_0 be the right vertex of e. If r_i, $i \geq 1$, has an elder (i.e., left) sibling, let u_i denote the next elder sibling of r_i. We say that T is *active* at depth i, $i \geq 0$ if r_i has at least 2 children and $DL(T(r_{i+1}))$ is a prefix of $DL(T(u_{i+1}))$. If the root of T is an edge e, let u_0 denote the left vertex of e. If $DL(T(r_0))$ is a prefix of $DL(T(u_0))$, we say that T is active at depth -1. The minimum depth at which T is active is called the *copy depth* of T ($CD(T)$), which is defined as ∞ if no such a depth exists.

For ease of notation, without confusing we may write a vertex v_i as i in the following.

Lemma 1 ([13]) *The vertices to which a new leaf p can be appended are vertices on the rightmost path of T with depth smaller than the* limit depth *(denoted by $LD(T)$) of T, which is*

(1) *k if $CD(T) = \infty$;*

(2) *i if $CD(T) = i < \infty$ and $DL(T(r_{i+1})) = DL(T(u_{i+1}))$;*

(3) *$d(p - L) - 1$ if $CD(T) = i < \infty$ and $DL(T(r_{i+1})) \neq DL(T(u_{i+1}))$, where L is the number of vertices of the subtree $T(u_{i+1})$.* ■

Now we can explain how to add p to T. Due to the page limit, we only explain the case when n is even and there is a unicentroid. Referring to Lemma 1, we consider the next four cases.

(i) The number of vertices on the rightmost subtree adjacent to v_1 is $n/2 - 1$. In this case, we can append p only to v_1, otherwise v_1 will not be the unicentroid. p can be labeled by the same or lighter label than the next vertex left to p.

(ii) Else if (1) in Lemma 1 holds, adding p as the rightmost leaf with an arbitrary label can give a new node to $\mathcal{F}$.

(iii) Else if (2) in Lemma 1 holds, we can append p to r_i as its rightmost child and label it by an arbitrary label $\ell \leq \ell(r_{i+1})$.

(iv) Otherwise (4) in Lemma 1 holds. Let y be the parent of vertex $p - L$. We can append p to vertex $y + L$ as its rightmost child, and label p by an arbitrary label $\ell \leq \ell(p - L)$.

Notice that cases (ii)–(iv) add p only to the limit depth of T. To get other nodes of $\mathcal{F}$, we can simply move the added new leaf p from the limit depth to its parent, and label it by the same or lighter label than the next vertex next to p. We repeat this until p gets to the root.

3.2 Bounding operation

There are two kinds of bounding operations. One is by the feature vector. The other is by the degrees of vertices. Checking them is not difficult but we have to do it efficiently.

3.2.1 Updating feature vectors

We have three kinds of feature vectors. One is the given feature vector. The second is $f_K(T)$ for the partial tree T. The third is the difference of $f_K(T_b)$ and $f_K(T)$ for a child T_b of T. For efficiency, these feature vectors are all managed by the data structure *trie.*

Trie is a data structure used for fast searching. It consists of a vertex-rooted tree, in which a nonnegative integer $w(v)$ and a label $l(v) \in \Sigma$ (the key) are assigned to each vertex v except the root. No siblings share the same label. Suppose a feature vector g is stored in a trie $F(g)$. For any label sequence $s = (l_1, l_2, \ldots, l_i)$ with $g(s) > 0$, there exist vertices $v_0, v_1, \ldots, v_i$ satisfying the next conditions.

1 v_0 is the root of $F(g)$. For all $j = 1, 2, \ldots, i$, v_{j-1} is the parent of v_j.

2 $l(v_1) = l_1$, $l(v_2) = l_2$, $\ldots$, $l(v_i) = l_i$.

3 $w(v_i) = g(t)$.

Let us explain how to update the trie (i.e., the feature vector) of the current partial tree. Suppose T and $f_K(T)$ are known, and a new leaf p is just added with label l_1. Denote the new tree by T_1. We want to compute $f_K(T_1)$.

Let d_1 be the difference between $f_K(T_1)$ and $f_K(T)$. It must be a feature vector for paths in T_1 starting from p, and thus can be obtained by a DFS (with path length limit K) in T_1 starting from p. $F(d_1)$ can be constructed during the DFS. By applying DFS in $F(d_1)$ and $F(f_K(T))$ simultaneously, we can obtain $F(f_K(T_1))$ by composing $F(f_K(T))$ and $F(d_1)$.

Once we have done for one label, the feature vectors of other nodes differ only by the label of p can be easily found. Moreover, we can store $F(f_K(T))$ globally for efficiency. When a child is generated, we compose the difference, and when the search for the child finished, we decompose the difference. Further details are omitted due to space limitation.

3.2.2 bond-cut

We introduce a bounding operation, called *bond-cut* based degree constraints. Let T be the current partial tree. We append new vertex only to vertices on the rightmost path of T. This means that the multiplicity of edges not on the rightmost path can be decided.

Consider the first time when a new leaf p is appended to the tree T (cases (i)–(iv) in the branching operation). Let the tree be T_1. Denote the parent of a vertex x in T_1 by $q(x)$. If $w = q(p) = p - 1$, then we cannot cut T_1 by degree (i.e., there is no violation). Otherwise $w \neq p - 1$. Let P be the path from $p - 1$ to w. We update and cut nodes by referring cases (i)–(iv) stated in the branching operation. Let $deg_{T_1}(v)$ denotes the degree of a vertex v in T_1.

(i) Find the multiplicity for all edges (starting from the leaf) on P by $val(\ell(v))$. If the multiplicity of some edge is nonpositive, cut T_1. If $deg_{T_1}(v_1) > val(\ell(v_1))$, then T_1 can be cut still.

(ii) Cut T_1 if $val(\ell(w)) < deg_{T_1}(w)$.

(iii) The multiplicity of any edge $\{v, q(v)\}$ on P is the same as the multiplicity of $\{v - L, q(v - L)\}$. Cut T_1 and all its younger siblings if $val(\ell(w)) - deg_{T_1}(w) \leq -2$. If $val(\ell(w)) - deg_{T_1}(w) = -1$, cut T_1 only.

(iv) The multiplicity of any edge $\{v, q(v)\}$ on P is the same as the multiplicity of the edge $\{v - L, q(v - L)\}$.

Consider other nodes of $\mathcal{F}$ (see the last paragraph in 3.1). Suppose w is not an endpoint of the bicentroid. Let $z = q(w)$ and a new tree $T_2 \in \mathcal{F}$ is obtained by appending p to z. Notice the multiplicities of edges on P in T_2 are equal to those in T_1. For edge $\{z, w\}$, we can compute its multiplicity by $val(\ell(w))$ and $deg_{T_2}(w)$. If $val(\ell(z)) - deg_{T_2}(z) \leq -2$, we can cut T_2 and all its younger siblings. If $val(\ell(z)) - deg_{T_2}(z) = -1$, then we cut T_2 only. Repeat this until p gets to the root. Note that the label of p does not matter in bond-cut.

4 Problem with bonds condition

This section introduces a new formulation of the problem to restrict a class of multi-trees.

Let $\ell_1, \ell_2 \in \Sigma$ and b be a positive integer, we call $c = (\{\ell_1, \ell_2\}, b)$ a *bond label*, and b the *bond* of c. Let C_Σ denote the set of all bond labels. We suppose that there always exists a unique label H with $val(\mathtt{H}) = 1$. The H*-removal transformation* is to remove all vertices labeled H. Let $\Sigma^* = \Sigma \cup C_\Sigma$. The *single bond transformation* is to replace multiple edges $\{u, v\}$ by a new vertex w and two new edges $\{u, w\}$ and $\{w, v\}$. The label of w is set to $(\{\ell(u), \ell(v)\}, b)$, where b is the multiplicity of $\{u, v\}$. The *bond of edge* $\{u, v\}$ is defined by

(1) the bond of $\ell(v)$ if $\ell(u) \in \Sigma$ and $\ell(v) \in C_\Sigma$,

(2) 1 otherwise (i.e., $\ell(w), \ell(v) \in \Sigma$).

The *bond degree* of a vertex $v \in \Sigma^*$ is defined by $\widetilde{deg}(v) = \sum\{b(\{v, w\}) \mid w \text{ is adjacent to } v\}$. See Figure 5 for an illustration for the transformations.

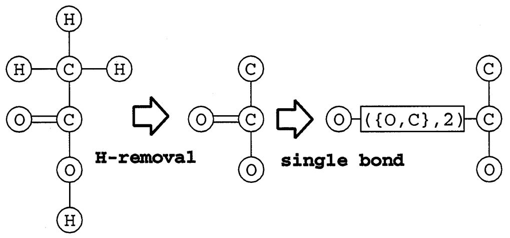


Figure 5: Illustration for the two types of transformations: H-removal and single bond.

Now we can give the new problem definition.

Problem 2 *Given a finite set Σ, $\Sigma' \subseteq C_\Sigma$, $val : \Sigma\cup\Sigma' \to \mathbb{Z}^+$, $K \in \mathbb{Z}^+$, $g : (\Sigma\cup\Sigma')^{\leq K+1} \to \mathbb{Z}$, find all $(\Sigma\cup\Sigma')$-labeled trees $T = (V, E)$ such that $f_K(T) = g$ and $\widetilde{deg}(v) \leq val(\ell(v))$ for all $v \in V$, where $\widetilde{deg}(v)$ and $\ell(v)$ denote the bond degree and the label of v respectively.*

We can design an algorithm, called B to this problem by sharing the same branching operation with Algorithm A (the details are omitted). The difference is the bounding operation. Algorithm B also uses the next degree-cut: when a branching operation tries to attach a new vertex p to v, expand the node as long as $\widetilde{deg}(v) \leq val(l(v))$ holds, otherwise terminate the branching operation. Note that no bond-cut will be used in Algorithm B.

5 Computational experiment

We have tested our algorithms on a Linux PC with CPU AMD Sempron 3000+. The time limit was set to 1800 seconds. The instances were obtained from the chemical compounds in the KEGG LIGAND database (see http://www.genome.jp/kegg/). For Problem 1, we randomly picked up multiple trees containing the hydrogen atom and treat benzene ring as one atom of valence 6. Instances of Problem 2 were obtained by applying the H-removal and single bond transformations.

The results are shown in Table 1, where "name" is the name of chemical compound (KEGG number). Numbers n_1, n_2 and n_3 are the numbers of the atoms, atoms except hydrogen and the number of vertices for Problem 2, respectively. K is the level, and c_time is the CPU time in second. T.O. means "time over", "ppe" is the number of nodes of the family tree that were expanded, "fs" is the number of feasible solutions found and "fc_time" is the time when the first feasible solution was found.

From the table, we can observe that when K increases, ppe drastically decreases. This is because the constraints used for bounding increased (the number of feasible solutions decreases drastically). However, there are cases in which ppe decreases but the CPU time increases. This is due to the updating complexity of feature vector. Notice the CPU time of Algorithm B is much less than Algorithm A. This is due to the next observations.

(1) The percentage of hydrogen atoms is large.

(2) Most of the edge multiplicities are 1 (i.e., single bond).

Hence the decrease of vertices due to H-removal transformation is much larger than the increase of vertices due to single bond.

6 Conclusion and future work

In this paper, we have shown two branch-and-bound algorithms for inferring chemical structure problems. In the branching operation, we use labeled tree to avoid duplicate output of isomorphic trees. For the bounding operation in the first algorithm, we have developed an efficient bond-cut and employ new data structure and methods for efficiency.

Moreover, by exploiting a condition on the number of multiple bonds, we introduced a new problem formulation and showed an even more efficient algorithm. Computational experiments show that both algorithms can solve many instances including a large one (C07178) of 46 atoms (18 excluding hydrogen) for all $K = 2, 3, \ldots, 7$. The second algorithm can even solve an instance of 61 atoms for all $K = 2, 3, \ldots, 7$.

It is left as a future work to solve instances with larger number of atoms.

References

[1] T. Akutsu and D. Fukagawa. Inferring a graph from path frequency. *Lecture Notes in Computer Science*, 3537:371–392, 2005.

[2] T. Akutsu and D. Fukagawa. On inference of a chemical structure from path frequency. *Proc. BIOINFO 2005*, 96–100, 2005.

[3] T. Akutsu and D. Fukagawa. Inferring a chemical structure from a feature vector based on frequency of labeled paths and small fragments. *The 5th Asia Pacific Bioinformatics Conference* (APBC 2007), to appear.

[4] G. H. Bakir, J. Weston and B. Schölkopf. Learning to find pre-images. *Advances in Neural Information Processing Systems*, 16:449–456, 2003.

[5] G. H. Bakir, A. Zien and K. Tsuda. Learning to find graph pre-images. *Lecture Notes in Computer Science*, 3175:253–261, 2004.

Table 1: Computational experiment result (see Section 5)

			Algorithm A				Algorithm B			
name	n1/n2/n3	K	c_time	ppe	fs	fc_time	c_time	ppe	fs	fc_time
C03343	37/15/17	1	T.O.	288,904,523	55,509	87.88	281.76	19,266,745	570,773	6.01
C03343	37/15/17	2	9.15	859,084	9	3.26	0.16	5,167	9	0.13
C03343	37/15/17	3	10.58	636,078	2	5.81	0.21	4,615	2	0.17
C03343	37/15/17	4	10.74	442,816	1	6.22	0.22	4,092	1	0.17
C03343	37/15/17	5	12.50	401,439	1	7.36	0.25	3,476	1	0.20
C03343	37/15/17	6	7.82	212,685	1	3.89	0.23	2,911	1	0.19
C03343	37/15/17	7	6.08	149,044	1	2.57	0.24	2,676	1	0.19
C07530	43/15/16	1	T.O.	211,304,644	7,921	12.87	95.04	4,102,957	73,711	0.09
C07530	43/15/16	2	313.83	18,084,598	55	70.82	1.66	43,651	55	0.05
C07530	43/15/16	3	104.21	3,579,381	1	40.68	0.81	16,131	1	0.09
C07530	43/15/16	4	43.67	1,088,774	1	13.97	0.52	8,019	1	0.08
C07530	43/15/16	5	22.34	408,376	1	7.09	0.42	5,625	1	0.07
C07530	43/15/16	6	21.99	333,876	1	7.37	0.35	4,643	1	0.07
C07530	43/15/16	7	24.23	333,876	1	8.15	0.38	4,643	1	0.08
C07178	46/18/19	1	T.O.	240,462,194	16,389	10.54	1775.00	77,619,751	70,170	1.82
C07178	46/18/19	2	201.25	13,216,617	16	2.32	1.00	21,502	16	0.02
C07178	46/18/19	3	19.16	733,117	2	0.54	0.91	11,956	2	0.02
C07178	46/18/19	4	4.07	115,381	1	0.76	0.30	3,154	1	0.11
C07178	46/18/19	5	4.23	96,857	1	0.79	0.24	2,144	1	0.08
C07178	46/18/19	6	4.67	93,086	1	0.89	0.26	2,089	1	0.09
C07178	46/18/19	7	4.22	74,677	1	0.83	0.28	2,089	1	0.10
C03690	61/23/25	1	T.O.	262,113,862	0		T.O.	62,818,868	0	
C03690	61/23/25	2	T.O.	129,353,690	0		149.61	2,986,237	1,198	0.19
C03690	61/23/25	3	T.O.	92,688,141	0		105.19	1,465,099	8	0.35
C03690	61/23/25	4	T.O.	53,031,082	0		46.21	510,058	4	0.35
C03690	61/23/25	5	T.O.	39,624,064	0		33.65	283,553	2	0.41
C03690	61/23/25	6	T.O.	32,815,752	0		18.98	132,439	1	13.02
C03690	61/23/25	7	T.O.	30,108,057	0		15.40	101,098	1	10.39

[6] E. Byvatov, U. Fechner, J. Sadowski and G. Schneider. Comparison of support vector machine and artificial neural network systems for drug/nondrug classification. *Journal of Chemical Information and Computer Sciences*, 43:1882–1889, 2003.

[7] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge Univ. Press, 2000.

[8] M. Deshpande, M. Kuramochi, N. Wale and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowledge and Data Engineering*, 17:1036–1050, 2005.

[9] H. Kashima, K. Tsuda and A. Inokuchi. Marginalized kernels between labeled graphs. *Proc. 20th Int. Conf. Machine Learning*, 321–328, 2003.

[10] V. Kvasnička and J. Pospichal. Constructive Enumeration of Acyclic Molecules. *Collect Czech Chem Commun*, 56:1777-1802, 1991.

[11] P. Mahé, N. Ueda, T. Akutsu, J-L. Perret and J-P. Vert. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *Journal of Chemical Information and Modeling*, 45:939–951, 2005.

[12] H. Nagamochi. A detachment algorithm for inferring a graph from path frequency. *Lecture Notes in Computer Science*, 4112:274–283, 2006.

[13] S. Nakano and T. Uno. Efficient Generation of Rooted Trees. *NII Technical Report*, NII-2003-005E, 2003.