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

2006－AL－106 (4)
2006／5／18

分子合成経路発見のためのグラフタイリングアルゴリズム

†末松 宏一 ‡荒木 通啓 °渋谷 哲朗

†東京大学大学院情報理工学系研究科コンピュータ科学専攻
〒113-0033 東京都文京区本郷 7-3-1
E-mail: pascal@is.s.u-tokyo.ac.jp
‡東京大学医科学研究所ヒトゲノム解析センター
〒108-8639 東京都港区白金台 4-6-1
E-mail: maraki@hgc.jp
°東京大学医科学研究所ヒトゲノム解析センター
〒108-8639 東京都港区白金台 4-6-1
E-mail: tshibuya@hgc.jp

概要: 現在、小さな分子同士の特定の箇所を既知の酵素によって結合させ、次第に大きな分子を合成していくことによって未知の分子を合成することが研究されている。この際目的分子のグラフ構造を与えられた分子のグラフ構造を用いて敷き詰めることが必要となるが、この問題は一般に非常に難しく膨大な計算コストがかかることが知られている。我々は分子の特徴を用いたアルゴリズムを構成し、敷き詰められる対象の分子の原子数を N、敷き詰める部品となる全ての分子に含まれる原子数を M としたときに $O(NM^3)$ の前処理を行うことによって可能な敷き詰め方を一つあたり定数時間で高速に列挙できることを示す。またそれを実装してこのアルゴリズムの有用性を検証する。さらにその後、同じ前処理が $O(NM)$ で実行可能であることを示す。

Graph tiling algorithm for molecular synthesis analysis

†Koichi Suematsu ‡Michihiro Araki °Tetsuo Shibuya

†Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan.
E-mail: pascal@is.s.u-tokyo.ac.jp
‡Human Genome Center, Institute of Medical Science, University of Tokyo
4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan.
E-mail: maraki@hgc.jp
°Human Genome Center, Institute of Medical Science, University of Tokyo
4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan.
E-mail: tshibuya@hgc.jp

Abstract: An enzyme works between specific molecules to compound a larger molecule and lots of such reaction is already known, thus to discover a new synthetic pathway by using such reactions is being researched. In the method, it is necessary to tile the target's molecular structure by using the graph structures of given molecules. The problem is shown to be very difficult and usually takes very long time. We here show an algorithm which considers the molecular graph structure and shows that every possible tiling results can be got in constant time after short time pre-calculation. We also implemented the algorithm and confirmed that it works at very high speed.

1 Introduction

In biology, compounding a molecule such as medicine with the given molecular structure is attracting a big interest. However, there is no general method to achieve this now, because the synthetic pathway used to compound a molecule is given only by the past experiences of analyzing pathways. There are also some molecules whose pathway is known but which cannot be synthesized because of technical problems. Be-

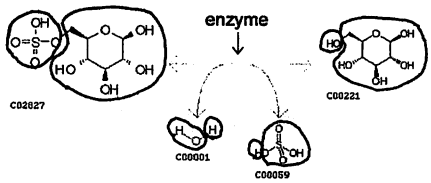

Figure 1: Example of tree tiling problem

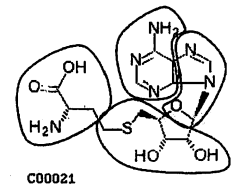

Figure 2: Example of tree tiling problem

cause of such reasons, there are many molecules that are medically of great interests or importance but cannot be synthesized. This results in a problem that such molecule cannot be mass-produced.

The method used now costs high because it is not completely automated. An efficient and easy synthetic pathway to make a molecule with the given structure is expected to enable mass production of cheaper medical molecules. Moreover, discovering the best synthetic pathway with fewer procedures, better reaction efficiency, less technical limitations, etc. of the molecule is also important when the molecule has multiple synthetic pathways. These features enable synthesizing a lot of molecule at low cost.

Thus we focused on designing an algorithm to suggest a new synthetic pathway. Finding a new pathway through combining the individual reaction is a possible way to help this.

An enzyme works between specific molecules to compound a larger molecule, and lots of such reactions are already known. These reaction data of biological pathway is open to public by KEGG[6], and we can get biological pathways on the internet. Now, some molecular blocks which consist molecules appear in these pathways are being listed. Molecular block is a group of atoms, and the uniting of atoms in a block does not change before and after the reaction. A sample of molecular block is shown in Figure 1. There are four blocks in the figure and their change of uniting means chemical reaction.

To find a new pathway, the next step is to find the best combination of these molecular blocks to compose a target molecule like Figure 2.

In this paper, we aim to propose an efficient graph tiling algorithm. The graphs we consider are molecular structures, thus the maximum degree is limited to 4 and the graph is not so complicated as to be represented as some combination of trees.

To suggest an efficient algorithm for the problem, we first give a polynomial time algorithm to tile a tree of bounded degree with some smaller unordered unrooted labeled trees. The algorithm aims to reduce redundant calculation in every step through detecting many kinds of symmetry and identifying the isomorphic graphs. It works at $O(NM^3)$ time, where N is the size of target tree and M is the number of all vertices in pattern trees.

The algorithm gives a combination of patterns that are used in tiling results. By using the result, each tiling results can be obtained in constant time. It also enables to obtain the number of tiling results in the time proportional to the size of the target tree.

We also implemented the algorithm and showed the order is just a redundant upper bound of the algorithm. The algorithm works almost linear to N and M with practical data we tested.

Then we improved the algorithm and show that the calculation time can be reduced to $O(MN)$.

We also show a plan for tiling a graph, which may be efficient in solving the problem.

2 Preliminaries

Graph tiling problem Graph tiling problem is defined as follows. Tiling a given graph G with a given set of graph g is to divide G into $S = \{g_1, g_2, \cdots g_n\}$ which satisfies the following conditions.

1. g_i is isomorphic to one of the elements in g.
2. $\cup_{i=1}^{n} V(g_i) = V(G)$
3. $V(g_i) \cap V(g_j) = \phi$ (for every $1 \leq i < j \leq n$)

Notice that a graph in the given set of graphs g can be used more than once, or can be left without being used.

Tree tiling problem The tree tiling problem is a kind of graph tiling problem, with a restriction that every graph appears in the problem is a tree. The problem argued in this thesis is related to labeled or unlabeled unordered unrooted trees.

Other definitions We use the word 'target tree' for the meaning of the tree to be tiled, and 'pattern trees' for the meaning of the trees to tile a target tree with.

3 Related works

Graph tiling problem Given a graph H, the H-decomposition problem is stated as follow: Can the edge set of an input graph G be partitioned into subgraphs, isomorphic to H?

The problem is shown to be NP-complete[3]. The problem of graph tiling just exchanged the vertices and edges of this problem, which shows that the problem of graph tiling is essentially NP-complete. Our problem is stated as a restricted problem of this problem on the point that the graphs in our problem have bounded degree 4, for they are molecules.

Tree tiling algorithm The problem of tiling a tree or tiling a graph has been hardly done up to now, but a kind of tree tiling problem similar to our problem exists in compiler theory, which is used to convert the tree data structure into sequential instructions by using a tiling result[4]. In the problem, some existing methods use dynamic programming and solve the problem in linear time [1]. The limitation is that, those methods only considers labeled ordered trees.

The problem with unordered tree has not been considered in the existing algorithms, and they cannot be simply applied to our problem. But the method in compiler shows that bottom-up tiling method with dynamic programming is suitable for obtaining tiling results effectively.

General method used in tree problem Top-down and bottom-up tiling strategies were made up in order to find a pattern from trees effectively[5]. Many effective algorithms were constructed by using these methods.

Tree normalizing algorithm An algorithm of transforming a labeled or unlabeled unordered tree into an ordered tree is designed[7]. This algorithm seems to transform the tiling problem with unordered tree into the problem with ordered tree that makes it possible to apply the compiler tiling algorithm to this problem, but the problem is that this algorithm is a kind of normalizing an unordered tree, so this algorithm can be used only when the remaining trees after tiling some of the target tree are guaranteed to be normalized.

Numbering tree structures An algorithm to represent a tree structure as a number exists [7]. The number cannot reconstruct the original tree but the point is that the trees with the same structure have the same number. In the thesis, the number is generated dynamically at every depth of a tree to reduce the computational complexity, but the idea of representing a tree in a number can be used in other ways. This thesis shows a linear time algorithm for enumerating the trees who are isomorphic to the bottom-up subtree.

Feature of biological molecules In general, problems of comparing two graphs are intractable. For example, subgraph isomorphism problem is NP-hard. Many of biological structure are represented using graph structure, but for the reason, methods using graph theory was not a main stream of analyzing biological pathway. However, their width of the graph was shown to be very small[9], which means that the structure is not very far from trees. In other words, a biochemical compound becomes tree when some of their atoms are removed.

4 Approach to the Graph Tiling Problem

Later, we show an efficient algorithm for tree tiling, which takes time proportional to the size of target tree. The efficiency depends on three features.

1. A bottom-up tiling approach can be taken which enables efficient branch hunting.
2. Identifying a larger structure from their chil-

dren and its label is easy.

3. The number of bottom-up subtrees got in the initialization process is linear to the number of vertices in pattern tree. This works in two ways, less processing time and less use of memory.

On the other hand, graph tiling problem cannot be simply solved by using bottom-up approach because there is generally no leaf in a graph. This makes it difficult to apply the sequence : find a match, remove the matched tree which is isomorphic to a pattern tree, and makes the problem smaller. Moreover, when we try to use the similar approach, it is necessary to enumerate the subgraph of all patterns in the preprocessing, because the bottom-up approach cannot be taken. The number of subtrees is enormous, then the processing time and the memory used by the algorithm becomes so large, which is not practical.

Thus we decided to devise another method. In general, molecular structures do not unit too closely. Moreover, they are often consisted of a structure near to tree and some small complex part. From these features, we can treat molecular graphs as some trees when some vertices of a graph and edges which contains removed vertices as their edge point are removed. Most molecules appear in pathways become some trees when at most 3 vertices are removed. Thus we may solve the graph tiling problem efficiently.

5 Approach to the Tree Tiling Problem

5.1 Overview

In solving the tree tiling problem, we aimed to enumerate each tiling result in constant time because there was a possibility that the number of results becomes $O(e^n)$ where n is the size of the target tree.

The problem causing the enormous number of results is that the tiling results of the parts nearer to a leaf than a certain point are independent of the tiling results of the parts nearer to the root than a certain point. Therefore, we decided to express all the solutions in the shape of the pattern of combining. Thus the number of matched patterns we should get becomes polynomial number, and it can be easily confirmed to be able to enumerate each of the solutions in constant time by combining them. Moreover, after getting the combination, we can count up the number of all tiling results in $O(n)$ time, where n is the size of the target tree.

We here explain a fast algorithm to calculate such combination.

5.2 Main Idea of the Algorithm

We organized an algorithm by using a dynamic programming method with bottom-up searching method, which makes it possible to find subtrees which are isomorphic to a pattern tree in short time. Moreover, we reduced the time in checking isomorphism of trees from $O(n)$ to $O(1)$ by using the strategy of tree-index described below, where n is the size of trees. We pre-calculate layered structure of trees to make the comparison and to combines trees faster. The main problem in tree tiling is to find the isomorphic tree that can appear in the tiling result fast, so the pre-calculation method of this kind is efficient.

In addition, to avoid testing the same tree more than once, we identified the same structure in different trees. This also enables to detect symmetry in a tree, so this method works in two ways, and greatly reduces the computing time.

5.2.1 Bottom-up tiling approach

Definition When a rooted tree T is given, S is a bottom-up subtree of T if for any vertex $v \in V$ that is not a leaf of T we have $children(v) \subset V$, where V is vertices of S.

The problem in top-down tiling approach is that the order of the children is not fixed. The key factor is to diverge from one point to one or more points.

By using bottom-up method, we should just combine some points into one, thus the combination can be brought together in one with the time proportional to the number of the children by using function F described in section 5.2.2.

The method integrates some points into one at each step, and the explosion of the computational complexity caused by the permutation of the children can be suppressed. Thus the bottom-up approach enables fast comparison between pattern trees and bottom-up subtree of the target tree. By using this method, the number of vertices in the subtree we should check becomes smaller linearly. To apply the bottom-up technique to this problem, we adopt

the method to cut out the matched pattern from the leaf. If two or more matches exist in the procedure, it is necessary to think ahead all of that after cutting out those patterns.

5.2.2 Tree Normalization by Indexing Trees

It is necessary to regularize the tree to solve this problem by using a similar technique to the tiling of the ordered tree. It is possible to treat by sorting the identifier as well as the ordered tree can give identifier to the unordered tree. We decided to allot the integer as identifier of the tree by using the fact that the number of all trees that appeared in the algorithm is little. It works efficiently compared with the retrieval of the pattern etc. because this can show the relation between the tree and the tree by integers.

In our algorithm, comparison and uniting of trees often appears. Uniting of trees means making a new tree from the children and the label of itself. The root of the new tree has label L and every root of its children $\{t_1, t_2, \cdots, t_n\}$ will be jointed to the new root. In other words, those roots become children of the root vertex.

To decrease the computational complexity of those basic procedures, we indexed every rooted trees appears in our tiling algorithm. We call the number tree-index, which is an integer which is unique to the structure of an unordered rooted tree. In our algorithm described below, the number of unordered rooted tree is bounded by the size of pattern trees, which means the number does not become so large in practical problem.

Tree-index enables us to check whether given two trees are isomorphic or not in constant time, on condition that the tree-index of each tree is already calculated.

Theorem Let sum of number of vertices contained in the tree of all patterns be M. The number of trees got by this technique is less than $3M$.

Proof The number of trees that are isomorphic to a pattern tree is obviously M or less. The tree with m vertices are consists of $m-1$ edges, and the sum of the degree of every point is $2m-2$. When we remove a point from a tree, the number of newly created graphs is the degree of the removed point, thus the upper bound of the subtrees in the algorithm is at most $2m-2$, which is less than twice the size of the tree. For the reason, the newly created trees appear in the algorithm is less than $2M$. Thus, the number of trees appear in the algorithm is $M+2M$ at most above.

From the theorem, the indexing is shown to be practical because the maximum number appears in the indexing process is at most $3M$, which is small enough to be compared in constant time.

5.3 Tree Tiling Algorithm

5.3.1 Data Structure in Tiling a Given Tree

- In the target tree, each vertex v_i holds a set $s(v_i)$, which contains set of pairs $P_j = (I_j, M_j)$. I_j stands for the tree-index of a subtree which is isomorphic to the subtree and the root of the subtree is v_i in the target tree. M_j stands for the matching of the children of v_i, showing which child correspond to which tree-index. M_j is used to show the result of matching.
- Each vertex v_i holds a set $S(v_i)$, which is almost the same as $s(v_i)$. The only difference is that the pair whose flag corresponding to the tree-indexes is not checked is not in the set.

5.3.2 Main Algorithm

Algorithm 1 *First, we initialize tree-index table T, which includes unions each of which consist of tree structure, its tree-index, and a flag. The flag means whether there is such pattern tree that is isomorphic to the structure or not.*

1. *Initialize T as $\{(\phi, 0, false)\}$, which stands for a virtual tree with no vertex.*
2. *List up and index all bottom-up tree structures.*
   (a) *Let N be an integer 1.*
   (b) *For each pattern tree p, and for each vertex v in p, apply the following.*
   (c) *Think as if v is a root vertex of p, and call the rooted tree p'. Try to find if p' is already in T. If p' is already included in T, change the flag in the union to true. Else, put the pair $(s, N, true)$ into T and increment N.*

(d) *Think as if v is removed from p, then pattern tree p will be divided into some subtrees whose root vertex is a children of v. For each subtree s, try to find if s is already in T. If s is already included in T, do nothing. Else, put the pair $(s, N, false)$ into T and increment N.*

3. *Construct a synthesis table. For each pattern tree p, apply the following.*

   (a) *For each vertex v in p, think a tree p whose root vertex is v. We call the rooted tree p'.*

   (b) *Think as if v is removed from p, then pattern tree p will be divided into some subtrees. For each subtree, regard the child of v as a root of the tree and call it v'_i.*

   (c) *Put tree-index of p', tree-indexes of v'_i, and label of v into synthesis table so that we can get the tree-index of p' from the combination of other data.*

Algorithm 2 *Here is the main part of the algorithm. The following are applied to vertices by post-order. Let v be the vertex we are looking at, and $v_1, v_2, \cdots, v_n$ be the children of v.*

1. *When v has no child, regard the vertex as it has a child and the child have returned a set $\{0\}$. Then continue the following. 0 is a tree-number of null tree.*
2. *When there is a child with condition $s(v_i) = \phi$, no valid tiling exists.*
3. *Otherwise, choose a tree-index from each $s(v_i)$. For every combination of the elements in $s(v_i)$, check whether the combination of the tree-indexes consists a indexed tree or not from synthesis table. The check can be done in constant time and the number of possible combination is at most $(3M)^3$, so the order of the time if $O(M^3)$. For each tree-index, do the following.*

   If the flag corresponding to the tree-number is checked, the pattern is isomorphic to one of the pattern trees. If the condition is satisfied, add the tree-index into $S(v_i)$ and add 0 to $s(v_i)$ which represents an empty tree. Otherwise, add the tree-index to $s(v_i)$.

5.3.3 Computing Time

Initializing part Let m_i be the number of vertices in ith pattern tree. 4, which is the maximum degree of the graph, mutiplied by m_i bottom-up subtrees are created at each tile and the time of creation is m_i when we used post-order array in the process. Thus the time with the ith pattern tree is $O(m_i^2)$. The total time in initialization becomes $\sum_i (m_i^2)$.

Main part Let N be the number of vertices in the target tree, M be the sum of the number of vertices in all pattern trees. The number of elements in $s(v)$ that each point possesses is at most $3M$ because the size of table T is at most $3M$. 4 pieces of such parts of patterns are combined in one larger piece, then the number of combination is at most $(3M)^4$. This order can be improved easily, because most vertex does not have 4 children. The only vertex which has possibility of having 4 children is the root vertex, and we can choose root vertex with condition that it has only one child. If the root vertex has 4 children, choose a leaf of the tree. The leaf has only one neighbours as definition, so choose the leaf as a root of the tree. Thus the maximum children number becomes 3, and the order of combination becomes $O(M^3)$. This type of combination is executed at every vertex, so the computational time by applying this method is $O(NM^3)$.

6 Experimental Results

In the experiments, we chose such target tree that can be tiled with the given patterns because the tiling time becomes very short in the case the target tree cannot be tiled with the patterns. We used a machine with Pentium 4 2.80GHz processor and 512MB memory to measure the calculation time.

Figure 3 represents the relation between the size of target tree and the computing time with the condition that the target trees and the pattern trees are labeled like biological molecules. To measure the time in the same condition, we used the same pattern trees in experiments of the graph. The sizes of the target trees shown in the graph are 1599 and 817, one is about twice as large as the other in the size.

Figure 4 shows the relation between sum of 'square of the number of vertices in a pattern

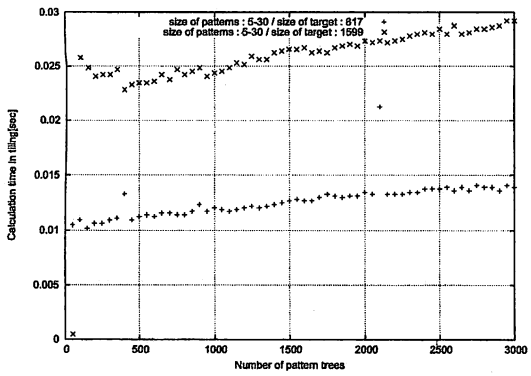

Figure 3: Calculation time of the main part : the influence of the size of target tree

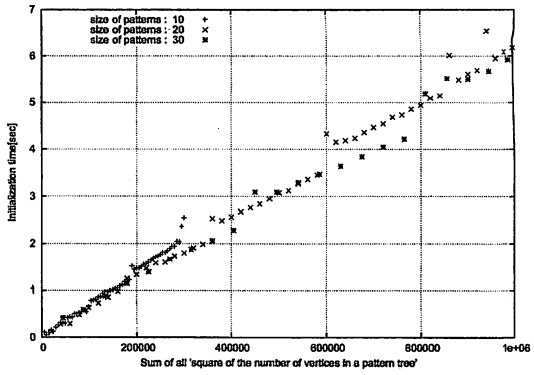

Figure 4: Calculation time of initializing part : the relation between the number of vertices in the pattern trees and the time in initializing

tree' and its initialization time, which are expected to be proportional.

7 Discussion

7.1 Environment

Some notches found in the graphs are thought to be caused by the garbage collection system of Java for the reason that the dot on the right side of the graph should have longer calculation time than the dot on the left side of the graph because they have more pattern trees to match which makes the problem complicated.

7.2 About figures

In Figure 3, tiling a target tree with 1599 vertices takes about twice as long time as the tiling of a target tree with 817 vertices at each size of pattern tree. This result follows the order of algorithm that the time in tiling a tree is linear to the size of a target tree.

Figure 3 does not agree with the time order $O(NM^3)$ we presented. This result can be explained as follows: the number of patterns with n vertices increases rapidly because the used trees were labeled, so the increase of patterns was not effective in increasing the matched patterns.

Figure 4 shows the relation between the sum of all 'square of the number of vertices in a pattern' and the time in initialization. This figure shows that these two data are proportional, which matches the expectation in the method section.

The pre-calculation time is not short and maybe we should use faster algorithm. However, the pre-calculated trees can be used in any tiling problem when the pattern trees are the same, so the pre-calculation should be done only once in most problems even if the target tree changes. This means the more the number of target trees are, the less the ratio of pre-calculation time occupying in the total time becomes.

7.3 More Efficient Order

From Figure 3, the order we proposed seems to be too redundant, and we tried to reduce the order. The order was owe to the time of combining children, so we tried to improve the time.

The improved algorithm is here. The number of possible combined new tree-index is at most $3M$, so we test whether each of the tree represented by the tree-index can be made of not. Each check time is constant so the time of is $O(M)$. We combine children at each vertex, so the order of the tree tiling problem is bounded by $O(MN)$.

8 Conclusion

In the paper, we showed an algorithm for tiling a tree whose computational complexity is $O(NM^3)$ and the implementation showed that the algorithm works at very hi-speed, almost linear to M with practical data which simulates the structure of molecules.

The point limiting the order of the tree tiling algorithm is combination of children. There is probably a method to reduce the computa-

tional complexity by restricting the combination of children. In the approach, the synthesis table we described in our method may be of some help, because the table is deeply related to layered structure of a tree.

We showed that the algorithm can be used to tile the tree, and that it can be used to starting works with the problem of graph tiling problem. By using the method and the possibility of each reaction, we can probably show the best pathway to synthesize a given molecule.

The algorithm for tiling a graph was also shown, which may not be very efficient with our very simple approach, but each calculation time in the problem of tiling a labeled tree was done about 1/100 seconds even if the size of target tree is about 1000. With smaller target trees, we can calculate the tiling results more effectively because the calculation time is proportional to the size of a target tree. By using the point, the graph tiling problem can probably be solved in acceptable time.

The approach with tree-width is also seems to be a promising method in the graph tiling problem, by using the feature that the graph structure of biological molecules are similar to tree. The algorithm described in this paper also uses the feature that the graphs are similar to tree, but the approaches are quite different from each other. By using the feature, there may be more efficient algorithm in tiling a graph.

Tree-width is also a feature of pathway and it is known that only some special enzyme can change the tree-width of a molecule. We can get possible pathways by using the method described in the thesis, but we can choose pathways with higher possibility by using the tree-width data. For example, when tree-width changes but there are no enzyme to bring such change, the possibility of the pathway reduces. Such feature of biological pathway helps to estimate the best pathway, and the feature may be of some help in reducing the computing time, for the reason that the pathway with less possibility is not worth calculating.

In the next stage of our research, we plan to create an algorithm to list up some pathways with high possibility from the target molecular structure, and implement the algorithm to examine the efficiency.

Acknowledgement

We appreciate Tetsuji Kuboyama for giving us informative algorithms in trees.

References

[1] Alfred V. Aho, Mahadevan Ganapathi, and Steven W. K. Tjiang. Code generation using tree matching and dynamic programming. *ACM Trans. on Programming Languages and Systems*, Vol. 11, No. 4, pp. 491–516, 1989.

[2] Robin Cohen. Investigation of processing strategies for the structual analysis of arguments. In *Proceedings of the 19th annual meeting on Association for Computational Linguistics*, pp. 376–377, 1981.

[3] Dorit Dor and Michael Tarsi. Graph decomposition is NPC — a complete proof of Holyer's conjecture. pp. 252–263, 1992.

[4] Philip J. Hatcher and Thomas W. Christopher. High-quality code generation via bottom-up tree pattern matching. In *Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 119–130, 1986.

[5] Christoph M. Hoffman and Michael J. O'Donnell. Pattern matching in trees. Vol. 29, No. 1, pp. 68–95, 1982.

[6] Minoru Kanehisa, Susumu goto, Masahiro Hattori, Kiyoko F. Aoki-Kinoshita, Masumi Itoh, Shuichi Kawashima, Toshiaki Katayama, Michihiro Araki, and Mika Hirakawa. From genomics to chemical genomics: new developments in kegg. *Nucleic Acids Research*, Vol. 34, pp. 354–357, 2006.

[7] Fabrizio Luccio, Antonio Mesa Enriquez, Pablo Olivares Rieumont, and Linda Pagli. Bottom-up subtree isomorphism for unordered labeled trees. Technical report, Università Di Pisa, 2004.

[8] Ron Shamir and Dekel Tsur. Faster subtree isomorphism. In *Israel Symposium on Theory of Computing Systems*, pp. 126–131, 1997.

[9] Atsuko Yamaguchi, Kiyoko F. Aoki, and Hiroshi Mamitsuka. Graph complexity of chemical compounds in biological pathways. *Genome Informatics*, Vol. 14, pp. 376–377, 2003.