Graph Inference from Walks

Maruyama, Osamu
Research Institute of Fundamental Information Science, Kyushu University

出版情報：RIFIS Technical Report. 123, 1995-12. 九州大学理学部附属基礎情報学研究施設
バージョン：
権利関係：

Graph Inference from Walks

by

Osamu Maruyama

A dissertation
submitted to the Department of Information Systems
in the
Interdisciplinary Graduate School of Engineering Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Science

at the

KYUSHU UNIVERSITY

Committee in charge:

Professor	Satoru Miyano, Chair
Professor	Setsuo Arikawa
Professor	Fumihiro Matsuo
Associate Professor	Ayumi Shinohara

December 1995

Abstract

Let G be an edge-colored graph. A walk on G is a path in G containing all edges of G. The trace of a path in G is the sequence of the edge-colors seen in the path. The *graph inference from a walk* is the problem of, given a string x of colors, finding an edge-colored graph with the minimum number of edges which realizes a walk with trace x.

This thesis is devoted to the study of the graph inference from a walk and its modified versions called the *graph inference from partial walks* and the *walk realizability problem*. Here, a partial walk on a graph is a path in the graph. These graph inference problems are closely related to inferring a Markov chain from its output, the identification of a finite state automaton from its input/output behavior, and the problem of constructing an edge-weighted graph from distance data.

For the graph inference from a walk, all results so far known are concerned with degree-bounded graphs as follows: (i) Raghavan gave an $O(n \log n)$-time algorithm to solve the graph inference from a walk for graphs of bounded degree two, which are linear chains and cycles, where n is the length of a given string. (ii) He also showed that the following variant of the graph inference from a walk for graphs of bounded degree k is NP-complete for any $k \geq 3$; Given a string x and a positive integer K, to find a graph of bounded degree k with at most K *nodes* which realizes a walk with trace x.

First, we focus our attention on the structure of trees without any degree-bound constraint. We consider the *tree inference from a walk* that is the problem of finding from a string x the smallest undirected edge-colored tree that realizes a walk with trace x. Our first contribution is an algorithm that solves this problem in time linear in the length of x. This linear-time algorithm is developed by devising a tree rewriting

system with the Church-Rosser property.

The NP-completeness result of (ii) on degree-bounded graphs and our linear-time algorithm on trees naturally raise a question whether the problem for trees of bounded degree k is solvable in polynomial time for $k \geq 3$. Our second contribution is a negative answer to this question. We prove that for any $k \geq 3$, the graph inference from a walk for trees of bounded degree k is NP-complete even if the number of colors is restricted to $k+1$. The number $k+1$ of colors is optimal because when the number of colors is at most k, the problem for trees of bounded degree k is solvable in linear time.

Our third contribution is a thorough refinement of the above NP-completeness result on trees of bounded degree k for $k \geq 3$. We consider fairly simple trees of bounded degree three called (1,1)-caterpillars. A (1,1)-caterpillar is a tree obtained from a linear chain l by attaching at most one other appendage edge to each node of the linear chain l. The shape of a (1,1)-caterpillar is very similar to that of its backbone linear chain. By exploiting a technique of embedding a tree into a (1,1)-caterpillar, we prove that the problem is NP-complete even for this class of simple trees. This result is strong enough to convince us that the class of (1,1)-caterpillars is one of the simplest classes of graphs for which the graph inference from a walk is NP-complete.

Because of the NP-completeness result on trees of bounded degree k ($k \geq 3$), we further investigate the problem from the view point of how nearly optimal solutions can be produced in polynomial-time. We prove that there is no polynomial-time approximation scheme (PTAS) for the problem unless P=NP by showing that the problem is MAX SNP-hard. Under the same assumption that P$\neq$NP, we show that the problem can not be approximated in polynomial time with ratio less than two. In similar ways, we can show that all the above results concerning polynomial-time approximation hold even for (1,1)-caterpillars. From these nonapproximability results, it seems to be still

rather hard to approximate the problem on degree-bounded trees.

Instead of inferring graphs from a single walk, we consider the problem of inferring graphs from a finite number of partial walks. We ask: Given a finite set S of strings of colors, what is the smallest tree T such that, for each string $x \in S$, the tree T realizes some partial walk with trace x. We call the problem the *tree inference from partial walks*. In contrast with the case of a single walk, we prove that the tree inference from partial walks turns to be NP-complete. This problem remains NP-complete even if the number of colors is restricted to three. On the other hand, the problem with at most two colors is solvable in linear time. Moreover, it is shown that there is no polynomial-time approximation scheme for the problem unless P=NP. However we yield a polynomial-time approximation algorithm for the problem. The performance of the algorithm is guaranteed in terms of the *compression* in the trees T constructed, that is, $||S|| - |T|$ where $||S||$ is the total length of the strings in S and $|T|$ is the number of edges in T.

We also consider the problem of inferring a linear chain from partial walks. While the linear chain inference from a single walk, as stated above in (i), is solvable in polynomial time, the problem from partial walks is NP-complete even if the number of colors is three (The problem with at most two colors is solvable in linear time). Although the problem is shown to be unfortunately MAX SNP-hard, we present a polynomial-time approximation algorithm where the approximation performance is analyzed in terms of the length of a linear chain.

Our last contribution is a series of results on the walk realizability problem. The *walk realizability problem* is to decide if, given a graph G in addition to a string x of colors, G realizes a walk with trace x, i.e., if the string x can be produced by walking on all edges of G. We introduce a linear chain decomposition of a graph and show

that the size of the linear chain decomposition is a key to the walk realizability. The problem is shown to be in NLOG if a graph is decomposable into a constant number of linear chains. We next show that the problem is solvable in polynomial time for trees T decomposable into $O(\log m)$ linear chains, where m is the number of edges of T. However, the problem for the class of trees, which are decomposable into $O(m)$ linear chains, turns NP-complete even for (1,1)-caterpillars, which are simple trees of bounded degree three.

Acknowledgments

Foremost I would like to thank my supervisor, Prof. Satoru Miyano. From the very beginning, he boosted my mathematical self-confidence by treating me as a peer. He has collaborated on much of my research. The guts of research he showed me have left a deep impact on me. This exceptional honesty gave me a generous look at the inner working of a truly creative mind. He has greatly influenced my graduate career and I will be forever grateful for these five years I have spent working with him. Knowing Prof. Miyano has been, without any doubt, the high point of my intellectual life.

I enjoyed the numerous meetings I have had with Prof. Setsuo Arikawa, during which he monitored my progress, bolstered my confidence and at the same time provided me with his candid opinion on my work. Every meeting with him was a highly fulfilling experience.

I wish to thank Prof. Fumihiro Matsuo, who has supported my research activity in various ways. His study group has been one of the best sources of information to me.

Prof. Ayumi Shinohara has been a constant source of inspiration and an infinite source of interesting issues to me. He introduced me to the area of graph inference and much of my initial as well as current work have been done with the fruitful discussions with him. He has been very generous with his time, during which I gleaned immense technical knowledge from him. Like Prof. Miyano, Prof. Shinohara's intellectual honesty has always made it possible for me to learn from him.

Prof. Yasuo Kawahara has created a pleasant atmosphere. I wish to express my gratitude to him for valuable comments and criticisms on many occasions.

Thanks are due to Prof. Marek Karpinski at University of Bonn. I had studied there

over one month, which was a wonderful experience for me, and the work of Chapter 4 was started there.

Thanks are also due to Ms. Erika Tateishi, who coauthored one of my papers. It was fantastic for me to share her ideas with me.

I wish I could here express my gratitudes and thanks to all of great and wonderful people who helped, encouraged and advised me in my research of these five years. If I did, the part of acknowledgments would be thicker than the main part. Without anyone, I could not have finished any portion of the work done here.

I greatly thank the Japan Society for the Promotion of Science (JSPS) for its fellowship that supported me for my last two years in graduate school.

Last, I would like to thank my mother, who has been most loving and understanding me especially in my absence at home. This thesis is dedicated to Yasusuke Maruyama (1937-1989).

Contents

Chapter 1

Introduction

Imagine a robot trying to construct a street map of an unfamiliar city by walking along all streets in the city. Typically, each street has the name of itself. However, the names of streets are not necessarily distinct, i.e., some streets have the same name. This situation evidently makes the task of the robot more challenging. Now suppose that the robot has so poor eyesight that the robot can only see the name of the street that the robot is walking on.

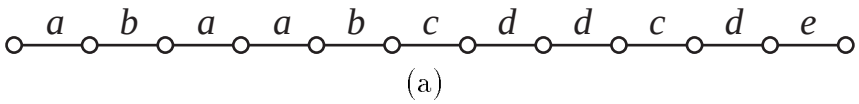
This problem can be formalized as follows: Let G be an edge-colored graph. A walk on G is a path in G containing all edges of G. The trace of a path in G is the sequence of the colors seen in the path.

`Instance:` The trace of a walk on a graph G.

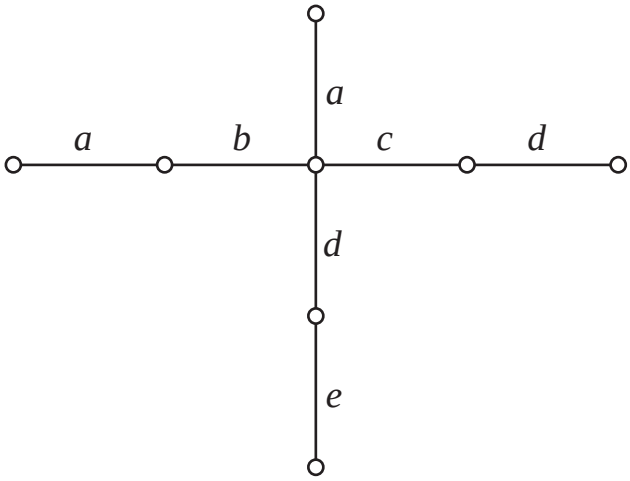
`Output:` The graph G.

For a particular trace of a string x, there may be many graphs that are consistent with x, i.e., realize a walk with trace x. For example, the string $x = abaabbaabcddcdeed$ can be produced as the trace of a walk on any of the graphs in Fig. 1.1.

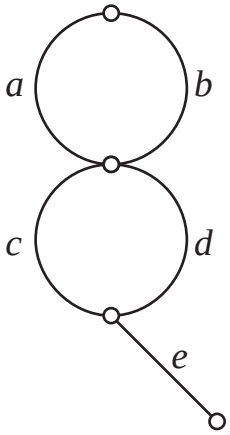
For a specified class C of edge-colored graphs and a string x, we wish to find a graph $G \in C$ realizing a walk with trace x such that the number of edges in G is minimized. Let C be a class of edge-colored graphs. The problem for C can be defined



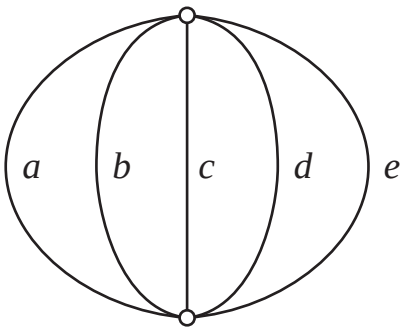
(a)



(b)



(c)



(d)

Figure 1.1: Each graph realizes a walk with trace $x = abaabbaabcddcdeed$.

as follows:

Graph Inference from a Walk for C

`Instance:` A string x over a finite alphabet Σ.

`Output:` A graph in C with the minimum number of edges which realizes a walk with trace x.

In Fig. 1.1, (a) is the optimum solution for linear chains, (b) is the optimum solution for trees, (c) is the optimum solution for graphs of bounded degree four, and (d) is the optimum solution for graphs of bounded degree five. The graph inference from a walk is the main problem we intensively discuss in this thesis. Next, we modify the problem and investigate the *graph inference from partial walks* and the *walk realizability problem*. A partial walk on a graph is defined as a path in the graph. Especially, the graph inference from partial walks provides another method of graph inference where we are given the traces obtained by a finite number of partial walks on a graph. We define the problem as follows:

Graph Inference from Partial Walks for C

`Instance:` A finite set S of strings over a finite alphabet Σ.

`Output:` A graph G in C with the minimum number of edges such that, for each string x in S, the graph G realizes a partial walk with trace x.

These graph inference problems can be classified as the problem of reconstructing a graph G from (partial) structural information of G such as inferring a Markov chain from its output, the identification of a finite state automaton from its input/output behavior, and the problem of constructing an edge-weighted graph from distance data.

Such reconstruction problems of graphs have attracted much attention and have been studied from various viewpoints in Computer Science.

1.1 Importance of the graph inference problems

Rudich [Rud85] considered the problem of inferring a Markov chain from its output. Here, a Markov chain is a finite state machine, each state of which has one arc labeled 0 and one arc labeled 1 with the transition probabilities on the two arcs summing up to one. The output of a Markov chain is a string over 0 and 1 which is produced in infinite transitions from states to states according to the transition probabilities of the Markov chain. Rudich [Rud85] developed an algorithm that for the binary output of a Markov chain, in the limit, reconstructs the underlying Markov chain structure as well as the associated transition probabilities. The graph inference from a walk can be viewed as reconstructing the structure of a Markov chain from its output.

Gold [Gol78] and Angluin [Ang78] considered the problem of identifying the smallest automaton consistent with given input/output behavior. This problem is closely related to the following graph inference problem for directed graphs: We are given two finite sets S_p and S_n of strings called *positive examples* and *negative examples*, respectively, and asked to find an edge-colored directed graph G such that for each positive example $x \in S_p$, the graph G realizes some partial walk with trace x, and that for each negative example $y \in S_n$, any partial walk with trace y cannot be realized in G. If S_n is empty, i.e., there are no negative examples, and, in addition, the number of edges of G is minimized, the above graph inference problem is equivalent to the graph inference from partial walks. For the identification problem of automata, Gold [Gol78] and Angluin [Ang78] showed that the problem is, in general, NP-complete. Moreover, Pitt and Warmuth [PW93] gave an interesting negative result on approximation algorithms

for the problem. They showed that, for any constant k, no polynomial-time algorithm can be guaranteed to find a consistent finite automaton of size at most $(opt)^k$, where opt is the size of the smallest finite automaton consistent with given behavior.

From the viewpoint of learning theory, Angluin [Ang87] established a learning algorithm for finite state automata with membership and equivalence queries. Rivest and Schapire [RS93] presented an extension of Angluin's algorithm to the problem of learning finite state automata from its input/output behavior, even in the absence of a means of resetting the machine to a start state. The absence of the reset mechanism is recovered by inference of a *homing sequence* for an unknown automaton. Rivest and Schapire [RS94] also studied learning algorithms using an alternative representation for finite state automata based on a quantity called the diversity. Freund et al. [FKR+93] gave algorithms for learning finite state automata on the basis of a single long walk in an average case.

When instead of a color a numerical value is assigned into each edge of a graph, it would be also interesting to infer a graph from distance data between nodes. The problem of constructing an edge-weighted graph (in many cases, tree) from distance data is motivated by applications in molecular biology, computational linguistics, and other areas. Therefore various optimization criteria of the problem have been considered. One of the standard problems is described as follows: We are given an $n \times n$ distance matrix $D = (d_{i,j})$, and asked to find an edge-weighted tree T with distinguished leaves $v_1, v_2, \ldots, v_n$ that realizes D, i.e., the cost $d^T_{i,j}$ of the path from v_i to v_j is exactly $d_{i,j}$ for each $1 \leq i \leq j \leq n$, and such that the number of edges is minimized. For this case, polynomial-time algorithms are given in [WSSB77, Hei89, CR89]. However, other optimization criteria for the problem of inferring a graph from distance data were proposed and NP-completeness results were reported [Day87, Win88, FKW95].

For example, in [FKW95] it is shown that for an optimization criterion to find a tree T with the minimized sum of edge-weights such that $d^T_{i,j} \geq d_{i,j}$, there is a constant $\epsilon > 0$ such that no polynomial-time algorithm can approximate the optimal solution within a ratio of n^ϵ unless P=NP. In order to handle actual numeric unreliable data in many applications, interesting approaches were taken in [Fel82, FKW95, KW95]. Especially, Kannan and Warnow [KW95] proposed to construct a tree from partial orders of pairwise distances of distinguished nodes. They showed that the consistency problem is NP-complete in general, but for unweighted binary trees the construction problem can be solvable in polynomial time.

In the model of the graph inference from a walk, the available information to reconstruct an edge-colored graph G is only the trace information of some single walk on G. If we could have access to G in order to get more information of G, it may be easier to reconstruct G. Such a situation has been dealt with in the problem of learning an unknown graph, and some of important results under various conditions were published (e.g., [DP90, BS94, BRS95]). Broadly speaking, in the problem, we must reconstruct an unknown graph G by wandering actively through G and gathering partial information of G we can see there. Bender and Slonim [BS94] showed that two cooperating robots can learn any strongly connected graph with n distinguishable nodes in expected time polynomial in n without membership queries while simultaneously learning a homing sequence. To find homing sequences, the algorithm uses additional information (e.g., indegree, outdegree, or color of nodes). Betke et al. [BRS95] consider to learn a complete map of an unknown environment (i.e., a graph) by a natural search method, called the *piecemeal search*. The "piecemeal constraint" requires that each of the learner's exploration phases must be of limited duration. They devised linear-time piecemeal-search learning algorithms on some kinds of graphs. Deng and Papadim-

itriou [DP90] devised algorithms to explore all edges of an unknown directed, strongly connected graph. They evaluated the algorithm-performance by means of the ratio of the total number of edges traversed divided by the optimal number of traversals.

1.2 Contributions

In this section, we briefly view our contributions to the graph inference from a walk and related works. The first and second parts are concerned with the graph inference problems from a walk and from partial walks, respectively. The third part is on the walk realizability problem.

1.2.1 Graph inference from a walk

A walk on an edge-colored graph G is a path in G which contains all edges of G, and the trace of a path is the string of edge-colors seen in the path. Let C be a class of graphs. The graph inference from a walk for C is defined as the problem of finding, for a string x, a graph in C with the minimum number of edges which realizes a walk for x.

The graph inference problem from a walk was first deeply discussed by Aslam and Rivest [AR90]. Let k be a fixed positive integer. They considered the problem for undirected graphs of bounded degree k, and settled the case for $k = 2$. A graph of bounded degree two can be characterized as either a linear chain or a cycle. They devised $O(n^3)$-time and $O(n^5)$-time algorithms on linear chains and cycles, respectively, where n is the length of a given string. In sequence, Raghavan [Rag94] developed a faster algorithm on graphs of bounded degree two running in $O(n \log n)$ time. Note that the graph inference problem is trivial for *directed* graphs of bounded degree two. On the other hand, Raghavan [Rag94] gave a hardness result on a variant of the problem for graphs of bounded degree k ($k \geq 3$) that is, given a string x and a positive integer

K, to find a graph of bounded degree k with at most K *nodes* which realizes a walk with trace x is NP-complete for all $k \geq 3$. However, it has not been known if the original problem of the graph inference from a walk for graphs of bounded degree k is NP-complete yet.

The above results, which are about graphs with a bounded degree, are all for the graph inference from a walk. For this problem, we first settle the case for undirected trees without any degree-bound constraint, called the *tree inference from a walk*. By inquiring into some structural properties of such trees, we develop an $O(n)$-time algorithm to solve the problem, i.e., to produce, from a string x, the smallest tree realizing a walk with trace x, where n is the length of x. Because the structure of trees is more complex than that of graphs of bounded degree two, it is quite remarkable that our algorithm on trees runs faster than Raghavan's algorithm on graphs of bounded degree two, which is currently the fastest one on such graphs. We define a tree rewriting system which is based on a binary relation on the set of undirected edge-colored trees. This tree rewriting system is a key idea of the linear-time algorithm. We show that the tree rewriting system satisfies the Church-Rosser property. In addition, when rewriting a tree T by the system, the resulting tree is strictly smaller than T. The combination of these properties is essentially a reason to allow the tree inference from a walk to have a polynomial-time algorithm.

Second, we ask if the graph inference from a walk for trees of bounded degree three is solvable in polynomial time. This question is inspired by our linear-time algorithm on trees and the NP-completeness result by Raghavan on graphs of bounded degree three. When the answer to the question is affirmative, we can say that the structure of trees is a stronger factor in the complexity of the graph inference problem than bounding the maximum degree of graphs. Unfortunately, the problem is settled negatively as follows:

We prove that the graph inference from a walk for trees of bounded degree three is NP-complete. However, this result does not give us any idea of whether or not the problem with a constant number of colors is still NP-complete because the reduction we give to show the NP-hardness of the problem does not bound the alphabet size of colors by a constant. By exploiting a technique of reducing the number of colors on trees, we then show that the problem for trees of bounded degree three remains NP-complete even if the number of colors is restricted to four. This result can be generalized in the following way: For any $k \geq 3$, the problem for trees of bounded degree k is NP-complete even when the number of colors is $k+1$. It can be said that the number $k+1$ of colors is optimal since the problem for trees of bounded degree k with at most k colors is solved by the linear-time algorithm that we construct for the tree inference from a walk.

Third, we consider very simple trees with degree-bound three, called (1,1)-caterpillars. Let α and β be positive integers. An (α, β)-caterpillar is defined as a tree constructed from a linear chain, called the *backbone*, by attaching at most α other appendage linear chains with length at most β, called *hairs*, to each node of the backbone. Obviously, an (α, β)-caterpillar is a tree of bounded degree $\alpha + 2$. Especially, a (1,1)-caterpillar is so simple that it looks like a linear chain because each node of the backbone, which is a linear chain, has at most one hair with length one. We prove that the graph inference

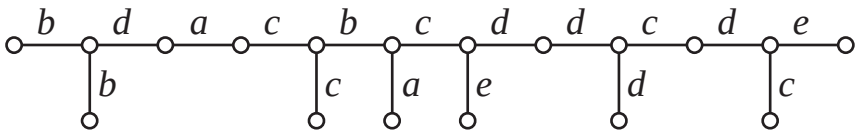


Figure 1.2: A (1,1)-caterpillar.

from a walk for (α, β)- caterpillars is NP-complete for each $\alpha \geq 1$ and $\beta \geq 1$. It is very interesting that, even for (1,1)-caterpillars, the NP-completeness result still stands

up. As compared with the result that the linear chain inference from a walk is solvable in polynomial time, which was given by Aslam and Rivest [AR90] and Raghavan [Rag94], it can be said that allowing each node of a linear chain to have at most one appendage edge makes the graph inference from a walk intractable. Here, let b be the length of the backbone of a fixed arbitrary (1,1)-caterpillar. Our reduction showing the NP-hardness for (1,1)-caterpillars does not work for the restricted case that the total number of hairs is at most $O(\log b)$. From this observation, the graph inference from a walk for (1,1)-caterpillars with $O(\log b)$ hairs seems to be solvable in polynomial time although it has not been proved yet. We believe that the class of (1,1)-caterpillars can be regarded as one of the simplest classes of graphs for which the graph inference from a walk is NP-complete.

Under the natural assumption P$\neq$NP, the graph inference from a walk for trees of bounded degree k ($k \geq 3$) does not have any polynomial-time algorithms because of its NP-completeness. We wish to develop polynomial-time approximation algorithms for the problem, but yet we give some hardness results on polynomial-time approximation of the problem. We prove that the problem has no polynomial-time approximation scheme (PTAS) unless P=NP. We say that a problem has a PTAS if for every fixed $\epsilon > 0$, the problem can be approximated with ratio $1 + \epsilon$ in polynomial time. More precisely speaking, we prove that the problem is MAX SNP-hard. The complexity class MAX SNP is a class of optimization problems which is syntactically defined by Papadimitriou and Yannakakis [PY91], together with a concept of reduction, called L-reduction, that preserves approximation schemes. By the result due to Arora et al. [ALM$^+$92], a MAX SNP-hard problem has no PTAS unless P=NP. Moreover, we give another nonapproximability result of the graph inference from a walk for trees of bounded degree k for $k \geq 3$. We show that there is no polynomial-time approximation

algorithm with ratio less than two unless P=NP. This is a trivial consequence of the reduction showing the NP-hardness.

Surprisingly, all the above results concerning polynomial-time approximation are shown to be true even for (1,1)-caterpillars. These nonapproximability results assert that it is even hard to approximately solve the graph inference from a walk for degree-bounded trees in polynomial time.

All the above our results of the graph inference from a walk, which are provided in Chapter 2, are published in [MM92, MM95a].

1.2.2 Graph inference from partial walks

In Chapter 3, we discuss the problem of inferring a graph from a finite number of partial walks instead of a single walk. A partial walk on a graph G is defined as a path in G. Let C be a class of graphs. The *graph inference from partial walks for* C is the problem of, given a finite set S of strings over an finite alphabet, finding a graph with the minimum number of edges which realizes some partial walk with trace x for each string $x \in S$. Obviously, this problem is an extended version of the graph inference from a walk because when $|S| = 1$, i.e., the number of strings in S is exactly one, these problems are equivalent.

First, we consider the problem for undirected trees, called the *tree inference from partial walks*. As previously stated, one can infer a tree from a single walk in linear time. However, we show that the tree inference from partial walks is NP-complete. From this result, we can say that the trace of a single walk on a tree T has more information of T for reconstructing T than the traces of a finite number of partial walks on T. This observation would be true for arbitrary graphs. The tree inference from partial walks is still NP-complete even if the alphabet size of colors is three. The number three of colors is optimal because if the number of colors is at most two, the

problem is solvable in time linear in the total length of strings in S by applying our linear-time algorithm for the tree inference from a walk.

We also discuss how to approximately solve the problem in polynomial time. We prove a negative result that the problem has no PTAS unless P=NP by showing the MAX SNP-hardness of the problem. On the other hand, fortunately, the problem is shown to be approximately solvable in polynomial time. The performance of our approximation algorithm is guaranteed in terms of the *compression* in the trees T constructed, that is, $||S|| - |T|$ where $||S||$ is the total length of the strings in S and $|T|$ is the number of edges in T. We prove that there is a polynomial-time approximation algorithm to find, from a finite set S of strings, a tree T realizing some partial walk with trace x for each string $x \in S$ such that $C \geq C_m/(|S| - 1)$, where C is the compression in T and C_m is the maximum compression for S. The algorithm is constructed by employing the polynomial-time approximation algorithm for the minimum common supertree problem [YM93].

We also consider the problem for undirected linear chains, which is referred to the *linear chain inference from partial walks*. This problem is closely related to the shortest common superstring problem [GMS80] because the *directed* version of our problem, i.e., the graph inference from partial walks for directed linear chains is intrinsically equivalent to the shortest common superstring problem. This problem is one of the most basic and important problems in Computer Science and has plentiful results (e.g., [GMS80, TU88, Tur89, Li90, Ukk90, BJL+94, TY93, Mid94, Gus94, JL94, KPS94, CGPR94, AS94, JT95]). The minimum common supertree problem [YM93], whose polynomial-time approximation algorithm is employed in such an algorithm for the tree inference from partial walks, is also related to the shortest common superstring because the minimum common supertree problem can be viewed as an extension of

the shortest common superstring. In the same way as the case of trees, we prove that the linear chain inference from partial walks is NP-complete, while the linear chain inference from a single walk is solvable in polynomial time [AR90, Rag94]. This result holds even if the alphabet size is three. The number three is also optimal because the problem with at most two colors is solvable in linear time. To show the NP-hardness of the problem, we give a reduction from the shortest common superstring problem. It is interesting that although the shortest common superstring problem is still NP-complete even if the alphabet size is restricted to two [GMS80], yet the linear chain inference from partial walks is solvable in linear time under the same constraint on the alphabet size.

We show that the linear chain inference from partial walks has no PTAS unless P=NP, which is a direct consequence of the MAX SNP-hardness. But, we can give polynomial-time approximation algorithms that employ approximation algorithms for the problem of shortest superstrings with flipping [JLD92]. A superstring of a set S of strings with flipping is a string s which contains, for each string $x \in S$, either x or the reversal of x as a substring of s. The performances of our approximation algorithms are analyzed in terms of the length of a linear chain l and the compression in l. Let S be a finite set of strings. We show that one approximation algorithm satisfies that $C \geq C_m/2$, where C is the compression in the produced linear chain l and C_m is the maximum compression for S, and that another algorithm finds a linear chain with at most $3 \cdot opt(S)$ edges, where $opt(S)$ is the number of edges in the shortest linear chain realizing all partial walks for S.

The results in Chapter 3 are based on [MM92].

1.2.3 Walk realizability problem

Imagine a guard with a security agency having a sideline. The task which the agency takes upon the guard is to patrol all streets of a city by walking along the streets. His sideline consists of many kinds of actions such as withdrawing his savings from a bank, visiting a company and buying something at shops. He knows a sequence of actions which makes a profit. However, he can not always take any actions anywhere. For example, when withdrawing money, there must be a bank there. Therefore, it is needed for him to test if the profitable sequence of actions can be realized while performing his duty as a guard, i.e., walking on all streets of the city.

We formalize and discuss the above problem in Chapter 4. As a model of a city, we consider an edge-colored graph. Each color itself corresponds to an action of the guard's sideline. Now suppose that when the guard traverses an edge e, the guard must take the action assigned to e. Let C be a class of graphs. Our problem is defined as follows:

Walk Realizability Problem for C

`Instance:` A string x over a finite alphabet Σ and a graph G.

`Question:` Does G realize a walk with trace x?

The walk realizability problem is closely related to the graph inference from a walk. When we regard a string x as a series of actions and a graph as a constraint on actions, the graph inference problem is to design the smallest constraint graph which accepts the series of actions. On the other hand, the walk realizability problem is to test if a constraint graph can be swept by a given series of actions.

Another related problem is the partial walk realizability problem, which is to ask, given a graph G and a string x, if G realizes a partial walk whose sequence of edge-

colors is x, where a partial walk on a graph G is a path in G. It is easily seen that the partial walk realizability problem is complete for NLOG [Sav70]. These two problems differ on the point that all edges must be traversed with a specified sequence of edge-colors for the walk realizability problem while the partial walk realizability problem does not require this condition.

We introduce a notion of the linear chain decomposition of a graph and show that the size of the linear chain decomposition is a key to the complexity of the walk realizability problem. Our first observation on the walk realizability problem is that it is solvable nondeterministically in log-space, i.e., in the class NLOG, if a graph is undirected and decomposable into a constant number of linear chains. This result also holds for directed graphs.

Furthermore, we show that the walk realizability problem is solvable in polynomial time for undirected trees T decomposable into $O(\log m)$ linear chains, where m is the number of edges in T. Unfortunately, it does not seem that our algorithm on such trees is applicable to general graphs decomposable into $O(\log m)$ linear chains.

In contrast, we prove that the walk realizability problem for the class of undirected trees, which are decomposable into $O(m)$ linear chains, turns NP-complete. We refine this result by providing further observations on the problem. We say that a node is proper if all of the edges incident to the node have mutually distinct colors. We can easily see that if all nodes in a given tree are proper, the walk realizability problem is solvable in polynomial time by applying the linear-time algorithm we develop for the tree inference from a walk. However, it is quite remarkable that the walk realizability problem becomes NP-complete for the class of trees having exactly one node being not proper. Obviously, if trees are directed, the walk realizability problem is solvable in polynomial time.

A (1,1)-caterpillar is an undirected tree obtained from a linear chain l by attaching at most one hair to each node of the linear chain l. For such simple trees like linear chains, we show that the walk realizability problem remains NP-complete though the problem for linear chains is in NLOG. It is really interesting that the NP-completeness result still holds even if the number of colors is two. When any (1,1)-caterpillar is monotone, i.e., the number of colors is one, the problem is trivial. By an easy modification of a reduction showing that the walk realizability problem for (1,1)-caterpillars is NP-hard, the problem for the class of meshes can be shown NP-complete, where a mesh is an undirected graph with the node set $V = \{v_{i,j} \mid 1 \leq i, j \leq n\}$ and the edge set $E = \{\{v_{i,j}, v_{i,j+1}\} \mid 1 \leq i \leq n, 1 \leq j \leq n-1\} \cup \{\{v_{i,j}, v_{i+1,j}\} \mid 1 \leq i \leq n-1, 1 \leq j \leq n\}$.

The results of this subsection are based on [MM95b].

Chapter 2

Inferring a Graph from a Walk

In this chapter we consider the graph inference from a walk. After reviewing some important results given by Aslam and Rivest [AR90] and Raghavan [Rag94], we present our results on the problem. First, we give necessary definitions and notations used throughout this thesis.

2.1 Preliminaries

Let Σ be a finite alphabet. The set of all strings over Σ is denoted by Σ^*. The reversal of a string x is written as x^R. The length of a string x is denoted by $|x|$. To represent the cardinality of a set S, we use the same notation $|S|$. The concatenation of strings x and y is written as $x \cdot y$, or simply xy. For strings $x_1, \ldots, x_n$, $\prod_{i=1}^{n} x_i$ denotes $x_1 x_2 \cdots x_n$. Especially, if $x = x_1 = \cdots = x_n$, we denote $\prod_{i=1}^{n} x_i$ by $(x)^n$, or simply x^n. For convenience, x^0 is defined as the empty string ε.

A *color* is a symbol in Σ. We will consider edge-colored graphs $G = (V, E, c)$, where $c : E \to \Sigma$ is called the *edge-coloring of* G. Hereafter a graph means an edge-colored graph without any notice. Let G be a graph. A *partial walk* on G is a path in G. Let $v_0, v_1, \ldots, v_n$ be the sequence of nodes of a partial walk w on G. We denote v_i by $w[i]$ for $0 \leq i \leq n$. We call $w[0]$ and $w[n]$ the *start* and *end* nodes of the partial walk w, respectively. A partial walk w is said to be *closed* if the start node of w coincides

with the end node. Let $e_1, \ldots, e_n$ be the sequence of edges of a partial walk w on a graph $G = (V, E, c)$. The *trace* of w is $\prod_{i=1}^{n} c(e_i)$. Let x be a string in Σ^*. If w is a partial walk with trace x, w is called a *partial walk for* x. For a graph G, we say that G *realizes* a partial walk for x if there is a partial walk for x on G. Let $x = \prod_{i=1}^{n} s_i$ with s_i in Σ. For a partial walk w for x, the *subwalk* of w for substring $\prod_{i=j}^{k} s_i$ with $1 \leq j \leq k \leq n$ is the partial walk with the node sequence of $w[j-1], w[j], \cdots, w[k]$. If a partial walk on G contains all edges of G, it is called a *walk on* G.

A node v of a graph is said to be *proper* if the colors assigned to the edges incident to v are mutually distinct. A graph G is said to be *proper* if all nodes of G are proper. For graphs G and G', if G and G' are isomorphic including edge labels, we identify G with G' without any notice.

A *cycle* is an undirected graph $l = (V, E, c)$ with $V = \{v_1, v_2, \ldots, v_m\}$ and $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{m-1}, v_m\}, \{v_m, v_1\}\}$. A *linear chain* is an undirected graph $l = (V, E, c)$ with $V = \{v_1, v_2, \ldots, v_m\}$ and $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{m-1}, v_m\}\}$. The *label* of l is $\prod_{i=1}^{m-1} c(\{v_i, v_{i+1}\})$. A *directed linear chain* is $l = (V, E, c)$ where $V = \{v_1, v_2, \ldots, v_m\}$ and $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{m-1}, v_m)\}$. The label of the directed linear chain l is defined in the same way as that of a linear chain.

Let C be a class of graphs. The *graph inference from a walk for* C is defined as follows:

Graph Inference from a Walk for C

`Instance`: A string x over a finite alphabet Σ.

`Output`: A graph in C with the minimum number of edges which realizes a walk for x.

2.2 Degree-bounded graphs

If C is the collection of all graphs, i.e., there is no constraint on graphs in C, the solution of the graph inference from a walk for C is immediate. For example, given a string x of the colors a, b, c, d, the graph in Fig. 2.1 is the smallest graph realizing a walk for x.

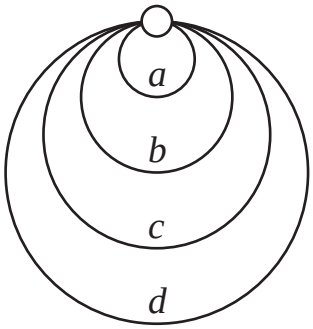


Figure 2.1: When given a string x of n colors, any graph realizing a walk for x has at least n edges.

The graph in Fig. 2.1 is allowed to contain self-loops. Even if any self-loops are not allowed, the solution is still clear (see Fig. 2.2).

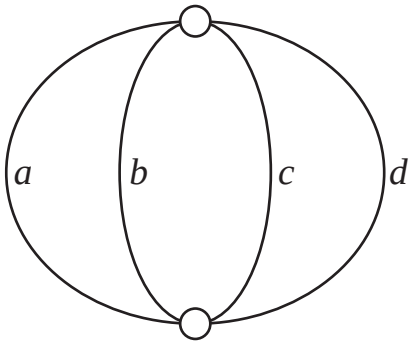


Figure 2.2:

Evidently, the maximum degrees of both the graphs in Fig. 2.1 and 2.2 depend on the alphabet size, i.e., the number of colors of the string x. The following question arises

out of the above fact on degrees: When the maximum degree of graphs is bounded by a constant, say k, is it easy to find from a string x the smallest graph of bounded degree k realizing a walk with trace x? For the question, Aslam and Rivest [AR90] and Raghavan [Rag94] gave some important results.

Aslam and Rivest [AR90] developed polynomial-time algorithm for the graph inference from a walk for undirected graphs of bounded degree two. Such a graph can be classified as a linear chain or a cycle. Their algorithm runs in $O(n^3)$ time and $O(n^5)$ time for linear chains and cycles, respectively, where n is the length of a given string. A faster algorithm on graphs of bounded degree two is given by Raghavan [Rag94]:

Theorem 2.1 (Raghavan [Rag94]) The graph inference from a walk for graphs of bounded degree two is solvable in $O(n \log n)$ time, where n is the length of a given string.

Notice that if graphs of bounded degree two are *directed*, the graph inference from a walk turns to be trivial.

Let $k \geq 3$ be a fixed integer. Raghavan solved a problem similar to the graph inference from a walk for graphs of bounded degree k, which was asked to be tractable or NP-complete by Aslam and Rivest [AR90].

Theorem 2.2 (Raghavan [Rag94]) Let $k \geq 3$. Given a string x and a positive integer K, the problem of deciding if there exists a graph of bounded degree k with at most K nodes which realizes a walk for x is NP-complete.

2.3 Trees without any degree-bound constraint

As we have seen in Section 2.2, all of the previous works are connected with degree-bounded graphs. In this section, we consider the graph inference from a walk for

undirected trees without any degree-bound constraint. We call the problem the *tree inference from a walk*.

Example 2.1 *Given the string $x = daabbcacbbcbbabcaccbbabbcdbbc$, what is the smallest tree realizing a walk for x? That is isomorphic to the tree in Fig. 2.3.*

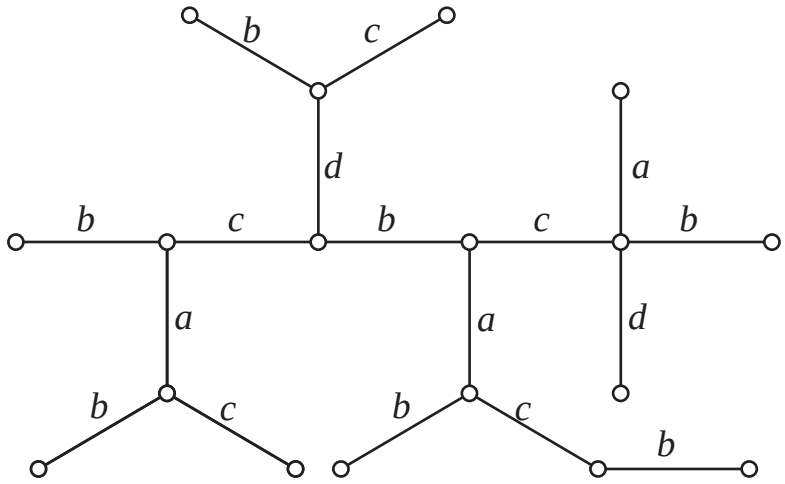


Figure 2.3:

We prove the following theorem:

Theorem 2.3 *The tree inference from a walk is solvable in $O(n)$ time, where n is the length of a given string.*

We here construct a tree rewriting system and then give a proof of Theorem 2.3. In fact, we have another simple proof of the theorem, which was given in [MM92]. The reason why the former has been selected is as follows: The tree rewriting system provides us a clear understanding of the problem as well as some helpful properties concerning the graph inference from a walk and the graph inference from partial walks, which will be used in the later sections and chapters.

Let S be an arbitrary set. Let $\rightarrow_X$ be a binary relation on S, i.e., a subset of $S \times S$. We denote $(x, y) \in \rightarrow_X$ by $x \rightarrow_X y$. The composition of two binary relation is defined in a usual way. The reflexive and transitive closure of $\rightarrow_X$ is denoted by $\stackrel{*}{\rightarrow}_X$ and the transitive closure of $\rightarrow_X$ is denoted by $\stackrel{+}{\rightarrow}_X$. An element $x \in S$ is called *irreducible* if there is no $y \in S$ with $x \rightarrow_X y$. If $x \stackrel{*}{\rightarrow}_X y$ and y is irreducible, then we say that y is an $\rightarrow_X$-*normal form of* x, denoted by $\widehat{X}(x)$. A binary relation $\rightarrow_X$ is *noetherian* if there is no infinite sequence $x_1 \rightarrow_X x_2 \rightarrow_X \cdots \rightarrow_X x_n \rightarrow_X \cdots$. A binary relation $\rightarrow_X$ is *confluent* if, for any $x, y_0, y_1 \in S$, $x \stackrel{*}{\rightarrow}_X y_0$ and $x \stackrel{*}{\rightarrow}_X y_1$ imply that there is $z \in S$ with $y_0 \stackrel{*}{\rightarrow}_X z$ and $y_1 \stackrel{*}{\rightarrow}_X z$. A binary relation $\rightarrow_X$ is *locally confluent* if, for any $x, y_0, y_1 \in S$, $x \rightarrow_X y_0$ and $x \rightarrow_X y_1$ imply that there is $z \in S$ with $y_0 \stackrel{*}{\rightarrow}_X z$ and $y_1 \stackrel{*}{\rightarrow}_X z$. Let f be a function from S to $\mathcal{N}$, where $\mathcal{N}$ is the set of nonnegative integers. A binary relation $\rightarrow_X$ is *strictly decreasing* with respect to the function f if, for any $x, y \in S$, $x \rightarrow_X y$ implies $f(x) > f(y)$.

The following results are well known.

Proposition 2.1 (Huet [Hue80])

(1) For a noetherian relation, it is confluent if and only if it is locally confluent.

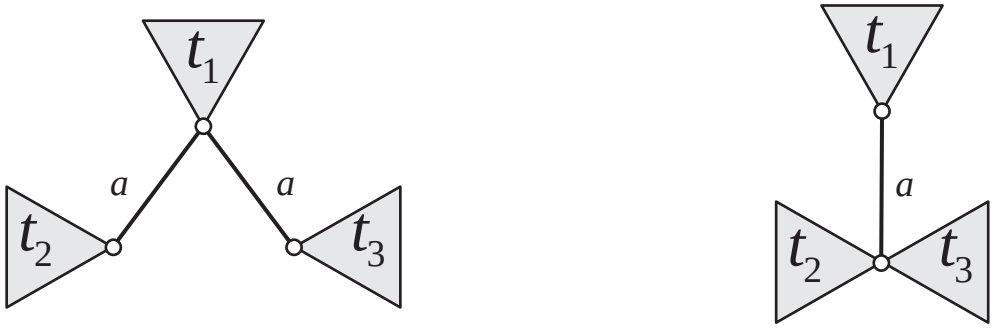
(2) Let $\rightarrow_X$ be a confluent binary relation on a set S. Then, for each element $x \in S$, an $\rightarrow_X$-normal form of x is unique if it exists.

Proposition 2.2 (Aslam and Rivest [AR90]) If a binary relation $\rightarrow_X$ on S is confluent and strictly decreasing with respect to a function $f : S \rightarrow \mathcal{N}$, then $x \stackrel{*}{\rightarrow}_X y$ implies $f(\widehat{X}(x)) \leq f(y)$ for any x and $y \in S$.

We here define a binary relation on the set of trees.

Definition 2.1 *Let $T_1 = (V, E, c)$ be a tree which includes adjacent edges $e_1 = \{v_1, v_2\}$ and $e_2 = \{v_2, v_3\}$ with $c(e_1) = c(e_2)$ (see Fig. 2.4 (a)). Let T_2 be the tree obtained*

from T_1 by identifying v_3 with v_1 together with the adjacent edges e_1 and e_2 (see Fig. 2.4 (b)). For such trees T_1 and T_2, we say that T_2 is an edge-folding of T_1. The binary relation $\rightarrow_F$ on the set of trees is defined as the set of pairs (T_1, T_2) such that T_2 is an edge-folding of T_1.



(a) T_1 (b) T_2

Figure 2.4: t_1, t_2 and t_3 in (a) and (b) are arbitrary trees and a is an arbitrary color.

Trivially, $\rightarrow_F$ is strictly decreasing with respect to the number of edges. Hereafter, for a string x, we denote a linear chain of label x by l_x. The following lemma would be trivial:

Lemma 2.1 *Let x be a string and T be a tree. The tree T realizes a walk for x if and only if $l_x \xrightarrow{*}_F T$.*

Notice that $\rightarrow_F$ is noetherian. For proving Lemma 2.2, it is sufficient to show that $\rightarrow_F$ is locally confluent by Proposition 2.1.

Lemma 2.2 *The binary relation $\rightarrow_F$ is confluent.*

We can see that for each string x, an $\rightarrow_F$-normal form of x uniquely exists by Proposition 2.1 (2).

Lemma 2.3 *Let x be a string. The $\rightarrow_F$-normal form of x, i.e., $\widehat{F}(l_x)$, is the smallest tree realizing a walk for x.*

Proof. Since $l_x \stackrel{*}{\rightarrow}_F \widehat{F}(l_x)$, it follows from Lemma 2.1 that $\widehat{F}(l_x)$ also realizes a walk for x. Let T be a tree which realizes a walk for x. By Lemma 2.1, the linear chain l_x satisfies $l_x \stackrel{*}{\rightarrow}_F T$. Since the binary relation $\rightarrow_F$ is confluent (Lemma 2.2) and strictly decreasing with respect to the number of edges, we see from Proposition 2.2 that $\widehat{F}(l_x)$ has edges less than or equal to T. Hence $\widehat{F}(l_x)$ is the smallest tree that realizes a walk w for x. □

We can easily see the fact that, for any tree T realizing a walk for a string x, the tree T is proper if and only if T is isomorphic to $\widehat{F}(l_x)$ because of Lemma 2.1 and the definition of the binary relation $\rightarrow_F$. By this fact and Lemma 2.3, the following lemma is clear:

Lemma 2.4 *Let x be a string. The smallest tree realizing a walk for x is unique and proper.*

Evidently, the algorithm EDGE-FOLD produces a proper tree T, which is represented as an array indexed on the vertices and the colors. It is clear that T realizes a walk for a given string. Thus, we have the following:

Lemma 2.5 *The algorithm* EDGE-FOLD *produces from a string x the smallest tree realizing a walk for x.*

The number of steps executed by every iteration of the loop of EDGE-FOLD is bounded by a constant. Thus EDGE-FOLD runs in $O(n)$ time.

$/* \ x = x_1 \cdots x_n \ (x_i \in \Sigma) \ */$

begin

$u := 1$; $v := 1$;

for $i := 1$ **to** n

if $T[u, x_i] = 0$ **then**

$v := v + 1$;

$T[u, x_i] = v$; $T[v, x_i] := u$; $/* \ \{u, v\}$ is an edge labeled $x_i \ */$

$u := v$;

else $u := T[u, x_i]$

endif

end;

return T

end;

Algorithm : EDGE-FOLD

2.4 Trees of bounded degree three

In this section, we consider the graph inference from a walk for trees with degree-bound. This problem arises in the light of the NP-completeness result on graphs of bounded degree k for $k \geq 3$ (Theorem 2.2) and our linear-time algorithm on trees (Theorem 2.3). Our question is whether or not the graph inference from a walk for trees of bounded degree three is tractable. We answer the question negatively and moreover show that it is still hard to solve the problem approximately.

Here, we give the decision version of the graph inference from a walk. Let C be a

class of graphs. The decision problem for C can be described as follows:

`Instance`: A string x over a finite alphabet Σ and a positive integer K.

`Question`: Is there a graph in C with at most K edges which realizes a walk for x?

We first address the graph inference problem for the class of trees of bounded degree three. After that, we restrict such trees into simpler trees of bounded degree three, called (1,1)-caterpillars.

2.4.1 General results

We shall see an example of the decision problem on trees of bounded degree three.

Example 2.2 *Let $K = 5$. For a string $x = abbccd$, the graph in Fig. 2.5 is a tree of bounded degree three with K edges which realizes a walk for x. On the other hand, for $y = abbccddccaad$, there is no tree of bounded degree three with at most K edges which realizes a walk for y.*

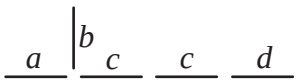


Figure 2.5: The graphical representation of a node, $\circ$, is omitted.

Our main object of this subsection is to prove the following theorem:

Theorem 2.4 *The graph inference from a walk for the class of undirected trees of bounded degree three is NP-complete.*

The following fact, which is trivial, is useful when we construct a tree realizing a closed walk:

Fact 2.1 *Suppose that a tree t realizes a closed walk w for a string x. If a symbol a occurs in x exactly twice, then t includes exactly one edge labeled a and the edge is traversed exactly twice by w in distinct directions.*

Proof of Theorem 2.4. Obviously, the problem is in NP. To show that the problem is NP-hard, we give a reduction from the vertex cover problem [GJ79], which is to decide if, given a graph $G = (V, E)$ and a positive integer K, there is a vertex cover of size at most K for G, that is, a subset $U \subseteq V$ with $|U| \leq K$ such that for each edge $\{u, v\} \in E$ at least one of u and v belongs to U. We denote this problem by VERTEX COVER. Especially, when graphs are restricted to graphs of bounded degree k without any self-loop, VERTEX COVER is denoted by k-DEGREE VERTEX COVER. It is known that 3-DEGREE VERTEX COVER remains NP-complete [GJ79]. Let $K \leq |V|$ be a positive integer and $G = (V, E)$ be a graph of bounded degree three without any self-loop, where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We will define an alphabet Σ and construct a string x over Σ and a positive integer K' such that there is a tree of bounded degree three with at most K' edges which realizes a walk for x if and only if G has a vertex cover of size K or less. Hereafter a tree means a tree of bounded degree three. The alphabet Σ is defined as follows:

$$\begin{aligned}\Sigma \;=\; & \{a_1, a_2\} \cup \{g_1, g_2, g_3\} \cup \{h_1, h_2\} \cup \{r_1, r_2\} \cup \{s_1, s_2\} \\ & \cup \{0, 1, \#\} \\ & \cup \{\alpha_i, \overline{\alpha}_i \mid i = 1, 2, 3\} \\ & \cup \{\beta_{i,j} \mid i = 1, \ldots, n \text{ and } j = 1, 2, 3\} \\ & \cup V.\end{aligned}$$

In order to define the string x, we introduce the four kinds of strings as follows:

(1) W and W': Let $\mu = 53n + m + 2$, where $n = |V|$ and $m = |E|$. We define

$$W \;=\; \prod_{i=1}^{\mu} b_i,$$

$$W' \;=\; \prod_{i=1}^{\mu}(\#\#b_i),$$

where $b_i = 0$ if i is even and $b_i = 1$ if i is odd. We will see that the string W occurs in x many times and W' once. Trivially, only a linear chain of label W realizes a walk for W. It is clear from Lemma 2.4 that the tree T_W in Fig. 2.6 is the smallest tree without any degree bound that realizes a walk for W'. These graphs have μ and 2μ edges, respectively. Later, it will be shown that any tree realizing a walk for x with at most K' edges must have exactly one subgraph isomorphic to T_W and all partial walks for W and W' must be realized in this unique T_W.

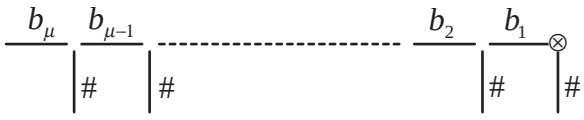


Figure 2.6: T_W

(2) $X_{i,j}$ with $i \in \{1,\ldots,n\}$ and $j \in \{1,2,3\}$: First let

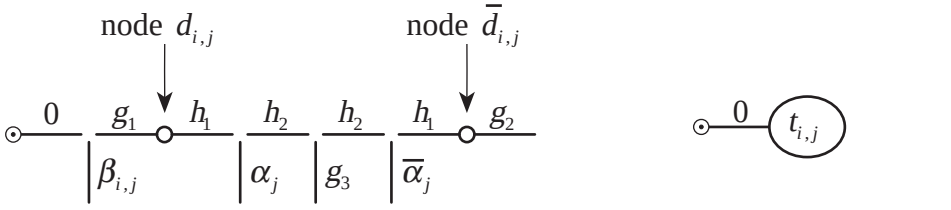
$$x_{i,j} \;=\; 0g_1h_1h_2h_2h_1g_2g_2h_1\ \overline{\alpha}_j\overline{\alpha}_j\ h_2g_3g_3h_2\ \alpha_j\alpha_j\ h_1g_1\ \beta_{i,j}\beta_{i,j}\ 0,$$

for $i \in \{1,\ldots,n\}$ and $j \in \{1,2,3\}$. We define $X_{i,j}$ by using $x_{i,j}$ as

$$\begin{aligned} X_{i,j} \;=\; & (a_1\#\#a_1)^i\ 1\ (r_1r_2r_2r_1)^{\lfloor(3-j)/2\rfloor}\ (a_2\#\#a_2)^j \\ & x_{i,j}\ (s_1s_1)^{\lfloor j/3\rfloor}\ (a_2a_2)^j\ 1\ (s_2s_2)^{\lfloor i/n\rfloor}\ (a_1a_1)^i. \end{aligned}$$

Note that $\lfloor(3-j)/2\rfloor$ is equal to 1 if $j = 1$ and 0 otherwise. Consider the tree $t_{i,j}$ in Fig. 2.7. Then it is clear from Fact 2.1 that if a walk w for $X_{i,j}$ is closed then the subwalk of w for the substring $x_{i,j}$ of $X_{i,j}$ must be a closed walk in the tree $t_{i,j}$. We say that a node of a tree is *free* if its degree is less than three. A prefix of x is a concatenation of the strings $X_{1,1}, X_{1,2}, X_{1,3}, X_{2,1}, \ldots, X_{n,3}$, W'

and W's, which is called *TEMPLATE*. It will be seen that a tree realizing a walk for *TEMPLATE* with at most K' edges is unique up to isomorphism.



(a) $t_{i,j}$ (b) abbreviation for $t_{i,j}$

Figure 2.7: The tree $t_{i,j}$ has two free internal nodes, which are denoted by $d_{i,j}$ and $\overline{d}_{i,j}$. The node $\odot$ is the start and end node of the closed walk for $x_{i,j}$.

(3) $\langle v,j\rangle$ with $v \in V$ and $j \in \{1,2,3\}$: We define

$$\langle v,j\rangle \;=\; (a_1a_1)^n\ 1r_1\ vv\ r_1\ (a_2a_2)^j\ 0g_1\ vv\ h_1\ \alpha_j\alpha_j\ h_1g_10\ (a_2a_2)^j\ 1\ (a_1a_1)^n.$$

Notice that the string vv appears in $\langle v,j\rangle$ twice. The tree in Fig. 2.8 realizes a closed walk for $\langle v,j\rangle$ starting and ending at node $\otimes$.

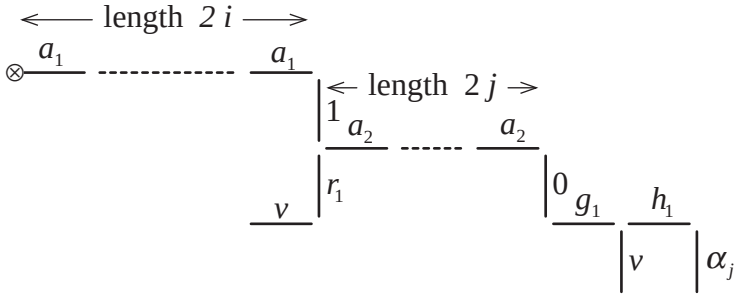


Figure 2.8: $i \in \{1,\ldots,n\}$.

(4) $\langle e\rangle$ with $e \in E$: Let $e = \{u,v\} \in E$. We define

$$\begin{aligned}\langle e\rangle \;=\;& (a_1a_1)^K\ 1\ (a_2a_2)^3\ 0g_1h_1h_2h_2h_1\ uu\ h_1h_2h_2h_1\ vv\ h_1h_2h_2h_1g_10\\ & (a_2a_2)^3\ 1\ (a_1a_1)^K.\end{aligned}$$

Notice that $(a_1a_1)^n$ is not a prefix of $\langle e \rangle$, but $(a_1a_1)^K$ is a prefix. The strings uu and vv appear in $\langle e \rangle$ once, respectively. The tree in Fig. 2.9 realizes a closed walk for $\langle e \rangle$ starting and ending at node $\otimes$.

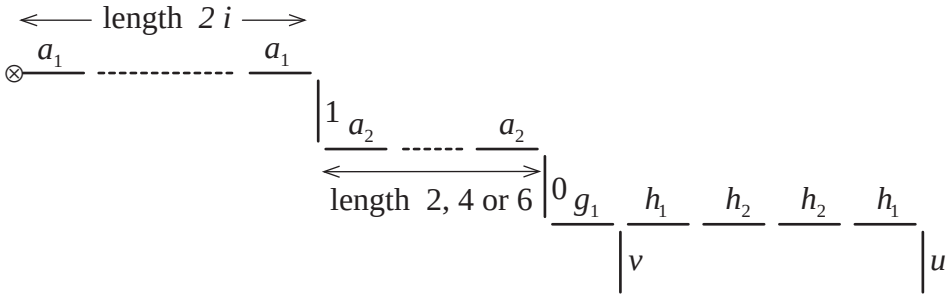


Figure 2.9: $i \in \{1, \ldots, K\}$.

Let

$$TEMPLATE = W^R W' W^R \prod_{i=1}^{n} (\prod_{j=1}^{3} (X_{i,j} W W^R)),$$

$$VERTEX = \prod_{j=1}^{3} (\prod_{i=1}^{n} (\langle v_i, j \rangle W W^R)),$$

$$EDGE = \prod_{i=1}^{m} (\langle e_i \rangle W W^R).$$

We then define

$$x = TEMPLATE \cdot VERTEX \cdot EDGE.$$

Finally, let $K' = 53n + m + 1 + 2\mu$, which is equal to $3\mu - 1$ since $\mu = 53n + m + 2$.

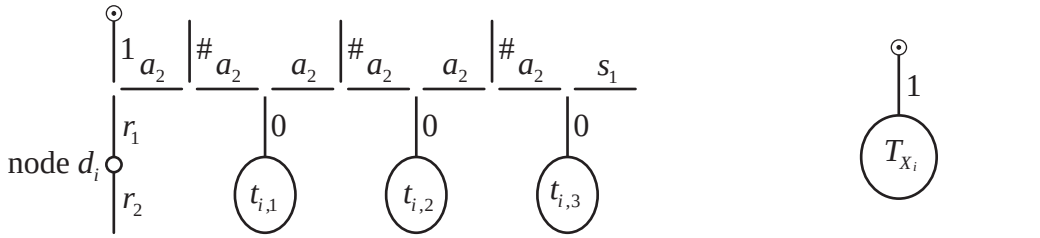
It is easy to see how the construction can be accomplished in polynomial time. We claim that there is a vertex cover of G with size at most K if and only if there is a tree of bounded degree three with at most K' edges which realizes a walk for x.

Suppose that G has a vertex cover U with $|U| \leq K$. Recall that n is the number of nodes in G. At first, for each $i \in \{1, \ldots, n\}$, we construct the tree T_{X_i} in Fig. 2.10.

It is obvious that for each $j \in \{1,2,3\}$, the tree T_{X_i} realizes a closed partial walk for the substring of $X_{i,j}$,

$$1\ (r_1r_2r_2r_1)^{\lfloor(3-j)/2\rfloor}\ (a_2\#\#a_2)^j\ x_{i,j}\ (11)^{\lfloor j/3\rfloor}\ (a_2a_2)^j\ 1,$$

such that the closed partial walk starts and ends at the node $\odot$ in Fig. 2.10.



(a) T_{X_i} (b) abbreviation for T_{X_i}

Figure 2.10: T_{X_i} has one internal free node d_i in addition to the internal free nodes $d_{i,j}$ and $\overline{d}_{i,j}$ in $t_{i,j}$ for $j = 1,2,3$.

We next construct the tree T_{TEM} in Fig. 2.11, which contains all the trees $T_{X_1}, \ldots, T_{X_n}$ as subgraphs of T_{TEM}. It is easy to check that T_{TEM} realizes a walk for the string *TEMPLATE* with the start node $\oplus$ and the end node $\otimes$.

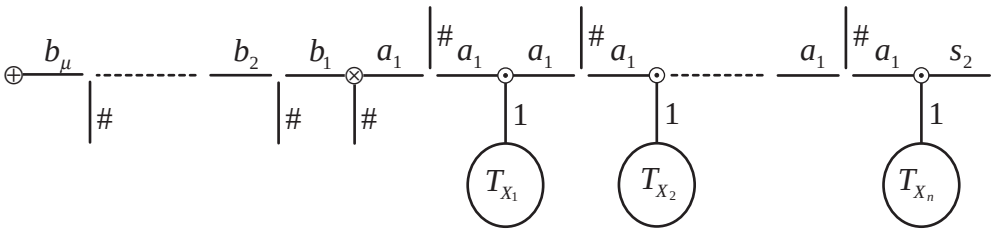


Figure 2.11: T_{TEM}

Here, we consider an assignment from V to the set of positive integers to n. Recall that U is a vertex cover of G with size at most K. Let b be a bijection $b : V \to \{1, \ldots, n\}$ such that $b(v) \leq K$ for all $v \in U$. For each $v \in V$, if $b(v) = i$, four new edges labeled

v are attached to T_{TEM} such that they are adjacent to the four nodes $d_i, d_{i,1}, d_{i,2}$ and $d_{i,3}$ of T_{TEM}, respectively.

Then the resulting tree realizes a walk for $TEMPLATE \cdot VERTEX$ since for $v \in V$ and $j \in \{1,2,3\}$, a closed partial walk for $\langle v, j \rangle$ is realized in the tree such that the start and end node is $\otimes$ in Fig. 2.11, which is also the end node of a partial walk for $TEMPLATE$. We call the tree the *vertex-selection tree for* b.

We finally consider a partial walk for the string $EDGE$. Since U is a vertex cover, for an edge $e = \{u, v\} \in E$, at least one of u and v belongs to U. Let $f : E \to U$ be a function which for each $e \in E$, returns one endpoint of e in U. For each edge $e = \{u, v\} \in E$, if $f(e) = u$, a new edge e_{new} labeled v is attached to the vertex selection tree for b such that e_{new} is adjacent to a free node of the nodes $\overline{d}_{b(u),1}$, $\overline{d}_{b(u),2}$ and $\overline{d}_{b(u),3}$. If $f(e) = v$, a new edge labeled u becomes adjacent to a free node of the nodes $\overline{d}_{b(v),1}$, $\overline{d}_{b(v),2}$ and $\overline{d}_{b(v),3}$. Such a free node must exist since for any $v \in U$, $|f^{-1}(v)|$ is less than or equal to three. For example, consider nodes $v, u_1, u_2, u_3 \in V$ and edges $\{v, u_1\}, \{v, u_2\}, \{v, u_3\} \in E$. If $b(v) = i \leq K$ and $f(\{v, u_1\}) = f(\{v, u_2\}) = f(\{v, u_3\}) = v$, then Fig. 2.12 becomes a subgraph of the resulting tree.

It is obvious from the construction that the resulting tree is a tree of bounded degree three with at most K' edges which realizes a walk for $x = TEMPLATE \cdot VERTEX \cdot EDGE$.

Conversely, suppose that there is a tree T of bounded degree three with at most K' edges which realizes a walk w for $x = TEMPLATE \cdot VERTEX \cdot EDGE$. Hereafter, for a substring y of x, the subwalk for y means the subwalk of w for y without any notice. The *induced subgraph of the subwalk for* y means the subgraph of T that is induced by all the edges in the subwalk for y. At first, we consider the induced subgraph of the subwalk for $W^R W' W^R$, which is a prefix of x.

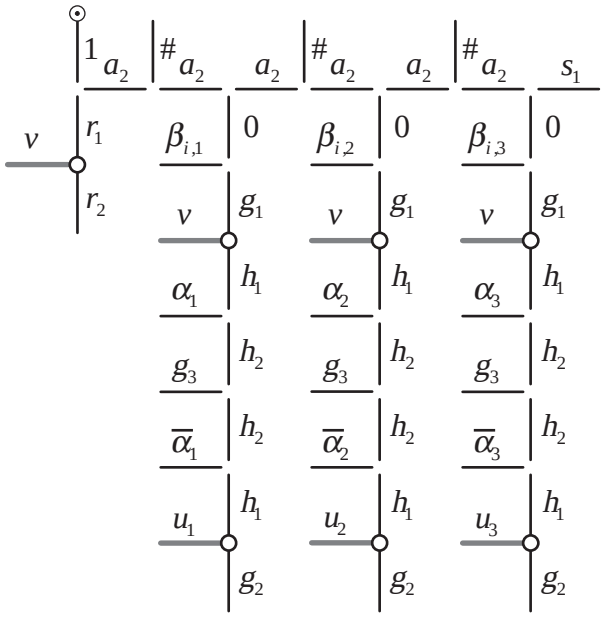


Figure 2.12: This tree is obtained from the tree T_{X_i} by attaching new edges to internal free nodes of T_{X_i}.

Claim 2.1 *Any tree of bounded degree three with at most K' edges that realizes a walk for $W^R W' W^R$ is isomorphic to the tree T_W.*

Proof of Claim 2.1. Let t be a tree of bounded degree three with at most K' edges that realizes a walk for $W^R W' W^R$. If T_W is not a subgraph of t, the tree t must contains adjacent edges with the same color #. In this case, t have at least $3\mu+1$ edges (see Fig. 2.13 (a)), a contradiction since $K' = 3\mu - 1$. Thus, T_W must be a subgraph of t. If T_W is a proper subgraph of t, i.e., T_W is not isomorphic to t, then t must be a tree with either 3μ or 4μ edges (see Fig. 2.13 (b)), which is also a contradiction. Therefore, t is isomorphic to T_W. ∎

By Claim W-claim, the induced subgraph of the subwalk for $W^R W' W^R$ is isomorphic to T_W. Moreover, this subgraph isomorphic to T_W contains exactly one linear chain of label W. Let us denote this linear chain by l_W. Notice that string WW^R occurs $6n + m$ times in x. We denote the ith WW^R occurring in x by $(WW^R)_i$ for

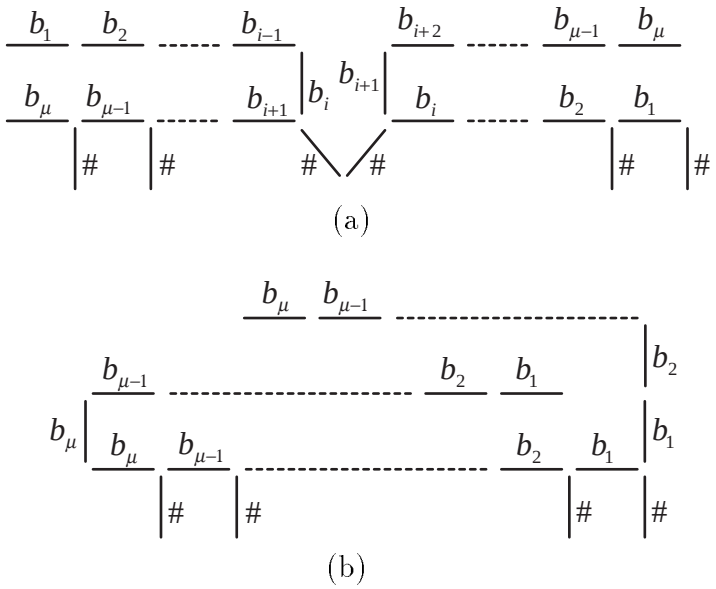


Figure 2.13: (a) T_W is not a subgraph of t. (b) T_W is a subgraph of t.

$1 \leq i \leq 6n + m$.

Claim 2.2 *For any $i \in \{1, 2, \ldots, 6n + m\}$, the induced subgraph of the subwalk for $(WW^R)_i$ coincides with the linear chain l_W.*

Proof of Claim 2.2. If the induced subgraph of the subwalk for $(WW^R)_i$ for some $i \in \{1, \ldots, 6n + m\}$ and the induced subgraph of the subwalk for $W^R W' W^R$, which is isomorphic to T_W, are disjoint, T must contain at least 3μ ($> K'$) edges, which is a contradiction. Therefore these induced subgraphs share some nodes for any i.

Consider the substring $(WW^R)_1$. By the following three reasons, we can see that the start node of the subwalk for $(WW^R)_1$ coincides with the end node of the subwalk for $W^R W' W^R$, which corresponds to the node $\otimes$ of T_W in Fig. 2.6.

1. The node of T corresponding to the node $\otimes$ of T_W in Fig. 2.6 has one adjacent edge labeled a_1 since the next symbol of the prefix $W^R W' W^R$ of x is a_1. We denote this edge by e_{a_1}.

2. It is easy to check that the subwalk for the substring $X_{1,1}$ of x, which occurs between the prefix $W^R W' W^R$ and the substring $(WW^R)_1$, cannot have any common edge with the induced subgraph of the subwalk for $W^R W' W^R$.

3. Notice that the last symbol of $X_{1,1}$ is also a_1. If the subwalk for the symbol a_1 does not coincide with the edge e_{a_1}, the subwalk for $(WW^R)_1$ cannot share any common nodes with the induced subgraph of the subwalk for $W^R W' W^R$, which is a contradiction.

If the induced subgraph of the subwalk for $(WW^R)_1$ does not coincide with the linear chain l_W, T must have a node with degree exceeding three, a contradiction. Thus the claim holds for $i = 1$. In a similar way, we can see that the claim holds in a consecutive manner for $i = 2, 3, \ldots, 6n + m$. ∎

Notice that for each $i \in \{1, 2, \ldots, 6n + m\}$, the subwalks for $(WW^R)_i$ is a closed partial walk starting and ending at the node corresponding to the node $\otimes$ in Fig. 2.6. Let

$$S = \{X_{i,j}, \langle v_i, j\rangle \mid i = 1, \ldots, n \text{ and } j = 1, 2, 3\} \cup \{\langle e_i\rangle \mid i = 1, \ldots, m\},$$

which consists of $X_{1,1}$ and the strings Z_i such that Z_i occurs in x as

$$x = \cdots (WW^R)_{i-1} Z_i (WW^R)_i \cdots$$

for $i = 2, 3, \ldots, 6n + m$. The following is clear from Claim 2.2:

Claim 2.3 *For each string Z in S, the subwalk for Z must be closed at the node corresponding to the node $\otimes$ in Fig. 2.6.*

We next consider the prefix $W^R W' W^R X_{1,1} W W^R$ of x. By carefully folding the prefix $W^R W' W^R X_{1,1} W W^R$, we can see the following claim:

Claim 2.4 *Let $T_{X_{1,1}}$ be the tree in Fig. 2.14. Any tree of bounded degree three with at most K' edges that realizes a walk for $W^R W' W^R X_{1,1} W W^R$ is isomorphic to the tree $T_{X_{1,1}}$.*

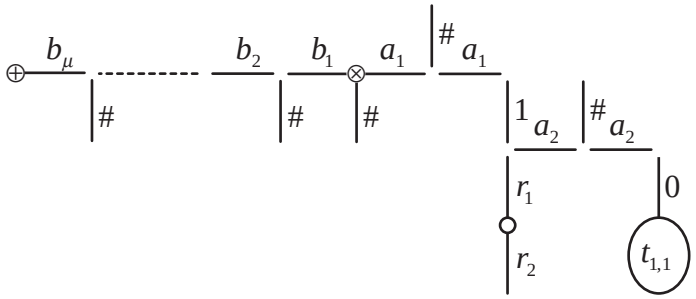


Figure 2.14: $T_{X_{1,1}}$

Consider the substring $X_{1,2}$ appearing after the prefix $W^R W' W^R X_{1,1} W W^R$. For convenience, we denote the prefix $W^R W' W^R \prod_{j=1}^{2}(X_{1,j} W W^R)$ of x by $X_{1,(1,2)}$. See Fig. 2.15. The nonshaded part of the tree is isomorphic to $T_{X_{1,1}}$. For each $i \in \{1,2,3,4\}$, let $t^{(i)}$ be the tree consisting of the shaded part $p^{(i)}$ and the nonshaded part. Then it is not hard to see that $t^{(1)}$, $t^{(2)}$, $t^{(3)}$, $t^{(4)}$ are the only trees of bounded degree three with at most K' edges which can realize a walk for $X_{1,(1,2)}$. By considering the substrings $X_{1,3}$ and $X_{n,1}$ that appear after $X_{1,(1,2)}$ in x, we can show the following:

Claim 2.5 *The possible form of the induced subgraph of the subwalk for $X_{1,(1,2)}$ is only $t^{(4)}$ in Fig. 2.15.*

Proof of Claim 2.5. If the induced subgraph of the subwalk for $X_{1,(1,2)}$ is either $t^{(1)}$ or $t^{(2)}$, then we can see from Claim 2.3 that there is no possibility to realize any closed partial walk for the substring $X_{n,1}$ in T which must start and end at the node corresponding to the node $\otimes$ in Fig. 2.15. Next consider the case that the induced

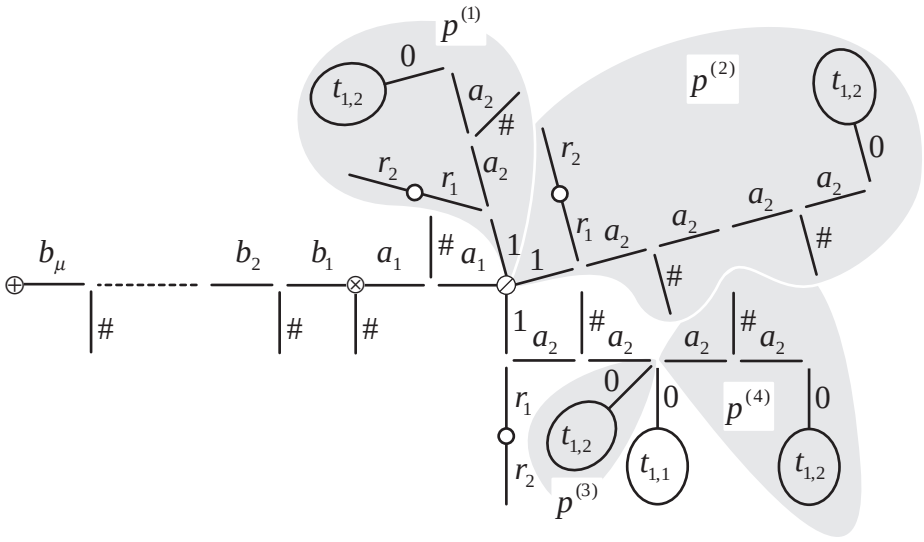


Figure 2.15: The four shaded parts are denoted by $p^{(1)}, p^{(2)}, p^{(3)}$ and $p^{(4)}$, respectively.

subgraph of the subwalk for $X_{1,(1,2)}$ is $t^{(3)}$. Let us denote a tree realizing a closed walk for a substring of $X_{1,3}$,

$$1\ (r_1r_2r_2r_1)^{\lfloor(3-j)/2\rfloor}\ (a_2\#\#a_2)^j\ x_{i,j}\ (s_1s_1)^{\lfloor j/3\rfloor}\ (a_2a_2)^j\ 1$$

by $t'_{1,3}$. We consider the subwalk for $X_{1,3}$ in T. In this case, the induced subgraph of the subwalk for the prefix $X_{1,(1,2)}\ X_{1,3}$ of x must be isomorphic to a tree obtained by attaching $t'_{1,3}$ to $t^{(3)}$ such that the node in $t'_{1,3}$ of degree one having the edge labeled 1 is identified with the node $\oslash$ of $t^{(3)}$. In the same way as the cases of $t^{(1)}$ and $t^{(2)}$, this implies no possibility to realize any closed partial walk for the substring $X_{n,1}$ in T. ∎

For convenience, we denote by X_1 the prefix $W^RW'W^R\ \prod_{j=1}^{3}(X_{1,j}WW^R)$. Let $T_{X_{1,(1,2,3)}}$ be the tree in Fig. 2.16. In a way similar to the above argument, it can be seen that any tree of bounded degree three with at most K' edges that realizes a walk for X_1 is isomorphic to the tree $T_{X_{1,(1,2,3)}}$.

We next consider the substring $X_{2,1}$ appearing after the prefix X_1. There are two possibilities to realize a walk for $X_1\ X_{2,1}WW^R$ by a tree of bounded degree three

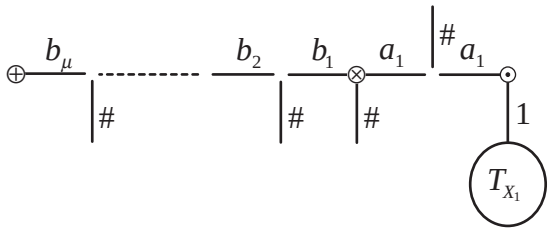


Figure 2.16: $T_{X_{1,(1,2,3)}}$.

with at most K' edges. See the tree in Fig. 2.17. The nonshaded part of the tree is isomorphic to $T_{X_{1,(1,2,3)}}$. One possibility is the tree consisting of the shaded part $\tilde{p}^{(1)}$ and the nonshaded part. The other is the tree consisting of the shaded part $\tilde{p}^{(2)}$ and the nonshaded part. It is clear that if the former is a subgraph of T then T cannot realize any closed partial walk for $X_{n,1}$. Therefore, any tree of bounded degree three with at most K' edges that realizes a walk for $X_1\ X_{2,1}WW^R$ must be isomorphic to the latter.

By continuing this argument for $X_{2,2}, X_{2,3}, X_{3,1}, \ldots, X_{n,3}$, we can obtain the following:

Claim 2.6 *Any tree of bounded degree three with at most K' edges that realizes a walk for $TEMPLATE = W^RW'W^R\prod_{i=1}^{n}(\prod_{j=1}^{3}(X_{i,j}WW^R))$ is isomorphic to the tree T_{TEM}.*

Next, consider the substring $\langle v, 1\rangle$ for $v \in V$. By Claim 2.3, the subwalk for $\langle v, 1\rangle$ is closed, and the start and end node of the subwalk for $\langle v, 1\rangle$ coincides with the node corresponding to the node $\otimes$ in Fig. 2.11. We next consider the subwalk for the prefix $(a_1a_1)^n$ of $\langle v, 1\rangle$. By Claim 2.6, the induced subgraph of the subwalk for *TEMPLATE* is isomorphic to T_{TEM}. Moreover, this subgraph contains exactly one linear chain of label $(a_1a_1)^n$. Let us denote this linear chain by $l_{(a_1a_1)^n}$. It can be easily seen that

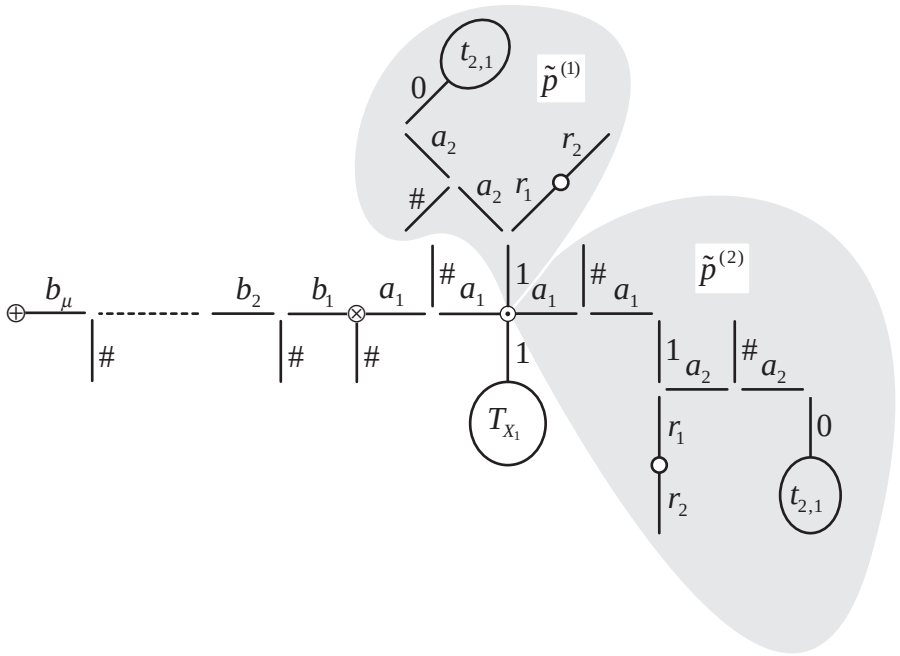


Figure 2.17: The two shaded parts are denoted by $\tilde{p}^{(1)}$ and $\tilde{p}^{(2)}$, respectively.

the induced subgraph of the subwalk for the prefix $(a_1a_1)^n$ of $\langle v, 1\rangle$ is a subgraph of $l_{(a_1a_1)^n}$. The subwalk for the substring

$$1\ (r_1r_2r_2r_1)^{\lfloor(3-j)/2\rfloor}\ (a_2\#\#a_2)^j\ x_{i,j}\ (s_1s_1)^{\lfloor j/3\rfloor}\ (a_2a_2)^j\ 1$$

of $X_{i,j}$, denoted by $sub_{X_{i,j}}$, must be closed since the subwalk for $X_{i,j}$ is closed (see Claim 2.3). Thus, the induced subgraph of the subwalks for $sub_{X_{i,1}}$, $sub_{X_{i,2}}$ and $sub_{X_{i,3}}$ must be isomorphic to T_{X_i}. The start and end nodes of these subwalks coincide and correspond to the node $\odot$ of T_{X_i} in Fig. 2.11. We denote this induced subgraph isomorphic to T_{X_i} by T_i and the node of T_i corresponding to the node $\odot$ of T_{X_i} by $\odot_i$. The end node of the subwalk for the prefix $(a_1a_1)^n$ of $\langle v, 1\rangle$ must coincide with $\odot_i$ for some $i \in \{1, \ldots, n\}$, but does not coincide with the node corresponding to the node $\otimes$ in Fig 2.11 since the next symbol occurring after the prefix $(a_1a_1)^n$ of $\langle v, 1\rangle$ is 1 and $\otimes$ does not have any adjacent edges labeled 1. We say that the subwalk for $\langle v, 1\rangle$ *selects*

T_i if the end node of the subwalk for the prefix $(a_1a_1)^n$ of $\langle v,1\rangle$ coincides with $\odot_i$.

Suppose that the subwalk for $\langle v,1\rangle$ selects T_i. Consider the substring

$$1r_1\ vv\ r_1a_2a_2 0g_1\ vv\ h_1\ \alpha_1\alpha_1\ h_1g_1 0a_2a_2 1$$

of $\langle v,1\rangle$ appearing after the prefix $(a_1a_1)^n$. We denote the substring by $sub_{\langle v,1\rangle}$. By Fact 2.1 and Claim 2.3, the subwalk for $sub_{\langle v,1\rangle}$ must be closed. It is trivial that any tree of bounded degree three realizing a walk for $sub_{\langle v,1\rangle}$ is the tree in Fig. 2.18. This implies that the induced subgraph of the subwalks for *TEMPLATE* and $\langle v,1\rangle$ is

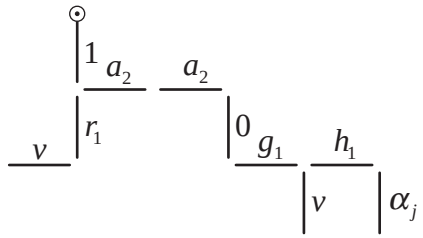


Figure 2.18: The node $\odot$ is the start and end node of a walk for $sub_{\langle v,1\rangle}$.

isomorphic to the tree obtained from T_{TEM} by attaching two new edges labeled v to the nodes d_i and $d_{i,1}$, respectively. Note that the remainder $(a_1a_1)^n$ of $\langle v,1\rangle$ finishes the subwalk for $\langle v,1\rangle$ in T.

We should notice that if the subwalk for $\langle v,1\rangle$ has selected T_i then no other subwalk for $\langle v',1\rangle$ with $v' \neq v$ can select T_i since the node d_i has not been free and any of the labels of the three adjacent edges of d_i is not v'.

For an arbitrary bijection $b : V \to \{1,\ldots,n\}$, let $T_{TEM,b}$ be the tree obtained from T_{TEM} by attaching two new edges labeled v to $d_{b(v)}$ and $d_{b(v),1}$ of T_{TEM} for all $v \in V$. It is easy to see that the following holds:

Claim 2.7 *A tree T' is a tree of bounded degree three with at most K' edges that realizes a walk for $TEMPLATE \cdot \prod_{i=1}^{n}(\langle v_i,1\rangle WW^R)$ if and only if T' is isomorphic to*

$T_{TEM,b}$ *for some bijection* $b : V \to \{1, \ldots, n\}$.

Next, consider the substring $\langle v, 2\rangle$. In the same way as the subwalk for $\langle v, 1\rangle$, we also say that the subwalk for $\langle v, 2\rangle$ *selects* T_i if the end node of the subwalk for the prefix $(a_1a_1)^n$ of $\langle v, 2\rangle$ coincides with $\odot_i$. The string vv occurs in $\langle v, 2\rangle$ twice as a substring. If the subwalk for $\langle v, 1\rangle$ selects T_i then the subwalk for $\langle v, 2\rangle$ must also select T_i, since the subwalk for the first vv coincides with the edge labeled v adjacent to the node corresponding to d_i. Moreover, the subwalk for the second occurrence of vv must be the edge labeled v adjacent to the node corresponding to $d_{b(v),2}$.

By a similar argument on $\langle v, 3\rangle$, the following claim holds:

Claim 2.8 *A tree* T' *is a tree of bounded degree three with at most* K' *edges realizing a walk for* $TEMPLATE \cdot VERTEX$ *if and only if* T' *is isomorphic to the vertex-selection tree for some bijection* $b : V \to \{1, \ldots, n\}$.

Since we have suppose that there is a tree T of bounded degree three with at most K' edges which realizes a walk for x, the tree T is isomorphic to the vertex-selection tree for a bijection $b : V \to \{1, \ldots, n\}$. We consider the substring $\langle e\rangle$ for $e = \{u, v\} \in E$. In the same way as the subwalks for $\langle v, 1\rangle$ and $\langle v, 2\rangle$, we also say that the subwalk for $\langle e\rangle$ *selects* T_i if the end node of the subwalk for the prefix $(a_1a_1)^K$ of $\langle e\rangle$ coincides with $\odot_i$. It is clear that the subwalk for $\langle e\rangle$ must select T_i with $i \leq K$.

Suppose that the subwalk for $\langle e\rangle$ selects T_i for some $i \in \{1, \ldots, K\}$. Consider the substring $(a_2a_2)^3$ of $\langle e\rangle$ appearing after the prefix $(a_1a_1)^K 1$. We denote the induced subgraph of the subwalk for $x_{i,j}$ by $t'_{i,j}$, which is isomorphic to $t_{i,j}$, and denote the start and end node of the subwalk for $x_{i,j}$ by $\odot_{i,j}$, which corresponds to the node $\odot$ of $t_{i,j}$ in Fig. 2.7. If the end node of the subwalk for $(a_1a_1)^K 1(a_2a_2)^3$ coincides with the node $\odot_{i,j}$, we say that the subwalk for $\langle e\rangle$ *selects* $t'_{i,j}$.

Suppose that the subwalk for $\langle e \rangle$ selects $t'_{i,j}$. We denote the substring

$$0g_1h_1h_2h_2h_1\ uu\ h_1h_2h_2h_1\ vv\ h_1h_2h_2h_1g_10$$

of $\langle e \rangle$ by $sub_{\langle e \rangle}$, which follows after the prefix $(a_1a_1)^K1(a_2a_2)^3$. By Fact 2.1 and Claim 2.3, the subwalk for $sub_{\langle e \rangle}$ must be closed. Note that $b^{-1}(i)$ is either u or v. If not, we can easily see that T must have a node with degree exceeding three, which is a contradiction. If $b^{-1}(i) = v$, the subwalk for $sub_{\langle e \rangle}$ must include the edge labeled u being adjacent to the node corresponding to $\overline{d}_{i,j}$ in the vertex-selection tree for b (see Fig. 2.19). If $b^{-1}(i) = u$, the subwalk for $sub_{\langle e \rangle}$ must include the edge labeled v being

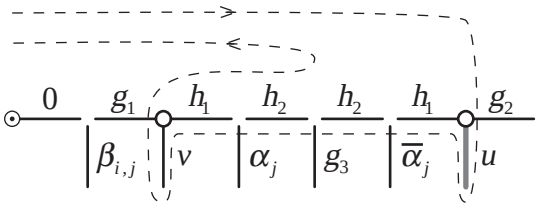


Figure 2.19: The broken line is the subwalk for $sub_{\langle e \rangle}$.

adjacent to the node corresponding to $\overline{d}_{i,j}$ in the vertex-selection tree for b (see Fig. 2.20).

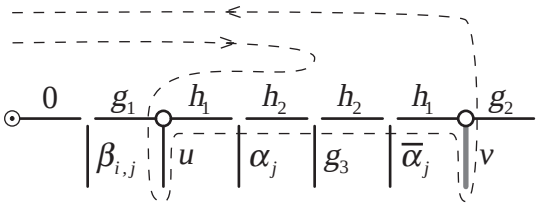


Figure 2.20: The broken line is the subwalk for $sub_{\langle e \rangle}$.

The suffix $(a_2a_2)^31(a_1a_1)^K$ of $\langle e \rangle$ finishes the subwalk for $\langle e \rangle$ in T. Notice that if the subwalk for $\langle e \rangle$ has selected $t'_{i,j}$ then no other subwalk for $\langle e' \rangle$ can select $t'_{i,j}$.

Let $U = \{b^{-1}(i) \mid i \in \{1, \ldots, K\}\}$. It should be clear from the form of T that U is a vertex cover of G with size K.

□

Theorem 2.4 does not mention anything about the size of the alphabet Σ. We then try to show the lower bound of the alphabet size so that the problem remains NP-complete. Here is a problem: If a given string x is long and written over a few symbols, Fact 2.1 would be not useful to determine the tree realizing a closed walk for the string x. In such a case, the following fact is helpful:

Fact 2.2 *Let x be a string. Suppose that t is an arbitrary tree realizing a closed walk w for x, and that t' is the smallest tree realizing a walk w' for x.*

1. *For an edge e of t' traversed in w' exactly twice, let $e = \{w'[i], w'[i+1]\} = \{w'[j], w'[j+1]\}$ with $0 \leq i < j \leq |x|-1$. Then, $\{w[i], w[i+1]\} = \{w[j], w[j+1]\}$.*

2. *For an edge e of t' traversed in w' exactly four times, let*

$$\begin{aligned} e &= \{w'[i_1], w'[i_1+1]\} \\ &= \{w'[i_2], w'[i_2+1]\} \\ &= \{w'[i_3], w'[i_3+1]\} \\ &= \{w'[i_4], w'[i_4+1]\} \end{aligned}$$

with $0 \leq i_1 < i_2 < i_3 < i_4 \leq |x|-1$. Then, one of the following three cases holds:

Case 1: $\{w[i_1], w[i_1+1]\} = \{w[i_2], w[i_2+1]\}\ (= e_0)$,
$\{w[i_3], w[i_3+1]\} = \{w[i_4], w[i_4+1]\}\ (= e_1)$
and $e_0 \neq e_1$.

Case 2: $\{w[i_1], w[i_1+1]\} = \{w[i_4], w[i_4+1]\}\ (= e_0)$,
$\{w[i_2], w[i_2+1]\} = \{w[i_3], w[i_3+1]\}\ (= e_1)$
and $e_0 \neq e_1$.

Case 3: $\{w[i_1], w[i_1+1]\} = \{w[i_2], w[i_2+1]\} =$
$\{w[i_3], w[i_3+1]\} = \{w[i_4], w[i_4+1]\}$.

Fact 2.2 can be proved by Lemma 2.6 and the fact that the shortest path between two nodes in a tree is unique.

Lemma 2.6 *Suppose that t is an arbitrary tree realizing a walk for a string x and t' is the smallest tree realizing a walk for x. We denote those walks by w and w', respectively. Then, for any integers $0 \leq i < j \leq |x| - 1$, if $\{w'[i], w'[i+1]\} \neq \{w'[j], w'[j+1]\}$, then $\{w[i], w[i+1]\} \neq \{w[j], w[j+1]\}$.*

This lemma can be easily proved from Lemmas 2.2 and 2.3.

Unfortunately, the NP-hardness of our problem still holds with a constant number of colors as follows:

Theorem 2.5 *The graph inference from a walk for the class of undirected trees of bounded degree three is NP-complete even if the alphabet size is restricted to four.*

Proof. We give a reduction to show the NP-hardness by coding the string x defined in the previous proof into a string x' over the alphabet $\Sigma' = \{0, 1, \#, \$\}$.

We first make a string over Σ', which is denoted by $code(i, j)$. For integers $j, k \geq 0$, the notation j_k denotes the kth bit of the binary representation of j, i.e., $j_0 + j_1 \cdot 2 + j_2 \cdot 2^2 + \cdots + j_m \cdot 2^m$ for some integer $m \geq \lfloor \log j \rfloor$. For integers $i \geq j \geq 1$, the string $code(i, j)$ is defined as follows:

$$code(i,j) = \begin{cases} j_0 j_0 & \text{if } i = 1 \\ j_0 \cdot \prod_{k=1}^{\lfloor \log i \rfloor} (\#\#\$\#\#j_k) \cdot \prod_{k=1}^{\lfloor \log i \rfloor} (j_{\lfloor \log i \rfloor - k + 1}\$) \cdot j_0 & \text{if } i \geq 2 \end{cases}$$

Recall that the alphabet Σ defined in the proof of Theorem 2.4 is

$$\begin{aligned} \Sigma \;=\; & \{a_1, a_2\} \cup \{g_1, g_2, g_3\} \cup \{h_1, h_2\} \cup \{r_1, r_2\} \cup \{s_1, s_2\} \\ & \cup \{0, 1, \#\} \\ & \cup \{\alpha_i, \overline{\alpha}_i \mid i = 1, 2, 3\} \\ & \cup \{\beta_{i,j} \mid i = 1, \ldots, n \text{ and } j = 1, 2, 3\} \\ & \cup V. \end{aligned}$$

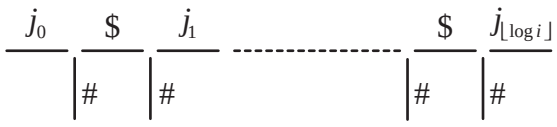


Figure 2.21: The tree of bounded degree three that realizes a closed walk for $code(i,j)$.

The string x' is obtained from x by replacing the symbols and substrings of x which are listed in the left column of Table 2.1 with those in the right column.

Thus the substrings of x which are $x_{i,j}, X_{i,j}, \langle v_i, j\rangle, \langle e\rangle, W$ and W' are transformed as follows:

(1) For $i \in \{1,2,\ldots,n\}$ and $j \in \{1,2,3\}$,

$$\begin{aligned} x_{i,j} &= 0\#\$\#\#\$\#\#\$\ code(1,(j+1)mod2)\ \#00\#\ code(3,j)\ \$\# \\ &\quad (\$\#\#\ code(3n,3(i-1)+j)\ \$)\ 0. \\ X_{i,j} &= (\$\#\#\$)^i\ 1\ (\#\$\$\#)^{\lfloor (3-j)/2 \rfloor}\ (\$\#\#\$)^j\ x_{i,j}\ (11)^{\lfloor j/3 \rfloor}\ (\$\$)^j\ 1 \\ &\quad (00)^{\lfloor i/n \rfloor}(\$\$)^i. \end{aligned}$$

(2) For $v_i \in V$,

$$\begin{aligned} \langle v_i, j\rangle &= (\$\$)^n\ 1\#\ code(n,i)\ \#\ (\$\$)^j\ 0\#\ code(n,i)\ \$\ code(3,j)\ \$\#0 \\ &\quad (\$\$)^j\ 1\ (\$\$)^n. \end{aligned}$$

(3) For $e = \{v_i, v_j\} \in E$,

$$\begin{aligned} \langle e\rangle &= (\$\$)^K\ 1\ (\$\$)^3\ 0\#\$\#\#\$\ code(n,i)\ \$\#\#\$\ code(n,j)\ \$\#\#\$\#0 \\ &\quad (\$\$)^3\ 1\ (\$\$)^K. \end{aligned}$$

(4) $W = \prod_{i=1}^{\mu'} b_i$ and $W' = \prod_{i=1}^{\mu'}(\#\#b_i)$, where

$$\mu' = 58n + 2 + 3n(4\lfloor \log 3n \rfloor + 1) + (4n+m)(4\lfloor \log n \rfloor + 1).$$

and b_i is defined in the former proof.

a_1	\$
a_2	\$
g_1	#
g_2	#
g_3	0
h_1	\$
h_2	#
r_1	#
r_2	\$
s_1	1
s_2	0
0	0
1	1
#	#
$\alpha_i\alpha_i$	$code(3,i)$
$\overline{\alpha}_i\overline{\alpha}_i$	$code(1,(i+1)mod2)$
$\beta_{i,j}\beta_{i,j}$	$\#\# \; code(3n,3(i-1)+j) \; \$$
$v_iv_i(v_i \in V)$	$code(n,i)$.

Table 2.1: For each row, the symbol or string of the left column is replaced with those of the right column.

Then x' is also written as follows:

$$x' = TEMPLATE \cdot VERTEX \cdot EDGE$$

where

$$\begin{aligned} TEMPLATE &= W^R W' W^R \prod_{i=1}^{n} (\prod_{j=1}^{3} (X_{i,j} W W^R)), \\ VERTEX &= \prod_{j=1}^{3} (\prod_{i=1}^{n} (\langle v_i, j \rangle W W^R)), \\ EDGE &= \prod_{i=1}^{m} (\langle e_i \rangle W W^R). \end{aligned}$$

Let

$$K'' = 58n + 3n(4\lfloor \log 3n \rfloor + 1) + (4n + m)(4\lfloor \log n \rfloor + 1) + 1 + 2\mu'.$$

This transformation can be also done in polynomial time. We claim that there is a vertex cover of G with size at most K if and only if there is a tree of bounded degree three with at most K'' edges which realizes a walk for x'. We leave the verification of the claim to the reader. □

By slightly modifying the reduction in the proof of Theorem 2.5, we can obtain a result on trees of bounded degree k for each $k \geq 3$.

Theorem 2.6 *For any integer $k \geq 3$, the graph inference from a walk for undirected trees of bounded degree k is NP-complete even if the alphabet size is restricted to $k+1$.*

The number $k + 1$ of the alphabet size in the above theorem is optimal since the problem for trees of bounded degree k with at most k colors is solvable in linear time. As a trivial consequence of Lemma 2.4, the smallest tree realizing a walk for a string x of at most k colors has no node of degree exceeding k. Therefore, we can solve this case by the linear-time algorithm for the tree inference from a walk, which is provided in Section 2.3.

2.4.2 (1,1)-caterpillars

We have shown that the graph inference from a walk for trees of bounded degree three is NP-complete. On the other hand, the problem for linear chain, as stated in Section 2.2, is solvable in polynomial time. Notice that the class of linear chains is a proper subclass of trees of bounded degree three. By putting some constraint on the structure of such trees, we consider simple trees of bounded degree three which are similar to linear chains, called $(1,1)$-*caterpillars*.

A caterpillar is a tree which is created by a linear chain, called the *backbone*, and various other appendage linear chains attached to the nodes of the backbone, called *hairs* (e.g., [HMM91, CM90]). Let α and $\beta \geq 0$ be integers. We define an (α, β)-caterpillar as a caterpillar such that the number of hairs of a node in the backbone is at most α and the maximum length of hairs is at most β. Thus, a (1,1)-caterpillar is just created from a linear chain l by attaching at most one edge to each node of the linear chain l.

In contrast with the result by Aslam and Rivest [AR90] and Raghavan [Rag94] that the graph inference from a walk is solvable in polynomial time for linear chains, the problem turns to be intractable if each node of a linear chain l is allowed to append at most one edge to the linear chain l.

Theorem 2.7 *The graph inference from a walk for the class of (1,1)-caterpillars is NP-complete.*

Proof. We also give a reduction from 3-DEGREE VERTEX COVER. Let $K \leq |V|$ be a positive integer and $G = (V, E)$ be a graph of bounded degree three without any self-loop, where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We must construct a string x over an alphabet Σ and a positive integer K' such that there is a $(1,1)$-caterpillar T

with at most K' edges which realizes a walk for x if and only if G has a vertex cover of size K or less.

The basic idea of the reduction is the same as the reduction in the proof of Theorem 2.4. For example, we will define a string W, which is similar to the string W defined in the proof of Theorem 2.4, so that if a string y occurs in x as $x = \cdots W^R y W \cdots$ then the subwalk for y must be closed.

Let

$$\begin{aligned}\Sigma &= \{a_1, a_2, a_3\} \\ &\quad \cup \{0, 1\} \\ &\quad \cup \{g\} \\ &\quad \cup \{\alpha_1, \alpha_2, \alpha_3\} \\ &\quad \cup \{\beta_{i,j} \mid i = 1, \ldots, n \text{ and } j = 1, \ldots, 12\}.\end{aligned}$$

We define $X_i, \langle v \rangle, \langle e \rangle$ and W as follows:

(1) For $i \in \{1, 2, \ldots, n\}$,

$$\begin{aligned}X_i &= g\ (\beta_{i,1})^2\ g\ (\beta_{i,2})^2\ gg\ (\beta_{i,3})^2 \\ &\quad \textstyle\prod_{j=1}^{3} (a_1\ \alpha_j \alpha_j\ a_2 a_3\ (\beta_{i,3j+1})^2\ a_3 a_2\ (\beta_{i,3j+2})^2\ a_1\ (\beta_{i,3j+3})^2).\end{aligned}$$

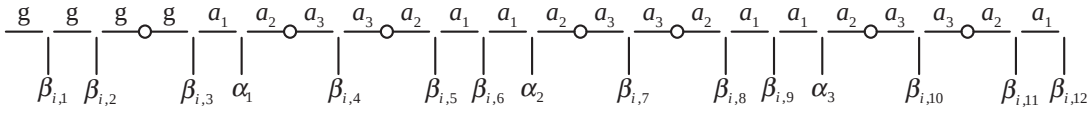


Figure 2.22: A (1,1)-caterpillar which realizes a walk for X_i.

(2) For $v \in V$,

$$\begin{aligned}\langle v \rangle &= (g^4\ (a_1 a_2 a_3 a_3 a_2 a_1)^3\ )^{n-1}\ g^4 g\ vv\ g\ \textstyle\prod_{j=1}^{3} (a_1\ \alpha_j \alpha_j\ a_2\ vv\ a_3 a_3 a_2 a_1) \\ &\quad ((a_1 a_2 a_3 a_3 a_2 a_1)^3\ g^4\ )^n.\end{aligned}$$

(3) For $e = \{u, v\} \in E$,

$$\begin{aligned}\langle e \rangle &= (g^4\ (a_1 a_2 a_3 a_3 a_2 a_1)^3\ )^K\ (a_1 a_2 a_3 a_3 a_2 a_1)^2 \\ &\quad a_1 a_2 a_3 a_3\ uu\ a_3 a_3\ vv\ a_3 a_3 a_2 a_1 \\ &\quad (a_1 a_2 a_3 a_3 a_2 a_1)^2\ ((a_1 a_2 a_3 a_3 a_2 a_1)^3\ g^4\ )^K.\end{aligned}$$

(4) $W = \prod_{i=1}^{\mu} b_i$, where $\mu = 41n + m + 1$ and b_i is the same as that in the proof of Theorem 2.4.

We then define

$$x = TEMPLATE \cdot VERTEX \cdot EDGE$$

where

$$\begin{aligned} TEMPLATE &= W^R \prod_{i=1}^{n} X_i \, ((a_1a_2a_3a_3a_2a_1)^3 g^4)^n \, WW^R, \\ VERTEX &= \prod_{i=1}^{n} (\langle v_i \rangle WW^R), \\ EDGE &= \prod_{i=1}^{m} (\langle e_i \rangle WW^R). \end{aligned}$$

Finally, let $K' = 41n + m + \mu$.

This transformation can be done in polynomial time. We claim that G has a vertex cover of size at most K if and only if there is a (1,1)-caterpillar with at most K' edges that realizes a walk for x. The proof is not hard and left to the reader. □

We can show the following theorem by a similar reduction:

Theorem 2.8 *For any integers $\alpha, \beta \geq 1$, the graph inference from a walk for (α, β)-caterpillars is NP-complete.*

2.5 Approximability

As we have seen, the graph inference from a walk is NP-complete for degree-bounded trees even if such trees are restricted to $(1, 1)$-caterpillars. For the NP-complete problems, we investigate to approximately solve them in polynomial-time.

2.5.1 MAX SNP

We first show that the graph inference from a walk for trees of bounded degree three is MAX SNP-hard. From the result due to Arora et al. [ALM+92], it follows that there is no polynomial time approximation scheme for the problem unless P=NP.

Let Π_1 and Π_2 be two optimization (maximization or minimization) problems. We say that Π_1 *L-reduces* to Π_2 if there are polynomial time algorithms f and g and constants α and $\beta > 0$ such that:

1. Given an instance I_1 of Π_1 with optimal cost $opt(I_1)$, the algorithm f produces an instance I_2 of Π_2 with optimal cost $opt(I_2)$ that satisfies

$$opt(I_2) \leq \alpha \cdot opt(I_1),$$

and

2. Given any feasible solution s_2 of I_2, the algorithm g produces a solution s_1 of I_1 such that

$$|cost(s_1) - opt(I_1)| \leq \beta \cdot |cost(s_2) - opt(I_2)|.$$

Some basic facts about L-reductions hold as follows: First, the composition of two L-reductions is also an L-reduction. Second, if problem Π_1 L-reduces to problem Π_2 and Π_2 can be approximated in polynomial time with relative error δ, i.e., there is an algorithm $\mathcal{A}$ for Π_2 with

$$\delta \geq \frac{|opt(x) - cost(\mathcal{A}(x))|}{opt(x)},$$

then Π_1 can be approximated with relative error $\alpha\beta\delta$. In particular, if Π_2 has a polynomial time approximation scheme, then so does Π_1.

MAX SNP_0 is the class of maximization problems defined syntactically in Papadimitriou and Yannakakis [PY91, Pap94]. It is known that every problem in this class can

be approximated within *some* constant factor. MAX SNP is defined as the class of all optimization problems that are L-reducible to a problem in MAX SNP_0. A problem Π in MAX SNP is said to be MAX SNP-complete if for any other problem Π' in MAX SNP, there is an L-reduction from Π' to Π. Papadimitriou and Yannakakis [PY91, Pap94] showed that several natural, well-studied optimization problems such as the vertex cover and independent set problems on degree-bounded graphs, Max Cut, various versions of the maximum satisfiability problem are MAX SNP-complete. We say that a problem Π is MAX SNP-hard if every problem in MAX SNP can be L-reduced to Π. Various optimization problems have been reported to be MAX SNP-hard (e.g., [Zha95, JWZ95, GVY94, GMM94, Kan94, ZJ94, GVY93, PSW93, PY93, Kan92, Ihl91, Kan91, BJL+94, BP89]).

Theorem 2.9 *The graph inference from a walk for the class of undirected trees of bounded degree three is MAX SNP-hard.*

Proof. We give an L-reduction from 4-DEGREE VERTEX COVER, which is shown to be MAX SNP-complete in [PY91, Pap94]. The L-reduction is constructed in the following way: First, we construct a reduction from 4-DEGREE VERTEX COVER to show the NP-hardness of the graph inference problem for trees of bounded degree three in the same way as the reduction in the proof of Theorem 2.4. Secondly, we modify the reduction from 4-DEGREE VERTEX COVER so that the resulting reduction becomes a L-reduction implying that the graph inference from a walk for trees of bounded degree three is MAX SNP-hard.

We briefly describe a key of the L-reduction. A string x will be defined using the strings

$X_{i,j}$ for $i \in \{1,2,\ldots,n\}$ and $j \in \{1,2,3,4\}$,
$\langle v,j \rangle$ for $v \in V$ and $j \in \{1,2,3,4\}$,
$\langle e \rangle$ for $e \in E$,
W' and W,

which play the same role as those of $X_{i,j}, \langle v, j\rangle, \langle e\rangle, W'$ and W in the proof of Theorem 2.4, respectively. Suppose that there is a tree T of bounded degree three with at most $3\mu - 1$ edges which realizes a walk w for x, where $\mu = 70n + m + 2$. It should be easily seen that the induced subgraph, denoted by T_i, of the subwalks of w for $X_{i,1}, X_{i,2}, X_{i,3}$ and $X_{i,4}$ is isomorphic to the tree in Fig. 2.23. We denote by $\overline{d}_i$ the new free node of the tree, which is one endpoint of the edge labeled r_2 and also one endpoint of the edge labeled r_3.

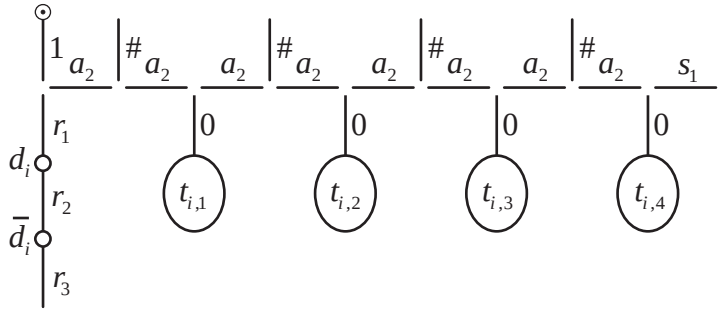


Figure 2.23: This tree has one internal free node $\overline{d}_i$ in addition to the nodes d_i and $d_{i,j}, \overline{d}_{i,j}$ in $t_{i,j}$ for $j = 1, 2, 3, 4$.

Furthermore, it is also easy to see that if the subwalk for $\langle e\rangle$ selects T_i, i.e., the end node of the subwalk for the prefix $(a_1a_1)^n$ of $\langle e\rangle$ coincides with the node of T_i corresponding to the node $\odot$ of the tree in Fig. 2.23, then the node of T_i corresponding to the node $\overline{d}_i$ of the tree in Fig. 2.23 must have an adjacent edge labeled γ. Then, the number of edges labeled γ in a tree realizing a walk for x coincides with the size of a vertex cover.

We define the string x as follows:

(1) For $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, 3, 4\}$,

$$\begin{aligned} x_{i,j} &= 0g_1h_1h_2h_2h_1g_2g_2h_1\ \overline{\alpha}_j\overline{\alpha}_j\ h_2g_3g_3h_2\ \alpha_j\alpha_j\ h_1g_1\ \beta_{i,j}\beta_{i,j}\ 0. \\ X_{i,j} &= (a_1\#\#a_1)^i\ 1\ (r_1r_2r_3r_3r_2r_1)^{\lfloor(4-j)/3\rfloor}\ (a_2\#\#a_2)^j \\ &\quad x_{i,j}\ (s_1s_1)^{\lfloor j/4\rfloor}\ (a_2a_2)^j\ 1\ (s_2s_2)^{\lfloor i/n\rfloor}\ (a_1a_1)^i. \end{aligned}$$

(2) For $v \in V$ and $j \in \{1,2,3,4\}$,

$$\langle v,j\rangle \;=\; (a_1a_1)^n\ 1r_1\ vv\ r_1\ (a_2a_2)^j\ 0g_1\ vv\ h_1\ \alpha_j\alpha_j\ h_1g_10\ (a_2a_2)^j\ 1\ (a_1a_1)^n.$$

(3) For $e = \{u,v\} \in E$,

$$\begin{aligned}\langle e\rangle \;=\;& (a_1a_1)^n\ 1\ (r_1r_2\gamma\gamma r_2r_1)\ (a_2a_2)^4\ 0\\ & g_1h_1h_2h_2h_1\ uu\ h_1h_2h_2h_1\ vv\ h_1h_2h_2h_1g_10\ (a_2a_2)^4\ 1\ (a_1a_1)^n.\end{aligned}$$

(4) $W = \prod_{i=1}^{\mu} b_i$ and $W' = \prod_{i=1}^{\mu}(\#\#b_i)$, where $\mu = 70n + m + 2$ and b_i is defined in the proof of Theorem 2.4.

(5) $x = TEMPLATE \cdot VERTEX \cdot EDGE$

where

$$\begin{aligned} TEMPLATE &= W^RW'W^R \prod_{i=1}^{n}(\prod_{j=1}^{4}(X_{i,j}WW^R)),\\ VERTEX &= \prod_{j=1}^{4}(\prod_{i=1}^{n}(\langle v_i,j\rangle WW^R)),\\ EDGE &= \prod_{i=1}^{m}(\langle e_i\rangle WW^R). \end{aligned}$$

The string x can be produced in polynomial time. The first condition of L-reduction is satisfied with $\alpha = 2120$ since $\lceil \frac{m}{4} \rceil \le opt(G)$ and $\lceil \frac{n}{5} \rceil \le opt(G)$, where $opt(G)$ is the size of minimum covers of G.

We next define an algorithm g as follows: Let s_2 be a solution of our problem, i.e., a tree T of bounded degree three which realizes a walk for x. If s_2 has at most $3\mu - 1$ edges, the algorithm g returns the subset U of V such that for each $v \in V$, v is in U if and only if there is $i \in \{1,\ldots,n\}$ satisfying that the node of T corresponding to d_i of the tree in Fig. 2.23 has an adjacent edge labeled v and the node $\tilde{d}_i$ of T corresponding to $\overline{d}_i$ is not free, i.e., $\tilde{d}_i$ has an adjacent edge labeled γ. Otherwise, g returns V. Then it is trivial that the second condition holds with $\beta = 1$. □

We omit the proof of the following theorem:

Theorem 2.10 *For any integers $\alpha, \beta \geq 1$, the graph inference from a walk for (α, β)-caterpillars is MAX SNP-hard.*

2.5.2 Hardness of approximations

Let π be an optimization problem and $\mathcal{A}$ be an algorithm to approximately solve π. The algorithm $\mathcal{A}$ is said to be ϵ-approximation if the ratio is less than or equal to ϵ, i.e.,

$$\frac{\mathcal{A}(x)}{opt(x)} \leq \epsilon,$$

where $\mathcal{A}(x)$ is the cost of the solution produced by $\mathcal{A}$ from x and $opt(x)$ is the optimal cost for x.

Theorem 2.11 *Let $\epsilon < 2$. If there is a polynomial-time ϵ-approximation algorithm for the graph inference from a walk for the class of undirected trees of bounded degree three, then $P = NP$.*

Proof. Let $c > 0$ be any fixed constant. We modify the reduction in the proof of Theorem 2.4 by replacing μ with $c \cdot \mu$. This just means that the string W is stretched c times in the length. We denote this resulting reduction by R_c. We can easily see that the reduction R_c also gives the NP-hardness of the graph inference from a walk for trees of bounded degree three.

Let x be the string produced by R_c and T be a tree of bounded degree three which realizes a walk for x. The following fact holds:

1. Assume that G has a vertex cover U with $|U| \leq K$. If T is the smallest tree of bounded degree three which realizes a walk for x, T has $(2c+1)\mu - 1$ edges. Otherwise, T has at least $4c \cdot \mu$ edges.

2. If G does not have any vertex cover U with $|U| \leq K$, then T has at least $4c \cdot \mu$ edges.

Since $\frac{4c}{2c+1} \cdot ((2c+1)\mu - 1) < 4c\mu$, it holds that a $\frac{4c}{2c+1}$-approximation algorithm returns a tree of bounded degree three with $(2c+1)\mu - 1$ edges which realizes a walk for x if and only if G has a vertex cover U with $|U| \leq K$. For any $\epsilon < 2$, there is a constant c with $\epsilon \leq 4c/(2c+1)$. □

It is trivial that for some integer $k \geq 3$ and constant $\epsilon < 2$, if there is a polynomial-time ϵ-approximation algorithm for the graph inference from a walk for trees of bounded degree k then $P = NP$. This result still holds even if the alphabet size is restricted to $k+1$.

It is easy to show the following result:

Theorem 2.12 *Let $\epsilon < 2$. If there is a polynomial-time ϵ-approximation algorithm for the graph inference from a walk for (α, β)-caterpillars for some integers $\alpha, \beta \geq 1$, then $P = NP$.*

2.6 Concluding remarks

We have established a linear-time algorithm to produce from a string x the smallest tree realizing a walk for x. We have, however, shown that the problem turns to be NP-complete by putting the constraint that the maximum degree of trees is bounded by a constant. Furthermore, we have shown that the tree inference problem for trees of bounded degree k is still NP-complete even if the alphabet size is restricted to $k+1$ for any $k \geq 3$. On the other hand, the NP-completeness result still remains even if such trees are restricted to (1,1)-caterpillars. Moreover, we have seen that the NP-complete problem has no polynomial-time approximation scheme, and that the approximation ratio cannot be less than two unless P=NP. The following problems remain open:

(1) Is there any polynomial-time ϵ-approximation algorithm for the graph inference from a walk for trees of bounded degree three, for some $\epsilon \geq 2$?

(2) Let C be the class of (1,1)-caterpillars T with at most $O(\log b)$ hairs, where b is the length of the backbone of T. Is the graph inference from a walk for C solvable in polynomial time?

(3) Does the graph inference from a walk for $(1,1)$-caterpillars remain NP-complete even if the alphabet size is restricted to a constant?

(4) Let $\epsilon \geq 2$. Is there any polynomial-time ϵ-approximation algorithm for the graph inference from a walk for $(1,1)$-caterpillars?

(5) Is the graph inference from a walk for (α, β)-caterpillars solvable in polynomial time if either α or β is unbounded, or both are unbounded?

Chapter 3

Inferring a Graph from Partial Walks

In this chapter we consider the problem of inferring a graph from a finite number of partial walks instead of a single walk. For a finite set S of strings and a graph G, we say that G *realizes all partial walks for* S if, for each string x in S, the graph G realizes a partial walk for x. Let C be a class of graphs. We define the problem for C as follows:

Graph Inference from Partial Walks for C

`Instance`: A finite set S of strings over a finite alphabet Σ and a positive integer K.

`Question`: Is there a graph $G \in C$ with at most K edges such that G realizes all partial walks for S?

Obviously, the graph inference from partial walks is a natural extension of the graph inference from a walk.

3.1 Tree inference from partial walks

In this section we consider the *tree inference from partial walks* that is to find the smallest undirected tree realizing all partial walks for given strings. Notice that when

the number of given strings is one, the problem is equivalent to the tree inference from a single walk.

The following is the main result in this section:

Theorem 3.1 *The tree inference from partial walks is NP-complete. Furthermore, this problem remains NP-complete even if the following conditions hold simultaneously:*

1. *The alphabet size is restricted to three.*
2. *The maximum degree is bounded by three.*

Proof. It is easy to see that the tree inference from partial walks is in NP. First, we give a reduction from VERTEX COVER [GJ79]. Second, we modify the reduction so as to show the NP-hardness of the problem with the constraint on the alphabet size and the maximum degree.

Recall that VERTEX COVER is to decide if, given a graph $G = (V, E)$ and a positive integer K, there is a vertex cover of size at most K for G, that is, a subset $C \subseteq V$ with $|C| \leq K$ such that for each edge $\{u, v\} \in E$ at least one of u and v belongs to C. Let $G = (V, E)$ be a graph with $|V| = n$ and K be a positive integer. For G and K, We define an alphabet Σ as $\Sigma = V \cup \{a_0, a_1, \ldots, a_{\lceil n/2 \rceil}\} \cup \{b_1, b_2, \ldots, b_{n+1}\}$. In order to define a set S of strings over Σ, we introduce the following notations for strings:

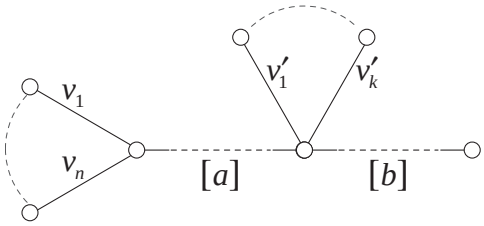
$$\begin{aligned} [a] &= a_{\lceil n/2 \rceil} \cdots a_1 a_0 a_1 \cdots a_{\lceil n/2 \rceil} \\ [b] &= b_1 \cdots b_{n+1} \ . \end{aligned}$$

Note that $[a]^R = [a]$. Then S consists of the following strings:

$$\begin{aligned} &\textit{base-string} : u[a][b] \quad \text{for } u \in V, \\ &\textit{edge-string} : u[a]v \quad \text{for } \{u, v\} \in E. \end{aligned}$$

Finally, let $K' = 2n + 2\lceil n/2 \rceil + 2 + K$. This transformation can be done in polynomial time. We claim that G has a vertex cover of size at most K if and only if there is a tree with at most K' edges which realizes all partial walks for S.

Suppose that G has a vertex cover C with $|C| \leq K$. For a subset $U = \{v'_1, \ldots, v'_k\}$ of V, let $T(U)$ be the tree in Fig. 3.1. It is obvious that $T(C)$ realizes all partial walks



$T(U)$

Figure 3.1: $V = \{v_1, \ldots, v_n\}$ and $U = \{v'_1, \ldots, v'_k\} \subseteq V$.

for S. It can be easily checked that T contains at most K' edges since $|C| \leq K$.

Conversely, suppose that there is a tree T with at most K' edges realizing all partial walks for S. Without loss of generality, we can assume that T is proper by the following fact:

Fact 3.1 *Let S be a finite set of strings and T be a tree realizing all partial walk for S. Then, for the $\rightarrow_F$-normal form of T, i.e., $\widehat{F}(T)$, the following statements are true:*

(1) $\widehat{F}(T)$ is proper.

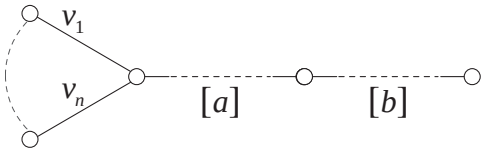
(2) $\widehat{F}(T)$ is not larger than T.

(3) $\widehat{F}(T)$ also realizes all partial walks for S.

This fact is trivial because, for trees T_1 and T_2 with $T_1 \rightarrow_F T_2$, if T_1 realizes a partial walk for a string x then so does T_2. Notice that if T is proper then any subgraph of

T is proper. It is easy to see that for each string x in S, any tree realizing a walk for x is isomorphic to a linear chain of label x. We first consider the base-strings, each of which contains exactly one $[a][b]$ as a substring.

Claim 3.1 *Let T_b be the tree in Fig. 3.2. Any proper tree with at most K' edges that realizes all partial walks for the base-strings is isomorphic to the tree T_b.*



T_b

Figure 3.2: $V = \{v_1, \ldots, v_n\}$.

Proof of Claim 3.1. It can be easily checked that if such a tree is not isomorphic to T_b then it contains at least $|[a][b]|+|[b]|+|V| = 3n+2\lceil n/2\rceil+3$ edges. This contradicts the assumption that the number of edges in T is at most K'. ∎

In a similar way, we can show the following:

Claim 3.2 *A tree T' is a proper tree with at most K' edges realizing all partial walks for S if and only if T' is isomorphic to the tree $T(C')$, where $C' \subseteq V$.*

Then we can assume that for some $C' \subseteq V$, the tree T is isomorphic to $T(C')$. It is obvious that $|C'|$ is at most K since T has at most K' edges. It should be clear that C' gives a vertex cover of G whose size has been shown at most K.

Next, we modify the reduction into another one to show that the tree inference from a walk remains NP-complete even if the size of alphabet is three and simultaneously trees are of bounded degree three. Let $\Sigma = \{0, 1, \#\}$. For convenience, we assume

that $V = \{0, \ldots, n-1\}$. For a nonnegative integer i, we denote by i_j the jth bit of the binary representation of i such that $i = i_0 2^0 + i_1 2^1 + \cdots + i_{m-1} 2^{m-1}$ for some $m \geq \lfloor \log i \rfloor + 1$. Let $\overline{i_j} = 1$ if $i_j = 0$ and $\overline{i_j} = 0$ otherwise. For a pair (h, i) of integers with $0 \leq i \leq 2^h - 1$, the strings $b_1(h,i), b_2(h,i)$ and $b_3(h,i)$ are defined as follows:

$$
\begin{aligned}
b_1(h,i) &= \#i_0\#i_1 \cdots \#i_{h-1}. \\
b_2(h,i) &= \#i_0\overline{i_0}\#i_1\overline{i_1} \cdots \#i_{h-1}\overline{i_{h-1}}. \\
b_3(h,i) &= \#i_0\overline{i_0}i_0\#i_1\overline{i_1}i_1 \cdots \#i_{h-1}\overline{i_{h-1}}i_{h-1}.
\end{aligned}
$$

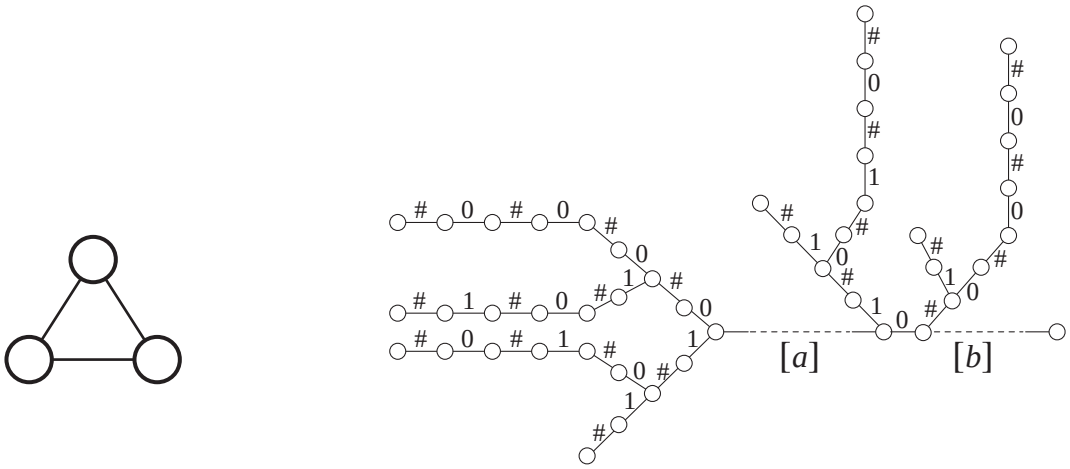
Let $q = \lceil \log n \rceil$. Using these strings, we make the strings $[i]$ for $0 \leq i \leq 2^q - 1$, $[a]$ and $[b]$ as follows:

$$
\begin{aligned}
[i] &= b_1(q,i) \text{ for } 0 \leq i \leq 2^q - 1. \\
\tilde{a} &= \prod_{i=0}^{2^q-1} \#0101 b_2(q,i). \\
[a] &= \tilde{a}\#0\#\tilde{a}^R. \\
[b] &= \prod_{i=0}^{2^q-1} 01010101 b_3(2q,i)\#.
\end{aligned}
$$

Note that $|[a]| = 2^{q+1}(3q+5)+3$ and $|[b]| = 2^q(8q+9)$. Let S be the set of strings as follows:

$$
\begin{aligned}
&\textit{base-string}: && [i][i][a][b] && \text{for } i \in V. \\
&\textit{branch-string}: && [i][a][i]^R && \text{for } 0 \leq i \leq 2^q - 1. \\
&\textit{edge-string}: && [i][i][a][j]^R[j]^R && \text{for } \{i,j\} \in E.
\end{aligned}
$$

Finally, let $K' = 2q(n+K) + 2^q(14q+27) - 6$. This transformation can be done in polynomial time. We claim that G has a vertex cover of size at most K if and only if there is a tree with at most K' edges which realizes all partial walks for S (see Fig.3.3). This claim can be proved in a similar way of the case that any restriction is not put on the size of alphabet. We leave it for the reader to verify the claim.



G T

Figure 3.3: The tree T realizes all partial walks for the strings produced by the second reduction from the graph G.

□

We can show that the number three of colors in Theorem 3.1 is optimal by the following result:

Theorem 3.2 *The tree inference from partial walks is solvable in linear time if the size of alphabet is at most two.*

Proof. Let Σ be an alphabet of size at most two and x be a string over Σ. We say that x is *alternate* if the ith bit of x, denoted by x_i, is different from x_{i+1}. By Lemma 2.3, the smallest tree realizing a walk for x is the linear chain l such that the label of l is alternate. We denote the alternate string for x by $a(x)$. For a finite set S of strings over Σ, let $a(S) = \{a(y) \mid y \in S\}$. It is obvious that $a(S)$ can be produced by applying the linear-time algorithm for the tree inference from a walk, provided in Section 2.3. It can be done in time linear in the total length of the strings in S.

If the longest string of $a(S)$, denoted by l, is unique, then l is the label of the shortest linear chain realizing all partial walks for S. Otherwise, there exist two distinct, longest

strings with length $2k+1$ for some k. In this case, the label of the shortest linear chain is the alternate string with length $2k$. □

Next, we show that the tree inference from partial walks has no polynomial-time approximation scheme unless P=NP by proving that the problem is MAX SNP-hard [ALM+92].

Theorem 3.3 *The tree inference from partial walks is MAX SNP-hard.*

Proof. Recall that VERTEX COVER is denoted by k-DEGREE VERTEX COVER when graphs are restricted to graphs of bounded degree k without any self-loop. As mentioned in the proof of Theorem 2.9, it is known that 4-DEGREE VERTEX COVER is MAX SNP-complete [PY91, Pap94]. Let f be the reduction in the proof of Theorem 3.1 from 4-DEGREE VERTEX COVER. Then the first condition is satisfied with $\alpha = 15$ since $\lceil n/5 \rceil \leq opt(G)$, where $opt(G)$ is the size of minimum covers of G.

Next, we define an algorithm g as follows: We can assume that a feasible solution of the tree inference from partial walks is a proper tree which realizes all partial walks for given strings. If a feasible solution s_2 has at most $3n + 2\lceil n/2 \rceil + 2$ edges, then s_2 is a tree isomorphic to $T(C)$ for some $C \subseteq V$, which is defined in the proof of Theorem 3.1. In the case, the algorithm g returns C. Otherwise, g returns V. Then it is trivial that the second condition holds with $\beta = 1$. □

3.2 Linear chain inference from partial walks

We here consider the *linear chain inference from partial walks*, i.e., the problem of finding the shortest linear chain realizing all partial walks for a finite set of strings.

Example 3.1 *Let* $S = \{abbaabcdeedc, cbeddeedc, ebceee\}$. *The graph in Fig. 3.4 is the shortest linear chain which realizes partial walks for* S.

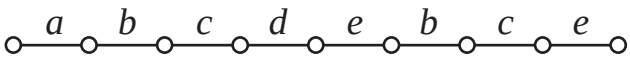


Figure 3.4:

Since if a directed linear chain l realizes a walk for a string x then the label of the directed linear chain l is x, the graph inference from partial walks for the class of directed linear chains is essentially equivalent to the shortest common superstring problem, for which the NP-completeness result was given by Gallant et at. [GMS80]. Thus, the graph inference from partial walks for the class of directed linear chains is NP-complete.

Recall that the linear chain inference from a single walk is shown to be solvable in polynomial time by Aslam and Rivest [AR90] and Raghavan [Rag94] (see Section 2.2). Unfortunately, the linear chain inference from partial walks turns to be intractable in the same way as inferring trees.

Theorem 3.4 *The linear chain inference from partial walks is NP-complete even if the size of alphabet is restricted to three.*

Proof. We give a reduction from the shortest common superstring problem [GMS80], where the shortest common superstring problem is to decide if, given a finite set S of strings over a finite alphabet Σ and a positive integer K, there is a superstring for S with length at most K, that is, a string $s \in \Sigma^*$ with $|s| \leq K$ such that each string $x \in S$ is a substring of s. It is known that the problem is NP-complete even if $|\Sigma| = 2$ [GMS80]. Let S be a finite set of strings over the alphabet $\Sigma = \{0, 1\}$ and K be a positive integer. We first define an alphabet Σ' as $\Sigma' = \Sigma \cup \{\#\}$, where $\#$ is a new symbol not in Σ. For a string $b = b_1 b_2 \cdots b_m$ with $b_1, b_2, \ldots, b_m \in \Sigma$, we create a string

$$b' = \prod_{i=1}^{m} (01\#b_i\#)01.$$

Let S' be the set of the strings b' for all $b \in S$. Finally, let $K' = 5K + 2$. This transformation can be done in polynomial time.

Note that the label of a linear chain realizing a walk for a string $x' \in S'$ is only x' (see Theorem 5 of [AR90]). It is clear that there is a superstring s for S with $|s| \leq K$ if and only if all partial walks for S' are realized in a linear chain with K' edges or less.

□

The number three of colors in Theorem 3.4 is optimal because of the following result:

Corollary 3.1 *The linear chain inference from partial walks is solvable in linear time if the size of alphabet is at most two.*

This is a trivial consequence from the proof of Theorem 3.2 since, for any set S of strings over at most two colors, the smallest tree realizing all partial walks for S is in the form of a linear chain.

By the fact that the shortest common superstring problem is MAX SNP-hard [BJL+94] and the fact that the reduction in the proof of Theorem 3.4 is an L-reduction, the following holds:

Theorem 3.5 *The linear chain inference from partial walks is MAX SNP-hard.*

By this theorem and the result due to Arora et al. [ALM+92], there is no polynomial-time approximation algorithm for the linear chain inference from partial walks unless P=NP.

3.3 Polynomial-time approximation algorithms

In two previous sections, we have shown that both of the tree inference from partial walks and the linear chain inference from partial walks are NP-complete and MAX

SNP-hard, while the graph inference problems from a single walk for such graphs have been shown solvable in polynomial time.

In this section, we construct polynomial-time approximation algorithms for both NP-complete problems. We will show that an approximation algorithm for the problem on trees can be constructed by employing an approximation algorithm for the minimum common supertree problem [YM93]. The minimum common supertree problem can be shown NP-complete because the proof in Theorem 3.1 is also the proof of the NP-hardness of the problem. For the minimum common supertree problem, a polynomial-time approximation algorithm was given by Yamaguchi and Miyano [YM93].

On the other hand, we also show that an approximation algorithm on linear chains can be developed by employing an approximation algorithm for the shortest common superstring with flipping, for which polynomial-time approximation algorithms were given by Jiang et al. [JLD92].

3.3.1 On trees

The tree inference from partial walks has the following approximation algorithm that is analyzed in terms of the *compression* in the constructed tree T, that is, in terms of $k - l$, where k is the total length of given strings and l is the number of edges of the tree T.

Theorem 3.6 *There is a polynomial-time approximation algorithm to find a tree T realizing all partial walks for a set finite S of strings such that $C \geq C_m/(|S| - 1)$, where C is the compression in T and C_m is the maximum compression for S.*

In approximately solving the tree inference from partial walks, the observation in the following lemma is a key to our approach.

Lemma 3.1 *Let S be a finite set of strings and T be the smallest tree realizing all partial walks for S. Then, for each string $x \in S$, the smallest tree realizing a walk for x is a subgraph of T.*

This lemma is trivial from Fact 3.1. For a finite set R of trees, a tree T is called a *supertree for* R if for each tree $t \in R$, the tree t is a subgraph of T. Let l_x be a linear chain of label a string x. Recall that $\widehat{F}(l_x)$, i.e., the $\rightarrow_F$-normal form of l_x, is isomorphic to the smallest tree realizing a walk for x (see Lemma 2.3). For a finite set S of strings, we denote the set $\{\widehat{F}(l_x) \mid x \in S\}$ by $\widehat{F}(S)$. Given a finite set S of strings, if we could find the smallest supertree for $\widehat{F}(S)$, it would be the required tree in the tree inference from partial walks by Lemma 3.1. Though the problem of finding the smallest supertree is easily seen to be NP-complete from the proof of Theorem 3.1, there is an approximation algorithm which, given a finite set R of trees, constructs a supertree T for R satisfying $C \geq C_m/(|R|-1)$, where C is the compression in T and C_m is the maximum compression for R [YM93]. Thus, by employing the algorithm, the algorithm in Theorem 3.6 can be given. Notice that for each $x \in S$ if the smallest tree realizing a walk for x is isomorphic to a linear chain of label x, we cannot expect any merit by constructing $\widehat{F}(S)$ in the algorithm of Theorem 3.6.

3.3.2 On linear chains

Fortunately, we can also construct polynomial-time approximation algorithms for the linear chain inference from partial walks in the same way as the tree inference from partial walks because we have a lemma corresponding to Lemma 3.1.

Lemma 3.2 *Let l be the shortest linear chain realizing all partial walks for a set S of strings. Then, for each string $x \in S$, the shortest linear chain realizing a walk for x is a subgraph of l.*

This can be easily shown by using binary relations on strings which was introduced in [AR90]. A string s is called a *superstring with flipping for a set S of strings* if for each string $x \in S$, either x or x^R is a substring of s. In a similar way, the compression in a superstring with flipping can be defined. Since there is an approximation algorithm which finds a superstring s with flipping for a given set S of strings such that $C \geq C_m/2$ where C is the compression in s and C_m is the maximum compression for S [JLD92], the following approximation algorithm for the linear chain inference from partial walks can be given:

Theorem 3.7 *There is a polynomial-time approximation algorithm to find a linear chain l realizing all partial walks for a set S of strings such that $C \geq C_m/2$, where C is the compression in l and C_m is the maximum compression for S.*

Jiang et al. [JLD92] also developed an approximation algorithm which constructs a superstring s with flipping with length at most three times *opt*, where *opt* is the shortest length.

Theorem 3.8 *There is a polynomial-time algorithm to find a linear chain with length at most three times opt(S) which realizes all partial walks for a finite set S of strings, where opt(S) is the length of the shortest linear chain realizing all partial walks for S.*

3.4 Concluding remarks

We have shown that the tree inference from partial walks is NP-complete. Furthermore, we have proved that there is no polynomial-time approximation scheme unless P=NP. Fortunately, we have also obtained a polynomial-time approximation algorithm for the problem. The approximation ratio is analyzed in terms of the compression in the tree produced by the algorithm. Since the ratio is not bounded by a constant, it is open if

there is a polynomial-time approximation algorithm with ratio bounded by a constant. The performance of approximation algorithms for the tree inference from partial walks should be also evaluated by the ratio of the number of edges in the produced tree to the minimum number of edges because the problem is the minimization problem of the number of edges in a tree realizing all partial walks for a given set of strings. However, we have not given any result on the ratio in terms of the number of edges. It also remains open if the tree inference from partial walks has a polynomial-time approximation algorithm such that the number of edges in the produced tree is bounded by a constant times the minimum.

We have also discussed the linear chain inference from partial walks and shown the NP-completeness and the MAX SNP-hardness, which implies that there is no polynomial-time approximation scheme unless P=NP [ALM+92]. We have presented a polynomial-time approximation algorithm that the length of the linear chain produced is bounded by three times the minimum. The algorithm employs polynomial-time approximation algorithms for the problem of the shortest superstring with flipping given by Jiang et al. [JLD92]. Their approximation algorithms were developed by slightly modifying polynomial-time approximation algorithms for the shortest common superstring problem [TU88, Tur89, BJL+94]. Especially, Blum et al. [BJL+94] devised a polynomial-time approximation algorithm that the length of the superstring produced is bounded by three times the minimum. Better ratios of polynomial-time approximation algorithms for the shortest common superstring problem have been reported by Teng and Yao [TY93], Czumaj et al. [CGPR94], Kosaraju et al. [KPS94] and Armen and Stein [AS94]. The ratios are $26/9 \approx 2.889$, $17/6 \approx 2.833$, $176/63 \approx 2.794$ and 2.75, respectively. By exploiting such new approximation algorithms, we could develop polynomial-time approximation algorithms with better ratio for the linear chain

inference from partial walks.

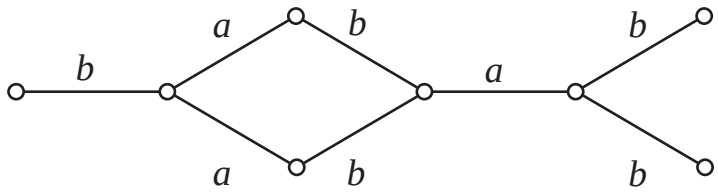
Chapter 4

Realizing a Walk on a Graph

In this chapter we discuss the walk realizability problem, in which we are given a graph G in addition to a string x, and asked if G realizes a walk for x. Obviously, this problem is closely related to the graph inference from a walk. The aim of considering the walk realizability problem is to see the complexity of deciding if there is a walk for a string x on a graph G when the string x and the graph G are given simultaneously.

Here, we shall see an example of the problem.

Example 4.1 *Let $x = bbbabbbbababba$ and G be the graph in Fig. 4.1. Does G realize a walk for x? The answer is "yes."*



G

Figure 4.1:

4.1 Walk realizability problem

Let C be a class of graphs. We recall the definition of the walk realizability problem for C below:

Walk Realizability Problem for C

`Instance:` A string x over a finite alphabet Σ and a graph $G \in C$.

`Question:` Does G realize a walk for x?

A partial walk on a graph G is a path in G, which is not required to include all edges of G. In the same way as the walk realizability problem, we define the *partial walk realizability problem* as the problem of deciding if a graph G realizes a partial walk for a string x. We can see that the partial walk realizability problem is complete for NLOG by an easy reduction from the threadable maze problem [Sav70].

We first consider the walk realizability problem for graphs of bounded degree two, i.e., cycles and linear chains.

Lemma 4.1 *(1) The walk realizability problem for cycles is in NLOG.*

(2) The walk realizability problem for linear chains is in NLOG.

Proof. (1) Let $G = (V, E, c)$ be a cycle and $x = x_1 x_2 \cdots x_n$ be a string with $x_i \in \Sigma$ for $1 \leq i \leq n$. We consider the following algorithm:

Let M_l and M_r be markers on nodes.

Guess a node v_0 and put M_l and M_r on v_0.

Let RET be a variable having "yes" or "no."

RET:= "no."

Execute the following for $1 \leq i \leq n$ in order:

1. Guess a node v_i and check if $\{v_{i-1}, v_i\} \in E$ and $c(\{v_{i-1}, v_i\})$ is x_i.
   If not, stop and return "no."
2. Move M_l (M_r) to the leftmost (rightmost) node among the visited nodes $v_0, v_1, \ldots, v_i$.
3. RET:="yes" if the two markers M_l and M_r are put on the same node.

Return RET.

It is clear that G realizes a walk for x if and only if the algorithm returns "yes". Obviously, this algorithm uses only log-space for keeping the nodes on which the two markers are put.

(2) By a similar argument to the walk realizability problem for cycles, we can show the result for linear chains. □

In order to investigate the walk realizability for more complex graphs, we define the linear chain decomposition of a graph. Let G be an undirected connected graph. The *linear chain decomposition of a graph* G is simply defined by splitting every node v of degree $k \geq 3$ into k new nodes of degree one. Formally, the linear chain decomposition of G is obtained by repeating the following procedure until there is no node with degree greater than or equal to three: Let v be a node with degree $k \geq 3$ and let $u_1, \ldots, u_k$ be its adjacent nodes. Then by replacing v with k new nodes $v_1, \ldots, v_k$, we put new

edges $\{u_1, v_1\}, \ldots, \{u_k, v_k\}$ instead of the edges $\{u_1, v\}, \ldots, \{u_k, v\}$, where the color of $\{u_i, v_i\}$ is the same as that of $\{u_i, v\}$ for $i = 1, \ldots, k$. Obviously, the degree of v_i is one. After the above replacement, the resulting graph is a collection of linear chains. The *size* of the linear chain decomposition of G, denoted by $\mathrm{lcd}(G)$, is the number of components of the resulting graph.

Note that the following equality holds:

$$\mathrm{lcd}(G) = \frac{1}{2} \cdot \sum_{v \in V \text{ with } \deg(v) \geq 3} \deg(v),$$

where $\deg(v)$ is the degree of the node v.

Let C be a class of graphs and $f(m)$ be a function on the nonnegative integers. The class C is said to be $O(f(m))$-*decomposable* if there exists a constant $c > 0$ such that $\mathrm{lcd}(G) \leq c \cdot f(m)$ for every graph $G = (V, E, c) \in C$, where m is the number of edges in G. Trivially, any class C of graphs is $O(m)$-decomposable.

Theorem 4.1 *The walk realizability problem for any $O(1)$-decomposable class of undirected graphs is in NLOG.*

Proof. Let $G = (V, E, c)$ be a connected graph in an $O(1)$-decomposable class. The number of the linear chains produced by the linear chain decomposition of G is $O(1)$ from the definition of the decomposition. In order to keep each of the linear chains, it is sufficient to store the leftmost and rightmost nodes of the linear chain, which requires only log-space. Thus, by applying an algorithm similar to that in Lemma 4.1, we can construct a nondeterministic algorithm in log-space. □

In the same way, we can define the linear chain decomposition of a directed graph. It is easy to see that the walk realizability problem for any $O(1)$-decomposable class of directed graphs is also in NLOG.

4.2 Realizing walks on trees

In this section we consider the walk realizability for some kinds of undirected trees. The walk realizability problem for any class of directed trees is trivial. Hereafter a tree means an undirected tree.

Theorem 4.2 *The walk realizability problem for any $O(\log m)$-decomposable class of trees is solvable in polynomial time.*

Proof. Let $T = (V, E, c)$ be a tree in an $O(\log m)$-decomposable class of trees and $x = x_1 x_2 \cdots x_n$ be a string of colors, where x_i is a color for $1 \leq i \leq n$. It is sufficient to show that there is a polynomial time algorithm to decide if there is a partial walk for x on T including all nodes of degree one in T. Let $l_1, l_2, \ldots, l_{m'}$ be the nodes of degree one in T. Notice that $m' \leq \mathrm{lcd}(T) \leq O(\log m)$, where m is the number of edges in T. We define a function $\phi : V \times \{1, 2, \ldots, |x|\} \times V \times \{0,1\}^{m'} \to \{0,1\}$ as follows:

$$\phi(v, i, v', \tilde{l}_1, \tilde{l}_2, \ldots, \tilde{l}_{m'}) = \begin{cases} 1 & \text{if there is a partial walk } w \text{ for } x_1 x_2 \cdots x_i \text{ from } v \text{ to } v' \text{ such that, for each } 1 \leq k \leq m', w \text{ includes } l_k \text{ if } \tilde{l}_k = 1, \\ 0 & \text{otherwise.} \end{cases}$$

It is not so hard to see that there exists a polynomial time algorithm for producing a table representing the function ϕ. Clearly, there is a pair of nodes v and v' of T such that $\phi(v, |x|, v', 1, 1, \ldots, 1) = 1$ if and only if T realizes a partial walk for x passing through all the nodes $l_1, l_2, \ldots, l_{m'}$. Since such a pair of nodes can be found in polynomial time by looking up the table, it can be decided in polynomial time if T realizes a walk for x. □

The walk realizability for any $O(\log n)$-decomposable class of trees would not be P-complete since we can show that the problem is solvable in $O(\log |x|)$ time on the priority CRCW PRAM with polynomial-processors.

We next consider the class of trees, which is $O(m)$-decomposable. If a node v is not proper, we say that v is non-proper.

Theorem 4.3 *The walk realizability problem for the class of trees is NP-complete even if the number of non-proper nodes in a tree is exactly one.*

Proof. Obviously, the problem is in NP. We reduce the satisfiability problem (SAT) to the problem. The problem SAT is to decide if, given a collection C of clauses on a finite set U of variables, there is a truth assignment for U that satisfies all the clauses in C. Let $U = \{u_0, u_1, \ldots, u_{n-1}\}$ be a finite set of variables and $C = \{c_0, c_1, \ldots, c_{m-1}\}$ be a collection of clauses on U. From U and C, we must construct a finite alphabet Σ, a tree $T = (V, E, c)$, where $c : E \to \Sigma$, and a string over Σ such that T realizes a walk for x if and only if C is satisfiable.

We set $\Sigma = \{\#\} \cup U \cup C$, and define T as the tree of the form in Fig. 4.2.

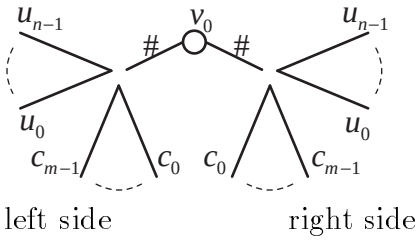


Figure 4.2: The node v_0 is a unique non-proper node.

For a variable $u \in U$, by $C_p(u)$ ($C_n(u)$, resp.) we denote the set of clauses containing positive (negative, resp.) literals of u. In order to construct the string x, we first define strings $u_i^{(p)}$ and $u_i^{(n)}$ for each $i = 0, 1, \ldots, n-1$ and SWEEP:

$$u_i^{(p)} = \#u_i u_i \prod_{c_k \in C_p(u_i)} (c_k c_k)\#$$

$$u_i^{(n)} = \#u_i u_i \prod_{c_k \in C_n(u_i)} (c_k c_k)\#$$

$$\text{SWEEP} \quad = \quad \# \prod_{k=1}^{m} (c_k c_k) \#$$

The string x is defined by

$$x \quad = \quad \prod_{i=0}^{n-1} (u_i^{(p)} u_i^{(n)}) \text{SWEEP}$$

The reduction can be done in polynomial time. We claim that C is satisfiable if and only if T realizes a walk for x.

Let $S = \{u_i^{(p)}, u_i^{(n)}, \text{SWEEP} \mid i = 0, 1, \ldots, n-1\}$. Note that the strings of S cover the string x completely without any overlaps between strings in S. The following is obvious:

Fact 4.1 *For each string $s \in S$, the tree T realizes exactly two partial walks for s, which are closed with v_0; One is realized in the right side of T and another is in the left side.*

First, suppose that C is satisfiable, i.e., there is a truth assignment $f : U \rightarrow \{\textbf{true}, \textbf{false}\}$. Let w be a partial walk for x, which will be shown a walk for x, such that for each $i = 0, 1, \ldots, n-1$, the following are satisfied:

1. The subwalk $w_{u_i^{(p)}}$ of w for $u_i^{(p)}$ is on the right side and the subwalk $w_{u_i^{(n)}}$ of w for $u_i^{(n)}$ is on the left side if $f(u_i) =$**true**. Otherwise, $w_{u_i^{(n)}}$ is in the right side and $w_{u_i^{(p)}}$ is in the left side.

2. The subwalk of w for SWEEP is on the left side.

Such a partial walk w is unique. Obviously, all the edges labeled a variable in U are traversed by w. We can see that all the edges labeled a clause in C in the right side of T are also traversed by w since

$$C = \bigcup_{f(u_i)=\textbf{true}} C_p(u_i) \bigcup \bigcup_{f(u_i)=\textbf{false}} C_n(u_i).$$

Trivially, all the edges labeled a clause in C in the left side are traversed by the subwalk of w for SWEEP. Thus w is a walk for x on T.

Conversely, suppose that T realizes a walk w for x. The strings in S including a variable $u_i \in U$ are only $u_i^{(p)}$ and $u_i^{(n)}$. On the other hand, T has exactly two edges labeled u_i where one is in the right side of T and another is in the left side. Since w traverses the two edges, one of the following (1) and (2) holds:

(1) The subwalk of w for $u_i^{(p)}$ is on the right side and that for $u_i^{(n)}$ of w is on the left side.

(2) The subwalk of w for $u_i^{(n)}$ is on the right side and that for $u_i^{(p)}$ of w is on the left side.

Recall that the subwalk of w for SWEEP must be on either the right side or the left side. Without loss of generality, we can assume that the subwalk of w for SWEEP is on the left side. For each $i = 0, \ldots, n-1$, we define the string $\tilde{u}_i$ as follows:

$$\tilde{u}_i = \begin{cases} u_i^{(p)} & \text{if the subwalk of } w \text{ for } u_i^{(p)} \text{ is on the right side of } T, \\ u_i^{(n)} & \text{otherwise.} \end{cases}$$

Then, we can see that the edges labeled a clause in C in the right side of T are traversed by the subwalks for $\tilde{u}_i$ for $0 \leq i \leq n-1$. We here construct a function $\tilde{f} : U \to \{\mathbf{true}, \mathbf{false}\}$ such that

$$\tilde{f}(u_i) = \begin{cases} \mathbf{true} & \text{if } \tilde{u}_i = u_i^{(p)}, \\ \mathbf{false} & \text{otherwise.} \end{cases}$$

It is clear that $\tilde{f}$ is a truth assignment of C, namely, C is satisfiable. □

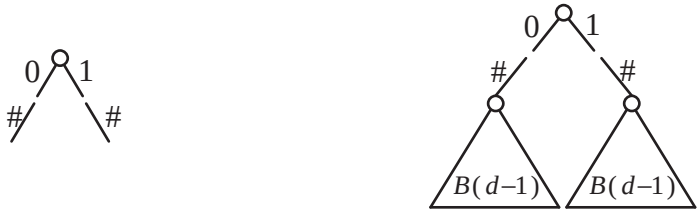
When the number of non-proper nodes in a tree T is restricted to zero, the walk realizability problem is solvable in polynomial time by applying the linear-time algorithm for the tree inference from a walk, provided in Section 2.3, since the tree T

is proper. Thus, the restriction with respect to the number of non-proper nodes in Theorem 4.3 is optimal.

Theorem 4.4 *The walk realizability problem for the class of trees is NP-complete even if the following conditions hold simultaneously:*

1. *The number of non-proper nodes is exactly one.*
2. *The degree-bound is three.*
3. *The alphabet size is three.*

Proof. We modify the reduction in the proof of Theorem 4.3. Let $U = \{u_0, u_1, \ldots, u_{n-1}\}$ be a finite set of variables and $C = \{c_0, c_1, \ldots, c_{m-1}\}$ be a collection of clauses on U. Let $\Sigma = \{0, 1, \#\}$. To construct a tree T, we create trees $B(d)$ and $\widetilde{B}(d)$ for each integer $d \geq 1$. First, $B(d)$ is defined recursively in the way of Fig. 4.3. Second, the



(a) $B(1)$ (b) $B(d)$

Figure 4.3:

tree $\widetilde{B}(d)$ is defined as the smallest subgraph of $B(\lceil \log d \rceil)$ that realizes all partial walks for

$$\{ \prod_{j=1}^{\lceil \log d \rceil} (b(i,j)\#) \mid i = 0, 1, \ldots, d-1 \}.$$

Note that such a partial walk on $B(\lceil \log d \rceil)$ must start at the top node of $B(\lceil \log d \rceil)$ and be unique. For example, $\widetilde{B}(6)$ is isomorphic to the tree in Fig. 4.4. The tree T is defined as the tree of the form in Fig. 4.5.

For $u \in U$, let $C_p(u)$ and $C_n(u)$ be the sets of clauses defined in the proof of Theorem 4.3. For a variable $u_i \in U$ and a clause $c_k \in C$, by code(u_i) and code(c_k) we denote the strings

$$\prod_{j=1}^{\lceil \log n \rceil} (b(i,j)\#) \quad \text{and} \quad \prod_{j=1}^{\lceil \log m \rceil} (b(k,j)\#),$$

respectively. We define the strings $u_i^{(p)}$ and $u_i^{(n)}$ for each $i = 0, 1, \ldots, n-1$ and SWEEP as follows:

$$\begin{aligned} u_i^{(p)} &= \#0\#\text{code}(u_i)(\text{code}(u_i))^R\#01\# \prod_{c_k \in C_p(u_i)} (\text{code}(c_k)(\text{code}(c_k))^R)\#1\# \\ u_i^{(n)} &= \#0\#\text{code}(u_i)(\text{code}(u_i))^R\#01\# \prod_{c_k \in C_n(u_i)} (\text{code}(c_k)(\text{code}(c_k))^R)\#1\# \\ \text{SWEEP} &= \#1\# \prod_{k=0}^{m-1} (\text{code}(c_k)(\text{code}(c_k))^R)\#1\# \end{aligned}$$

The string x is defined as follows:

$$x = \prod_{i=1}^{n} (u_i^{(p)} u_i^{(n)}) \text{SWEEP}$$

The reduction can be done in polynomial time. We claim that C is satisfiable if and only if T realizes a walk for x. The proof is not hard and left to the reader. □

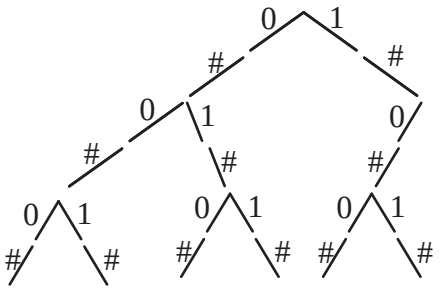


Figure 4.4: $\widetilde{B}(6)$.

This restriction is optimal with respect to the degree bound. If the degree bound of a tree is two then such a tree is of the form of a linear chain. Recall that the walk

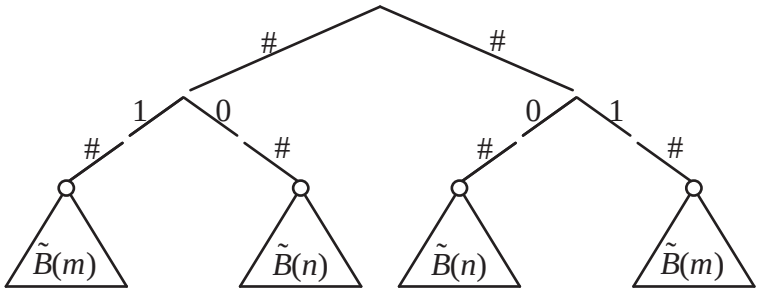


Figure 4.5: T.

realizability problem for linear chains has been shown to be in NLOG (see Lemma 4.1 (2)). The optimality with respect to the alphabet size in the walk realizability problem for the class of trees is settled as follows:

Theorem 4.5 *The walk realizability problem for the class of* $(1,1)$*-caterpillars is NP-complete even if the alphabet size is two.*

Obviously, if the alphabet size is one, the walk realizability for any class of trees is solvable in linear time. Thus, the number two of the alphabet size in Theorem 4.5 is optimal. Notice that the class of $(1,1)$-caterpillars is a class of trees of bounded degree three and also $O(m)$-decomposable.

Proof. We first give a reduction without any restriction on the alphabet size and then modify it so that the alphabet size is two. A basic idea of the reduction is the same as the reduction in the proof of Theorem 4.3. Let $U = \{u_0, u_1, \ldots, u_{n-1}\}$ be a finite set of variables and $C = \{c_0, c_1, \ldots, c_{m-1}\}$ be a collection of clauses on U. Let $\Sigma = \{\#, a\} \cup U \cup C$. We define T as the tree in Fig. 4.6. The strings $u_i^{(p)}$ and $u_i^{(n)}$ for $i = 0, 1, \ldots, n-1$ are defined as follows:

$$u_i^{(p)} = \#(aa)^{i+1} u_i u_i (aa)^{(n-i-1)+m} \prod_{c_k \in C_p(u_i)} (c_k c_k (aa)^m)(aa)^n \#$$

$$u_i^{(n)} = \#(aa)^{i+1} u_i u_i (aa)^{(n-i-1)+m} \prod_{c_k \in C_n(u_i)} (c_k c_k (aa)^m)(aa)^n \#$$

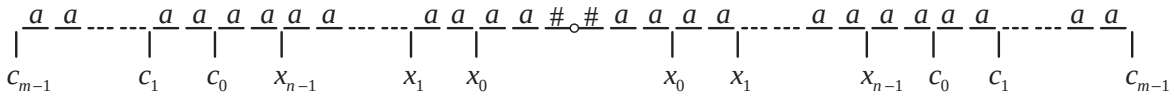


Figure 4.6:

$$\text{SWEEP} = \#(aa)^n \prod_{k=0}^{m-1} (aac_kc_k)(aa)^{m+n}\#,$$

where the definition of $C_p(u)$ ($C_n(u)$) is as in the proof of Theorem 4.3. The string x is defined by

$$x = \prod_{i=0}^{n-1} (u_i^{(p)} u_i^{(n)})\text{SWEEP}.$$

The (1,1)-caterpillar T and the string x can be constructed in polynomial time. It is not hard to show that C is satisfiable if and only if T realizes a walk for x.

We next give another reduction. Let $\Sigma' = \{0,1\}$ and s_2, s_3, s_4 be the strings over Σ' defined as follows:

$$\begin{aligned} s_2 &= 01010 \\ s_3 &= 0101010 \\ s_4 &= 010101010 \end{aligned}$$

Let $\tilde{m}$ be the smallest even integer greater than or equal to $\lceil \log m \rceil$. We define the string x as follows:

$$\text{code}(u_i) = \prod_{j=1}^{\lceil \log n \rceil} (b(i,j)^2 0).$$

$$\text{code}(c_k) = 000 \prod_{j=1}^{\tilde{m}} (b(k,j)^2 0)11.$$

$$\begin{aligned} u_i^{(p)} &= s_4 (s_2 s_2 0^{\lceil \log n \rceil})^i s_2 s_2 \text{code}(u_i) (s_2 s_2 0^{\lceil \log n \rceil})^{n-i-1} \\ &\quad s_3 \prod_{c_k \in C_p(u_i)} ((s_2 s_2 0^{\tilde{m}})^m \text{code}(c_k)) (s_2 s_2 0^{\tilde{m}})^m s_3 (0^{\lceil \log n \rceil} s_2 s_2)^n s_4. \end{aligned}$$

$$u_i^{(n)} = s_4 (s_2 s_2 0^{\lceil \log n \rceil})^i s_2 s_2 \text{code}(u_i) (s_2 s_2 0^{\lceil \log n \rceil})^{n-i-1}$$

$$s_3 \prod_{c_k \in C_n(u_i)} ((s_2 s_2 0^{\tilde{m}})^m \mathrm{code}(c_k))(s_2 s_2 0^{\tilde{m}})^m s_3 (0^{\lceil \log n \rceil} s_2 s_2)^n s_4.$$

$$\mathrm{SWEEP} = s_4 (s_2 s_2 0^{\lceil \log n \rceil})^n s_3 \prod_{c_k \in C} (s_2 s_2 \mathrm{code}(c_k))(0^{\tilde{m}} s_2 s_2)^m s_3 (0^{\lceil \log n \rceil} s_2 s_2)^n s_4.$$

$$x = \prod_{i=0}^{n-1} (u_i^{(p)} u_i^{(n)}) \mathrm{SWEEP}.$$

Let T' be the tree in Fig. 4.7. We define T as the tree obtained by identifying the node v_0 of T' with the node corresponding to v of a copy of T'.

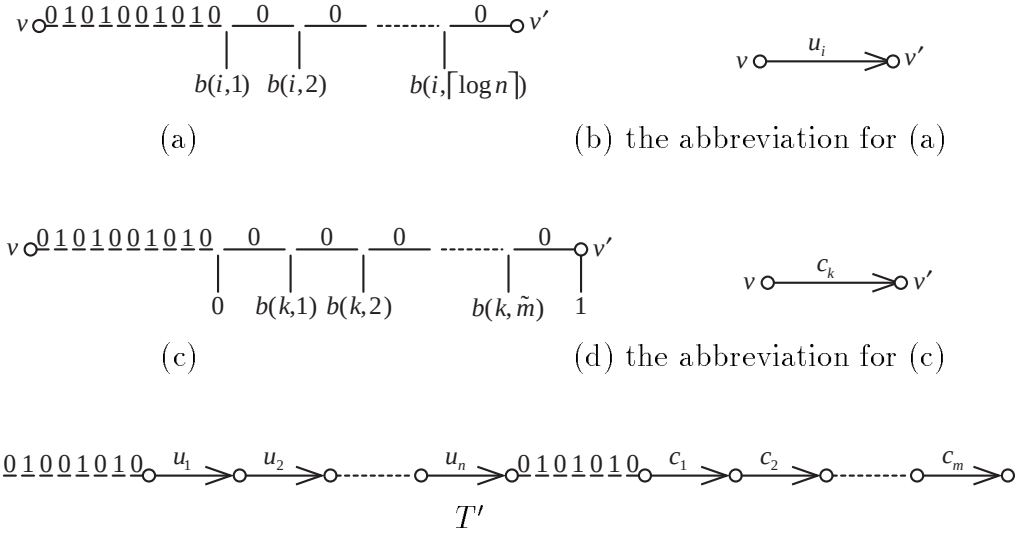


Figure 4.7:

This reduction can be also done in polynomial time. In the reduction, C is satisfiable if and only if T realizes a walk for x. The verification of it is left to the reader. □

Note that the walk realizability problem for any class of $(1,1)$-caterpillars with at most $O(\log m)$ hairs is solvable in polynomial time, where m is the number of edges, because of Theorem 4.2.

A *ladder* is an undirected graph $G = (V, E, c)$ with $V = \{v_{i,j} \mid i = 1,2, j = 1,\ldots,n\}$ and $E = \{\{v_{i,j}, v_{i,j+1}\} \mid i = 1,2, j = 1,\ldots,n-1\} \cup \{\{v_{1,j}, v_{2,j}\} \mid j = 1,\ldots,n\}$. A

mesh is an undirected graph $G = (V, E, c)$ with $V = \{v_{i,j} \mid 1 \leq i, j \leq n\}$ and $E = \{\{v_{i,j}, v_{i,j+1}\} \mid 1 \leq i \leq n, 1 \leq j \leq n-1\} \cup \{\{v_{i,j}, v_{i+1,j}\} \mid 1 \leq i \leq n-1, 1 \leq j \leq n\}$. The following results can be easily shown by modifying the reduction in the proof of Theorem 4.5.

Corollary 4.1 *1. The walk realizability problem for the class of ladders is NP-complete even if the alphabet size is three.*

2. The walk realizability problem for the class of meshes is NP-complete even if the alphabet size is three.

4.3 Concluding remarks

We have introduced a replacement method of graphs called the linear chain decomposition and investigated the computational complexity of the walk realizability problem for various graphs G which are classified by the size of the linear chain decomposition of G and the underlying structure of G. We have shown that the problem for any $O(1)$-decomposable class is in NLOG, although the NLOG-completeness has not been proved yet. We have also devised a polynomial-time algorithm to solve the problem for any $O(\log m)$-decomposable class of trees, where m is the number of edges. We have also discussed the walk realizability problem for $O(m)$-decomposable classes of trees. We have shown that the problem for trees is, in general, NP-complete.

We can modify the walk realizability problem as a maximization problem. Let C be a class of graphs.

MAX WRP(C) (MAX Walk Realizability Problem)

`Instance:` A string x over a finite alphabet Σ and a graph G.

`Problem:` Find a walk w for x on G such that the number of edges traversed by w is maximized.

It is clear that for a class C of graphs, if the walk realizability problem for C is NP-complete then the decision version of MAX WRP(C), i.e., the problem of, given a string x, a graph G and a positive integer K, deciding if there is a walk w for x on G such that the number of edges traversed by w is at least K, is also NP-complete. It is not hard to show that for any class C of graphs, MAX WRP(C) has a polynomial-time approximation algorithm. But, the approximation ratio of the algorithm depends on the input size, that is, the ratio is not bounded by a constant. Search for polynomial-time approximation algorithm with better ratio is one of our future works.

Chapter 5

Conclusions

In this thesis we have discussed the graph inference from a walk, that was first studied by Aslam and Rivest [AR90]. They showed that the problem for graphs of bounded degree two is solvable in polynomial time. Raghavan [Rag94] improved the time-complexity of the algorithm by Aslam and Rivest. Raghavan also showed that a variant of the graph inference from a walk for graphs of bounded degree k is NP-complete for any $k \geq 3$.

Our results in this thesis on the graph inference from a walk have established a deeper understanding of the problem. We have constructed a linear-time algorithm for the tree inference from a walk by devising a tree rewriting system which satisfies the Church-Rosser property. However, we have proved that the problem with the constraint on degree, namely, the graph inference from a walk for trees of bounded degree k, turns to be NP-complete for any $k \geq 3$. By these two results, we can say that bounding the maximum degree is a stronger factor in solving the graph inference from a walk than the underlying structure of trees.

Giving consideration to the above observation on trees, we have defined (1,1)-caterpillars, which are still trees of bounded degree three but have much simpler structure because the form of a (1,1)-caterpillar is similar to that of a linear chain. Although the linear chain inference from a walk is known to be solvable in polynomial

time [AR90, Rag94], surprisingly, the graph inference from a walk for (1,1)-caterpillars has been shown to be NP-complete. Furthermore, we have also seen that it is still hard to approximately solve the graph inference problem even for such simple trees. Thus, we conclude the graph inference from a walk by stating that the constraint of bounding degree by a constant is strong enough to make the graph inference from a walk intractable.

In Chapter 3, we have discussed the tree inference from a finite number of partial walks instead of a single walk. We have seen that the tree inference problem turns to be NP-complete in contrast with the case of a single walk. In addition, we can say that the constraint of bounding degree by a constant is not a strong factor in computing the problem because we have proved that the problem is NP-complete regardless of existing such a constraint on degree. We have also discussed the linear chain inference from partial walks. In the same way as the case of trees, we have shown that the problem from partial walks for linear chains turns to be NP-complete, while the linear chain inference from a walk is known to be solvable in polynomial-time [AR90, Rag94]. By comparing the NP-completeness results on trees and linear chains with the tractability of the graph inference from a single walk for trees and linear chains, we can conclude that allowing any finite number of partial walks makes the graph inference problem computationally intractable.

In Chapter 4, we have investigated the walk realizability problem in order to see the difficulty of realizing a walk for a string x on a given graph G. For this problem, we have shown that the complexity of various cases classified in terms of the linear chain decomposition we have introduced. We recall that a node v is said to be proper if the colors of the edges incident to v are mutually distinct. Obviously, if all nodes of a graph G are proper, the walk realizability problem for such graphs is solvable in

polynomial time. However, we have seen that, when one node of a graph is allowed to be not proper, the problem turns to be intractable. Hence, we conclude the walk realizability problem by stating as follows: Allowing just one node being not proper makes the problem intractable.

Bibliography

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[Ang78] D. Angluin. On the complexity of minimum inference of regular sets. *Inform. Control*, 39:337–350, 1978.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75:87–106, 1987.

[AR90] J. A. Aslam and R. L. Rivest. Inferring graphs from walks. In *Proceedings of the 3rd Workshop on Computational Learning Theory*, pages 359–370, 1990.

[AS94] C. Armen and C. Stein. A $2\frac{3}{4}$-approximation algorithm for the shortest superstring problem. Unpublished manuscript, 1994.

[BJL+94] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *J. Assoc. Comput. Mach.*, 41:630–647, 1994.

[BP89] M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inform. Process. Lett.*, 32:171–176, 1989.

[BRS95] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18:231–254, 1995.

[BS94] M. A. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 75–85, 1994.

[CGPR94] A. Czumaj, L. Gasieniec, M. Poitrow, and W. Rytter. Parallel and sequential approximation of shortest superstrings. In *Proceedings of the 4th Scandinavian Workshop on Algorithm Theory*, volume 824 of *Lecture Notes in Computer Science*, pages 95–106. Springer-Verlag, 1994.

[CM90] M. P. Chytil and B. Monien. Caterpillars and context-free languages. In *Proc. 7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 70–81. Springer-Verlag, 1990.

[CR89] J. C. Culberson and P. Rudnicki. A fast algorithm for constructing trees from distance matrices. *Inform. Process. Lett.*, 30:215–220, 1989.

[Day87] W. H. E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bull. Math. Bio.*, 49:461–467, 1987.

[DP90] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 355–361, 1990.

[Fel82] J. Felsenstein. Numerical methods for inferring evolutionary trees. *Quart. Rev. Bio.*, 57:379–404, 1982.

[FKR+93] Y. Freund, M. Kearns, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 315–324, 1993.

[FKW95] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13:155–179, 1995.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[GMM94] G. Galbiati, F. Maffioli, and A. Morzenti. A short note on the approximability of the maximum leaves spanning tree problem. *Inform. Process. Lett.*, 52:45–49, 1994.

[GMS80] J. Gallant, D. Maier, and J. A. Storer. On finding minimal length superstrings. *J. Comput. System Sci.*, 20:50–58, 1980.

[Gol78] E. M. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978.

[Gus94] D. Gusfield. Faster implementation of a shortest superstring approximation. *Inform. Process. Lett.*, 51:271–274, 1994.

[GVY93] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming*, volume 700 of *Lecture Notes in Computer Science*, pages 64–75. Springer-Verlag, 1993.

[GVY94] N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming*, volume 820 of *Lecture Notes in Computer Science*, pages 487–498. Springer-Verlag, 1994.

[Hei89] J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bull. Math. Bio.*, 51:597–603, 1989.

[HMM91] J. Haralambides, F. Makedon, and B. Monien. Bandwidth minimization: An approximation algorithm for caterpillars. *Math. Systems Theory*, 24:169–177, 1991.

[Hue80] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *J. Assoc. Comput. Mach.*, 27:797–821, 1980.

[Ihl91] E. Ihler. The complexity of approximating the class Steiner tree problem. In *Proceedings of the 17th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 570 of *Lecture Notes in Computer Science*, pages 85–96. Springer-Verlag, 1991.

[JL94] T. Jiang and M. Li. Approximating shortest superstrings with constraints. *Theoret. Comput. Sci.*, 134:473–491, 1994.

[JLD92] T. Jiang, M. Li, and D. Du. A note on shortest superstrings with flipping. *Inform. Process. Lett.*, 44:195–199, 1992.

[JT95] T. Jiang and V. G. Timkovsky. Shortest consistent superstrings computable in polynomial time. *Theoret. Comput. Sci.*, 143:113–122, 1995.

[JWZ95] T. Jiang, L. Wang, and K. Zhang. Alignment of trees-an alternative to tree edit. *Theoret. Comput. Sci.*, 143:137–148, 1995.

[Kan91] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Inform. Process. Lett.*, 37:27–35, 1991.

[Kan92] V. Kann. On the approximability of the maximum common subgraph problem. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 377–388. Springer-Verlag, 1992.

[Kan94] V. Kann. Maximum bounded H-matching is MAX SNP-complete. *Inform. Process. Lett.*, 49:309–318, 1994.

[KPS94] S. R. Kosaraju, J. K. Park, and C. Stein. Long tours and short superstrings. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 166–177, 1994.

[KW95] S. K. Kannan and T. J. Warnow. Tree reconstruction from partial orders. *SIAM J. Comput.*, 24:511–519, 1995.

[Li90] M. Li. Towards a DNA sequencing theory. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 125–134, 1990.

[Mid94] M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theoret. Comput. Sci.*, 125:205–228, 1994.

[MM92] O. Maruyama and S. Miyano. Inferring a tree from walks. In *Proceedings of the 17th Symposium on Mathematical Foundations of Computer Science*, volume 629 of *Lecture Notes in Computer Science*, pages 383–391. Springer-Verlag, 1992. To appear in *Theoret. Comput. Sci.*, 162, 1996.

[MM95a] O. Maruyama and S. Miyano. Graph inference from a walk for trees of bounded degree 3 is NP-complete. In *Proceedings of the 20th Symposium*

on Mathematical Foundations of Computer Science, volume 969 of *Lecture Notes in Computer Science*, pages 257–266. Springer-Verlag, 1995.

[MM95b] O. Maruyama and S. Miyano. Taking a walk on a graph. Technical Report RIFIS-TR-CS-116, Research Institute of Fundamental Information Science, Kyushu University, July 1995.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[PSW93] D. Peleg, G. Schechtman, and A. Wool. Approximating bounded 0-1 integer linear programs. In *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems*, pages 69–77. IEEE, 1993.

[PW93] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *J. Assoc. Comput. Mach.*, 40:95–142, 1993.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43:425–440, 1991.

[PY93] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18:1–11, 1993.

[Rag94] V. Raghavan. Bounded degree graph inference from walks. *J. Comput. System Sci.*, 49:108–132, 1994.

[RS93] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Inform. Comput.*, 103:299–347, 1993.

[RS94] R. L. Rivest and R. E. Schapire. Diversity-based inference of finite automata. *J. Assoc. Comput. Mach.*, 41:555–589, 1994.

[Rud85] S. Rudich. Inferring the structure of a Markov chain from its output. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 321–326, 1985.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.*, 4:177–192, 1970.

[TU88] J. Tarhio and E. Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theoret. Comput. Sci.*, 57:131–145, 1988.

[Tur89] J. S. Turner. Approximation algorithms for the shortest common superstring problem. *Inform. Comput.*, 83:1–20, 1989.

[TY93] S.-H. Teng and F. Yao. Approximating shortest superstrings. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 158–165, 1993.

[Ukk90] E. Ukkonen. A linear-time algorithm for finding approximate shortest common superstrings. *Algorithmica*, 5:313–323, 1990.

[Win88] P. Winkler. The complexity of metric realization. *SIAM J. Disc. Math.*, 1:552–559, 1988.

[WSSB77] M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *J. Theor. Biol.*, 64:199–213, 1977.

[YM93] A. Yamaguchi and S. Miyano, 1993. Personal communication.

[Zha95] L. Zhang. On the approximation of longest common nonsupersequences and shortest common nonsubsequences. *Theoret. Comput. Sci.*, 143:353–362, 1995.

[ZJ94] K. Zhang and T. Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Inform. Process. Lett.*, 49:249–254, 1994.