Graph Inference from Walks

丸山, 修
Interdisciplinary Graduate School of Engineering Sciences, Kyushu University

https://doi.org/10.11501/3084068

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:

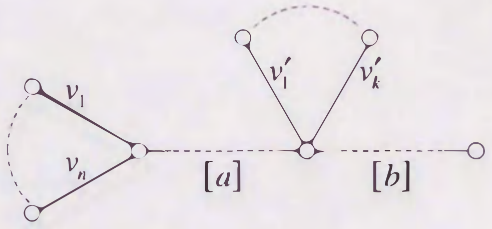
$$\begin{aligned} [a] &= a_{\lceil n/2 \rceil} \cdots a_1 a_0 a_1 \cdots a_{\lceil n/2 \rceil} \\ [b] &= b_1 \cdots b_{n+1} . \end{aligned}$$

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

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

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 $|C| \le K$. 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 $|C| \le K$.

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.

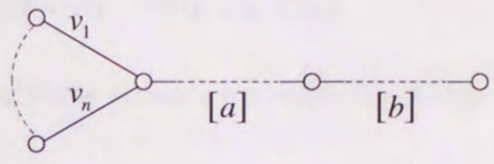
(2) $\widehat{F}(T)$ is not larger than T.

(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:

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

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.

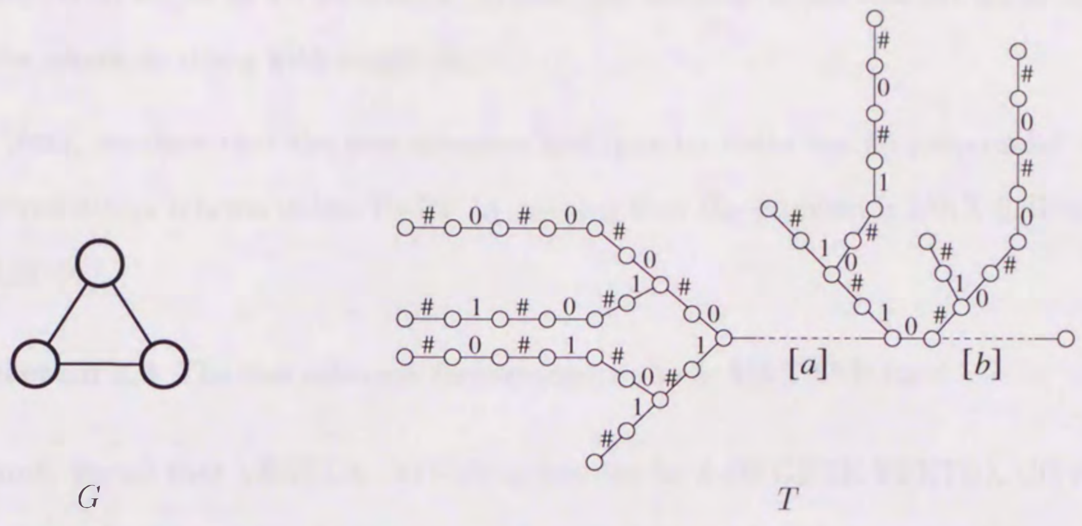


Figure 3.3: The tree T realizes all partial walks for the strings produced by the second reduction from the graph G.

□

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.*

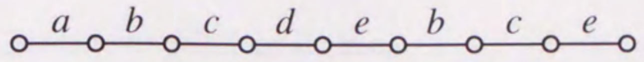


Figure 3.4:

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. $\square$

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 $\hat{F}(l_x)$, 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 $\{\hat{F}(l_x) \mid x \in S\}$ by $\hat{F}(S)$. Given a finite set S of strings, if we could find the smallest supertree for $\hat{F}(S)$, 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 $\hat{F}(S)$ 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."*

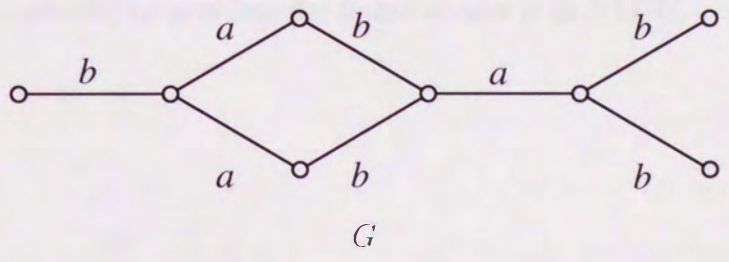


Figure 4.1:

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 $\phi : V \times \{1, 2, \ldots, |x|\} \times V \times \{0,1\}^{m'} \rightarrow \{0,1\}$ 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.

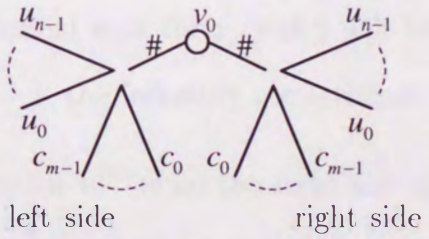


Figure 4.2: The node v_0 is a unique non-proper node.

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:

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

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

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

The string x is defined by

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

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 $f : U \to \{\mathbf{true}, \mathbf{false}\}$. 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 $f(u_i) = \mathbf{true}$. 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

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

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. □

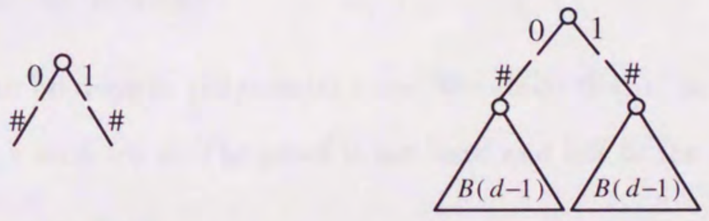
When the number of non-proper nodes in a tree T is restricted to zero, the walk realizability problem is solvable in polynomial time by applying the linear-time algorithm for the tree inference from a walk, provided in Section 2.3, since the tree T

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 $\tilde{B}(d)$ 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 $\tilde{B}(d)$ 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, $\tilde{B}(6)$ 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 $\mathrm{code}(u_i)$ and $\mathrm{code}(c_k)$ 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:

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

The string x is defined as follows:

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

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. □

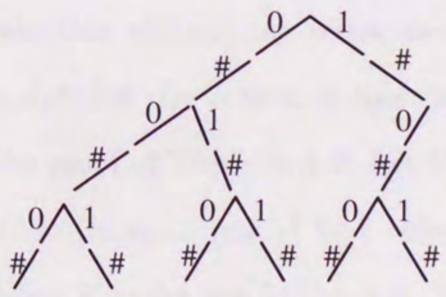


Figure 4.4: $\tilde{B}(6)$.

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

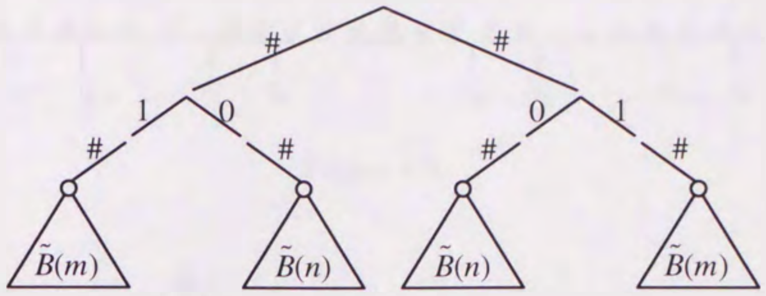


Figure 4.5: T.

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:

$$\begin{aligned} u_i^{(p)} &= \#(aa)^{i+1}u_iu_i(aa)^{(n-i-1)+m} \prod_{c_k \in C_p(u_i)} (c_kc_k(aa)^m)(aa)^n\# \\ u_i^{(n)} &= \#(aa)^{i+1}u_iu_i(aa)^{(n-i-1)+m} \prod_{c_k \in C_n(u_i)} (c_kc_k(aa)^m)(aa)^n\# \end{aligned}$$

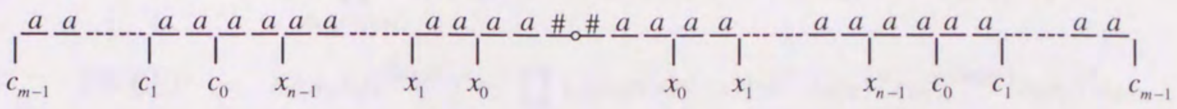

a a ...... a a a a a a ...... a a a a # # a a a a ...... a a a a a a ...... a a
c_{m-1}
c_1
c_0
x_{n-1}
x_1
x_0
x_0
x_1
x_{n-1}
c_0
c_1
c_{m-1}

Figure 4.6:

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

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

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

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:

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

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

$$\begin{aligned} u_i^{(p)} \;=\;& s_4(s_2s_20^{\lceil \log n \rceil})^i s_2s_2\mathrm{code}(u_i)(s_2s_20^{\lceil \log n \rceil})^{n-i-1} \\ & s_3 \prod_{c_k \in C_p(u_i)}((s_2s_20^{\tilde{m}})^m \mathrm{code}(c_k))(s_2s_20^{\tilde{m}})^m s_3(0^{\lceil \log n \rceil}s_2s_2)^n s_4. \end{aligned}$$

$$u_i^{(n)} \;=\; s_4(s_2s_20^{\lceil \log n \rceil})^i s_2s_2\mathrm{code}(u_i)(s_2s_20^{\lceil \log n \rceil})^{n-i-1}$$

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

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

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

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'.

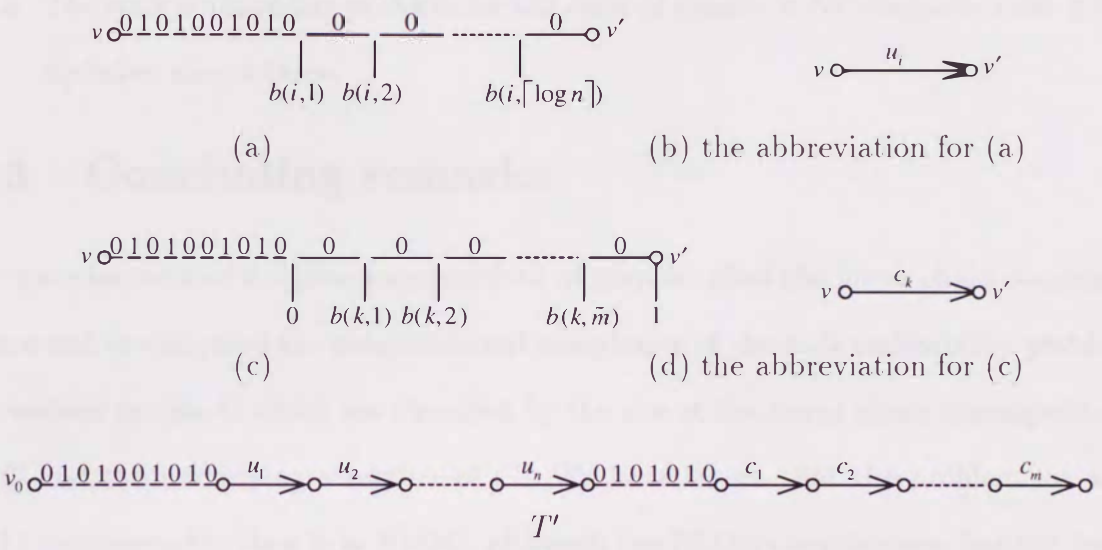


Figure 4.7:

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.

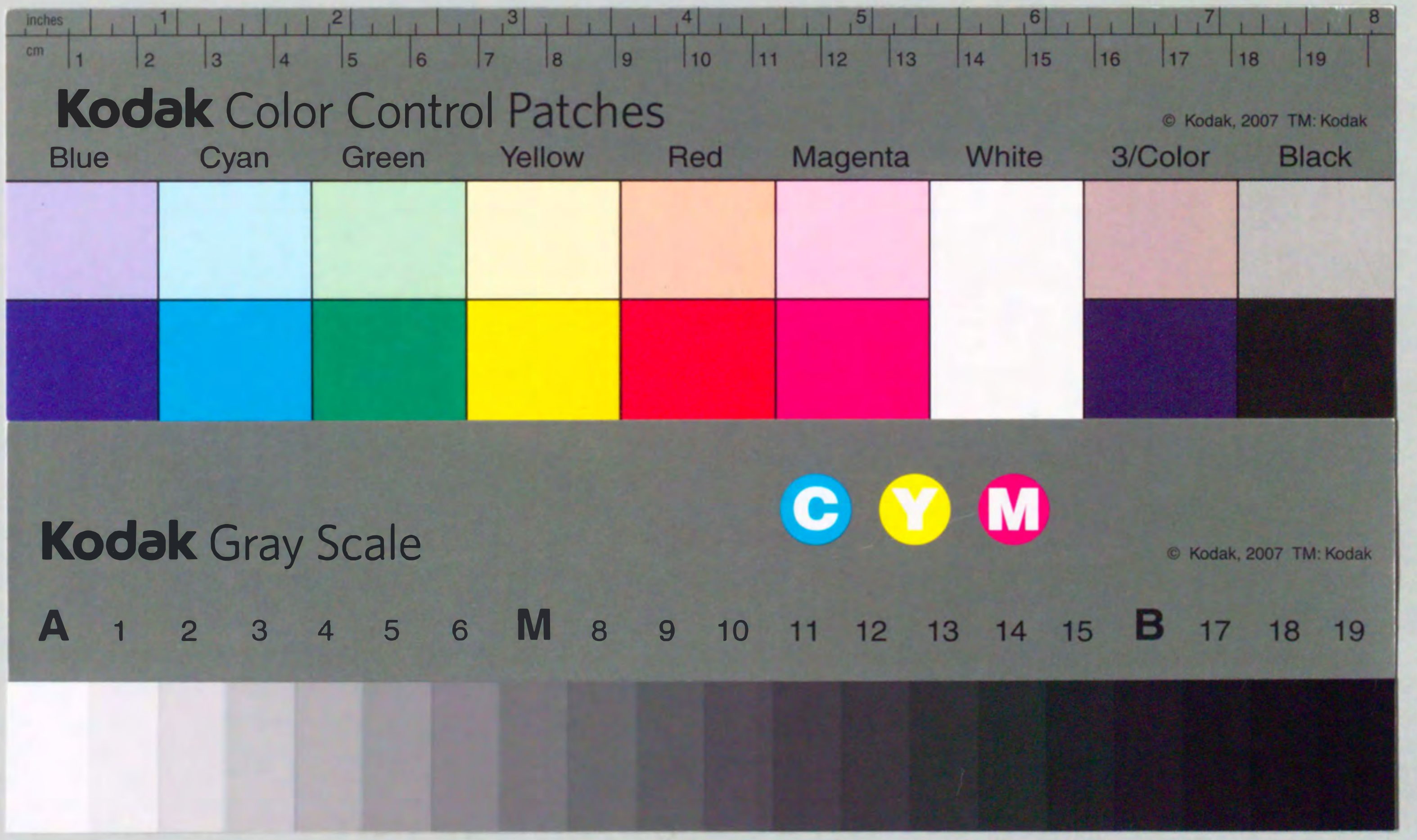
[Win88] P. Winkler. The complexity of metric realization. *SIAM J. Disc. Math.*, 1:552–559, 1988.

[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.


inches 1 2 3 4 5 6 7 8
cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue Cyan Green Yellow Red Magenta White 3/Color Black
C Y M
Kodak Gray Scale
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19