Doctoral Dissertation

Computational Complexity of Graph Embedding Problems Inspired by Protein Folding and Design

Feng Tianfeng

Supervisor: Ryuhei Uehara

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Information Science

March, 2022

Abstract

Protein folding is a central problem in bioinformatics. The protein folding problem asks how a protein's amino acid sequence dictates its three-dimensional atomic structure. This problem has wide applications and a long history dating back to the 1960s. From the viewpoint of theoretical computer science, there is ongoing research aiming at revealing insights into reality by working on simplified abstract models. A protein is modeled by an abstract linkage in a lattice space, which has been proven to be an extremely useful tool for reasoning about the complexity of protein structure prediction and design. Since protein functionality is controlled by the native state structure, these models provide the ideas of theoretical investigations for future use in studies on real proteins.

In this thesis, we aim to explore the computational tractability or intractability of protein structure prediction and design using techniques from computational complexity theory and algorithm design and analysis. Inspired by the popular hydrophobic-polar (HP) model, we have developed some new graph-theoretic models and problems that may be seen as variations or extensions of the standard HP one. We then studied the tractability of these problems, by finding polynomial-time algorithms or by proving that they are NP-hard. In our model, we combine the basic ideas of protein folding with the complementary problem of *protein design*, where the goal is to synthesize a protein of a given shape (and function) from an amino acid sequence. We modeled proteins and amino acid chains in two different settings. In each setting, we studied structure prediction and design of proteins by folding amino acid chains.

The first setting is protein folding prediction and design in grid graphs. It is inspired by the famous hydrophobic-polar (HP) model for protein folding. A protein in the HP model is represented as an abstract open chain, where each link has a unit length and each joint is marked either hydrophobic or polar. Grid graphs are graphs that form a regular tiling of the 2D plane or 3D space; these graphs are the standard setting for the traditional HP model. Here a protein is a (connected) subgraph G of the grid graph, possibly with colors assigned to nodes. An amino acid chain is a path graph P with colored nodes. We thus propose the bicolored path embedding problem. A graph is said *bicolored* if each vertex is assigned a label in the set {red, blue}. For a given bicolored path P and a given bicolored graph G, our problem asks whether we can embed P into G in such a way as to match the colors of

the vertices. In our model, G represents a protein's "blueprint," and P is an amino acid sequence that has to be folded to form (part of) G.

In this setting, we first prove that the bicolored path embedding problem is NP-complete even if P is monochromatic (e.g., all its vertices have the same color). Then we prove that the problem is NP-complete even if G and P are bicolored and have the same number of vertices. The latter result is a fairly technical reduction from the Hamiltonian path problem. The importance of this result lies in the fact that, when G has the same number of vertices as P, it represents an exact "blueprint" of the protein we want to obtain, as opposed to just an "ambient space" for it.

Next, we contrast these hardness results with a polynomial-time algorithm for the case where G is a grid of fixed height: thus, the bicolored path embedding problem, parameterized according to the height of G, is in XP. The technique we used is dynamic programming, and the significance of this result is that it offers an efficient algorithm for a generic grid G, although the algorithm's performance deteriorates with the height of G (but not as much with its width).

We further showed that the classical problem of maximizing H-H contacts is also NP-hard in the context of the bicolored path embedding problem. Note that, in previous work, it has been established that the problem of maximizing H-H contacts is NP-hard when G is not given, and P can be embedded in any way on a grid.

In the second setting, we studied the protein folding prediction and design in general graphs. These are graphs with either colored vertices or edges of a given length. Here a protein is a graph: the idea is that a protein has a "high-level" shape that can be represented by some graph G, even if at "low level" the protein is just a chain. An amino acid chain is a path P, possibly with colored nodes or fixed edge lengths. The prediction problem asks, for a given amino acid chain, what graphs it can fold into (i.e., what graphs it can be mapped onto), satisfying some local constraints. The design problem asks, for a given graph (i.e., a protein), whether we can design an amino acid chain that can be mapped onto it satisfying some local constraints.

In this setting, we first study graphs with colored vertices. We prove that the bicolored path embedding problem is NP-complete even if G is a dense graph with the same number of vertices as P. Here, the previous remark about G being an exact "blueprint" of a protein holds, since it has the same number of vertices as P. Additionally, the fact that G is dense (i.e., it has a quadratic number of edges) makes this result surprising for another reason: intuitively, a blueprint with many edges should allow greater leeway in the construction of an embedding of P. As it turns out, a greater amount of freedom does not necessarily translate into our ability to easily

find embeddings. We also prove a complementary result: the problem of constructing a path P that embeds in a given bicolored graph G maximizing H-H contacts is Poly-APX-hard. In particular, it has no polynomial-time approximation algorithm with a sub-polynomial approximation ratio unless P = NP.

We also considered a different model, where G is a (non-colored) graph with given edge lengths, and P is a linkage (a path with edge lengths). Our goal is to find an embedding of P in G that matches the lengths of edges. The problem asks if there is an edge-weighted Eulerian path of target graph G spanned by the linkage P. We showed that the problem is strongly NP-hard even if edges have only two possible lengths. Together with the fact that the problem is solvable in linear time if edge lengths are all the same, this result gives a precise characterization of the problem's tractability.

We tackled this intractability result by considering two different variants of the problem. In the first variant, we allow the edges in P to be elastic, that is, we can stretch or shrink the edges in the elastic linkage P. The goal is to minimize the elastic ratio of the embedding. Remarkably, we found that when G is a path, there is a polynomial-time algorithm based on dynamic programming. In the second variant, we allow P to cover an edge of G twice or more. We showed that the problem is NP-hard even if G consists of a single edge. Furthermore, with the requirement that each edge of G is covered by P exactly twice, we obtained three hardness results and one polynomial-time algorithm when G and edge lengths are restricted.

This research has applications that go beyond protein folding and design, and has laid the foundations for interesting future developments, as well.

Keywords— protein folding problem, HP (hydrophobic-polar) model, embedding problem, Hamiltonian path problem, edge-weighted Eulerian path problem

Acknowledgement

Prof. Ryuhei Uehara, my supervisor, has been quite helpful. I am very grateful to him for his direction and support throughout my studies, as well as for motivating me to strive for greater success in the future.

I'd also like to express my gratitude to my family and friends for their patience and company.

My best friend Xu Rui, who was one of the most essential persons in my life, is the last person I'd like to thank. In heaven, I wish you peace and happiness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

In this chapter, we will introduce the background of the protein folding problem and the motivation for this research topic.

1.1 The HP-Model

The protein folding problem asks how a protein's amino acid sequence dictates its three-dimensional atomic structure. This problem has wide applications and a long history dating back to the 1960s [12]. From the viewpoint of theoretical computer science, there is ongoing research aiming at revealing insights into reality by working on simplified abstract models.

One of the most popular such models is the *hydrophobic-polar* (HP) model [10, 11, 14, 20, 23, 26, 9, 17]. A protein in the HP model is represented as an abstract open chain, where each link has unit length and each joint is marked either H (hydrophobic, i.e., non-polar) or P (hydrophilic, i.e., polar). A protein is usually envisioned as a path embedded in a grid within the 2D or 3D lattice, where each joint in the chain maps to a point on the lattice, and each link maps to a single edge. The HP model of energy specifies that a chain desires to maximize the number of *H-H contacts*, which are pairs of H nodes that are adjacent on the lattice but not adjacent along the chain. The Figure 1.1 shows an example of counting the number of *H-H contacts*, and the Figure 1.2 shows the structures of a given sequence in the HP-model. The *optimal folding problem* in the HP model asks to find an embedding of a sequence of Hs and Ps on the 2D square lattice that maximizes the number of H-H contacts.

Previous results on the HP model mostly concern the 2D square lattice, where commonly used techniques can be criticized for relying on the properties of parity in the lattice. For example, [19, 24, 27] provide several

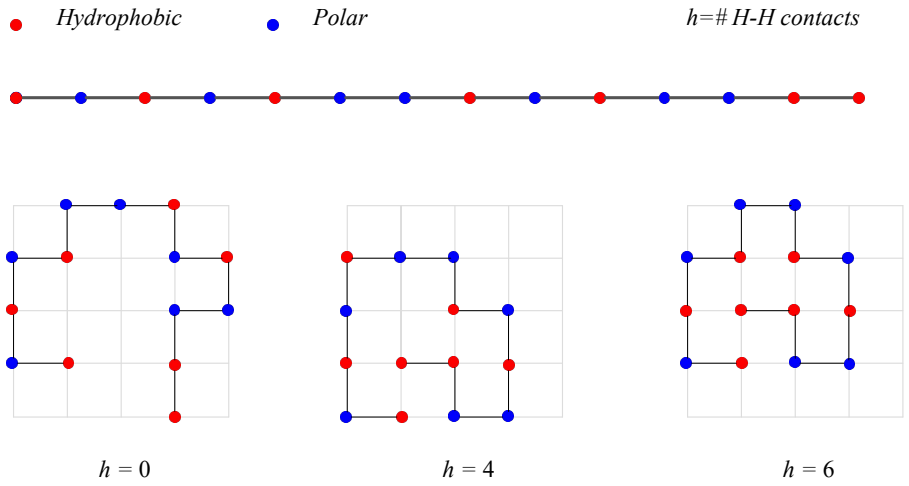

Figure 1.1: An example of counting the number of *H-H contacts*. Three possible configurations of embedding an amino acid chain in 2D lattice are given, the number of *H-H contacts* are 0, 4, and 6.

approximation algorithms, all of which bound the maximum number of H-H contacts in terms of the number of odd-parity H nodes and the number of even-parity H nodes in the chain. This is because two H nodes can be embedded in adjacent nodes on the square lattice only if their distance along the chain is odd, i.e., if they have opposite parity. Such observations make sense in the discrete setting, but have no obvious meaning in the real protein folding problem that the theory aims to model. Thus, parity-related arguments should not be taken as the *only* reason as to why an amino acid chain can or cannot fold in a certain way.

Solving the optimal folding problem in a square lattice is shown to be NP-hard in [7]. Although the proof does not suffer from the aforementioned parity-related issues, it constructs a hard-to-fold chain whose properties heavily rely on Hadamard codes and Hamming distances. Again, appealing to these inherently discrete concepts is a departure from the continuous nature of real protein folding.

A final point of criticism to the traditional HP model is that the number of H-H contacts is not the only possible measure that may be used to capture the intricate physical and chemical laws that describe how a real protein folds.

In mathematics and theoretical computer science, there is a vast literature on embeddings of paths and graphs. The general problem of embedding an unlabeled path into a given graph is well-known to be NP-hard [7]. However, the graph embedding problem is efficiently solvable in some special cases.

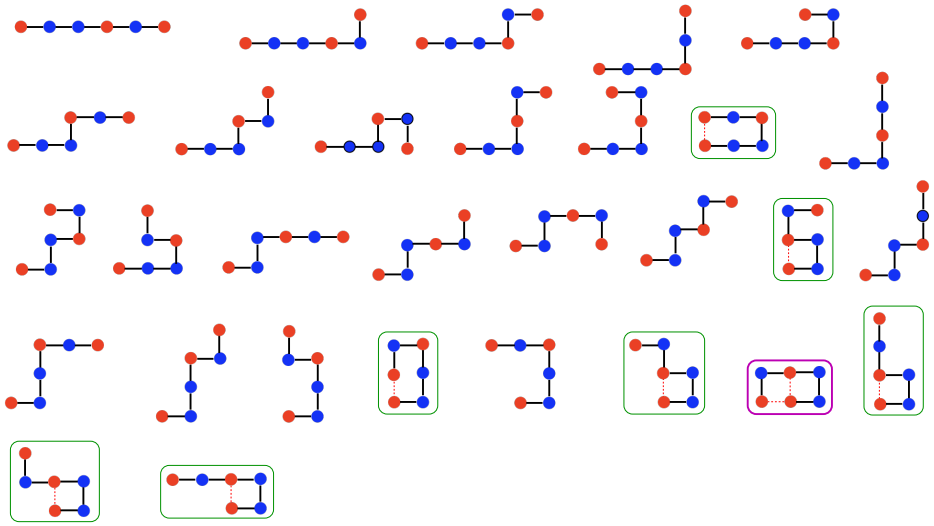

Figure 1.2: Structures in the HP-model of the sequence "HPPHPH". Structures with H-H contacts are circled. The structure with the largest number of H-H contacts is circled in purple.

Notably, there is a linear-time algorithm for embedding graphs of constant size into planar graphs [15]. Unfortunately, these results say nothing about *labeled* graphs, which are the focus of the research on protein folding.

Restricting our attention to labeled graphs, there are substantially fewer works. To the best of our knowledge, the literature in this field is limited to exponential-time algorithms for the general embedding problem, e.g., [6, 22].

1.2 Our Approach and Motivation

Our critique of the standard HP model has inspired us to formulate the *bicolored path embedding problem*, which, apart from being an interesting graph embedding problem in its own right, may be seen as a new variant of the protein folding problem within the HP model.

In our model, we combine the basic ideas of protein folding with the complementary problem of *protein design*, where the goal is to synthesize a protein of a given shape (and function) from an amino acid sequence. Thus, we provide the "blueprint" of the folded shape of a protein, in the form of an input (grid) graph G with colors assigned to its vertices, and we ask if a given colored path P can be (injectively) embedded in G in such a way that vertex colors match. In other terms, we are effectively asking whether a given amino acid sequence can fold into (part of) a protein with a prescribed

structure. Since the HP model has nodes of only two types, we assume both G and P to be *bicolored*, say, with colors "red" and "blue."

Furthermore, we consider a different model where the amino acid chain is modeled as a simple linkage (i.e., a path with assigned edge lengths). Here a protein is a graph with fixed edge lengths. For a given (non-colored) path P and a (non-colored) graph G with edge lengths, our goal is to find an embedding of P in G that matches the lengths of edges.

The significance of our model is that it more accurately captures some of the crucial and practical problems of protein folding. These problems do not only concern the way a given amino acid chain folds spontaneously, but also involve the design of proteins with desired attributes and shapes. Additionally, in our analysis we strive to avoid any argument that seems too closely related to the arbitrary choices we made when designing our model (for example, none of our proofs relies on the fact that a grid is 2-vertex-colorable, unlike some previous works [19, 24, 27]).

The underlying idea of our research is that, in biological processes, nature "solves" some computational problems related to protein folding in a seemingly efficient way. In order to explain these phenomena, the approach of theoretical computer science is to formulate abstract models of proteins and amino acid chains and study the computational complexity of protein folding problems under these models.

1.3 Contents of Thesis

The thesis is organized as follows. In Chapter 2, we give a formal definition of the bicolored path embedding problem and linkage simulation problem inspired by the protein folding model.

In Chapter 3, we study the protein folding prediction and design in grid graphs. In this setting, a protein is a (connected) subgraph G of the grid graph, possibly with colors assigned to nodes. An amino acid chain is a path graph P with colored nodes.

In Section 3.1, we prove that the path embedding problem is NP-complete even if P is monochromatic (e.g., all its vertices are blue), and then in Section 3.2 we consider the case where G is a rectangular grid, which is the standard assumption in the HP model. We prove that the bicolored path embedding problem is NP-complete even if G and P have the same order (i.e., number of vertices). Next, we contrast this hardness result with a polynomial-time algorithm for the case where G is a grid of fixed height; thus, our embedding problem, parameterized according to the height of G, is in XP. In Section 3.3, we show that maximizing red-red contacts in the

bicolored path embedding problem (defined in the same way as H-H contacts in the HP model) is also NP-hard, even if G is a rectangular grid. We remark that, in previous work, it has been established that the problem of maximizing H-H contacts is NP-hard when G is not given, and P can be embedded in any way on a grid [7]. We also propose the bicolored synthesis problem. It is a complementary problem of constructing a path P that embeds in a given bicolored graph G maximizing red-red contacts, which is related to the protein design.

In Chapter 4, we study the protein folding prediction and design in general graphs. These are graphs with either colored vertices or edges of a given length. In this setting, a protein is a graph, the idea is that a protein has a "high-level" shape that can be represented by some graph G, even if at "low level" the protein is just a chain. An amino acid chain is a path P, possibly with colored nodes or fixed edge lengths.

In Section 4.1, we first study graphs with colored vertices. We prove that the bicolored path embedding problem is NP-complete even if G is a dense graph with the same number of vertices as P. In Section 4.2, we prove that the bicolored synthesis problem is Poly-APX-hard. In particular, it has no polynomial-time approximation algorithm with a sub-polynomial approximation ratio, unless P = NP.

In Section 4.3, we also consider a different model, where P and G are (non-colored) graphs with given edge lengths, and our goal is to find an embedding of P in G that matches the lengths of edges. The first interesting result is that this problem is strongly NP-hard even if edge lengths are quite restricted (Theorem 7). We remind that if they consist of unit length edges, the problem is linear-time solvable. We thus tackle this problem in two different ways in Section 4.4 and 4.5.

In Section 4.4, we allow the edges in P to be elastic. We consider a linkage is *elastic*, that is, the length of one line segment is not fixed and can be changed a little bit. This situation is natural not only in the context of the linkage simulation but also in the approximation algorithm. Formally, we allow the edges in P to be elastic to fit the vertices of P to ones of G. Our goal is to minimize the stretch/shrink ratio of each edge of P. We show that when G is a path, this can be solved in polynomial time by dynamic programming.

In Section 4.5, we allow P to cover an edge of G twice or more. In this situation, we can simulate G by P even if G does not have an Eulerian path. We first show that the problem is weakly NP-hard even if G is an edge (Theorem 9). In fact, this problem is similar to the ruler folding problem (see, e.g., [2, 13]). Furthermore, in Section 4.5.2, with the requirement that each edge of G is covered by P exactly twice, we obtained three hardness results

and one polynomial-time algorithm when G and edge lengths are restricted.

Finally, in Chapter 5, we discuss the contribution and possible applications of this research. In Section 5.2, we conclude this thesis with a list of open problems and some directions for future research.

The main results are shown in (Table 1.1).

Protein folding prediction and design in grid graphs	
Bicolored Path Embedding Problems:	
General Grid Graphs	NP-hard
Rectangular Grid Graphs	NP-hard
Fixed-Height Rectangular Grid Graphs	Polynomial-time solvable
Maximizing Red-Red Contacts	NP-hard
Protein Folding Prediction and Design in General Graphs	
Bicolored Path Embedding Problem in General Graphs	NP-hard
Bicolored Synthesis Problem	Poly-APX-hard
Weighted Eulerian Path Problem	NP-hard
Elastic Linkage Problem	Polynomial-time solvable
Covering Problem of a Tree by a Path:	
General Cover Problem	NP-hard
Tree Traversal Problem (Case 1)	NP-hard
Tree Traversal Problem (Case 2)	Polynomial-time solvable

Table 1.1: Main Results

Chapter 2

Definitions and Preliminaries

We modeled proteins and amino acid chains in two different settings: In each setting, we studied (a) structure prediction and (b) design of proteins by folding amino acid chains.

(1) Protein folding prediction and design in grid graphs: This model is inspired by the famous hydrophobic-polar (HP) model for protein folding. A protein in the HP model is represented as an abstract open chain, where each link has a unit length and each joint is marked either hydrophobic or polar. Grid graphs are graphs that form a regular tiling of the 2D plane or 3D space; these graphs are the standard setting for the traditional HP model. Here a protein is a (connected) subgraph G of the grid graph, possibly with colors assigned to nodes. An amino acid chain is a path graph P with colored nodes.

(a) Prediction: Given an amino acid chain, can we embed it into the grid in such a way that some local constraints are satisfied? (E.g., the colors on the chain match the colors on the grid, or the H-H contacts are maximized, etc.)

(b) Design: Given a pattern on the grid (i.e., a protein), can we design an amino acid chain that can fold into it, satisfying some local constraints? (E.g., at every hydrophobic amino acid there must be a 90-degree turn in the folded state, the H-H contacts are maximized, etc.)

In this setting, we propose the ***bicolored path embedding problem*** asking whether a given bicolored path P can be embedded in a bicolored grid graph G with color matching.

(2) Protein folding prediction and design in general graphs: These are graphs with either colored vertices or edges of a given length. Here a protein is a graph: the idea is that a protein has a "high-level" shape that can be represented by some graph G, even if at "low level" the protein is just a chain. An amino acid chain is a path P, possibly with colored nodes or fixed edge lengths.

(a) Prediction: Given an amino acid chain, what graphs can it fold into (i.e., what graphs can it be mapped onto), satisfying some local constraints?

(b) Design: Given a graph (i.e., a protein), can we design an amino acid chain that can be mapped onto it satisfying some local constraints?

In this setting, we first study the ***bicolored path embedding problem*** in general graphs, and then we propose the ***linkage simulation problem*** asks whether a given linkage P can simulate a target edge-weighted graph G.

In the next two sections, we will give the definitions of these two problems and the definitions related to their derivative problems.

2.1 Bicolored Path Embedding Problem

In this thesis, a graph is said "bicolored" if each of its vertices is assigned one of two possible colors, e.g., red or blue:

Definition 1. *A* bicolored graph *is a labeled undirected graph* $G = (V, E, \omega)$*, where* $\omega \colon V \to \{red, blue\}$*.*

If the image of ω is {blue}, then G is *monochromatic*. A *bicolored path* is a bicolored graph with the topology of a path, i.e., such that $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{\{v_i, v_{i+1}\} \mid 1 \leq i < n\}$.

If $P = (V, E, \omega)$ is a bicolored path with $V = \{v_1, v_2, \ldots, v_n\}$ and $G = (V', E', \omega')$ is a bicolored graph, we say that a function $f \colon V \to V'$ is an *embedding* of P into G if:

- f is injective, i.e., $i \neq j \implies f(v_i) \neq f(v_j)$.
- f maps edges of P to edges of G, i.e., $\{f(v_i), f(v_{i+1})\} \in E'$ for all $1 \leq i < n$.
- f respects colors, i.e., $\omega(v) = \omega'(f(v))$ for all $v \in V$.

Definition 2. *The* bicolored path embedding problem *asks whether a given bicolored path P has an embedding into a given bicolored graph G.*

In the context of the bicolored path embedding problem, the graph G is called the *blueprint*.

Our first observation is that the NP-complete *Hamiltonian path* problem (i.e., given a graph, decide if there is a walk that visits every vertex exactly once [18]) is a special case of our bicolored path embedding problem. Namely, if both P and G are monochromatic and have the same order, then an embedding of P into G is precisely a Hamiltonian path in G.

It follows that the problem of finding a bijective embedding of a monochromatic path P is NP-complete. Furthermore, it remains NP-complete for all classes of blueprints G where the Hamiltonian path problem is NP-complete. These include *split graphs*, which are graphs whose vertices can be partitioned into a clique and an independent set. Finding a Hamiltonian path in a split graph is NP-complete even if the clique and the independent set have the same order, as shown in [25].

In particular, such graphs are *dense*, i.e., they have a number of edges that are quadratic in the number of vertices. The fact that our path embedding problem is NP-complete even for dense blueprints is somewhat surprising: intuitively, a blueprint G with many edges should allow greater leeway in the construction of an embedding of P. As it turns out, a greater amount of freedom does not necessarily translate into our ability to easily find embeddings.

The second class of interest is that of *grid graphs*, sometimes also called *lattice graphs*:

Definition 3. *A* grid graph *is an induced subgraph of a regular tiling of the plane.*

Note that there are only three possible regular tilings of the plane: the square lattice, the triangular lattice, and the hexagonal lattice.

Grid graphs are the typical setting of the standard HP model. It is known that the Hamiltonian path problem is NP-complete even if G is a grid graph in the square lattice, in the triangular lattice, or in the hexagonal lattice [3]. Thus, so is our monochromatic path embedding problem.

Furthermore, the *bicolored* path embedding problem is NP-complete when G is a *solid* grid graph. Formally, let us define $R(a, b)$ as the grid graph in the square lattice whose vertex set is $\{(i, j) \mid 1 \leq i \leq a \text{ and } 1 \leq j \leq b\}$.

Definition 4. *A* rectangular *grid graph is any subgraph of the square lattice that is isomorphic to $R(a, b)$ for some integers a and b.*

Rectangular grid graphs are also called *solid* grid graphs in the square lattice.

Given any grid graph G' on n vertices in the square lattice, we can find the smallest integers a and b such that the graph $G = R(a, b)$ contains an induced subgraph G'' isomorphic to G' (intuitively, G is the "bounding rectangle" of G''). We color in blue all vertices of G'', and we color in red all other vertices of G. This operation is called "completing" G' to a solid grid graph G. Now, we can embed a path P of n blue vertices into G if and only if G' has a Hamiltonian path: this proves that the problem is NP-complete.

We can easily generalize the previous observation to solid graphs in the triangular or hexagonal lattice, which are defined similarly, as equilateral triangles and equilateral hexagons, respectively. This NP-completeness result will be strengthened in the next chapter, where we consider *bijective* embeddings.

2.2 Linkage Simulation Problem

Linkages are mechanisms composed of stiff, rigid chains, which are jointed at freely rotating joints. The Figure 2.1 shows a desk-lamp linkage, and the linkage flexes at the circled joints. A linkage is a collection of fixed-length 1D segments joined at their endpoints forming a path (See [13] for further details). In this research, we only consider simple linkages:

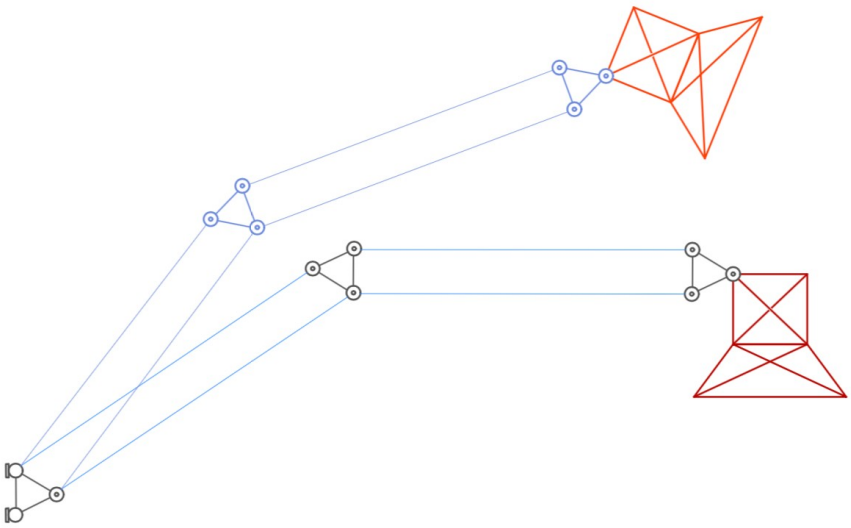

Figure 2.1: A linkage. The linkage flexes at the circled joints; the two left most joints are pinned to the plane. The shaded lamp structure is rigid.

Definition 5. *A* linkage *is a path* $P = (V, E, \ell)$*, where* $\ell : E \to \mathbb{R}$.

If $P = (V, E, \ell)$ is a linkage with $V = \{v_1, v_2, \ldots, v_n\}$, where v_i is an endpoint, then $e_i = \{v_i, v_{i+1}\}$ is an edge in $E = \{\{v_i, v_{i+1}\} \mid 1 \le i \le n-1\}$, and its length is given by $\ell(e_i)$.

Now we consider the following situation in Figure 2.2.

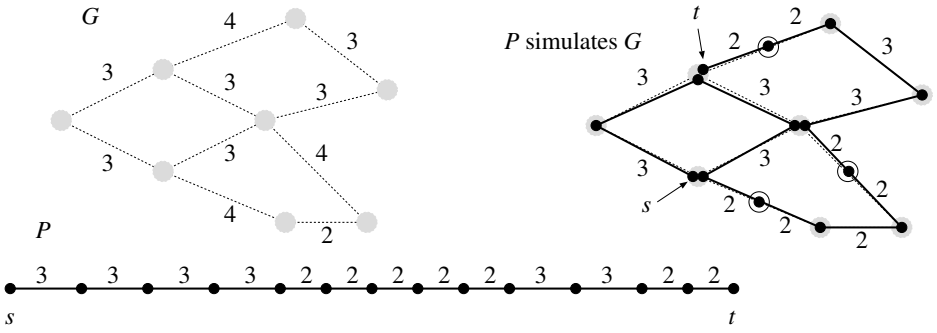

Figure 2.2: A simple example. A linkage P can simulate a given target graph G as shown in the figure. When P simulates G, each circled joint is fixed, and two joints of the linkage on the same vertex of G move with synchronization.

You are given a simple undirected target graph $G = (V', E')$, and a linkage $P = (V, E)$ as above with length function $\ell : E \cup E' \to \mathbb{R}$. Our mission is *to simulate* the target graph G by the given linkage P. The joints in P are programmable, and each joint (or vertex) of G should be simulated by a joint of P, however, we can also put the joints of P on some internal points of edges of G because they can be fixed. Therefore, our problem can be formalized as finding the following mapping ϕ from P to G:

- Each vertex of G should be mapped from some vertices of P;
- Each edge of G should be mapped from a subpath of P by ϕ;
- Each edge of P should be mapped to an edge of G, which may span from one internal point to another on the edge.

Definition 6. *The* linkage simulation problem *asks if there exists a mapping* ϕ *from a given linkage* P *to a given target graph* G.

The decision problem asks if there exists a mapping ϕ from P to G. That is, it asks if there is an Eulerian path of G spanned by P such that (1) when P visits a vertex in G, a vertex of P should be put on it, and (2) some vertices in P can be put on internal points of edges of G. When all edges

in P and G have the same length, it is easy to solve that in the linear time since the problem is the ordinary Eulerian path problem. In the context of formal languages, there are some variants of the Eulerian path problem with some constraints (see [4] for a comprehensive survey). However, as far as the authors know, the *linkage simulation problem*, our variant of the Eulerian path problem has not been investigated, while the situation is quite natural.

In this thesis, we only consider a simple undirected graph $G = (V, E)$. A *path* $P = (v_1, v_2, \ldots, v_n)$ consists of n vertices with $n-1$ edges e_i joining v_i and v_{i+1} for each $i = 1, \ldots, n-1$. The vertices v_1 and v_n of the path are called *endpoints*.

Let $G = (V', E')$ and $P = (V, E)$ be a graph and a path $(v_1, v_2, \ldots, v_n)$. Let $\ell : E' \cup E \to \mathbb{R}$ be an edge-length function of them. We say the linkage P can *simulate* the graph G if each edge in G is spanned by at least one subpath of P, and no subpath of P properly joins two non-adjacent vertices in G. We formalize the notion of simulation by a mapping ϕ that maps each vertex V in P to a *point* in G as follows. For each edge $e' = \{u, v\} \in E'$, we consider e' as a line segment (u, v) of length $\ell(e')$. Then the *intermediate point* p at distance $t\ell(e')$ from u is denoted by $p = tv + (1-t)u$, where $0 < t < 1$. We note that the endpoints of an edge e' are not considered intermediate points of e'. Now we first define a set of *points* in G by V' and all intermediate points on edges of E'. Then we define a mapping ϕ from V to points of G as follows. To make it clear, we first divide V in P into two subsets V_o and V_i such that each vertex in V_o is mapped to a vertex in V', and each vertex in V_i is mapped to an intermediate point of G. In our problems, we assume that $\phi(v_0)$ and $\phi(v_n)$ should be in V_o. That is, P should start and end at vertices in G. Depending on the restrictions, we consider some different simulation problems as follows.

Weighted Eulerian path problem We consider the mapping ϕ from V_o to V' that satisfies some conditions as follows; (1) for every $v' \in V'$, there is at least one vertex $v \in V_o$ with $\phi(v) = v'$; (2) for each edge $e' = \{v', u'\} \in E'$, there is a pair of vertices v_i and v_j in V_o such that (2a) $\ell(e') = \sum_{k=i}^{j-1} \ell(e_k)$, and (2b) there is no other vertex v_k is in V_o between v_i and v_j. Intuitively, each edge e' in G corresponds to a subpath $(v_i, \ldots, v_j)$ in P, and vice versa. In other words, some vertices in P are mapped to some intermediate points in G, and the corresponding joints of the linkage are fixed when the linkage P simulates the target graph G. We note that by the length condition (2a), we can assume that when the subpath $(v_i, \ldots, v_j)$ in P simulates an edge $e' = \{\phi(v_i), \phi(v_j)\}$, it is not allowed to span it in a zig-zag way. We also add one condition: (3) for each edge (v_k, v_{k+1}) in P, $\{\phi(v_k), \phi(v_{k+1})\}$ should

be on an edge e' in G. That is, there is a subpath $(v_i, \ldots, v_k, v_{k+1}, \ldots, v_j)$ with $i \leq k < k+1 \leq j$ such that (3a) $e' = \{\phi(v_i), \phi(v_j)\}$ and $\phi(v_k)$ is on an intermediate point on e' for each $i < k < j$. In other words, all edges in P are used for spanning some edges in G, and there is no other subpath in P joining two endpoints in G by the length condition (2a). Then we say that the linkage P can *simulate* the graph G if there is a mapping ϕ satisfying the conditions (1), (2), and (3). This problem can be seen as the Eulerian path problem with edge weights.

Definition 7. *The* weighted Eulerian path problem *asks if there is an Eulerian path of a graph G spanned by a path P with length consistency.*

We show proved that the weighted Eulerian path problem is strongly NP-hard even if edge lengths are quite restricted. We thus consider two variants of the weighted Eulerian path problem.

The first variant is the elastic linkage problem.

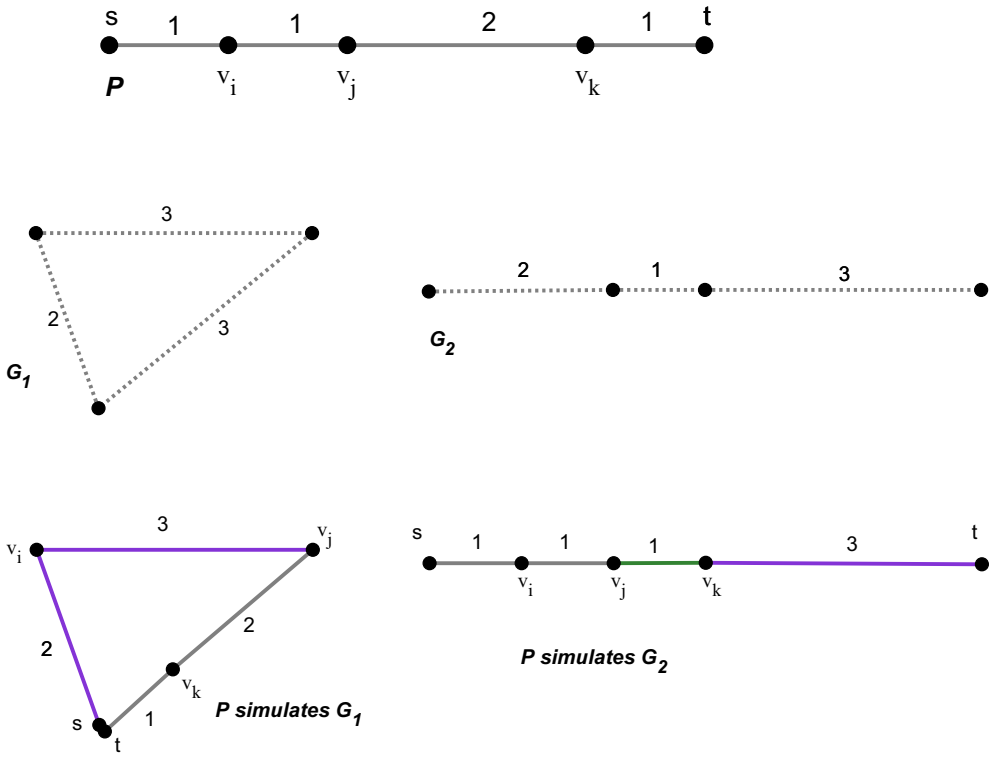

Figure 2.3: A simple example. A elastic linkage P simulates the given target graphs G_1 and G_2 as shown in the figure. Any edge in P can be stretched (colored purple) or shrank (colored green) to match some certain edge lengths in G_1 and G_2.

Elastic linkage problem In this problem, we allow all edges in P to be elastic to simulate the path G by the path P. As shown in Figure 2.3, some edges in P are stretched (colored purple) or shrank (colored green) to match some certain edge lengths in the target graphs.

Definition 8. *The* elastic ratio ρ *of an edge* e *is* $\max\{l'/l, l/l'\}$*, where* l *is the length of the edge* $e = \{u, v\}$ *in a path* P *and* l' *is the length of the segment* $\{\phi(u), \phi(v)\}$ *in a graph* G*.*

Intuitively, the length of edge e is changed from l on P to l' on G. For a given graph $G = (V', E')$ and a path $P = (V, E)$, it is easy to observe that P can simulate G (with elastic edges) if and only if G has an Eulerian path and $|V'| \leq |V|$. When G has an Eulerian path by P with elastic edges, the *elastic ratio* of the mapping is defined by the maximum elastic ratio of all edges in P.

Definition 9. *The* elastic ratio of a mapping ϕ *is* $\max\{\rho_1, \rho_2, \ldots, \rho_{n-1}\}$*, where* ρ_i *is the elastic ratio of the edge* e_i *in a path* P*.*

Then the elastic linkage problem asks to minimize the elastic ratio of the mapping from P to G for given G and P. We show that the elastic linkage problem can be solved in polynomial time by dynamic programming when G is a path.

The other variant of the weighted Eulerian path problem is the traverse problem by a path.

We first consider the general cover problem. Since a linkage is flexible at each joint, so we can fold it back and forth. That is, when we simulate a target graph G by a given linkage P, an edge in G can be covered twice or more, see Figure 2.4. We noticed that the general cover problem of G by P is NP-complete even if G is an edge, we thus consider a more restricted case.

Traverse problem by a path In this variant, we allow the mapping to map two subpaths of P to an edge of G, however, we do not allow P to be elastic, or its ratio is fixed to 1. For a given graph $G = (V', E')$ and a path $P = (V, E)$, when all edges have a unit length, it is easy to observe that P can simulate G in this manner if and only if G is connected and $2|E| = |E'|$: We first perform the depth-first search on G and traverse this search tree (see Figure 2.5). We also consider its edge-weighted version as the *traverse problem* for a graph G and a path P. In this research, in fact, we only investigate the cases that G is a tree.

Definition 10. *A* tree *is a connected acyclic graph.*

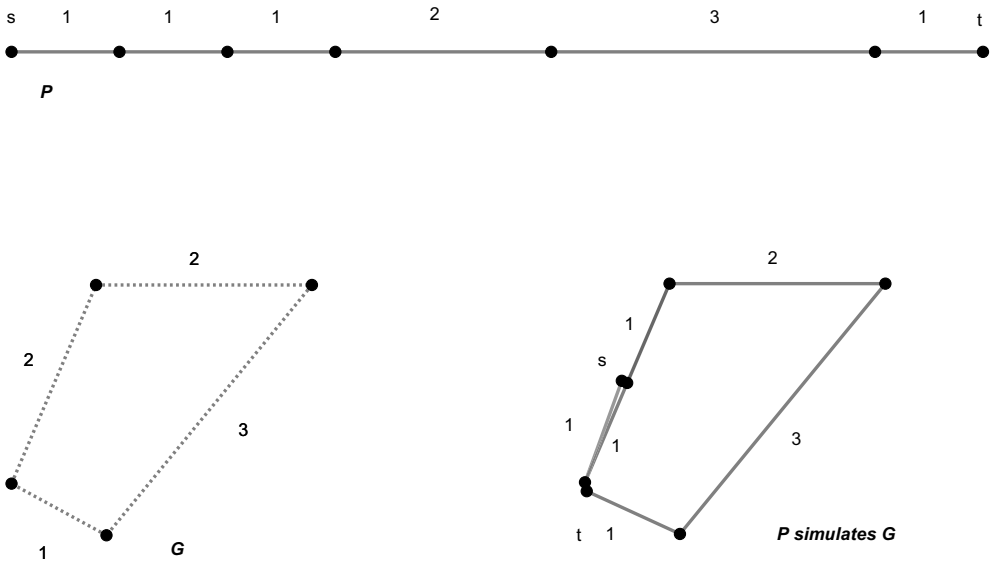

Figure 2.4: General cover problem. A linkage P can simulate a given target graph G as shown in the figure. An edge in G can be covered twice or more by some links in P.

Precisely, we consider the following problem: For a given tree G and a path P (with edge lengths), the traverse problem asks if G has a trail by P such that each edge of G is traversed exactly twice. (We note that trees form a representative class of graphs that have no Eulerian paths.) We first mention that this problem is quite easy when each edge has a unit length. The answer is yes if and only if $G = (V, E)$ is connected and P contains $2|E|$ edges. From the practical viewpoint, it seems to be reasonable when we simulate a graph by a linkage. However, this problem is still strongly NP-hard even in quite restricted cases; (1) G is a star, and P consists of edges of only two different lengths, and (2) G is a spider, and all edges are of two different lengths. On the other hand, the problem is polynomial-time solvable when G is a star and its edge lengths are of k different values for some fixed k.

Definition 11. *A* complete bipartite graph $K_{n,m}$ *is a* $G = (X, Y, E)$ *such that* $|X| = n$, $|Y| = m$, *and every pair of a vertex in* X *and a vertex in* Y *is joined by an edge.*

Definition 12. *A* star *is a complete bipartite graph* $K_{1,n-1}$.

Definition 13. *A* spider *is a tree that has only one vertex of degree greater than 2.*

Definition 14. *The* center *in a star or a spider is the unique vertex of degree greater than or equal to 3.*

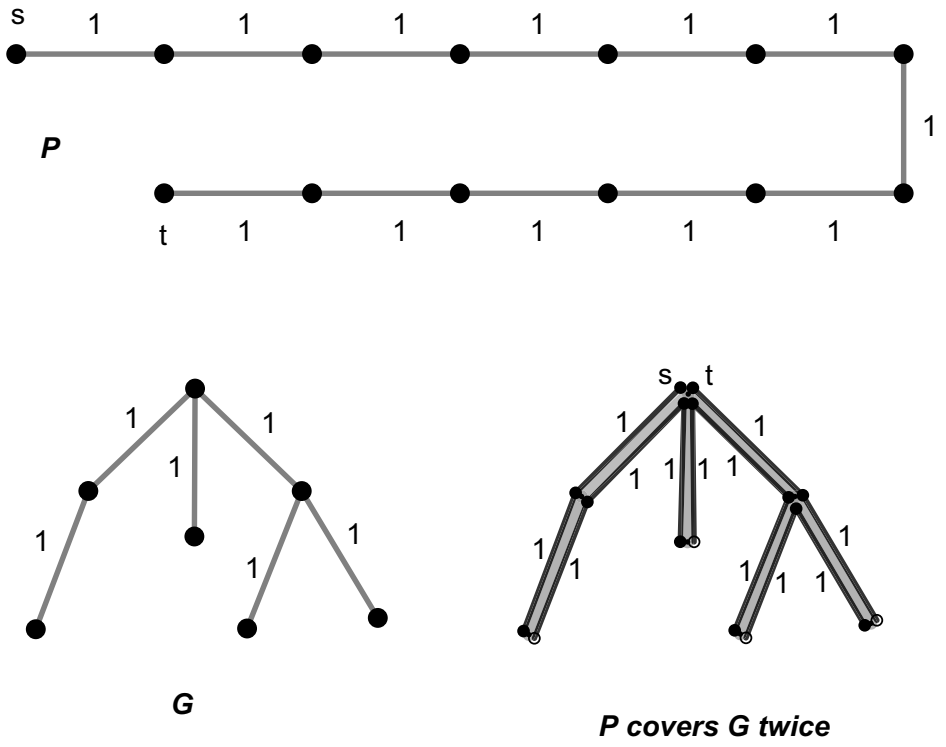

Figure 2.5: A tree G is covered by a linkage P twice by the depth first serach manner.

Linkage Simulation Problem:

We now summarize and define this linkage simulation problem in another way.

For a given path (linkage) $P = (V, E, \ell)$, and a target graph $G = (V', E', \ell')$, with $|V| = n$ and $|V'| = n'$, we define $\tilde{P} = (\tilde{V}, \tilde{E})$ and $\tilde{G} = (\tilde{V}', \tilde{E}')$ as follows:

$\tilde{V}$=$\{\tilde{v}_1, \tilde{v}_2, \tilde{v}_3, \ldots, \tilde{v}_n\} \subset \mathbb{R}$ where

$$\tilde{v}_i = \begin{cases} 0 & \text{if } i = 1 \\ \tilde{v}_{i-1} + \ell(\{v_{i-1}, v_i\}) & \text{if } i > 1 \end{cases}$$

$\tilde{V}' = \{\tilde{u}_1 = \{1, 0, 0, 0, \ldots\}, \tilde{u}_2 = \{0, 1, 0, 0, \ldots\}, \tilde{u}_3 = \{0, 0, 1, 0, \ldots\}, \ldots, \tilde{u}_{n'} = \{0, 0, \ldots, 0, 1\}\} \subset \mathbb{R}^{n'}$.

$\tilde{E}' = \{\tilde{u}_i, \tilde{u}_j\}$ for each $\{u_i, u_j\} \in E'$.

$S \subset \mathbb{R}^{n'}$ is the union of all segments $\tilde{u}_i\tilde{u}_j$ with $\{\tilde{u}_i, \tilde{u}_j\} \in \tilde{E}'$.

Let $f\colon \tilde{v}_1\tilde{v}_n \to S$ be a continuous function that is linear on the segment $\tilde{v}_i\tilde{v}_{i+1}$ for all $1 \le i < n$, such that $f^{-1}(\{\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \ldots, \tilde{u}_{n'}\}) \subseteq \{\tilde{v}_1, \tilde{v}_2, \tilde{v}_3, \ldots, \tilde{v}_n\}$.

Let $\nabla f(x)$ denote the gradient of f at point $x \in \tilde{v}_1\tilde{v}_n \subset \mathbb{R}$.

Define $\tilde{\ell}\colon S \setminus \tilde{V}' \to \mathbb{R}^+$ such that, for any internal point $x \in \tilde{u}_i\tilde{u}_j \subseteq S$, we have $\tilde{\ell}(x) = \ell'(\{u_i, u_j\})$.

The stretch factor of a point $x \in \tilde{v}_1\tilde{v}_n \setminus \tilde{V}$ is $\rho(x) = \dfrac{\|\nabla f(x)\| \cdot \tilde{\ell}(f(x))}{\sqrt{2}}$.

The elastic ratio of a point $x \in \tilde{v}_1\tilde{v}_n \setminus \tilde{V}$ is $\rho'(x) = max\{\rho(x), \dfrac{1}{\rho(x)}\}$.

***Weighted Eulerian Path Problem*:**

- S is covered once, i.e., $\forall x \in S \setminus f(\tilde{V})$, we must have $|f^{-1}(x)| = 1$.
- f maps the first and last vertex of $\tilde{P}$ to vertices of $\tilde{G}$, i.e., $f(\{\tilde{v}_1, \tilde{v}_n\}) \subseteq \tilde{V}'$.
- Stretching is not allowed, i.e., $\rho(x) = 1$ for all $x \in \tilde{v}_1\tilde{v}_n \setminus \tilde{V}$.

***Elastic Linkage Problem*:**

- S is covered once, i.e., $\forall x \in S \setminus f(\tilde{V})$, we must have $|f^{-1}(x)| = 1$.
- f maps the first and last vertex of $\tilde{P}$ to vertices of $\tilde{G}$, i.e., $f(\{\tilde{v}_1, \tilde{v}_n\}) \subseteq \tilde{V}'$.

***General Cover Problem*:**

- S is covered, i.e., $f(\tilde{v}_1\tilde{v}_n) = S$.
- Stretching is not allowed, i.e., $\rho(x) = 1$ for all $x \in \tilde{v}_1\tilde{v}_n \setminus \tilde{V}$.

***Traverse Problem of a Tree by a Path (Cover Twice)*:**

- G is a tree.
- S is covered twice, i.e., $\forall x \in S \setminus f(\tilde{V})$, we must have $|f^{-1}(x)| = 2$.
- f maps the first and last vertex of P to vertices of G, i.e., $f(\{\tilde{v}_1, \tilde{v}_n\}) \subseteq \tilde{E}'$.

- Stretching is not allowed, i.e., $\rho(x) = 1$ for all $x \in \tilde{v}_1\tilde{v}_n \setminus \tilde{V}$.

In this thesis, we will often use the following problems to show the hardness of our problems: *Hamiltonian path problem* and *3-Partition problem*.
A *Hamiltonian path* is a path between two vertices of a graph that visits each vertex of the graph exactly once. The problem of finding a Hamiltonian path of a given general graph is well known to be NP-complete [28].
In this research, we focus on square grid graphs, and more specifically, rectangular grid graphs. Therefore, the NP-completeness of the Hamiltonian path problem in grid graphs shown in [3, 4, 8] is important in our context.
Itai et al. also proved that the Hamiltonian path problem for general grid graphs is NP-complete [21]. We will use this result, as well as the *3-Partition problem*.
The 3-Partition problem is to decide if a given multiset $A = \{a_1, a_2, \ldots, a_{3m}\}$ of $3m$ positive integers can be partitioned into m subsets, each of which has the same sum B.

3-Partition Problem

Input: An integer B and a multiset A of $3m$ integers $A = \{a_1, a_2, \ldots, a_{3m}\}$ with $B/4 < a_i < B/2$.

Output: Determine if A can be partitioned into m multisets $S_1, S_2, \ldots, S_m$ such that $\sum_{a_j \in S_i} a_j = B$ for every i.

Without loss of generality, we can assume that $\sum_{a_i \in A} a_i = mB$, and $|S_i| = 3$.
It is well known that the 3-Partition problem is strongly NP-complete [18].

Chapter 3

Protein Folding Prediction and Design in Grid Graphs

In this Chapter we focus on blueprint graphs G which are *grid graphs*, i.e., they are obtained from regular tilings of the plane (sometimes these are also called *lattice graphs*). This is the typical setting of the standard HP model.

3.1 Monochromatic Path

Let us first assume that the path P is monochromatic, i.e., all its vertices have the same color (say, blue). We will prove that the embedding problem is already NP-hard in such a simple setting.

Theorem 1. *The bicolored path embedding problem is NP-complete even if the blueprint G is a (square, triangular, or hexagonal) grid graph, and P is a monochromatic path.*

Proof. Let P be a path that consists of only blue vertices, and G be a bicolored grid graph (square, triangular or hexagonal) containing the same number of blue vertices. Examples are shown in Figure 3.1.

We reduce the Hamiltonian path problem for square grid graphs, triangular grid graphs, or hexagonal grid graphs to our problem [3, 4, 8]. Precisely, we can use the same strategy in [3, 4, 8] to prove the theorem. In [3, 4, 8], they reduced the Hamiltonian path problem in a 3-regular planar graph to one in a subset of a grid graph. More precisely, for a given 3-regular planar graph, they represent the graph by a subset of a grid graph. Then the problem asks if we can visit all grid points exactly once on the subset of a grid graph. In our case, the subset of a grid graph can be represented by blue vertices, and we fill the other grid points by red vertices. Then it is straightforward that

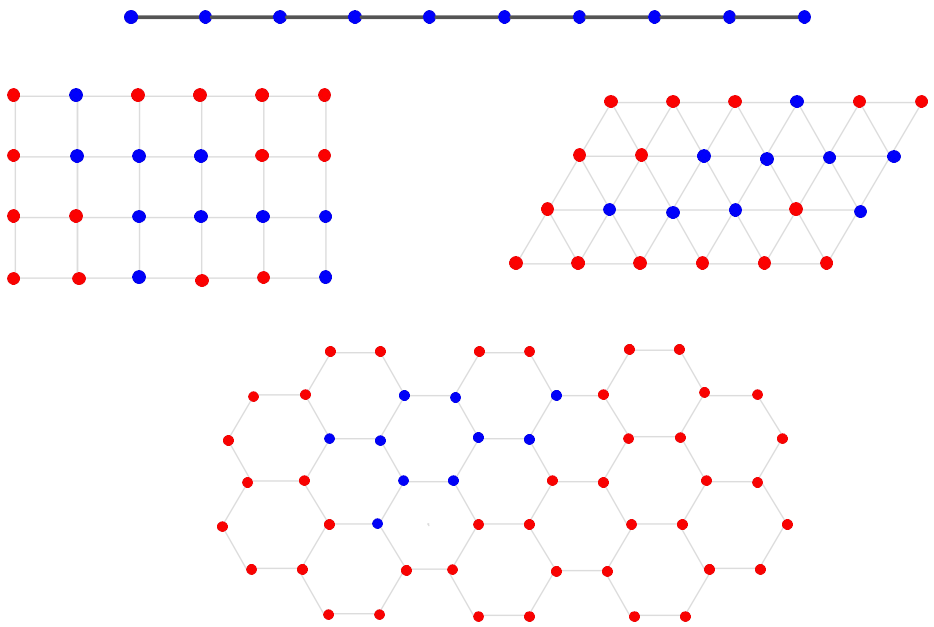

Figure 3.1: A bicolored path consists of ten blue vertices and three types of bicolored grid graphs that contain ten blue vertices.

the results in [3, 4, 8] can be translated into the bicolored path embedding problem. Thus we can conclude that the bicolored path embedding problem is NP-complete each of square grid graphs, triangular grid graphs, and hexagonal grid graphs. □

3.2 Embeddings in Rectangular Grids

In this section, we focus on *rectangular* blueprints, i.e., blueprints that are rectangular grid graphs, as defined in the previous section. As already mentioned, this is the typical setting of the traditional HP model of protein folding.

3.2.1 Bijective Embeddings

Let us first consider the case where the bicolored blueprint G is a "precise" description of a protein, i.e., it has to be matched exactly by the amino acid sequence represented by the bicolored path P. In other words, G and P have the same number of vertices, and the embedding should therefore be bijective.

As observed at the end of Section 3.1, the non-bijective bicolored embedding problem in a rectangular grid is NP-complete. On the other hand, the bijective embedding problem in a rectangular grid is polynomial-time solvable if P and G are monochromatic: indeed, this is equivalent to the Hamiltonian path problem in a rectangular grid, which is solved in [21].

In the following theorem, we will close the gap between the two aforementioned results: We will show that the *bijective bicolored* path embedding problem is NP-complete.

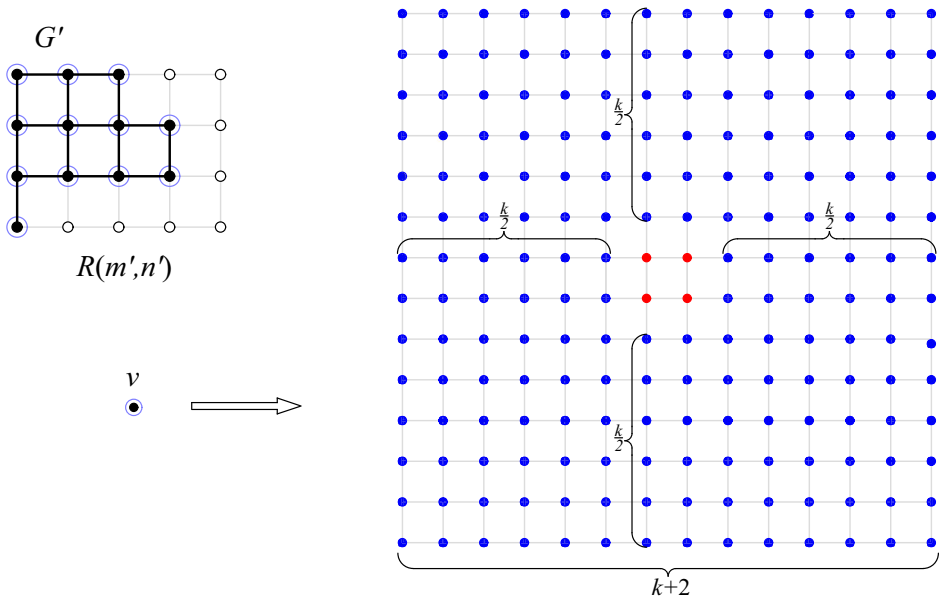

Figure 3.2: Example of a grid graph G' and the transformation of a vertex v of G' into the $(k+2) \times (k+2)$ block B_v (in this example, $m' = 5$, $n' = 4$, and $k = 12$)

Theorem 2. *The bicolored path embedding problem is NP-complete even if the blueprint G is a rectangular grid with the same number of vertices as the path P.*

Proof. We will give an NP-hardness reduction from the Hamiltonian path problem on grid graphs in the square lattice, which is NP-complete [21].

Construction of the reduction We start from a rectangular grid $R(m', n')$ with an induced subgraph G' (which is an instance of the Hamiltonian path problem), and we construct the blueprint G by "expanding" each vertex v of $R(m', n')$ into a $(k+2) \times (k+2)$ block B_v (where k is a large-enough even

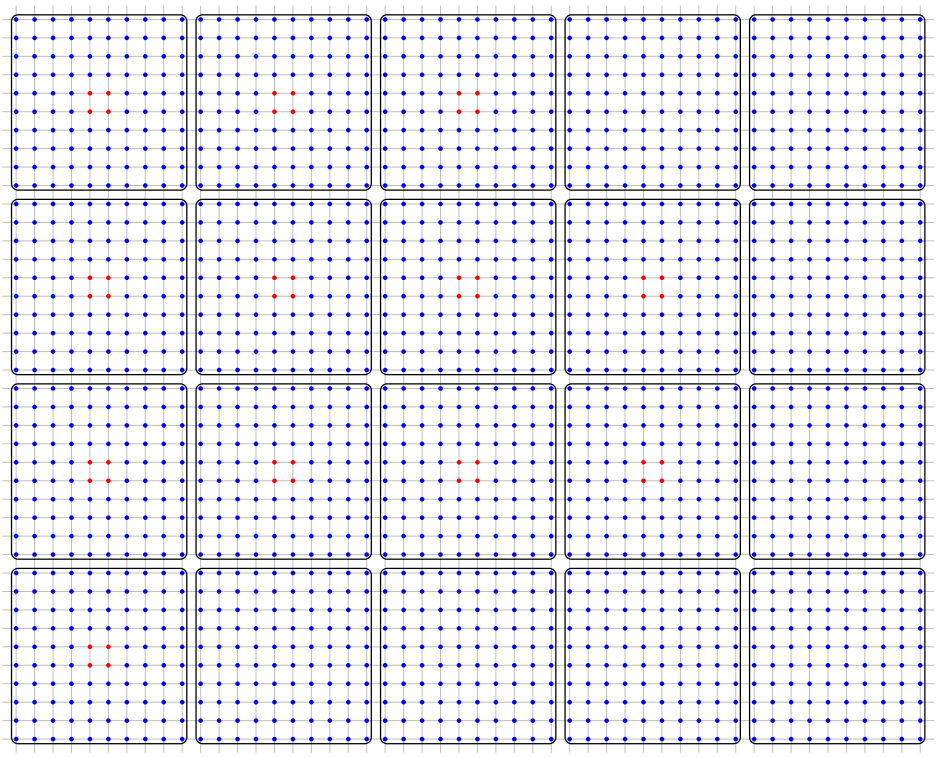

Figure 3.3: Complete construction of G from the graph G' of Figure 3.2 (now with $k = 8$): each of the circled blocks represents a vertex in the original graph G'

constant, defined later). If v is not a vertex of G', then all vertices of B_v are blue; if v is a vertex of G', then B_v is illustrated in Figure 3.2 (right): its four central vertices are red, and all other vertices are blue. The order of G is therefore $(k+2) \cdot m' \times (k+2) \cdot n'$; an example of the full construction is shown in Figure 3.3.

The path P is constructed as follows. Let P' be a path consisting of 4 red vertices followed by $2k$ blue vertices. P is made up of n consecutive copies of P', where n is the order of G', followed by a trail of blue vertices such that the total length of P matches the order of G. Namely, the final trail of P consists of $(k+2)^2 \cdot m'n' - (2k+4)n$ blue vertices. Refer to Figure 3.4 for an example.

Embedding the first part of the path In order to embed P into G, we have to start from a set of four red vertices in some block B_v, and then move to another set of four red vertices in some other block B_w. Since we must

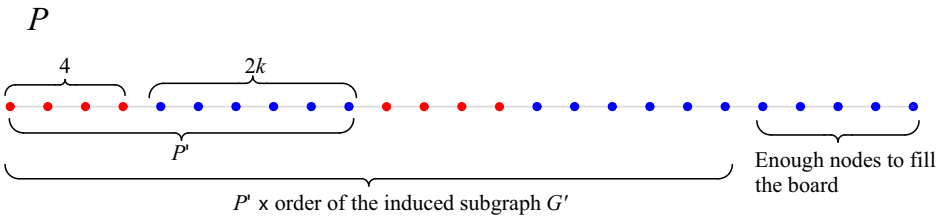

Figure 3.4: Construction of the path P (in this example, $k = 3$ and G' has order 2)

traverse exactly $2k$ blue vertices between these two red sets, this is possible only if v and w are adjacent in G' (note that a "diagonal" move would take $2k + 1$ steps on blue vertices). Thus, embedding P into G is impossible if G' is not Hamiltonian.

Assume now that G' is Hamiltonian. We can embed all copies of P' into G by "mimicking" a Hamiltonian path in G' and moving from one set of red vertices to the next by covering the $2 \times k$ rectangle between them in a zig-zag fashion. Eventually, the region of G covered by all the copies of P' looks like a winding "tube" of width 2, as sketched in Figure 3.5.

Embedding the trailing blue vertices Now we have to cover the remaining part of G with the trailing sequence of blue vertices of P. Observe that this part of G is connected, because the copies of P' were embedded according to a Hamiltonian path, which has no cycles, and therefore did not disconnect G.

In order to cover this last region, we partition it into maximal "horizontal rectangles," i.e., in such a way that no two rectangles touch each other along vertical edges. Figure 3.5 shows an example of the partition. Then we do a depth-first traversal of these rectangles. When we visit a new rectangle R (perhaps coming from its parent rectangle R'), we cover R as exemplified in Figure 3.6: we further divide it into smaller rectangular "tiles," one for each unvisited neighboring rectangle. After completely covering a tile, we continue the depth-first traversal by visiting its adjacent rectangle R'' in the partition. When we backtrack from R'' and get back to R, we move to the next tile of R, and so on.

Note that the last tile we visit is again adjacent to R', and thus we are able to backtrack once all tiles of R have been completely covered (if R is the root of the spanning tree, we just terminate). It is straightforward to prove by induction that this embedding algorithm completely covers all rectangles in the partition.

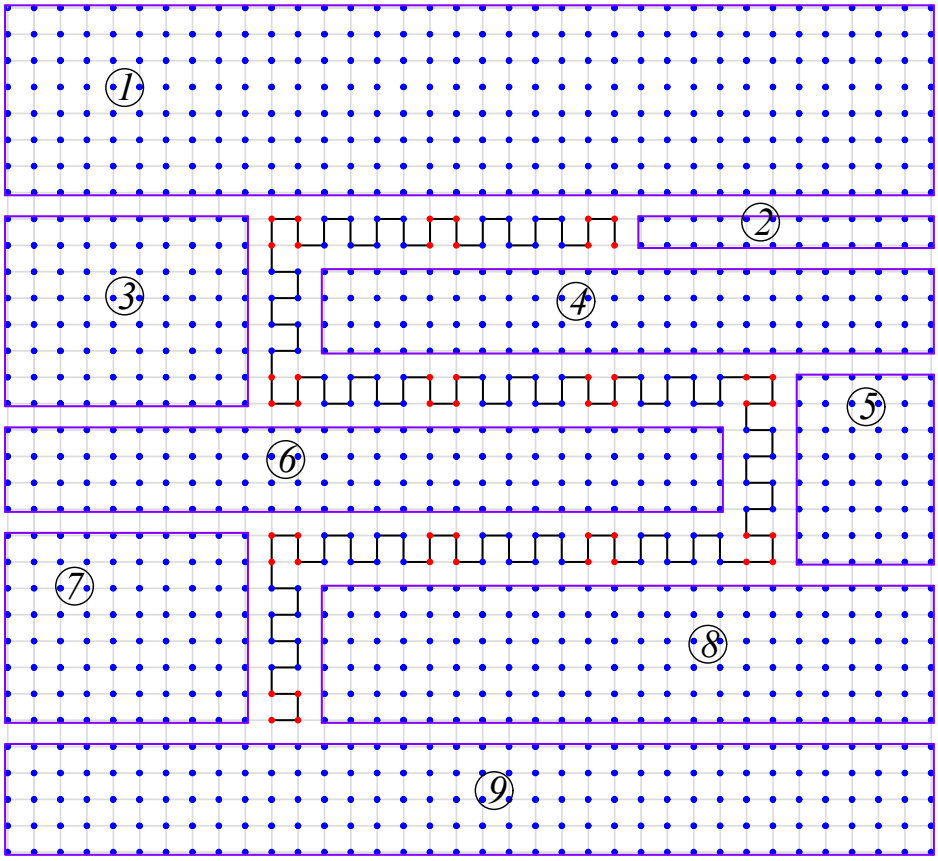

Figure 3.5: Example of a partition into rectangles of the region not covered by the zig-zagging copies of P'

Detailed construction of tiles We still need to prove that it is possible to construct the tiles in such a way that each of them can be covered completely before moving on to the next rectangle. We will use a result from [21], where the grid graphs containing a Hamiltonian path with assigned endpoints have been characterized. The characterization includes some special cases of small order, but since our k is a large constant, we can ignore them.

What we can gather from [21] is that, if the order (i.e., the number of vertices) of a tile is even and one of its sides is longer than four vertices, then there is a Hamiltonian path in the tile with any assigned endpoints having odd distance along the grid. Because k is a large even constant, we can indeed subdivide each rectangle in the appropriate number of tiles, arranged as exemplified in Figure 3.6, each of which has an even order and at least

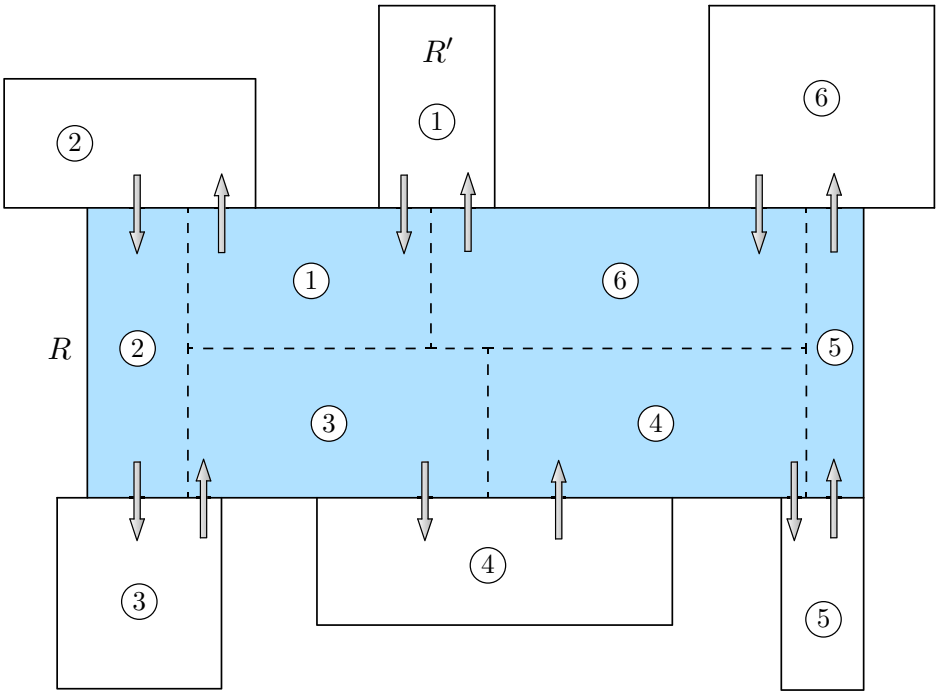

Figure 3.6: The shaded rectangle R is subdivided into six tiles, one for each neighboring rectangle. The numbers and arrows show the order in which tiles and neighboring rectangles are covered. The path comes from the parent rectangle R', covers a tile, and moves to the next unvisited rectangle, etc. When R is fully covered, the path returns to the parent R'.

one side longer than four vertices (choosing $k = 32$ abundantly suffices).

When covering a tile, we want to start on an edge and either end on the same edge or on the opposite edge. For example, referring to Figure 3.6, when covering tile 1 we want to enter and exit from the same edge, but when covering tile 2 we want to enter from an edge and exit from the opposite one. So, it is sufficient to choose any pair of starting and ending vertices (along the appropriate edges) having odd distance, and the characterization in [21] guarantees that there is a Hamiltonian path in the tile having these starting and ending vertices. It follows that we can embed P into G. □

3.2.2 Fixed-Height Rectangular Blueprints

We can contrast our previous hardness results with an embedding algorithm that runs in polynomial time, provided that the blueprint G is a rectangular grid of fixed height k.

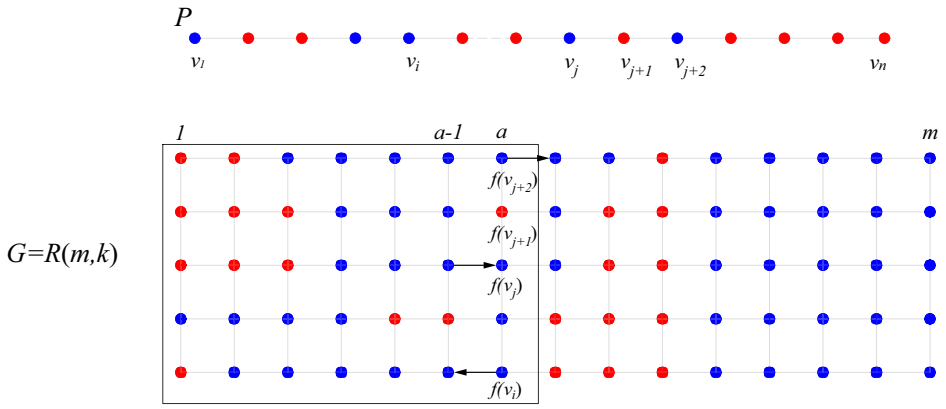

Figure 3.7: Example of a sub-problem of the dynamic-programming algorithm for rectangular blueprints. The sub-problem specifies which vertices of the path P should be embedded in the ath column of G, and where: this is indicated by the function f. Each arrow represents a *direction bit*: the vertex v_{i+1} should be mapped to the left of $f(v_i)$, etc. The value of the direction bit at $f(v_{j+1})$ is irrelevant, because both v_j and v_{j+2} are mapped to the ath column. The sub-problem asks if there exists a partial embedding of P in the sub-grid going from the first column to the ath column that matches vertex colors and satisfies the constraints imposed by f and the direction bits.

Lemma 1. *For all integers $a > 2$ and $b > 0$,*

$$\min\{(a+1)^b, (b+1)^a\} < e \cdot a^b.$$

Proof. It is well known from elementary calculus that, if $k > 0$, the function $f_k(x) = (1 + k/x)^x$ is monotonically increasing for $x > 0$, and its limit as x approaches $+\infty$ is e^k. Hence, by rearranging terms, we have, for every $x, k > 0$,

$$(x+k)^x < e^k x^x. \tag{3.1}$$

If $a \geq b$, we plug $k = 1$ and $x = a$ in Equation 3.1, obtaining

$$(a+1)^b = ((a+1)^a)^{\frac{b}{a}} < (e \cdot a^a)^{\frac{b}{a}} = e^{\frac{b}{a}} \cdot a^b \leq e \cdot a^b.$$

If $a < b$, by a similar reasoning, we have $(b+1)^a < e \cdot b^a$. To conclude, it is now sufficient to prove that $b^a \leq a^b$. This is done by plugging $k = b - a$ and $x = a$ in Equation 3.1:

$$b^a = (a + (b-a))^a < e^{b-a} \cdot a^a < a^{b-a} \cdot a^a = a^b,$$

recalling that, by assumption, $e < 3 \leq a$. □

Theorem 3. *Given a bicolored rectangular grid G of order $m \times k$ and a bicolored path P of order n, the embedding problem for G and P can be solved in $O(2^k n^{2k} m)$ time.*

Proof. Let G be a bicolored $m \times k$ grid, and let P be a bicolored path of n vertices. If $n \leq 2$, the problem can be trivially solved in $O(km)$ time by searching G for one or two adjacent vertices with colors matching P. Hence, let us assume that $n > 2$.

Sub-problem specification Our approach is based on dynamic programming, where a sub-problem consists in embedding part of P (not necessarily all of P) into a sub-grid of G going from the first column to the ath column, with $1 \leq a \leq m$. A sub-problem's specification also contains a description of the intersection between a hypothetical embedding of P and the ath column of G: for each vertex w in the ath column, the sub-problem specifies which vertex v_i of P is mapped to w (if any), as well as an extra bit of information: the *direction bit*. This bit encodes whether the left or right neighbor of v_i along P should be mapped to the left neighbor of w (if such information is incompatible with the rest of the specification, the direction bit is ignored). Figure 3.7 sketches a sub-problem with a function f specifying which vertices of P are mapped into the ath column.

Thus, the total number of sub-problems is $2^k c_{n,k} m$, where the factor m represents the possible choices of a, and $c_{n,k}$ is the number of ways a subset of vertices of P can be injectively mapped into a column of G. Informally, we can say that $c_{n,k}$ is the number of ways two paths of length n and k (representing P and a column of G, respectively) can "intersect."

Solving sub-problems The output to a sub-problem is "Yes" if an embedding of part of P satisfying the given constraints exists, "No" if it does not exist, and "N/A" if the sub-problem specifies no intersection on the ath column, and it is not possible to embed P entirely to the left of the ath column (this implies that P must be embedded entirely to the right of the ath column, but we are still unable to determine if this is possible).

Solving a sub-problem S for column a amounts to finding a sub-problem S' for column $a-1$ with a "Yes" answer such that the specifications of S and S' are compatible. In other words, the mappings described by S and S' on columns a and $a-1$ should (i) match the colors in G and P, and (ii) match with each other. For example, assume that the sub-problem S is the one illustrated in Figure 3.7, where the direction bit at $f(v_i)$ indicates that the vertex v_{i+1} of P should be mapped to the left neighbor w' of $w = f(v_i)$ (where

w is in column a). Then, the sub-problem S' should agree with this specification: namely, its function f' should indicate that $f'(v_{i+1}) = w'$ (which is in column $a - 1$).

Full algorithm To summarize, the algorithm for solving a sub-problem S for column a is as follows:

- If $a = 1$, then:
  - If S specifies that the embedding of P does not intersect column 1, then return "N/A."
  - Else, if S specifies that the embedding of P intersects column 1 in a way that (i) matches vertex colors, (ii) whenever it maps two consecutive vertices v_i and v_{i+1} of P to column 1, it maps them to adjacent vertices, and (iii) the direction bits of S specify that the embedding of P continues to the right (whenever this makes sense), then return "Yes."
  - Else, return "No."
- If $a > 1$, and S specifies that the embedding of P does not intersect column a, then:
  - If there is a compatible sub-problem S' for column $a - 1$ with answer "Yes," then return "Yes" (by "compatible" we mean that the direction bits of S' imply that no vertex of P mapped to column $a - 1$ should have a neighbor mapped to column a).
  - Else, return "N/A."
- If $a > 1$, and S specifies that the embedding of P has some intersections with column a, then:
  - If S specifies that the embedding of P intersects column a in a way that (i) matches vertex colors, (ii) whenever it maps two consecutive vertices v_i and v_{i+1} of P to column a, it maps them to adjacent vertices, (iii) there is a sub-problem S' for column $a - 1$ with answer "Yes" or "N/A" that is compatible with S, and (iv) if $a = n$, the direction bits of S specify that the embedding of P continues to the left (whenever this makes sense), then return "Yes."
  - Else, return "No."

Optimizations and remarks As an optimization, we do not have to look up *all* sub-problems S' for column $a-1$, but only the ones whose direction bits are compatible with S. In other words, we only need to choose which vertices of P are mapped to the column $a-1$ and where, and the correct direction bits can be inferred. Hence, in order to solve S, it is sufficient to look up at most $c_{n,k}$ sub-problems.

Also, for each sub-problem S', the compatibility test between S' and S can be done in constant amortized time. Indeed, the sub-problems S' are enumerated by locally changing the function that maps points on the column $a-1$ to vertices of P; as each of these local changes takes place, the corresponding compatibility check is performed in constant time. Hence, S can be solved in $O(c_{n,k})$ time.

As there are $2^k c_{n,k} m$ sub-problems in total, it takes $O(2^k c_{n,k}^2 m)$ time to solve all of them. In the end, the algorithm returns "Yes" if there is a sub-problem for $a = n$ with a "Yes" answer; it returns "No" otherwise.

Correctness and running time The correctness of this algorithm can be proved straightforwardly by induction. Note that the distinction between "N/A" and "Yes" implies that, if the final answer is "Yes," then at least some vertices of P have indeed been embedded somewhere in G. If this is the case, then the compatibility tests between columns guarantee that all of P has been correctly embedded.

In order to show that our algorithm has the desired running time, it remains to prove that $c_{n,k} = O(n^k)$. Recall that $c_{n,k}$ is the number of ways P can intersect a column of G. We can give two upper bounds on this number. Each of the k vertices in a column of G may intersect one of the n vertices of P or none of them. This yields at most $(n+1)^k$ different configurations in total, and thus $c_{n,k} \leq (n+1)^k$. Note that this is insufficient to conclude that $c_{n,k} = O(n^k)$, because k is not a constant. Let us give a second upper bound: each of the n vertices of P may be mapped either to one of the k vertices in the given column of G or to a different column. This yields $c_{n,k} \leq (k+1)^n$. Now, Lemma 1, with $a = n$ and $b = k$, gives

$$c_{n,k} \leq \min\{(n+1)^k, (k+1)^n\} < e \cdot n^k = O(n^k),$$

as required (recall that we were assuming that $n > 2$). □

We immediately have the following:

Corollary 1. *The bicolored path embedding problem where the blueprint G is a rectangular grid, parameterized according to the height of G, is in XP.*

Proof. According to Theorem 3, if G has order $m \times k$ and P has order n, there is an algorithm that solves the embedding problem for G and P in $O(2^k n^{2k} m)$ time. Now, if k is a constant, the running time of the algorithm is $O(n^{2k} m)$, hence polynomial. □

3.3 Maximizing Red-Red Contacts

Finally, let us turn to the problem of maximizing red-red contacts in the context of the bicolored path embedding problem. Recall that, according to the HP model of energy, an amino acid chain tends to fold in a way that maximizes the number of H nodes that are close together in the folded state, even if they are not adjacent along the chain. In other words, when G and P are given, we seek an embedding of P into G that covers a large number of adjacent red vertices of G without traversing the edges that connect them with each other.

Definition 15. *A* red-red contact *in an embedding of P into G is a pair of adjacent red vertices u, v in G such that the embedding of P covers both u and v, but does* not *contain the edge $\{u, v\}$.*

The problem of maximizing red-red contacts in the bicolored path embedding problem is also NP-hard, even when restricted to instances where the path P is guaranteed to be embeddable into G, and even when G is a solid grid graph (in a square, triangular, or hexagonal lattice).

Theorem 4. *Given a bicolored solid grid graph G and a bicolored path P that can be embedded in G, it is NP-hard to find an embedding of P in G that maximizes red-red contacts.*

Proof. We will describe a reduction from the Hamiltonian path problem in the case of a square lattice. A similar construction can be used for triangular and hexagonal lattices, as well.

Given a connected input grid graph G' on n vertices, we construct a rectangular grid graph G by juxtaposing three *blocks*, each of which is in turn a rectangular grid graph. Figure 3.8 shows a sketch of the whole construction.

Block 2 of G is constructed by completing G' to a rectangular grid graph, as we did at the end of Section 3.1. That is, Block 2 of G is the smallest rectangular graph $R(a, b)$ containing (an isomorphic copy of) G' as an induced subgraph; we color in blue the n vertices of this subgraph, and we color in red the $r = ab - n$ remaining vertices in $R(a, b)$.

Next we define $k = r + b + 1$, and we construct Block 1 as a grid graph isomorphic to $R(\lceil k/b \rceil, b)$ whose vertices are all red. Note that Block 1 has at least k vertices.

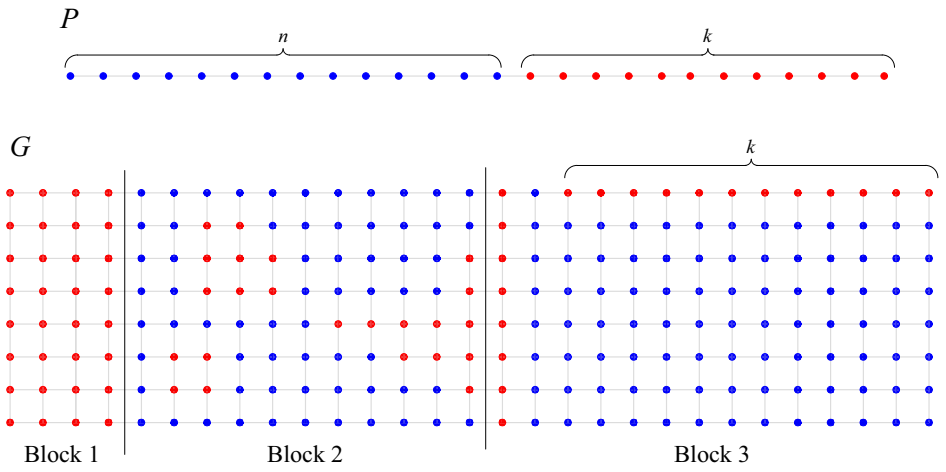

Figure 3.8: Sketch of the NP-hardness reduction for the problem of maximizing red-red contacts (the value of n should match the number of blue vertices in Block 2)

Block 3 of G is isomorphic to $R(\max\{k,n\}+2,b)$, and is colored as shown in Figure 3.8. Namely, the column adjacent to Block 2, i.e., the leftmost column, is all red; each of the k rightmost columns has the topmost vertex in red and all other vertices in blue; all other columns are entirely blue.

Finally, the path P consists of n blue vertices followed by k red vertices.

Without loss of generality, we may assume that, if a Hamiltonian path exists in G', one of its endpoints s must be on the perimeter of the bounding rectangle of G'. (This is a well-known fact in Hamiltonicity theory; see for example [21].) When constructing G, we can embed G' into Block 2 in such a way that s is in the column adjacent to Block 1. Thus, if G' has indeed a Hamiltonian path, we can embed the blue part of P in Block 2 and the red part of P in Block 1, which produces a large number of red-red contacts.

On the other hand, if G' does not have a Hamiltonian path, we can only embed P in Block 3, which yields no red-red contacts. This is because embedding some red vertices of P in Block 1 would force us to embed all the blue vertices in Block 2, which is impossible because G' is not Hamiltonian. Also, if we embedded some red vertices in the leftmost column of Block 3, we would have to embed all of them in this column or in Block 2 (because G' is connected, and thus there is no path of red vertices connecting Block 1 with Block 3). However, there are only $r+b$ red vertices in this region, and therefore we cannot fit all the $k = r+b+1$ red vertices of P. The only possibility is to embed these k red vertices in the topmost row of Block 3, which always yields a feasible embedding, as Block 3 has a large-enough blue

region to fit the n blue vertices of P, as well.

In conclusion, P can always be embedded in G; however, finding an embedding that produces any red-red contacts at all is NP-hard. □

We also consider a complementary problem of constructing a path P that embeds in a given bicolored graph G maximizing red-red contacts, which is related to the protein design. The Figure 3.9 shows a simple example. We are given a bicolored grid G, the goal is to find a Hamiltonian path P such that embedding P in a grid in such a way as to maximize red-red contacts yields again G.

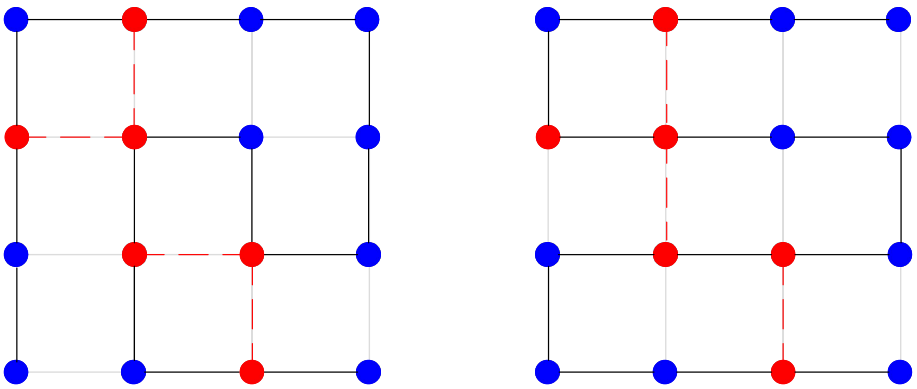

Figure 3.9: Two Hamiltonian paths in a bicolored grid with different number of H-H contacts.

We call this problem ***bicolored synthesis problem***. We will introduce the result we got about this problem in the next chapter.

Chapter 4

Protein Folding Prediction and Design in General Graphs

In this chapter, we consider two different models. We study both bicolored path embedding problems and linkage simulation problems in general graphs.

4.1 Bijective Embedding in a Dense Graph

Let us first consider the case where the bicolored blueprint G is a "precise" description of a protein, i.e., it has to be matched exactly by the amino acid sequence represented by the bicolored path P. In other words, G and P have the same number of vertices, and the embedding should therefore be bijective.

We will show that the embedding problem is NP-hard even if G is a dense graph. This is a somewhat surprising result: intuitively, a blueprint with many edges should allow greater leeway in the construction of an embedding of P. As it turns out, a greater amount of freedom does not necessarily translate into our ability to easily find embeddings.

Theorem 5. *The bicolored path embedding problem is NP-complete even if the blueprint G is a dense graph of the same size as the path P.*

Proof. We give a reduction from the strongly NP-complete *3-Partition* problem [18]. We recall that the input to the 3-Partition problem is a multiset of $3m$ positive integers $\{a_1, a_2, \ldots, a_{3m}\}$, and the goal is to decide whether it can be partitioned into m multisets of equal sum S. Our reduction is sketched in Figure 4.1.

The path P has length $m \cdot (S+1)$, and is made up of m consecutive copies of a sub-path denoted by P_{S+1}, which in turn consist of a red vertex followed

by S blue vertices. The blueprint G contains a complete bipartite graph $K_{6m,m}$ with m red vertices on one side and $6m$ blue vertices on the other side. These blue vertices are further connected in all possible ways, forming a clique Q of size $6m$ (the gray box in the figure). Additionally, for each a_i, we construct a clique of $a_i - 2$ blue vertices (in the 3-Partition problem, we can safely assume that $a_i > 2$), and we connect two of its vertices with two vertices of Q.

The construction can be done in polynomial time. It is easy to see that the graph G is dense (i.e., asymptotically, the number of edges is a quadratic function of the number of vertices), it has the same size as P, and there is an embedding of P into G if and only if the a_i's can be partitioned into multisets of sum S.

We show that the 3-Partition problem has a solution if and only if P can be embedded in G with color matching. First, we must embed red vertices of P in the red vertices of the complete bipartite graph $K_{6m,m}$ in G. Then S blue vertices of each subpath P_{S+1} should be embedded in S blue vertices of G, hence each subpath P_{S+1} should be embedded in S blue vertices in the set of cycles C_{a_i}. That is, each subpath P_{S+1} is embedded in exactly three cycles C_{a_i}, C_{a_j} and C_{a_k} for some i, j, k with $a_i + a_j + a_k = S$. Clearly, each embedding of a subpath P_{S+1} gives a subset of A, and the collection of these subsets gives us a solution to the 3-Partition problem and vice versa. □

4.2 Bicolored Synthesis Problem

For a bicolored path P and a bicolored graph G, let $m_{P,G}$ be the maximum number of red-red contacts across all embeddings of P into G. In the previous section, we showed that computing $m_{P,G}$ is NP-hard, even assuming that the solution space is non-empty. Now, in the spirit of protein synthesis, we formulate the following problem:

Definition 16. *Given a bicolored graph G, the* bicolored synthesis problem *asks for the bicolored path P of the same order as G that maximizes $m_{P,G}$.*

Translated into the language of protein folding, we are given the "form" of a protein (i.e., a bicolored graph G), and we ask for the amino acid chain that is most likely to fold into a protein of that particular form.

We will show that this problem is Poly-APX-hard, i.e., the optimum is NP-hard to approximate within a sub-polynomial ratio.

Theorem 6. *The bicolored synthesis problem is Poly-APX-hard.*

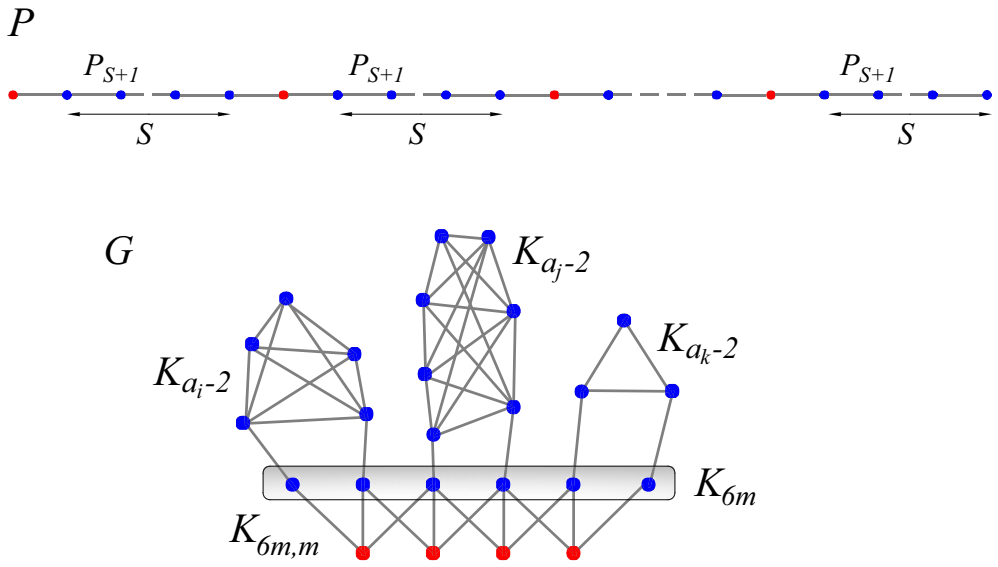

Figure 4.1: Sketch of the NP-hardness reduction from the 3-Partition problem. For clarity, the edges of the clique K_{6m}, as well as some edges of the complete bipartite graph $K_{6m,m}$, have been omitted.

Proof. We will give an approximation-preserving reduction from the *Independent Set* problem, which is Poly-APX-complete [5]. For the reduction, we borrow the *edge gadget* from [18], which is illustrated in Figure 4.2, top. The top six vertices (next to the letter u) constitute the "top half" of the gadget; the other six are the "bottom half". If this gadget is part of a larger graph, there are only three ways a Hamiltonian path can traverse it, as shown in the figure.

Now, given a connected graph $G' = (V', E')$, where $V' = \{v_1, \ldots, v_n\}$, we will construct a bicolored graph G that implements our approximation-preserving reduction as follows (an example is shown in Figure 4.3). First we construct an edge gadget for each edge in E'. Then we connect edge gadgets together in such a way that, for each vertex $v_i \in V'$, there is a path, called "strand," that traverses (either the top or the bottom half of) each of the edge gadgets corresponding to edges incident to v_i in G'. For example, in Figure 4.3, the three gadgets labeled $\{v_2, v_3\}$, $\{v_1, v_2\}$, and $\{v_2, v_4\}$ are connected together in sequence, forming a strand whose endpoints are labeled v_2 in the figure. This represents the fact that the vertex $v_2 \in V'$ is adjacent to v_1, v_3, and v_4 in G'. It follows that each edge gadget is shared by precisely two strands.

Next, we construct a *selector gadget*, shown at the top of Figure 4.3, which simply consists of a path of $n + 3$ vertices, the endpoints of which are called

s and t. We connect each vertex of the selector gadget, except s and t, to both endpoints of all the previously constructed strands; these connections are represented by dashed edges in the figure.

Finally, we color all vertices of G blue except for $2n$ of them, as follows. For each vertex $v_i \in V'$, we choose one adjacent vertex v_j, we identify the edge gadget corresponding to $\{v_i, v_j\}$, and we color in red the two central vertices in the gadget that belong to the strand of v_j, as shown in Figure 4.3.

This completes the construction. We will now prove that G' has an independent set $S \subseteq V'$ of k vertices if and only if there is a bicolored path P of the same order as G that can be bijectively embedded in G forming exactly k red-red contacts. This will imply that our reduction is approximation-preserving. In the example in Figure 4.3, the independent set is $S = \{v_1, v_4\}$.

Note that the existence of an embedded path P that forms k red-red contacts is equivalent to the existence of a Hamiltonian path in G that avoids exactly k edges whose endpoints are both red. Now, any Hamiltonian path in G must have endpoints in s and t, and use the selector gadget to access some of the strands. Once a strand has been chosen, it must be followed until the end; after that, the path goes to the next vertex of the selector gadget, and then into another strand. Along the strand of vertex v_i, each encountered edge gadget $\{v_i, v_j\}$ has to be covered in one of two possible ways, depending on whether the strand of v_j will be traversed or not. If the strand of v_j is not going to be traversed, the path must cover all vertices in the edge gadget $\{v_i, v_j\}$; for example, see the edge gadget $\{v_1, v_2\}$ in the figure, where the strand of v_2 is traversed and the strand of v_1 is not. In this case, if the edge gadget contains two red vertices corresponding to v_j, these vertices will form a red-red contact.

Thus, if a Hamiltonian path forms two red-red contacts corresponding to

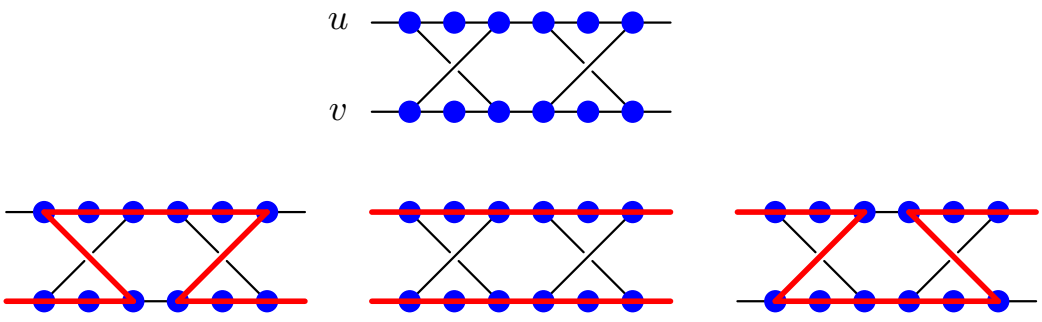

Figure 4.2: Illustration of the *edge gadget* used in Theorem 6 (see [18]). The bottom part of the figure shows the three possible ways a Hamiltonian path can traverse this gadget.

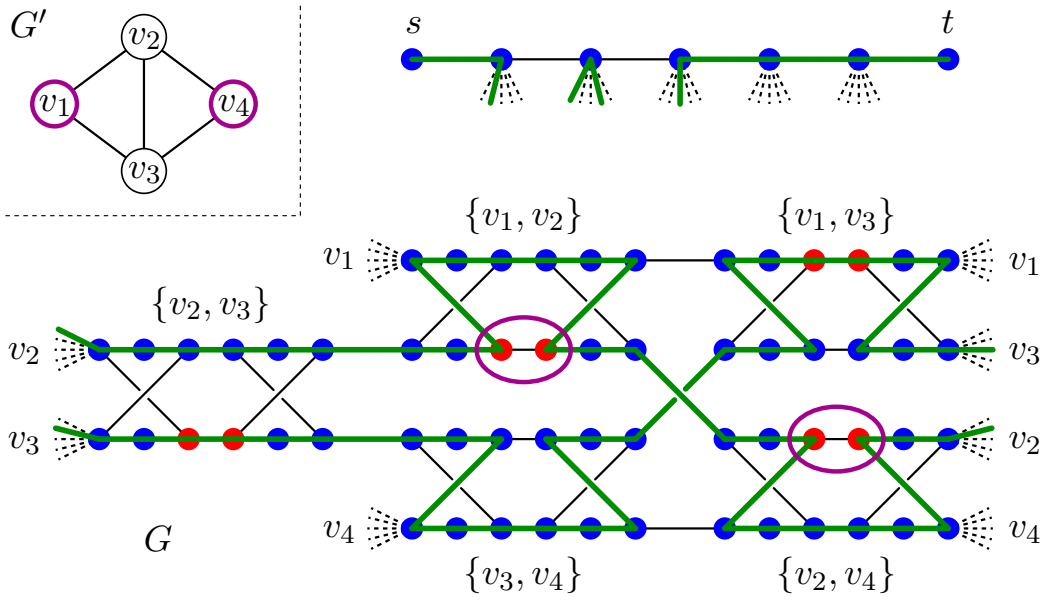

Figure 4.3: Example of the approximation-preserving reduction used in Theorem 6. As the dashed edges suggest, each vertex in G labeled $v_1, \ldots, v_4$ is adjacent to all vertices in the top *selector gadget*, except s and t. The vertices of G' circled in purple constitute an independent set that corresponds to a path embedded in G (drawn in green) with as many red-red contacts.

vertices v_a and v_b, it means that it does not traverse the strands of v_a and v_b. Hence v_a and v_b are not adjacent in G', otherwise there would be an edge gadget $\{v_a, v_b\}$ in G that is not covered by the path. So, the set of red-red contacts determined by a Hamiltonian path corresponds to an independent set of G'. Conversely, if S is an independent set, traversing the strands of the vertices in $V' \setminus S$ yields a Hamiltonian path with $|S|$ red-red contacts. □

4.3 Weighted Eulerian Path Problem

Now we show the main theorem in this section.

Theorem 7. *Let P, G, ℓ be a path, an undirected graph, and a length function, respectively. Then the weighted Eulerian path problem is strongly NP-hard even if $\ell(e)$ is either 1 or 2 for any e in P and G.*

Proof. It is easy to see that the problem is in NP. Therefore we show the hardness. We reduce the 3-Partition problem to the weighted Eulerian path problem.

Let P_{B+1} be a path that consists of B consecutive edges of length 2, and P_4 be a path that consists of 3 consecutive edges of length 1. Then the path P is obtained by joining m subpaths P_{B+1} and m subpaths P_4 alternatingly, that is, P is constructed by joining $P_{B+1}, P_4, P_{B+1}, P_4, \ldots, P_4, P_{B+1}$, and P_4. The graph G is constructed as follows. For each i with $1 \leq i \leq 3m$, we construct a cycle C_{a_i} of a_i edges of length 2. We also construct m cycles C_3 of 3 edges of length 1. Then these $4m$ cycles share a special vertex c in common. That is, G is a cactus that consists of $4m$ cycles, and all vertices have degree 2 except the common vertex c that has degree $8m$. The construction is illustrated in Figure 4.4.

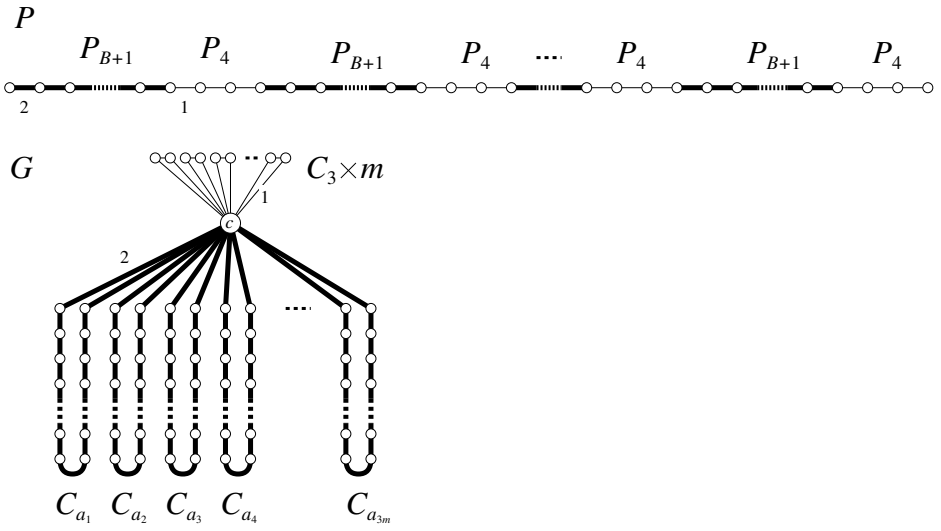

Figure 4.4: Construction of P and G; bold lines are of length 2, and thin lines are of length 1. Each P_{i+1} consists of i edges and each C_i consists of i edges.

It is easy to see that this is a polynomial-time reduction. Thus, we show that A has a solution if and only if P can simulate G.

We first observe that no edge of length 2 in P_{B+1} in P can cover a cycle C_3 in G. Therefore, when P covers G, every C_3 of G has to be covered by P_4 in P. Thus, each endpoint of P_4 should be on c in G, and no edge in P_{B+1} can cover edges in C_3. Hence, each subpath P_{B+1} in P covers exactly B edges in the set of cycles C_{a_i} that consists of edges of length 2. Since $B/4 < a_i < B/2$ for each i, each subpath P_{B+1} covers exactly three cycles C_{a_i}, C_{a_j} and C_{a_k} for some i, j, k with $a_i + a_j + a_k = B$. Clearly, each cover for a subpath P_{B+1} gives a subset of A, and the collection of these subsets gives us a solution to the 3-Partition problem and vice versa. □

4.4 Elastic Linkage Problem

In this section, we consider the elastic linkage problem for two paths G and P:

Elastic linkage problem from path to path

Input: Two paths $G = (V', E')$ and $P = (V, E)$ with length function ℓ.

Output: a mapping ϕ with minimum elastic (or stretch/shrink) ratio.

In this problem, we allow all edges in P to be elastic to simulate the path G by the path P.

We let G is a path $(u_1, u_2, \ldots, u_{n'})$ and P is a path $(v_1, v_2, \ldots, v_n)$, see Figure 4.5.

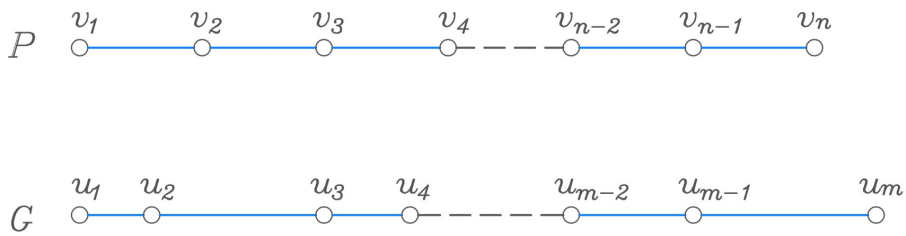

Figure 4.5: A path P and a path G, path P is an elastic linkage, edges in P are allowed to be elastic to fit the vertices of P to ones of G.

Without loss of generality, we assume that $n' \leq n$. Since each vertex in G should be mapped from only one vertex in P, it should be $\phi(v_1) = u_1$ and $\phi(v_n) = u_{n'}$, otherwise the elastic ratio will be infinity.

We show a polynomial-time algorithm for this problem based on dynamic programming.

First, we show a technical lemma when G is just an edge. In this case, the optimal value is achieved when all ratios are even.

Lemma 2. *Assume that G consists of an edge $e' = (u_1, u_2)$. When $P = (V, E)$ is a path, the minimum elastic ratio is achieved when the ratio of each $e \in E$ takes the same value.*

Proof. Assume that the length of the edge $e' = (u_1, u_2)$ in G is L, $E = \{e_1, e_2, \ldots, e_{n-1}\}$, and the length of each e_i is l_i. For a mapping ϕ, let r_i be

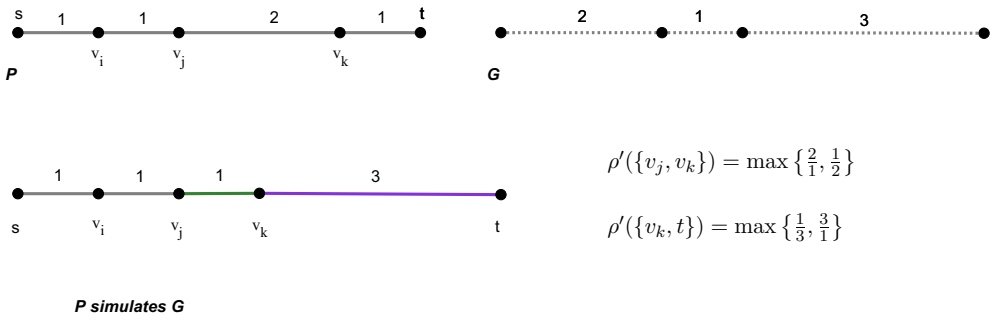

Figure 4.6: An elastic linkage P simulates a path G. The elastic ratio of an edge $\{v_j, v_k\}$ in P is obtained by comparing the ratio of its original length to the stretched length and the ratio's reciprocal.

the ratio of the edge e_i for each $i = 1, 2, \ldots, n-1$. Then we have $r_1 l_1 + r_2 l_2 + \cdots + r_{n-1} l_{n-1} = L$.

Assume the maximum among r_i for all $1 \leq i \leq n-1$ is r_k, and the minimum among r_i for all $1 \leq i \leq n-1$ is r_h. Thus, it is obvious that $r_k \geq L/(l_1 + l_2 + \cdots + l_{n-1})$, $1/r_h \geq (l_1 + l_2 + \cdots + l_{n-1})/L$.

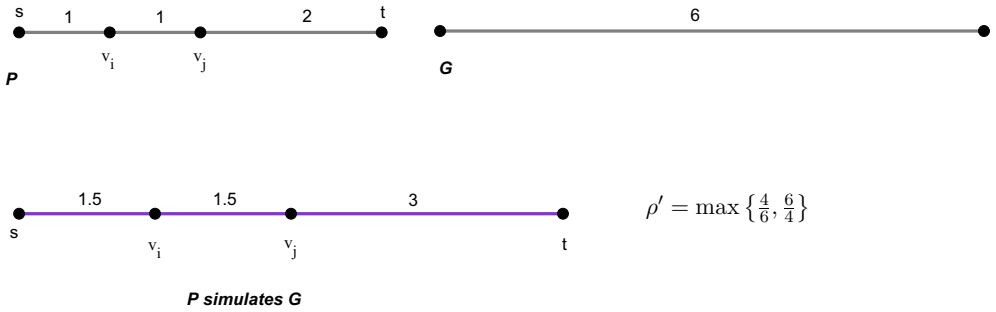

Figure 4.7: An elastic linkage P simulates an edge G, the minimum elastic ratio is achieved when all edges in P are stretched by the same factor.

According to the definition, the elastic ratio er of this mapping is the maximum among r_i and its reciprocal for all $1 \leq i \leq n-1$ (see Figure 4.6). That is, er equals the larger of r_k and $1/r_h$.

When $r_1 = r_2 = \cdots = r_{n-1}$, $\max\{r_k, 1/r_h\}$ takes the minimum. That means the minimum elastic ratio can be achieved if and only if the ratio of each $e \in E$ takes the same value. Figure 4.7 gives us an example. An elastic linkage P simulates an edge G, the minimum elastic ratio is achieved when all edges in P are stretched by the same factor. □

Now we turn to the main theorem.

Theorem 8. *We can solve the elastic linkage problem from path to path in* $O(n^3)$ *time.*

Proof. We assume path $P = (v_1, v_2, \ldots, v_n)$, the length of each edge $\{v_i, v_{i+1}\}$ is l_i, path $G = (u_1, u_2, \ldots, u_{n'})$, the length of each edge $e'_j = \{u_j, u_{j+1}\}$ is w_j, and $n \geq n' \geq 2$.

We define two functions as follows for $i > i' \geq j$:

$$dist(v_{i'}, v_i) = l_{i'} + l_{i'+1} + \cdots + l_{i-1}, \text{ and}$$
$$Ser(v_{i'}, v_i, e'_j) = \max\left\{\frac{w_j}{dist(v_{i'}, v_i)}, \frac{dist(v_{i'}, v_i)}{w_j}\right\}.$$

That is, $dist(v_{i'}, v_i)$ is the length of the path $(v'_i, \ldots, v_i)$, and $Ser(v_{i'}, v_i, e'_j)$ is the minimum elastic ratio of all edges in the subpath $P' = (v_{i'}, v_{i'+1}, \ldots, v_i)$ of P that covers the edge $\{u_j, u_{j+1}\}$. We first precompute these functions as tables which will be referred in our polynomial-time algorithm. The computation of the corresponding table $Ser[(v_{i'}, v_i), e'_j]$ can be done as follows[1]: (1) for each $(v_{i'}, v_i)$ with $i' < i$, compute $dist(v_{i'}, v_i)$ and fill in the table $dist[v_{i'}, v_i]$, (2) for each $j = 0, 1, \ldots, n'$, compute $Ser(v_{i'}, v_i, e'_j)$ and fill in the table $Ser[(v_{i'}, v_i), e'_j]$. In (1), each $dist(v_{i'}, v_i)$ can be computed in a constant time by using $dist(v_{i'}, v_i) = dist(v_{i'}, v_{i-1}) + \ell(e_{i-1})$ when we compute the values of this table in the order of $(i - i') = 1, 2, 3, \ldots$. On the other hand, in (2), each $Ser(v_{i'}, v_i, e'_j)$ can be computed in a constant time. Therefore, the precomputation can be done in $O(n^3)$ time in total.

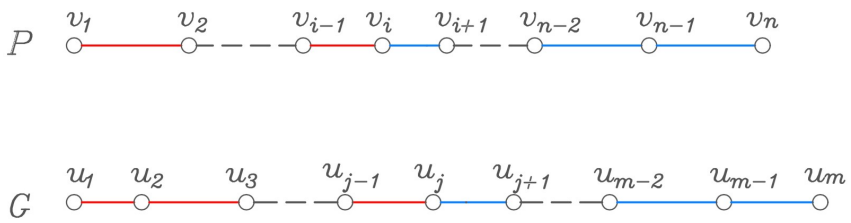

Figure 4.8: We consider to use the subpath $P' = (v_1, v_2, ..., v_i)$ of path P to simulate the subpath $G' = (u_1, u_2, ..., u_j)$ of path G, and we have $\phi(v_1) = u_1$, $\phi(v_i) = u_j$.

To solve the elastic linkage problem efficiently, we define two more functions $ER(v_i, u_j)$ and $M(v_i, u_j)$ as follows. First, $ER(v_i, u_j)$ is the minimum

[1]In this thesis, for a function $f()$ and a predicate $p()$, their corresponding tables (or arrays in program) are denoted by $f[]$ and $p[]$, respectively.

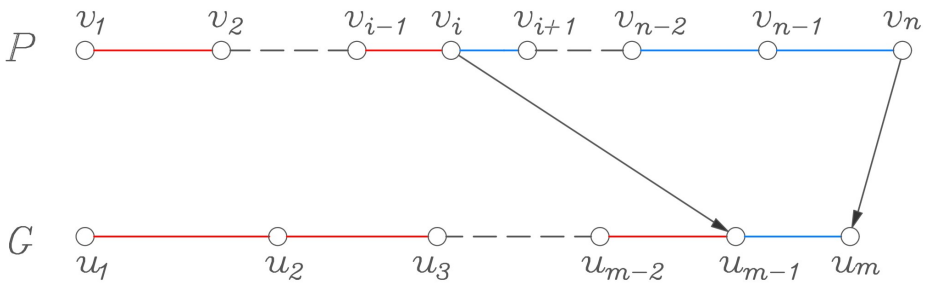

Figure 4.9: We consider to use the subpath $P' = (v_1, v_2, ..., v_i)$ of path P to simulate the subpath $G' = (u_1, u_2, ..., u_{m-1})$ of path G, and we have $\phi(v_1) = u_1$, $\phi(v_i) = u_{m-1}$.

elastic ratio of the mappings from the subpath $P' = (v_1, v_2, \ldots, v_i)$ of P to the subpath $G' = (u_1, u_2, \ldots, u_j)$ of G, see Figure 4.8 and Figure 4.9. Then we have the following:
$ER(v_i, u_j) =$

$$\begin{cases} Ser(v_1, v_i, e'_1) & j = 2 \\ \min\{\max\{ER(v_k, u_{j-1}), Ser(v_k, v_i, e'_{j-1})\} k = j-1, j, \ldots, i-1\} & j > 2 \end{cases}$$

Our goal is to obtain the mapping from P to G with elastic ratio $ER(v_n, u_{n'})$.

Next, $M(v_i, u_j)$ is a sequence of j vertices of path P that represents the mapping with minimum elastic ratio from the subpath P' to the subpath G'. The first and last vertices in $M(v_i, u_j)$ are v_1 and v_i. Then we have the following:

$$M(v_i, u_j) = \begin{cases} (v_1, v_i) & \text{when } j = 2 \\ (M(v_\tau, u_{j-1}), v_i) & \text{when } j > 2, \end{cases}$$

where τ is determined by the following equation;

$$ER(v_i, u_j) = \max\{ER(v_\tau, u_{j-1}), Ser(v_\tau, v_i, e'_{j-1})\}.$$

Then our goal is to obtain $M(v_n, u_{n'})$. The $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ can be obtained simultaneously by the dynamic programming technique. In the tables of $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$, $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ are easy to get if the values in the $(n'-2)$-nd row are available. The table $ER(v_n, u_{n'})$ is filled from $j = 2$, that is, for each $v_i = v_1, v_2, \ldots, v_n$, $ER(v_i, u_1) = Ser(v_1, v_i, e'_1)$ has already been computed, and accordingly, the first row of table $M(v_n, u_{n'})$ is $M(v_i, u_1) = (v_1, v_i)$. After filling in the first row of the tables, it is easy to get the values in the second row, the third row, up to the $(n'-2)$-nd row and finally get $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$. Each element of the table $ER(v_n, u_{n'})$

can be computed in $O(n)$ time, and each element of the table $M(v_n, u_{n'})$ can be computed in constant time. Therefore, the computation of $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ can be done in $O(n^3)$ time, and the precomputation also can be done in $O(n^3)$ time. Thus, the algorithm runs in $O(n^3)$ time, which means the elastic linkage problem can be solved in polynomial time. □

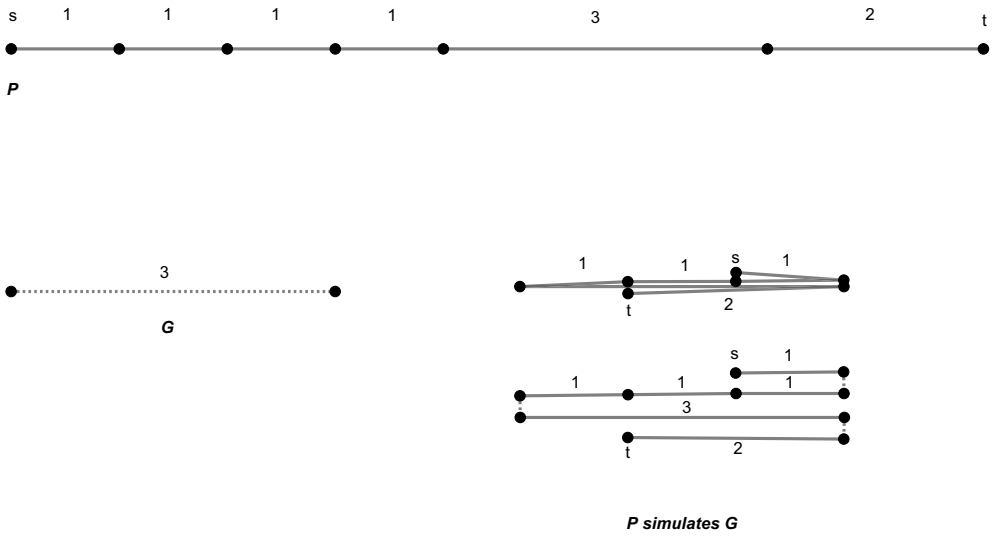

Figure 4.10: General cover problem. P is a linkage and G is an edge with length $L = 3$.

4.5 Traverse Problem of a Tree by a Path

In this section, we focus on the traverse problem of G by P. In this variant, we allow P to cover an edge of G twice.

4.5.1 General Cover Problem

Before the traverse problem, we consider a more general case that allows P to cover an edge of G twice or more. This general simulation problem is similar to the following ruler folding problem:

Ruler Folding: Given an integer L and a polygonal chain with links of integer length $\ell_1, \ldots, \ell_{n-1}$, can the chain be folded flat so that its total folded length is L?

The details of this problem and related results can be found in [18]. In our context, we have the following theorem:

Theorem 9. *The general simulation problem of G by P is NP-complete even if G is an edge.*

Proof. We can reduce the ruler folding problem to our problem by just letting G be an edge of length L (see Figure 4.10). □

We note that the ruler folding problem is weakly NP-complete, and we have a simple pseudo-polynomial-time algorithm that runs in $O(nL)$ time as follows;

Input: Set of integers $S = \{\ell_1, \ldots, \ell_{n-1}\}$ and an integer L
Output: Determine if there is $I \subseteq \{1, \ldots, n-1\}$ with $\sum_{i \in I} \ell_i == L$
begin
  Initialize array $a[0], \ldots, a[L]$ by 0;
  Set $a[0] = 1$;
  foreach $i = 1, \ldots, n-1$ **do**
    foreach $j = 0, \ldots, L$ **do**
      if $a[j] == 1$ *and* $j + \ell_i \leq L$ **then** $a[j + \ell_i] = 1$;
    end
  end
  if $a[L] == 1$ **then** output "Yes";
  else output "No";
end

4.5.2 Tree Traversal Problem

Now we turn to the traverse problem. Even if a connected graph G has no Eulerian path, when we allow P to visit each edge in G twice, we can visit all vertices of G by a path in the depth-first search manner.

Therefore, we consider the following *traversal problem* as a kind of the linkage simulation problem:

Input: A path $P = (V, E)$ that forms a path $(v_1, v_2, \ldots, v_n)$, and a graph $G = (V', E')$ with length function $\ell : E \cup E' \to \mathbb{R}$.

Output: A mapping ϕ from P to G such that each edge in G is mapped from exactly two subpaths of P, or "No" if it does not exist.

We first observe that it is linear-time solvable when each edge has the unit length just by depth-first search. Therefore, it is an interesting question that asks the computational complexity when ℓ maps to few distinct values, especially, ℓ maps to two distinct values.

NP-Completeness Results

We give three hardness results about the traversal problem even if the graph G is a simple tree T and the edge lengths are quite restricted.

Theorem 10. *The traversal problem of a tree T by a path P is strongly NP-complete in each of the following cases: (1) T is a star $K_{1,n-1}$, and P consists of edges of two different lengths. (2) T is a spider, and all edges in G and P are of length p and q, where (2a) p and q are any two positive integers that are relatively prime, or (2b) $p = 1$ and $q = 2$.*

Proof. Since it is clear that each of the problems is in NP, we show their hardness. We will give polynomial-time reductions from the 3-Partition to our problems.
(1) T is a star $K_{1,4m+1}$. Among $4m+1$ edges, the length of $m+1$ edges is B, and the other $3m$ edges have length a_i for each $i = 1, 2, \ldots, 3m$ (Figure 4.11). The construction of P is as follows. Let P' be a path that consists of $2B$ edges of length 1, and P'' be a path that consists of 2 edges of length B. Then the path P is obtained by joining $m+1$ subpaths P'' and m subpaths P' alternatingly, that is, P is constructed by joining $P'', P', P'', P', P'', \ldots, P', P''$ as shown in Figure 4.11.

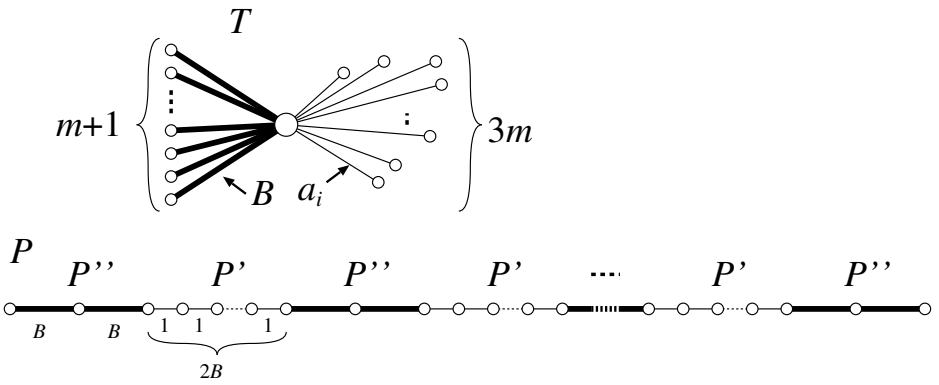

Figure 4.11: Reduction to $K_{1,4m+1}$ and a path P.

The construction is done in polynomial time. Thus, we show that the 3-Partition problem has a solution if and only if the constructed cover problem

has a solution. We first observe that P'' cannot cover any short edge of length a_i in T. Therefore, each P'' should cover each edge of length B in T twice. Hence all of the endpoints of P''s (and hence P') are on the central vertex of T. Therefore, if P can cover T properly, it is easy to see that each P' should cover three edges of length a_i, a_j, and a_k with $a_i + a_j + a_k = B$ exactly twice. This concludes the proof of (1).
(2a) This reduction is similar to (1). Let P' be a path that consists of $2B$ edges of length p, and P'' be a path that consists of 2 edges of length q. Then the path P is obtained by joining $m+1$ subpaths P'' and m subpaths P' alternatingly. On the other hand, the spider T is obtained by sharing the central vertex of $4m+1$ subpaths (Figure 4.12). Among $4m+1$ subpaths, $m+1$ paths are just edges of length q. The other $3m$ subpaths are of a_i edges for each $1 \le i \le 3m$, and each edge has length p. Since p and q are relatively prime, P'' cannot cover each of the edges of length p. Therefore, their endpoints (and the endpoints of P') share the central vertex of T. Thus, each P' gives us the solution of the 3-Partition as in (1), which completes the proof of (2a).
(2b) The reduction itself is the same as (2a) except $p = 1$ and $q = 2$. In this case, we observe that no edge of length 1 can be covered by any edge of length 2 in P''. Therefore, each edge of P'' of length 2 should cover the edges of T of length 2. Thus, each P' gives us the solution of the 3-Partition as in (2a), which completes the proof of (2b). □

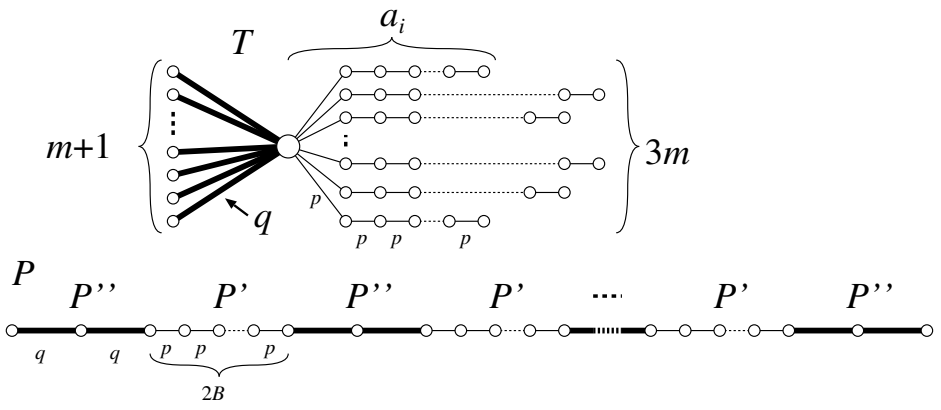

Figure 4.12: Reduction to spider of two different lengths.

Polynomial Time Solvable Case

In Theorem 10, we show that the traversal problem of a tree by a path is NP-hard even if we strictly restrict ourselves. Now we turn to show a polynomial-time algorithm for the case that we furthermore restrict. The Figure 4.13 shows the constructions of a path P and a tree T in this case, T is a star and its edge lengths are of k different values.

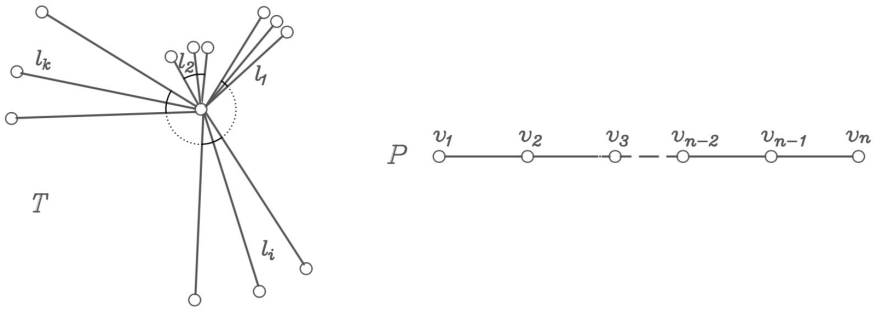

Figure 4.13: Constructions of a star T with k different edge lengths $\ell_1, \ell_2, \ldots, \ell_k$ and a path P of length n.

Theorem 11. *Let T be a star $K_{1,n'}$ and the number of distinct lengths of its edges is k. Let P be any path of length n. Without loss of generality, we suppose $2n' \leq n$. Then the traversal problem of T by P can be solved in $O(n^{k+1})$ time and $O(n^k)$ space. That is, it is polynomial-time solvable when k is a constant.*

Proof. We suppose that each edge of T has a length in $L = \{\ell_1, \ell_2, \ldots, \ell_k\}$, and T contains L_i edges of length ℓ_i for each i. For a vertex v_i in P and length ℓ_j in L, we define a function $\mathrm{pre}(v_i, \ell_j)$ as follows;

$$\mathrm{pre}(v_i, \ell_j) = \begin{cases} v_k \text{ there is a vertex } v_k \text{ with } k < i \text{ on } P \\ \quad \ell(e_k) + \ell(e_{k+1}) + \cdots + \ell(e_i) = \ell_j. \\ \phi \text{ otherwise} \end{cases}$$

We first precompute this function as a table which will be referred to in our polynomial-time algorithm. To distinguish the function $\mathrm{pre}(v_i, \ell_j)$, we refer to this table as $\mathrm{pre}[v_i, \ell_j]$ which uses $O(nk)$ space. The computation of pre[] can be done as follows; (0) initialize pre[] by ϕ in $O(nk)$ time, (1) sort L in $O(k \log k)$ time, and (2) for each $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, k-1$, the

vertex v_i fills the table $\mathrm{pre}[v_{i'}, \ell_j] = v_i$. In (2), the vertex v_i can fill $\mathrm{pre}[v_{i'}, \ell_j] = v_i$ in $O(n+k)$ time. Therefore, the precomputation takes $O(n(n+k)+k\log k)$ time in $O(nk)$ space.

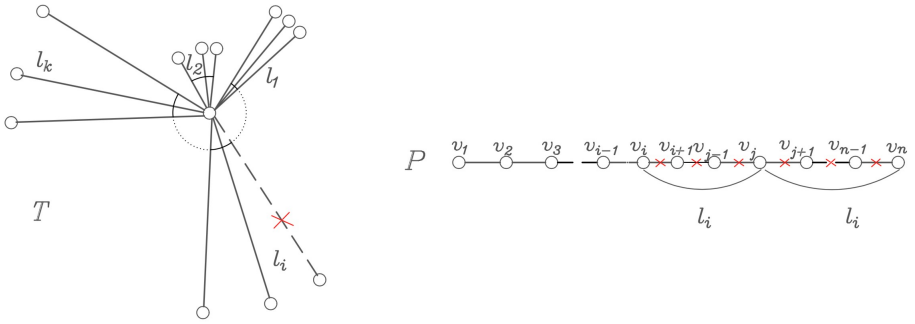

Figure 4.14: For each edge in T, find a subpath of P which covers this edge exactly twice.

Now we turn to the computation for the traversal problem. To do that, we define a predicate $F(d_1, d_2, \ldots, d_k, v_i)$ which is defined as follows: When there is a cover of a subtree T' of T that consists of d_1 edges of length ℓ_1, d_2 edges of length ℓ_2, ..., and d_k edges of length ℓ_k by the subpath $P' = (v_1, v_2, \ldots, v_i)$ when v_1 and v_i are put on the center of T, $F(d_1, d_2, \ldots, d_k, v_i)$ is *true*, and *false* otherwise. (For notational convenience, we define that $F(d_1, d_2, \ldots, d_k, \phi)$ is always false.) Thus, our goal is to determine if $F(L_1, L_2, \ldots, L_k, v_n)$ is true or false. The predicate $F(d_1, d_2, \ldots, d_k, v_i)$ is determined by the following recursion;
$F(d_1, d_2, \ldots, d_k, v_i) =$

$$\bigvee_{1 \le j \le k} ((\mathrm{pre}(v_i, \ell_j) \neq \phi) \wedge F \begin{array}{l}(d_1, \ldots, d_j - 2, \ldots, d_k, \\ \mathrm{pre}(\mathrm{pre}(v_i, \ell_j), \ell_j)))\end{array} .$$

That is, for the vertex v_i, we have to have two vertices $v_{i'} = \mathrm{pre}(v_i, \ell_j)$ and $v_{i''} = \mathrm{pre}(\mathrm{pre}(v_i, \ell_j), \ell_j)$ such that $\ell(e_{i'}) + \ell(e_{i'+1}) + \cdots + \ell(e_i) = \ell_j$ and $\ell(e_{i''}) + \ell(e_{i''+1}) + \cdots + \ell(e_{i'}) = \ell_j$ for some j with $1 \le j \le k$. The correctness of this recursion is trivial. The basic idea is shown in Figure 4.14.

The predicate $F(L_1, L_2, \ldots, L_k, v_n)$ is computed by a dynamic programming technique. That is, the table $F[d_1, d_2, \ldots, d_k, v_i]$, corresponding to the predicate $F(L_1, L_2, \ldots, L_k, v_n)$, is filled from $d_1 = 0, d_2 = 0, \ldots, d_k = 0$ for the center vertex c, which is true. Then, we increment in the bottom up manner; that is, we increment as $(d_1, d_2, \ldots, d_k) = (0, 0, \ldots, 0, 1)$, $(0, 0, \ldots, 1, 0)$, ...,

$(0, 1, \ldots, 0, 0)$, $(1, 0, \ldots, 0, 0)$, $(0, 0, \ldots, 0, 2)$, $(0, 0, \ldots, 1, 1)$, …, $(0, 1, \ldots, 0, 1)$, $(1, 0, \ldots, 0, 1)$, and so on. The number of combinations of $(d_1, d_2, \ldots, d_k)$ is $L_1 \cdot L_2 \cdot \cdots \cdot L_k \leq n'^k = O(n^k)$, and the computation of $F[d_1, d_2, \ldots, d_k, v_i]$ for the $(d_1, d_2, \ldots, d_k)$ can be done in linear time. Therefore, the algorithm runs in $O(n^{k+1})$ time and $O(n^k)$ space. □

Chapter 5

Concluding Remarks

This research starts from the observation that standard techniques for the HP model can be criticized for embedding proteins in a square 2D lattice and relying on the properties of parity in the lattice. In fact, such parity-related observations have no meaning in the real protein folding problem that the theory aims to model. Also, the standard measure being used in the HP model for how well an amino acid chain folds into a protein (namely, the number of amino acids of a certain type that are close together in the embedding) is not the only possible way to capture the intricate physical and chemical laws that describe how a real protein folds.

These observations led us to some new variants of the protein folding model, which we defined and explored in this thesis. We combined the basic ideas of protein folding with the complementary problem of protein design, thus formulating several protein folding problems in terms of embedding of paths into graphs.

Inspired by protein folding in the HP model, we proposed the bicolored path embedding problem and linkage simulation problem. We studied these problems in both grid graphs and general graphs. We showed that these problems are NP-hard in several settings, and polynomial-time solvable under some constraints.

5.1 Contribution and Conclusion

Most of our results indicate that several protein folding problems are computationally intractable in our models. In general, knowing that a problem is computationally hard (e.g., NP-hard) should discourage us from seeking an efficient solution. However, in the context of protein folding, a proof of NP-hardness can also provide insights on the deeper reasons why nature works in

a certain way. For example, the fact that protein folding is computationally hard in a given model might be evidence that the model is incorrect and should be modified. Nonetheless, it could also mean that the model is accurate, but the instances of the problem that have been proved to be hard never occur in practice. In this case, a hardness result sheds some light on which patterns and configurations are naturally avoided in biological systems, and why.

As a result of this research, we now have some alternative models for protein folding and design, and some efficient algorithms for solving typical problems in these models. We also have several results that indicate when such problems are computationally intractable, which is a contribution to our theoretical understanding of protein folding and graph embedding problems in general.

Indeed, since this thesis studies fairly general and abstract graph-theoretic problems, its applications are not limited to protein folding and design, but extend to any problem where a foldable chain has to be packed into a space of a certain shape while satisfying some constraints. A notable application is in robotics: some of the folding models we introduced pertain to linkages, which serve as a model for robot arms and mechanisms.

Furthermore, this research did some ground work to enable new and exciting lines of investigation including, for example, smoothed analysis. The models we have created and the intractability results we obtained offer the ideal setting for studying protein folding in the presence of small amounts of random noise in the input data. This new direction of research, based on smoothed analysis, would offer a theoretically valid explanation as to why nature is able to efficiently solve protein folding problems, while the same problems are classified as intractable by traditional worst-case complexity theory.

5.2 Future Work

In this section, we will discuss some open questions related to this research and possible future research directions.

Our hardness results indicate that certain general problems related to protein folding are probably intractable by computers. Interpreting some of these results in practical terms is ultimately a philosophical matter: since nature does indeed seem to solve NP-hard protein folding problems in an efficient way, this may indicate that our ways of modeling computation (e.g., Turing machines) do not correctly reflect the way nature works. It may also indicate that discrete models of protein folding such as the HP model do

not faithfully capture the essence of real protein folding, or that our NP-hardness reductions produce instances of amino acid chains and proteins that are unlikely to be found in real biological systems. To address the latter issue, we suggest studying the same problems under the *smoothed analysis* paradigm: adding small amounts of random noise to an amino acid chain may be sufficient to eliminate the special patterns that cause protein folding to be computationally intractable.

5.2.1 Open Problems

Another way of interpreting our hardness results is as an "upper bound" on what can be done efficiently by computers. For example, Theorem 6 states that guessing an amino acid chain that is likely to fold into a protein of a given arbitrary shape is a hopelessly hard problem. This knowledge should discourage us from attempting to find efficient algorithms for the general problem, and direct us toward special cases or relaxations of the problem. For instance, it would be interesting to know if Theorem 6 remains true when G is a grid graph: we leave this as an open question.

Some other open questions are whether the running time of the dynamic-programming algorithm in Section 3.2.2 and Section 4.4 can be improved.

Also, it would be interesting to find other natural classes of blueprints for which the bicolored path embedding problem is polynomial-time solvable. Moreover, we would like to study more variants of the weighted Eulerian path problem. For example, the advanced elastic linkage problem, which combines the ideas of the elastic linkage problem and the tree traversal problem. In this problem, we do not only want P to cover an edge of G twice or more, but we also let the edges in P be elastic.

5.2.2 New Techniques

Aside from the above results, there are also some promising directions and techniques that we are currently exploring. These methods include ***Smoothed Analysis*** and ***Linear Programming***.

Smoothed Analysis

In theoretical Computer Science, Smoothed Analysis is a method to measure the complexity of an algorithm. It is a hybrid between worst-case analysis and average analysis, taking the best aspects of both approaches.

Smoothed Analysis is based on the assumption that the input to real problems is subject to small random perturbations. The premise is that

what makes some problems (NP-)hard is the coexistence of several factors that are statistically unlikely. For example, there are algorithms (such as the simplex algorithm for linear programming) that run in exponential time in the worst case, but become very efficient as soon as small perturbations are introduced in the input data.

A similar phenomenon may very well be occurring with protein folding. Indeed, random perturbations are extremely common in chemical reactions, and amino acid chains spontaneously fold into proteins by undergoing local deformations that may be described by simple algorithms. Essentially, nature can solve the protein folding problem efficiently in spite of its NP-hardness, and the explanation may be that the input data always has some random noise. Smoothed Analysis provides the theoretical tools to validate this assumption and make it rigorous.

Thus, our approach is to find greedy and local protein-folding algorithms, inspired by the laws of Chemistry and Biology, which necessarily converge very slowly on some pathological inputs (since the problem is NP-hard in general), but become very efficient as soon as small perturbations are introduced in the input data. Our goal is to use Smoothed Analysis to prove that such algorithms achieve good approximation ratios with high probability.

Linear Programming

In previous works, the problem of optimal protein folding in the HP model (maximizing some objective function given by local constraints) has been successfully modeled as a Mixed Integer Programming problem (where some of the variables are real numbers, and some of them are integers). Typically, in this line of research, a Linear Optimization solver is used to find an optimal solution. This step takes exponential time, and the technique is only applicable to very small instances; also, previous works are mostly experimental.

Our approach is to study the continuous relaxation of a suitably modeled problem in the HP model. In practice, the protein folding problem is modeled as a Linear Programming problem where all variables are real, and the optimal solution is found efficiently by standard methods. This solution may be fractional; this means that, instead of giving the exact position of every amino acid in the folded state, it only gives a probability distribution describing where it is likely to be. The problem we are studying now is how to sample from this probability distribution to obtain a valid embedding for the amino acid chain which achieves a good approximation of the optimum.

Bibliography

[1] Kupferman, Orna, and Gal Vardi. “Eulerian paths with regular constraints”, 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016). Leibniz International Proceedings in Informatics (LIPIcs), Vol. 58, pp. 62:1–62:15, 2016.

[2] Michael R. Garey and David S. Johnson. Computers and Intractability. Vol. 174. San Francisco: freeman, 1979.

[3] E. M. Arkin, S. P. Fekete, K. Islam, H. Meijer, J. S. B. Mitchell, Y. Núñez-Rodríguez, V. Polishchuk, D. Rappaport, and H. Xiao, “Not being (super) thin or solid is hard: A study of grid Hamiltonicity,” Computational Geometry, 42(6–7):582–605, 2009.

[4] Buro, Michael. ”Simple Amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs.” In International Conference on Computers and Games, pp. 250-261. Springer, Berlin, Heidelberg, 2000.

[5] C. Bazgan, B. Escoffier, and V. Paschos, “Completeness in standard and differential approximation classes: Poly-(D)APX- and (D)PTAS-completeness”, Theoretical Computer Science, 339(2–3):272–292, 2005.

[6] P.-A. Champin and C. Solnon, “Measuring the similarity of labeled graphs,” in Proceedings of the 5th International Conference on Case-Based Reasoning Research and Development (ICCBR’03), 80–95, 2003.

[7] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis, “On the complexity of protein folding,” Journal of Computational Biology, 5(3):423–465, 1998.

[8] Papadimitriou, Christos H., and Umesh V. Vazirani. ”On two geometric problems related to the travelling salesman problem.” Journal of Algorithms 5, no. 2 (1984): 231-246.

[9] K. A. Dill, "Theory for the folding and stability of globular proteins," Biochemistry, 24(6):1501–1509, 1985.

[10] K. A. Dill, S. Banu Ozkan, M. Scott Shell, and T. R. Weikl, "The protein folding problem," Annual Review of Biophysics, 37:289–316, 2008.

[11] K. A. Dill, S. Banu Ozkan, T. R. Weikl, J. D. Chodera, and V. A. Voelz, "The protein folding problem: when will it be solved?" Current Opinion in Structural Biology, 17(3):342–346, 2007.

[12] K. A. Dill and J. L. MacCallum, "The protein-folding problem, 50 years on," Science, 338(6110):1042–1046, 2012.

[13] E. D. Demaine and J. O'Rourke, Geometric folding algorithms: Linkages, origami, polyhedra, Cambridge University Press, 2007.

[14] Y. Duan and P. A. Kollman, "Computational protein folding: From lattice to all-atom," IBM Systems Journal, 40(2):297–309, 2001.

[15] D. Eppstein, "Subgraph Isomorphism in Planar Graphs and Related Problems," Journal of Graph Algorithms and Applications, 3(3):1–27, 1999.

[16] T. Feng, R. Uehara, and G. Viglietta, "Bicolored path embedding problems in protein folding models," In Proceedings of the 37th European Workshop on Computational Geometry (EuroCG'21), 24:1–24:9, 2021.

[17] K. Fun Lau and K. A. Dill, "A lattice statistical mechanics model of the conformational and sequence spaces of proteins," Macromolecules, 22(10):3986–3997, 1989.

[18] M. R. Garey and D. S. Johnson, Computers and intractability, Freeman San Francisco, 1979.

[19] W. E. Hart and S. Istrail. "Fast protein folding in the hydrophobic-hydrophilic model within three-eighths of optimal," Journal of Computational Biology, 3(1):53–96, 1996.

[20] W. E. Hart and A. Newman, "The computational complexity of protein structure prediction in simple lattice models," in Handbook of Computational Molecular Biology, CRC Press, 1–24, 2001.

[21] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter, "Hamilton paths in grid graphs," SIAM Journal on Computing, 11(4):676–686, 1982.

[22] S.-M. Hsieh, C.-C. Hsu, L.-F. Hsu, "Efficient method to perform isomorphism testing of labeled graphs," in Proceedings of the 6th International Conference on Computational Science and Its Applications (ICCSA'06), 422–431, 2006.

[23] T. Jiang, Q. Cui, G. Shi, and S. Ma, "Protein folding simulations of the hydrophobic- hydrophilic model by combining tabu search with genetic algorithms," The Journal of Chemical Physics, 119(8):4592–4596, 2003.

[24] G. Mauri, G. Pavesi, and A. Piccolboni. "Approximation algorithms for protein folding prediction," in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), 945–946, 1999.

[25] H. Müller, "Hamiltonian circuits in chordal bipartite graphs," Discrete Mathematics, 156(1–3):291–298, 1996.

[26] A. Neumaier, "Molecular modeling of proteins and mathematical prediction of protein structure," SIAM Review, 39(3):407–460, 1997.

[27] A. Newman. "A new algorithm for protein folding in the HP model," in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02), 876–884, 2002.

[28] Karp, Richard M. "Reducibility among combinatorial problems." In Complexity of computer computations, pp. 85-103. Springer, Boston, MA, 1972.

Publications

Journal Articles

- Tianfeng Feng, Ryuhei Uehara, and Giovanni Viglietta. **"Bicolored Path Embedding Problems Inspired by Protein Folding Models."** – *IEICE Transactions on Information and Systems*, Vol. E105-D, No.3, in printing, 2021.

- Tianfeng Feng, Leonie Ryvkin, Jérôme Urhausen, and Giovanni Viglietta. **"Complexity of Critter Crunch."** – *IEICE Transactions on Information and Systems*, Vol. E105-D, No.3, in printing, 2021.

Conference Papers

- Tianfeng Feng, Takashi Horiyama, Yoshio Okamoto, Yota Otachi, Toshiki Saitoh, Takeaki Uno, and Ryuhei Uehara. **"Computational Complexity of Robot Arm Simulation Problems."** – *The 12th International Workshop on Combinatorial Algorithms (IWOCA 2018)*, Lecture Notes in Computer Science Vol. 10979, pp. 177-188, 2018.

- Tianfeng Feng, Ryuhei Uehara, and Giovanni Viglietta. **"Bicolored path embedding problems in protein folding models."** – In Proceedings of the 37th European Workshop on Computational Geometry (EuroCG'21), pp. 24:1-24:9, 2021.