ALGORITHMIC ASPECTS OF SOME VERTEX ORDERING PROBLEMS

by

Akbar Rafiey

B.Sc., University of Tehran, 2013

Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Mathematics
Faculty of Science

Approval

Name:	**Akbar Rafiey**
Degree:	**Master of Science (Mathematics)**
Title:	***ALGORITHMIC ASPECTS OF SOME VERTEX ORDERING PROBLEMS***
Examining Committee:	**Chair:** Luis A. Goddyn Professor

Ladislav Stacho
Senior Supervisor
Associate Professor

Cedric Chauve
Co- Supervisor
Professor

Jan Manuch
Supervisor
Adjunct Professor

Ramesh Krishnamurti
Internal Examiner
Professor
Department of Computer Science

Date Defended: 11 August 2016

Abstract

Combinatorial Optimization problems play central role in applied mathematics and computer science. A particular class of combinatorial optimization problems is vertex ordering problems whose goal is to find an ordering on vertices of an input graph in such way that a certain objective cost is optimized. This thesis focuses on three different vertex ordering problems.

First, we consider a variant of Directed Feedback Arc Set problem with applications in computational biology. We present an efficient heuristic algorithm for this problem. Next, we concentrate on parameterized complexity of Minimum Linear Arrangement problem while the parameter is size of vertex cover. We formulate the problem as Quadratic Integer Programming and propose two new techniques to tackle the problem. Finally, we introduce a vertex ordering problem on bipartite graphs with applications in different areas. We give a general algorithmic strategy for the problem, as well as, polynomial-time algorithms for three classes of graphs.

Keywords: Combinatorial Optimization; Vertex Ordering Problems; Directed Feedback Arc Set; Minimum Linear Arrangement; Bipartite Ordering Problem

Acknowledgements

I would like to thank all the people who helped me during my master studies. First of all, I would like to thank my supervisors. I am thankful to Ladislav Stacho and Jan Manuch for many useful comments on the first drafts of this thesis and for many useful discussions and meetings. I would like to thank Cedric Chauve. Without him the results in the third chapter of this thesis would not have existed. He introduced me to the field of computational biology. He was really patient with me during this project. I am also hugely thankful to him for providing the experimental results in the third chapter.

I would like to thank my brother Arash. I have learned a lot from him. He was always available for discussion on discrete math problems. I am extremely thankful to him for his time and guidance during my time at SFU.

I would also like to thank my friends for filling my life with things other than discrete mathematics, and would like to show my profound gratitude to my family. I owe all of my accomplishments to the unconditional love and support of my parents.

Contents

List of Tables

List of Figures

Chapter 1

Notations and Preliminaries

In applied mathematics and theoretical computer science *combinatorial optimization* problems consist of finding optimal object over a finite set of objects. In many such problems, exhaustive search over set of feasible solutions which is discrete is not possible. Famous examples of problems involving combinatorial optimization are *Traveling Salesman Problem*, briefly TSP, *Minimum Spanning Tree* (MST) problem, and *Shortest Path* problem. Many problems in this field are inspired by practice. We use combinatorial optimization in our daily life. We search for short paths to our destinations (Google map). A mailman or a doctor tries to plan his tour in an efficient way. Companies aim to assign their jobs to a person and machines in a way that they use least amount of resources and as fast as possible. These forms of elementary problems are not just considered by mathematicians, but also are considered in real world.

Combinatorial optimization arises in different areas such as *operation research*, *algorithm theory* and *computational complexity theory*. Its important applications span over several fields including *machine learning*, *artificial intelligence*, *mathematics*, *auction theory* and *software engineering*.

A particular class of combinatorial optimization problems are *vertex ordering* problems whose goal is to find a linear ordering on the vertices of an input graph in such way that a certain objective cost is optimized. A linear ordering is a labeling of the vertices of a graph with distinct integers. A large amount of relevant problems in different areas can be formulated as vertex ordering problems including *network optimization*, *VLSI circuit design*, *computational biology*, *graph theory*, *scheduling*, etc. On the other hand, the minimal values of some vertex ordering costs are also related to an interesting graph theoretic invariant of graphs. In most cases of vertex ordering problems, exhaustive search over the feasible solutions set is not practical and possible. A well known example is TSP that can be seen as a vertex ordering problem. Since numerous of practical problems can be modeled as vertex ordering problems, various algorithmic and optimization approaches have been developed to tackle these problems.

This thesis concentrates on some of vertex ordering problems and is organized as follows. In the remaining part of this chapter, we give the necessary notation and definitions for the thesis. In Chapter 2, a general framework for vertex ordering problems is given, as well as, some well studied and some relatively new vertex ordering problems are defined and related results are surveyed. The focus of Chapters 3, 4 and 5 is on three vertex ordering problems, namely DIRECTED FEEDBACK ARC SET, MINIMUM LINEAR ARRANGEMENT and BIPARTITE ORDERING PROBLEM. In this chapter we also discuss our new results and conjectures.

1.1 Decision Problems and Optimization Problems

Problem A *problem* or *language* is a set L of strings over a finite alphabet Σ, that is $L \subseteq \Sigma^*$. A string $s \in L$ is a *yes* instance of L and a string $s \notin L$ is a *no* instance of L. A *decision problem* asks if a given s is in L or not. A *parameterized problem* is a subset Φ of $\Sigma^* \times \mathbb{N}$, that is, every instance of Φ is a pair, a string over Σ^* and a natural number called the *parameter*. In an *optimization problem* we are given as input an instance s, defining a set $\mathcal{F}(s)$ of *feasible solutions* for a problem. The objective is to find an element $x \in \mathcal{F}(s)$ that minimizes or maximizes a certain problem specific *objective function* $g(x)$. If the objective is to minimize (maximize) $g(x)$ the problem is referred to as a minimization (maximization) problem. The minimum (maximum) value of the objective function over all feasible solutions for an instance s is called the *optimal value* of the objective function.

1.2 Big-Oh notation and the class $\mathcal{P}$

Algorithm An *algorithm* for the problem L is a set of rules that precisely defines a sequence of operations so that for a string s it determines whether $s \in L$. The *running time* of an algorithm is measured in the number of steps (operations) the algorithm performs to verify if $s \in L$.

One of the core concepts in computer science and algorithm design is running time of algorithms. To express running time of an algorithm we employ big-Oh notation which suppresses constant factors and lower order terms. That is, for two functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ we say $f(n) = O(g(n))$ if there are constants $c_0, c_1 > 0$ such that for all $x \geq c_0$, $f(x) \leq c_1 g(x)$. A natural question to ask, is for which problems we have efficient algorithms. Informally by efficient, we mean that we are able to apply the algorithm on large input instances, and get an answer in reasonable time. As this is not a precise definition, it is fairly hard to work with. Therefore we say that an algorithm is efficient if it runs in polynomial time. That is, if there exists a constant c independent of the input so that the algorithm runs in $O(n^c)$ time where n is size of the input. Now, obviously this definition is too wide, by this definition an algorithm running in $O(n^{100})$ is efficient, something that is not the case. In fact, if executed today on the earth's fastest computer, the algorithm would not terminate within the life span of the universe even on an input of size 2! However, incomplete as this definition may seem, it has proven practical. This is because for most problems that we find polynomial time algorithms the exponent c mentioned above is 10 or less. We denote the set of all problems that are solvable in polynomial time by $\mathcal{P}$.

A superset of class $\mathcal{P}$ is the class $\mathcal{NP}$. The class $\mathcal{NP}$ is the class of problems that are *polynomial time verifiable.* That is, if we are given a problem instance and a solution, there is a polynomial time algorithm that can check whether the proposed potential solution indeed is a solution of the given instance.

1.3 Reductions and $\mathcal{NP}$-completeness

There are many important problems for which no polynomial time algorithm is known. In fact, it is the case that they have proven very hard to find efficient algorithms for. These problems are often easy to formulate, however, in their simplicity they seem not to be solvable by anything but a brute force search. In 1972 Cook and Levin provided an explanation for why these problems are so difficult to solve [20, 64]. They showed that these problems are interconnected and providing a polynomial time algorithm for one of them means in fact having a polynomial time algorithm for all of them. The concept of *reductions* from one problem to another was used by them to show this. A reduction is simply an algorithm to transform an instance of one problem to an instance of another, so that the solution of the second can be used to solve the first. For decision problems, problems where the output should be either yes or no, we next formalize this notion. For a decision problem P and an instance I of the problem, let the answer to I with respect to P be denoted $A_P(I)$. This answer is either *True* ($I \in P$) or *False* ($I \notin P$). Now, a reduction from a problem P to another problem P' is a function f that takes instances of P to instances of P' so that $A_{P'}(f(I)) = A_P(I)$. We say that f is a *polynomial time reduction* if there is a polynomial time algorithm computing f. Therefore, if there is a polynomial time reduction from some problem A to a problem B then a polynomial time algorithm for B automatically yields a polynomial time algorithm for A. Seen from this point of view, we can say that B is at least as hard as A, or $A \leq_T B$.

What Cook and Levin realized is that some of the problems in $\mathcal{NP}$ are special in the way that they can be used to encode all other problems in the class. We say that a problem A is $\mathcal{NP}$-complete, or belongs to the class $\mathcal{NPC}$ if $A \in \mathcal{NP}$ and for all $B \in \mathcal{NP}$, $B \leq_T A$. The famous result of Cook and Levin states that 3-SAT is $\mathcal{NP}$-complete. In 3-SAT problem we are given a boolean formula of form $C = (x_1^1 \vee x_2^1 \vee x_3^1) \wedge (x_1^2 \vee x_2^2 \vee x_3^2) \wedge \cdots \wedge (x_1^n \vee x_2^n \vee x_3^n)$ where each $C_i = (x_1^i \vee x_2^i \vee x_3^i)$ is called a *clause* and each x_i^j is either a positive or negative occurrence of a boolean variable. The problem asks whether there is a truth assignment to variables such that each clause is satisfied. If interested, the reader may consult [83] for the proof of the following theorem.

Theorem 1. *[83]* 3-SAT $\in \mathcal{NPC}$.

Extensive body of research have appeared and flourished to tackle the difficulty of these problems and in particular to attempt to settle the famous problem "$\mathcal{P}$ versus $\mathcal{NP}$" [6, 41, 46]. Approximation algorithm and fixed-parametrized tractability are among hottest topics in algorithm design that have drawn computer scientists' attention [22, 29, 75, 88, 89]. Next, we provide a brief introduction to these two areas of research. Related results for vertex ordering problems are surveyed later on.

1.4 Approximation Algorithms

One of the classical methods to deal with intractable problems is to design an approximation algorithm that in polynomial time finds provably close enough solution to an optimal one.

Definition 1. *[89] An α-approximation algorithm for an optimization problem is a polynomial time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution. In this case we say the problem is α-approximable.*

Definition 2. *[89] A polynomial-time approximation scheme ($\mathcal{PTAS}$) is a family of algorithms $\{\mathscr{A}_\epsilon\}$, where for each $\epsilon > 0$, $\mathscr{A}_\epsilon$ is a $(1+\epsilon)$-approximation algorithm (for minimization problems) or a $(1-\epsilon)$-approximation algorithm (for maximization problems).*

Note that the running time of the algorithm $\mathscr{A}_\epsilon$ is allowed to depend arbitrarily on ϵ. This dependence could be exponential in $1/\epsilon$, or worse.

Definition 3. *[89] A fully polynomial-time approximation scheme ($\mathcal{FPTAS}$) is an approximation scheme such that the running time of $\mathscr{A}_\epsilon$ is bounded by a polynomial in $1/\epsilon$.*

A combinatorial optimization problem belongs to the class $\mathcal{APX}$ if it is α-approximable for some constant α. It is known that $\mathcal{FPTAS} \subseteq \mathcal{PTAS} \subseteq \mathcal{APX}$, where the inclusions are strict if and only if $\mathcal{P} \neq \mathcal{NP}$. We refer interested reader to [89], where they find a through tour of important paradigms for designing approximation algorithms for combinatorial optimization problems.

1.5 Fixed-Parameterized Tractability

A parameterized problem P can be considered as a set of pairs (I, k) where I is the problem instance and k (usually an integer) is the parameter. A parameterized problem P is *fixed-parameter tractable* if membership of (I, k) in P can be decided in time $O(f(k)|I|^c)$, where

$|I|$ is the size of I, $f(k)$ is a computable function, and c is a constant independent from k and I. The class of fixed parameter tractable problems is denoted $\mathcal{FPT}$. Downey and Fellows [30] introduced a framework from complexity theory and derived a hierarchy of complexity classes $\mathcal{W}[t]$ (the $\mathcal{W}$ hierarchy), whose definition is based on logical circuit families and fixed-parameter reduction. A *fixed-parameter reduction* is defined as follows. Let P and P' be parameterized problems with parameters k and k', respectively. A fixed-parameter reduction R from P to P' is a function that transforms a problem instance (I, k) of P into a problem instance (I', k') of P', such that:

i) $(I, k) \in$ P if and only if $(I', k') \in$ P' with $k' \leq g(k)$ for a fixed computable function g

ii) R is of complexity $O(f(k)|I|^c)$

The parametrized analog of the well known $\mathcal{NP}$ is the class $\mathcal{W}[1]$, and $\mathcal{W}[1]$ hardness is the basic evidence that a parameterized problem is not likely to be fixed-parameter tractable. We refer to [30] for the formal definition of complexity classes $\mathcal{W}[t]$.

1.6 Graph Notations

An *undirected graph* (*directed graph* or *digraph*) $G = (V, E)$ is a set $V = V(G)$ of vertices and a set $E = E(G)$ of edges (arcs) that is a subset of the set of unordered (ordered) pairs of V. If $(v, u) \in E$, then we say v and u are *adjacent* or *connected.* In this thesis, we will say that G has $|V| = n$ vertices and $|E| = m$ edges. If $(v, v) \notin E$, G is referred to as a *simple graph.* If $(u, v) \in E \Rightarrow (v, u) \in E$ the graph is *undirected*, otherwise it is called a *directed* graph or simply *digraph.* If some weight function $w : E \to \mathbb{R}$ is supplied, we say that G is weighted and that $w((u, v))$ is the weight of the edge (u, v). A weight function can be defined similarly on the vertices. All graphs in this thesis are simple and undirected unless stated otherwise. For a simple undirected graph G, its *complement*, is defined as $\overline{G} = (V, \{(u, v) : u \neq v, (u, v) \notin E\})$.

A sequence P of vertices, $v_1, v_2, \ldots, v_n$ where $(v_i, v_{i+1}) \in E(G)$ is called a *walk* of length $n - 1$. If $i \neq j \Rightarrow v_i \neq v_j$, P is a *path.* A *closed walk* is a walk $v_0, v_1, v_2, \ldots, v_n, v_0$ from a vertex v_0 back to itself. A *cycle* is a closed walk with all vertices (and hence all edges) distinct (except the first and last vertices). An undirected graph G is called *tree* if it contains no cycle. A *rooted tree* is a tree in which one vertex has been designated as the *root.* In a rooted tree a *child* of a vertex $v \in V(T)$ is a vertex directly adjacent to v when moving away from the root. Note that it is well defined as a tree contains no cycles. A *leaf* is a vertex with no children. The *parent* of v is a vertex directly adjacent to v when moving towards the root Recall that since there is no cycle in a tree then there is only one such a vertex. Vertices with the same parent are called *siblings.* A vertex is called *descendant* (*ancestor*) of v if it is reachable from v by repeatedly proceeding from a parent to a child (a child to its parent). A tree is *binary* if each vertex has at most two children.

A graph $H = (V_H, E_H)$ is a *sub-graph* of $G = (V, E)$ if $V_H \subseteq V$ and $E_H \subseteq E$. Given a subset of vertices $W \subseteq V$, the *sub-graph* of G *induced* by W is the graph $G[W] = (W, E \cap W \times W\})$. For $G = (V, E)$, $G - S$ denotes induced sub-graph by $V \setminus S$ where $S \subseteq V$. For $F \subseteq E$, $G - F$ denotes graph $G' = (V, E \setminus F)$.

The set of adjacent vertices to a vertex v is called its *neighborhood* and denoted by $N(v)$ i.e. $N(v) = \{u \in V : (v, u) \in E\}$. The *degree* of a vertex v is $d(v) = |N(v)|$. The *maximum degree* of a graph G is $\Delta(G) = \max\limits_{v \in V} d(v)$.

In digraph $D = (V, A)$, a vertex $v \in V(D)$ is an *in-neighbor* (*out-neighbor*) of a vertex u if $(v, u) \in A$ ($(u, v) \in A$). The *in-degree* (*out-degree*) of a vertex v is the number of its in-neighbors (out-neighbors) in D. If a simple digraph $D = (V, E)$ has no cycles, we say that D is *acyclic.* A *topological* ordering of the digraph D is an ordering of the vertices $(v_1, v_2, \dots, v_n)$ so that for every arc (v_i, v_j) in A we have that $i < j$. One can show that a simple digraph D has a topological ordering if and only if it is acyclic.

We denote a *bipartite graph* by $H = (B, S, E)$ where the vertex set for H is partitioned into two parts B and S and $E \subseteq B \times S$.

A *clique* K in a graph G is a subset of the vertices of G so that for each distinct u and v in K, $(u, v) \in E(G)$. A clique in a bipartite graph is called a *biclique.* An *independent* set I is a clique in $\overline{G}$. A *vertex cover* of G, denoted by $vc(G)$, is a subset of vertices in G such that for each edge $(u, v) \in E(G)$, at least one of u or v is in $vc(G)$.

A *tree decomposition* of a graph G is a pair (X, T) where T is a tree whose vertices are called *bags* and $X = (\{X_i : i \in V(T)\})$ is a collection of subsets of $V(G)$ such that

1. $\cup_{i \in V(T)} X_i = V(G)$,
2. for each edge $(u, v) \in E(G)$, there is an $i \in V(T)$ such that $u, v \in X_i$, and
3. for each $v \in V(G)$ the set of bags $\{i : v \in X_i\}$ forms a subtree of T.

The *width* of a tree decomposition $(X_i : i \in V(T), T)$ equals $\max_{i \in V(T)}\{|X_i| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. We use notation $tw(G)$ to denote the treewidth of a graph G. A tree decomposition is called a *nice tree decomposition* if the following conditions are satisfied: Every bag of the tree T has at most two children; if a bag t has two children t_1 and t_2, then $X_t = X_{t_1} = X_{t_2}$; and if a bag t has one child t_1, then either $|X_t| = |X_{t_1}| + 1$ and $X_{t_1} \subset X_t$ or $|X_t| = |X_{t_1}| - 1$ and $X_t \subset X_{t_1}$. It is possible to transform a given tree decomposition into a nice tree decomposition in time $O(|V| + |E|)$ [10]. Note that this transformation does not change $tw(G)$.

Chapter 2

Introduction to Vertex Ordering Problems

Many vertex ordering problems with various cost functions have been considered [12, 27]. Because of their applications and beautiful theoretical aspects, they have been studied from different algorithmic aspects such as exact algorithms, fixed parametrized tractability, approximation algorithms [27]. In this chapter, we first give a general setting for vertex ordering problems. In the rest of this chapter, some well studied and a relatively new vertex ordering problems are defined. A survey on related results to the introduced vertex ordering problems is also provided.

2.1 General Framework for Vertex Ordering Problems

A *vertex ordering* π of a graph $G = (V, E)$ with n vertices is a bijective function $\pi : V \to \{1, 2, \dots, n\}$. We denote by $\Pi(G)$ the set of all vertex orderings of the graph G. An ordering is also called a *layout* or a *linear arrangement* or a *labeling* or a *numbering* of the vertices of a graph. A natural way to represent an ordering π on vertices of graph G is to align its vertices on a line, mapping vertex v to position $\pi(v)$, as shown in Figure 2.1. We may often refer to position of a vertex in π as its *rank* in π. Moreover, we say u has a lower (higher) rank than v if $\pi(u) < \pi(v)$ ($\pi(v) < \pi(u)$). For a vertex ordering π and $v \in V$, we denote by $\pi_{\preceq,v}$ the set of vertices that appear before v in the ordering. More precisely, $\pi_{\preceq,v} = \{u \in V : \pi(u) \leq \pi(v)\}$.

An *ordering cost* is a function F that associates to each vertex ordering π of a graph G an integer $F(\pi, G)$. Let F be an ordering cost; the (optimization) vertex ordering problem associated with F consists in determining some vertex ordering $\pi^* \in \Pi(G)$ of an input graph G such that

$$F(\pi^*, G) = \min_{\pi \in \Pi(G)} F(\pi, G) := F(G).$$

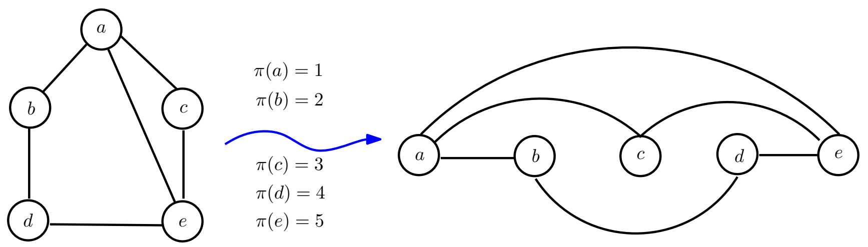


Figure 2.1: An example of aligning vertices of a graph to a line under ordering π.

In terms of exact algorithm, Bodlaender *et al.* in [12] considered vertex ordering problems with the two following cost functions

$$\max_{v \in V} f(G, \pi_{\prec,v}, v) \tag{2.1}$$

$$\sum_{v \in V} f(G, \pi_{\prec,v}, v) \tag{2.2}$$

where $f(G, S, v)$ for $S \subseteq V(G)$ is a polynomial time computable function and the value of $f(G, S, v)$ does not depend on the ordering of set S. They showed that any vertex ordering problem with cost functions in form (2.1) or (2.2) can be solved in both (a) time and space $O^*(2^n)$, and (b) time $O^*(4^n)$ and polynomial space, where $O^*(f(n))$ is shorthand for $O(f(n).poly(n))$. Several examples of vertex ordering problems that can be formulated in form (2.1) or (2.2) are given in [12].

Theorem 2. *[12] Let f be a polynomial time computable function, mapping each 3-tuple, consisting of a (di)graph $G = (V, E)$, a vertex set $S \subseteq V$, and a vertex $v \in V$ to an integer. Then we can compute in $O^*(2^n)$ time and $O^*(2^n)$ space the values in 2.1 and 2.2 for a given (di)graph $G = (V, E)$.*

Without going into details, the approach uses dynamic programming. At ith iteration of their algorithm, they find optimal solutions for all the subsets $S \subset V$ of size i having solutions for all subsets of size $i - 1$ from the previous iteration. They improved the space complexity by employing recursion instead of dynamic programming, however, it increases time complexity to $O^*(4^n)$.

Theorem 3. *[12] Let f be a polynomial time computable function, mapping each 3-tuple, consisting of a (di)graph $G = (V, E)$, a vertex set $S \subseteq V$, and a vertex $v \in V$ to an integer. Then we can compute in $O^*(4^n)$ time and polynomial space the values in 2.1 and 2.2 for a given (di)graph $G = (V, E)$.*

2.2 Directed Feedback Arc Set Problem

Directed Feedback Arc Set, briefly DFAS, is a well studied vertex ordering problem that has been considered by computer scientists and mathematicians for decades. In this section, we formally define the problem. Next, we motivate the problem and discuss some of its applications. Related results and algorithmic techniques that are applied to solve this problem are mentioned afterwards.

2.2.1 Problem Definition

DFAS problem is defined on directed graphs. In a digraph D, a set S of arcs is called *feedback arc set (FAS)* if $D - S$ is acyclic ($D - S$ does not contain any directed cycle). For the depicted digraph in Figure 2.2, removing arcs $(4, 1)$ and $(3, 1)$ leaves an acyclic digraph. Hence, set $F = \{(4, 1), (3, 1)\}$ is a FAS set for the given digraph. Similarly, removing vertex 1 from the digraph yields an acyclic digraph so $F = \{1\}$ is FVS for the digraph.

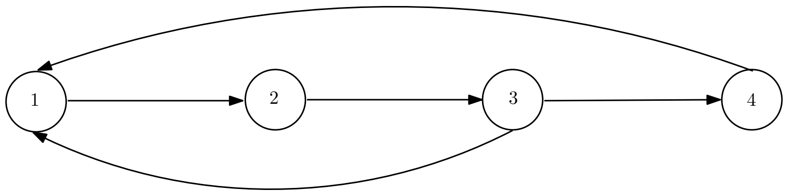


Figure 2.2: An example for DFAS problem.

Let $\tau_A(D)$ denote the minimum number of elements in directed feedback arc set of D. In the *weighted* version of DFAS problem, a non-negative integer weight is assigned to each arc. In [12] and [50], the authors proved that (weighted) DFAS problem is equivalent to a vertex ordering problem. The following proposition shows that DFAS problem can be formulated as a vertex ordering problem.

Proposition 1. *[12, 50] Let $D = (V, A)$ be a directed graph, and let $w : A \to N$ be a function that assigns each arc a non-negative integer weight. Let k be a positive integer. There exists a set of arcs $S \subseteq A$ with $(V, A - S)$ acyclic and $\sum_{(v,u)\in S} w((v, u)) \leq k$, if and only if there is a vertex ordering π on V, such that $\sum_{\substack{(v,u)\in A \\ \pi(v)>\pi(u)}} w((v, u)) \leq k$.*

Therefore, the cost function for (weighted) DFAS problem is given by:

$$DFAS(\pi, D) = \sum_{v\in V} \sum_{\substack{(v,u)\in A \\ \pi(v)>\pi(u)}} w((v, u)).$$

The decision version of DFAS problem is

DFAS(D,k)
Input: A digraph $D=(V,A)$ and an integer k.
Question: Is there some ordering $\pi \in \Pi(G)$ on vertices of D such that $DFAS(\pi,G) \leq k$?

Directed Feedback Vertex Set (abbreviated to DFVS) problem is defined analogously to DFAS problem. A set S of vertices in D is called *directed feedback vertex set* if $D-S$ is acyclic. Let $\tau_V(D)$ denote the minimum number of elements in directed feedback vertex set of D. It turns out that DFAS and DFVS are of the same complexity up to a polynomial factor.

Proposition 2. *[50] For every digraph D there exist digraphs D' and D'' such that $\tau_V(D) = \tau_A(D')$ and $\tau_A(D) = \tau_V(D'')$. The digraphs D' and D'' can be constructed from D in polynomial time.*

Proof. We construct D' as follows. For each vertex $v \in V(D)$, create two new vertices v^- and v^+ in $V(D')$ i.e. $V(D') = \{v^-, v^+ : \forall v \in V(D)\}$. The arc set in D' consists of: (v^-, v^+) for all $v \in V(D)$ and $(v^+, u^-) \in A(D')$ for each $(v,u) \in A(D)$. The construction for D'' is as follows: for each arc $a_1 = (v,u) \in A(D)$ we consider vertex $e_1 \in V(D'')$. Plus, for two vertices $e_1 = (v_1,u_1) \in V(D'')$ and $e_2 = (v_2,u_2) \in V(D'')$ there is an arc $(e_1,e_2) \in A(D'')$ if $u_1 = v_2$. Often, D'' is called the line graph of D. It is not difficult to verify that the equality of this proposition indeed hold. □

2.2.2 Applications and Motivations

DFVS problem has a variety of applications in areas such as operating systems [82], database systems [45], and circuit testing [62]. An electronic circuit can be modeled as a digraph by letting the gates be vertices and the wires between the gates be the arcs. Finding the small set of arcs whose removal gives us an acyclic digraph can be useful in reducing hardware overhead needed for testing the circuit using so-called scan registers (see [59]). In particular, the DFVS problem has played an essential role in the study of deadlock recovery in database systems and in operating systems [45, 82]. In such a system, the status of system resource allocations can be represented as a directed graph D (i.e., the system resource allocation graph), and a directed cycle in D represents a deadlock in the system. Therefore, in order to recover from deadlocks, we need to abort a set of processes in the system, i.e., to remove a set of vertices in the digraph D, so that all directed cycles in D are broken. Equivalently, we need to find a FVS in the digraph D. In Chapter 3, an application of the problem in computational biology is discussed.

2.2.3 Related Results

It was proven in [55] by Karp that DFAS is $\mathcal{NP}$-complete. Bang-Jensen and Thomassen [8] proved that DFVS is $\mathcal{NP}$-complete even for tournaments they also conjectured that DFAS also is $\mathcal{NP}$-complete for tournaments. A *tournament* is a directed graph in which every pair of distinct vertices is connected by an arc. Their conjecture was proven to be correct independently in [3, 13]. Interestingly, DFAS is polynomial time solvable for planar digraphs [7, 67], however, DFVS is $\mathcal{NP}$-complete for planar digraphs [46].

Feedback Vertex Set problem (briefly FVS) on undirected graphs is known to be fixed-parameter tractable. There are two main approaches that are used to design $\mathcal{FPT}$ algorithms for FVS, namely, *shortest cycle* approach and *iterative compression* approach. A very nice overview of the *shortest cycle* approach is given in [50]. The approach is as follows. Without loss of generality, if $G = (V, E)$ has a vertex of degree one, that vertex is not in FVS. If a vertex $v \in V(G)$ has degree two then, either v is in a cycle of length three (in which case we have a short cycle of G) or we can replace the edges (u, v) and (v, w) incident to v by (u, w), delete vertex v from G (accordingly delete edges incident to v) without creating parallel edges and without changing $\tau_V(G)$. By continuing this process we are left with a graph which either has a cycle C of length three or the minimum degree of G is 3. By a theorem of Erdos and Posa [33], in a graph with minimum degree three there is a cycle C of length at most $2\log|V| + 1$. For each vertex $v \in V(C)$, run the algorithm recursively on $G - v$ with parameter $k - 1$. The running time of the algorithm is $O^*(2\log|V| + 1)^k$. Since $(\log|V|)^k \leq |V| + (3k\log k)^k$ then FVS is $\mathcal{FPT}$ on undirected graphs. A faster algorithm is given in [24] using *iterative compression* approach.

Iterative compression approach was articulated as a general $\mathcal{FPT}$ design technique in [25]. We give a very high level overview of the approach that is used to design an $\mathcal{FPT}$ algorithm for FVS problem in undirected graphs. The problem setting is as follows: we are given a graph $G = (V, E)$, an integer parameter k, and a set $S \subseteq V$ of size $k + 1$ such that $G - S$ is a forest; the question is to decide whether G has a feedback vertex set of size at most k. To see how answering this question is helpful for solving FVS problem, we proceed as follows. Suppose there is an algorithm $\mathscr{A}$ that can answer this question in time $O(f(k)n^c)$. Order vertices of G in an arbitrary way. Let $G[v_1, v_2, \ldots, v_{k+3}]$ be the induced graph by the first $k+3$ vertices. Trivially, this graph has a feedback vertex set of size $k+1$. Run algorithm $\mathscr{A}$ to check whether it has feedback vertex set of size k or not. If no, then G cannot have a feedback vertex set of size k, and we are done. If yes, then let S' be a solution of size at most k. Now consider the graph $G[v_1, \cdots, v_{k+4}]$ on the first $k + 4$ vertices. This graph has a feedback vertex set of size $k + 1 : S' \cup v_{k+4}$. Now we have a situation that is identical to the one we started out with. Invoke algorithm $\mathscr{A}$ on the graph $G[v_1, \cdots, v_{k+4}]$ and repeat as before. The term " iterative compression" should make more sense now: we iteratively include new vertices, obtain a solution of size $k + 1$ in the process, and try to compress it. If we fail to compress the solution at some stage, it implies that the original instance does not have a feedback vertex set of size k. Such an algorithm $\mathscr{A}$ is given in [24].

It was a long open standing problem whether DFAS is fixed-parameter tractable [31]. Eventually after fifteen years, in [15], the authors positively answer this question by developing new algorithmic techniques and using iterative compression approach. They gave an algorithm with running time $4^k k! n^{O(1)}$. Note that by Proposition 2, DFAS and DFVS have the same parametrized complexity.

Theorem 4. *[15] The* DFVS *problem is solvable in time $O(n^4 4^k k^3 k!)$.*

We point out that it is straight forward to see that Theorems 2 and 3 can be applied to DFAS problem. To be more precise, the cost function $DFAS(\pi, D) = \sum_{v \in V} f(D, \pi_{\prec,v}, v)$ is of form (2.2) where $f(D, \pi_{\prec,v}, v) = \sum_{\substack{(v,u)\in A \\ \pi(v)>\pi(u)}} w((v,u))$. As a result, we have the following two theorems

Theorem 5. *[12] There is an algorithm that solves* DFAS *problem in $O^*(2^n)$ time and $O^*(2^n)$ space for a given digraph $D = (V, A)$.*

Theorem 6. *[12] There is an algorithm that solves* DFAS *problem in $O^*(4^n)$ time and polynomial space for a given digraph $D = (V, A)$.*

2.3 Minimum Linear Arrangement Problem

For years, many computer scientists have been fascinated by the interesting algorithmic aspect of Minimum Linear Arrangement problem. That is because this problem has applications in various areas including job scheduling, graph drawing, etc [27, 79]. Besides to its applications, its difficulties to design an efficient algorithm for, challenge lots of algorithm designers. Here, we formally define the problem and mention some of its applications. At the end of this section, a brief overview of the algorithmic results for this problem is given.

2.3.1 Problem Definition

The Minimum Linear Arrangement, briefly MLA, problem consists in finding a linear ordering of the vertices of a given graph such that the sum of the *edge lengths* is minimized. For an edge $e = (u, v)$, *length* of e with respect to an ordering π is $l(e, \pi) = |\pi(u) - \pi(v)|$. As an example, in Figure 2.3, the length of edge between vertex 1 and vertex 4 is three, with respect to the given ordering of vertices. The formulation of cost function for MLA is as follows:

$$LA(\pi, G) = \sum_{(u,v)\in E} |\pi(u) - \pi(v)|$$

The decision version of MLA problem is

MLA(G, k)
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a linear ordering $\pi \in \Pi(G)$ on vertices of G such that $LA(\pi, G) \leq k$?

An example is given in Figure 2.3. In the example the cost function $LA(\pi, G) = 8$ since the length for $l(1, 3) = 2$ and $l(1, 4) = 3$ plus $l(1, 2) = l(2, 3) = l(3, 4) = 1$.

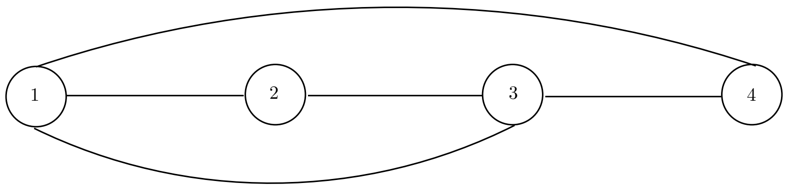


Figure 2.3: An example for MLA problem.

2.3.2 Applications and Motivations

Harper defined MLA problem in 1964 to develop error-correcting codes with minimal average absolute error [52]. Ten years later it became important for the VLSI technology, as it was considered as a simplified mathematical model of the placement phase, where the nodes represent the modules and edges of the graph correspond to the interconnections [87]. Another application of the MLA is in the area of graph drawing, as the sum of all edge lengths can be seen as a criterion for a good presentation of a graph [80]. The problem finds other applications in a range of fields, including the software diagram layout in general and especially for entity relationship models [16] and data flow diagrams [44]. It has been shown to be relevant to solve the Single Machine Job Scheduling problem [1, 78]. Furthermore, it has even been used in computational biology [56], for example as an over-simplified model of some nerve activity modeling in the cortex [71].

2.3.3 Related Results

Garey *et al.* [46] proved that MLA is intractable in general.

Theorem 7. *[46]* MLA $\in \mathcal{NPC}$.

MLA remains $\mathcal{NP}$-complete even on bipartite graphs [36]. However, there are special cases in which the MLA can be solved efficiently. We present several of these cases in the following. Shiloach 1979 [81], gave an algorithm for MLA on trees with time complexity $O(n^{2.2})$. Later on in 1988 [18], Chung by using structural properties of tree combined with novel ideas gave a faster algorithm with time complexity $O(n^{\frac{\log 3}{\log 2}})$ for MLA on trees. We refer to [27, 79] where more results have been discussed for MLA problem.

MLA is also known to be fixed-parameter tractable when the parameter is $m+k$ where m is number of edges of the graph. Note that MLA of graph G is at least m. Consequently, parameterized MLA is trivially solvable when the parameter is an integer k (we reject a graph with more than k edges and solve MLA by brute force if the graph has at most k edges). Gutin *et al.* in [49] gave an algorithm which decides in time $O(m+n+5.88^k)$ whether a given graph with m edges and n vertices admits a minimum linear arrangement of cost at most $m+k$.

Most parameterized problems are $\mathcal{FPT}$ parameterized by the *treewidth* of the input graph. However, vertex ordering problems are a notable exception (see also [38] for further examples of parameterized problems that are $W[1]$- hard when parameterized by the input treewidth, or even the input vertex cover number). Parameterized by the treewidth of the input graph, BandWidth is known to be hard for $W[t]$ for all t (this follows from the results in [11]). Whether something similar holds for MLA is unknown. This general situation motivates studying the complexity of these problems, parameterized by structural parameters even stronger than treewidth. A powerful parameter that makes many of vertex ordering problems tractable is the size of the *vertex cover* of the input graph. In [40], authors investigated the parameterized complexity of various vertex ordering problems considering the size of the minimum vertex cover of the input graph as the parameter. The main ingredient of their approach is a classical algorithm for Integer Linear Programming when parameterized by dimension, designed by Lenstra and later improved by Kannan. Using the structural property of vertex cover as the parameter they formulate BandWidth, CutWidth Imbalance and Distortion as an Integer Linear Programming with bounded number of variables. Thereafter, they applied the result of Lenstra [63] and Kannan [54].

Theorem 8. *([54, 63, 42])* p-VARIABLE INTEGER LINEAR PROGRAMMING FESEABILITY *can be checked using $O(p^{2.5p+o(p)}.L)$ arithmetic operations and space polynomial in L. Here L is number of bits needed to represent the input.*

It turns out that the objective function for MLA parametrized by vertex cover is not a linear one. In fact, the objective function is quadratic. Fellows *et al.* [39] took the first step to answer the question *whether* MLA *considering minimum vertex cover as the parameter is $\mathcal{FPT}$?.* They gave a factor $(1+\epsilon)$-approximation algorithm for MLA parameterized by (ϵ, k), where k is the vertex cover size of the input graph.

Lokshtanov in [65] positively answered this question. Not only he gives an $\mathcal{FPT}$ algorithm for MLA parameterized by vertex cover number, but also his approach gives an $\mathcal{FPT}$ algorithm for any Quadratic Integer Programming with bounded number of variables and coefficients. It is discussed in more detail in Chapter 4.

Note that, in order to apply Theorems 2 and 3 for MLA problem we need to reformulate cost function for MLA.

Proposition 3. *[12] For each graph $G = (V, E)$, and for each vertex ordering π of G,*

$$LA(\pi, G) = \sum_{(u,v)\in E} |\pi(u) - \pi(v)| = \sum_{v\in V} |\{(x, y) \in E : \pi(x) < \pi(v) < \pi(y)\}|$$

It follows from Proposition 3 that MLA problem can be written in the form 2.2, that is $MLA(G) = \min_{\pi\in\Pi(G)} \sum\limits_{v\in V} f(G, \pi_{\prec,v}, v)$ where $f(G, \pi_{\prec,v}, v) = |\{(x, y) \in E : \pi(x) < \pi(v) < \pi(y)\}|$. As a result we we have the following two theorems:

Theorem 9. *[12] There is an algorithm that solves* MLA *problem in $O^*(2^n)$ time and $O^*(2^n)$ space for a given graph $G = (V, E)$.*

Theorem 10. *[12] There is an algorithm that solves* MLA *problem in $O^*(4^n)$ time and polynomial space for a given graph $G = (V, E)$.*

In terms of approximation algorithm, several approximation algorithms have been proposed for MLA and DFAS problem. The first nontrivial approximation algorithm for MLA and DFAS was given by Leighton and Rao [61]. Their approximation factor is $O(\log^2 n)$ in the unweighted case. Leighton and Rao achieve this bound by using an $O(\log n)$-approximation algorithm for the bisection directed separator, which, in turn, is found by approximating special cuts. Using the technique called *spreading metrics technique* Even *et al.* [35] gave an $O(\log n \log\log n)$-approximation algorithm for MLA. Later, Rao and Richa [1998] gave $O(\log n)$-approximation algorithm for general graphs, and $O(\log\log n)$-approximation algorithm for planar graphs. Both results appear in Rao and Richa [77], they also use the spreading metric technique. Their technique works for the general case of graphs with weighted edges. The latest results, obtained through semidefinite relaxations, and combining techniques from the previous results, have been obtained by Charikar *et al.* [14], and Feige and Lee [37].

Theorem 11. *[14, 37] Given a graph $G = (V, E)$, $|V| = n$, with non-negative edge weights $w : E \to \mathbb{R}$, there exists a polynomial time algorithm which can approximate* MLA *to within a factor of $O(\sqrt{\log n}\log\log n)$ of the optimum.*

Theorem 12. *[34] Given a digraph $D = (V, A)$, $|V| = n$, with non-negative edge weights $w : A \to \mathbb{R}$, there exists a polynomial time algorithm which finds* DFAS *and* DFVS *of weight $O(\tau^* \ln n \log\log n)$ where τ^* denotes the value of the optimum fractional solution of the problem at hand.*

It is not known, whether the MLA can be approximated within a constant factor in polynomial time. It was proved in [43] that a polynomial-time approximation scheme for dense graphs exists with running time $O(n^{1/\epsilon})$. Unfortunately, this result cannot be generalized. It was shown in [5] that no polynomial-time approximation scheme exists for an arbitrary input graph with the standard assumption that $\mathcal{NP}$-complete problems cannot be solved in randomized sub-exponential time.

Theorem 13. *[5] Let $\epsilon > 0$ be an arbitrarily small constant. If there is a $\mathcal{PTAS}$ for* MLA *problem, then* SAT *can be decided with high probability by a randomized algorithm that runs in time $O(2^{n^\epsilon})$ where n is the instance size.*

2.4 Bipartite Ordering Problem

This section is devoted to introduce relatively new vertex ordering problem. This problem which we name Bipartite Ordering, was first defined to model a pathway between two RNA structures. In addition to its application in computational biology, we find it useful and interesting to model job scheduling and product management problems. Not many algorithmic results are known for this problem. First, we give a precise definition for the problem. After that, we motivate the problem by connecting it to job scheduling and product management problems. Finally, we mention few known relevant results.

2.4.1 Problem Definition

Bipartite Ordering problem is defined on bipartite graphs. A vertex ordering on bipartite graph $H = (B, S, E)$ is a bijection $\pi : B \cup S \to \{1, 2, \ldots, |B \cup S|\}$. For a vertex ordering π and $v \in B \cup S$, we denote by $\pi_{\preceq,v}$ the set of vertices that appear before v in the ordering. More precisely, $\pi_{\preceq,v} = \{u \in B \cup S : \pi(u) \leq \pi(v)\}$. An ordering π is called *valid* if for a vertex $v \in S$ all its neighbors appear before v that is for each edge $e = (u, v) \in E$ where $u \in B$ and $v \in S$, we have $\pi(u) < \pi(v)$. The cost function for Bipartite Ordering Problem is defined as follows:

$$bg(\pi, H) = \min_{v \in B \cup S} f(H, \pi_{\preceq,v}),$$

where π is a valid ordering and f is a polynomial time computable function that maps each couple consisting of a bipartite graph $H = (B, S, E)$ and a vertex set $Q \subseteq B \cup S$ to an integer as follows:

$$f(H, Q) = |Q \cap S| - |Q \cap B|.$$

The decision version of Bipartite Ordering Problem is

Bipartite Ordering Problem(H, k)
Input: A bipartite graph $H = (B \cup S, E)$ and an integer k.
Question: Is there a valid ordering $\pi \in \Pi(H)$ on vertices of H such that $k + bg(\pi, H) \geq 0$?

In Weighted Bipartite Ordering Problem, we associate a weight or *price* p_i to each vertex $v_i \in B \cup S$. Accordingly set $f(H, Q) = \sum_{y \in Q \cap S} p_y - \sum_{x \in Q \cap B} p_x$.

Consider Figure 2.4. For the given bipartite graph H a valid ordering π is given. The ordering is valid since s_1 appears after b_1 and b_2 plus, s_2 and s_3 appear after b_1, b_2, b_3. Note that $f(H, \pi_{\preceq, s_1}) = -1$, $f(H, \pi_{\preceq, b_3}) = -2$ and $f(H, \pi, \preceq_{s_3}) = 0$. Observe that $bg(\pi, H) = -2$.

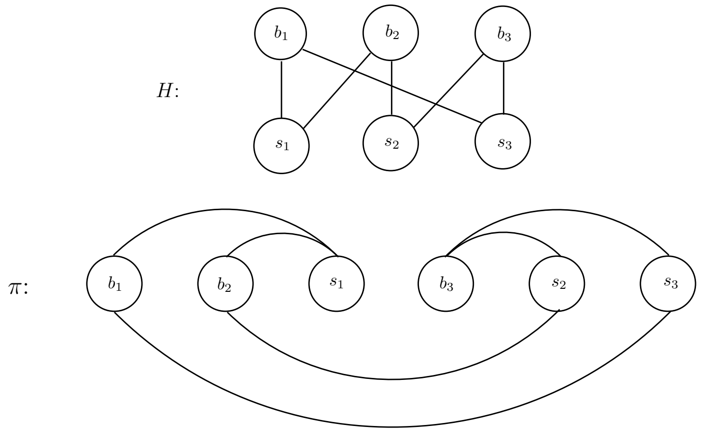


Figure 2.4: An example for Bipartite Ordering Problem problem.

2.4.2 Motivation and Application

The setting of *job scheduling with precedence constraints* is a natural problem that has been extensively studied (see, e.g., [21, 72]). A number of variations of the problem have been studied; we begin by stating one. The problem is formulated as a directed acyclic graph where the vertices are jobs and arcs between the vertices impose precedence constraints. Job j must be executed after job i is completed if there is an arc from j to i. Each job i has a weight w_i and processing time t_i. A given ordering of executing the jobs results in a completion time C_i for each job. Previous work has focused on minimizing the weighted completion time $\sum_{i=1}^{n} w_i C_i$. This can be done in the single-processor or multi-processor setting, and can be considered in settings where the precedence graph is from a restricted graph class. The general problem of finding an ordering that respects the precedence constraints and minimizes the weighted completion time is NP-complete. Both approximation algorithms and hardness of approximation results are known [4, 5, 72, 90].

In Weighted Bipartite Ordering Problem, we consider a different objective than previous works. In our setting, each job j has a net profit (positive or negative) p_j. Our focus is on the *budget* required to realize a given ordering or schedule, and we disregard the processing time. We imagine that the jobs are divided between those with negative p_i, jobs in B that must be *bought*, and jobs with a non-negative p_i, jobs in S that are *sold*. Set

B could consist of raw inputs that must be purchased in bulk in order to produce goods in S that can be sold. A directed graph $H = (B, S, E)$ encodes the precedence constraints inherent in the production: an arc from $j \in S$ to $i \in B$ implies that item i must be bought before item j can be produced and sold. At each step $1 \leq r \leq n$ of the process, let $j_1, j_2, ..., j_r$ be the jobs processed thus far, and let $bg_r = \sum_{i=1}^{r} p_{j_i}$ be the total budget up to this point. Our goal is an ordering that respects the precedence constraints and keeps the minimal value of bg_r as high as possible. One can view (the absolute value of) this optimal value as the *capital* investment required to realize the production schedule. There has been considerable study of scheduling with precedence constraints, but to our knowledge there has not been any work on the objective function we propose (budget minimization).

The Bipartite Ordering Problem is a natural variation of scheduling with precedence constraints problems. As described above the problem can be used to model the purchase of supplies and production of goods when purchasing in bulk. Another way to view the problem is that the items in B are training sessions that employees must complete before employees (vertices in S) can begin to work.

We began studying the problem as a generalization of an optimization problem in molecular folding. The folding problem asks for the energy required for secondary RNA structures to be transformed from a given initial folding configuration $\mathcal{C}_1$ into a given final folding configuration $\mathcal{C}_2$ [47, 73, 86]. The bipartite graph ordering problem models this situation as follows: vertices in B are base pairs that are to be removed from $\mathcal{C}_1$, vertices in S are base pairs that are to be added, and an edge from j to i indicates that base pair i must be removed before base pair j can be added. The price p_i of a vertex is set according to the net energy that would result from allowing the given base pair to occur, with base pairs that must be broken requiring a positive energy and base pairs that are to be added given a negative energy. The goal is to determine a sequence of transformations that respects these constraints and still keeps the net energy throughout at a minimum [1]. Figure 2.5 shows how an instance of the RNA folding problem is transformed into the bipartite graph ordering problem.

2.4.3 Related Results

The Bipartite Ordering problem has been studied only in the setting of unit prices and most attention has been devoted to graph classes corresponding to typical folding patterns (in particular for so-called circle bipartite graphs). Authors in [69] show that the molecular folding problem is in $\mathcal{NPC}$ even when restricted to circle bipartite graphs; thus the bipartite graph ordering problem is $\mathcal{NP}$-complete as well when restricted to *circle bipartite graphs.* A graph G is called a *circle graph* if its vertices are the chords of a circle and two vertices are adjacent if corresponding chords intersect. The circle bipartite graphs can be represented

[1]Note that the molecular folding problem is a minimization problem, and can be made a maximization problem by negating the energies.

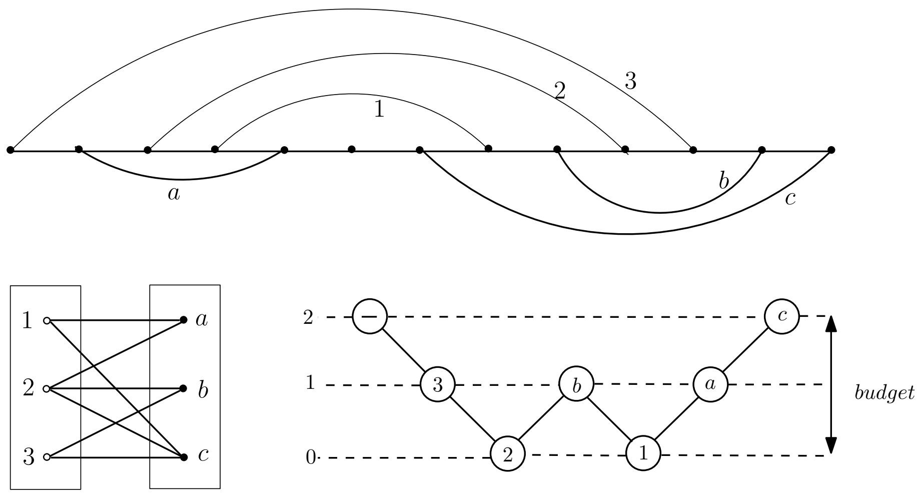


Figure 2.5: The top graph is an instance of the RNA folding problem, with folds 1, 2, and 3 to be removed (bought), folds a, b, and c to be added (sold); an edge cannot be added until edges that cross it are removed. A budget of two is needed and an optimal ordering is 3, 2, b, 1, a, c.

Table 2.1: Vertex Ordering Problems

Problem	Name	Cost
Minimum Linear Arrangement	MLA	$LA(\pi, G) = \sum_{(u,v)\in E} \lvert\pi(u) - \pi(v)\rvert$
Directed Feedback Arc Set	DFAS	$DFAS(\pi, D) = \sum_{v\in V} \sum_{\substack{(v,u)\in A \\ \pi(v)>\pi(u)}} w((v, u))$
Bipartite Ordering Problem	BOP	$bg(H) = \max_{v\in(B\cup S)} f(H, \pi_{\prec,v})$

as two sets A, B where the vertices in A are a set of non-crossing arcs on a real line and the vertices in B are a set of non-crossing arcs from a real line; there is an edge between a vertex in A and a vertex in B if their arcs cross. The top graph in Figure 2.5 is a circle bipartite graph shown with its representation.

Theorem 14. *[69]* Bipartite Ordering Problem *is in class $\mathcal{NPC}$ even for circle bipartite graphs.*

Table 2.1 summarizes the vertex ordering problems we defined and their cost functions.

Chapter 3

Dating Species Tree with Lateral Transfers

In this chapter we consider a variant of DFAS problem. The problem, called *Dating a Species Tree with Lateral Transfers*, briefly DFAST, is of interest because of its application in editing *Phylogenetic Trees* and *Gene Trees*. Here, we first define the problem and discuss motivations and the connection to phylogenetic trees. We present an algorithm for the problem and show that our algorithm is often better than simple greedy approach by implementing both algorithms on different data sets.

3.1 Introduction and Problem Definition

As for DFAS problem, in DFAST problem we are given a digraph D and a weight function that assigns non-negative weights to arcs of D. DFAST asks for a set of arcs with minimum weight whose removal leads to an acyclic digraph. However, in DFAST the input digraph is of special type. An input for DFAST is described as follows. Let $T = (V, A_T)$ be a directed binary tree with root vertex $r \in V(T)$ such that (u, v) is an arc in A_T if and only if v is a child of u. Consider digraph $D = (V, A_T \cup \mathcal{A})$ which has the same vertex set as T and its arc set is the union of tree arcs A_T and non-tree arcs $\mathcal{A}$. The restriction on $\mathcal{A}$ is that $(u, v) \notin \mathcal{A}$ if v is either an ancestor of u or its descendant. Cost function $w : \mathcal{A} \to \mathbb{R}_+$ assigns non-negative weights to the non-tree arcs. Weight of $F \subseteq \mathcal{A}$ is equal to $w(F) = \sum\limits_{(u,v)\in F} w((u, v))$. Task is to find a set of arcs $F \subseteq \mathcal{A}$ with minimum weight so that $D - F$ is acyclic.

DFAST

Input: A graph $D = (V, A_T \cup \mathcal{A})$ and an integer k.

Task: Either find a set $F \subseteq \mathcal{A}$ such that $w(F) \leq k$ and $D - F$ is a DAG or report such a set does not exist.

An instance of the problem is given in Figure 3.1. In the figure, directed binary tree T with root r is depicted in black and non-tree arcs are in blue. Blue arcs have unit weight. Arcs from ancestor to descendant (dashed arc (r, v_8)) and from descendant to ancestor (dashed arc (v_7, v_1) are not permitted in inputs for DFAST problem. $F = \{(v_6, v_1), (v_3, v_2)\}$ is a feedback arc set for the given digraph with $w(F) = 2$.

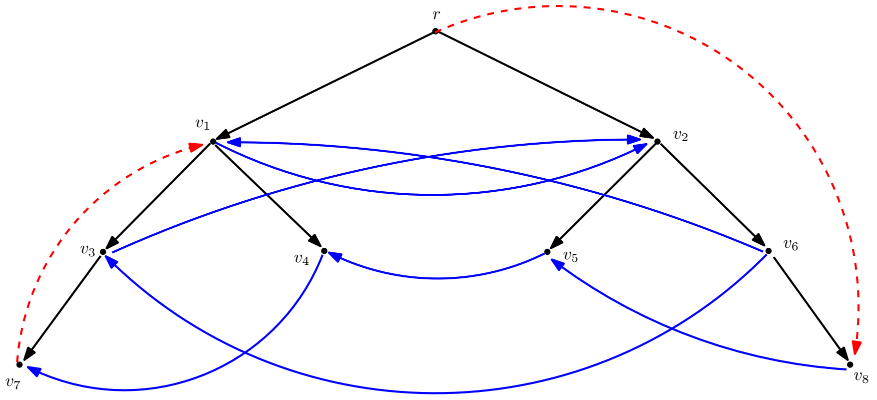


Figure 3.1: An example of an input for DFAST problem.

Lemma 1. DFAST $\in \mathcal{NPC}$.

Proof. Let digraph $D' = (V, A)$ be an arbitrary instance of weighted DFAS problem where $V(D') = \{v_1, v_2, \cdots, v_n\}$. From D' we construct digraph D that is an instance of DFAST. Let $T = (V, A_T)$ be directed binary tree so that

- $V(T) = V(D') \cup \{u_1, u_2, \cdots, u_n\}$ and
- (u_i, v_i) and (u_i, u_{i+1}) are in A_T for $1 \leq i \leq n - 1$
- $(u_n, v_n) \in A_T$

Now $D = (V(T), A_T \cup A(D'))$ is an instance for DFAST. Note that this transformation is polynomial. An example of such a transformation is given in Figure 3.2. It is clear that $F \subseteq A'$ is a FAS for D' of weight k if and only if it is an FAS for D of weight k. □

As discussed in the Introduction chapter, DFAS can be encoded as a vertex ordering problem. In the same way DFAST can be seen as a vertex ordering problem. We say an ordering π on $V(D)$ is *tree-consistent* if $\pi(u) < \pi(v)$ for $(u, v) \in A_T$. In fact, the underlying tree T can be seen as a *partial ordering* on vertices that is $\pi(u) < \pi(v)$ if and only if the

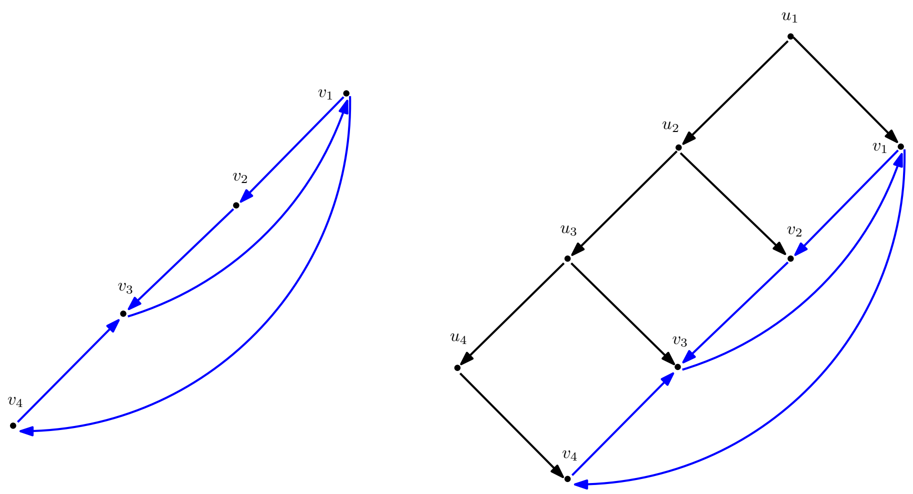


Figure 3.2: An example for the transformation in Lemma 1.

unique directed path from the root to v passes through u. Therefore, the cost function for DFAST is as follows:

$$DFAST(\pi, D) = \sum_{u \in V} \sum_{\substack{(u,v)\in\mathcal{A} \\ \pi(u)>\pi(v)}} w((u,v)).$$

where π is a tree-consistent ordering on $V(D)$.

3.1.1 Biological Background and Motivation

An important and central aspect in history of Earth is the pattern and timing of diversification of the species. However, for macro-organisms such as animals and plants there are abundant fossil records and genomic data that are progressively yielding a comprehensive picture of timing of species diversification [9, 28], the dating of the evolution of microbial life remains largely difficult to picture [53, 58]. That is mainly because there are not sufficient bacterial and archaeal fossils and the existing one cannot be traced to a specific lineage. Moreover, interpreting the patterns of molecular data is difficult. The exchange of genes between distinct species, called *lateral gene transfers*, makes it more difficult to picture the relationships between lineages [84]. Various models and methods have been proposed during years to give a better view of the pattern and timing of species inhabit on Earth [19, 26]. In [84], authors present an approach to construct a dated species tree for a given set of species. However, the outputted species tree still has errors that needs to be edited. To illustrate the point, consider Figure 3.3. It says that Chlamydiae gene and Gram-positive gene are from gene F. Spirochaete gene and Planctomycetes gene are from gene D and Actinobacteria's gene is from gene E. Besides, there are exchanges between genes that are shown in blue. As an example, the blue arc from gene F to E tells that gene E could possibly be created from gene F by a lateral transfer. Therefore, gene F could possibly be an ancestor of gene E. The same for the arc between genes E and C. However, the created cycle between genes C, F and E says that one of the blue arcs is not a correct one. Indeed, the given species tree with lateral transfers needs to be edited in order to have a meaningful dated species tree.

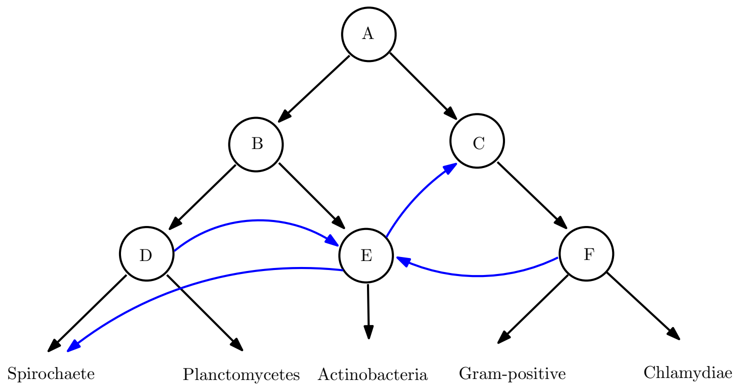


Figure 3.3: An example for dated species tree with lateral transfers (blue arcs).

3.2 Algorithm

In this section we provide an algorithm to the problem. In general picture, our algorithm has two main phases. In the first phase, a greedy method is employed to find an upper bound for a given instance. The given upper bound computed by the greedy algorithm is then used in a branch-and-bound strategy. It turns out that the branch-and-bound strategy can solve instances of moderate size exactly. In the second phase, we extract sub-problems of a reasonable size. By reasonable size, we mean instances that can be solved exactly by the branch-and-bound algorithm. After solving relatively small sub-problems it remains to merge the orderings of sub-problems. By applying a simple dynamic programming approach, given orderings for sub-problems are merged together and yield a total ordering which is tree-consistent.

3.2.1 Phase 1: Greedy Heuristic Branch and Bound

Let digraph $D = (V, A_T \cup \mathcal{A})$ with underlying directed binary tree $T = (V, A_T)$ be an input of the problem. Clearly, for any tree-consistent ordering π we have $\pi(r) = 1$ where $r \in V$ is the root for T. For $S \subset V(D)$, the set of *possible extensions*, denoted by L_S, is define to be the set of vertices whose parents are in S.

$$L_S = \{v \in V(D) : u \in S \text{ where } u \text{ is the parent of } v\}.$$

To illustrate on the above definition, $L_{\{r\}}$ contains children of r. From now on, throughout this section by an ordering we mean a tree-consistent one. For a vertex u let the set of *backward* arcs be $\{(u,v) : (u,v) \in \mathcal{A}, \pi(v) < \pi(u)\}$. In a similar way, let the set of *forward* arcs for u be $\{(u,v) : (u,v) \in \mathcal{A}, \pi(u) < \pi(v)\}$. Observe that if F is the set of all backward arcs from each vertex in ordering π, then $D-F$ is an acyclic digraph. We set the *contribution* of a vertex $v \notin S$ with respect to set S to be:

$$c_S(v) = \sum_{\substack{(v,u)\in\mathcal{A}\\ u\in S}} w((v,u)) - \sum_{\substack{(v,u')\in\mathcal{A}\\ u'\notin S}} w((v,u')).$$

In our greedy approach, we first start by setting $\pi(r) = 1$ and $S = \{r\}$. For our greedy criteria at each step we greedily choose a vertex from L_S with minimum contribution with respect to S, and consider it as the next vertex in the output ordering (also add it to S). As we choose next vertex from L_S, we remove it from L_S and add its children to L_S. Note that as we add or remove vertices from L_S, we need to update the contribution of vertices in L_S with respect to S.

Algorithm 1 Greedy Algorithm for DFAST

```
 1: Input: Digraph $D = (V, T_A \cup \mathcal{A})$ with underlying directed binary tree $T = (V, A_T)$
    with root $r$
 2: Output: A tree-consistent ordering $\pi$ on $V(D)$ and directed feedback arc set $F$
 3: $F := \emptyset$
 4: $\pi(r) = 1$ and $S = \{r\}$
 5: Add children of $r$ to list $L_S$.
 6: for i=2 to n do
 7:    Let $v$ be the vertex of $L_S$ with minimum contribution with respect to $S$
 8:    $\pi(i) = v$ and $S \leftarrow S \cup \{v\}$
 9:    Add children of $v$ to $L_S$
10:    Remove $v$ from $L_S$
11:    Add backward arcs from $v$ to $F$
12:    Update contribution of vertices in $L_S$ with respect to $S$
13: return $\pi$ and $F$
```

Lemma 2. *Set F returned by Algorithm 1 is a directed feedback arc set for D and π is a tree-consistent ordering on $V(D)$.*

Proof. By definition, in any tree-consistent ordering π the root r has to appear first in the ordering, i.e., $\pi(r) = 1$. Note that in Steps 5 and 12, vertices are added to list of possible extensions L_S, if their parents are already considered in S. In other words, Steps 5 and 12 guarantee that each vertex has higher rank in π than its parent. Therefore, an output ordering by Algorithm 1 is a tree-consistent one. Plus, for any ordering if we remove all the backward arcs of the vertices the remaining digraph $(D - F)$ is acyclic. Step 12 adds all the backward arcs to F that guarantees F is a feedback arc set for D. $\square$

Let us to discuss another straightforward greedy approach. First, sort non-tree arcs in an increasing order with respect to their weights. At each iteration, consider the non-tree arc with minimum weight. If that arc is part of a cycle then delete it and add it to F. Repeat this process for all non-tree arcs in the digraph. The pseudocode is given in Algorithm 2.

Algorithm 2 Greedy Algorithm for DFAST

```
1: Input: Digraph $D = (V, T_A \cup \mathcal{A})$ with underlying directed binary tree $T = (V, A_T)$
   with root $r$ and $m$ non-tree arcs
2: Output: A directed feedback arc set $F$
3: $F := \emptyset$
4: Let $L$ be the sorted list of non-tree arcs in an increasing order according to their weights

5: for $i = 1$ to $m$ do
6:   if the $i$th arc in $L$ is a part of a cycle in $D$ then
7:     Delete it from $D$ and add it to $F$
8: return $F$
```

We are ready to propose our branch and bound algorithm. The criteria for us to bound on different branches of the algorithm is follows. Suppose at some point of the algorithm we have $S \subset V(D)$ plus set of possible extensions L_S. Vertices in L_S are sorted in increasing order according to their contribution with respect to S. Adding the first vertex from L_S to stack S increases the cost of the ordering. If the new cost is more than the cut off, we stop branching and consider the second vertex in L_S as our next candidate. But if the new cost is less than the cut off, we add that vertex to S and update L_S accordingly. We continue the process till we have all the vertices added to S. The pseudocode is given in Algorithm 3.

Note that associate to a feedback arc set there might be different orderings. As an example, consider the digraph in Figure 3.1. $F = \{(v_6, v_1), (v_3, v_2)\}$ is a feedback arc set for the digraph. However, there are more than one ordering for which F is the set of backward arcs. Two orderings $\{r, v_1, v_2, v_6, v_3, v_8, v_5, v_4, v_7\}$ and $\{r, v_1, v_2, v_6, v_8, v_5, v_4, v_3, v_7\}$ have the same set of backward arcs. A weakness of Algorithm 3 is that it tries all the orderings with the same set of backward arcs which can possibly slows down the algorithm. Indeed, it is the reason why Algorithm 3 does not terminate for some instances. One way to overcome this issue is to set a bound for number of iterations on the loop at Step 8 in the algorithm. Such a bound can be part of input. In our simulations we consider such a parameter (Table 3.1).

Algorithm 3 Branch-and-Bound Algorithm for DFAST

```
1: Input: Digraph $D = (V, T_A \cup \mathcal{A})$ with underlying directed binary tree $T = (V, A_T)$ rooted at $r$.
2: Output: A tree-consistent ordering $\pi$ on $V(D)$ and directed feedback arc set $F$.
3: Let $Cost$ be the cost of set $F$ returned by Algorithm 1
4: Stack $S = \emptyset$, Score=0
5: $S.push\_back(r)$
6: Add children of $r$ to list $L_S$
7: Sort vertices in $L_S$ in increasing order according to their contribution with respect to $S$
8: while $S \neq \emptyset$ do
9:   if $S.top()$ has no possible extension on list $L_S$ then
10:    if $|S| = |V(T)|$ then
11:      if $Score < Cost$ then
12:        $Cost = Score$
13:      return the ordering on elements of $S$
14:    $v = S.top()$ and $S.pop()$
15:    $Score = Score - \sum_{u \in S} w((v, u))$
16:    Remove children of $v$ from $L_S$
17:    Update contribution of vertices in $L_S$ with respect to $S$
18:    Sort vertices in $L_S$ in increasing order according to their contribution with respect to $S$
19:  else
20:    Let $x$ be the next extension for $S.top()$ on list $L_S$
21:    if $Score + \sum_{v \in V(T) \setminus S} w((v, x)) \leq Cost$ then
22:      $S.push\_back(x)$
23:      Add children of $x$ to $L_S$
24:      Sort vertices in $L_S$ in increasing order according to their contribution with respect to $S$
25:      $Score = Score + \sum_{v \in V(T) \setminus S} w(v, x)$
26:    else
27:      Go to the next extension for $S.top()$ on list $L_S$
28: return No solution better than greedy one exists
```

3.2.2 Phase 2: Dynamic Programming

In the first phase we gave an algorithm that, according to our experimental results, is efficient for digraphs of moderate size. Here, by taking advantage of the structure of the underlying directed binary tree, we apply a dynamic programming approach for instances of relatively larger size. Let T_v denote the subtree hanging out from v in T that is set of all vertices that are reachable from v by repeatedly proceeding from parent to child. Besides, let D_v be the induced digraph by vertices in T_v and let $|D_v|$ be the number of its vertices. Moreover, for a vertex v, let v_l denote its left child and v_r denote its right child.

Intuitively, the dynamic programming algorithm proceeds as follows. It starts at the root r. At each step for a vertex v, if $|D_v| \leq k$ (k is a parameter that is part of an input) then it calls Algorithm 3 for D_r, else, it branches on D_{v_l} and D_{v_r}. It remains to have a merging procedure for two given orderings.

Assume that π_x and π_y are two orderings for D_x and D_y, respectively. In order to merge these two orderings we proceed in a dynamic programming manner as follows. First, the resulted ordering π is consistent with π_x and π_y that is if $\pi_x(u) < \pi_x(v)$ or $\pi_y(u) < \pi_y(v)$ then $\pi(u) < \pi(v)$. Second, we consider a dynamic programming table M so that $M[i][j]$ represents the cost associated to the optimal merging of $\pi_x(1), \ldots, \pi_x(i)$ and $\pi_y(1), \ldots, \pi_y(j)$. In addition, to compute $M[i][j]$ we use the formula in Line 10 of Algorithm 4 which basically is saying $M[i][j]$ is either $M[i][j-1]$ plus the cost of adding $\pi_y(j)$ as the next vertex in π or $M[i-1][j]$ plus the cost of adding $\pi_x(i)$ as the next vertex in π. The following algorithm gives the pseudocode of our merging strategy.

Algorithm 4 Merging Two Orderings

```
1: **Input:** Ordering $\pi_x$ of $V(D_x)$ and Ordering $\pi_y$ of $V(D_y)$;
2: **Output:** A merged ordering of $\pi_x$ and $\pi_y$;
3: Let $M$ be a $|V(D_x)+1| \times |V(D_y)+1|$ matrix;
4: for $i = 1$ to $|V(D_y)+1|$ do
5:     $M[0][i]$ = total weight of incoming arcs to $\pi_y(i)$ from $V(D_x)$ and $\pi_y(j)$ for $i < j \leq |V(D_y)|$
6: for $i = 1$ to $|V(D_x)+1|$ do
7:     $M[i][0]$ = total weight of incoming arcs to $\pi_x(i)$ from $V(D_y)$ and $\pi_x(j)$ for $i < j \leq |V(D_x)|$
8: for $i = 1$ to $|V(D_x)+1|$ do
9:     for $j = 1$ to $|V(D_y)+1|$ do
10:
```

$$M[i][j] = \min \begin{cases} M[i-1][j] + \sum_{i'=i+1}^{|V(D_x)|} w((\pi_x(i'), \pi_x(i))) + \sum_{j'=j}^{|V(D_y)|} w((\pi_y(j'), \pi_x(i))) \\ M[i][j-1] + \sum_{i'=i}^{|V(D_x)|} w((\pi_x(i'), \pi_y(j))) + \sum_{j'=j+1}^{|V(D_y)|} w((\pi_y(j'), \pi_y(j))) \\ \end{cases}$$

Having the exact algorithm which is efficient for small instances and the dynamic programming one, gives us enough ingredients to describe our final algorithm.

Algorithm 5 Solving DFAST

```
**Input:** Directed Graph $D = (V, A_T \cup \mathcal{A})$ and integer $K$;
**Output:** Ordering of vertices in $D$;
if Size of $D$ is less than $K$ then
    return Branch-and-Bound($D$)
else
    Let $r$ be the root of $T$
    $\pi$ = Merging-Two-Ordering(Solving DFAST($D_{r_l}$),Solving DFAST($D_{r_r}$))
    Append $r$ to the beginning of $\pi$
    return $\pi$
```

3.3 Experimental Results

We used software package SimPhy [68] to generate a random dated species tree with 500 species as its leaves. From that species tree, we generate dated lateral transfer species trees, using Amalgamated likelihood estimation (ALE), that is a probabilistic approach to exhaustively explore all reconciled species trees that can be amalgamated as a combination of clades observed in a sample of gene trees [85]. Now, we have an instance for DFAST problem. We randomly delete some species (leaves) from the tree. Furthermore, we preprocess the data by removing subtrees that have no incoming and outgoing non-tree arcs.

The results provided in Table 3.1 show that our algorithm often provides better results than the Greedy Algorithm. The last tow columns in Table 3.1 show the weight of feedback arc set found by the Greedy Algorithm and Algorithm 5. However, a weakness of our algorithm is that, in the branch-and-bound procedure (Algorithm 3) where we spend a lot of time trying on orderings with the same cost for small subtrees.

Size of input	cut size	number of iterations	number of non-tree arcs	Greedy Algorithm	Algorithm 5
41	100	10^9	9	0	0
50	100	10^7	19	0	0
69	100	10^7	73	1.47	1.47
100	100	10^7	333	2.54	2.54
100	25	10^9	333	2.54	2.54
100	2	10	333	2.54	2.54
98	100	10^7	929	8.91	8.91
98	25	10^7	929	8.91	8.91
98	2	10	929	8.91	8.91
100	100	10^7	1673	59.36	59.36
100	25	10^7	1673	59.36	54.98
100	2	10	1673	59.36	54.98
86	100	10^7	3514	410.08	410.08
86	25	10^9	3514	410.08	403.38
86	25	10^7	3514	410.08	403.08
86	2	10	3514	410.08	401.82
106	100	10^7	5546	488.68	479.78
106	25	10^9	5546	488.68	479.61
106	25	10^7	5546	488.68	479.61
106	2	10	5546	488.68	479.50
101	100	10^7	5838	632.43	603.32
101	25	10^9	5838	632.43	596.15
101	25	10^7	5838	632.43	596.15
101	2	10	5838	632.43	596.15
112	100	10^7	8845	1063.56	1055.46
112	25	10^9	8845	1063.56	1049.07
112	25	10^7	8845	1063.56	1049.07
112	2	10	8845	1063.56	1053.66

Table 3.1: Experimental results on random species tree with various number of non-tree arcs (lateral transfers). The last two columns show the weight of the feedback arc set found by the Greedy Algorithm and Algorithm 5, respectively.

Chapter 4

Minimum Linear Arrangement Parametrized by Vertex Cover

As it was mentioned in the introduction chapter, vertex ordering problems are mostly intractable in terms of parametrized complexity. To deal with their difficulties, different parameters have been considered in literature [39, 40]. A powerful parameter is vertex cover number of a graph. In [40], the authors investigated parameterized complexity of several vertex ordering problems considering vertex cover number as the parameter.

Without going into details we give a general overview of their approach. For a given graph $G = (V, E)$, let its vertex cover be of size k, i.e., $|vc(G)| = k$. Let $I = V \setminus vc(G)$, note that I is an independent set. We associate a *type* with each vertex in I.

Definition 4. *[40] The type of a vertex $v \in I$ is $N(v)$. For a set $S \subseteq vc(G)$ the set $I(S)$ is the set of all vertices in I of type S and $M_S = |I(S)|$.*

We are looking for an ordering $\pi \in \Pi(G)$ that minimizes certain objective function. In order to do that, we loop over all possible permutations of vertices in $vc(G)$. Note that we have $k!$ possible ordering on $vc(G)$. According to each fixed ordering on $vc(G)$ we find the best possible ordering on vertices in I. From now on, we concentrate on finding an optimal ordering on vertices in I while assuming a fixed order on vertices in $vc(G)$. Let us fix an ordering on vertices in $vc(G)$. Let $vc = \{c_1, c_2, \cdots, c_k\}$ and c_i appears before c_j for $1 \leq i < j \leq k$. From now on, we assume that an ordering $\pi \in \Pi(G)$ is consistent with ordering on vertices in $vc(G)$ i.e. $\pi(c_i) < \pi(c_j)$ for $1 \leq i < j \leq k$. We say a vertex $v \in I$ is in *gap* $1 \leq i \leq k-1$ if $\pi(c_i) < \pi(v) < \pi(c_{i+1})$. Similarly, v is in gap 0 if $\pi(v) < \pi(c_1)$ and it is in gap k if $\pi(c_k) < \pi(v)$.

Note that each ordering consists of distribution of vertices of different types inside gaps. There are two main questions we encounter. First, how many vertices of certain type S are in any of the $k + 1$ gaps. Second, for an optimal ordering, in what way vertices of different types are ordered inside gaps? Hence, we reduce the original vertex ordering problem to a problem in which we are seeking for an optimal distribution of vertices in I insides gaps. Here is where the power of vertex cover as a parameter comes to play. The structural properties of vertex cover leads us to answer the second question and extract helpful properties of an optimal ordering inside each gap. This useful observation leads us to formulate some vertex ordering problems as an *Integer Programming.*

Depending on the cost function for a specific vertex ordering problem, we might end up with an *Integer Linear Programming.* In such a case the following well known result can be applied to achieve an $\mathcal{FPT}$ algorithm. Consider the following Integer Linear Program.

$$\begin{aligned} &\text{Minimize: } f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} \\ &\text{Subject to: } \mathbf{A}\mathbf{x} \geq \mathbf{b} \end{aligned} \tag{4.1}$$

Where $\mathbf{A} \in \mathbb{Z}^{m\times d}$, $\mathbf{b} \in \mathbb{Z}^{\mathbf{m}}$ and $\mathbf{x}, \mathbf{w} \in \mathbb{Z}^d$. Assuming the dimension d (number of variables) is fixed, next theorem states that there is an algorithm that solves ILP (4.1) in $\mathcal{FPT}$ time.

Theorem 15. *([54, 63, 42]) ILP (4.1) can be solved using $O(d^{2.5d+o(d)} \cdot L \cdot \log(MN))$ arithmetic operations and space polynomial in L. Here, L is the number of bits to encode input, N is the maximum of the absolute values any variable can take, and M is an upper bound on the absolute value of the minimum taken by the objective function.*

However, it might be the case that for some vertex ordering problems the objective function is not linear. The MLA problem is an example of such problems for which the objective function (parameterized by the vertex cover number) is *quadratic.* To best of our knowledge there is no such result as Theorem 15 for *Quadratic Integer Programming.* However, interesting techniques and tools have been developed to efficiently solve Quadratic Integer Programs in particular cases [23, 60, 76].

The rest of this chapter is organized as follows. We formulate MLA as a Quadratic Integer Program while it is parameterized by the vertex cover number. Next, we will define *Graver Basis* that is a powerful tool in discrete optimization used in Integer Programming. Some related results and applications of Graver Basis are discussed. We conclude this chapter by proposing several interesting conjectures.

4.1 MLA Parameterized by Vertex Cover

Given a graph $G = (V, E)$ with vertex cover of size k, we aim to find an optimal ordering for MLA on G in time $O(f(k)poly(n))$ where $f(k)$ is a function depending only on k. Without loss of generality, we assume that G contains no isolated vertex (vertex with degree 0). For vertex $v \in I$ we define $\delta(v)$, with respect to π, to be:

$$\delta(v) = |\{x \in N(v) : \pi(v) < \pi(x)\}| - |\{x \in N(v) : \pi(x) < \pi(v)\}|$$

The following lemma tells us how vertices of different types are ordered within a gap. The same lemma was proved independently in [39] as well.

Lemma 3. *For each gap i, $0 \leq i \leq k$, placing vertices in I from left to right in a non-decreasing order by $\delta(v)$ gives an optimal ordering within that gap.*

Proof. Let π be an optimal arrangement and $u, v \in I$ be vertices of different types that appeared in the same gap. We claim that if $\delta(v) < \delta(u)$ and $\pi(u) < \pi(v)$ then we can swap u, v and obtain an arrangement with a smaller cost. Let S be the set of vertices such that $x \in S$ if $\pi(u) < \pi(x) < \pi(v)$.

The change of the cost by swapping u, v is:

$$\delta(v)|S| - \delta(u)|S| < 0$$

Note that δ is equal for the vertices of the same type in gap i. Therefore, we can place all of the vertices of the same type as a contiguous block in gap i without harming the optimality of the arrangement.

□

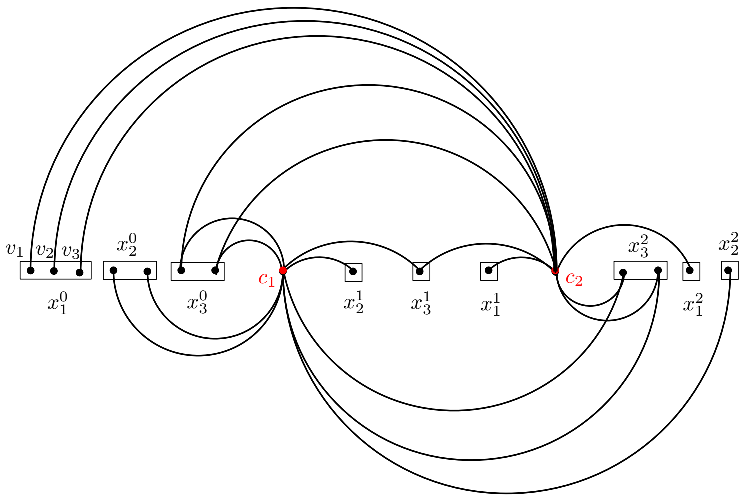


Figure 4.1: An example for MLA parameterized by vertex cover $\{c_1, c_2\}$.

An example is given in Figure 4.1, where there are two vertices in vertex cover of the graph, $\{c_1, c_2\}$. Accordingly, there are three gaps and three types of vertices in $I = V \backslash vc(G)$. Type "1" are vertices adjacent to c_2 ($M_1 = 5$), type "2" are those which are adjacent to c_1 ($M_2 = 4$), and type "3" are adjacent to both c_1 and c_2 ($M_3 = 5$). As it is depicted, vertices of different types are distributed in three gaps. For instance, vertices of type "3" are distributed as follows. There are two of them in gap "0" ($x_3^0 = 2$), there is one in gap "1" ($x_3^1 = 1$) and there are two in gap "2" ($x_3^2 = 2$).

Notice that for a given graph with two vertices in its vertex cover, we have nine variables that are $x_1^0, x_1^1, x_1^2, \cdots, x_3^1, x_3^2$. Each variable stands for number of vertices of certain type in a gap. For example, x_3^2 represents the number of vertices of type "3" in gap number "2". $x_3^2 = \alpha$ means that there α vertices of type "3" in gap number "2".

Followed by Lemma 3, the ordering of types inside each gap is fixed regardless of number of their vertices. In the case of vertex cover of size two, in gap "1" vertices of type "3" always appear after vertices of type "2" and before vertices of type "1" i.e. in Figure 4.1, x_3^1 is after x_2^1 and before x_1^1. Observe that MLA problem is equivalent to find an optimal distribution of vertices of different types inside gaps.

4.1.1 Quadratic Integer Programming Formulation for MLA

To illustrate how the objective function is quadratic for MLA, we continue our explanations using Figure 4.1. Consider the edge (v_3, c_2). It spans over all the vertices that are represented by $x_2^0, x_3^0, x_2^1, x_3^1, x_1^1$. Plus, it spans the vertex c_1 as well. In other words,

i) for edge (v_3, c_2) we have $\pi(c_2) - \pi(v_1) = x_2^0 + x_3^0 + x_2^1 + x_3^1 + x_1^1 + 1$

ii) for edge (v_2, c_2) we have $\pi(c_2) - \pi(v_1) = x_2^0 + x_3^0 + x_2^1 + x_3^1 + x_1^1 + 1 + 1$

iii) for edge (v_1, c_2) we have $\pi(c_2) - \pi(v_1) = x_2^0 + x_3^0 + x_2^1 + x_3^1 + x_1^1 + 1 + 2$.

One can see that the contribution of all vertices represented x_1^0 to the cost of an ordering is $x_1^0(x_2^0 + x_3^0 + x_2^1 + x_3^1 + x_1^1) + \frac{(x_1^0-1)(x_1^0)}{2} + x_1^0$.

Let $\mathbf{x}^T = \begin{pmatrix} x_1^0 & x_1^1 & x_1^2 & x_2^0 & x_2^1 & x_2^2 & x_3^0 & x_3^1 & x_3^2 \end{pmatrix}$ be the vector of variables and $\mathbf{V}$ be a 9×9 symmetric matrix. Matrix $\mathbf{V}$ is defined to be the coefficient matrix for our objective function. For instance the first row of $\mathbf{V}$ represents the coefficient of x_1^0 in the objective function. That is the coefficient of $(x_1^0)^2$ is $\mathbf{V}_{[1,1]} = \frac{1}{2}$, coefficient of $x_1^0 \cdot x_1^1$ is $\mathbf{V}_{[1,2]} = 1$, coefficient of $x_1^0 \cdot x_2^0$ is $\mathbf{V}_{[1,4]}$ and so on.

$$\mathbf{V} = \begin{array}{c|ccccccccc|} & x_1^0 & x_1^1 & x_1^2 & x_2^0 & x_2^1 & x_2^2 & x_3^0 & x_3^1 & x_3^2 \\ x_1^0 & \frac{1}{2} & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ x_1^1 & 1 & \frac{1}{2} & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ x_1^2 & 0 & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0 & 1 \\ x_2^0 & 1 & 0 & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0 \\ x_2^1 & 1 & 0 & 0 & 0 & \frac{1}{2} & 1 & 1 & 1 & 1 \\ x_2^2 & 0 & 1 & 1 & 0 & 1 & \frac{1}{2} & 0 & 1 & 1 \\ x_3^0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ x_3^1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ x_3^2 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}$$

Notice that besides the quadratic terms, there are linear terms as well. In $x_1^0(x_2^0 + x_3^0 + x_2^1 + x_3^1 + x_1^1) + \frac{(x_1^0-1)(x_1^0)}{2} + x_1^0$, we have linear term x_1^0. It shows that c_1 is between two end points of any edge that is adjacent to c_2 and a vertex in x_1^0. For linear terms we consider vector $\mathbf{w}$ that is a 9×1 integer vector.

$$\mathbf{w} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

In addition, from terms $\frac{(x_i^j - 1)(x_i^j)}{2}$ for $1 \leq i \leq 3$ and $0 \leq j \leq 2$, we have the linear term $-\frac{x_i^j}{2}$. We abuse the notation and write $-\frac{\mathbf{1}}{\mathbf{2}}$ for such a 9×1 vector. In total the linear part of the objective function is $\mathbf{w} - \frac{\mathbf{1}}{\mathbf{2}}$.

Next, we describe constrains. First constrain is that for type S in gap number "i" we have $x_S^i \geq 0$. Second, for type S, $\sum_{i=0}^{2} x_S^i = M_S$. Hence, vector $\mathbf{x}$ must satisfy the constrain $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{b} = \begin{pmatrix} M_1 & M_2 & M_3 \end{pmatrix}$ and $\mathbf{A}$ is as follows.

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Therefore, for vector $\mathbf{x}^T = \begin{pmatrix} x_1^0 & x_1^1 & x_1^2 & x_2^0 & x_2^1 & x_2^2 & x_3^0 & x_3^1 & x_3^2 \end{pmatrix}$ the quadratic program is

$$\text{Minimize: } f(\mathbf{x}) = \mathbf{x}^T \mathbf{V} \mathbf{x} + \mathbf{x}^T (\mathbf{w} - \frac{\mathbf{1}}{\mathbf{2}})$$
$$\text{Subject to: } \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$$

In general, when size of vertex cover is k, number of variables is $(k+1)(2^k - 1)$ and matrix $\mathbf{V}$ is a symmetric $(k+1)(2^k-1) \times (k+1)(2^k-1)$ matrix. Moreover, $\mathbf{w}$ and $\frac{\mathbf{1}}{\mathbf{2}}$ are $(k+1)(2^k-1) \times 1$ matrices. To move forward and show how the entries of matrices $\mathbf{V}$ and $\mathbf{w}$ look like in general case, we need to introduce some notation. For type S in gap "i", let L_S^i (R_S^i) denote number of vertices in vertex cover that have rank less (grater) than or equal to i. More precisely,

$$L_S^i = |\{c_j \in vc(G) : \pi(c_j) \leq i\}|$$
$$R_S^i = |\{c_j \in vc(G) : \pi(c_j) \geq i\}|.$$

For x_i^j and $x_{i'}^{j'}$ and $j < j'$ the coefficient of term $x_i^j \cdot x_{i'}^{j'}$ in the objective function is

$$\mathbf{V}_{[x_i^j, x_{i'}^{j'}]} = L_{i'}^j + R_i^{j'+1}$$

and for x_i^j and $x_{i'}^j$ when x_i^j appears before $x_{i'}^j$ we have

$$\mathbf{V}_{[x_i^j, x_{i'}^j]} = L_{i'}^j + R_i^{j+1}$$

plus, the entries on diagonal are

$$\mathbf{V}_{[x_i^j, x_i^j]} = \frac{L_i^j + R_i^{j+1}}{2}$$

Note that an upper bound of entries of $\mathbf{V}$ is $k-1$. Let $\mathbf{w}$ be a vector of length $(2^k - 1)(k+1)$ such that, for x_i^j, $\mathbf{w}^{(x_i^j)}$ denotes the number of vertices in $vc(G)$ that are spanned by edges incident to vertices in x_i^j, plus number of $(c_{j'}, c_{j''})$ edges so that $j' \leq j \leq j''$. The following quadratic integer program encodes MLA of graph G for a fixed ordering on $vc(G)$.

$$\begin{aligned} &\text{Minimize: } f(\mathbf{x}) = \mathbf{x}^T\mathbf{V}\mathbf{x} + \mathbf{x}^T(\mathbf{w} - \frac{\mathbf{1}}{\mathbf{2}}) \\ &\text{Subject to: } \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{4.2}$$

where $\mathbf{x}$ is a vector in $\mathbb{Z}_+^{(k+1)(2^k-1)}$ and $\mathbf{b}$ is a vector in $\mathbb{Z}_+^{2^k-1}$ and $\mathbf{b}^{(i)} = n_i$ denotes number of vertices of type i in I. The constrains matrix A is a $(2^k-1) \times ((k+1)(2^k-1))$ binary matrix defined as follows:

$$\mathbf{A} = \begin{pmatrix} \overbrace{1\cdots1}^{k+1} & 0\cdots0 & \cdots & 0\cdots0 \\ 0\cdots0 & \overbrace{1\cdots1}^{k+1} & \cdots & 0\cdots0 \\ \vdots & \vdots & \vdots & \vdots \\ 0\cdots0 & 0\cdots0 & \cdots & \overbrace{1\cdots1}^{k+1} \end{pmatrix}$$

Notice that each row of $\mathbf{A}$ consists of a consecutive block of 1s and the rest of its entries are 0s [1]. To be more precise, the ith row of $\mathbf{A}$, denoted by $\mathbf{A}_i$, is

$$\mathbf{A}_i = \begin{pmatrix} \overbrace{0\cdots0}^{(k+1)(i-1)} & \overbrace{1\cdots1}^{k+1} & \overbrace{0\cdots0}^{(k+1)(2^k-i-1)} \end{pmatrix}$$

4.2 Graver Basis, Definitions and Applications

Here we focus on solving special types of ILP and QIP. As it was mentioned before, a well known result due to Lanstra [63] says that ILP is tractable when the dimension (number of variables) is fixed. Since this result has found much use in parameterized complexity, it is natural to ask if there is another property of ILP besides fixed dimension which makes it tractable. Similarly, what properties make QIP tractable. In this section we turn our attention to *Graver bases* that makes ILP and QIP tractable in some cases. We base the

[1]This type of matrices are commonly known as matrices that have *Consecutive 1 Property* (C1P).

following exposition on the excellent book of De Loera, Hemmecke and Koppe [23] and a paper of Lee, Onn, Romanchuk and Weismantel [60].

4.2.1 Definitions and Concepts

Let $A \in \mathbb{Z}^{m\times n}, \mathbf{b} \in \mathbb{Z}^m, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n$, and an objective function $f : \mathbb{R}^n \to \mathbb{R}$ be given. Consider the following Integer Programming.

$$(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f} : \min\{f(\mathbf{z}) : A\mathbf{z} = \mathbf{b}, \mathbf{l} \leq \mathbf{b} \leq \mathbf{u}, \mathbf{z} \in \mathbb{Z}^n\} \tag{4.3}$$

We investigate a simple technique that can solve the above $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$ efficiently in several cases. Let us start with the general idea. The essence of these techniques is a simple augmentation procedure: given an initial solution $\mathbf{x}_0$ to $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$ we repeatedly search for augmenting directions until we reach an optimal solution. To do so, we need an *optimality certificate* or *test set*. An *optimality certificate* set is a set $\mathcal{T} \subseteq \mathbb{Z}^n$ such that for a non-optimal solution $\mathbf{x}_0$ there exists a vector $\mathbf{t} \in \mathcal{T}$ and a positive integer α with

(1) $\mathbf{x}_0 + \alpha\mathbf{t}$ feasible and

(2) $f(\mathbf{x}_0) > f(\mathbf{x}_0 + \alpha\mathbf{t})$.

Two questions immediately arise: how big is $\mathcal{T}$ (because we possibly need to enumerate all of it to find an augmenting step), and how many augmenting steps do we need to reach the optimum?

Let us define the *Graver basis* of a matrix A, denoted by $\mathcal{G}(A)$. For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we write $\mathbf{x} \sqsubseteq \mathbf{y}$ and say that $\mathbf{x}$ is *conformal* to $\mathbf{y}$ if $\mathbf{x}^{(i)}\mathbf{y}^{(i)} \geq 0$ (that is, $\mathbf{x}, \mathbf{y}$ lie in the same orthant) and $|\mathbf{x}^{(i)}| \leq |\mathbf{y}^{(i)}|$ for all $i = 1, 2, \ldots, n$.

Definition 5 (Graver basis). *[23] For a given matrix $A \in \mathbb{Z}^{m\times n}$, we define the Graver basis $\mathcal{G}(A)$ of A to be the set of all $\sqsubseteq$-minimal elements in $ker_{\mathbb{Z}^n}(A) = \{\mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} = \mathbf{0}\} \setminus \{\mathbf{0}\}$.*

As an example, let $\mathbf{A} = \begin{pmatrix}1 & 1 & 1\end{pmatrix}$ then

$$\mathcal{G}(A) = \{\pm\begin{pmatrix}1 & -1 & 0\end{pmatrix}^T, \begin{pmatrix}1 & 0 & -1\end{pmatrix}^T, \pm\begin{pmatrix}0 & 1 & -1\end{pmatrix}^T\}.$$

Note that $\mathcal{G}(A)$ is always symmetric, that is, if $\mathbf{g} \in \mathcal{G}(A)$, then $-\mathbf{g} \in \mathcal{G}(A)$. Following lemma shows that $\mathcal{G}(A)$ is a finite set.

Lemma 4 (Gordan-Dickson Lemma, $\sqsubseteq$-version). *Every infinite set $S \subseteq \mathbb{Z}^n$ contains only finitely many $\sqsubseteq$-minimal points.*

Set C in vector space over the real numbers is called *convex* if, for all $\mathbf{x}$ and $\mathbf{y}$ in C and all t in the interval $[0, 1]$, the point $(1-t)\mathbf{x}+t\mathbf{y}$ also belongs to C. A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex function* if its domain is a convex set and $f((1-t)\mathbf{x}+t\mathbf{y}) \leq (1-t)f(\mathbf{x})+tf(\mathbf{y})$

for all $\mathbf{x}$ and $\mathbf{y}$ in its domain and $0 \leq t \leq 1$. It is proven that $\mathcal{G}(A)$ is an optimality certificate for $(IP)_{A,\mathbf{b},\mathbf{l},\mathbf{u},f}$ when f is a *separable convex function* (for all choices of $\mathbf{b}, \mathbf{l}, \mathbf{u}$).

Definition 6 (Convex Separable Function)**.** *[23] A convex function $f : \mathbb{R}^n \to \mathbb{R}$ is separable if it can be written as $f(\mathbf{z}) = \sum_{j=1}^{n} f_j(z_j)$ for convex functions $f_1, \cdots, f_n : \mathbb{R} \to \mathbb{R}$.*

Moreover, if instead of taking any augmenting step we always take the best augmenting step (that is an augmenting step that decreases the function at most), the number of steps needed to obtain the optimum is polynomial in n and the encoding lengths of $A, \mathbf{b}, \mathbf{l}, \mathbf{u}$ [23]. The natural question is then in which cases can we find the best augmenting step quickly.

The most obvious case is when $|\mathcal{G}(A)|$ is polynomial. In this most favorable case many extensions are known to various objectives: f separable convex, $f(W\mathbf{x})$ concave with $W \in \mathbb{Z}^{d \times n}$ for fixed d, $f(W\mathbf{x}) + g(\mathbf{x})$ with W as before and f, g separable convex [76], and for some quadratic (even non-convex) f [60]. However, few natural examples where $|\mathcal{G}(A)|$ is polynomial are known. Next, we elaborate on the case where f is quadratic.

Proposition 4. *[60] It is $\mathcal{NP}$-hard to determine the optimal value of the problem*

$$\min\{f(\mathbf{x}) = \mathbf{x}^T V \mathbf{x} + \mathbf{x}^T \mathbf{w} + a : A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{b} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n\} \tag{4.4}$$

even when $\mathcal{G}(A)$ is given and the function is convex quadratic with matrix $V = \mathbf{v}\mathbf{v}^T$.

Even if, Proposition 4 says that it is hard to find an optimal feasible solution of QIP (4.4), by forcing some restriction on matrix V we can solve it efficiently. In order to formally state such restriction we need some definitions and lemmas.

We say two vectors $\mathbf{x}$ and $\mathbf{y}$ are *sign-compatible* if $\mathbf{x}^{(i)}\mathbf{y}^{(i)} \geq 0$ for all $i = 1, \ldots, n$. Next lemma shows that every element in $ker_{\mathbb{Z}^n}(A)$ can be written as a finite sign-compatible integer linear combination of vectors in $\mathcal{G}(A)$.

Lemma 5. *[23] Every $\mathbf{z} \in ker_{\mathbb{Z}^n}(A)$ can be written as a finite sign-compatible nonnegative integer linear combination $\mathbf{z} = \sum_{i=1}^{r} \alpha_i \mathbf{g}_i$ with $r \leq 2n - 2$, $\alpha_i \in \mathbb{Z}_+$, $\mathbf{g}_i \in \mathcal{G}(A)$ and $\mathbf{g}_i \sqsubseteq \mathbf{z}$ for all i.*

Let $\mathbf{x}_0$ be an initial starting point and $\mathbf{x}_1$ be a feasible point so that $f(\mathbf{x}_1) < f(\mathbf{x}_0)$. Note that $A(\mathbf{x}_1 - \mathbf{x}_0) = A\mathbf{x}_1 - A\mathbf{x}_0 = \mathbf{b} - \mathbf{b} = \mathbf{0}$. Hence, $\mathbf{x}_1 - \mathbf{x}_0$ is in $ker_{\mathbb{Z}^n}(A)$. By Lemma 5, we may assume that $\mathbf{x}_1 = \mathbf{x}_0 + \sum_{i=1}^{t} \mu_i \mathbf{g}_i$ where $\mu_i \in \mathbb{Z}_+$ and $\mathbf{g}_i \in \mathcal{G}(A)$ for all $1 \leq i \leq t$. Therefore, starting from an initial point we can go to any other feasible point just using Graver bases. The question here is that how we can find a combination of Graver basis vectors that leads us to a better point. In some cases it is easy to find a better point. The following simple lemma shows that we can quickly minimize a given quadratic function in a given direction.

Lemma 6. *[60] There is an algorithm that, given bounds* $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^n_\infty$*, direction* $\mathbf{g} \in \mathbb{Z}^n$*, point* $\mathbf{z} \in \mathbb{Z}^n$ *with* $\mathbf{l} \leq \mathbf{z} \leq \mathbf{u}$*, and quadratic function* $f(x) = \mathbf{x}^T V \mathbf{x} + \mathbf{w}^T \mathbf{x} + a$ *with* $V \in \mathbb{Z}^{n \times n}, \mathbf{w} \in \mathbb{Z}^n$*, and* $a \in \mathbb{Z}$*, solves in polynomial time the univariate problem*

$$\min\{f(\mathbf{z} + \mu \mathbf{g}) : \mu \in \mathbb{Z}_+, \mathbf{l} \leq \mathbf{z} + \mu \mathbf{g} \leq \mathbf{u}\}.$$

For some types of QIP (4.4), considering one direction of Graver basis is sufficient to find a better point. For instance, it is the case when $\mathbf{V}$ is in the *dual quadratic Graver cone* of $\mathbf{A}$.

Definition 7. *Dual quadratic Graver cone of a matrix* $\mathbf{A}$*, denoted by* $Q^*(\mathbf{A})$*, is the set of all symmetric matrices* $\mathbf{V}$ *such that* $\mathbf{g}^T \mathbf{V} \mathbf{h} \geq 0$ *for all* $\mathbf{g}, \mathbf{h} \in \mathcal{G}(\mathbf{A})$ *where*

i) $\mathbf{g} \neq \mathbf{h}$ *and*

ii) $\mathbf{g}^{(i)} \mathbf{h}^{(i)} \geq 0$ *for all* i*.*

Intuitively, the reason why it is enough to consider one direction of Graver basis when $\mathbf{V} \in Q^*(\mathbf{A})$ is as follows. Having $f(\mathbf{x}_1) < f(\mathbf{x}_0)$ implies that:

$$\begin{aligned} & f(\mathbf{x}_0 + \sum_{i=1}^{t} \mu_i \mathbf{g}_i) - f(\mathbf{x}_0) < 0 \\ \Rightarrow & \sum_{i=1}^{t} (\mu_i \mathbf{x}_0^T \mathbf{V} \mathbf{g}_i + \mu_i^2 \mathbf{g}_i^T \mathbf{V} \mathbf{g}_i + \mu_i \mathbf{g}_i^T \mathbf{w}) + \sum_{i \neq j}^{t} \mu_i \mathbf{g}_i^T \mathbf{V} \mu_j \mathbf{g}_j < 0 \end{aligned}$$

Since $\mathbf{V} \in Q^*(\mathbf{A})$ then the term $\sum_{i \neq j}^{t} \mu_i \mathbf{g}_i^T \mathbf{V} \mu_j \mathbf{g}_j$ is a positive term. As a result, there has to be an index $1 \leq i \leq t$ such that $\mu_i \mathbf{x}^T \mathbf{V} \mathbf{g}_i + \mu_i^2 \mathbf{g}_i^T \mathbf{V} \mathbf{g}_i + \mu_i \mathbf{g}_i^T \mathbf{w} < 0$. Equivalently, for $\mathbf{g}_i \in \mathcal{G}(\mathbf{A})$, we have

$$\begin{aligned} & \mu_i \mathbf{x}_0^T \mathbf{V} \mathbf{g}_i + \mu_i^2 \mathbf{g}_i^T \mathbf{V} \mathbf{g}_i + \mu_i \mathbf{g}_i^T \mathbf{w} < 0 \\ \Rightarrow & f(\mathbf{x}_0 + \mu_i \mathbf{g}_i) - f(\mathbf{x}_0) < 0 \end{aligned}$$

Hence, in the case when $\mathbf{V} \in Q^*(\mathbf{A})$, and $\mathbf{x}$ is not an optimal point for QIP (4.4), then there exists an augmenting direction $\mathbf{g} \in \mathcal{G}(\mathbf{A})$ and $\mu \in \mathbb{Z}_+$ so that $f(\mathbf{x} + \mu \mathbf{g}) < f(\mathbf{x})$. Recall that by Lemma 6 we can minimize quadratic function f in a given direction (in time complexity $O(1)$). In fact, it is proven that if we find the best augmenting direction at each step then we can find an optimal point in polynomial time. By best augmenting direction of a point $\mathbf{x}$ we mean $\mathbf{g} \in \mathcal{G}(\mathbf{A})$ and $\mu \in \mathbb{Z}_+$ so that $f(\mathbf{x} + \mu \mathbf{g}) \leq f(\mathbf{x} + \mu' \mathbf{g}')$ for all $\mathbf{g}' \in \mathcal{G}(\mathbf{A})$ and $\mu' \in \mathbb{Z}+$.

Theorem 16. *[60] There is an algorithm that, given $\mathbf{A} \in \mathbb{Z}^{m \times n}$, its Graver basis $\mathcal{G}(\mathbf{A})$, bounds $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^n_\infty$, $\mathbf{b} \in \mathbb{Z}^m$, integer matrix $\mathbf{V} \in Q^*(\mathbf{A})$ in the dual quadratic Graver cone, $\mathbf{w} \in \mathbb{Z}^n$, and $a \in \mathbb{Z}$, solves in polynomial time the quadratic integer program*

$$\min\{f(\mathbf{x}) = \mathbf{x}^T V \mathbf{x} + \mathbf{x}^T \mathbf{w} + a : A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{b} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n\}$$

Recall that the starting phase of the augmenting procedure we described is finding an initial feasible point. Next lemma shows that, given the Graver basis, we can also find an initial feasible point or assert that the given set is empty or infinite, in polynomial time.

Lemma 7. *[60] There is an algorithm that, given integer $m \times n$ matrix $\mathbf{A}$, its Graver basis $\mathcal{G}(\mathbf{A}), \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n_\infty$, and $\in \mathbb{Z}^m$, in polynomial time, either finds a feasible point in the set $S = \{x \in \mathbb{Z}^n : \mathbf{A}\mathbf{x} = , \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$ or asserts that S is empty or infinite.*

4.2.2 Graver Basis and MLA Parameterized by Vertex Cover

The main focus of this part is on establishing connections between the QIP for MLA parameterized by vertex cover and Graver Bases. We attempt to solve the problem in $\mathcal{FPT}$ time using Graver Bases machinery. Even if, we do not solve the problem completely, we propose new techniques and conjectures which we believe could be used for other problems as well.

We start off with constrains matrix $\mathbf{A} \in \mathbb{Z}_2^{(2^k-1)\times(2^k-1)(k+1)}$ in QIP (4.2). This matrix is nicely structured that guarantees us a polynomial size of $\mathcal{G}(\mathbf{A})$.

Claim 1. *For* MLA, *$\mathcal{G}(\mathbf{A})$ contains $k(k+1)^2(2^k-1)$ vectors.*

Proof. Each row of $\mathbf{A}$ consists of $k+1$ consecutive 1s, and 0s everywhere else. Let us say 1s are in positions $(k+1)(i-1), (k+1)(i-1)+1, \ldots, (k+1)i$ for some row $\mathbf{A}_i$. We show any vector $\mathbf{g}$ of length $(2^k-1)(k+1)$ is in $\mathcal{G}(\mathbf{A})$ if and only if

$$\mathbf{g}^{(j)} = \begin{cases} 1 & j = \alpha \\ -1 & j = \beta \\ 0 & otherwise \end{cases}$$

for some α and β, $(k+1)(i-1) \leq \alpha, \beta \leq (k+1)i$. It is easy to observe that $\mathbf{Ag} = \mathbf{0}$. Hence, $\mathbf{g} \in ker_{\mathbb{Z}^n}(\mathbf{A})$. It remains to show that $\mathbf{g}$ and $-\mathbf{g}$ are $\sqsubseteq$ elements of $ker_{\mathbb{Z}^n}(A)$. Suppose it is not the case and there exists $\mathbf{h} \in ker_{\mathbb{Z}^n}(\mathbf{A})$ so that $\mathbf{h} \sqsubseteq \mathbf{g}$. By definition, $|\mathbf{h}^{(j)}| \leq |\mathbf{g}^{(j)}|$ for $1 \leq j \leq (k+1)(2^k-1)$. As a result, $\mathbf{h}^{(j)} = 0$ for all $j \neq \alpha, \beta$ and $|\mathbf{h}^{(\alpha)}| = |\mathbf{h}^{(\beta)}| = 1$. It can happen if and only if $\mathbf{h} = \mathbf{g}$ or $\mathbf{h} = -\mathbf{g}$. □

Before introducing our methodology, we point out that Theorem 16 cannot be applied to QIP (4.2). For instance, consider the case for $k = 2$. There are $\mathbf{g}, \mathbf{h} \in \mathcal{G}(\mathbf{A})$ such that $\mathbf{g}^T\mathbf{V}\mathbf{h} < 0$. For example consider

$$\begin{pmatrix}1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}\frac{1}{2} & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0\\ 1 & \frac{1}{2} & 0 & 0 & 0 & 1 & 1 & 1 & 1\\ 0 & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0 & 1\\ 1 & 0 & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0 & \frac{1}{2} & 1 & 1 & 1 & 1\\ 0 & 1 & 1 & 0 & 1 & \frac{1}{2} & 0 & 1 & 1\\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0\\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1\\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1\end{pmatrix}\begin{pmatrix}0\\0\\0\\0\\-1\\1\\0\\0\\0\end{pmatrix} = -2$$

Definition 8. *Let $\mathbf{x} \in \mathbb{Z}^n$ be a feasible point for QIP(4.2). We say $\mathbf{x}$ is a s-stable point, if there does not exist a set $S \subseteq \mathcal{G}(A)$ of size at most s such that $f(\mathbf{x} + \sum\limits_{\mathbf{g}_i \in S} \mu_i \mathbf{g}_i) < f(\mathbf{x})$ with $\mu_i \in \mathbb{Z}_+$.*

As an example $\mathbf{x} \in \mathbb{Z}^n$ is a 1-stable point if there does not exist a direction $\mathbf{g} \in \mathcal{G}(A)$ and $\mu \in \mathbb{Z}_+$ such that $f(\mathbf{x} + \mu\mathbf{g}) < f(\mathbf{x})$. Note that, according to the definition, a s-stable point is t-stable for all $1 \leq t \leq s$. Next, we provide some properties of 1-stable points.

Observation 1. *Let $\mathbf{x}$ be a 1-stable point of IQP(4.2) and $\mathbf{g} \in \mathcal{G}(A)$ be a vector such that $\mathbf{g}^{(i)} = 1$ and $\mathbf{g}^{(j)} = -1$. If $\mathbf{g}^T\mathbf{V}\mathbf{g} < 0$ then either $\mathbf{x}^{(i)} = 0$ or $\mathbf{x}^{(j)} = 0$.*

Proof. Assume $\mathbf{x}^{(i)} \neq 0$ and $\mathbf{x}^{(j)} \neq 0$. In this case both $\mathbf{x} + \mathbf{g}$ and $\mathbf{x} - \mathbf{g}$ are feasible points.

$$f(\mathbf{x} + \mathbf{g}) - f(\mathbf{x}) = 2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w}. \tag{4.5}$$

$$f(\mathbf{x} - \mathbf{g}) - f(\mathbf{x}) = -2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mathbf{g}^T\mathbf{V}\mathbf{g} - \mathbf{g}^T\mathbf{w}. \tag{4.6}$$

Since $\mathbf{g}^T\mathbf{V}\mathbf{g} < 0$ then either $f(\mathbf{x} + \mathbf{g}) - f(\mathbf{x}) < 0$ or $f(\mathbf{x} - \mathbf{g}) - f(\mathbf{x}) < 0$ which is in contradiction to $\mathbf{x}$ being a 1-stable point. □

Observation 2. *Let $\mathbf{x}$ be a 1-stable point of IQP(4.2) and $\mathbf{g} \in \mathcal{G}(\mathbf{A})$ be a vector such that $\mathbf{g}^T\mathbf{V}\mathbf{g} > 0$. If $\mathbf{x} - \mathbf{g}$ is a feasible solution to IQP(4.2) then*

$$2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mathbf{g}^T\mathbf{w} < \mathbf{g}^T\mathbf{V}\mathbf{g} < 2k.$$

Proof. Since $\mathbf{x}$ is an 1-stable point and $\mathbf{x} - \mathbf{g}$ then $f(\mathbf{x}) < f(\mathbf{x} - \mathbf{g})$. Therefore, the following inequality holds:

$$\begin{aligned} 0 &< f(\mathbf{x} - \mathbf{g}) - f(\mathbf{x}) \\ 0 &< -2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mathbf{g}^T\mathbf{V}\mathbf{g} - \mathbf{g}^T\mathbf{w} \\ \Rightarrow 2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mathbf{g}^T\mathbf{w} &< \mathbf{g}^T\mathbf{V}\mathbf{g} < 2k. \end{aligned}$$

Recall that the entries of $\mathbf{V}$ are at most $k-1$. Therefore, one can see that $\mathbf{g}^T\mathbf{V}\mathbf{g}$ is at most $2k$. □

Observation 3. *Let $\mathbf{x}$ be a 1-stable point for QIP(4.2) and $\mathbf{y} = \mathbf{x} + \mu\mathbf{g} + \lambda\mathbf{h}, \mathbf{y}' = \mathbf{y} + \mathbf{g}$ be feasible points with $\mu, \lambda \in \mathbb{Z}_+$ and $\mathbf{g}, \mathbf{h} \in \mathcal{G}(A)$. If $f(\mathbf{y}) < f(\mathbf{x})$ and $\mathbf{g}^T\mathbf{V}\mathbf{g} < 0$ then, $f(\mathbf{y}') < f(\mathbf{x})$.*

Proof. Suppose $f(\mathbf{x} + (\mu+1)\mathbf{g} + \lambda\mathbf{h}) > f(\mathbf{x})$. It implies that

$$\begin{aligned} f(\mathbf{x} + (\mu+1)\mathbf{g} + \lambda\mathbf{h}) - f(\mathbf{x}) = 2(\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{x} + (\mu+1)^2\mathbf{g}^T\mathbf{V}\mathbf{g} + (\mu+1)\mathbf{g}^T\mathbf{w} + \\ 2\lambda\mathbf{h}^T\mathbf{V}\mathbf{x} + \lambda^2\mathbf{h}^T\mathbf{V}\mathbf{h} + \mathbf{h}^T\mathbf{w} + 2(\mu+1)\lambda\mathbf{g}^T\mathbf{V}\mathbf{h} \end{aligned} \tag{4.7}$$

has a positive value. Moreover, since $f(\mathbf{y}) < f(\mathbf{x})$, the following term has a negative value

$$f(\mathbf{y}) - f(\mathbf{x}) = 2\mu\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu^2\mathbf{g}^T\mathbf{V}\mathbf{g} + \mu\mathbf{g}^T\mathbf{w} + 2\lambda\mathbf{h}^T\mathbf{V}\mathbf{x} + \lambda^2\mathbf{h}^T\mathbf{V}\mathbf{h} + \mathbf{h}^T\mathbf{w} + 2\mu\lambda\mathbf{g}^T\mathbf{V}\mathbf{h} \tag{4.8}$$

Note that since $\mathbf{x}$ is a *1-stable* point then, $2\mu\lambda\mathbf{g}^T\mathbf{V}\mathbf{h}$ and $2(\mu+1)\lambda\mathbf{g}^T\mathbf{V}\mathbf{h}$ are the only negative terms in (4.7) and (4.8).

From (4.7) being positive and (4.8) being negative we have:

$$\begin{aligned} 0 &< 2\mathbf{g}^T\mathbf{V}\mathbf{x} + (2\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} + 2\lambda\mathbf{g}^T\mathbf{V}\mathbf{h} \\ \Rightarrow 2\lambda|\mathbf{g}^T\mathbf{V}\mathbf{h}| &< 2\mathbf{g}^T\mathbf{V}\mathbf{x} + (2\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} \end{aligned} \tag{4.9}$$

On the other hand, since $\mathbf{x}$ is an *1-stable* point and (4.8) is a negative term, we conclude that

$$\begin{aligned} 2\mu\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu^2\mathbf{g}^T\mathbf{V}\mathbf{g} + \mu\mathbf{g}^T\mathbf{w} + 2\mu\lambda\mathbf{g}^T\mathbf{V}\mathbf{h} &< 0 \\ \Rightarrow 2\mu\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu^2\mathbf{g}^T\mathbf{V}\mathbf{g} + \mu\mathbf{g}^T\mathbf{w} + &< 2\mu\lambda|\mathbf{g}^T\mathbf{V}\mathbf{h}| \\ \Rightarrow 2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} &< 2\lambda|\mathbf{g}^T\mathbf{V}\mathbf{h}| \end{aligned} \tag{4.10}$$

As a result of (4.9) and (4.10), $0 < (\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{g}$ which is in contradiction to the assumption that $\mathbf{g}^T\mathbf{V}\mathbf{g} < 0$.

□

Lemma 8. *[23] Let $\mathbf{z}_0$ and $\mathbf{z}_1$ be feasible solutions to $A\mathbf{z} = \mathbf{b}$, $\mathbf{l} \leq \mathbf{z} \leq \mathbf{u}$. Moreover, let $\mathbf{z}_1 - \mathbf{z}_0 = \sum_{i=1}^{r} \alpha_i\mathbf{g}_i$ be a nonnegative integer linear sign-compatible decomposition into Graver basis elements $\mathbf{g}_i \in \mathcal{G}(A)$. Then for all choices of $\beta_1, \cdots, \beta_r \in \mathbb{Z}$ with $0 \leq \beta_j \leq \alpha_j$, $j = 1, \cdots, r$, the vector $\mathbf{z}_0 + \sum_{i=1}^{r} \beta_i\mathbf{g}_i$ is also a feasible solution to $A\mathbf{z} = \mathbf{b}$, $\mathbf{l} \leq \mathbf{z} \leq \mathbf{u}$.*

Lemma 9. *Let* $\mathbf{x}$ *be a 1-stable point for QIP(4.2) and* $\mathbf{y} = \mathbf{x} + \mu\mathbf{g} + \sum \lambda_i \mathbf{h}_i$ *be a feasible point with* $\mu, \lambda_i \in \mathbb{Z}_+$ *and* $\mathbf{g}, \mathbf{h}_i \in \mathcal{G}(A)$. *If* $f(\mathbf{y}) < f(\mathbf{x})$ *and* $\mathbf{g}^T\mathbf{V}\mathbf{g} < 0$ *then, for a feasible point* $\mathbf{y}' = \mathbf{x} + (\mu+1)\mathbf{g} + \sum \lambda_i \mathbf{h}_i$ *we have* $f(\mathbf{y}') < f(\mathbf{x})$.

Proof. By Lemma 5 we assume that $\mathbf{g}$ and $\mathbf{h}_i$s are sign-compatible. Suppose $f(\mathbf{y}') > f(\mathbf{x})$ i.e. $f(\mathbf{y}') - f(\mathbf{x}) > 0$. It implies that

$$\begin{aligned} f(\mathbf{y}') - f(\mathbf{x}) = {} & 2(\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{x} + (\mu+1)^2\mathbf{g}^T\mathbf{V}\mathbf{g} + (\mu+1)\mathbf{g}^T\mathbf{w} \\ & + \sum (2\lambda_i {\mathbf{h}_i}^T\mathbf{V}\mathbf{x} + {\lambda_i}^2 {\mathbf{h}_i}^T\mathbf{V}\mathbf{h}_i + {\mathbf{h}_i}^T\mathbf{w}) \\ & + \sum_{i,j} \lambda_i\lambda_j {\mathbf{h}_i}^T\mathbf{V}\mathbf{h}_j \\ & + 2(\mu+1)\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i \end{aligned}$$

is a positive term. Moreover, since $f(\mathbf{y}) < f(\mathbf{x})$ then the following term has a negative value,

$$\begin{aligned} f(\mathbf{y}) - f(\mathbf{x}) = {} & 2\mu\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu^2\mathbf{g}^T\mathbf{V}\mathbf{g} + \mu\mathbf{g}^T\mathbf{w} \\ & + \sum (2\lambda_i {\mathbf{h}_i}^T\mathbf{V}\mathbf{x} + {\lambda_i}^2 {\mathbf{h}_i}^T\mathbf{V}\mathbf{h}_i + {\mathbf{h}_i}^T\mathbf{w}) \\ & + \sum_{i,j} \lambda_i\lambda_j {\mathbf{h}_i}^T\mathbf{V}\mathbf{h}_j \\ & + 2\mu\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i \end{aligned}$$

We point out that $\sum\limits_{i,j} \lambda_i\lambda_j {\mathbf{h}_i}^T\mathbf{V}\mathbf{h}_j + 2\mu\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i$ is the only negative term in the above equation. As a result of $f(\mathbf{y}') - f(\mathbf{x}) > 0$ and $f(\mathbf{y}) - f(\mathbf{x}) < 0$, it is concluded that:

$$\begin{aligned} & f(\mathbf{y}) - f(\mathbf{x}) < f(\mathbf{y}') - f(\mathbf{x}) \\ & \Rightarrow 2\mu\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu^2\mathbf{g}^T\mathbf{V}\mathbf{g} + \mu\mathbf{g}^T\mathbf{w} + 2\mu\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i < \\ & 2(\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{x} + (\mu+1)^2\mathbf{g}^T\mathbf{V}\mathbf{g} + (\mu+1)\mathbf{g}^T\mathbf{w} + 2(\mu+1)\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i \\ & \Rightarrow 0 < 2\mathbf{g}^T\mathbf{V}\mathbf{x} + (2\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} + 2\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i \end{aligned}$$

Note that we might assume that $2\mu\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu^2\mathbf{g}^T\mathbf{V}\mathbf{g} + \mu\mathbf{g}^T\mathbf{w} + 2\mu\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i < 0$, as otherwise, the feasible point $\mathbf{x} + \sum \lambda_i \mathbf{h}_i$ (it is feasible because of Lemma 8) has a smaller cost function than $\mathbf{x}$. Immediately, the following two inequalities hold:

$$2|\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i| < 2\mathbf{g}^T\mathbf{V}\mathbf{x} + (2\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} \tag{4.11}$$

$$2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mu\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} < 2|\sum \lambda_i \mathbf{g}^T\mathbf{V}\mathbf{h}_i| \tag{4.12}$$

which implies that $\mathbf{g}^T\mathbf{V}\mathbf{g} > 0$. A contradiction to the assumption that $\mathbf{g}^T\mathbf{V}\mathbf{g} < 0$, that completes the proof. □

Lemma 10. *Let* $\mathbf{x}$ *be a 1-stable point for QIP(4.2) and* $\mu, \lambda \in \mathbb{Z}_+$ *are maximum so that* $\mathbf{y} = \mathbf{x} + (\mu+1)\mathbf{g} + \lambda\mathbf{h}$ *is a feasible point with* $\mathbf{g}, \mathbf{h} \in \mathcal{G}(A)$*. If* $f(\mathbf{y} - \mathbf{g}) < f(\mathbf{x})$, $\mathbf{g}^T\mathbf{V}\mathbf{g} > 0$ *and* $\mathbf{h}^T\mathbf{V}\mathbf{h} > 0$ *then,* μ *and* λ *are at most* $2k$.

Proof. Without loss of generality, we assume that $\mu \leq \lambda$. Furthermore, note that $\mathbf{x} + (\mu + 1)\mathbf{g} + \lambda\mathbf{h}$ is a feasible point and $f(\mathbf{x} + (\mu+1)\mathbf{g} + \lambda\mathbf{h}) > f(\mathbf{x})$. As a result, we have the following:

$$\begin{aligned}
f(\mathbf{x} + \mu\mathbf{g} + \lambda\mathbf{h}) - f(\mathbf{x}) &< f(\mathbf{x} + (\mu+1)\mathbf{g} + \lambda\mathbf{h}) - f(\mathbf{x}) \\
\Rightarrow 0 &< 2\mathbf{g}^T\mathbf{V}\mathbf{x} + (2\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} + 2\lambda\mathbf{g}^T\mathbf{V}\mathbf{h} \\
\Rightarrow 2\lambda|\mathbf{g}^T\mathbf{V}\mathbf{h}| &< 2\mathbf{g}^T\mathbf{V}\mathbf{x} + (2\mu+1)\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} \\
\xrightarrow{\mu\leq\lambda, 0<\mathbf{g}^T\mathbf{V}\mathbf{g}} 2\lambda|\mathbf{g}^T\mathbf{V}\mathbf{h}| &< 2\mathbf{g}^T\mathbf{V}\mathbf{x} + (2\lambda+1)\mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} \\
\Rightarrow \lambda(2|\mathbf{g}^T\mathbf{V}\mathbf{h}| - 2\mathbf{g}^T\mathbf{V}\mathbf{g}) &< 2\mathbf{g}^T\mathbf{V}\mathbf{x} + \mathbf{g}^T\mathbf{V}\mathbf{g} + \mathbf{g}^T\mathbf{w} \\
\xrightarrow{\text{By Lemma 2}} 2\lambda(|\mathbf{g}^T\mathbf{V}\mathbf{h}| - \mathbf{g}^T\mathbf{V}\mathbf{g}) &< 2\mathbf{g}^T\mathbf{V}\mathbf{g} < 4k \\
\xrightarrow{0<|\mathbf{g}^T\mathbf{V}\mathbf{h}|-\mathbf{g}^T\mathbf{V}\mathbf{g}} \mu &\leq \lambda < 2k
\end{aligned}$$

□

Backed up by the above lemmas and observations, an interesting conjecture is that the length of an augmenting direction needed to be taken from a non-optimal point $\mathbf{x}$ towards a better point can be found in time bounded by a function depending only on k. In other words, for non-optimal point $\mathbf{x}$ there exists a better point $\mathbf{x}' = \mathbf{x} + \sum \mu_i \mathbf{g}_i$ where there is an algorithm that finds μ_i in time $O(\mathscr{F}(k))$ when $\mathscr{F}$ is a function depending only on k. Next, we formally state our conjecture.

Conjecture 1. *Let* $\mathbf{x}$ *be a 1-stable point for QIP(4.2). If* $\mathbf{x}$ *is not optimal then there exists a feasible point* $\mathbf{y} = \mathbf{x} + \sum \mu_i \mathbf{g}_i$ *such that*

- $\mu_i \in \mathbb{Z}_+$ *and* $\mathbf{g}_i \in \mathcal{G}(A)$ *and*
- $M_j - \alpha \leq \mu_i \leq M_j$ *or* $1 \leq \mu_i \leq \beta$ *where* α *and* β *are bounded by a function of* k *and*
- $f(\mathbf{y}) < f(\mathbf{x})$.

We note that Conjecture 1 is true for the cases we discussed in Lemmas 9 and 10. An interesting question is that whether it is possible to extend the ideas in the proofs of these two Lemmas and prove or disprove Conjecture 1. Based on the above conjecture we propose the following algorithm.

Algorithm 6 Bounded Augmenting Steps

```
1:  Input: Graph $G = (V, E)$ with vertex cover of size $k$ i.e $|vc(G)| = k$ and fixed ordering
    $\pi$ on $vc(G)$
2:  Output: An optimal distribution of vertices in $V \setminus vc(G)$ with respect to $\pi$
3:  Compute matrices $\mathbf{V}, \mathbf{w}$ and $\mathbf{A}$ of QIP (4.2)
4:  Find an initial point $\mathbf{z}$ (Lemma 7)
5:  if there exists $\mathbf{g} \in \mathcal{G}(\mathbf{A})$ and $\mu \in \mathbb{Z}_+$ so that $f(\mathbf{z} + \mu\mathbf{g}) < f(\mathbf{z})$ then
6:      $\mathbf{z} \leftarrow \mathbf{z} + \mu\mathbf{g}$
7:  for $i = 2$ to $|\mathcal{G}(\mathbf{A})|$ do
8:      for each set $S$ consists of $i$ Graver basis vectors do
9:          if there exit $\mu_j$s such that $f(\mathbf{z} + \sum_{\mathbf{g}_j \in S} \mu_j \mathbf{g}_j) < f(\mathbf{z})$ then
10:             $\mathbf{z} \leftarrow \mathbf{z} + \sum_{\mathbf{g}_j \in S} \mu_j \mathbf{g}_j$
11:             go to Step 5
12: return $\mathbf{z}$
```

Notice that the maximum value that the objective function of QIP (4.2) can have is $O(n^3)$. Indeed, the maximum value occurs if the input graph is a clique with n vertices in which case the cost of any ordering is $\sum_{i=1}^{n} \frac{(i-2)(i-1)}{2} + \frac{(n-i-1)(n-i)}{2}$. Therefore, in Algorithm 6, number of augmenting steps required to reach an optimal point is at most $O(n^3)$ staring from any initial point.

Conjecture 2. *Algorithm 6 runs in $\mathcal{FPT}$ time.*

Suppose the time complexity of Algorithm 6 is $O(F(k)n^3)$ where $F(k)$ is a function depending only on k. For a vertex cover of size k there are $k!$ orderings. For each fixed ordering on $vc(G)$, we need to call Algorithm 6 and find an optimal distribution of vertices in $V \setminus vc(G)$. Over all the $k!$ cases we choose the one with minimum cost. In total it requires $O(k!F(k)n^3)$ time to find an optimal ordering for MLA problem on graph G.

4.3 Karush-Kuhn-Tucker Conditions

In this section we suggest another approach to design an $\mathcal{FPT}$ algorithm for QIP (4.2). At a high level, the essence of our approach is as follows. We impose *Karush-Kuhn-Tucker (KKT)* conditions on feasible (not necessarily integral) points for QIP (4.2). Let S^* be the set of all feasible points that satisfies the conditions. There are two crucial conjectures. First, S^* contains polynomial number of points. Second, an optimal point is close enough to some points in S^*. Assuming our conjectures are correct, we enumerate all points in S^* and search within a bounded distance around each point in S^* for optimal integral feasible solution. We continue by discussing KKT-conditions.

KKT conditions are first order conditions for a solution to be optimal in a nonlinear programming. Consider minimizing nonlinear function $f(\mathbf{x})$ subject to constrains $g_i(\mathbf{x}) \leq 0$ $(i = 1, \ldots, m)$ and $h_j(\mathbf{x}) = 0$ $(j = 1, \ldots, l)$. Suppose that $f, g_i, h_j : \mathbb{R}^n \to \mathbb{R}$, are continuously differentiable functions at point $\mathbf{x}^*$. If $\mathbf{x}^*$ is a local optimum that satisfies some regularity conditions, then there exist constants μ_i $(i = 1, \ldots, m)$ and λ_j $(j = 1, \ldots, l)$, called KKT multipliers, such that:

i. $-\nabla f(\mathbf{x}^*) = \sum_{i=1}^{m} \mu_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{l} \lambda_j \nabla h_j(\mathbf{x}^*)$

ii. $g_i(\mathbf{x}^*) \leq 0$ for all $i = 1, \ldots, m$

iii. $h_j(\mathbf{x}^*) = 0$ for all $j = 1, \ldots, l$

iv. $\mu_i \geq 0$ for all $i = 1, \ldots, m$

v. $\mu_i g_i(\mathbf{x}^*) = 0$ for all $i = 1, \ldots, m$

Note that in order for a minimum point $\mathbf{x}^*$ to satisfy the above KKT conditions, the problem should satisfy some *regularity conditions*; one of the most used one is that, if g_i and h_j are *affine functions*, then no other condition is needed.

Definition 9. *We say a function $f : \mathbb{R}^m \to \mathbb{R}^n$ is affine if there is a linear function $L : \mathbb{R}^m \to \mathbb{R}^n$ and a vector $\mathbf{b} \in \mathbb{R}^n$ such that $f(\mathbf{x}) = L(\mathbf{x}) + \mathbf{b}$ for all $\mathbf{x} \in \mathbb{R}^m$.*

4.3.1 KKT conditions and MLA(G,vc(G)) problem

Here we apply the KKT conditions on MLA(G, k) problem. The constraints for MLA(G, k) are as follows:

$$\begin{aligned}
g_i^j &: -x_i^j \leq 0 \text{ for } i = 0, \cdots, (2^k - 1) \text{ and } 0 \leq j \leq k \\
h_1 &: x_1^0 + x_1^1 + \cdots + x_1^k - \mathbf{b}^{(1)} = 0 \\
h_2 &: x_2^0 + x_2^1 + \cdots + x_2^k - \mathbf{b}^{(2)} = 0 \\
&\vdots \\
h_{(2^k-1)} &: x_{(2^k-1)}^0 + x_{2^k-1}^2 + \cdots + x_{(2^k-1)}^k - \mathbf{b}^{(2^k-1)} = 0
\end{aligned}$$

The partial derivative of $f(\mathbf{x}) = \mathbf{x}^T \mathbf{V} \mathbf{x} + \mathbf{x}^T \mathbf{w}$ with respect to each variable x_i^j is given by the following formula:

$$\frac{\partial f}{\partial x_i^j} = \mathbf{w}^{(x_i^j)} + 2 x_i^j \mathbf{V}_{[x_i^j, x_i^j]} + \sum_{j' \neq j, i' \neq i} x_{i'}^{j'} \mathbf{V}_{[x_i^j, x_{i'}^{j'}]} \tag{4.13}$$

Plus the partial derivatives of the constrains with respect to variables x_i^j are:

$$\frac{\partial h_t}{\partial x_i^j} = \begin{cases} 1 & \text{if } t = i \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

$$\frac{\partial g_{i'}^{j'}}{\partial x_i^j} = \begin{cases} -1 & \text{if } i' = i \text{ and } j' = j \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Therefore, the first KKT-condition can be written as:

$$\begin{aligned} -\nabla f(\mathbf{x}^*) &= \sum_{i,j} \mu_i^j \nabla g_i^j(\mathbf{x}^*) + \sum_{j=1}^{2^k-1} \lambda_j \nabla h_j(\mathbf{x}^*) \\ \Rightarrow -\frac{\partial f}{\partial x^{*j'}_{i'}} &= \sum_{i,j} \mu_i^j \frac{\partial g_i^j}{\partial x^{*j'}_{i'}} + \sum_{j=1}^{2^k-1} \lambda_j \frac{\partial h_j}{\partial x^{*j'}_{i'}} \\ \Rightarrow -\mu_i^j + \lambda_i &= -(\mathbf{w}^{(x^{*j'}_{i'})} + 2x^{*j'}_{i'}\mathbf{V}_{[x^{*j'}_{i'}, x^{*j'}_{i'}]} + \sum_{j' \neq j, i' \neq i} x_i^j \mathbf{V}_{[x_i^j, x^{*j'}_{i'}]}) \end{aligned} \tag{4.16}$$

Let $S^* = \{\mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_r^*\}$ be the set of all points (not necessarily integral) that satisfy the KKT conditions for the QIP (4.2). Note that constrains g_i^j and h_i are linear functions, hence a minimum point has to satisfy KKT conditions. Consider the fifth KKT condition. That condition is a quadratic equation and according to it, we have $\mu_i^j \cdot x_i^j = 0$ for all i and j. Note that it implies that either $\mu_i^j = 0$ or $x_i^j = 0$. However, number of these possibilities is $O(4^k)$. For each possible assignment of 0 to μ_i^j and x_i^j, we end up with a linear system with a bounded number of variables. As a result, we can enumerate all points in S^* in $\mathcal{FPT}$ time.

Notice that S^* contains all the local optimum points. Furthermore, observe that for a local optimum point, partial derivatives in all directions are zero. According to the definition of 1-stable point, from a 1-stable point we cannot reach to a better point using only one element of the Graver basis. Hence, we conjecture that there is polynomial number of 1-stable points and each 1-stable point is close enough to a point in S^*.

Conjecture 3. *There are polynomially many 1-stable points for QIP(4.2).*

Conjecture 4. *If $\mathbf{x}$ is a 1-stable point, then there exists a point $\mathbf{x}^* \in S^*$ and a function $\mathcal{F}(k)$ depending only on k such that $|\mathbf{y}^{(i)} - \mathbf{x}^{*(i)}| < \mathcal{F}(k)$, for all i.*

Based on the aforementioned conjectures, we propose the following algorithm. It is worth mentioning that, provided that there are bounded number of points that satisfy KKT conditions and optimal points are close enough to KKT points, the following algorithm finds an optimal in $\mathcal{FPT}$ time. Suppose the time complexity of Algorithm 7 is $O(F(k).poly(n))$ where $F(k)$ is a function depending only on k. For a vertex cover of size k there are $k!$ orderings. For each fixed ordering on $vc(G)$, we need to call Algorithm 7 and find an optimal distribution of vertices in $V \setminus vc(G)$. Over all the $k!$ cases we choose the one with minimum cost. In total it requires $O(k!.F(k).poly(n))$ time to find an optimal ordering for MLA problem on graph G.

Algorithm 7 Search Around KKT points

```
1: Input: Graph $G = (V, E)$ with vertex cover of size $k$ i.e $|vc(G)| = k$ and fixed ordering
   $\pi$ on $vc(G)$
2: Output: An optimal distribution of vertices in $V \setminus vc(G)$ with respect to $\pi$
3: Compute matrices $\mathbf{V}, \mathbf{w}$ and $\mathbf{A}$ of QIP (4.2)
4: Let $\mathbf{z}$ be an initial feasible point
5: Let $S^*$ be the set of all points $\mathbf{x}^* \in \mathbb{R}^{(k+1)(2^k-1)}$ that satisfy KKT-conditions and
   $\mathbf{Ax} = \mathbf{b}$
6: for each $\mathbf{x}^* \in S^*$ do
7:     Let $S'$ be the set of all integer feasible points $\mathbf{y}$ so that $|\mathbf{y}^{(i)} - \mathbf{x}^{*(i)}| < \mathcal{F}(k)$
8:     Let $\mathbf{y}^*$ be the point in $S'$ with minimum $f(\mathbf{y}^*)$
9:     if $f(\mathbf{y}^*) < f(\mathbf{z})$ then
10:        $\mathbf{z} \leftarrow \mathbf{y}^*$
11: return $\mathbf{z}$
```

Chapter 5

Ordering with Precedence Constraints and Budget Minimization

In Section 2.4 we introduced the bipartite graph ordering problem, which is equivalent to a generalization of a molecular folding problem on RNA sequences. We initiate the study of which graph classes admit polynomial-time exact solutions. We also give the first results for the weighted version of the problem; previous work on the molecular folding problem assumed considered unweighted version of the problem [86]. Recall that, for a bipartite graph $H = (B, S, E)$, we address vertices in B as *bought* jobs and assign them negative weights (prices) p_i, and vertices in S with a non-negative weights p_i are called *sold* jobs. Bipartite graph H encodes the precedence constraints inherent in the production: an edge from $j \in S$ to $i \in B$ implies that item i must be bought before item j can be produced and sold. At each step $1 \leq r \leq n$ of the process, let $j_1, j_2, ..., j_r$ be the jobs processed thus far, and let $bg_r = \sum_{i=1}^{r} p_{j_i}$ be the total budget up to this point. Our goal is an ordering that respects the precedence constraints and keeps the minimal value of bg_r as high as possible.

Exponential-time Exact Algorithm The previous best exact algorithm for the molecular folding problem on circle bipartite graphs has running time $n^{O(K)}$, where K is value of the optimal budget [86]. We observe that K is $\Omega(n)$ when vertex prices are ± 1 (and can be much larger when vertex prices are arbitrary), as follows. Let $\mathcal{P}$ be a projective plane of order $p^2 + p + 1$ with p prime. The projective plane of order $n = p^2 + p + 1$ consists of n lines each consisting of precisely $p + 1$ points, and n points so that each of them is intersected by precisely $p + 1$ lines. We construct a bipartite graph with each vertex in B corresponding to a line from the projective plane, each vertex in S corresponding to a point from the projective plane, and a connection from $b \in B$ to $s \in S$ if the projective plane point corresponding to s is contained in the line corresponding to $b \in B$. Vertices in B are given weight -1, and vertices in S are given weight 1. Note that the degree of each vertex in B is $p+1$. One can observe that the neighbourhood of every set of $p+1$ vertices in S is at least $p^2 - \binom{p}{2}$. This implies that in order to be able to sell the first $p+1$ vertices in S the budget decreases by at least $p^2 - \binom{p}{2} + p \geq \frac{n}{2}$. It motivates us to give an exact algorithm for arbitrary bipartite graphs (Section 5.2).

Theorem 17. *Given a bipartite graph $H = (B, S, E)$, the bipartite graph ordering problem on H can be solved in (a) time and space $O^*(2^n)$, and (b) time $O^*(4^n)$ and polynomial space, where $n = |B \cup S|$ and $O^*(f(n))$ is shorthand for $O(f(n) \cdot \text{poly}(n))$.*

Polynomial-time Cases Our main result gives a general technique that can be used to solve the bipartite graph ordering problem for classes of bipartite graphs that satisfy certain properties. We apply the technique to a number of bipartite graph classes. The bipartite graphs we consider here have been considered for other types of optimization problems. In particular *convex bipartite graphs* (those for which there exists an ordering of the vertices in B where the neighborhood of each vertex in S is a set of consecutive vertices) are of interest in biology [2, 66] and also energy production applications where resources (in our case bought vertices) can be assigned (bought) and used (sold) within a number of successive time steps [57]. There are several recognition algorithms for convex bipartite graphs [51, 70]. A bipartite graph is called *trivially perfect* if it is obtained from a union of two trivially perfect bipartite graphs H_1, H_2 or by joining every sold vertex in trivially perfect bipartite graph H_1 to every bought vertex in trivially perfect bipartite graph H_2. A single vertex is also a trivially perfect bipartite graph. These bipartite graphs have been considered in [17, 32, 72]. *Co-bipartite graphs* have a similar definition with a slightly different join operation. See Sections 5.5 and 5.6 for the precise definitions.

For trivially perfect bipartite graphs and co-bipartite graphs, due to the recursive nature of the definition of these graphs it is natural to attempt a divide and conquer strategy. However, a simple approach of solving sub-problems and using these to build up to a solution of the whole problem fails because one may need to consider all possible orderings of combining the sub-problems. The main difficulty to overcome in proving Theorem 18 is to determine which sub-graph to process first, and we develop a general approach that applies to the graph classes mentioned above. Note that each of our results holds for weighted version of the problem. We frame our algorithm in a way amenable to extending the result to include other graph classes, with the hope of fully characterizing the graphs for which our problem is polynomially solvable.

Theorem 18. *Given a bipartite graph $H = (B, S, E)$, the bipartite graph ordering problem on H can be solved in polynomial time if H is one of the following: a convex bipartite graph, a trivially perfect bipartite graph, a co-bipartite graph or a tree.*

5.1 Some Simple Classes of bipartite graphs

In this section we state some simple facts about the bipartite graph ordering problem and give a simple self-contained proof that the problem can be solved for trees. We provide this section to assist the reader in developing an intuition for the problem. Note that from now on we assume that each sold vertex is processed as soon as it can be.

Bicliques First we note that if H is a biclique with $|B| = K$ then $bg(H)$ (the budget required to process H) is K.

As the next step, consider a disjoint union of bicliques $H_1, H_2, \ldots, H_m$ where each H_i is a biclique between bought vertices B_i and sold vertices S_i. Intuitively we should first process those H_i with $|S_i| \geq |B_i|$. This is indeed correct as formalized in Lemma 12 in Section 5.3. After processing H_i with $|S_i| \geq |B_i|$, which we call *positive* (formally defined in Section 5.3), we are left with bicliques $H_i = (B_i, S_i)$ where $|B_i| > |S_i|$. Up to this point we may have built up some positive budget.

In processing the remaining H_i the budget steadily goes down — because the H_i are bicliques and disjoint, and the remaining sets are not positive. As we shall see momentarily, we can process those H_is with the largest $|S_i|$ first in order to minimize $bg(H)$. Consider an optimal strategy *opt*. Suppose on the contrary that $1 \leq i \leq m$ is the first index in *opt* where $|S_i| > |S_{i+1}|$ but *opt* processes H_{i+1} right before H_i. If K is the budget before this step, we first have that $K - |B_{i+1}| + |S_{i+1}| \geq |B_i|$ because otherwise there would not be sufficient budget after processing H_{i+1} to process H_i. Since we assumed that $|S_i| > |S_{i+1}|$ we have $K - |B_i| + |S_i| > |B_{i+1}|$. Thus, we could first process H_i and then H_{i+1}. We have thus given a method to compute an optimal strategy for a disjoint union of bicliques: first process positive sets, and then process bicliques in decreasing order of $|S_i|$.

Paths and Cycles We next consider a few even easier cases. Note that a simple path can be processed with a budget of at most 2, and a simple cycle can be processed with a budget of 2 in un-weighted version of the problem.

Trees and Forests Next we assume the input graph is a tree and the weights are $-1, 1$ (for vertices in B and S, respectively). Let H be a tree, or in general a forest. Note that any leaf has a single neighbor (or none, if it is an isolated vertex). We can thus immediately process any sold leaf s by processing its parent in the tree and then processing s. This requires an initial budget of only 1. After repeating the process to process all sold leaves in S, we are left with a forest where all leaves are bought vertices in B. We can first remove from consideration any disconnected bought vertices in B (these can, without loss of generality, be processed last). We are left with a forest H'.

We next take a sold vertex s_1 (which is not a leaf because all sold leaves in S have already been processed) and process all of its neighbours. After processing neighbours of

s_1 we can process s_1 and return 1 unit to the budget. Note that because H' is a forest, the neighbourhood of s_1 has intersection at most 1 with the neighbourhood of any other sold vertex in S. Because we have already processed all sold leaves from H, we know that only s_1 can be processed after processing its neighbours.

After processing s_1, we may be left with some sold leaves in S. If so, we deal with these as above. We note that if removing the neighbourhood of s_1 does create any sold leaves, then each of these has at least one bought vertex in B that is its neighbour and is not the neighbour of any of the other sold leaves in S. When no sold leaves remain, we pick a sold vertex s_2 and deal with it as we did with s_1.

This process is repeated until all of H' is processed. We note that after initially dealing with all sold leaves in S, we gain at most a single sold leaf at a time. That is, the budget initially increases as we process sold vertices and process their parents in the tree, and then the budget goes down progressively, only ever temporarily going up by a single unit each time a sold vertex is processed. Note that the budget initially increases, and then once it is decreasing only a single sold vertex is processed at a time. This implies that the budget required for our strategy is $|B| - |S| + 1$, the best possible budget for a graph with $1, -1$ weights.

5.2 An Exponential-time Exact Algorithm

A vertex ordering on graph $H = (B, S, E)$ is a bijection $\pi : B \cup S \to \{1, 2, \ldots, |B \cup S|\}$. For a vertex ordering π and $v \in B \cup S$, we denote by $\pi_{\preceq,v}$ the set of vertices that appear before v in the ordering (including v itself). More formally, $\pi_{\preceq,v} = \{u \in B \cup S \mid \pi(u) \leq \pi(v)\}$.

As it was mentioned in Section 2.1, the authors in [12] show that any vertex ordering problem on graphs of form $\min\limits_{\pi \in \Pi(G)} \max\limits_{v \in V} f(G, \pi_{\prec,v}, v)$ can be solved in both (a) time and space $O^*(2^n)$, and (b) time $O^*(4^n)$ and polynomial space, where n is the number of vertices in the graph and $O^*(f(n))$ is shorthand for $O(f(n) \cdot poly(n))$ (Theorems 2 and 3 in Chapter 2.1). We show that our ordering problem has the form needed to apply this result.

Let $\Pi(Q)$ be the set of all valid orderings of a set $Q \subseteq B \cup S$ and f be a function that maps each pair consisting of a graph $H = (B, S, E)$ and a vertex set Q to an integer as follows:

$$f(H, Q) = |Q \cap S| - |Q \cap B|.$$

Note that the function f is polynomially computable. Now, if we restrict the weights of vertices to be ± 1 (vertices in B have weight -1 and vertices in S have weight 1) we can express the bipartite graph ordering problem as follows:

$$bg(H) = \min_{\pi \in \Pi(B \cup S)} \max_{v \in B \cup S} |f(H, \pi_{\prec,v})|.$$

The right hand side of this equation is in the form required to apply the result of [12], giving Theorem 17 for the case of ± 1 weights. The result for arbitrary weights p_i, with p_x negative for $x \in B$ and p_y non-negative for $y \in S$, follows by setting $f(H, Q) = \sum\limits_{y \in Q \cap S} p_y - \sum\limits_{x \in Q \cap B} p_x$.

5.3 Definitions and Concepts

In this section we define key terms and concepts that are relevant to algorithms that solve the bipartite graph ordering problem on general bipartite graph. We use the graph in Figure 5.1 as an example to demonstrate each of our definitions. The reader is encouraged to consult the figure while reading this section. Recall that bipartite graph $H = (B, S, E)$ encodes the precedence constraints inherent in the production: an arc from $j \in S$ to $i \in B$ implies that item i must be bought before item j can be produced and sold. At each step $1 \leq r \leq n$ of the process, let $j_1, j_2, ..., j_r$ be the jobs processed thus far, and let $bg_r = \sum_{i=1}^{r} p_{j_i}$ be the total budget used up to this point. Our goal is an ordering that respects the precedence constraints and keeps the maximal value of bg_r as small as possible.

Let I be a set of vertices. $|I|$ refers to the cardinality of set I. When applying our arguments to weighted graphs, with vertex x having price p_x, we let $|I|$ to be $\sum_{x \in I} |p_x|$. Each of our results holds for weighted graphs by letting $|I|$ refer to the weighted sum of prices of vertices in I in all definitions and arguments.

We use K to denote the budget or capital available to process an input bipartite graph. As vertices are processed, we let K denote the current amount of capital available for the rest of the graph.

Definition 10. *Let $H = (B, S, E)$ be a bipartite graph. For a subset $I \subseteq B$ of bought vertices in H, let $N^*(I)$ be the set of all vertices in S whose entire neighborhood lie in I.*

Definition 11. *We say a set $I \subseteq B$ is* prime *if $N^*(I)$ is non-empty and for every proper subset $I' \subset I$, $N^*(I')$ is empty.*

Note that the bipartite graph induced by a prime set I and $N^*(I)$ is a bipartite clique. For any strategy to process an input bipartite graph H, we look at the budget at each step of the algorithm. Suppose our initial budget is K. Knowing which subsets of B are prime, one can see that every optimal strategy can be modified to start with processing a prime subset (Lemma 11). This leaves a budget of $K - |I| + |N^*(I)|$ to process the rest of the bipartite graph. An example for prime sets is given in Figure 5.1. For the given graph prime sets are $\{J_1, J_2\}$, J, I, I_1 with $N^*(\{J_1, J_2\}) = D$, $N^*(J) = F$, $N^*(I) = L$, and $N^*(I_1) = Q$.

Lemma 11. *There is an optimal strategy for* Bipartite Ordering Problem *on bipartite graph $H = (B, S, E)$ without isolated vertex, that starts with a prime set.*

Proof. Let π be an optimal strategy that does not start with a prime set. Suppose $2 \le i \le n$ is the first position in π where $\pi(i) \in S$ and $M = \{\pi(1), \pi(2), \ldots, \pi(i-1)\}$. Let set $I \subseteq M$ be the smallest set with $N^*(I) \neq \emptyset$. Note that such a set I exists since all the adjacent vertices to $\pi(i)$ are among vertices in M. Observe that changing the processing order on vertices in M does not harm optimality. Therefore, we can change π by processing vertices in I at first, without changing the budget. In addition, we can process $N^*(I)$ immediately after processing I. $\square$

Our algorithm will generally try to first process subsets I that increase (or at least, do not decrease) the budget. We call such subsets *positive*, and call I *negative* if processing it would decrease the budget.

Definition 12. *A budget of $I \subset B$ is the minimum budget r needed to process $H[I \cup N^*(I)]$, denoted by $bg(H[I \cup N^*(I)]) = r$. For simplicity we write $bg(I) = r$ if H is clear from the context.*

Definition 13. *A set $I \subseteq B$ is called* positive *if $|I| \le |N^*(I)|$ and it is* negative *if $|I| > |N^*(I)|$. For a given budget K, I is called* positive minimal *(with respect to budget K) if it is positive, I has budget at most K, and every other positive subset of I has budget more than K. In other words, I is smallest among all the subsets of I that is positive and has budget at most K.*

For the given graph in Figure 5.1, I_1 is the only positive minimal set and $N^*(I_1) = O$ contains 7 vertices. Note that, in general, there can be more than one positive minimal set. Positive minimal sets are key in our algorithms for computing the budget because these are precisely the sets that we can process first, as can be seen from Lemma 12. In the graph of Figure 5.1, the positive set I_1 would be the first to be processed.

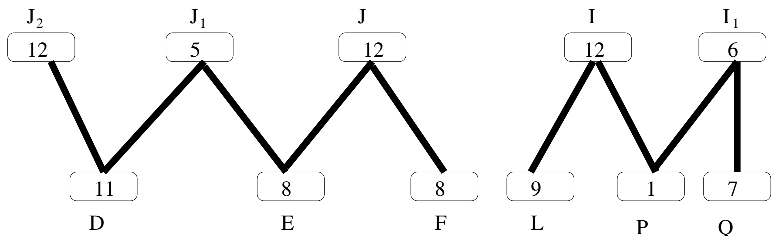


Figure 5.1: A convex graph that we use as an example for the definitions related to our algorithm. Each bold line shows a complete connection, i.e. the induced sub-graph by $I \cup L$ is a biclique. The numbers in the boxes are the number of vertices. The sets J_1, J_2, J, I, I_1 are the items B to be bought, with each vertex having weight -1. The sets D, E, F, L, P, O are the items S to be sold, with each vertex having weight 1.

Lemma 12. *Let $H = (B, S, E)$ be a bipartite graph that can be processed with budget at most K. If H contains a positive minimal set (with respect to K) then there is a strategy* *for H with budget K that begins by processing a positive minimal subset I such that for all $I' \subset I$ we have $|N^*(I')| - |I'| \leq |N^*(I)| - |I|$.*

Proof. Let I be a positive minimal set in H. Suppose the optimal process *opt* does not process I all together and hence processes the sequence $L_1, I_1, L_2, I_2, \ldots, L_t, I_t, L_{t+1}$ of disjoint subsets of B where $I = I_1 \cup I_2 \cup ... \cup I_t$ is a positive minimal set and $L_j \neq \emptyset$, $2 \leq j \leq t$. Note that according to *opt* for all L_i, $1 \leq i \leq t+1$ we have $bg(L_i) \leq K - |S_2| + |N^*(S_2)|$ in graph $H \setminus S_2$ where $S_2 = \cup_{j=1}^{i-1} L_j \cup \cup_{j=1}^{i-1} I_j$. First consider the case when $|N^*(\cup_{j=1}^{i-1} I_j)| - |\cup_{j=1}^{i-1} I_j| \leq |N^*(I)| - |I|$. Let $S_1 = \cup_{j=1}^{i-1} L_j$. For this case we have

$$\begin{aligned}
K - |S_2| + |N^*(S_2)| &= K - |S_1| - |\cup_{j=1}^{i-1} I_j| + |N^*(S_2)| \\
&= K - |S_1| + |N^*(S_1)| - |\cup_{j=1}^{i-1} I_j| + |N^*(\cup_{j=1}^{i-1} I_j)| + \\
&|N^*(S_2) \setminus (N^*(S_1) \cup N^*(\cup_{j=1}^{i-1} I_j))| \\
&\leq K - |S_1| + |N^*(S_1)| - |I| + |N^*(I)| + \\
&|N^*(S_2) \setminus (N^*(S_1) \cup N^*(\cup_{j=1}^{i-1} I_j))| \\
&\leq K - |I \cup S_1| + |N^*(I \cup S_1)|
\end{aligned}$$

Therefore $bg(L_i)$ in graph $H_1 = H \setminus (I \cup S_1 \cup N^*(I \cup S_1))$ is at most $K - |I \cup S_1| + |N^*(I \cup S_1)|$. Together with $bg(I) \leq K$, we conclude that, there is another optimal process that considers I first and then $L_1, L_2, \ldots, L_t, L_{t+1}$ next.

Now consider the case when $|N^*(\cup_{j=1}^{i-1} I_j)| - |\cup_{j=1}^{i-1} I_j| \geq |N^*(I)| - |I|$. Note that since I is a positive minimal set then processing $H[\cup_{j=1}^{i-1} I_j \cup N^*(\cup_{j=1}^{i-1} I_j)]$ needs budget more than K. On the other hand, *opt* processes $H[S_2]$ with budget at most K. Therefore, during processing $H[S_2]$ there exists a $1 \leq \beta \leq i-1$ such that $\cup_{j=1}^{\beta} L_j \cup \cup_{j=1}^{\beta} I_j$ is a positive set. Minimum such t gives us a positive minimal set. It completes the proof. ☐

Given a bipartite graph H, Lemma 12 suggests a basic strategy, if there are positive sets, find a positive minimal subset I, process it. When a given subset I is processed, we would consider the remaining bipartite graph and again try to find a positive minimal subset to process, if one exists. Note that $H \setminus (I \cup N^*(I))$ may have positive sets even if H does not. For example, in the graph of Figure 5.1, $H' = (J_2 \cup J_1 \cup J, D \cup E \cup F)$ has no positive set, but J is positive in $H' \setminus J_1$. When a subset $I \subseteq B$ is processed we generally would like to process any sets that are positive in the remaining bipartite graph. That is, we would like to process $c\ell(I)$, defined below. For our purpose we order all the prime sets lexicographically, by assuming some ordering on the vertices of B.

Definition 14. *Given current budget K and given $I \subseteq B$, let $c\ell_K(I) = \cup_{i=1}^{r} I_i \cup I$ where each $I_i \subseteq B$, $1 \leq i \leq r$ is the lexicographically first positive minimal subset in $H_i = H \setminus (\cup_{j=0}^{i-1} I_j \cup N^*(\cup_{j=0}^{i-1} I_j))$ ($I_0 = I$) such that in H_i we have $bg(I_i) \leq K - |\cup_{j=0}^{i-1} I_j| + |N^*(\cup_{j=0}^{i-1} I_j)|$. Here*

r is the number of times the process of processing a positive minimal set can be repeated after processing I.

When the initial budget K is clear from context, we use $c\ell(I)$ rather than $c\ell_K(I)$. Note that $c\ell(I)$ could be only I, in this case $r = 0$. For instance consider Figure 5.1. In the graph induced by $\{J, J_1, J_2, I, D, E, F, L, P\}$ we have $c\ell(J) = J \cup J_1$ with respect to any current budget K at least 12.

5.3.1 General Strategy

It may not always be the case that all positive sets can be identified in polynomial time. But, if positive sets can be identified, the following is a general strategy for processing an input bipartite graph H and given budget K.

1. If there exist positive sets in B, process a positive minimal set I, set $H = (B \setminus I, S \setminus N^*(I))$, update K to $K - |I| + |N^*(I)|$ and repeat step 1.

2. If no positive set exists, choose in some way the next prime set I to process, set $H = (B \setminus I, S \setminus N^*(I))$, update K to be $K - |I| + |N^*(I)|$ and goto step 1.

Note that each time a prime set I is processed, we end up processing $c\ell(I)$. Even if we can identify the prime and positive sets, it remains to determine in the second step the method for choosing the next prime to process. We address this issue and give the full algorithm and proof for Theorem 18 in the next section. Note that Lemma 12 implies that without loss of generality we can assume that when a prime set I is processed the remainder of $c\ell(I)$ is processed next, as it is stated in the next corollary.

Corollary 1. *Let $H = (B, S)$ be a bipartite graph that can be processed with budget at most K with an ordering that processes prime set I first. Then there is a strategy for H that processes $c\ell(I)$ first and uses budget at most K.*

5.4 Algorithm and Correctness of Proof for Theorem 18

In this section we give the algorithm and proof for Theorem 18, that we can solve the bipartite graph ordering problem for some classes of graphs. From the previous section it remains to determine how to choose which prime set to process first when there are no positive sets that can be processed.

Definition 15. *Let I, J be prime subsets. We say that I is* potentially after *J for current budget K if*

1. *$|I| > K$, or*

2. *$bg(c\ell(J) \setminus c\ell(I)) > K - |c\ell(I)| + |N^*(c\ell(I))|$*

Definition 15 is a first attempt at choosing which prime set to process first. The idea is to consider whether it is possible to process I before J. Item 2 in the definition states that J could not be processed immediately after I. However, this formula is not sufficient in general because we must consider orderings that do not process I and J consecutively, and we must take into account that for whatever ranking we define on the prime sets the ranking may change as the algorithm processes prime sets. For clarification we have singled out the case when I and J are processed consecutively in the proof of correctness of the algorithm.

If two prime sets I and J are not processed consecutively by the *opt* strategy, we should adapt Item 2 of Definition 15 to take into account all vertices that would be processed in between by our algorithm. We call this set of vertices the "Superset" of J with respect to I, defined precisely by the recursive Definitions 16 and 17.

Definition 16. *Let I and J be two prime subsets. For current budget K, the* Superset of J with respect to I, denoted as $Superset_I(J)$, *is defined as follows. $Superset_I(J)$ contains $c\ell(J)$ and at each step a set $c\ell(J_i)$ is added into $Superset_I(J)$ from $B \setminus Superset_I(J)$ where J_i is first according to the lexicographical order of prime sets such that no prime set is before J_i according to the ordering in Definition 17. We stop once $c\ell(I)$ lies in $Superset_I(J)$.*

Definition 17. *For current budget K, we say prime subset I is* after *prime subset J if*

1. *$|I| > K$, or*
2. *$bg(Superset_I(J) \setminus c\ell(I)) > K - |c\ell(I)| + |N^*(c\ell(I))|$*

Definition 17 states that I is after J if it is too large for the current budget (Item 1) or cannot be processed before J using the ordering implied by Definitions 16 and 17 (Item 2). Note that if I is processed right after $c\ell(J)$ then Item 2 in Definition 17 agrees with Definition 15. In the graph induced by $\{J, J_1, J_2, I, D, E, F, L, P\}$ in Figure 5.1, we have $Superset_I(J) = J \cup J_1 \cup J_2$ with respect to any current budget K at least 12.

We point out that Definitions 16 and 17 are recursive, and a naive computation of the ranking would not be efficient. We describe how to efficiently compute the ranking for the classes of graphs of Theorem 18 using dynamic programming in Sections 5.5 and 5.6. The main description of our general strategy is given in Algorithm 8.

Algorithm 8 Budget ($H = (B, S, E)$, K)

```
1: Input: $H = (B, S, E)$ and budget $K$
2: Output: "True" if we can process $H$ with budget at most $K$, "False" otherwise.
3: if $S = \emptyset$ and $K \geq 0$ then
4:     return True
5: if $|I| > K$ for all prime $I \subseteq B$ then
6:     return False
7: if there is a positive minimal subset $I$ with $bg(I) \leq K$ then
8:     return Budget $(H[B \setminus I, S \setminus N^*(I)], K - |I| + |N^*(I)|)$
9: if a positive set $I$ with the smallest budget has $bg(I) > K$ then
10:    return False
11: Let $I$ be the lexicographically first prime subset with no other prime set before it
    according to ordering in Definition 17.
12: if no such $I$ exists then
13:    return False
14: else
15:    return Budget $(H[B \setminus I, S \setminus N^*(I)], K - |I| + |N^*(I)|)$
```

The algorithm determines whether $bg(H) \leq K$. Note that the exact optimal value can be obtained by using binary search, and since the optimal value is somewhere between 0 and $|B|$ the exact computation is polynomial if the decision problem is polynomial.

Before considering the running time for the graph classes of Theorem 18 we first demonstrate that the algorithm in Algorithm 8 decides correctly, though possibly in exponential time, for any instance $(H = (B, S), K)$.

Lemma 13. *For any K and bipartite graph H, the Budget algorithm (Algorithm 8) correctly decides if $bg(H) \leq K$ or not.*

Proof. We show that if $bg(H) = K$ then there exists an optimal solution opt' with budget K in which subset I as described in the algorithm is processed first. We use induction on the size of B, meaning we assume that for smaller instances, there is an optimal process that considers the prime subsets according to the rules of our algorithm.

Correctness of Lines 3–4 is clear and correctness of Lines 5–6 follow by Lemma 11. The correctness of steps 7–8 follow from Lemma 12. Suppose Lines 9– 10 were incorrect. Then all positive subsets would have budget above K. Let I^+ be one such subset and yet if Lines 9–10 were incorrect there would be a way to process I^+ with budget at most K in H. In that case, we would process some negative set I^- which somehow reduces the budget of processing I^+; this can only be so if $I^- \cap I^+ \neq \emptyset$. In this case the following Claim 2 states that $I^+ \cup I^-$ is itself a positive set with budget at most K, a contradiction to the premise of Step 9–10.

Claim 2. *Suppose that I^+ is a positive subset with $bg(I^+) > K$ and I^- is a negative subset where $bg(I^-) \leq K$ and $I^+ \cap I^- \neq \emptyset$. If $bg(I^+ \cup I^-) \leq K$ then $I^+ \cup I^-$ forms a positive subset.*

Proof. Let $X = I^- \cap I^+$. By the assumption that I^+ can be processed after processing I^- we have $bg(I^+ \setminus X) \leq K - |I^-| + |N^*(I^-)|$. On the other hand, since $bg(I^+) > K$ then $K - |X| + |N^*(X)| < bg(I^+ \setminus X)$. From these two we can conclude that:

$$|N^*(I^-)| > |I^-| - |X| \tag{5.1}$$

Moreover, because I^+ is a positive set then $|N^*(I^+)| \geq |I^+|$. By (5.1), I^+ being positive, and the fact that $|N^*(S \cup T)| \geq |N^*(S)| + |N^*(T)|$ for any S and T, we have $|N^*(I^+ \cup I^-)| \geq |N^*(I^+)| + |N^*(I^-)| \geq |I^+| + |I^-| - |X| = |I^+ \cup I^-|$, i.e., $I^+ \cup I^-$ is a positive subset. □

We are left to verify Lines 12–15, so we continue by assuming there are no positive subsets. Let I be the first prime set according to Definition 17. Suppose for the sake of contradiction that the optimal solution *opt* processes prime subset J before I. In what follows we show that we can modify *opt* and process I as the first prime set. Note that, since there is no positive subset at the beginning, *opt* processes $c\ell(J) \setminus J$ after J.

Case 1. Suppose that by induction hypothesis (rules of our algorithm) the *opt* would place $I \setminus c\ell(J)$ first in $H' = (B \setminus c\ell(J), S \setminus N^*(c\ell(J)))$. In this case $Superset_I(J) \setminus c\ell(I)$ is just $c\ell(J) \setminus c\ell(I)$, and in this case Definitions 15 and 17 coincide.

We show that we can modify *opt* to process $c\ell(I)$ first and then $J \setminus c\ell(I)$ next while still using budget at most K. Suppose this is not the case. Now we have the following

(a) $K - |c\ell(J)| + |N^*(c\ell(J))| \geq bg(c\ell(I) \setminus c\ell(J))$

(b) $bg(c\ell(J) \setminus c\ell(I)) > K - |c\ell(I)| + |N^*(c\ell(I))|$

The inequality (a) follows by *opt* being an ordering with budget at most K that processes $c\ell(J)$ first: since *opt* processes $c\ell(I) \setminus c\ell(J)$ after $c\ell(J)$, the budget must be at least $bg(c\ell(I) \setminus c\ell(J))$ after processing $c\ell(J)$. The inequality (b) follows from the assumption that we cannot process $c\ell(I)$ first and then immediately processing $J \setminus c\ell(I)$. However, this is a contradiction to the fact that I is before J according to Definition 17. We also note that since $bg(c\ell(J)) \leq bg(Superset_I(J) \setminus c\ell(I)) \leq K - |c\ell(I)| + |N^*(c\ell(I))|$, we can also process the entire $c\ell(J)$ after processing $c\ell(I)$. Therefore we can exchange processing $c\ell(I)$ with $c\ell(J)$ and follow the *opt* in the remaining.

Case 2. We are left with the case that $c\ell(J)$ is processed first by *opt*, and the rules of the algorithm (second item in Definition 17) would process some prime subset L different from $I \setminus c\ell(J)$ next. This would imply that there is some prime subset L that is considered before the last remaining part of I in $B \backslash c\ell(J)$. By induction hypothesis we may assume that the *opt* processes the prime subsets according to the second item in Definition 17. These would imply L is in $Superset_I(J)$. At some point I or the remaining part of I becomes the first set to process according to the rules of the algorithm and this happens at the last step of the definition of $Superset_I(J)$. However, since there is no other prime subset before I according to Definition 17 we have $bg(Superset_I(J) \setminus c\ell(I)) \leq K - |c\ell(I)| + |N^*(c\ell(I))|$. Therefore we can process $c\ell(I)$ first and next $c\ell(J) \setminus c\ell(J)$ and then follow *opt*.

It remains to show that if $bg(H) \leq K$ then there exists a prime subset I that is the lexicographically first prime subset with no other prime set before it according to the ordering in Definition 17. Suppose there exists an ordering for H with budget at most K as follows: $c\ell(J), c\ell(J_1), \ldots, c\ell(J_r)$. By induction assume that the Budget Algorithm returns "true" for instance $H \setminus \{c\ell(J) \cup N^*(c\ell(J))\}$ with budget $K - |c\ell(J)| + |N^*(c\ell(J))|$ and the output ordering is $c\ell(J_1), \ldots, c\ell(J_r)$. Therefore, by Definition 16, $Superset_{J_i}(J) = \cup_{t=1}^{i} c\ell(J_t)$ for $1 \leq i \leq r$. Observe that J is not after any prime subset by Definition 17 which leads us to have J as a valid "first" prime subset in H for the algorithm. □

A naive implementation of the algorithm would consider all possible orderings of prime sets to determine the ordering in step 4 of the algorithm, and in the worst-case an exponential number of sets may need to be considered to identify the prime and positive minimal sets. A careful analysis can be taken to show that the running time of the algorithm in the general case is exponential. In the next two sections we show that for the graph classes of Theorem 18 the running time is polynomial.

5.5 Polynomial Time Algorithm for Trivially Perfect Bipartite and co-bipartite Graphs

In this section we define trivially perfect bipartite graphs and co-bipartite graphs, and discuss the key properties that are used in our algorithm for solving the bipartite graph ordering problem in these bipartite graphs. In particular, it is possible to enumerate the prime sets of these graphs by looking at a way to construct the graphs with a tree of graph join and union operations.

The subclass of trivially perfect bipartite graphs called *laminar family bipartite graphs* were considered in [74] to obtain a polynomial time approximation scheme (PTAS) for special instances of a job scheduling problem. Each instance of the problem in [74] is a bipartite graph $H = (J, M, E)$ where J is a set of jobs and M is a set of machines. For every pair of jobs $i, j \in J$ the set of machines that can process i, j are either disjoint or one is a subset of the other. The trivially perfect bipartite graphs also play an important role in studying the list homomorphism problem. The authors of [32] showed that for these bipartite graphs, the list homomorphism problem can be solved in logarithmic space. They were also considered in the fixed parametrized version of the list homomorphism problem in [17].

We call these bipartite graphs "trivially perfect bipartite graphs" because the definition mirrors one of the equivalent definitions for trivially perfect graphs.

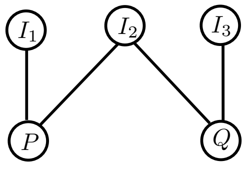
Definition 18 (trivially perfect bipartite graph, co-bipartite graph)**.** *A bipartite graph $H = (B, S, E)$ is called* trivially perfect *, respectively a* co-bipartite graph *if it can be constructed by applying the following operations.*

- *A bipartite graph with one vertex is both trivially perfect and a co-bipartite graph.*
- *If H_1 and H_2 are trivially perfect then the disjoint union of H_1 and H_2 is trivially perfect.*

  Similarly, the disjoint union of co-bipartite graphs is also a co-bipartite graph.
- *If H_1 and H_2 are trivially perfect then by joining every sold vertex in H_1 to every bought vertex in H_2, the resulting bipartite graph is trivially perfect.*

  If H_1 and H_2 are co-bipartite graphs, their complete *join—where every sold vertex in H_1 is joined to every bought vertex in H_2 and every bought vertex in H_1 is joined to every sold vertex in H_2—is a co-bipartite graph.*

An example of each type of graph is given in Figure 5.2. In the left figure (trivially perfect) $I = \{I_1, I_2\}$ and $J = \{I_2, I_3\}$ are prime sets. On the right figure (co-bipartite graph) prime sets are $R_1 = \{J_1, J_2, J_3\}, R_2 = \{J_1, J_2, J_4\}, R_3 = \{J_3, J_4, J_1\}, R_4 = \{J_3, J_4, J_2\}$ are prime sets.



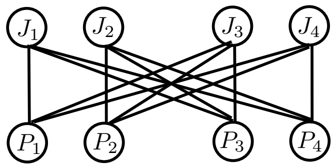


Figure 5.2: Each bold line shows a complete connection, i.e. the induced sub-graph by $I_1 \cup P$ is a biclique.

These two classes of bipartite graphs can be classified by forbidden obstructions, as follows.

Lemma 14 ([32, 48])**.** *H is trivially perfect if and only if it does not contain any of the following as an induced sub-graph: C_6, P_6.*

H is a co-bipartite graph iff it does not have any of the followings as an induced sub-graph

Our algorithm to solve $bg(H)$ for trivially perfect bipartite graphs and co-bipartite graphs centers around constructing H as in Definition 18. We view this construction as a

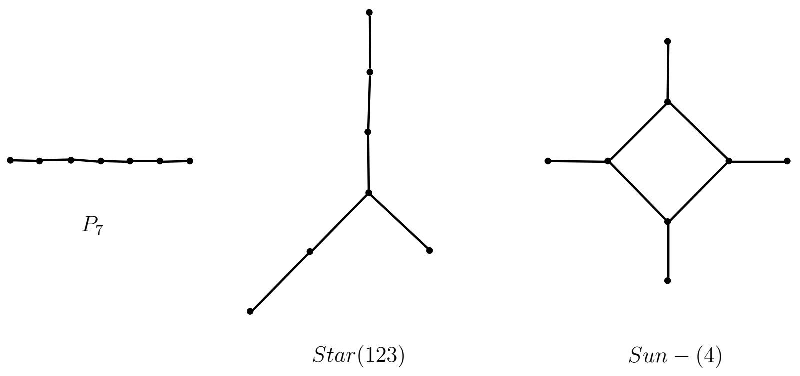


Figure 5.3: Forbidden constructions for co-bipartite graphs.

tree of operations that are performed to build up the final bipartite graph, and where the leaves of the tree of operations are *bicliques.* If H is not connected then the root operation in the tree is a disjoint union, and each of its connected components is a trivially perfect bipartite graph (respectively co-bipartite graph). If H is connected, then the root operation is a join. The following lemma shows how to find such a decomposition tree for given trivially perfect bipartite graph in polynomial time. For co-bipartite graph H a polynomial time algorithm to compute decomposition tree is given in [48].

Lemma 15. *Given a trivially perfect bipartite graph H with n vertices, there exists an algorithm that finds a decomposition tree for H in time $O(n^3)$.*

Proof. If H is not connected then the root of tree T is H and two children H_1, H_2 are chosen such that in H_1 contains all the connected component $H' = (B', S')$ of H where $|B'| < |S'|$ (if there is any) and H_2 contains all the other connected components. The root has a label "union". Note that if there exists only one such H' then $H_1 = H'$. If for every connected component of H the size of its bought vertices is smaller than the size of its sold vertices then one of them would be in H_2 and the rest lie in H_1.

If H is connected then we proceed as follows. Let $1 \leq m \leq n$ be a maximum integer such that the following test passes. Let B_2 be the set of vertices in B which have degree at least m and let $B_1 = B \setminus B_2$. Let S_1 be the set of all vertices in S that are common-neighborhood of all the vertices in B_2. If $|S_1| < m$ then the test fails. Now if there exists a vertex $v \in B_1$ such that $N(v) \not\subset S_1$ then the tests fails. If the test passes then let $S_2 = S \setminus S_1$ and let the root of T be H with label "join" and the left child of H is $H[B_1 \cup N(B_1)]$ and the right child of the root is $H_2 = H \setminus H_1$. Note that by the definition of trivially perfect bipartite graphs. If the test fails for every m then H is not trivially perfect.

We continue the same procedure from each node of the tree until each node becomes a biclique. Node that T has at most n nodes. For a particular m, checking all the condition of the test takes $O(n)$. Therefore the whole procedure takes $O(n^3)$. □

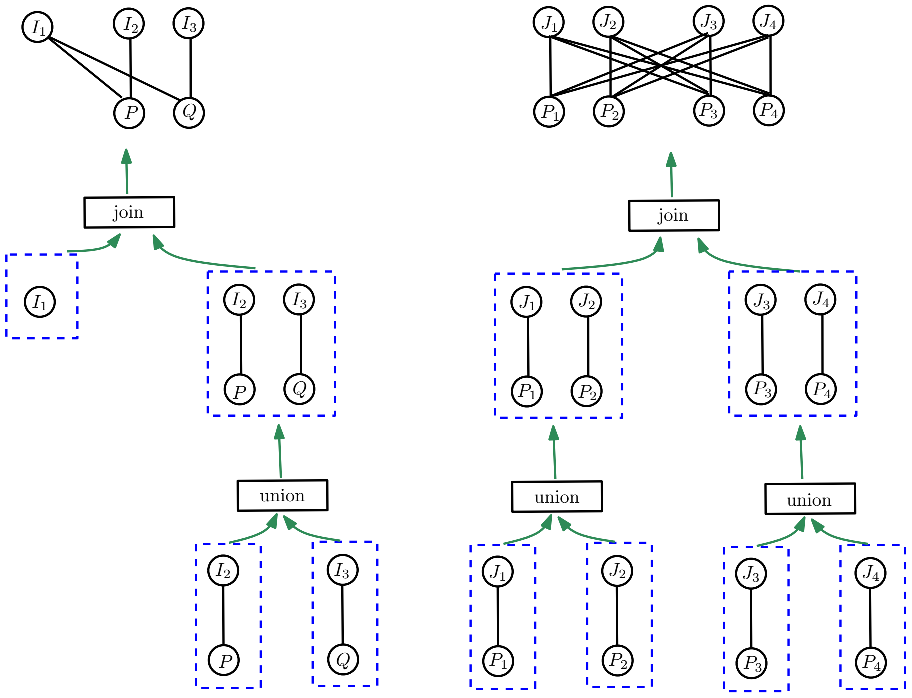


Figure 5.4: Decomposition tree associated to the graphs in Figure 5.2.

Algorithm 9 shows that how we traverse a decomposition tree in bottom-up manner and for each node of the tree we do a binary search to find the optimal budget for the graph associated to that node. Note that we assume for the graph associated to a particular node of tree the optimal budgets for its children have been computed and stored.

Algorithm 9 BudgetTPC (H, K)

```
1: Input: Trivially perfect bipartite (resp.) graph $H = (B, S, E)$, its decomposition tree $T$
2: Output: $bg(H)$
3: Start from leaves of $T$ and traverse $T$ in bottom-up manner:
4: Let $H_x = (B_x, S_x, E_x)$ be the associated graph to node $x$ of $T$
5: {Assume optimal strategies for children of $H_x$ are already computed}
6: $l = 1$ and $h = |B_x|$
7: while $l \leq h$ do
8:     if BudgetTriviallyPerfect($H_x, \lfloor \frac{l+h}{2} \rfloor$) (resp. BudgetCo-Bipartite) is True then
9:         $h = \lfloor \frac{l+h}{2} \rfloor$
10:    else
11:        $l = \lfloor \frac{l+h}{2} \rfloor + 1$
12: return l
```

In what follows we discuss how to find an optimal strategy for H when optimal strategies for H_1 and H_2 are given. First, recall that the first step of the Budget algorithm (Algorithm 8) deals with the positive minimal sets. Next we show that we can identify positive sets in polynomial time for these classes of bipartite graphs.

Lemma 16. *For given trivially perfect bipartite graph (respectively co-bipartite graph) H there is a polynomial-time algorithm that finds positive minimal sets.*

Proof. Let I be a positive minimal set and let H' be the induced sub-graph of H by $I \cup N^*(I)$. If H' is not connected then there is at least one connected component of H' that is positive, a contradiction to minimality of I. We continue by assuming H is connected. According to the decomposition of H there are $H_1 = (B_1, S_1, E_1)$ and $H_2 = (B_2, S_2, E_2)$ such that H_1 and H_2 are trivially perfect bipartite graph (respectively co-bipartite graphs) and H_1 is joined to H_2. First suppose H is a trivially perfect bipartite graph such that every bought vertex in B_1 is adjacent to every sold vertex in S_2. Observe that any positive set must include either a positive part of H_1 or all H_1 together with a positive part of H_2. In the former case, we search in H_1 for a positive set. In the later one, we search for a positive set I' in H_2 so that $|N^*(I')| - |I'| \geq |B_1| - |S_1|$. In either case, we repeat the same procedure and traverse the decomposition tree to find a positive set.

Second, suppose H is a co-bipartite graph with $H[B_1 \cup S_2]$ and $H[B_2 \cup S_1]$ are bipartite cliques. Observe that a positive set is either in $H[B_1 \cup B_2 \cup S_1]$ or in $H[B_1 \cup B_2 \cup S_2]$. In the first case, we search for a positive set I' in H_1 such that $|N^*(I')| - |I'| \geq |B_2|$. In the later case, we search for a positive set in I' in H_2 so that $|N^*(I')| - |I'| \geq |B_1|$. Therefore, we can find a positive set by repeating the same procedure and traversing decomposition tree of H. □

First we discuss our algorithm for trivially perfect bipartite graphs.In our algorithm we first deal with positive minimal sets. By the above lemma we ca traverse the decomposition tree and search for positive sets. If there exists a positive minimal set then we start processing that set. We repeat processing positive minimal sets until there dose not exist any of them. Now we are left with a trivially perfect bipartite graph that dose not contain a positive minimal set. If the graph is constructed by union operation it requires a merging function. Such a function is given in Algorithm 11. COMBINE function takes optimal solutions of two trivially perfect (respectively co-bipartite) graphs and return an optimal strategy for the union of them. In what follows, we give the description of our algorithm and prove its correctness.

Algorithm 10 BudgetTriviallyPerfect (H, K)

```
1: Input: Trivially perfect bipartite graph $H = (B, S, E)$ constructed from $H_1 = (B_1, S_1, E_1)$ and $H_2 = (B_2, S_2, E_2)$, $bg(H_1)$, $bg(H_2)$, its decomposition tree $T$, and budget $K$
2: Output: "True" if we can process $H$ with budget at most $K$, otherwise "False".
3: if $S = \emptyset$ and $K \geq 0$ or $H$ is a bipartite clique and $|B| \leq K$ then
4:   return True and process $H$
5: if $|I| > K$ for all prime $I \subseteq B$ then
6:   return False
7: if there is a positive minimal subset $I$ with $bg(I) \leq K$ then
8:   return Process $I$ and $N^*(I)$, call BudgetTriviallyPerfect $(H[B \setminus I, S \setminus N^*(I)], K - |I| + |N^*(I)|)$
9: if a positive set $I$ with the smallest budget has $bg(I) > K$ then
10:   return False
11: if $H$ is constructed by join operation between $H_1$ and $H_2$ then
12:   Assume $B_1$ and $S_2$ induce a bipartite clique i.e. $H[B_1 \cup S_2]$ is a bipartite clique.
13:   if $bg(H_1) > K$ then
14:     return False
15:   else if $bg(H_2) > K - |B_1| + |S_1|$ then
16:     return False
17:   else
18:     return True, first process $H_1$ then process $H_2$
19: if $H$ is constructed by union of $H_1$ and $H_2$ then
20:   if $bg(H_1) > K$ or $bg(H_2) > K$ then
21:     return False
22:   return Combine( $H_1, H_2, K$)
```

Algorithm 11 COMBINE (H_1, H_2, K)

```
1: Input: Optimal strategies for $H_1 = (B_1, S_1, E_1), H_2 = (B_2, S_2, E_2)$ and budget $K$
2: Output: "True" if we can process $H = H_1 \cup H_2$ with budget at most $K$, otherwise
   "False".
3: Let $J_1$ be the first prime set in $H_1$ and $J_2$ be the first prime set in $H_2$.
4: if $|J_1| > K$ or $bg(c\ell(J_2)) > K - |c\ell(J_1)| + |N^*(c\ell(J_1))|$ then
5:     Process $c\ell(J_2)$ and $N^*(c\ell(J_2))$
       Call COMBINE($H_1, H_2 \setminus (c\ell(J_2) \cup N^*(c\ell(J_2))), K - |c\ell(J_2)| + |N^*(c\ell(J_2))|$).
6: else if $|J_2| > K$ or $bg(c\ell(J_1)) > K - |c\ell(J_2)| + |N^*(c\ell(J_2))|$ then
7:     Process $c\ell(J_1)$ and $N^*(c\ell(J_1))$,
       Call COMBINE ($H_1 \setminus (c\ell(J_1) \cup N^*(c\ell(J_1))), H_2, K - |c\ell(J_1)| + |N^*(c\ell(J_1))|$).
8: else if no of the above two condition holds then
9:     return False
10: return True and processing strategy for $H$
```

Theorem 19. *For trivially perfect bipartite graphs H with n vertices the* BUDGETTRIVIALLYPERFECT *algorithm runs in $O(n^2)$ and correctly decides if H can be processed with budget K (Algorithm 10).*

Proof. The correctness of Lines 3–10 was discussed in Algorithm 8. Note that by Lemma 16, in polynomial time we can find positive minimal sets. We continue our argument by assuming that H is constructed form $H_1 = (B_1, S_1)$ and $H_2 = (B_2, S_2)$ either by "join" operation or "union" operation. First, suppose the operation between H_1 and H_2 is join so that vertices in B_1 are joint to vertices S_2 i.e $H[B_1 \cup S_2]$ is a bipartite clique. The correctness of Lines 11–18 follows from the structure of input graph $H = (B, S, E)$, that is we cannot process vertices in S_2 unless after processing all vertices in B_1.

Second, suppose the operation between H_1 and H_2 is union. In this case, we call COMBINE function. We proceed by showing the correctness of COMBINE function. Let J_1 be the first prime set in H_1 and J_2 be the first prime set in H_2. The following claim shows that that there is an optimal ordering for $H = H_1 \cup H_2$ such that either J_1 or J_2 is the first prime to process.

Claim 3. *Let $H_1 = (B_1, S_1)$ and $H_2 = (B_2, S_2)$ be two disjoint trivially perfect bipartite graphs ($H_1 \cap H_2 = \emptyset$). Suppose optimal strategies for computing the budget for H_1 and H_2 are provided. If J_1, J_2 are the first prime sets in H_1, H_2, respectively, according to Definition 17, then there is an optimal ordering for $H = H_1 \cup H_2$ such that either J_1 or J_2 is the first prime.*

Proof. Without loss of generality, we assume H_1 and H_2 are connected. The claim is correct for the case when H_1 and H_2 are bipartite cliques. Our proof is based on induction. By definition, let H_i be constructed from H_{i1} and H_{i2} by join operation where every sold vertex in H_{i2} is joint to every bought vertex in H_{i1}, for $i = 1, 2$. Accordingly, $J_1 \in H_{11}$ and $J_2 \in H_{21}$. It implies that we can reduce the problem to finding the first prime set to process either in H_{11} or in H_{21} that are trivially perfect bipartite graphs. By induction hypothesis, for $H_{11} \cup H_{12}$ the first prime is either the first prime in H_{11} or the first prime in H_{12}. It completes the proof. □

To complete the proof for correctness of Combine function, it remains to show that Lines 4–9 of Combine function correctly choose between J_1 and J_2 the first prime set to process in H. Consider $Superset_{J_2}(J_1)$. It contains $c\ell(J_1)$ and at each step a set $c\ell(J_i)$ is added into $Superset_{J_2}(J_1)$ from $B \setminus Superset_{J_2}(J_1)$ where J_i is first according to the lexicographical order of prime sets such that no prime set is before J_i according to the ordering in Definition 17. We stop once $c\ell(I)$ lies in $Superset_I(J)$. However, if it is the case when a prime set from H_2 is in $Superset_{J_2}(J_1)$, by the above claim, that prime has to be J_2. In other words, for trivially perfect graph $H = H_1 \cup H_2$, $Superset_{J_2}(J_1) = c\ell(J_1)$. As a result, correctness of Lines 4–9 follow from the correctness of Algorithm 8 where ordering primes according to their *Supersets* (Definition 17) obtains an optimal ordering.

Note, we still need to show how to keep track of $c\ell(J)$ for different prime sets J and answer queries such as $bg(c\ell(J)) \leq K$. Since all prime sets J must be leaves of the decomposition tree, there are only a polynomial number of prime sets to consider and by Lemma 15 we can determine what they are. It takes $O(n)$ time to traverse the decomposition tree. Now, it requires to update $c\ell(J)$ as we traverse the decomposition tree in bottom-up manner. Let H is associated to a node of the decomposition tree and it is constructed from H_1 and H_2 either by union or join operation. Without loss of generality, we assume there is no positive minimal set in H, as otherwise, we process them at first. Let $c\ell(J) \in B_1$. First, if the operation is union then $c\ell(J)$ does not change. Second, suppose the operation is join and every sold vertex in H_2 is adjacent to every bought vertex in H_1. If $c\ell(J)$ is the entire B_1 then for $c\ell(J) \in H$ is B_1 plus all positive minimal sets in H_2. If $c\ell(J) \subset B_1$ then in it does not change in H. Therefore, updating $c\ell(J)$ at each step takes at most $O(n)$ time. □

In what follows we show that there is a subclass of trivially perfect bipartite graphs that are also circle bipartite graphs. A bipartite graph $H = (B, S, E)$ is called a *chain graph* if the neighborhoods of vertices in B form a chain, i.e, if there is an ordering of vertices in B, say $w_1, w_2, \cdots, w_p$, such that $N(w_1) \supseteq N(w_2) \supseteq \cdots \supseteq N(w_p)$.

It is easy to see that the neighborhoods of vertices in S also form a chain. *Chain graphs* are subsets of both *trivially perfect bipartite graphs* and *circle bipartite graphs*. Any *chain graph* can be visualized as what is depicted in Figure 5.5(a), and the corresponding RNA model for the bipartite graph ordering problem looks like Figure 5.5(b).

Next we present a polynomial time algorithm for co-bipartite graphs. Our algorithm for this class of graphs is quite similar to Algorithm 10. The main difference is in the way we deal with co-bipartite graph $H = (B, S, E)$ when it is constructed from two co-bipartite

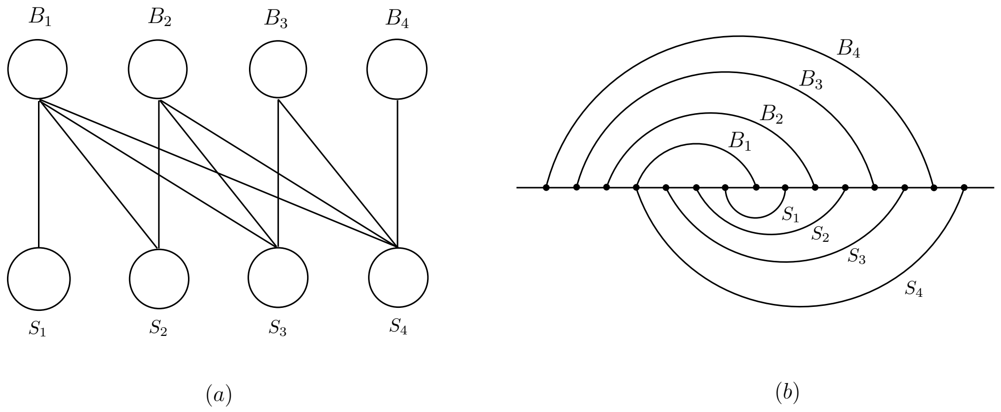


Figure 5.5: (a): Each bag B_i and S_i contains at least one vertex, for $1 \leq i \leq 4$. A line between B_i and S_j means that vertices in $B_i \cup S_j$ induce a complete bipartite graph for $1 \leq i, j \leq 4$. (b) : Each B_i and S_i arc represents a collection of arcs; the number of arcs which are represented by each B_i and S_j arc is equal to the number of vertices in bag B_i and S_i, for $1 \leq i, j \leq 4$.

graphs $H_1 = (B_1, S_1, E_1)$ and $H_2 = (B_2, S_2, E_2)$ by join operation. Recall that in join operation for co-bipartite graphs, $H[B_1 \cup S_2]$ and $H[B_2 \cup S_1]$ are bipartite cliques. Observe that in this case there are two possibilities for processing H:

- first process whole B_2 then solve the problem for H_1 with budget $K - |B_2|$, and at the end process S_2, or
- first process whole B_1 then solve the problem for H_2 with budget $K - |B_1|$, and at the end process S_1.

For the case when H is constructed from H_1 and H_2 by union operation we call COMBINE function (Algorithm 11). The description of our algorithm is given in Algorithm 12. The proof of correctness of Algorithm 12 is almost identical to the proof of Theorem 19.

Theorem 20. *Algorithm 12 in polynomial times decides if co-bipartite graph H can be processed with budget at most K.*

Algorithm 12 BudgetCo-Bipartite (H, K)

```
1: Input: Co-bipartite graph $H = (B, S, E)$ constructed from $H_1 = (B_1, S_1, E_1)$ and
   $H_2 = (B_2, S_2, E_2)$, $bg(H_1)$, $bg(H_2)$, its decomposition tree $T$, and budget $K$
2: Output: "True" if we can process $H$ with budget at most $K$, otherwise "False".
3: if $S = \emptyset$ and $K \geq 0$ or $H$ is a bipartite clique and $|B| \leq K$ then
4:   return True and process $H$
5: if $|I| > K$ for all prime $I \subseteq B$ then
6:   return False
7: if there is a positive minimal subset $I$ with $bg(I) \leq K$ then
8:   return Process $I$ and $N^*(I)$, call BudgetCo-Bipartite $(H[B \setminus I, S \setminus N^*(I)], K - |I| + |N^*(I)|)$
9: if a positive set $I$ with the smallest budget has $bg(I) > K$ then
10:  return False
11: if $H$ is constructed by join operation between $H_1$ and $H_2$ then
12:   if $bg(H_1) \leq K - |B_2|$ is True then
13:     return True and process $B_2$, process $H_1$ with budget $K - |B_2|$, process $S_2$
14:   else if $bg(H_2) \leq K - |B_1|$ then
15:     return True and process $B_1$, process $H_2$ with budget $K - |B_1|$, process $S_1$
16:   else
17:     return False
18: if $H$ is constructed by union of $H_1$ and $H_2$ then
19:   if $bg(H_1) > K$ or $bg(H_2) > K$ then
20:     return False
21:   return Combine( $H_1, H_2, K$)
```

5.6 Polynomial Time Algorithm for Convex Bipartite Graphs

A bipartite graph $H = (B, S, E)$ is called convex if there exists an ordering $\prec$ of the vertices in B such that the neighborhood of each vertex in S consists of consecutive vertices in $\prec$. We refer to a set of consecutive vertices in such an ordering as an *interval*. Figure 5.1 is an example of a convex bipartite graph. Note that the class of circle bipartite graphs $G = (X, Y)$, for which obtaining the optimal budget is NP-complete, contains the class of convex bipartite graphs.

In our algorithm for convex bipartite graphs we compute $Superset_J(I)$ for all primes I and J. Thereafter, we compare primes sets with respect to their *Superset*s and process the prime set which is first according to Definition 17. After processing the first prime we are left with a new instance that is a convex bipartite graph. For the new instance we repeat the same process. Note that the main difficulty is computing $Superset_J(I)$ for all prime sets I and J. Let $H = (B, S, E)$ be a convex bipartite graph and $\prec$ be an ordering on B as described above. We show the following lemma, which we use to prove Theorem 21.

Lemma 17. *Let v be the rightmost vertex in B according to $\prec$. Let $H_1 = (B - v, S - N(v), E_1)$ be a convex bipartite graph without positive minimal sets. Given the optimal strategy for $H_1 = (B - v, S - N(v))$, $Superset_I(J)$, for all $I, J \in H = (B, S, E)$, can be computed in time $O(n^3)$.*

Proof. Let I be the rightmost prime interval in H and $J \neq I$ be an interval in H. Without lose of generality, assume H does not have a positive minimal set. Note that $Superset_I(J)$ in H_1 and H might be different from each other.

Case 1. Consider computing $Superset_I(J)$. The first set to be added to $Superset_I(J)$ is $c\ell(J)$. After that we compare the first prime set $J_1 \in H'$ where $H' = H \setminus (c\ell(J) \cup I \cup N(I))$, assuming that we do have the order of prime sets to process in H' (this is obtained from the previously computed the optimal ordering in H'). Therefore, I should be compared with J_1 in H' according to the rules of Definition 17. Indeed, we have the same setting as the original setting that is we have the optimal strategy for convex bipartite graph H' and an interval I is added at the rightmost side of it. In this new instance we need to compute *Superset*s of all primes with respect to each other.

Case 2. Consider computing $Superset_J(I)$. Note that the first prime set added to $Superset_J(I)$ is $c\ell(I)$. Afterwards, we are left with an instance of the problem in which the neighbourhood of at most $O(n)$ intervals have been changed and because I was the rightmost interval and convexity of the graph, it requires processing the rightmost interval in the remaining instance to those changes occurred. Let $H' = H_1 \setminus c\ell(I) \cup N^*(c\ell(I))$ and I' be the rightmost prime in H'. Note that H' is a convex bipartite graph and optimal strategy for $H' \setminus I' \cup N(I')$ have been already computed. Therefore, we have a new instance with the same complexity of the original instance but with smaller size.

Case 3. Consider computing $Superset_J(J')$ where $J \neq J' \neq I$. In a similar way, $c\ell(J')$ is the first interval that is added to $Superset_J(J')$. $H' = H_1 \setminus c\ell(J') \cup N^*(c\ell(J'))$ is a convex bipartite graph with I as its rightmost prime. By our assumption optimal strategy in $H' \setminus I \cup N(I)$ is given. In the new instance H' we have the same setting as the original problem. However, the new instance has a smaller size than the original one.

Taking into account the aforementioned cases, we conclude that finding $Superset_J(I)$ for particular I and J requires $O(n)$. It is because we have $T(n) = T(n-1) + O(1)$ where $T(n)$ is the complexity for computing $Superset_J(I)$ and $O(1)$ stands for the comparison we do between two particular primes. Since there are potentially $O(n^2)$ prime intervals then we have $O(n^4)$ pairs of them. In fact the following observation shows that the total running time needed to compute *Superset*s of all pairs of prime intervals is $O(n^3)$ since we only need to compute *Superset* of $O(n^2)$ pairs of primes.

Observation 4. *Suppose $Superset_{J'}(J)$, for arbitrary prime intervals $J, J' \in B$, consists of $c\ell(J_0 = J) \cup c\ell(J_1) \cup \cdots \cup c\ell(J_r = J')$ while $c\ell(J_i)$ is added to $Superset'_J(J)$ before $c\ell(J_j)$ for $0 \leq i < j \leq r$. Then, $Superset_{J_i}(J) = c\ell(J_0 = J) \cup c\ell(J_1) \cup \cdots \cup c\ell(J_i)$.*

□

Theorem 21. *Algorithm 13 solves the* Bipartite Ordering Problem *on convex bipartite graphs in time $O(n^5)$.*

Proof. Let $H = (S, B, E)$ be a convex bipartite graph with an ordering on its vertices as described above. We start from the leftmost vertex v in B and move to the right, iteratively determining $Superset_I(J)$ and bg in larger sub-graphs of H from the right. We assume the optimal ordering for graph $H_1 = H \setminus \{v \cup N(v)\}$ has been computed according to the rules of Algorithm 13. We show how to find an optimal ordering for H following the rules of Algorithm 13.

First, we need to find all positive minimal sets. For convex bipartite graphs, prime sets, the closure of a prime set ($c\ell(I)$), and any positive minimal set is an interval; thus we must keep track of the budget needed to process intervals in B, and there are only polynomially many of these to consider (in particular, $O(n^2)$ where B has n vertices). Considering prime sets and closures of prime sets, when we consider an interval J we need to consider $N^*(J)$ together with vertices of S that are not initially in $N^*(J)$ but are initially in $N(J')$ where $N(J) \cap N(J') \neq \emptyset$. This is because after processing $J' \setminus J$ we may need to determine the budget for J in the new instance and here the neighborhood of $N^*(J)$ contains new vertices. However, the number of these new intervals is at most $O(n^2)$. This task is in fact computing the budget for a proper sub-interval J in B together with the appropriate set of vertices in S which may need to be considered in $N^*(J)$. From now on we assume we can answer the queries such as $bg(c\ell(J)) \leq K$ for a given w using a dynamic programming table.

Consider processing a positive minimal set. According to Definition 13, it does not have any proper positive subset. Therefore, it is the same as the case when we have a convex bipartite graph without any positive prime interval and no positive closure set (Definition 14). Note that once we have the *Superset*s for different prime intervals it is easy to verify whether a positive minimal set can be processed with the current budget.

We continue by assuming there is no positive prime interval in H_1. The main work here is finding $Supersets_I(J)$ for I and J being intervals in B. Once the $Supersets_I(J)$, for all $I, J \in B$, are computed we can compare them according to Definition 17 and find the first prime set to process. It is worth mentioning that $Superset_I(J)$ in H_1 might be different form $Superset_I(J)$ in H. Lemma 17 shows that the latter can be done derived from the former in time complexity $O(n^3)$ for all pairs of prime intervals.

Since we move from left to right on H, which has $|B| = n$, we repeat the above procedure n times, hence it takes $O(n^4)$ time to recognize the first prime to process. After processing the first prime, we are left with a new instance. The same procedure has to be repeated on the new instance to recognize the first prime to process in the new instance. Therefore, it takes $O(n^5)$ to process the whole graph. □

Algorithm 13 BudgetConvex (H, K)

```
1: Input: Convex bipartite graph $H = (B, S, E)$ with ordering $\prec$ on vertices in $B$ i.e.
   $b_1 \prec b_2 \prec \cdots \prec b_{|B|}$ and budget $K$
2: Output: Process $H$ with budget at most $K$, otherwise "False".
3: if $S = \emptyset$ and $K \geq 0$ or $H$ is a bipartite clique and $|B| \leq K$ then
4:   return Process $H$
5: if there is a positive minimal subset $I$ with $bg(I) \leq K$ then
6:   return Process $I$ and $N^*(I)$, call BudgetConvex ($H[B \setminus I, S \setminus N^*(I)], K - |I| +$
     $|N^*(I)|$)
7: if $|I| > K$ for all prime $I \subseteq B$ then
8:   return False
9: if a positive set $I$ with the smallest budget has $bg(I) > K$ then
10:  return False
11: for $i = 1$ to $|B|$ do
12:   Let $B_i = \{b_1, b_2, \ldots, b_i\}$ and $H_i = H[B_i \cup N^*(B_i)]$
13:   {We assume optimal strategy for $H_{i-1}$ has been computed}
14:   According to Lemma 17 find $Superset_J(I)$ for all pair of primes $I$ and $J$ in $H_i$
15: Let $I$ be the first prime to process according to Definitions 16 and 17
16: Process $I$ and $N^*(I)$
17: Call BudgetConvex ($H[B \setminus I, S \setminus N^*(I)], K - |I| + |N^*(I)|$)
```

Chapter 6

Conclusion and Open Problems

The focus of this thesis was on vertex ordering problems. In this thesis, we discussed a general formulation of vertex ordering problems. We turned our attention to three particular vertex ordering problems including Minimum Linear Arrangement (MLA) problem, Directed Feedback Arc Set (DFAS) problem, and Bipartite Ordering Problem (BOP).

In Chapter 3, we considered a variant of DFAS problem. The problem is defined on special type of digraphs. Considering DFAS on this type of digraphs is motivated by its application in computational biology. Indeed, the problem we defined is a modeling for editing species trees. We showed that, even if we restrict DFAS problem on this particular type of digraphs, the problem remains $\mathcal{NP}$-complete. For practical purposes we proposed an efficient algorithm. Simulations and experimental results show that our algorithm is more efficient than the existing simple greedy algorithm.

In Chapter 4, we discussed parameterzied complexity of MLA problem where the parameter is the size of minimum vertex cover of the input graph. In 2008, Fellows *et al.* [40] asked if the problem is fixed-parameter tractable. In [39], authors gave $(1+\epsilon)$-approximation algorithm in $\mathcal{FPT}$ time where parameters are ϵ and size of vertex cover of the input graph. In 2015, Daniel Lokshtanov [65] positively answered this open question. We formulate the problem as Quadratic Integer Programming (QIP). We noticed that Graver basis of the constrains matrix of our QIP are nicely structured and there are polynomial number of them. There are related results where QIP can be solved using Graver basis. Unfortunately, none of them can be used in our QIP. We introduced two new techniques to tackle the problem. In our first approach, we conjectured that from a non-optimal point for the QIP we can find a point with a better objective function value using augmenting step with bounded coefficients. Second, we applied KKT conditions on the QIP. We conjectured that number of points that satisfy KKT conditions is bounded and an optimal point is close enough to a point that satisfies KKT conditions.

In Chapter 5, we defined a new vertex ordering problem that can be used to model processes with precedence constraints. As with any optimization problem there are many avenues of attack. In this thesis we have focused on determining for which classes of graphs the bipartite graph ordering problem can be solved in polynomial time. Our ultimate goal in this direction is a dichotomy classification of polynomial cases and $\mathcal{NP}$-complete cases. The algorithm in the proof of Theorem 18 finds the optimal budget for all graphs H, and the algorithm was shown to run in polynomial time for the classes of graphs mentioned in Theorem 18. We pose the question whether the algorithm can be the basis of a dichotomy theorem: are there classes of graphs which can be solved in polynomial time but for which our algorithm does not run in polynomial time? As with all optimization problems the bipartite graph ordering problem can also be studied from a number of other angles, including approximation and hardness of approximation, fixed parameter algorithms, and faster exponential-time algorithms.

A particular graph class to consider in each of these areas is that of circle bipartite graphs, because these graphs are of particular interest in the application to calculate folding pathways between pairs of RNA secondary structures [47, 73, 86].

Bibliography

[1] Donald L Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6(1):40–54, 1977.

[2] Farid Alizadeh, Richard M Karp, Lee Aaron Newberg, and Deborah K Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Algorithmica*, 13(1-2):52–76, 1995.

[3] Noga Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 20(1):137–142, 2006.

[4] Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. On the approximability of single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 36(4):653–669, 2011.

[5] Christoph Ambuhl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 329–337. IEEE, 2007.

[6] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[7] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[8] Jørgen Bang-Jensen and Caresten Thomassen. A polynomial algorithm for the 2-path problem for semicomplete digraphs. *SIAM Journal on Discrete Mathematics*, 5(3):366–376, 1992.

[9] Michael J Benton and Francisco J Ayala. Dating the tree of life. *Science*, 300(5626):1698–1700, 2003.

[10] Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

[11] Hans L Bodlaender, Michael R Fellows, and Michael Hallett. Beyond np-completeness for problems of bounded width: Hardness for thew hierarchy. In *Proceedings of the 26th Annual Symposium on Theory of Computing*, pages 449–458.

[12] Hans L Bodlaender, Fedor V Fomin, Arie MCA Koster, Dieter Kratsch, and Dimitrios M Thilikos. A note on exact algorithms for vertex ordering problems on graphs. *Theory of Computing Systems*, 50(3):420–432, 2012.

[13] Pierre Charbit, Stéphan Thomassé, and Anders Yeo. The minimum feedback arc set problem is np-hard for tournaments. *Combinatorics, Probability and Computing*, 16(01):1–4, 2007.

[14] Moses Charikar, Mohammad Taghi Hajiaghayi, Howard Karloff, and Satish Rao. l 2 2 spreading metrics for vertex ordering problems. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1018–1027. Society for Industrial and Applied Mathematics, 2006.

[15] Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM (JACM)*, 55(5):21, 2008.

[16] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[17] Rajesh Chitnis, László Egri, and Dániel Marx. List h-coloring a graph by removing few vertices. In *European Symposium on Algorithms*, pages 313–324. Springer, 2013.

[18] Fan RK Chung. *Labelings of graphs.* Bell Communications Research. Morris Research and Engineering Center. Mathematical, Communications and Computer Sciences Research Laboratory, 1987.

[19] Francesca D Ciccarelli, Tobias Doerks, Christian Von Mering, Christopher J Creevey, Berend Snel, and Peer Bork. Toward automatic reconstruction of a highly resolved tree of life. *science*, 311(5765):1283–1287, 2006.

[20] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[21] José R Correa and Andreas S Schulz. Single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30(4):1005–1021, 2005.

[22] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

[23] Jesús A De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and geometric ideas in the theory of discrete optimization*, volume 14. SIAM, 2013.

[24] Frank Dehne, Michael Fellows, Michael A Langston, Frances Rosamond, and Kim Stevens. An o (2 o (k) n 3) fpt algorithm for the undirected feedback vertex set problem. In *International Computing and Combinatorics Conference*, pages 859–869. Springer, 2005.

[25] Frank Dehne, Mike Fellows, Frances Rosamond, and Peter Shaw. Greedy localization, iterative compression, and modeled crown reductions: New fpt techniques, an improved algorithm for set splitting, and a novel 2k kernelization for vertex cover. In *International Workshop on Parameterized and Exact Computation*, pages 271–280. Springer, 2004.

[26] Frédéric Delsuc, Henner Brinkmann, and Hervé Philippe. Phylogenomics and the reconstruction of the tree of life. *Nature Reviews Genetics*, 6(5):361–375, 2005.

[27] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.

[28] Emmanuel JP Douzery, Elizabeth A Snell, Eric Bapteste, Frédéric Delsuc, and Hervé Philippe. The timing of eukaryotic evolution: does a relaxed molecular clock reconcile proteins and fossils? *Proceedings of the National Academy of Sciences of the United States of America*, 101(43):15386–15391, 2004.

[29] G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

[30] Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness ii: On completeness for w [1]. *Theoretical Computer Science*, 141(1):109–131, 1995.

[31] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity.* Springer Science & Business Media, 2012.

[32] László Egri, Andrei Krokhin, Benoit Larose, and Pascal Tesson. The complexity of the list homomorphism problem for graphs. *Theory of Computing Systems*, 51(2):143–178, 2012.

[33] P Erdős and L Pósa. On the maximal number of disjoint circuits of a graph. *Publ. Math. Debrecen*, 9:3–12, 1962.

[34] Guy Even, J Seffi Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[35] Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM (JACM)*, 47(4):585–616, 2000.

[36] Shimon Even and Yossi Shiloach. Np-completeness of several arrangement problems. *Department of Computer Science, Israel Institute of Technology, Haifa, Isreal, Tech. Rep*, 1975.

[37] Uriel Feige and James R Lee. An improved approximation ratio for the minimum linear arrangement problem. *Information Processing Letters*, 101(1):26–29, 2007.

[38] Michael R Fellows, Fedor V Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011.

[39] Michael R Fellows, Danny Hermelin, Frances Rosamond, and Hadas Shachnai. Tractable parameterizations for the minimum linear arrangement problem. *ACM Transactions on Computation Theory (TOCT)*, 8(2):6, 2016.

[40] Michael R Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *International Symposium on Algorithms and Computation*, pages 294–305. Springer, 2008.

[41] Lance Fortnow and Steve Homer. A short history of computational complexity. *Bulletin of the EATCS*, 80:95–133, 2003.

[42] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[43] Alan Frieze and Ravi Kannan. The regularity lemma and approximation schemes for dense problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 12–20. IEEE, 1996.

[44] Chris P Gane and Trish Sarson. *Structured systems analysis: tools and techniques.* Prentice Hall Professional Technical Reference, 1979.

[45] Georges Gardarin and Stefano Spaccapietra. Integrity of data bases: A general lockout algorithm with deadlock avoidance. In *IFIP Working Conference on Modelling in Data Base Management Systems*, pages 395–412, 1976.

[46] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[47] Michael Geis, Christoph Flamm, Michael T Wolfinger, Andrea Tanzer, Ivo L Hofacker, Martin Middendorf, Christian Mandl, Peter F Stadler, and Caroline Thurner. Folding kinetics of large rnas. *Journal of molecular biology*, 379(1):160–173, 2008.

[48] Vassilis Giakoumakis and Jean-Marie Vanherpe. Bi-complement reducible graphs. *Advances in Applied Mathematics*, 18(4):389–402, 1997.

[49] Gregory Gutin, Arash Rafiey, Stefan Szeider, and Anders Yeo. The linear arrangement problem parameterized above guaranteed value. *Theory of Computing Systems*, 41(3):521–538, 2007.

[50] Gregory Gutin and Anders Yeo. Some parameterized problems on digraphs. *The Computer Journal*, 51(3):363–371, 2008.

[51] Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1):59–84, 2000.

[52] Lawrence H Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964.

[53] S Blair Hedges and Sudhir Kumar. *The timetree of life.* OUP Oxford, 2009.

[54] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

[55] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[56] Richard M Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 278–285. ACM, 1993.

[57] Kamyar Khodamoradi, Ramesh Krishnamurti, Arash Rafiey, and Georgios Stamoulis. Ptas for ordered instances of resource allocation problems. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 24. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[58] Andrew H Knoll. *Life on a young planet: the first three billion years of evolution on earth*. Princeton University Press, 2015.

[59] Arno Kunzmann and Hans-Joachim Wunderlich. An analytical approach to the partial scan problem. *Journal of Electronic Testing*, 1(2):163–174, 1990.

[60] Jon Lee, Shmuel Onn, Lyubov Romanchuk, and Robert Weismantel. The quadratic graver cone, quadratic integer minimization, and extensions. *Mathematical programming*, 136(2):301–323, 2012.

[61] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 422–431. IEEE, 1988.

[62] Charles E Leiserson and James B Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, 1991.

[63] Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, 8(4):538–548, 1983.

[64] L. Levin. Universal'nyie perebornyie zadachi (universal search problems: in russian). *DProblemy Peredachi Informatsii*, pages 265–266, 1973.

[65] Daniel Lokshtanov. Parameterized integer quadratic programming: Variables and coefficients. *arXiv preprint arXiv:1511.00310*, 2015.

[66] Wei-Fu Lu and Wen-Lian Hsu. A test for the consecutive ones property on noisy data-application to physical mapping and sequence assembly. *Journal of Computational Biology*, 10(5):709–735, 2003.

[67] L. C. Lucchesi. *A minimumax equality for directed graphs*. PhD thesis, University of Waterloo, 1976.

[68] Diego Mallo, Leonardo De Oliveira Martins, and David Posada. Simphy: Phylogenomic simulation of gene, locus, and species trees. *Systematic biology*, 65(2):334–344, 2016.

[69] Ján Maňuch, Chris Thachuk, Ladislav Stacho, and Anne Condon. Np-completeness of the direct energy barrier problem without pseudoknots. In *International Workshop on DNA-Based Computers*, pages 106–115. Springer, 2009.

[70] Ross M McConnell. A certifying algorithm for the consecutive-ones property. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 768–777. Society for Industrial and Applied Mathematics, 2004.

[71] Graeme Mitchison and Richard Durbin. Optimal numberings of an n*n array. *SIAM Journal on Algebraic Discrete Methods*, 7(4):571–582, 1986.

[72] Rolf H Möhring, Martin Skutella, and Frederik Stork. Scheduling with and/or precedence constraints. *SIAM Journal on Computing*, 33(2):393–415, 2004.

[73] Steven R Morgan and Paul G Higgs. Barrier heights between ground states in a model of rna secondary structure. *Journal of Physics A: Mathematical and General*, 31(14):3153, 1998.

[74] Gabriella Muratore, Ulrich M Schwarz, and Gerhard J Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters*, 38(1):47–50, 2010.

[75] Rolf Niedermeier. Invitation to fixed-parameter algorithms. 2006.

[76] Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

[77] Satish Rao and Andréa W Richa. New approximation techniques for some ordering problems. In *SODA*, volume 98, pages 211–219, 1998.

[78] R Ravi, Ajit Agrawal, and Philip Klein. Ordering problems approximated: single-processor scheduling and interval graph completion. In *International Colloquium on Automata, Languages, and Programming*, pages 751–762. Springer, 1991.

[79] Hanna Seitz. Contributions to the minimum linear arrangement problem. 2010.

[80] Farhad Shahrokhi, Ondrej Sỳkora, László A Székely, and Imrich Vrto. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001.

[81] Yossi Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal on Computing*, 8(1):15–32, 1979.

[82] Abraham Silberschatz, Peter B Galvin, Greg Gagne, and A Silberschatz. *Operating system concepts*, volume 4. Addison-Wesley Reading, 1998.

[83] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[84] Gergely J Szöllősi, Bastien Boussau, Sophie S Abby, Eric Tannier, and Vincent Daubin. Phylogenetic modeling of lateral gene transfer reconstructs the pattern and relative timing of speciations. *Proceedings of the National Academy of Sciences*, 109(43):17513–17518, 2012.

[85] Gergely J Szöllősi, Wojciech Rosikiewicz, Bastien Boussau, Eric Tannier, and Vincent Daubin. Efficient exploration of the space of reconciled gene trees. *Systematic biology*, page syt054, 2013.

[86] Chris Thachuk, Ján Manuch, Arash Rafiey, Leigh-Anne Mathieson, Ladislav Stacho, and Anne Condon. An algorithm for the energy barrier problem without pseudoknots and temporary arcs. In *Pacific Symposium on Biocomputing*, volume 15, pages 108–119, 2010.

[87] A Vannelli and GS Rowan. An eigenvector based approach for multistack vlsi layout. In *Proceedings of the midwest symposium on circuits and systems*, volume 29, pages 135–139, 1986.

[88] Vijay V Vazirani. *Approximation algorithms.* Springer Science & Business Media, 2013.

[89] David P Williamson and David B Shmoys. *The design of approximation algorithms.* Cambridge university press, 2011.

[90] Gerhard J Woeginger. On the approximability of average completion time scheduling under precedence constraints. *Discrete Applied Mathematics*, 131(1):237–252, 2003.