April 2013

NFA reduction via hypergraph vertex cover approximation

Timothy Ng
The University of Western Ontario

Supervisor
Dr. Roberto Solis-Oba
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

NFA REDUCTION VIA HYPERGRAPH VERTEX COVER APPROXIMATION

(Thesis format: Monograph)

by

Timothy Ng

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

In this thesis, we study the minimum vertex cover problem on the class of k-partite k-uniform hypergraphs. This problem arises when reducing the size of nondeterministic finite automata (NFA) using preorders, as suggested by Champarnaud and Coulon. It has been shown that reducing NFAs using preorders is at least as hard as computing a minimal vertex cover on 3-partite 3-uniform hypergraphs, which is NP-hard. We present several classes of regular languages for which NFAs that recognize them can be optimally reduced via preorders. We introduce an algorithm for approximating vertex cover on k-partite k-uniform hypergraphs based on a theorem by Lovász and explore the use of fractional cover algorithms to improve the running time at the expense of a small increase in the approximation ratio.

Keywords: regular languages, vertex cover, approximation algorithms, nondeterministic finite automata, approximate fractional covering, NFA reduction

Acknowledgements

I would like to thank Roberto Solis-Oba for his supervision, for encouraging me to work on what I was interested in, and for his advice and guidance, which helped shape this work.

I would like to thank my initial advisor, Sheng Yu, who sadly passed away, for enthusiastically guiding me through the early part of my graduate studies.

And finally, I would like to thank my family for the support they have given me throughout my studies.

Contents

Abstract **ii**

Acknowledgements **iii**

List of Figures **vi**

List of Tables **vii**

1 Introduction **1**

2 Regular Languages **4**
- 2.1 Preliminaries 4
- 2.2 Regular expressions 5
- 2.3 Finite automata 6
- 2.4 Converting regular expressions to finite automata 10

3 The Vertex Cover Problem **13**
- 3.1 Complexity theory 13
- 3.2 The minimal vertex cover problem 14
- 3.3 Approximation Algorithms 17

4 The NFA reduction problem **21**
- 4.1 Preliminaries 21
  - 4.1.1 Reducing NFAs based on equivalences 22
  - 4.1.2 Reducing NFAs based on preorders 24
- 4.2 Complexity of computing PRE-reduced NFAs 28

5 An approximation algorithm based on Lovász's theorem **34**
- 5.1 Lovász's theorem 34
- 5.2 The algorithm 37
- 5.3 Applying Lovász's theorem to graphs 41

6 Using Approximate Fractional Covers **47**
- 6.1 Approximate fractional covering algorithms 47
- 6.2 Approximate fractional vertex covers 48

7 Conclusions **52**

Bibliography **54**

Curriculum Vitae **58**

List of Figures

2.1 The DFA A, which recognizes 0,1-strings encoding even numbers 8
2.2 The NFA M accepting L_3 8
2.3 The DFA M' accepting L_3 9
2.4 The position automaton for $\alpha = aa(ba)^*(b + ba)$ 11

3.1 The hypergraph $H = (V, E)$ 15

4.1 The NFA N and the set cover instance $(Q, \Pi_R \cup \Pi_L)$ 24
4.2 EQ-reduced NFAs for the NFA N 25
4.3 The partial order $\preceq$ over Q 26
4.4 The NFA N 27
4.5 The hypergraph for the set cover instance $(Q, \pi_R \cup \pi_L \cup \pi_P)$ 27
4.6 The PRE-reduced NFA for N 27
4.7 NFAs which accept the language $\{a, b\}^2$ 28
4.8 NFAs which accept the language $(\{a, b\}^2)^*$ 28
4.9 NFAs which accept $L(b + aa^*b)$ and $L((b + aa^*b)^*)$ 30
4.10 A 1-cycle-free-path automaton which accepts the language $L(\alpha)$ 33

5.1 The hypergraph $H = (V, E)$ 39
5.2 The complete graph K_3 43

List of Tables

2.1 Table for transition function δ' 9
2.2 The table for follow(α, i) 11

Chapter 1

Introduction

Regular expressions describe regular languages and are recognized by one of the simplest theoretical models of computing, finite automata. Regular expressions are important tools in computer science which are used in many applications. The most well known uses include lexical analysis [2], pattern matching [45], computational linguistics [53], and circuit design [7]. However, over the last two decades that list has grown to include less obvious applications, including image compression [12], type theory [59], parallel processing [65], and software testing [18].

They are also of interest in more theoretical work. Automatic sequences [3] are integer sequences generated by finite automata and have applications in number theory and physics. Wolfram [67] studied the relationship between cellular automata and regular languages. Rubinstein [63] and Linster [47] use finite automata in the study of the prisoner's dilemma. Culik and Harju [11] and Head [27] use regular languages in DNA computing.

And of course, regular expressions are a fundamental topic of study in theoretical computer science. Regular languages and finite automata together comprise some of the oldest and well-studied topics in automata and formal language theory. For some time, it was believed that regular languages were exhausted as a topic of interest. However, in the early 90s, the study of regular languages was revived as regular languages gained use outside of their traditional applications as computing power became more plentiful and accessible [19].

The correspondence between regular expressions and finite automata allows us to take advantage of the strengths of both representations. Regular expressions are convenient for humans to specify and manipulate but are difficult for computers to process. On the other hand, finite automata are quite simple to implement in software, but are difficult for humans to use. This correspondence allows users to define regular languages using regular expressions and we can transform those regular expressions into finite automata for processing on computers.

Typically, such a transformation involves transforming regular expressions into equivalent nondeterministic finite automata (NFA). There are many efficient algorithms for doing so [35]. These NFAs can then converted into the equivalent minimal deterministic finite automata (DFA) for implementation as a simple and efficient structure for recognizing the language defined by the regular expression.

The main problem with this process is the size of DFAs, measured in the number of states. The size of a DFA can be exponential in the size of the equivalent NFA. As NFAs have at most the same number of states as the length of a regular expression, this means that DFAs are potentially exponential in the size of the input, which is the regular expression. This poses a challenge in terms of the memory required to store a DFA.

One way to mitigate the exponential blowup of states in the determinization process is to reduce the size of the NFA that is created from the regular expression before the determinization process. Unfortunately, the problem of minimizing a nondeterministic finite automaton is extremely difficult, specifically, it is PSPACE-complete [37]. In fact, given an n-state NFA, the problem is inapproximable to within a factor of $o(n)$ unless $\mathrm{P} = \mathrm{PSPACE}$ [23]. That is, there are no efficient approximation algorithms which can provide a solution within a factor linear in the number of states of the NFA.

Our work is based on a method to reduce NFAs by merging states based on a preorder relation defined on the set of states of the NFA [8]. It is shown in [34] that optimally reducing an NFA using preorders is at least as computationally hard as computing a minimal vertex cover on 3-partite 3-uniform hypergraphs, which is a problem known to be intractable. The vertex cover problem is an important and well-known combinatorial optimization problem, which can be extended to hypergraphs, in which hyperedges can contain more than two vertices. We are interested in efficient methods for finding good approximate solutions for the problem on the class of k-partite k-uniform hypergraphs for optimally reducing NFAs using preorders.

In this thesis, we present our approximation algorithm for the minimum vertex cover problem for k-partite k-uniform hypergraphs based on Lovász's theorem which computes vertex covers of size no greater than $\frac{k}{2}$ times the optimal solution. We also show how to use approximate fractional covers to improve the running time of our algorithm. We also examine whether there are classes of regular languages for which a PRE-reduced NFA is easy to compute or approximate. We give a partial answer to that question by presenting some examples of such classes.

In Chapter 2, we introduce regular languages and finite automata and some definitions and basic properties. In particular, we discuss the correspondence between regular languages and finite automata and in doing so, lay out the motivation for the NFA reduction problem. We introduce the NFA reduction problem and summarize prior work in approximating the problem as well as results demonstrating the hardness of the problem.

In Chapter 3, we introduce the minimum vertex cover problem and basic computational complexity definitions. We discuss why some problems are hard to compute and how we can work around those limits. We summarize how these ideas have been applied to the minimum vertex cover problem in prior work.

In Chapter 4, we introduce methods for reducing the size of NFAs. We present some results relating properties of regular languages with hardness of the NFA reduction problem. We examine hardness of approximation for certain families of regular languages and we give examples of families of regular languages for which optimal NFA reduction using preorders is computable in polynomial time.

In Chapter 5, we present our approximation algorithm kPartHypVC for the vertex cover problem on k-partite k-uniform hypergraphs, which is based on Lovász's theorem. We introduce Lovász's theorem, describe our algorithm, and discuss its complexity.

In Chapter 6, we explore the use of approximate fractional covers in place of exact fractional covers in our algorithm to improve the running time the algorithm. While linear programs are solvable in polynomial time, the time complexity of the algorithms are dependent on the number of variables to relatively high degree. By using an approximate fractional covering algorithm, the time complexity becomes dependent on the number of constraints and a small increase in the factor of approximation.

We conclude in Chapter 7 and we summarize our findings and discuss possible future work.

Chapter 2

Regular Languages

2.1 Preliminaries

An *alphabet* is a finite non-empty set of symbols. A *word* over an alphabet Σ is a finite sequence of symbols from Σ. The *length* of a word w is the number of symbols contained in w and is denoted $|w|$. The *empty word* is the word of length 0 and is denoted by ϵ.

The concatenation operation is an important operation on words, formed by juxtaposing two words together. The *concatenation* of two words $w = a_1a_2 \cdots a_m$ and $x = b_1b_2 \cdots b_n$ is the word $wx = a_1a_2 \cdots a_mb_1b_2 \cdots b_n$. Note that concatenation is not commutative in general, i.e. wx does not necessarily equal xw. For any integer $n \geq 0$ and word w over Σ, we define w^n by $w^0 = \epsilon$ and $w^n = ww^{n-1}$.

The set of all words, including ϵ, over the alphabet Σ is denoted Σ^*. A *language* L over Σ is a set of words over Σ and is a subset of Σ^*. The *empty language* is the language containing no words and is denoted $\emptyset$. As with words, we define the concatenation operation on languages. The concatenation of two languages L_1 and L_2 over Σ is defined by

$$L_1L_2 = \{w_1w_2 : w_1 \in L_1, w_2 \in L_2\}.$$

For any integer $n \geq 0$ and language L over Σ, we define L^n by $L^0 = \{\epsilon\}$ and $L^n = LL^{n-1}$. The star, or *Kleene closure*, of a language L is denoted L^* and is defined by

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

We denote the *positive closure* of L by L^+ and define by LL^*.

Example 2.1.1. Let $\Sigma = \{a, b, c\}$ be an alphabet. Then $L = \{a, bc, cba\}$ is a language over Σ and

$$L^* = \{\epsilon, a, bc, cba, aa, abc, acba, bca, bcbc, bccba, cbaa, cbabc, cbacba, ...\}$$

is the Kleene closure of L.

2.2 Regular expressions

A *regular expression* over the base alphabet Σ is a word α over the alphabet $\Sigma \cup \{\epsilon, \emptyset, (,), +, \cdot, *\}$. We denote by $L(\alpha)$ the language described by α. A regular expression α is defined recursively as follows:

- $\alpha = \emptyset$ is a regular expression for the language $L(\alpha) = \emptyset$.
- $\alpha = \epsilon$ is a regular expression for the language $L(\alpha) = \{\epsilon\}$.
- $\alpha = a$ for $a \in \Sigma$ is a regular expression for the language $L(\alpha) = \{a\}$.

Let β and γ be regular expressions. Then,

- $\alpha = \beta + \gamma$ is a regular expression for the language $L(\alpha) = L(\beta) \cup L(\gamma)$.
- $\alpha = \beta \cdot \gamma$ is a regular expression for the language $L(\alpha) = L(\beta)L(\gamma)$.
- $\alpha = \beta^*$ is a regular expression for the language $L(\alpha) = (L(\beta))^*$.

The Kleene star, $*$, has the highest precedence, followed by $\cdot$, the concatenation operation, followed by $+$, the union operation. Parentheses are used to group terms and explicitly define the intended order of operations. We say that two regular expressions α and β are equivalent if $L(\alpha) = L(\beta)$. For convenience, we often omit $\cdot$ when writing $\alpha \cdot \beta$ and write $\alpha\beta$ instead.

Example 2.2.1. Let α denote the regular expression $(0 + 1)^*0$ over $\Sigma = \{0, 1\}$. This regular expression describes all 0,1-strings that end in 0. If we take 0,1-strings to represent numbers written in base 2, we can say that this regular expression describes all nonempty 0,1-strings which encode even numbers in base 2.

We say that a language L is *regular* if $L = L(\alpha)$ for some regular expression α. The class of formal languages which can be described by a regular expression is called the class of *regular languages*. A class of languages is *closed* under an operation if when applying the operation to languages which belong to the class, the resultant language also belongs to the class. Here, we give some closure properties for operations that we will later use. This is helpful for guaranteeing that new languages that we define based on languages that we know to be regular are also regular.

Theorem 2.2.1. *The class of regular languages is closed under union, intersection, concatenation, and the Kleene star.*

2.3 Finite automata

A finite automaton is a theoretical machine which reads input in the form of words one symbol at a time. It contains a set of internal states and depending on the machine's current state and the input symbol being read, it can change to another state and read the next character. The machine either accepts or rejects the input. The machine accepts the input when upon reaching the end of the input word, the machine's internal state is in an accepting state.

Formally, a *deterministic finite automata* (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, q_0 is the initial state, and F is the set of accepting states. We can extend the transition function δ for words instead of symbols to the function $\hat{\delta} : Q \times \Sigma^* \to Q$ defined by

$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$$

where $x \in \Sigma^*$ and $a \in \Sigma$. For convenience, we use δ for $\hat{\delta}$. The language accepted by the DFA M, denoted $L(M)$, is defined

$$L(M) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}.$$

We say that two DFAs M_1 and M_2 are equivalent if $L(M_1) = L(M_2)$.

In a DFA, the next state is uniquely determined by the current state and input symbol being read. A natural generalization is to allow more than one possible transition for a given state and input symbol. This generalization leads to the notion of a nondeterministic finite automata.

Formally, a *nondeterministic finite automata* (NFA) is a 5-tuple $N = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, and F are defined in the same way as DFAs, and $\delta : Q \times \Sigma \to 2^Q$ is the transition function. Here, 2^Q denotes the power set, or the set of all subsets, of Q. Just as in DFA, we can extend the transition function to $\hat{\delta} : Q \times \Sigma^* \to 2^Q$, defined by

$$\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a)$$

where $x \in \Sigma^*$ and $a \in \Sigma$. Then the language accepted by the NFA N is defined by

$$L(N) = \{w \in \Sigma^* : \delta(q_0, w) \cap F \neq \emptyset\}.$$

We say that two NFAs N_1 and N_2 are equivalent if $L(N_1) = L(N_2)$.

We may also extend the definition of the transition function to include transitions on the empty string ϵ, which we call *ϵ-transitions*. NFAs which allow ϵ-transitions are known as *ϵNFAs*. ϵNFAs can recognize exactly the same class of languages as NFAs, and an ϵNFA can be easily transformed into an equivalent NFA [33]. Without loss of generality, we do not consider NFAs with ϵ-transitions.

A state $q \in Q$ of an NFA or DFA is *unreachable* if there is no path in the transition graph starting at q_0 and ending in q. A state $q \in Q$ is *dead* if there is no path from q to a final state. An NFA or DFA is *trim* if it contains no unreachable or dead states. We assume without loss of generality that NFAs are trim.

There are a few different ways to represent a finite automaton. We can represent them visually by drawing a directed graph called a transition diagram. We represent states as circles and accepting states are denoted by double-outlined circles. Transitions are drawn as arrows from a state to another state and the initial state is denoted by a headless arrow pointing to the initial state. For more complicated finite automata, it may be more helpful to describe it by its transition function in a table.

Example 2.3.1. We define a DFA $A = (Q, \Sigma, \delta, q_0, F)$ which accepts all 0,1-strings which encode even numbers in binary with $Q = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$, $F = \{q_1\}$, and δ is defined for every $q \in Q$ by

$$\delta(q, a) = \begin{cases} q_1 & \text{if } a = 0 \\ q_0 & \text{if } a = 1 \end{cases}$$

The transition diagram for A is given in Figure 2.1.

One might assume that the addition of nondeterminism in NFAs gives more computational power and allows the acceptance of more languages. This is not the case. The class of languages accepted by DFAs and the class of languages accepted by NFAs is exactly the same. Clearly, every DFA is an NFA, so all that needs to be shown is that every NFA can be transformed into an equivalent DFA.

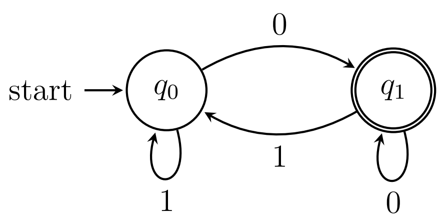

Figure 2.1: The DFA A, which recognizes 0,1-strings encoding even numbers

Theorem 2.3.1. *If M is a NFA, there exists an DFA M' such that $L(M) = L(M')$.*

We can construct the DFA M' from the NFA M by using a process called the subset construction. The idea behind the construction is to let the state set of M' be exactly the subsets of the states of M. This way, the state transitions are uniquely determined by the state and input symbol. The set of accepting states of M' is simply all of those subsets containing an accepting state of M.

Example 2.3.2. Figure 2.2 shows an NFA $M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_3\})$ which accepts the language of words over $\Sigma = \{a, b\}$ with a in the third position from the right, or formally, $L_3 = \{w \in \{a, b\}^* : w = xay, x \in \{a.b\}^*, y \in \{a, b\}^2\}$, is shown.

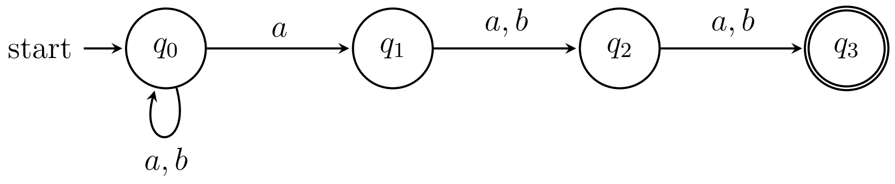

Figure 2.2: The NFA M accepting L_3

We perform the subset construction to build an equivalent DFA $M' = (Q', \Sigma, \delta', \{q_0\}, F')$. The transition function δ' is given in Table 2.1. Note that only states that are reachable are listed.

From the transitions given for δ', we can see that $Q' \subseteq 2^Q$ consists of eight states: $\{q_0\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_0, q_3\}, \{q_0, q_1, q_2\}, \{q_0, q_1, q_3\}, \{q_0, q_2, q_3\}$, and $\{q_0, q_1, q_2, q_3\}$. The set of accepting states F' consists of states that contain any of the accepting states in M. Since $F = \{q_3\}$, we have $F' = \{\{q_0, q_3\}, \{q_0, q_1, q_3\}, \{q_0, q_2, q_3\}, \{q_0, q_1, q_2, q_3\}\}$. The transition diagram for M' is given in Figure 2.3.

q	$\delta(q,a)$	q	$\delta(q,b)$
$\{q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$	$\{q_0\}$
$\{q_0,q_1\}$	$\{q_0,q_1,q_2\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$
$\{q_0,q_2\}$	$\{q_0,q_1,q_3\}$	$\{q_0,q_2\}$	$\{q_0,q_3\}$
$\{q_0,q_3\}$	$\{q_0,q_1\}$	$\{q_0,q_3\}$	$\{q_0\}$
$\{q_0,q_1,q_2\}$	$\{q_0,q_1,q_2,q_3\}$	$\{q_0,q_1,q_2\}$	$\{q_0,q_2,q_3\}$
$\{q_0,q_1,q_3\}$	$\{q_0,q_1,q_2\}$	$\{q_0,q_1,q_3\}$	$\{q_0,q_2\}$
$\{q_0,q_2,q_3\}$	$\{q_0,q_1,q_3\}$	$\{q_0,q_2,q_3\}$	$\{q_0,q_3\}$
$\{q_0,q_1,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$	$\{q_0,q_2,q_3\}$

Table 2.1: Table for transition function δ'

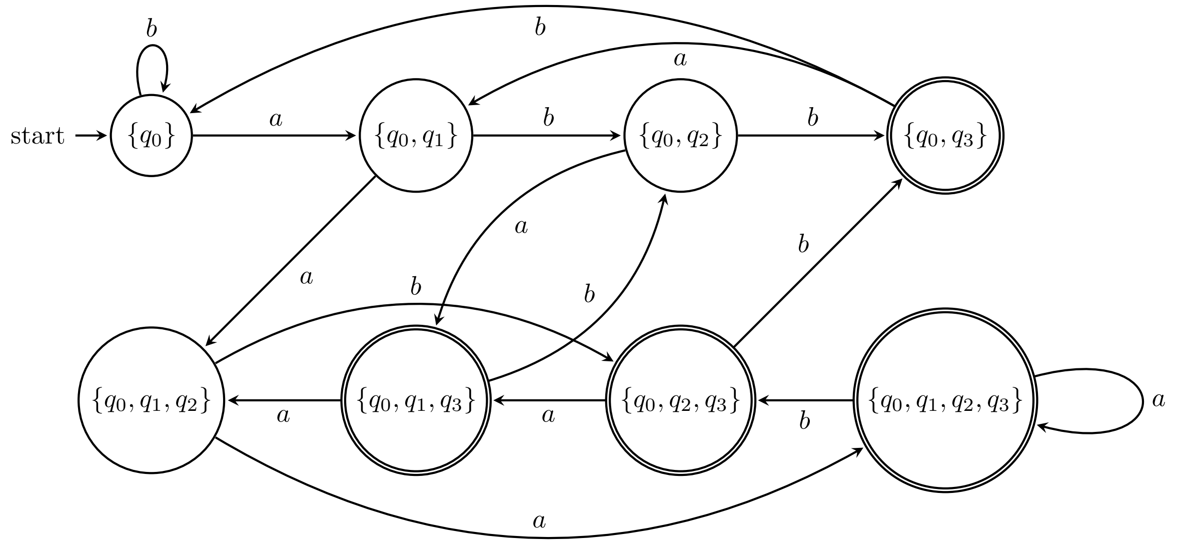

Figure 2.3: The DFA M' accepting L_3

2.4 Converting regular expressions to finite automata

The following theorem by Kleene [44] establishes the correspondence between regular expressions and finite automata.

Theorem 2.4.1. *A language is regular if and only if it is accepted by a finite automaton.*

Regular expressions and finite automata are two different tools that we use in different ways to describe regular languages. We use regular expressions to specify patterns to be matched, while finite automata are used to recognize whether a given input string belongs to a language. As we already mentioned, regular expressions are more convenient for human users to describe regular languages and are easier to write and manipulate than finite automata. On the other hand, finite automata are easier to implement in computer programs using structures like switch-case statements or adjacency matrices. The equivalence between languages described by regular expressions and languages accepted by finite automata means that we don't need to choose between the two.

The first step in implementing regular expressions is to transform the regular expression into an NFA. The simplest method to do this is to construct an automaton known as the position or Glushkov automaton [21, 51]. For a regular expression α, the position automaton has exactly $|\alpha| + 1$ states. There are other ways to construct an NFA for a regular expression with fewer states and transitions, such as partial derivative automata or follow automata, but these automata can be derived from the position automaton [9, 36].

For a regular expression α, let $\text{pos}(\alpha) = \{1, 2, ..., |\alpha|\}$. We mark each letter of α with its position. Let $\overline{\alpha}$ denote the marked regular expression over $\overline{\Sigma} = \{a_i : a \in \Sigma, i \in \text{pos}(\alpha)\}$. For example, if $\alpha = aa(ba)^*(b + ba)$, then $\overline{\alpha} = a_1 a_2 (b_3 a_4)^*(b_5 + b_6 a_7)$. We use the same notation to remove markings by letting $\overline{\overline{\alpha}} = \alpha$. For $a_i \in \overline{\Sigma}$ with $i \in \text{pos}(\alpha)$ and $u, v, w \in \overline{\Sigma}^*$, we define the following sets:

$$\begin{aligned} \text{first}(\alpha) &= \{i : a_i w \in L(\overline{\alpha})\} \\ \text{last}(\alpha) &= \{i : w a_i \in L(\overline{\alpha})\} \\ \text{follow}(\alpha, i) &= \{j : u a_i a_j v \in L(\overline{\alpha})\} \end{aligned}$$

The *position automaton* for α is $N_{\text{pos}}(\alpha) = (\text{pos}(\alpha) \cup \{0\}, \Sigma, \delta_{\text{pos}}, 0, \text{last}(\alpha) \cup \{0\})$ with δ_{pos} defined by

$$\delta_{\text{pos}}(i, a) = \begin{cases} \{j \in \text{follow}(\alpha, i) : a = \overline{a_j}\} & \text{if } i \neq 0 \\ \{j \in \text{first}(\alpha) : a = \overline{a_j}\} & \text{if } i = 0 \end{cases}.$$

Example 2.4.1. Let $\alpha = aa(ba)^*(b+ba)$. Then $\overline{\alpha} = a_1a_2(b_3a_4)^*(b_5+b_6a_7)$ and $\text{first}(\alpha) = \{1\}$ and $\text{last}(\alpha) = \{5, 7\}$. We define $\text{follow}(\alpha, i)$ for $1 \leq i \leq 7$ in Table 2.2. The position automaton $A_{\text{pos}}(\alpha)$ is given in Figure 2.4.

i	$\text{follow}(\alpha, i)$
1	$\{2\}$
2	$\{3, 5, 6\}$
3	$\{4\}$
4	$\{3, 5, 6\}$
5	$\emptyset$
6	$\{7\}$
7	$\emptyset$

Table 2.2: The table for $\text{follow}(\alpha, i)$

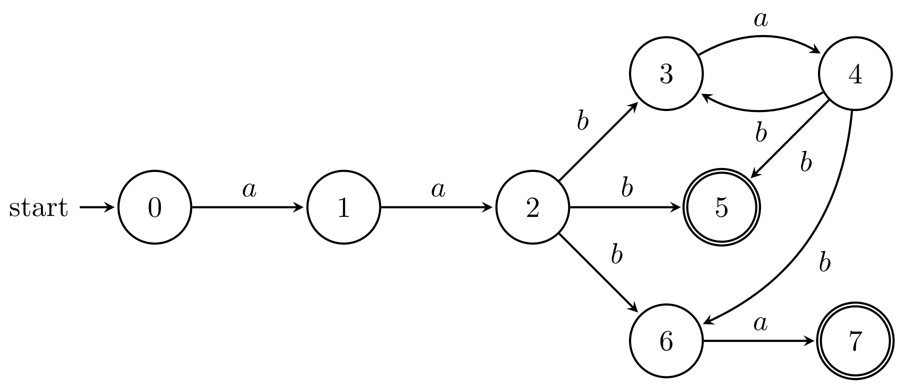

Figure 2.4: The position automaton for $\alpha = aa(ba)^*(b + ba)$

After we construct an NFA, we can transform it into a DFA using the subset construction. As we will see shortly, NFAs are much more succinct than DFAs. However, there are some tasks for which DFAs are more well-suited. For instance, membership testing is much more straightforward for a DFA since nondeterminism does not need to be modeled in implementation. Testing for equality is also much simpler since DFAs have a canonical form, the minimal DFA, which we will also introduce shortly.

Note that the DFA that results from the subset construction is not necessarily the minimal equivalent DFA. We would like these DFAs to be as small as possible. This is the problem of DFA minimization. The minimal DFA M for a language L is simply the DFA with the minimal number of states which accepts L. We have the following property as a result of theorems by Myhill [54] and Nerode [58]:

Theorem 2.4.2. *Every regular language is described by minimal DFA which is unique, up to isomorphism.*

For an NFA with n states, the number of states in an equivalent DFA can be up to 2^n. In other words, a DFA can be exponentially larger than the regular expression that we started out with. This is a problem for implementation because of the memory requirements for storing such large structures. While DFAs can be minimized in $O(n \log n)$ time via Hopcroft's algorithm [30], this still requires the possibly costly transformation from NFA to DFA.

One method to mitigate the large size of DFAs is to try to reduce the size of the NFA before the conversion to a DFA. What would be helpful is an efficient method to reduce the size of NFAs. Unfortunately, this problem is not as simple as it is for DFAs and will be discussed in further detail in Chapter 4.

Chapter 3

The Vertex Cover Problem

3.1 Complexity theory

We measure the time complexity of an algorithm by the number of operations that are performed as a function $f(n)$ of the length n of the input. We say that an algorithm runs in $O(g(n))$ time if there exists a constant $c > 0$ such that for every sufficiently large n, $f(n) \leq c \cdot g(n)$. We say that $f(n)$ is $o(g(n))$ if for every constant $c > 0$, we have $f(n) < c \cdot g(n)$ for all sufficiently large n. Similarly, we say that $f(n)$ is $\Omega(g(n))$ if there exists a constant $c > 0$ such that for every sufficiently large n, $f(n) \geq c \cdot g(n)$.

A *decision problem* is a problem which has an answer of either YES or NO. An *optimization problem* is a problem which attempts to minimize or maximize some value. When discussing the complexity of a problem, we are concerned with decision problems. However, an optimization problem can be easily reformulated as a decision problem which is no harder than the original optimization problem. In other words, the optimization problem can be solved in time that is at most a polynomial factor larger than that for the decision problem.

For instance, given a DFA, the DFA minimization problem asks for an equivalent DFA with the minimal number of states. The answer to the problem is a DFA. We can reformulate the problem as a decision problem by asking whether there exists an equivalent DFA of size at most k, for some integer k. The answer to this problem is either YES or NO. To find a minimal DFA equivalent to a given n-state DFA, we ask whether there is an equivalent DFA of size $1, 2, ..., n-1$, which means asking the decision problem at most $n-1$ times.

A problem is solvable in polynomial time if there exists an algorithm that solves it in $O(n^k)$ time for constant k. We define P to be the class of decision problems which are solvable in polynomial time. We consider an algorithm to be efficient if it has polynomial running time and we consider problems to be computationally feasible if they are in solvable in polynomial time.

We define NP to be the class of decision problems that have solutions which can be *verified* in polynomial time. That is, we can check whether a given solution for an instance of the problem is correct or not in polynomial time. Formally, a verifier V for a problem A is an algorithm that takes as input an instance I of A and a polynomial-size certificate C. V returns YES if C verifies that the answer to I is YES and NO otherwise. A can be verified in polynomial time if the time complexity of its verifier is polynomial in the size of I.

A problem is *NP-hard* if every problem in NP can be reduced in polynomial time to the problem. A problem is *NP-complete* if the problem is NP-hard and is also in NP. A problem A reduces to a problem B if there is a polynomial-time function f that transforms an instance I of A to an instance $f(I)$ of B such that I is an instance of A for which the answer is YES if and only if $f(I)$ is an instance of B for which the answer is YES.

By definition, any problem which is in P is clearly in NP, since a problem which is computable in polynomial time will have a solution which is verifiable in polynomial time. Thus, we have the inclusion P $\subseteq$ NP. However, it is currently not known whether P $=$ NP. Since every NP-complete problem reduces to every other NP-complete problem, if there is a polynomial time algorithm for any one NP-complete problem, then there is a polynomial-time algorithm for every problem in NP. And if there is no polynomial-time algorithm for any problem in NP, then no problem in NP has a polynomial-time algorithm. Showing either of these would resolve the P $=$ NP question.

In this thesis, we assume that P $\neq$ NP and thus, we consider problems which are NP-hard to be intractable.

3.2 The minimal vertex cover problem

A *graph* $G = (V, E)$ is a pair consisting of a finite nonempty set V of vertices and a collection E of pairs of V, called edges. A *hypergraph* $H = (V, E)$ is a generalization of a graph and is a pair consisting of a finite nonempty set V of vertices and a collection E of subsets of V, called hyperedges. H is k-*uniform* if $|e| = k$ for every hyperedge $e \in E(H)$. A hypergraph is *weighted* if it is equipped with a weight function $w : V \to \mathbb{Z}$. For convenience, we will refer to hyperedges as edges. We say a vertex v is *incident* to an edge e if $v \in e$. We define the *degree* of a vertex v to be the number of edges it is incident to. The degree of a hypergraph is the maximum degree of the vertices of the

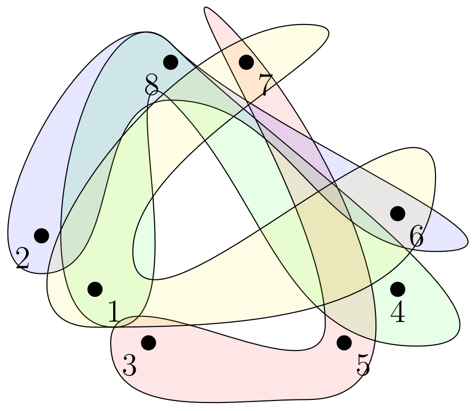

Figure 3.1: The hypergraph $H = (V, E)$

hypergraph.

A *k-colouring* of a hypergraph $H = (V, E)$ is a partition $(V_1, ..., V_k)$ of V into k colour classes such that each edge contains vertices from at least two colour classes. A hypergraph is *k-colourable* if it admits a k-colouring. A *strong k-colouring* is a partition $(V_1, ..., V_k)$ of V such that each edge contains at most one vertex of each colour. A hypergraph is k-strongly-colourable if it admits a strong k-colouring. A hypergraph is *k-partite* if the vertex set can be partitioned into k sets and every hyperedge contains at most one vertex from each partition.

Example 3.2.1. A hypergraph $H = (V, E)$ is drawn in Figure 3.1. H has vertex set $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and hyperedge set $E = \{\{1, 6, 7\}, \{1, 4, 8\}, \{2, 6, 8\}, \{3, 5, 7\}\}$. The vertices are represented as dots and each hyperedge is represented as a curve. Vertices contained in a curve belong to the hyperedge represented by the curve. Since each hyperedge of H has size exactly 3, H is 3-uniform. We can partition the vertex set of H into three colour classes: $\{1, 2, 3\}$, $\{4, 5, 6\}$, and $\{7, 8\}$. Thus, H is also 3-strongly-colourable.

The vertex cover problem is a fundamental combinatorial optimization problem. A *vertex cover* on a hypergraph $H = (V, E)$ is a subset of vertices $C \subseteq V$ such that every hyperedge in E contains at least one vertex from C. That is, for every $e \in E(H)$, we have $e \cap C \neq \emptyset$. We say a vertex $v \in C$ *covers* an edge e if $v \in e$ and we say that an edge e is *covered* with respect to a cover C if it is covered by a vertex in C. The *minimum vertex cover problem* is given a hypergraph $H = (V, E)$, find a vertex cover with minimal cardinality. For the weighted version of the problem, a weight function $w : V \to \mathbb{Z}$ on H is also given and we wish to find a vertex cover with minimal total weight.

Example 3.2.2. A vertex cover on the hypergraph H in Figure 3.1 would be the set $\{4,5,6\}$; this vertex cover is not the smallest one. The set $\{7,8\}$ is also a vertex cover for H and is also a minimal vertex cover. We know that a minimal vertex cover has size at least 2, since there is no single vertex which covers all four edges of H.

The minimum vertex cover problem was one of the first problems shown to be NP-complete by Karp in [40]. For hypergraphs, this problem is equivalent to the set cover problem and we can always reformulate a minimum vertex cover problem on hypergraphs as a minimum set cover problem. The minimum set cover problem asks, for an instance $I = (U, S)$, where U is the set of ground elements and $S \subseteq 2^U$ is a collection of subsets of U, to find a subcollection $C \subseteq S$ such that $\bigcup_{X \in C} X = U$ of minimal cardinality.

To formulate the instance I as an instance $J = (V, E)$ minimum hypergraph vertex cover problem, we map elements of U to edges in E and subsets from S to vertices in V. Let $f : U \cup S \to V \cup E$ by such a map with $V = \{f(x) : x \in S\}$ and $E = \{f(x) : x \in U\}$. Then for $x \in U$ and $s \in S$, we have $x \in X$ if and only if $f(s) \in f(x)$.

We are interested in the minimum vertex cover problem for k-uniform k-partite hypergraphs. This problem has applications in many areas, among others, NFA reduction, distributed data mining [16] and database schema mapping discovery [22]. When $k = 2$, or when the graph is *bipartite*, the problem is solvable in polynomial time, since by König's theorem, the problem is equivalent to finding a maximum matching. However, for $k \geq 3$, the problem is NP-complete.

Like many combinatorial optimization problems, we can formulate the vertex cover problem as an integer program. The following is the integer program for the vertex cover problem.

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} w(v) g(v) \\ \text{subject to} \quad & \sum_{v \in e} g(v) \geq 1, \forall e \in E \\ & g(v) \in \{0,1\}, \forall v \in V \end{aligned} \tag{3.1}$$

For each vertex $v \in V$, we assign a variable $g(v)$ which takes on a value of 1 if it is in the vertex cover and 0 otherwise. For each edge, at least one of its vertices must be in the vertex cover. This translates into the condition that the sum of the values of the variables for each vertex in an edge must be at least 1. We wish to find a vertex cover with the minimal total weight.

Note that solving an integer program is NP-hard, so reformulating the problem as an integer program does not change the hardness of the problem. However, we can relax the condition that variables take on integer values and allow the variables to be real-valued. This is the linear programming relaxation of the integer program. While it may initially seem strange to allow including, say, $\frac{1}{3}$ of a vertex in a vertex cover, the linear programming relaxation has the advantage of being able to be solved in polynomial time. The following linear programming relaxation defines the fractional vertex cover problem.

$$\begin{aligned} \text{minimize } & \sum_{v \in V} w(v) g(v) \\ \text{subject to } & \sum_{v \in e} g(v) \geq 1, \forall e \in E \\ & g(v) \geq 0, \forall v \in V \end{aligned} \tag{3.2}$$

A feasible solution g for Problem (3.2) is a *fractional vertex cover* of H. We denote the value of g by $|g| = \sum_{v \in V} g(v)$. We define the covering number $\tau(H)$ of H to be the size of the minimal vertex cover of H. Similarly, we let $\tau^*(H)$, the fractional covering number of H, denote the minimum value of g across all fractional covers of H.

We can use the fractional vertex cover to approximate the minimum vertex cover by rounding the values $g(v)$ to integer values. The *integrality gap* of the problem is the ratio of the value of the optimal solution of the linear programming relaxation to the value of the optimal solution for the integer program. For the minimum vertex cover problem, this is the ratio $\frac{\tau(H)}{\tau^*(H)}$.

3.3 Approximation Algorithms

While finding the optimal solution for many problems is intractable, approximate solutions can often be computed efficiently. We can measure the quality of an approximation algorithm by its approximation ratio. Given an optimization problem γ, let the *cost* or *value* of an optimal solution for γ be denoted by C^* and let C denote the value of a solution produced by an approximation algorithm $\mathcal{A}$. Then the *approximation ratio* of $\mathcal{A}$ on input of size n is $\rho(n)$, defined in [10] by

$$\max\left\{\frac{C}{C^*}, \frac{C^*}{C}\right\} \leq \rho(n)$$

and we say $\mathcal{A}$ a *ρ-approximation algorithm* and we call a solution computed by $\mathcal{A}$ a *ρ-approximation.*

Example 3.3.1. The simplest approximation algorithm for the minimal vertex cover problem on graphs is a greedy algorithm that selects vertices based on a maximal matching, giving a 2-approximation. A *matching* M of a graph $G = (V, E)$ is a subset of edges such that no two edges in M are adjacent to a common vertex. A *maximal matching* is a matching M of G such that M is no longer a matching if any edge not in M is added to it. A maximal matching M can be found simply by examining every edge and adding it to M if it is not adjacent to any edge already in M. This can be done in polynomial time.

For a graph $G = (V, E)$, given a maximal matching M, let C be the subset of vertices which are adjacent to the edges in M. C is a vertex cover of G, since C covers every edge in E. If there was an edge $e \in E$ that was left uncovered by C, then e would be an edge which could be added to M, contradicting the maximality of M.

Since each edge in M must be covered by one of the vertices it is incident to and the edges of M do not share any vertices, any cover must be at least as large as $|M|$. Therefore, for any optimal vertex cover C^*,

$$|M| \leq |C^*| \leq |C| = 2|M| \leq 2|C^*|.$$

Thus, this algorithm has an approximation ratio of $\frac{|C|}{|C^*|} \leq 2$.

The best approximation algorithm for the minimal vertex cover problem on hypergraphs achieves an approximation ratio of $O(\log d)$, where d is the maximum degree of the hypergraph [49]. When we restrict the size of the edges to at most k vertices, we can apply the same greedy algorithm as in Example 3.3.1 to achieve an approximation ratio of k. Many algorithms achieve approximation ratios slightly better than this simple k-approximation.

Lovász gives an upper bound of $\frac{k}{2}$ for the integrality gap for the minimum vertex cover problem on k-partite k-uniform hypergraphs [48]. In [1], Aharoni et. al give an example showing that this bound is tight. If we generalize further and consider k-partite r-uniform hypergraphs (or, equivalently, k-strongly-colourable r-uniform hypergraphs) with $r \neq k$, Aharoni et. al show an upper bound of

$$\frac{\tau(H)}{\tau^*(H)} \leq \frac{k-r+1}{k} r$$

for $k \geq (r-1)r$. For $r \leq k \leq (r-1)r$, they show an upper bound of

$$\frac{\tau(H)}{\tau^*(H)} \leq \frac{kr}{k+r} + \min\left\{\frac{k-r}{2k}\{u\}, \frac{r}{k}(1-\{u\})\right\},$$

where $u = \frac{k^2}{k+r}$ and $\{u\} = u - \lfloor u \rfloor$. If we consider k-colourable r-uniform hypergraphs, Krivelevich [46] shows an upper bound of $\max\left\{\frac{k+1}{k}r, r-1\right\}$ and gives two algorithms that compute approximate solutions that achieve the above approximation ratio.

A natural question to ask is whether we can approximate the optimal solution of a problem to within an arbitrary factor in polynomial time. Solving a problem exactly may be difficult, but if we can compute a solution that is within, say, 1% of an optimal solution, in many cases, we would be quite pleased with the result. For some problems, like the knapsack problem [31], it is possible to design an ρ-approximation algorithm for every $\rho > 1$.

However, it has been shown that problems like minimum set cover and minimum vertex cover cannot be ρ-approximated for every $\rho > 1$. A problem is said to be *inapproximable to within a factor of* ρ if there is no ρ-approximation algorithm for it that runs in polynomial time, unless P = NP. If such an algorithm existed, then it would imply that P = NP. This means that for the set cover and vertex cover problems, there is a limit to how good approximate solutions can be computed in polynomial time, assuming P $\neq$ NP.

Arora showed an integrality gap of $2 - \epsilon$, for some small $\epsilon > 0$, for the vertex cover problem on graphs [4], which matches the best known approximation ratio for algorithms based on a linear programming relaxation of the problem; Håstad [26] showed that the problem is inapproximable to within a factor of $\frac{7}{6} - \epsilon$, for certain small $\epsilon > 0$, and Dinur and Safra improved this bound to 1.36 [15].

For general hypergraphs with unbounded edge size, the problem is inapproximable to within a factor of $(1 - o(1)) \ln n$, where n is the number of hyperedges [17]. A number of inapproximability results were shown by using the PCP theorem, which relates proof checking with approximability [42]. The earliest inapproximability result for k-uniform hypergraphs was presented by Trevisan in [66] showing inapproximability to within $\Omega(k^{\frac{1}{19}})$. Holmerin improved this to $\Omega(k^{1-\epsilon})$ in [28]. Dinur et. al show inapproximability to within $k - 3 - \epsilon$ in [13] shortly before improving the result to $k - 1 - \epsilon$ in [14].

In [1], Aharoni et. al constructed an instance which matched the $\frac{k}{2}$ upper bound on the integrality gap given by Lovász. A recent result by Guruswami and Saket [25] shows that vertex cover on k-partite k-uniform hypergraphs is inapproximable to within a factor of $\frac{k}{4} - \epsilon$ for $k \geq 16$. This result is based on a reduction from k-uniform hypergraphs and uses the inapproximability result from [14]. The result is improved further by Sachdeva and Saket [64] to $\frac{k}{2} - 1 + \frac{1}{2k} - \epsilon$ for $k \geq 4$ who combine Dinur's multilayered PCP with the integrality gap instance from [1].

Improved inapproximability results can be achieved using the Unique Games Conjecture of Khot [41]. The UGC provides a convenient reduction for certain classes of problems for which using the PCP theorem may be challenging. Assuming that the UGC holds, Khot and Regev [43] show inapproximability of vertex cover on graphs within a factor of $2-\epsilon$. They also show inapproximability within $k-\epsilon$ for k-uniform hypergraphs under this assumption. In [25], Guruswami and Saket also include a UGC-based inapproximability result of $\frac{k}{2}-\epsilon$ for k-uniform k-partite hypergraphs. This result matches the bound achieved by Lovász.

Chapter 4

The NFA reduction problem

4.1 Preliminaries

While DFA minimization is computable in polynomial time, NFA minimization is known to be PSPACE-complete [37] and, therefore, is much more computationally difficult, assuming P $\neq$ PSPACE. PSPACE is the class of problems which can be solved with a polynomial amount of space and a problem A is PSPACE-complete if every other problem in PSPACE can be reduced into A in polynomial time. This means that if a PSPACE-complete problem could be solved in polynomial time, then all problems in PSPACE could also be solved in polynomial time. While it is clear that P $\subseteq$ NP $\subseteq$ PSPACE, it is not known whether P = PSPACE. Not only is the problem extremely difficult, but even approximating it is hard. Given an n-state NFA, computing an equivalent minimal NFA is inapproximable to within a factor of $o(n)$ unless P = PSPACE [23]. This means that any polynomial time algorithm that reduces the size of an NFA cannot give any guarantees on the size of the reduction that is sub-linear in the size of the given NFA unless P = PSPACE.

Because of the computational hardness of the problem, there have been a variety of approaches to try to make the problem feasible. There are exact minimization algorithms from Kameda and Weiner [38] and Melnikov [52] which are infeasible in practice. Ilie and Yu introduce a method which involves merging states based on equivalence relations [35]. Champarnaud and Coulon extend that idea by using preorder relations [8]. Geldenhuys, van der Merwe, and van Zijl propose a technique which transforms instances of NFA reduction into instances of SAT and using SAT solvers to compute reduced NFAs [20].

While NFA minimization is hard in general, a natural question is whether there are any families of languages or restrictions on automata for which the problem is feasible. Gruber and Holzer study the problem restricted to unary and finite languages [24]. They find that given an n-state DFA which accepts a finite language, finding an equivalent minimal NFA is DP-hard. DP is the class of problems which are the intersection of a problem in NP and a problem in co-NP, where co-NP is the class of problems whose complements are in NP [61]. For unary languages, if an n-state DFA is given, the problem has an $O(\sqrt{n})$-approximation, while the problem remains inapproximable to within a factor of $o(n)$ given an n-state NFA, unless P = NP.

Gramlich and Schnitger show in [23] that, given an n-state DFA, finding an equivalent minimal NFA is inapproximable to within a factor of $\frac{\sqrt{n}}{\text{poly}(\log n)}$. They also show that for an n-state NFA accepting a unary language, finding an equivalent minimal NFA is inapproximable to within a factor of $n^{1-\delta}$ for all $\delta > 0$. In [5], Björklund and Martens attempt to resolve a question posed by Malcher in [50], which asks whether there are any useful extensions of DFAs with limited amounts of nondeterminism for which minimization is tractable. They find that for δNFAs, a class of NFAs which have at most two computations for every input string, minimization remains NP-hard.

We base our work on research done by Ilie, Solis-Oba, and Yu in [34]. In the paper, the authors show how to compute optimal reductions by merging states based on the equivalences introduced in [35] and the preorders from [8]. To help define various language relations over states of NFAs, we give the following definitions. The language recognized by an NFA $N = (Q, \Sigma, \delta, q_0, F)$ is $\mathcal{L}(N) = \{w \in \Sigma^* : \delta(q_0, w) \cap F \neq \emptyset\}$. For states $p, q \in Q$, we define

$$\begin{aligned}\mathcal{L}_L(N,p) &= \{w \in \Sigma^* : p \in \delta(q_0, w)\}\\ \mathcal{L}_R(N,p) &= \{w \in \Sigma^* : \delta(p, w) \cap F \neq \emptyset\}\\ \mathcal{L}(N,p,q) &= \{w \in \Sigma^* : q \in \delta(p, w)\}\end{aligned}$$

For simplicity, we write $\mathcal{L}_L(p)$, $\mathcal{L}_R(p)$, and $\mathcal{L}(p,q)$, respectively when N is understood.

4.1.1 Reducing NFAs based on equivalences

In [35], Ilie and Yu define equivalence relations on the states of an NFA. An *equivalence relation* on a set S is a relation $\equiv \subseteq S \times S$ which satisfies the following for all $a, b, c \in S$:

- *Reflexivity*: $a \equiv a$
- *Symmetry*: if $a \equiv b$, then $b \equiv a$
- *Transitivity*: if $a \equiv b$ and $b \equiv c$, then $a \equiv c$

An equivalence relation partitions the set S into equivalence classes. Two elements $a, b \in S$ belong to the same equivalence class X if and only if $a \equiv b$. For a subset $T \subseteq S$, $T/_{\equiv}$ denotes the *quotient set* $\{[a]_{\equiv} : a \in T\}$, where $[a]_{\equiv}$ denotes the equivalence class containing a. Let $\equiv$ and $\cong$ be two equivalence relations over the set S. We say that $\cong$ is *coarser* than $\equiv$ if $a \equiv b$ implies $a \cong b$.

An equivalence relation $\equiv$ is *right-invariant* with respect to an NFA $N = (Q, \Sigma, \delta, q_0, F)$ if it satisfies the following:

1. $\equiv \subseteq (Q \setminus F)^2 \cup F^2$
2. for every $p, q \in Q$, $a \in \Sigma$, if $p \equiv q$, then $\delta(p, a)/_{\equiv} = \delta(q, a)/_{\equiv}$

For an NFA $N = (Q, \Sigma, \delta, q_0, F)$, the equivalence relation $\equiv_R$ is defined to be the coarsest right-invariant equivalence relation on a state set Q that satisfies the following:

$$(\mathcal{P}_1) \quad \equiv_R \cap (F \times (Q - F)) = \emptyset$$
$$(\mathcal{P}_2) \quad \forall p, q \in Q, \forall a \in \Sigma, (p \equiv_R q \Longrightarrow \forall q' \in \delta(q, a), \exists p' \in \delta(p, a), q' \equiv_R p')$$

The left equivalence $\equiv_L$ is defined to satisfy the same axioms on the reversed automaton, which is constructed by reversing the transitions of N and exchanging the initial and final states. Ilie, Navarro, and Yu show in [32] that equivalences over the state set can be computed using a partition refinement algorithm by Tarjan and Paige [60] with $O(m \log n)$ time and $O(m + n)$ space, where n is the number of states and m is the number of transitions.

Each equivalence class of $\equiv_R$ and $\equiv_L$ represents equivalent states that can be merged into a single state without modifying $L(N)$. We merge a state p with a state q by replacing all incoming and outgoing transitions of p with q and deleting p. Since $\equiv_R$ is right-invariant, it has the property that for some word $w = a_1 a_2 \cdots a_n \in \mathcal{L}_R(p)$, there is a path $p a_1 p_1 a_2 p_2 \cdots a_n p_n$ in N with $p_n \in F$ such that there exists a path $q a_1 q_1 \cdots a_n q_n$ in N with $p_i \equiv_R q_i$ for all $1 \leq i \leq n$ and $q_n \in F$. This implies that $w \in \mathcal{L}_R(q)$ and, thus, merging p and q does not change $L(N)$. The same reasoning applies for $\equiv_L$.

Denote by Π_R and Π_L the sets of equivalence classes of $\equiv_R$ and $\equiv_L$, respectively. A reduction is a subset $Y \subseteq \Pi_R \cup \Pi_L$ of the two partitions such that Y contains every state in Q. In [34], Ilie, Solis-Oba, and Yu show that the problem of finding a minimal reduction is equivalent to the minimum set cover problem where Q is the set of ground elements and $\Pi_R \cup \Pi_L$ is the family of subsets. Since Π_R and Π_L are partitions of Q, each state in Q is guaranteed to belong to exactly two subsets of the family of subsets. This particular version of the minimum set cover problem is equivalent to the minimal vertex cover problem on bipartite graphs, which can be solved in polynomial time [34]. We call the resulting optimally reduced NFA an EQ-*reduced NFA*. We note that this is not a minimal NFA.

Example 4.1.1. Let $N = (Q, \Sigma, \delta, q_0, F)$ be as in Figure 4.1. The equivalence classes for $\equiv_R$ are $\{q_0\}$, $\{q_1\}$, $\{q_2, q_3\}$, and $\{q_F\}$ and the equivalence classes for $\equiv_L$ are $\{q_0\}$, $\{q_1, q_2\}$, $\{q_3\}$, and $\{q_F\}$. The instance of vertex cover constructed from the partitions of the equivalences is also shown in Figure 4.1. The vertices are subsets of the state set which are equivalent, either under Π_R or Π_L. Each edge represents a state and is incident to the vertices which represent the subset which contains the state. Based on this instance, there are three possible EQ-reduced automata for N, which are given in Figure 4.2.

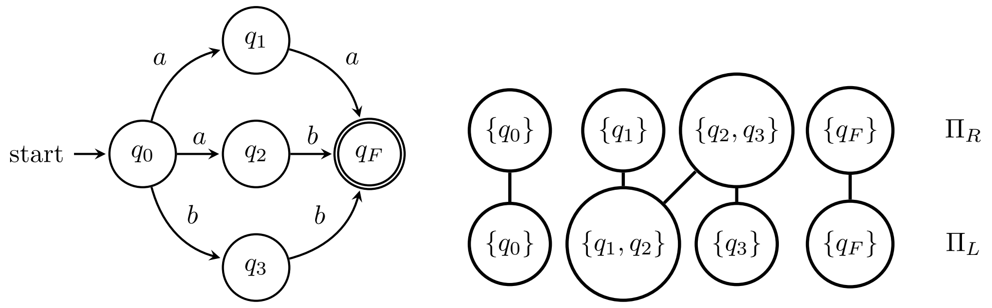

Figure 4.1: The NFA N and the set cover instance $(Q, \Pi_R \cup \Pi_L)$

4.1.2 Reducing NFAs based on preorders

In [8], Champarnaud and Coulon extend the idea of merging equivalent states by using preorder relations defined on the states of an NFA. A *preorder* on a set S is a relation $\leq \subset S \times S$ which satisfies the following for all $a, b \in S$:

- *Reflexivity*: $a \leq a$
- *Transitivity*: if $a \leq b$ and $b \leq c$, then $a \leq c$

A *partial order* is a preorder which is also *antisymmetric*: if $a \leq b$ and $b \leq a$, then $a = b$. If the preorder is symmetric, then by definition, it is an equivalence relation. A partially ordered set, or *poset*, is a pair consisting of a set and a partial order over the set.

The preorders are defined as follows.

$$p \leq_R q \text{ if } \mathcal{L}_R(p) \subseteq \mathcal{L}_R(q)$$
$$p \leq_L q \text{ if } \mathcal{L}_L(p) \subseteq \mathcal{L}_L(q)$$

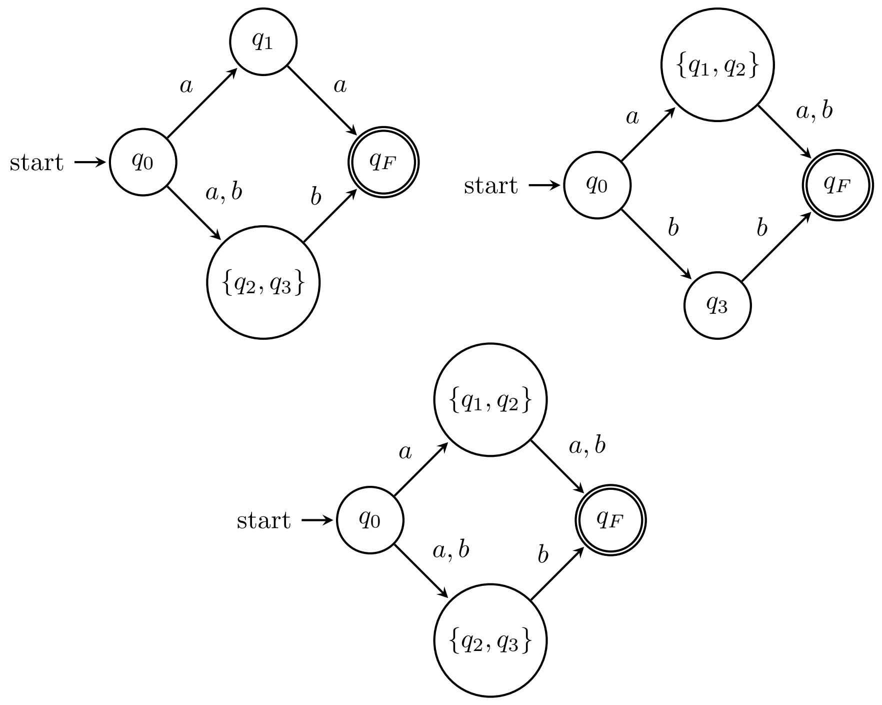

Figure 4.2: EQ-reduced NFAs for the NFA N

Ilie, Navarro, and Yu give an algorithm in [32] which computes preorders over the state set in $O(mn)$ time and $O(n^2)$ space, where n is the number of states and m is the number of transitions.

These preorders induce equivalence relations that are coarser than the equivalence relations defined above. Specifically, we define the equivalence relation $\cong_R$ if $p \leq_R q$ and $q \leq_R p$ and $\cong_L$ if $p \leq_L q$ and $q \leq_L p$. These equivalence relations induce a partition of the state set. We denote by π_L and π_R the partitions of the state set induced by the equivalences $\cong_L$ and $\cong_R$ respectively. The preorders also induce a partial order $\preceq$ on the state set: $p \preceq q$ iff $p \leq_R q$, $p \leq_L q$, and $\mathcal{L}(p,p) = \{\epsilon\}$. The partial order $\preceq$ induces a family π_P of subsets of Q: let the maximal elements of $\preceq$ be $\tilde{q}_1, \tilde{q}_2, ..., \tilde{q}_m$. Then $\pi_P = \{Q_{p_1}, ..., Q_{p_m}\}$, where $Q_{p_i} = \{q \in Q : q \preceq \tilde{q}_i\}$.

The reduction problem then becomes another instance of the minimum set cover problem, with Q as the set of ground elements and $\pi_R \cup \pi_L \cup \pi_P$ as the family of subsets. Each state belongs to exactly one subset in π_R, exactly one subset in π_L, and to at least one subset in π_P, since π_P is in general not a partition of the state set. It is shown in [34] that the optimal reduction using preorders is at least as hard as minimal vertex cover on 3-partite 3-uniform hypergraphs, which models the restricted case of the problem in which π_P is a partition of the state set. As a result, optimal preorder reduction is NP-hard. We call the NFA that is computed through an optimal reduction using preorders a PRE-*reduced NFA.*

Example 4.1.2. The following example is used by Champarnaud and Coulon [8] to demonstrate how preorders can capture a wider range of equivalent states than the equivalences defined in [35]. Let $N = (Q, \Sigma, \delta, q_0, F)$ be as depicted in Figure 4.4. The preorders on Q are as follows:

$$\begin{aligned}\leq_R &= \{(q_1,q_2),(q_2,q_1),(q_3,q_4),(q_5,q_4)\}\\ \leq_L &= \{(q_3,q_4),(q_5,q_4)\}\end{aligned}$$

These preorders give rise to the following equivalence classes for $\cong_R$ and $\cong_L$, respectively:

$$\begin{aligned}\pi_R &= \{\{q_0\},\{q_1,q_2\},\{q_3\},\{q_4\},\{q_5\},\{q_F\}\}\\ \pi_L &= \{\{q_0\},\{q_1\},\{q_2\},\{q_3\},\{q_4\},\{q_5\},\{q_F\}\}\end{aligned}$$

The preorders also induce the partial order $\preceq$, which is given in Figure 4.3 and induces the following family of subsets:

$$\pi_P = \{\{q_0\},\{q_1\},\{q_2\},\{q_3,q_4,q_5\},\{q_F\}\}$$

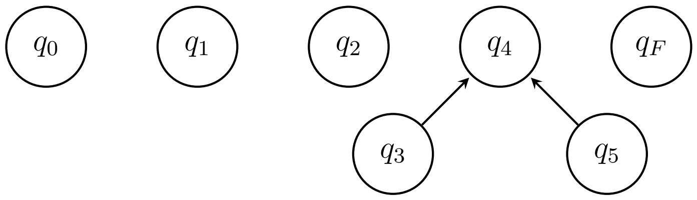

Figure 4.3: The partial order $\preceq$ over Q

Note that we have $q_1 \cong_R q_2$, since $q_1 \leq_R q_2$ and $q_2 \leq_R q_1$. This equivalence would not have been included in $\equiv_R$, since there are no states that are equivalent for q_3 or q_5 under $\equiv_R$. We also have $q_3, q_5 \preceq q_4$, which is another relation that would not have been captured by $\equiv_R$ and $\equiv_L$.

From the hypergraph of the set cover instance $(Q, \pi_R \cup \pi_L \cup \pi_P)$ in Figure 4.5, we can see a vertex cover $\{\{q_0\},\{q_1,q_2\},\{q_3,q_4,q_5\},\{q_F\}\}$. The PRE-reduced NFA is shown in Figure 4.6.

A question that may come to mind is whether for a regular language L, the PRE-reduced automaton for NFAs accepting L all have the same size. We show in the following example that this is not the case.

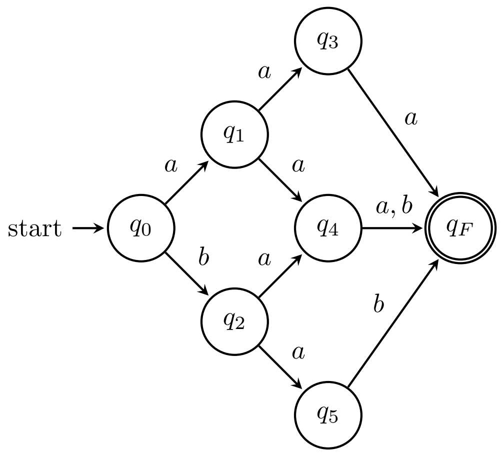

Figure 4.4: The NFA N

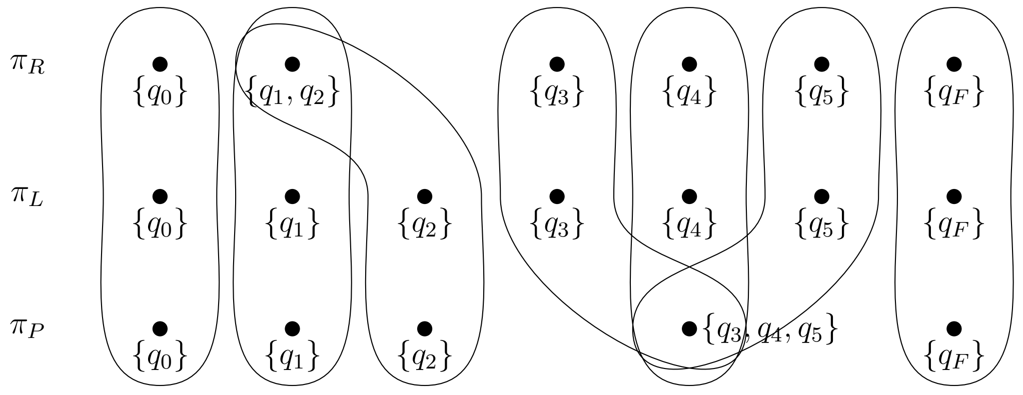

Figure 4.5: The hypergraph for the set cover instance $(Q, \pi_R \cup \pi_L \cup \pi_P)$

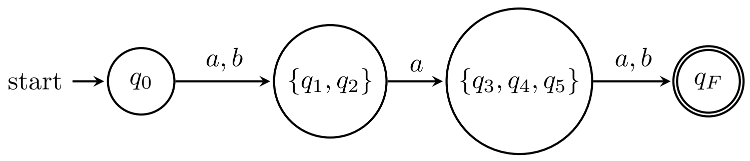

Figure 4.6: The PRE-reduced NFA for N

Example 4.1.3. In Figure 4.7, there are two equivalent NFAs which accept the language $\{a, b\}^2$. The NFA on the right is the PRE-reduced automaton derived from the NFA on the left, since $q_1 \preceq q_2$.

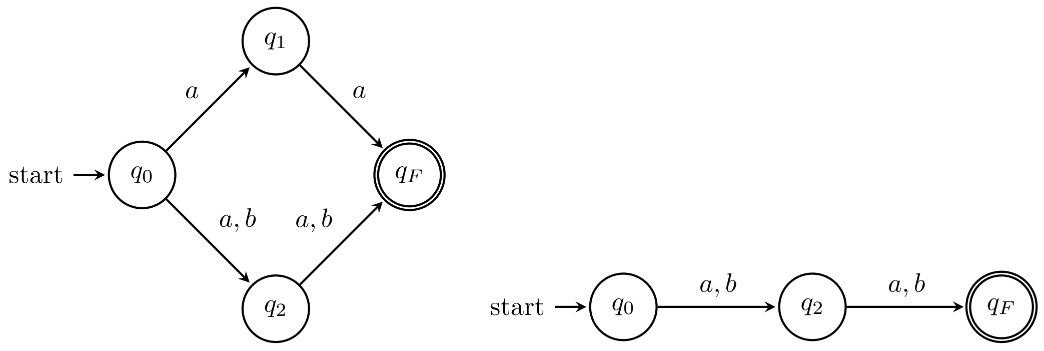

Figure 4.7: NFAs which accept the language $\{a, b\}^2$

Suppose we use the NFAs from Figure 4.7 to construct NFAs which accept $(\{a, b\}^2)^*$. The PRE-reduced NFAs derived from those automata are given in Figure 4.8. The automaton on the left is unable to be reduced further because although $q_1 \leq_R q_2$ and $q_1 \leq_L q_2$, we have $\mathcal{L}(q_2, q_2) \neq \{\epsilon\}$.

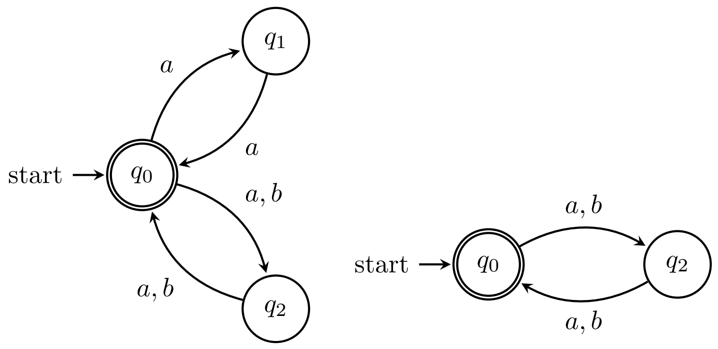

Figure 4.8: NFAs which accept the language $(\{a, b\}^2)^*$

4.2 Complexity of computing PRE-reduced NFAs

In [34], it is shown that when π_P is a partition of the set of states, the problem is at least as hard as the minimum vertex cover problem on 3-partite 3-uniform hypergraphs. As a result, optimal preorder reduction is NP-hard. However, π_P is not guaranteed to be a partition of the state set. While this does not change the hardness of computing the exact solution, this does affect how well we can approximate the problem. When π_P is not a partition of the state set, the resulting hypergraph instance is no longer guaranteed be 3-uniform and can possibly have a worst-case approximation ratio of $O(\log d)$, where d is the size of the largest edge. However, there are families of languages for which the PRE-reduced NFAs that recognize them can be computed in polynomial time. We now give several examples of such families. First, we have the following lemma.

Lemma 4.2.1. *Let N be an NFA. Suppose every state of N belongs to a cycle. Then N can be* PRE*-reduced in polynomial time.*

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$. Consider a state $p \in Q$. Since p belongs to a cycle, $\mathcal{L}(p,p) \neq \{\epsilon\}$. Thus, there is no state $q \in Q$ such that $p \preceq q$. This means that π_P consists of singletons and thus it does not contribute any candidate states to be merged when PRE-reducing N. All possible state merges occur in π_L and π_R, so the problem of computing the PRE-reduced NFA for N can be reduced to the minimum vertex cover problem on a bipartite graph [34]. □

Brzozowski and Cohen define a *regular star language* in [6] to be a language $L \subseteq \Sigma^*$ such that $L = R^*$ for some regular language $R \subseteq \Sigma^*$.

Lemma 4.2.2. *Let $L = R^* \subseteq \Sigma^*$ be a regular star language. Then given a trim NFA $N(R)$ accepting R, a* PRE*-reduced NFA accepting $L = R^*$ can be computed in time polynomial in the number of states and transitions of $N(R)$.*

Proof. Let $N(R) = (Q, \Sigma, \delta, q_0, F)$ denote an NFA accepting R. Recall that a trim NFA is an NFA with no unreachable or dead states. We follow the construction from [29] to construct an NFA $N(R^*)$ which accepts R^*. There are two cases to consider.

The first case is if $\epsilon \in R$. Let $N(R^*) = (Q, \Sigma, \delta', q_0, F)$. Then we define δ' as follows. For every $q \in Q$ and every $a \in \Sigma$,

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{if } q \notin F \\ \delta(q,a) \cup \delta(q_0,a) & \text{if } q \in F \end{cases}$$

Now consider all states $q \neq q_0$ in Q. Since each of these states has a path to a final state and all states are reachable from q_0, then every state $q \neq q_0$ has a path to itself. Therefore, each of these states belongs to a cycle. If q_0 is also contained in a cycle, then by Lemma 4.2.1, this NFA can be PRE-reduced in polynomial time.

If q_0 is not contained in a cycle, observe that $\mathcal{L}_L(q_0) = \{\epsilon\}$. Since q_0 is the initial state and we disallow ϵ-transitions, there are no other states $q \in Q$ such that $\epsilon \in \mathcal{L}_L(q)$. Thus, $q_0 \not\preceq q$ for any $q \in Q$. Since every other state is contained in a cycle, π_P consists of singletons and, thus, this NFA can be PRE-reduced in polynomial time.

The second case is if $\epsilon \notin R$. Let $N(R^*) = (Q \cup \{q_0'\}, \Sigma, \delta', q_0', F \cup \{q_0'\})$. Then we define δ' as follows. For every $q \in Q$ and every $a \in \Sigma$,

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{if } q \notin F \cup \{q_0'\} \\ \delta(q,a) \cup \delta(q_0,a) & \text{if } q \in F \\ \delta(q_0,a) & \text{if } q = q_0' \end{cases}$$

By similar reasoning as the first case, every state $q \in Q$ with the exception of q_0' and q_0 is guaranteed to be contained in a cycle. Clearly, q_0' is not contained in a cycle. By a similar argument as above, we can determine that $q_0' \not\preceq q$ for any $q \neq q_0'$. If q_0 is contained in a cycle, then the lemma follows.

If q_0 is not contained in a cycle, then q_0 is unreachable and can be removed. If q_0 were not unreachable, then there would be a path from some state $p \in Q$ to q_0. But this would imply q_0 was contained in a cycle, since we can reach a final state from q_0 and we can reach p from a final state. After removing q_0, every state except q_0' is contained in a cycle and $q_0' \not\preceq q$ for $q \neq q_0'$ implies that π_P consists of singletons and the lemma follows. $\square$

Example 4.2.1. Let $R = L(b + aa^*b)$ and let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA depicted in Figure 4.9, which accepts R. We build the NFA N' which accepts the language $L = R^* = L((b + aa^*b)^*)$, which is also shown in Figure 4.9.

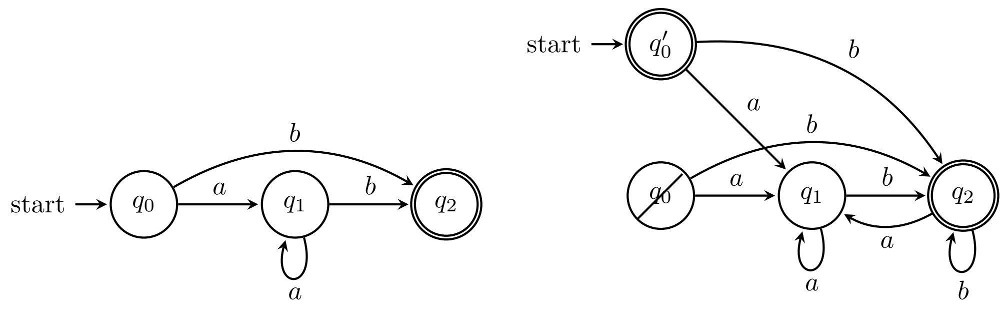

Figure 4.9: NFAs which accept $L(b + aa^*b)$ and $L((b + aa^*b)^*)$

Lemma 4.2.3. *Let $L_1 = R_1^*$ and $L_2 = R_2^*$ be regular star languages with regular languages $R_1, R_2 \subseteq \Sigma^*$. Then given NFAs $N(L_1)$ and $N(L_2)$ as in Lemma 4.2.2 that recognize L_1 and L_2, respectively,* PRE*-reduced NFAs accepting the following languages can be computed in time polynomial in the number of states and transitions of $N(L_1)$ and $N(L_2)$.*

1. $L_1 \cup L_2$

2. $L_1 \cap L_2$

3. $L_1 L_2$

Proof. We give an NFA that accepts each of the above languages by following the constructions given in [29]. These constructions all create new automata with a minimal number of states with respect to the size of $N(L_1)$ and $N(L_2)$. That is, the resultant NFAs are minimal only if the operand NFAs are also minimal. Let $N(L_i) = (Q_i, \Sigma, \delta_i, q_{0,i}, F_i)$ denote the NFA accepting the language R_i^*, for $i = 1, 2$. Let $N' = (Q', \Sigma, \delta', q_0', F')$ denote the new NFA.

1. For N' with $L(N') = L_1 \cup L_2$, let q_0' be a new initial state. Let $Q' = Q_1 \cup Q_2 \cup \{q_0'\}$. For all $q \in Q'$ and $a \in \Sigma$, define the transition function

$$\delta'(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2 \\ \delta_1(q_{0,1}, a) \cup \delta_2(q_{0,2}, a) & \text{if } q = q_0' \end{cases}.$$

   Since both $N(L_1)$ and $N(L_2)$ are NFAs constructed using the method from Lemma 4.2.2, every state $q \neq q_0'$ belongs to a cycle. If $q_{0,1}$ and $q_{0,2}$ do not belong to a cycle, then they are unreachable and can be removed. Consider $q_{0,1}$ and suppose it is reachable by some state $p \in Q'$. There is a path from $q_{0,1}$ to a final state in Q_1. But there is a path from any final state in Q_1 to p. Thus, $q_{0,1}$ would be contained in a cycle. The same reasoning applies to $q_{0,2}$.

   Only the new initial state does not belong to a cycle. Note that $\mathcal{L}_L(q_0') = \{\epsilon\}$ and since q_0' is the initial state and we disallow ϵ-transitions, there are no other states $q \in Q'$ such that $\epsilon \in \mathcal{L}_L(q)$. Thus, $q_0' \not\preceq q$ for any $q \in Q'$. Since every other state is contained in a cycle, π_P consists of singletons.

2. Let N' recognize $L_1 \cap L_2$. Then N' is the cross-product of $N(L_1)$ and $N(L_2)$, defined as follows:

$$\begin{aligned} Q' &= Q_1 \times Q_2 \\ F' &= F_1 \times F_2 \\ \delta'(\langle q_1, q_2\rangle, a) &= \langle \delta_1(q_1, a), \delta_2(q_2, a)\rangle, \text{ where } q_1 \in Q_1 \text{ and } q_2 \in Q_2 \\ q_0' &= \langle q_{0,1}, q_{0,2}\rangle \end{aligned}$$

Every state still belongs to a cycle in the cross-product of the two NFAs since every state already belonged to a cycle in the original respective NFA, with the possible exception of the initial states. However, $\mathcal{L}_L(q'_0) = \{\epsilon\}$ and since q'_0 is the initial state and we disallow ϵ-transitions, there are no other states $q \in Q'$ such that $\epsilon \in \mathcal{L}_L(q)$. Thus, $q'_0 \not\preceq q$ for any $q \in Q'$. Since every other state is contained in a cycle, π_P consists of singletons.

3. To build N' with $L(N') = L_1L_2$, connect the final states of $N(L_1)$ to the states that follow the initial transitions of $N(L_2)$: For all $q \in F_1$ and $a \in \Sigma$, we add the following transitions
$$\delta'(q,a) = \delta_2(q_{0,2}, a).$$
Since the only new transitions are from the final states of $N(L_1)$ to states of $N(L_2)$, every state belongs to a cycle with the possible exception of the initial states $q_{0,1}$ and $q_{0,2}$. If $q_{0,1}$ does not belong to a cycle, then $\mathcal{L}_L(q_{0,1}) = \{\epsilon\}$ and there are no states $q \in Q'$ with $q_{0,1} \preceq q$ since $\epsilon \notin \mathcal{L}_L(q)$ for any $q \neq q_{0,1}$. If $q_{0,2}$ is not contained in a cycle, then it is unreachable and can be removed. If it were reachable by some state $p \in Q'$, then there would be a path from $q_{0,2}$ to a final state and a path from that final state to p, implying that $q_{0,2}$ belonged to a cycle. Since every other state is contained in a cycle, π_P consists of singletons.

Since for each of these NFAs, π_P consists of singletons, they can all be PRE-reduced in polynomial time. □

We can extend this property of NFAs accepting star languages to a special class of NFAs which accepts exactly the class of union-free regular languages defined by Nagy in [56]. A *union-free regular language* is a regular language which is specified by a regular expression that does not contain the union operator. A *1-cycle-free-path automaton* (1cfpa) is an NFA which has a unique cycle-free accepting path from each of its states. The shortest word accepted by a 1cfpa is unique and is called its *backbone*. This word is accepted by the cycle-free path from the initial state to the final state. The other parts of the automaton are referred to as *loops* and *subloops*.

Example 4.2.2. Let A be the 1-cycle-free-path automaton in Figure 4.10. The automaton A accepts the language described by the regular expression
$$\alpha = (b(b^*(ab^*a)^*)^*a)^*aa^*bba(bb^*ab)^*b.$$
The backbone of A is *abbab* and the cycle-free path is $q_0q_1q_2q_3q_4q_F$.

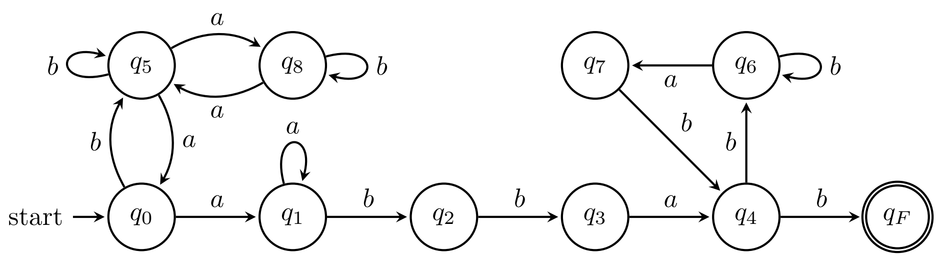

Figure 4.10: A 1-cycle-free-path automaton which accepts the language $L(\alpha)$

Theorem 4.2.4. *Let A be a 1-cycle-free-path automaton. Then a* PRE*-reduced NFA accepting $L(A)$ can be computed in time polynomial in the number of states and transitions of A.*

Proof. Let $A = (Q, \Sigma, \delta, q_0, \{q_F\})$. A state in A belongs to either the cycle-free path from q_0 to q_F or a loop or subloop. If a state belongs to a loop or subloop, it is contained in a cycle. By Lemma 4.2.1, any state that belongs to a cycle is not the child of any other state over the partial order $\preceq$.

Now consider a state $q \in Q$ on the cycle-free path from q_0 to q_F and let w be the backbone of A and $n = |w|$. Suppose there exists a state $p \in Q$ such that $p \preceq q$. Note that p cannot be in a loop or subloop, since $\mathcal{L}(p,p) = \{\epsilon\}$, and hence, p must be in the cycle-free path P from q_0 to q_F. Suppose p appears before q on this path. Then w_p, the subword of the backbone w that takes A from q_0 to p by following states of P must belong to $\mathcal{L}_L(q)$, since $p \preceq q$ requires that $\mathcal{L}_L(p) \subseteq \mathcal{L}_L(q)$. However, this is impossible as P has no cycles. A similar argument holds if p appears after q.

Thus no state on the cycle-free path is a child of any other state over the partial order $\preceq$ and thus, no states of A are related over $\preceq$. Thus, π_P consists of singletons and does not contribute any candidate states to be merged when PRE-reducing N so all possible state merges occur in π_L and π_R. Thus the PRE-reduced NFA for A can be computed in polynomial time. □

Chapter 5

An approximation algorithm based on Lovász's theorem

5.1 Lovász's theorem

When π_P is a partition of the state set, the problem of computing a PRE-reduced NFA reduces to the minimum vertex cover problem on 3-partite 3-uniform hypergraphs. In this section, we introduce Lovász's theorem, which tells us it becomes possible to approximate the PRE-reduced NFA to within a constant factor when π_P is a partition. Since the original proof of the theorem is in Hungarian, for completeness, we present Theorem 5.1.1, which is a slightly more general version of Lovász's theorem that includes hypergraphs with weights.

Recall that $\tau(H)$ denotes the value of a minimal vertex cover of H and $\tau^*(H)$ denotes the value of a minimal fractional vertex cover of H. Let $\mathbb{Z}$ denote the set of integers, let $\mathbb{Q}$ denote the set of rational numbers, and let $\mathbb{R}$ denote the set of real numbers.

Theorem 5.1.1 ([48]). *Let $H = (V, E)$ be a weighted k-partite k-uniform hypergraph with weight function $w : V \to \mathbb{Z}$. Then $\frac{\tau(H)}{\tau^*(H)} \leq \frac{k}{2}$.*

Proof. Let $H = (V, E)$ be an k-partite k-uniform hypergraph with partition $(V_1, ..., V_k)$ given. Since $\tau^*(H)$ is the value of an optimal solution for a linear programming problem with integral coefficients, there exists a minimal fractional cover g for H with respect to w such that $g(v) \in \mathbb{Q}$ for every $v \in V$. We can choose d such that $g(v) \cdot d$ is integral for every $v \in V$.

For all integers $k \geq 2$ and $m \geq 0$, as shown in Lemma 5.1.2 below, there exists an $k \times (m + 1)$ matrix A such that

1. every row of A is a permutation of $\{0, 1, ..., m\}$ and

2. the sum of every column is at most $\left\lceil \frac{km}{2} \right\rceil$.

Let $m = \left\lfloor \frac{2}{k}(d-1) \right\rfloor$. For every $0 \le j \le m$, we define T_j to be

$$T_j = \bigcup_{i=1}^{k} \{v \in V_i : d \cdot g(v) > a_{i,j}\}$$

where $a_{i,j}$ is the (i,j)-th entry of the matrix A.

We now show that every T_j is a vertex cover. Suppose not and that there exists an edge $e = \{v_1, ..., v_k\} \in E$ such that $e \cap T_j = \emptyset$. Then that means $d \cdot g(v_i) \le a_{i,j}$ for every $1 \le i \le k$ and hence,

$$\sum_{i=1}^{k} g(v_i) = \sum_{i=1}^{k} \frac{d \cdot g(v_i)}{d} \le \frac{\sum_{i=1}^{k} a_{i,j}}{d} \le \frac{\left\lceil \frac{km}{2} \right\rceil}{d} = \frac{\left\lceil \frac{k}{2} \left\lfloor \frac{2}{k}(d-1) \right\rfloor \right\rceil}{d} \le \frac{d-1}{d} < 1$$

which is a contradiction, since g is a fractional cover.

Since each row of A is a permutation of $\{0, 1, ..., m, \}$, by the way each cover T_j is defined, a vertex $v \in V$ is contained in at most $d \cdot g(v)$ vertex covers. So for every $v \in V$,

$$w(v) \cdot |\{0 \le j \le m : v \in T_j\}| \le w(v) g(v) d.$$

Then $\sum_{j=0}^{m} w(T_j) \le \sum_{v \in V} w(v) g(v) d = d\tau^*(H)$. Since $\tau(H)$ can be at most the average of the values of the fractional covers T_j, we have

$$\tau(H) \le \frac{\sum_{j=0}^{m} w(T_j)}{m+1} \le \frac{d\tau^*(H)}{\frac{2(d-1)}{k}} = \frac{k}{2} \cdot \frac{d}{d-1} \cdot \tau^*(H)$$

for all sufficiently large d. Then taking the limit as $d \to \infty$, we have $\tau(H) \le \frac{k}{2}\tau^*(H)$. □

We now give an explicit construction of the matrix A mentioned in the above proof.

Lemma 5.1.2. *For all integers $k \ge 2$ and $m \ge 0$, there exists an $k \times (m+1)$ matrix A such that*

1. *every row of A is a permutation of $\{0, 1, ..., m\}$ and*

2. *the sum of every column is at most $\left\lceil \frac{km}{2} \right\rceil$.*

Proof. Let R be the following $2 \times (m+1)$ matrix.

$$R = \begin{bmatrix} 0 & 1 & \cdots & m \\ m & m-1 & \cdots & 0 \end{bmatrix}$$

Clearly, every row of R is a permutation of $\{0, 1, ...m\}$. For the ith column of R, $i = 0, 1, ..., m$, the sum of its entries is

$$i + (m - i) = m = \frac{2m}{2} \leq \left\lceil \frac{km}{2} \right\rceil.$$

For the case when k is even, we define A to be R repeated $\frac{k}{2}$ times. Then for $i = 0, 1, ..., m$, the sum of the entries of the ith column of A is

$$\frac{k}{2} \cdot \frac{2m}{2} = \frac{km}{2} \leq \left\lceil \frac{km}{2} \right\rceil.$$

Now, consider when k is odd. Let the first $k-3$ rows of A be R repeated $\frac{k-3}{2}$ times. For $i = 0, 1, ..., m$, the sum of the entries of the ith column of the first $k-3$ rows is $\frac{(k-3)m}{2}$. If m is even, then we define the last three rows of A to be

$$\begin{bmatrix} 0 & 1 & \cdots & \frac{m}{2} & \frac{m}{2}+1 & \cdots & m \\ \frac{m}{2} & \frac{m}{2}+1 & \cdots & m & 0 & \cdots & \frac{m}{2}-1 \\ m & m-2 & \cdots & 0 & m-1 & \cdots & 1 \end{bmatrix}$$

For $i = 0, 1, ..., \frac{m}{2}$, the sum of the entries of the last three rows of the ith column of A is

$$i + \left(\frac{m}{2} + i\right) + (m - (2i)) = \frac{m}{2} + m + 2i - 2i = \frac{3m}{2}.$$

And for $i = \frac{m}{2} + 1, ..., m$, the sum of the entries of the last three rows of the ith column of A is

$$i + \left(i - \left(\frac{m}{2} + 1\right)\right) + (2m - 2i + 1) = 2i - 2i - \frac{m}{2} + 2m - 1 + 1 = \frac{3m}{2}.$$

Then the sum for the entries in the entire column is

$$\frac{(k-3)m}{2} + \frac{3m}{2} = \frac{km}{2}.$$

Finally, we consider the case when m is odd. The last three rows of A are defined as follows.

$$\begin{bmatrix} 0 & 1 & \cdots & \frac{m-1}{2} & \frac{m+1}{2} & \cdots & m-1 & m \\ \frac{m+1}{2} & \frac{m+3}{2} & \cdots & m & 0 & \cdots & \frac{m-3}{2} & \frac{m-1}{2} \\ m & m-2 & \cdots & 1 & m-1 & \cdots & 2 & 0 \end{bmatrix}$$

For $i = 0, 1, ..., \frac{m-1}{2}$, the sum of the entries of the last three rows of the ith column of A

is
$$i+\left(\frac{m+1}{2}+i\right)+(m-(2i))=\frac{m+1}{2}+m+2i-2i=\frac{3m}{2}+\frac{1}{2}.$$
And for $i=\frac{m+1}{2},...,m$, the sum of the entries of the last three rows of the ith column of A is
$$i+\left(i-\frac{m+1}{2}\right)+(2m-2i)=2i-2i-\frac{m+1}{2}+2m=\frac{3m}{2}+\frac{1}{2}.$$
Then the sum for the entries in an entire column is
$$\frac{(k-3)m}{2}+\frac{3m}{2}+\frac{1}{2}=\frac{km}{2}+\frac{1}{2}\leq\left\lceil\frac{km}{2}\right\rceil.$$
□

5.2 The algorithm

In this section, we design a $\frac{k}{2}$-approximation algorithm for the vertex cover problem on k-partite k-uniform hypergraphs based on Lovász's theorem. The intuition behind the algorithm is the idea that each column of the matrix A in the proof of the theorem defines a vertex cover, with each row representing one partition of the hypergraph. This gives a total of $m+1$ possible vertex covers. However, the number of *different* vertex covers in the above set could be much smaller than $m+1$. More specifically, for a hypergraph $H=(V,E)$ with partition $(V_1,V_2,...,V_k)$, the number of different vertex covers defined by the columns of A is at most $|V|+|V_k|+2$. We show this in the following lemma.

Lemma 5.2.1. *Let $H=(V,E)$ be a k-partite k-uniform hypergraph with partition $(V_1,...,V_k)$, with $|V_1|\geq|V_2|\geq\cdots\geq|V_k|$. Then there are at most $|V|+|V_k|+2$ different vertex covers given by the matrix A defined in the proof of Theorem 5.1.1.*

Proof. Let $g:V\rightarrow\mathbb{R}$ be a minimal fractional cover of H. Choose d such that $m=\left\lfloor\frac{2}{k}(d-1)\right\rfloor$ is odd. Every column a_j of A forms a cover T_j as shown by Lovász, where
$$T_j=\bigcup_{i=1}^{k}\{v\in V_i:d\cdot g(v)>a_{i,j}\}.$$

To determine the number of different vertex covers defined by A, we scan the columns of A from left to right and note when a vertex cover changes. Assume that the vertices of each partition $V_i=\{v_{i,1},v_{i,2},...,v_{i,|V_i|}\}$ are ordered such that $g(v_{i,l})\leq g(v_{i,l+1})$, $1\leq l<|V_i|$.

We begin with the first vertex cover T_0, which is defined by the first column of A. Observe that while moving along the rows of A from left to right, the cover T_j changes at a column a_j when there is a value $a_{i,j}$ and vertex $v \in V_i$ such that $d \cdot g(v) = a_{i,j}$; thus, this condition contributes at most $\sum_{i=1}^{k-1} |V_i| + 1$ possible different vertex covers, including the first vertex cover $T = T_0$.

When k is odd, there are two additional cases to consider: the $(k-1)$-th row contributes one additional cover at the $\left(\frac{m}{2}+1\right)$-th column, where the entries in A change from m to 0. Second, the kth row needs to be handled differently from the other rows since for $j < \frac{m}{2}$, $a_{k,j} - 1 = a_{k,j+\frac{m}{2}}$. Thus if we consider all entries $a_{k,j}$ such that $d \cdot g(v_{k,l}) \leq a_{k,j} < d \cdot g(v_{k,l+1})$ for some $v_{k,l} \in V_k$ and $1 \leq l < |V_k|$, half of these entries are in the left half of the matrix and the other half are in the right half of A. Since these entries are not contiguous in the matrix, the cover T changes twice as many times for each vertex $v \in V_k$, once in the left half and once in the right half. This defines $2(|V_k|+1)$ different covers. However, two of these covers coincide with covers T_0 and $T_{\frac{m}{2}}$, which were already defined for V_1 and V_{k-1}. Thus, the kth row gives at most $2|V_k|$ additional covers.

Therefore, there are at most $\sum_{i=1}^{k-1} |V_i| + 2|V_k| + 2 = |V| + |V_k| + 2$ different covers that can be formed from the columns of A. □

Example 5.2.1. Let $H = (V, E)$ be a hypergraph, with the following:

$$\begin{aligned} \text{vertices } V &= \{1,2,3,4,5,6,7\} \text{ with partition } (\{1,2,3\},\{4,5\},\{6,7\}) \\ \text{hyperedges } E &= \{\{2,4,6\},\{3,4,7\},\{1,5,7\}\} \\ \text{fractional vertex cover } g^T &= \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & 0 & \frac{2}{3} & \frac{1}{3} & 0 & \frac{1}{3} \end{bmatrix} \end{aligned}$$

Choose $d = 9$ so that $m = \lfloor \frac{2}{3}(d-1) \rfloor = 5$, which gives us $d \cdot g^T = \begin{bmatrix} 3 & 3 & 0 & 6 & 3 & 0 & 3 \end{bmatrix}$. Let A be the following matrix:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 0 & 1 & 2 \\ 5 & 3 & 1 & 4 & 2 & 0 \end{bmatrix}$$

The vertex covers for H derived from this matrix are $T_0 = T_1 = \{1,2,4\}$, $T_2 = \{1,2,4,7\}$, $T_3 = \{4,5\}$, $T_4 = T_5 = \{4,5,7\}$.

Since the number of different vertex covers can be much lower than $m+1$, rather than building the whole matrix A, we compute only those columns and the corresponding vertex covers for the different values $g(v)$ in each partition. After constructing every

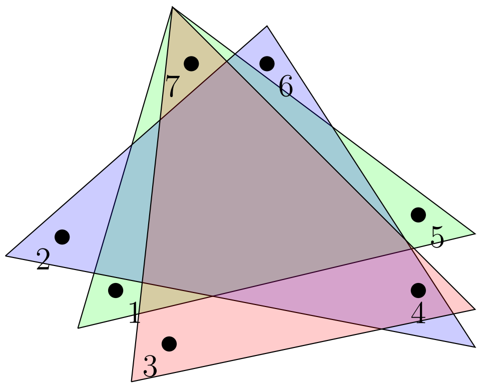

Figure 5.1: The hypergraph $H = (V, E)$

vertex cover, the algorithm selects the cover with the smallest weight. The algorithm is presented below as Algorithm 1.

Since the algorithm needs to solve the fractional vertex cover problem, the time complexity of kPartHypVC matches that of a $\frac{k}{2}$-approximation algorithm of Krivelevich for vertex cover on r-uniform k-colourable hypergraphs [46]. This is because computing the solution to a linear program dominates the time complexity of both algorithms. However, the number of vertex covers that are computed by our algorithm is $|V|+|V_k|+2$, while Krivelevich's algorithm needs to compute $O(k|V|)$ vertex covers.

Proposition 5.2.2. *Algorithm* kPartHypVC *has time complexity* $O(n^7 \log^2 T)$*, where* $n = |V|$ *and* $T = \max_{v\in V} w(v)$*.*

Proof. An optimal solution of a linear program with n variables can be computed via Karmarkar's algorithm. Karmarkar's algorithm performs $O(n^5 \log T^*)$ arithmetic operations on numbers with $O(n^2 \log T^*)$ bits for a total of $O(n^7 \log^2 T^*)$ bit operations, where T^* is the maximum absolute value of the input numbers [39]. Since the linear program formulation of the fractional vertex cover problem on a hypergraph $H = (V, E)$ has n variables and the largest input number is the maximum weight of the vertices, Karmarkar's algorithm can compute a fractional vertex cover in $O(n^7 \log^2 T)$ time.

Choosing d based on g can be done by finding the least common multiple (lcm) of n numbers. It is well known that $\text{lcm}(a, b) = \frac{ab}{\gcd(a,b)}$, where $\gcd(a, b)$ is the greatest common divisor of positive values a and b. We assume $a \leq b$ without loss of generality. It is well known that multiplication and division requires $O(\log^2 b)$ time. The greatest common divisor of two numbers a, b can be computed using the Euclidean algorithm, which has time complexity $O(\log b)$ [10]. Note that $\text{lcm}(a, b, c) = \text{lcm}(a, \text{lcm}(b, c))$. Since there are n numbers to consider, finding the lcm of all n numbers requires finding the lcm n times and performing n multiplications and divisions. Thus, this requires $O(n \log^2 h)$ time in total,

Algorithm 1 Approximating vertex cover on k-partite k-uniform hypergraphs

```
function KPARTHYPVC($H = (V, E), k, (V_1, V_2, ..., V_k)$)
    Let $(V_1, V_2, ..., V_k)$ be the given partition of $V$
    Find an optimal fractional cover $g : V \to \mathbb{R}$, with $g(v) = \frac{v_a}{v_b}, v_a, v_b \in \mathbb{Z}$ for every $v \in V$
    Find the least common multiple $c$ of the denominators of $g(v) = \frac{v_a}{v_b}$ for every $v \in V$
    $d \leftarrow kc$
    $m \leftarrow 2c - 1$
    $X \leftarrow \emptyset$
    if $k$ is odd then $k' \leftarrow k - 2$
    else $k' \leftarrow k$                    ▷ The last two rows of the odd case are handled separately
    for $r \leftarrow 1, ..., k'$ do
        for $i \in \{d \cdot g(v) \le m : v \in V_r\} \cup \{0\}$ do
            if $r$ is odd then $i' \leftarrow i$
            else $i' \leftarrow m - i$
            for $j \leftarrow 1, ..., k'$ do
                if $j$ is odd then $x[j] \leftarrow i'$
                else $x[j] \leftarrow m - i'$
            $X \leftarrow X \cup \{x\}$
    if $k$ is odd then
        for $i \in \{d \cdot g(v) \le m : v \in V_{k-1}\} \cup \{0\}$ do
            if $i < \frac{m}{2}$ then                    ▷ Handling the row which begins in the middle
                for $j \leftarrow 1, ..., k - 2$ do
                    if $j$ is odd then $x[j] \leftarrow i + \frac{m+1}{2}$
                    else $x[j] \leftarrow \frac{m-1}{2} - i$
            else
                for $j \leftarrow 1, ..., k - 2$ do
                    if $j$ is odd then $x[j] \leftarrow \frac{m+1}{2} - i$
                    else $x[j] \leftarrow i + \frac{m-1}{2}$
            $X \leftarrow X \cup \{x\}$
        for $i \in \{d \cdot g(v) \le m : v \in V_k\}$ do
            $i' \leftarrow i - 1$                    ▷ Handling the split row
            if $i$ is odd then
                for $j \leftarrow 1, ...k - 2$ do
                    if $j$ is odd then $x[j] \leftarrow \frac{m-i}{2}$, $y[j] \leftarrow m - \frac{i'}{2}$
                    else $x[j] \leftarrow \frac{m+1}{2}$, $y[j] \leftarrow 2i'$
            else
                for $j \leftarrow 1, ...k - 2$ do
                    if $j$ is odd then $x[j] \leftarrow m - \frac{i}{2}$, $y[j] \leftarrow \frac{m-i'}{2}$
                    else $x[j] \leftarrow 2i$, $y[j] \leftarrow \frac{m+i'}{2}$
            $X \leftarrow X \cup \{x, y\}$
        for $x \in X$ do                    ▷ The last two entries of each column are computed here
            $i \leftarrow x[0]$
            if $i \le \frac{m}{2}$ then $x[k-1] \leftarrow \frac{m+1}{2} + i$, $x[k] \leftarrow m - 2i$
            else $x[k-1] \leftarrow i - \frac{m+1}{2}$, $x[k] \leftarrow 2(m - i)$
    $\mathcal{T} \leftarrow \{T(x) : x \in X\}$, where $T(x) = \bigcup_{j=1}^{k} \{v \in V_j : d \cdot g(v) > x_j\}$
    $C \leftarrow T \in \mathcal{T}$ with minimal weight
    return $C$
```

where $h = \max_{v \in V} \left\{ v_b : g(v) = \frac{v_a}{v_b} \right\}$. By the above discussion on Karmarkar's algorithm, $\log h = \log T$, the maximum size of the numbers in the linear program. This gives time complexity $O(n \log T)$ for finding the lcm.

The algorithm goes through each vertex of each partition of the hypergraph. The computation of the columns for each cover is done in $O(k)$ time, i.e. linear in the length of the column. In total, the algorithm computes $O(n)$ different columns. Thus, this step has time complexity $O(kn)$. Building each cover requires $O(n)$ time and there are $O(n)$ covers, which takes $O(n^2)$ time. Finally, choosing the subset T with minimal weight can be done in time linear in the number of vertex covers, which is $O(n)$. Thus, the time complexity of the algorithm is $O(n^7 \log^2 T)$. $\square$

5.3 Applying Lovász's theorem to graphs

We now consider using Lovász's theorem to approximate vertex covers on graphs. Lovász's theorem relies on the number of partitions in the hypergraph to be the same as the size of the hyperedges. We can prove the same bounds as in [46] by relaxing the conditions on the columns to be included in the vertex cover. As these results are adapted from Lovász's theorem, the proofs will generally follow similar lines of argument as the proof of Theorem 5.1.1.

Theorem 5.3.1. *Let $G = (V, E)$ be a 3-partite graph with weight function $w : V \to \mathbb{Z}$. Then,*

$$\frac{\tau(G)}{\tau^*(G)} \leq \frac{4}{3}.$$

This bound is tight.

Proof. Let $G = (V, E)$ be a 3-partite graph with partition (V_1, V_2, V_3). Since $\tau^*(G)$ is the value of an optimal solution for a linear programming problem with integral coefficients, there exists a fractional cover $g : V \to \mathbb{R}$ such that $g(v) \in \mathbb{Q}$ for every vertex $v \in V$. Therefore, we can choose an integer d such that $d \cdot g(v)$ is integral for every vertex $v \in V$.

Let $m = \left\lfloor \frac{3}{4}(d-1) \right\rfloor$. Define a $3 \times (m+1)$ matrix A by

$$\begin{bmatrix} A_{0,\frac{1}{3}} \\ A_{\frac{1}{3},\frac{2}{3}} \\ A_{1,\frac{2}{3}} \end{bmatrix} = \begin{bmatrix} 0 & 1 & \cdots & \left\lfloor \frac{m}{3} \right\rfloor - 1 & \left\lfloor \frac{m}{3} \right\rfloor \\ \left\lceil \frac{m}{3} \right\rceil & \left\lceil \frac{m}{3} \right\rceil + 1 & \cdots & \left\lfloor \frac{2m}{3} \right\rfloor - 1 & \left\lfloor \frac{2m}{3} \right\rfloor \\ m & m-1 & \cdots & \left\lceil \frac{2m}{3} \right\rceil + 1 & \left\lceil \frac{2m}{3} \right\rceil \end{bmatrix}$$

$$A = \begin{bmatrix} A_{0,\frac{1}{3}} & A_{\frac{1}{3},\frac{2}{3}} & A_{1,\frac{2}{3}} \\ A_{\frac{1}{3},\frac{2}{3}} & A_{1,\frac{2}{3}} & A_{0,\frac{1}{3}} \\ A_{1,\frac{2}{3}} & A_{0,\frac{1}{3}} & A_{\frac{1}{3},\frac{2}{3}} \end{bmatrix}$$

Each row of A is a permutation of $\{0, 1, \ldots, m\}$. Each column of A contains entries which can be written as i, $\left\lceil \frac{m}{3} \right\rceil + i$, and $m - i$, with $0 \leq i \leq \left\lfloor \frac{m}{3} \right\rfloor$. Then,

$$\begin{aligned} i + \left\lfloor \frac{m}{3} \right\rfloor + i \leq 3 \left\lfloor \frac{m}{3} \right\rfloor &\leq \left\lceil \frac{4m}{3} \right\rceil \\ \left\lfloor \frac{m}{3} \right\rfloor + i + m - i \leq \left\lfloor \frac{m}{3} \right\rfloor + m &\leq \left\lceil \frac{4m}{3} \right\rceil \\ i + m - i \leq m &\leq \left\lceil \frac{4m}{3} \right\rceil \end{aligned}$$

Thus, in each column, the sum of any two entries is at most $\left\lceil \frac{4m}{3} \right\rceil$.

For each column j of A, define a set

$$T_j = \bigcup_{i=1}^{3} \{ v \in V_i : d \cdot g(v) > a_{i,j} \}$$

We claim that every T_j is a cover. Suppose it is not and $e = (u, v)$ is an edge not covered by T_j. Let $u \in V_k$ and $v \in V_{k'}$, with $1 \leq k, k' \leq 3$ and $k \neq k'$. Then,

$$g(u) + g(v) = \frac{d \cdot g(u) + d \cdot g(v)}{d} \leq \frac{a_{k,j} + a_{k',j}}{d} \leq \frac{\left\lceil \frac{4m}{3} \right\rceil}{d} = \frac{\left\lceil \frac{4}{3} \left\lfloor \frac{3}{4}(d-1) \right\rfloor \right\rceil}{d} \leq \frac{d-1}{d} < 1$$

which is a contradiction since g is a fractional cover.

Since each row of A is a permutation of $\{0, 1, \ldots, m\}$, we have

$$w(v) \cdot |\{0 \leq j \leq m : v \in T_j\}| \leq w(v) g(v) d$$

for every vertex $v \in V$. Then $\sum_{j=0}^{m} w(T_j) \leq \sum_{v \in V} d \cdot g(v) = d\tau^*(G)$. Since every T_j is a

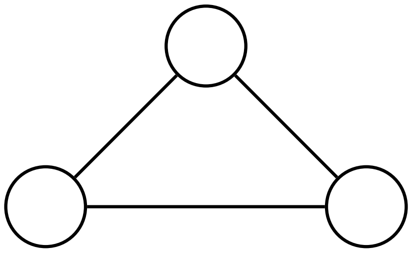

Figure 5.2: The complete graph K_3

cover, we have

$$\tau(G) \leq \frac{\sum_{j=0}^{m} w(T_j)}{m+1} \leq \frac{d\tau^*(G)}{\frac{3}{4}(d-1)}$$

which, if $d \to \infty$ gives

$$\frac{\tau(G)}{\tau^*(G)} \leq \frac{4}{3}.$$

To see that the bound is tight, consider the complete graph K_3 in which every vertex has weight 1, which is shown in Figure 5.2. In the optimal fractional vertex cover g, $g(v) = \frac{1}{2}$ for every $v \in V(K_3)$ and so $\tau^*(K_3) = |g| = \frac{3}{2}$. But a minimal vertex cover must contain at least two of the three vertices, so $\tau(K_3) = 2$. Thus, the approximation ratio of the fractional cover is

$$\frac{\tau(K_3)}{\tau^*(K_3)} = \frac{2}{\frac{3}{2}} = \frac{4}{3}.$$

□

We now describe an algorithm with a slightly better approximation ratio for 3-partite graphs.

Theorem 5.3.2. *Let $G = (V, E)$ be a 3-partite graph. Then,*

$$\frac{\tau(G)}{\tau^*(G)} \leq \frac{4}{3} - \frac{1}{3} \cdot \frac{|C_1|}{\tau^*(G)},$$

where C_1 is the set of vertices in the fractional cover with value 1.

Proof. Let $G = (V, E)$ be a 3-partite graph with partition (V_1, V_2, V_3) and let $g : V \to \mathbb{R}$ be a fractional cover with value $\tau^*(G)$. Since the solution to the vertex cover problem on graphs is half-integral [57], we can immediately choose vertices $v \in V$ with $g(v) = 1$ to be included into the cover. Let C_1 be the set of these vertices. We can also disregard any vertices with $g(v) = 0$. This leaves vertices with $g(v) = \frac{1}{2}$ and we let $V_{\frac{1}{2}} = \{v \in V : g(v) = \frac{1}{2}\}$.

Let $G_{\frac{1}{2}} = (V_{\frac{1}{2}}, E_{\frac{1}{2}})$ be the subgraph of G induced by $V_{\frac{1}{2}}$. Then $\tau^*(G_{\frac{1}{2}}) = \frac{1}{2}|V_{\frac{1}{2}}|$. For $i = 1, 2, 3$, we let $V_{i,\frac{1}{2}}$ denote the set of vertices $v \in V_i$ with $g(v) = \frac{1}{2}$. Without loss of generality, let $|V_{1,\frac{1}{2}}| \leq |V_{2,\frac{1}{2}}| \leq |V_{3,\frac{1}{2}}|$. Then $\tau(G_{\frac{1}{2}}) \leq |V_{1,\frac{1}{2}}| + |V_{2,\frac{1}{2}}| \leq \frac{2}{3}|V_{\frac{1}{2}}|$. This gives us

$$\frac{\tau(G_{\frac{1}{2}})}{\tau^*(G_{\frac{1}{2}})} \leq \frac{\frac{2}{3}|V_{\frac{1}{2}}|}{\frac{1}{2}|V_{\frac{1}{2}}|} = \frac{4}{3}.$$

Therefore,

$$\tau(G) \leq \tau(G_{\frac{1}{2}}) + |C_1| \leq \frac{4}{3}\tau^*(G_{\frac{1}{2}}) + |C_1| \leq \frac{4}{3}(\tau^*(G) - |C_1|) + |C_1| = \frac{4}{3}\tau^*(G) - \frac{1}{3}|C_1|.$$

Thus,

$$\frac{\tau(G)}{\tau^*(G)} \leq \frac{4}{3} - \frac{1}{3} \cdot \frac{|C_1|}{\tau^*(G)}.$$

□

Finally, we can generalize the proof of Theorem 5.3.1 to $k \geq 4$.

Theorem 5.3.3. *Let $G = (V, E)$ be a k-partite graph with weight function $w : V \to \mathbb{Z}$. Then,*

$$\frac{\tau(G)}{\tau^*(G)} \leq \frac{2(k-1)}{k}.$$

Proof. Let $G = (V, E)$ be a k-partite graph with partition $(V_1, V_2, \ldots, V_k)$. Since $\tau^*(G)$ is the value of an optimal solution for a linear programming problem with integral coefficients, there exists a fractional cover $g : V \to \mathbb{R}$ such that $g(v) \in \mathbb{Q}$ for every vertex $v \in V$. Therefore, we can choose an integer d such that $d \cdot g(v)$ is integral for vertex $v \in V$. We also include the restriction that $2(k-1)$ divides $d-1$.

Let $m = \frac{k}{2(k-1)}(d-1)$ and for $1 \leq i < j \leq k-1$, we define the following:

$$A[i,j] = \begin{bmatrix} \frac{im}{k} & \frac{im}{k}+1 & \cdots & \frac{jm}{k}-1 & \frac{jm}{k} \end{bmatrix}$$

We then define $A[k-1,k]$ as follows and note that the values of the row decrease when moving from left to right, rather than increasing as in $A[i,j]$.

$$A[k-1,k] = \begin{bmatrix} m & m-1 & \cdots & \frac{(k-1)m}{k}+1 & \frac{(k-1)m}{k} \end{bmatrix}.$$

We define a $k \times (m+1)$ matrix A by

$$A = \begin{bmatrix} A[0,1] & A[1,2] & \cdots & A[k-2,k-1] & A[k-1,k] \\ A[1,2] & A[2,3] & \cdots & A[k-1,k] & A[0,1] \\ \vdots & & & & \\ A[k-2,k-1] & A[k-1,k] & \cdots & A[k-4,k-3] & A[k-3,k-2] \\ A[k-1,k] & A[0,1] & \cdots & A[k-3,k-2] & A[k-2,k-1] \end{bmatrix}$$

Each row of A is a permutation of $\{0,1,\dots,m\}$. Each column of A contains entries which can be written as $\frac{jm}{k}+i$ for $0 \le j \le k-2$ and one entry of the form $m-i$, with $0 \le i \le \frac{m}{k}$. When adding any two entries of the column, for each $0 \le j < j' \le k-2$, we have the following:

$$\frac{jm}{k}+i+\frac{j'm}{k}+i \le \frac{(k-2)m}{k}+\frac{(k-1)m}{k} \le \frac{2(k-1)m}{k}.$$

In the case which involves adding the entry of the form $m-i$, we have the following:

$$\begin{aligned}\frac{jm}{k}+i+m-i \le \frac{(k-2)m}{k}+\frac{mk}{k} &\le \frac{(k-2)m+mk}{k} \\ &= \frac{2mk-2m}{k} = \frac{2(k-1)m}{k}\end{aligned}$$

Thus, in each column, the sum of any two entries is at most $\frac{2(k-1)m}{k}$.

For each column j of A, define a set

$$T_j = \bigcup_{i=1}^{k} \{v \in V_i : d \cdot g(v) > a_{i,j}\}$$

We claim that every T_j is a cover. Suppose it is not and $e=(u,v)$ is an edge not covered by T_j. Let $u \in V_i$ and $v \in V_{i'}$ with $1 \le i,i' \le k$ and $i \ne i'$. Then,

$$g(u)+g(v) = \frac{d \cdot g(u)+d \cdot g(v)}{d} \le \frac{a_{i,j}+a_{i',j}}{d} \le \frac{\frac{2(k-1)m}{k}}{d} = \frac{\frac{2(k-1)}{k}\frac{k}{2(k-1)}(d-1)}{d} \le \frac{d-1}{d} < 1$$

which is a contradiction, since g is a fractional cover.

Since each row of A is a permutation of $\{0,1,\dots,m\}$, we have

$$w(v) \cdot |\{0 \le j \le m : v \in T_j\}| \le w(v)g(v) \cdot d$$

for every vertex $v \in V$. Then $\sum_{j=0}^{m} w(T_j) \le \sum_{v \in V} w(v)g(v) \cdot d = d\tau^*(G)$. Since every T_j

is a cover, we have

$$\tau(G) \leq \frac{\sum_{j=0}^{m} w(T_j)}{m+1} \leq \frac{d\tau^*(G)}{\frac{k}{2(k-1)}(d-1)}$$

which, if $d \to \infty$ gives

$$\frac{\tau(G)}{\tau^*(G)} \leq \frac{2(k-1)}{k}.$$

□

Chapter 6

Using Approximate Fractional Covers

6.1 Approximate fractional covering algorithms

As seen earlier, computing the solution of a linear program can be quite costly as n, the number of variables, grows. Because the solution of a linear program is so expensive to compute, the time complexity of our vertex cover algorithm is dominated by this operation. Thus, the largest performance improvement that we can make to our algorithm kPartHypVC is by finding a less expensive way to compute the solution to the linear programming relaxation used in Line 3 of kPartHypVC.

One way to do this is to compute an approximate solution for the fractional vertex cover problem. Plotkin, Shmoys, and Tardos [62] give an algorithm that computes approximate solutions for a class of problems called packing and covering problems.

Definition 6.1.1. Let $P \subseteq \mathbb{R}^n$ be a convex set. A set is *convex* if for every pair of points in P, the line segment that joins the two points also lies entirely within P. A *covering problem* has the following form: given an $m \times n$ matrix A, an n-dimensional vector b with $b > 0$ such that $Ax \geq 0$ for all $x \in P$, find $x \in P$ such that $Ax \geq b$.

Note that we abuse notation here and in the following and write $x \geq y$ for vectors x and y to mean the components of x are greater than the corresponding components of y. We also use 0 to denote the zero vector with length n.

Instead of solving this problem exactly, which can be done in polynomial time using linear programming algorithms, the algorithm of Plotkin, Shmoys, and Tardos finds an $x \in P$ such that $Ax \geq (1 - \epsilon)b$, where $\epsilon > 0$ is the desired approximation factor. The advantage of this algorithm is that its time complexity is not dependent on the number of variables. Rather, the running time is polynomial in the number of constraints m, the approximation factor ϵ, and the width ρ of the set P relative to the problem $Ax \geq b$, which is defined as follows.

Definition 6.1.2. The width ρ of the set P relative to $Ax \geq b$ is defined by

$$\rho = \max_i \max_{x \in P} \frac{a_i x}{b_i}$$

where $a_i x$ is the ith component of Ax and b_i is the ith component of b.

The main idea of the algorithm in [62] is to maintain a point $x \in P$ that does not satisfy $Ax \geq b$ and repeatedly solve an optimization problem over P to find a new point $x' \in P$ that does not violate as many inequalities as x. Then x' is taken as the new point x and the above procedure is repeated until a point $x \in P$ satisfying $Ax \geq b$ is found. The optimization problem that needs to be solved to compute x' is the following.

Definition 6.1.3. Given an m-dimensional vector y, find $x' \in P$ such that

$$cx' = \max\{cx : x \in P\}, c = y^T A. \tag{6.1}$$

Since this subproblem is dependent on the specific covering problem that we wish to solve, the time complexity of the algorithm in [62] is expressed as the number of iterations of the subroutine that solves this subproblem.

6.2 Approximate fractional vertex covers

Theorem 6.2.1. *Let $H = (V, E)$ be a weighted k-partite k-uniform hypergraph with weight function $w : V \to \mathbb{Z}$ and $w_{\max} = \max_{v \in V}\{w(v)\}$. An ϵ-approximate fractional vertex cover for H can be computed in $O(|V| \log |V| \log(|V| w_{\max})(|E| + k \log^2 |E| + k\epsilon^{-2} \log(|E|\epsilon^{-2})))$ time.*

Proof. The approximate fractional cover algorithm of Plotkin, Shmoys, and Tardos requires $O(m + \rho \log^2 m + \rho\epsilon^{-2} \log(m\epsilon^{-1}))$ calls to a subroutine that solves the problem in Definition 6.1.3. The linear programming relaxation of the vertex cover program is given in (3.2). In this formulation, we have $n = |V|$ variables and $m = |E|$ constraints. To formulate the vertex cover problem in the form specified by Definition 6.1.1, we use the following formulation of (3.2):

$$\begin{aligned}
&\sum_{v\in V} w(v)g(v) = T \\
&\sum_{v\in e} g(v) \geq 1, \forall e \in E \\
&g(v) \geq 0
\end{aligned} \tag{6.2}$$

where T is the size of a minimum vertex cover. With this formulation of the problem we can define the convex set P as

$$P = \left\{ g \in [0,\infty)^n : \sum_{v\in V} w(v)g(v) = T \right\}.$$

The formulation of the subproblem in Definition 6.1.3 is then

$$\begin{aligned}
\text{maximize } &\sum_{v\in V} c(v)g(v) \\
\text{subject to } &\sum_{v\in V} w(v)g(v) = T \\
&g(v) \geq 0
\end{aligned} \tag{6.3}$$

where $c = y^T A$, and y is a given m-dimensional vector which is computed during the execution of the algorithm of Plotkin, Shmoys, and Tardos. Problem (6.3) is the fractional knapsack problem, where the items $v \in V$ have cost $c(v)$ and weight $w(v)$ and the knapsack has size T. Since $g(v)$ can take on fractional values, this problem is solvable in $O(n \log n)$ time by sorting vertices by the ratio $\frac{c(v)}{w(v)}$ for each vertex v and greedily choosing vertices in decreasing order of $\frac{c(v)}{w(v)}$ value.

Since T is not known, we need to compute it . As $T \leq \sum_{v\in V} w(v)$, we can use binary search on the set $\{1, 2, \ldots, \sum_{v\in V} w(v)\}$ to find the smallest value T for which problem (6.2) has a solution, which requires running the entire algorithm $O(\log \sum_{v\in V} w(v)) = O(\log(nw_{\max}))$ times, where $w_{\max} = \max_{v\in V}\{w(v)\}$. To determine the value of ρ, the width of the problem, note that each constraint in the linear program (3.2) is of the form $a_i g \leq b_i$, where a_i is the ith row of A and b_i is the ith component of b for $1 \leq i \leq m$. The entries of A only take on values 0 or 1 and the entries of b are all 1. Since $b_i = 1$ for every $1 \leq i \leq m$, we have $\rho = \max_i \sum_{j=1}^{n} a_{i,j}$, which is the size of the largest edge. Thus, for k-partite k-uniform hypergraphs, we have $\rho = k$. Then the total time complexity to find the approximate fractional cover is $O(|V| \log |V| \log(|V| w_{\max})(|E| + k \log^2 |E| +$

$k\epsilon^{-2}\log(|E|\epsilon^{-2})))$. □

Since the algorithm of [62] only computes an approximate fractional cover, if we use it in our algorithm KPARTHYPVC to compute a vertex cover for a k-partite k-uniform hypergraph, the value of the solution will not be as good as indicated in Theorem 5.1.1, but it will be very close as shown in the following theorem.

Definition 6.2.1. For $0 < \epsilon < 1$, an ϵ-approximate fractional covering of a hypergraph $H = (V, E)$ is a function $g_\epsilon : V \to \mathbb{R}$ such that $\sum_{v \in e} g_\epsilon(v) \geq 1 - \epsilon$ for every edge $e \in E(H)$. We define $|g_\epsilon|$ to be the value of the cover, given by $|g_\epsilon| = \sum_{v\in V} w(v)g_\epsilon(v)$. Let $\tau^*_\epsilon(H)$ denote the minimum of $|g_\epsilon|$ among all ϵ-approximate fractional covers of H.

Theorem 6.2.2. *For every k-partite k-uniform hypergraph H and $0 < \epsilon < 1$,*

$$\frac{\tau(H)}{\tau^*_\epsilon(H)} \geq \frac{k}{2}(1-\epsilon)^{-1}.$$

Proof. Let $H = (V, E)$ be a k-partite k-uniform hypergraph with partition $(V_1, \ldots, V_k)$ with weight function $w : V \to \mathbb{Z}$. Since $\tau^*_\epsilon(H)$ is the value of a minimal ϵ-approximate fractional cover, there exists an ϵ-approximate fractional cover $g_\epsilon : V \to \mathbb{R}$ such that $g_\epsilon(v) \in \mathbb{Q}$ for every vertex $v \in V$. Therefore, we can choose an integer d such that $(d-1)(1-\epsilon)$ and $d \cdot g_\epsilon(v)$ are integral for every vertex $v \in V$.

Let $m = \left\lfloor \frac{2}{k}(d-1)(1-\epsilon) \right\rfloor$. By Lemma 5.1.2, there exists a $k \times (m+1)$ matrix A such that its rows are permutations of $\{0, 1, \ldots, m\}$ and the sum of the entries in each of its columns is at most $\left\lceil \frac{km}{2} \right\rceil$.

Then for each column j of A, define a set

$$T_j = \bigcup_{i=1}^{k} \{v \in V_i : d \cdot g_\epsilon(v) > a_{i,j}\}.$$

We claim that every T_j is a cover. Suppose it is not and $e = (e_1, \ldots, e_k)$ is an edge not covered by T_j. Then $d \cdot g_\epsilon(v_i) \leq a_{i,j}$ for each $v_i \in e$.

Then,

$$\sum_{i=1}^{k} g_\epsilon(v_i) = \sum_{i=1}^{k} \frac{d \cdot g_\epsilon(v_i)}{d} \leq \frac{\sum_{i=1}^{k} a_{i,j}}{d} \leq \frac{\left\lceil \frac{km}{2} \right\rceil}{d} = \frac{\left\lceil \frac{k}{2} \left\lfloor \frac{2}{k}(d-1)(1-\epsilon) \right\rfloor \right\rceil}{d} \leq \frac{d-1}{d}(1-\epsilon) < 1-\epsilon$$

which is a contradiction, since g_ϵ is an ϵ-approximate fractional cover.

Since each row of A is a permutation of $\{0, 1 \ldots, m\}$, for every $v \in V$,

$$w(v) \cdot |\{0 \leq j \leq m : v \in T_j\}| \leq w(v)g_\epsilon(v)d.$$

Then $\sum_{j=0}^{m} w(T_j) \leq \sum_{v \in V} w(v)g_\epsilon(v)d = d\tau_\epsilon^*(H)$. Since every T_j is a cover, we have

$$\tau(H) \leq \frac{\sum_{j=0}^{m} w(T_j)}{m+1} \leq \frac{d\tau_\epsilon^*(H)}{\frac{2}{k}(d-1)(1-\epsilon)}$$

which gives

$$\frac{\tau(H)}{\tau_\epsilon^*(H)} \leq \frac{k}{2}(1-\epsilon)^{-1}$$

if $d \to \infty$. □

Theorem 6.2.3. *Let $H = (V, E)$ be a k-partite k-uniform hypergraph. Then a $\frac{k}{2}(1-\epsilon)^{-1}$ approximate vertex cover can be computed in $O(|V| \log |V| \log(|V|w_{\max})(|E| + k \log^2 |E| + k\epsilon^{-2} \log(|E|\epsilon^{-2})))$ time.*

Proof. By Theorem 6.2.2, using an ϵ-approximate fractional cover instead of an exact fractional cover gives a solution with approximation ratio $\frac{k}{2}(1 - \epsilon)^{-1}$. In algorithm KPARTHYPVC, instead of finding an exact solution, we use the approximate fractional covering algorithm of Plotkin, Shmoys, and Tardos to find an ϵ-approximate fractional cover. By Theorem 6.2.1, computing an ϵ-approximate fractional cover requires $O(|V| \log |V| \log(|V|w_{\max})(|E| + k \log^2 |E| + k\epsilon^{-2} \log(|E|\epsilon^{-2})))$ time. □

Chapter 7

Conclusions

In this thesis, we presented some results on approximating reductions of NFAs. Specifically, we showed some nontrivial families of regular languages for which a PRE-reduced NFA can be computed in polynomial time. In particular, if every state in an NFA belongs to a cycle, the PRE-reduced NFA can be computed in polynomial time. We show two examples of families of regular languages with NFAs that have this property: star languages and union-free languages.

We presented an approximation algorithm for the minimum vertex cover problem on k-partite k-uniform hypergraphs based on Lovász's theorem. The algorithm has an approximation ratio of $\frac{k}{2}$, which matches the integrality gap upper bound given by Lovász. The complexity of our algorithm is dominated by the need to compute a solution for a linear program. To improve the time complexity of the algorithm, we use approximate fractional covering algorithms to compute an approximate fractional cover instead of an exact vertex cover. We show that this improves the time complexity by removing the dependency on the number of variables of the LP instance at the expense of a small increase in the approximation ratio.

We would like to be able to identify structural properties of NFAs which determine whether a PRE-reduced NFA can be computed efficiently. For instance, it is known that every regular language can be written as the union of several union-free regular languages [55]. Since we know there is an NFA (specifically a 1-cycle-free path automaton) for each union-free regular language for which an equivalent PRE-reduced NFA can be computed in polynomial time, it may be possible to relate the structure of the partial order $\preceq$ of an automaton for the language to its union-free decomposition.

There are currently two best inapproximability results for the minimum vertex cover problem on k-partite k-uniform hypergraphs, depending on whether we assume that the Unique Games Conjecture (UGC) is true or false. If we assume that the UGC is true, then the problem is inapproximable to within a factor of $\frac{k}{2} - \epsilon$ for all $\epsilon > 0$ and $k \geq 3$. Thus, our algorithm achieves the best possible approximation. However, if we assume the UGC is false, then the problem has been shown to be inapproximable within a factor of $\frac{k}{2} - 1 + \frac{1}{2k} - \epsilon$ for all $\epsilon > 0$, but only for $k \geq 4$. This is the latest in a series of results that shows inapproximability closer and closer to $\frac{k}{2}$. It is believed that the problem is inapproximable to within a factor of $\frac{k}{2}$ but this question remains open.

Bibliography

[1] Ron Aharoni, Ron Holzman, and Michael Krivelevich. On a theorem of Lovasz on covers in r-partite hypergraphs. *Combinatorica*, 16:149–174, 1996.

[2] Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: Principles, Techniques and Tools*. Number 0 in Series in Computer Science. Addison-Wesley, 2006.

[3] J P Allouche and J O Shallit. *Automatic sequences: theory, applications, generalizations*. Cambridge University Press, 2003.

[4] Sanjeev Arora, Béla Bollobás, László Lovász, and Iannis Tourlakis. Proving Integrality Gaps without Knowing the Linear Program. *Theory of Computing*, 2:19–51, 2006.

[5] Henrik Björklund and Wim Martens. The tractability frontier for NFA minimization. *Journal of Computer and System Sciences*, 78(1):198–210, January 2012.

[6] Janusz A. Brzozowski and Rina Cohen. On Decompositions of Regular Events. *Journal of the ACM*, 16(1):132–144, January 1969.

[7] Janusz A. Brzozowski and Michael Yoeli. *Digital Networks*. Prentice-Hall, Inc., 1976.

[8] J Champarnaud and F Coulon. NFA reduction algorithms by means of regular inequalities. *Theoretical Computer Science*, 327(3):241–253, 2004.

[9] J Champarnaud and D Ziadi. Canonical derivatives , partial derivatives and finite automaton constructions. *Theoretical Computer Science*, 289(1):137–163, 2002.

[10] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Cliffson Stein. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 2nd edition, 2001.

[11] Karel Culik II and Tero Harju. Splicing semigroups of dominoes and DNA. *Discrete Applied Mathematics*, 31(3):261–277, 1991.

[12] Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. *Computers & Graphics*, 17(3):305–313, 1994.

[13] Irit Dinur, Venkatesan Guruswami, and Subhash Khot. Vertex Cover on k-Uniform Hypergraphs is Hard to Approximate within Factor (k-3-ϵ). Technical report, Electronic Colloquium on Computational Complexity, 2002.

[14] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A New Multilayered PCP and the Hardness of Hypergraph Vertex Cover. *SIAM Journal on Computing*, 34(5):1129–1146, January 2005.

[15] Irit Dinur and Shmuel Safra. The importance of being biased. In *Symposium on Theory of Computing*, pages 33–42, 2002.

[16] Tomas Feder, Rajeev Motwani, Liadan O'Callaghan, Rina Panigrahy, and Dilys Thomas. Online Distributed Predicate Evaluation. Technical report, Stanford University, 2003.

[17] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.

[18] Susumu Fujiwara, Gregor von Bochmann, Ferhat Khendek, Mokhtar Amalou, and Abderrazak Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.

[19] Yuan Gao, Kai Salomaa, and Sheng Yu. Transition Complexity of Incomplete DFAs. *Fundamenta Informaticae*, 110:143–158, 2011.

[20] Jaco Geldenhuys, Brink Van Der Merwe, and Lynette Van Zijl. Reducing Nondeterministic Finite Automata with SAT Solvers. In *Finite-State Methods and Natural Language Processing*, volume 6062, pages 81–92, 2010.

[21] V.M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.

[22] Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *Journal of the ACM*, 57(2):1–37, January 2010.

[23] Gregor Gramlich and Georg Schnitger. Minimizing NFA's and Regular Expressions. In *Symposium on Theoretical Aspects of Computer Science*, pages 399–411, 2005.

[24] Hermann Gruber and Markus Holzer. Computational Complexity of NFA Minimization for Finite and Unary Languages. In *Proceedings of the 1st International Conference on Language and Automata Theory and Applications*, pages 261–272, 2007.

[25] Venkatesan Guruswami and Rishi Saket. On the Inapproximability of Vertex Cover on k-Partite k-Uniform Hypergraphs. In *International Colloquium on Automata, Languages and Programming*, pages 360–371, 2010.

[26] Johan Hå stad. Some Optimal Inapproximability Results. *Journal of the ACM*, 48(4):798–859, 2001.

[27] Tom Head. Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.

[28] Jonas Holmerin. Improved Inapproximability Results for Vertex Cover on k-Uniform Hypergraphs. In *International Colloquium on Automata, Languages and Programming*, volume 2380, pages 1005–1016, 2002.

[29] Markus Holzer and Martin Kutrib. State Complexity of Basic Operations on Nondeterministic Finite Automata. In *International Conference on Implementation and Application of Automata*, volume 2608, pages 148–157, 2003.

[30] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196, 1971.

[31] Oscar H. Ibarra and Chul E. Kim. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *Journal of the ACM*, 22(4):463–468, 1975.

[32] Lucian Ilie, Gonzalo Navarro, and Sheng Yu. On NFA Reductions. In *Theory Is Forever, Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*, pages 112–124, 2004.

[33] Lucian Ilie, Baozhen Shen, and Sheng Yu. Fast Algorithms for Extended Regular Expression Matching and Searching. In *STACS*, volume 2607, pages 179–190, 2003.

[34] Lucian Ilie, Roberto Solis-Oba, and Sheng Yu. Reducing the Size of NFAs by Using Equivalences and Preorders. In *Combinatorial Pattern Matching*, pages 310–321, 2005.

[35] Lucian Ilie and Sheng Yu. Algorithms for Computing Small NFAs. In *International Symposium on Mathematical Foundations of Computer Science*, volume 2420, pages 328–340, 2002.

[36] Lucian Ilie and Sheng Yu. Follow automata. *Information and Computation*, 186(1):140–162, October 2003.

[37] Tao Jiang and B. Ravikumar. Minimal NFA Problems are Hard. *SIAM Journal on Computing*, 22(6):1117–1141, December 1993.

[38] T. Kameda and P. Weiner. On the State Minimization of Nondeterministic Finite Automata. *IEEE Transactions on Computers*, C-19(7):617–627, July 1970.

[39] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[40] Richard M Karp. Reducibility among combinatorial problems. In R E Miller and J W Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[41] Subhash Khot. On the power of unique 2-prover 1-round games. In *Symposium on Theory of Computing*, pages 767–775, 2002.

[42] Subhash Khot. Guest Column: Inapproximability Results via Long Code based PCPs. *ACM SIGACT News*, 36(2):25–42, 2005.

[43] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, May 2008.

[44] S.C. Kleene. Representation of Events in Nerve Nets and Finite Automata. Technical report, RAND Corporation (RM704), Santa Monica, CA, 1951.

[45] Donald Knuth, James H Morris, and Vaughan Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[46] Michael Krivelevich. Approximate Set Covering in Uniform Hypergraphs. *Journal of Algorithms*, 25(1):118–143, October 1997.

[47] Bruce G Linster. Evolutionary Stability in the Infinitely Repeated Prisoners' Dilemma Played by Two-state Moore Machines. *Southern Economic Journal*, 56(4):880–903, 1992.

[48] László Lovász. A kombinatorika minimax tételeiről (On the minimax theorems of combinatorics). *Mathematical Lapok*, 26:209–264, 1975.

[49] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[50] Andreas Malcher. Minimizing finite automata is computationally hard. *Theoretical Computer Science*, 327(3):375–390, November 2004.

[51] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9(1):39–47, 1960.

[52] B.F. Melnikov. A new algorithm of the state-minimization for the nondeterministic finite automata. *Journal of Applied Mathematics and Computing*, 6(2):277–290, 1999.

[53] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[54] J. Myhill. Finite automata and the representation of events. Technical report, Wright Patterson AFB (WADD TR-57-624), 1957.

[55] Benedek Nagy. A normal form for regular expressions. Technical report, CDMTCS-252, supplemental material for DLT 2004, 2004.

[56] Benedict Nagy. Union-free regular languages and 1-cycle-free-path-automata. *Publ. Math. Debrecen*, 68(1-2):183–197, 2006.

[57] G. L. Nemhauser and L. E. Trotter. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.

[58] A. Nerode. Linear automata transformation. *Proceedings of AMS*, 9:541–544, 1958.

[59] Oscar Nierstrasz. Regular types for active objects. In *Object-oriented programming systems, languages, and applications*, pages 1–15, 1993.

[60] Robert Paige and Robert Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[61] Christos H Papadimitriou and Mihalis Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.

[62] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast Approximation Algorithms for Fractional Packing and Covering Problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[63] Ariel Rubinstein. Finite Automata Play the Repeated Prisoner's Dilemma. *Journal of Economic Theory*, 39(1):83–96, 1986.

[64] Sushant Sachdeva and Rishi Saket. Nearly Optimal NP-Hardness of Vertex Cover on k-Uniform k-Partite Hypergraphs. In *Approximation, Randomization, and Combinatorial Optimzation Algorithms and Techniques*, pages 327–338, 2011.

[65] Kai Salomaa and Sheng Yu. Synchronization expressions with extended join operation. *Theoretical Computer Science*, 207:73–88, 1998.

[66] Luca Trevisan. Non-approximability results for optimization problems on bounded degree instances. In *Symposium on Theory of Computing*, pages 453–461, 2001.

[67] Stephen Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96(1):15–57, 1984.

Curriculum Vitae

Name:	Timothy Ng
Post-Secondary Education and Degrees:	University of Waterloo Waterloo, ON, Canada 2006 – 2011 BMath
	The University of Western Ontario London, ON, Canada 2011 – 2013 M.Sc.
Honours and Awards:	Queen Elizabeth II Graduate Scholarship in Science and Technology 2012 – 2013
Related Work Experience:	Teaching Assistant The University of Western Ontario 2011 – 2012