NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Set Partitioning via Inclusion-Exclusion

Dimitrios - Kyriakos - Koutsoulis

Supervisor: Stavros Kolliopoulos, Professor

ATHENS

SEPTEMBER 2016

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κατάτμηση συνόλων μέσω της Αρχής Εγκλεισμού-Αποκλεισμού

Δημήτριος - Κυριάκος - Κουτσούλης

Επιβλέπων: Κολλιόπουλος Σταύρος, Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2016

BSc THESIS

Set Partitioning via Inclusion-Exclusion

Dimitrios K. Koutsoulis
S.N.: 1115201000063

Supervisor: Stavros Kolliopoulos, Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κατάτμηση συνόλων μέσω της Αρχής Εγκλεισμού-Αποκλεισμού

Δημήτριος Κ. Κουτσούλης
Α.Μ.: 1115201000063

Επιβλέπων: Σταύρος Κολλιόπουλος, Καθηγητής

ABSTRACT

The present work is a study of the paper by Andreas Björklund, Thore Husfeldt and Mikko Koivisto, ”Set partitioning via inclusion-exclusion” [1]. The main aim of the writer was for the ideas presented to be as accessible as possible to undergraduate students.

We prove the principle of inclusion-exclusion and define the zeta transform while also giving an algorithm that computes it.

Given a n element set $\mathcal{N}$ and a family $\mathcal{F}$ of subsets of $\mathcal{N}$ we provide an exact algorithm that computes the number of k-partitions in time $\mathcal{O}(2^n \text{poly}(n))$. We also provide others that solve similar problems like k-covers, sum of weighted partitions and max-weighted partition.

We then provide examples of problems which are reducible to the ones solved above and for which the reduction does not dominate the time complexity.

The aforementioned algorithms are optimized for time with the space complexity being also exponential. Considering that the responsibility for this falls squarely on the calculations for the z-transform, we provide alternate ways of solving the previous problems where we substitute the z-transform by polynomial space tools with the drawback of them being more costly on time.

We conclude with an approximation algorithm for the Chromatic Number Problem in polynomial space.

SUBJECT AREA: Exact Algorithms

KEYWORDS: set partition, graph coloring, exact algorithm, zeta transform, inclusion-exclusion

ΠΕΡΙΛΗΨΗ

Το παρόν έργο αποτελεί μελέτη του paper των Andreas Björklund, Thore Husfeldt και Mikko Koivisto, ”Set partitioning via inclusion-exclusion” [1]. Κύριος στόχος κατά τη συγγραφή ήταν να καταστούν οι έννοιες που παρουσιάζονται όσο το δυνατόν περισσότερο εύληπτες από προπτυχιακούς φοιτητές.

Αποδεικνύουμε την αρχή εγκλεισμού-αποκλεισμού και ορίζουμε το z-μετασχηματισμό ενώ δίνουμε και έναν αλγόριθμο που τον υπολογίζει.

Δεδομένου ενός συνόλου $\mathcal{N}$, n στοιχείων και μιας οικογένειας $\mathcal{F}$ υποσυνόλων του $\mathcal{N}$ καθώς και ενός ακεραίου k, παρέχουμε έναν ακριβή αλγόριθμο που υπολογίζει το πλήθος των k-κατατμήσεων σε $\mathcal{O}(2^n \mathrm{poly}(n))$ χρόνο. Επίσης παρέχουμε και άλλους οι οποίοι λύνουν παρόμοια προβλήματα όπως η καταμέτρηση των k-καλυμμάτων, η άθροιση κατατμήσεων με βάρη και η εύρεση της πιο βαριάς κατάτμησης.

Στη συνέχεια παρέχουμε παραδείγματα προβλημάτων τα οποία ανάγονται σε αυτά που λύσαμε παραπάνω και για τα οποία οι αναγωγές δεν απαιτούν πολύ χρόνο.

Οι προαναφερθέντες αλγόριθμοι στοχεύουν στον ελάχιστο χρόνο, με τη χωρική πολυπλοκότητα να είναι εκθετική. Δεδομένου ότι την ευθύνη για αυτό φέρουν αποκλειστικά οι υπολογισμοί του z-μετασχηματισμού, δίνουμε εναλλακτικούς τρόπους επίλυσης των παραπάνω χωρίς τη χρήση του z-μετασχηματισμού σε πολυωνιμικό χώρο. Το μειονέκτημα αυτών είναι ότι χρειάζονται περισσότερο χρόνο.

Κλείνουμε με έναν προσεγγιστικό αλγόριθμο πολυωνυμικού χώρου ο οποίος λύνει το Πρόβλημα Χρωματικού Αριθμού Γραφήματος.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ακριβείς Αλγόριθμοι

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: κατάμηση συνόλου, χρωματισμός γραφήματος, ακριβής αλγόριθμος, z-μετασχηματισμός, αρχή εγκλεισμού-αποκλεισμού

ACKNOWLEDGEMENTS

I want to thank Professor Stavros Kolliopoulos for helping me with the preparation and the choice of the current work's subject.

CONTENTS

PREFACE **7**

1 INTRODUCTION **8**
1.1 The Principle of Inclusion Exclusion 8
1.2 Zeta Transform 8
1.3 Model of Computation 9

2 RESULTS **10**
2.1 Set Cover 10
2.2 Set Partition 10
2.3 Sum of Weighted Partitions 11
2.4 Finding a Heaviest Partition 13

3 APPLICATIONS **15**
3.1 Graph Coloring 15
3.1.1 Chromatic Number 15
3.1.2 Chromatic polynomial 15
3.1.3 List Coloring 16
3.1.4 Chromatic Sum 16
3.2 Other Graph Partitioning Problems 17
3.2.1 Counting Set Packings 17
3.2.2 Max k-Cut 17
3.3 Bayesian Partition Models 18

4 POLYNOMIAL SPACE **19**
4.1 Chromatic Number 19
4.2 Domatic Number 20

5 APPROXIMATION **21**
5.1 Chromatic Number Approximation 21
5.2 Further Approximations 21

6 CONCLUSION **23**

REFERENCES **24**

PREFACE

Given a n element set $\mathcal{N}$, a family $\mathcal{F}$ of subsets of $\mathcal{N}$ and an integer k we will count all possible partitions, sum the product $f_1(S_1) \cdots f_k(S_k)$ and maximize the sum $f_1(S_1) + \ldots + f_k(S_k)$ over all partitions $(S_1, \ldots, S_k)$ of $\mathcal{N}$.

To achieve the above we will first prove the principle of inclusion-exclusion and define the z-transform. Besides these tools we will also use dynamic programming and self reducibility.

While at first we optimize for time, we will also provide alternatives to the z-transform which is our main space bottleneck. In both cases we provide examples of interesting applications like graph coloring and other partitioning problems that are reducible to the above. Unless otherwise mentioned all proofs that follow come from [1].

An outline of the thesis follows in list format:

1. **INTRODUCTION**
   We begin laying our infrastructure in the form of tools which we will be using throughout the rest of this work. These include the zeta transform and the principle of inclusion-exclusion.

2. **RESULTS**
   Our most general results which consist of algorithms that compute interesting set properties. We begin with an algorithms that finds the number of set covers of a set, followed by another one for the number of set partitions. We then solve the more general Sum of Weighted Partitions problem and conclude with a way to find the heaviest among the possible partitions of a set.

3. **APPLICATIONS**
   We apply the algorithms from the previous chapter to solve problems defined on graphs. We begin with several problems related to graph coloring and then move to graph partitioning problems. We conclude this chapter with an application on bayesian model-based clustering.

4. **POLYNOMIAL SPACE**
   We prioritize space complexity and substitute the zeta transform, which is our most space-hungry tool, by others that are cheaper on space yet costlier on time. It is in this spirit that we solve the Chromatic and Domatic Number problems anew.

5. **APPROXIMATION**
   We prioritize speed over accuracy and provide a quick approximate solution to the Chromatic Number problem. We then generalize this in the form of guidelines aimed at tackling a class of similar problems.

6. **CONCLUSION**
   We briefly note what has be shown up to this point and provide references to related work done by other researchers after the publication of the paper under study. We close with some suggestions on where further research on this specific area should possibly focus.

1 INTRODUCTION

We will provide a description for each one of the tools that we will be using, namely the Inclusion-Exclusion principle from combinatorics, the zeta transform and a slightly different big O notation along with the model of computation that we will follow.

1.1 The Principle of Inclusion Exclusion

Let B be a finite set and $\mathcal{F} = \{A_1, A_2, \ldots, A_n\}$ a family of subsets of B. Let $w : 2^B \to \mathbb{R}$ be a weight function defined as $w(A) = \sum_{a \in A} w(a)$, where $A \subseteq B$ and $w : B \to \mathbb{R}$ is another weight function. The Principle of Inclusion Exclusion posits that

$$w(B \setminus \bigcup_{i=1}^{n} A_i) = w(B) + \sum_{\substack{X \subseteq \{1,\ldots,n\} \\ |X| \geq 1}} (-1)^{|X|} w(\bigcap_{i \in X} A_i)$$

Proof. We will sum the appearances of $w(a)$ for each $a \in B$ on each side and show that they are equal on both sides of the equation. If $a \notin A_i$ for all $A_i \in \mathcal{F}$ then it appears once in the left side's w's sum expansion. On the right side it also appears only once and that is in $w(B)$. On the other hand, if $a \in A_i$ for some $i_1, \ldots, i_n$ such that $\{i_1, \ldots, i_n\} = I$, then $w(a)$ doesn't appear on the left side while on the right side it's added for each even-sized i-combination of I and substracted for each odd-sized one and that is for all $i \in I$, formally

$$\sum_{X \subseteq I} (-1)^{|X|} w(a) \;=\; w(a) \sum_{i=0}^{|I|} \binom{|I|}{i} (-1)^i \;=\; 0$$

by the binomial theorem.

In the case where w is the indicator function of B we get the version of the principle of inclusion-exclusion that counts elements of subsets.

1.2 Zeta Transform

Let $\mathcal{N}$ be a n-element set, we define a function $z : \{2^{\mathcal{N}} \to \mathbb{R}\} \to \{2^{\mathcal{N}} \to \mathbb{R}\}$, in relation to all functions $f : \{2^{\mathcal{N}} \to \mathbb{R}\}$, such that

$$z(f) = \hat{f} \qquad \text{and} \qquad \hat{f}(Y) = \sum_{S \subseteq Y} f(S) \text{ for } Y \subseteq \mathcal{N}$$

We say that $\hat{f}$ is the zeta transform of f.

The naive way of computing the zeta transform takes $O(3^n)$ additions but we can reduce this to $O(n2^n)$ using Yate's method [51] [32, Section 4.3.4]. We term the resulting algorithm the fast zeta transform.

Proof. Let $A = \{a_1, \ldots, a_n\}$ be a set of n elements. We define $g_0, \ldots, g_n$ for $Y \subseteq A$ by $g_0(Y) = f(Y)$ and

$$g_i(Y) = \begin{cases} g_{i-1}(Y) + g_{i-1}(Y \setminus \{a_i\}), & \text{if } a_i \in Y \\ g_{i-1}(Y), & \text{otherwise} \end{cases} \quad (i = 1, \ldots, n).$$

We can show by induction that

$$g_i(Y) = \sum_{\substack{X = Y \setminus I \\ I \subseteq \{a_1, \ldots, a_i\}}} f(X), \qquad \text{with each } X \text{ appearing only once in the sum}$$

For $i = 0$ the above is obiously true. We suppose that it's true for $i = k - 1$ and try to prove the case where $i = k$. If $a_k \notin Y$ then

$$g_k(Y) = g_{k-1}(Y) = \sum_{\substack{X=Y\setminus I \\ I\subseteq\{a_1,\ldots,a_{k-1}\}}} f(X)$$

from the inductive hypothesis, which is equal to

$$\sum_{\substack{X=Y\setminus I \\ I\subseteq\{a_1,\ldots,a_k\}}} f(X)$$

since $a_k \notin Y$.
On the other hand, if $a_k \in Y$, then we can partition the set of all X into two sets, one given by removing subsets $I \subseteq \{a_1, \ldots, a_{k-1}\}$ from Y, accounted for by the first summand and the other given by removing $I \cup \{a_k\}$ from Y, accounted for by the second summand of the first branch. Thus the proof by induction is complete.

From the above we have that

$$g_n(Y) = \sum_{\substack{X=Y\setminus I \\ I\subseteq A}} f(X) = \sum_{X\subseteq Y} f(X) = \hat{f}(Y)$$

We can compute all $g_n(Y)$ from $g_{n-1}(Y)$ in $O(2^n)$ additions which gives us $O(n2^n)$ additions if we iterate from $i = 1$ to n. If $max_{Y\subseteq A} f(Y) = M$ for our selection of f then the integers that will occur in our computations are within the bounds $[-2^n M, 2^n M]$, since it's $O(2^n)$ additions of at most M sized integers. Therefore, if we use schoolbook addition, we need $O(\log(2^n M)) = O(\log(2^n) + \log(M)) = O(n + \log(M))$ time for each one which gives us $O(n^2 2^n + 2^n \log(M))$ total time abbreviated as $O^*(2^n \log(M))$ as we will soon see.

1.3 Model of Computation

We can will be using another variant of the big O notation symbolized O^* where for a function $f(n)$ which is the product of which one or more factor are polynomial functions and at least one is an exponential function, we can ignore all the polynomial factors e.g. $O(2^n \text{polynom}(n)) = O^*(2^n)$, the following is also implied $O(n \log(n)) = O(2^{\log(n)} \text{polylog}(n)) = O^*(2^{\log(n)}) = O^*(n)$.

We use the random access machine model of computation. We add 2 b-bit integers in $O(b)$ time and multiply them in $O(b \log(b) \log(\log(b))) = O^*(b)$ time [44].

2 RESULTS

In this chapter we will provide algorithms that compute interesting and somewhat fundamental properties of sets, alongside informal proofs of their correctness. We will optimize for time complexity and prove an upper bound for each one of them.

2.1 Set Cover

Let N be a n-element set and $\mathcal{F}$ a family of subsets of N. A k-cover is a tuple $(S_1, \ldots, S_k)$ where $S_i \in \mathcal{F}$ such that $S_1 \cup \cdots \cup S_k = \mathcal{N}$, where there is overlap and each S_i may appear more than once in a tuple. We want to compute $c_k(\mathcal{F})$ the number of k-covers. To that end we can use the principle of inclusion-exclusion where w is the indicator function of B, which B is the set of all k-tuples mentioned above. We define a family of subsets $\{A_1, \ldots, A_n\}$, $A_i \subseteq B$ such that A_i includes all k-tuples but those the i-th element of N belongs to. We can see that $c_k(\mathcal{F})$ is the number of k-tuples in B that appear in none of the A_i. Therefore from the principle of inclusion-exclusion we have that

$$c_k(\mathcal{F}) = |B| + \sum_{\substack{X \subseteq \{1,\ldots,n\} \\ |X| \geq 1}} (-1)^{|X|} \left| \bigcap_{i \in X} A_i \right|.$$

Let $a(X) = |\{S \in \mathcal{F} : S \cap X = \varnothing\}|$, intuitively the number of sets in $\mathcal{F}$ that don't overlap with X. Since our definition of a k-cover allows for overlap and repetition and since each A_i is a k-tuple we have that $|\cap_{i \in X} A_i|$ equals the number of k-permutations with repetition on the number of sets in $\mathcal{F}$ that avoid X, therefore $|\cap_{i \in X} A_i| = a(X)^k$. Note that $|B| = a(\varnothing)^k$.

From the above we have that

$$c_k(\mathcal{F}) = |B| + \sum_{\substack{X \subseteq \{1,\ldots,n\} \\ |X| \geq 1}} (-1)^{|X|} a(X)^k$$

which we will prove that we can compute in $\mathcal{O}^*(2^n)$ time. We need $\mathcal{O}(n2^n) = \mathcal{O}^*(2^n)$ time to compute the zeta transform $\hat{f}$ of the indicator function of $\mathcal{F}$ and then another $\mathcal{O}^*(2^n)$ to compute all $a(X)$ like this
(Think of $\mathcal{N}$ as the n-element set of the first n natural numbers for now to simplify the notation)

$$a(X) = \sum_{S \subseteq \mathcal{N} \setminus X} f(S) = \hat{f}(\mathcal{N} \setminus X).$$

Finally we raise them to the k-th power and sum all of them still in $\mathcal{O}^*(2^n)$ time to compute $c_k(\mathcal{F})$.

2.2 Set Partition

Let $\mathcal{N}$ be a n-element set and $\mathcal{F}$ a family of subsets over $\mathcal{N}$. We call a k-partition a tuple $(S_1, \ldots, S_k)$ over $\mathcal{F}$ such that $S_1 \cup \cdots \cup S_k = \mathcal{N}$ and $S_i \cap S_j = \varnothing$ $(i \neq j)$. Note that any $S \in \mathcal{F}$ may appear more than once in a single tuple. We will prove that we can count the number $p_k(\mathcal{F})$ of k-partitions in $\mathcal{O}^*(2^n)$ time.

We define $a_k(X)$ to be the number of k-tuples $(S_1, \ldots, S_k)$ where for every S_i, $S_i \cap X = \varnothing$ and $|S_1| + \ldots + |S_k| = n$. Now lets recall the unweighted principle of inclusion-exclusion where in our case B is substituted by the set of k-tuples over $\mathcal{F}$ that satisfy the second

condition from above. We define a family of subsets $\{A_1, \ldots, A_n\}$, $A_i \subseteq B$ such that A_i includes all k-tuples but those the i-th element of N belongs to. We can see that $p_k(\mathcal{F})$ is the number of k-tuples in B that appear in none of the A_i. Therefore from the principle of inclusion-exclusion we have that

$$p_k(\mathcal{F}) = |B| + \sum_{\substack{X \subseteq \{1,\ldots,n\} \\ |X| \geq 1}} (-1)^{|X|} \left| \bigcap_{i \in X} A_i \right|.$$

But since we have that $|\cap_{i \in X} A_i| = a_k(X)$ from the definition of $a_k(X)$, we can rewrite the above as such

$$p_k(\mathcal{F}) = |B| + \sum_{\substack{X \subseteq \{1,\ldots,n\} \\ |X| \geq 1}} (-1)^{|X|} a_k(X).$$

Note that $|B| = a_k(\varnothing)$. We calculate functions $f^{(l)} : 2^{\mathcal{F}} \to \{0, 1\}$ for all l, defined as

$$f^{(l)}(S) = \begin{cases} 1, & \text{if } S \in \mathcal{F} \text{ and } |S| = l \\ 0, & \text{else} \end{cases}$$

and then we build a table containing all zeta transforms $\hat{f}^{(l)}$ of the above using the Fast zeta transform, all that in $O^*(2^n)$ time. Now let us define

$$g(j, m, \mathcal{N}, X) = \sum_{l_i + \ldots + l_j = m} \prod_{c=1}^{i} \hat{f}^{(l_c)}(\mathcal{N} \setminus X),$$

informally $g(j, m, \mathcal{N}, X)$ equals the number of j-tuples $(S_1, \ldots, S_j)$ over $\mathcal{F}$ for which $S_i \cap X = \varnothing$ for all i and $|S_1| + \ldots |S_j| = m$.

It is clear that $a_k(X) = g(k, m, \mathcal{N}, X)$. We also have that $g(1, m, \mathcal{N}, X) = \hat{f}^{(m)}(\mathcal{N} \setminus X)$ at no cost and from that we can compute $g(k, m, \mathcal{N}, X)$ using the recursion

$$g(j, m, \mathcal{N}, X) = \sum_{l=0}^{m} g(j-1, m-l, \mathcal{N}, X) \hat{f}^{(l)}(\mathcal{N} \setminus X)$$

by dynamic programming in under $O^*(2^n)$ time. We now got everything we need and are ready to apply the principle of inclusion-exclusion.

2.3 Sum of Weighted Partitions

In the previous section we counted the number of partitions of a given set. In this one we will assign a weight to each partition and try to find the sum of the weights of all valid partitions.

We use the same definitions as before for set $\mathcal{N}$, tuples $S = (S_1, \ldots, S_k)$ on $Y \subseteq N$ and family $\mathcal{F}$. Additionally we have that k may be larger than n and any S_i element of any S may be empty. our intuition about weight is formalized by a function $f : S \to \mathbb{N}$ which for some given functions $f_1, \ldots, f_k : S_i \to \mathbb{N}$, $S_i \subseteq \mathcal{N}$, is defined as $f(S) = f_1(S_1) \cdots f_k(S_k)$ where $S = \{S_1, \ldots, S_k\}$. Therefore what we need to compute is

$$p_k(f) = \sum_{S} f(S).$$

(As a side note we consider $f_1, \ldots, f_k$ to be bounded as such $|f_i| \leq M$ for some integer M.) We will prove that we can do so in $O^*(2^n k \log(M))$ time.

We define $b_k(X)$ to be the sum of all $f(S)$ where S are the k-tuples whose elements are subsets of $\mathcal{N} \setminus X$ and for which such tuple the following (1) is true $|S_1| + \ldots + |S_k| = n$. Now recall the weighted principle of inclusion-exclusion from before but let the set of k-tuples on $\mathcal{N}$ satisfying (1) substitute for B. Likewise our subsets $A_i \subseteq B$ are going to be the sets of k-tuples that for each A_i avoid the i-th element of $\mathcal{N}$. Given that the $p_k(f)$ we are looking for equals the number of k-tuples including all elements of $\mathcal{N}$ out of B, we have that

$$p_k(f) = \sum_{S \in (\overline{A_1} \cap \cdots \cap \overline{A_n})} f(S) = w(\overline{A_1} \cap \cdots \cap \overline{A_n}).$$

It is also clear that $w(\cap_{i \in X} A_i) = b_k(X)$ and $w(B) = b_k(\varnothing)$. Therefore we can compute

$$p_k(f) = w(B) + \sum_{\substack{X \subseteq \mathcal{N} \\ |X| \geq 1}} (-1)^{|X|} b_k(X),$$

in time $O^*(2^n)$ given every $b_k(X)$. We will now show that we can compute all $b_k(X)$ in time $O^*(2^n k \log(M))$ dominating the above.

We begin with the computation of all

$$\hat{f}_c^{(l)}(Y) = \sum_{\substack{S \subseteq Y \\ |S| = l}} f_c(S),\ Y \subseteq \mathcal{N},$$

which equals the complexity of computing the zeta transform of all $f_1, \ldots, f_k$ on $\mathcal{N}$ which is $O^*(2^n k \log(M))$. We will now use the above to compute all $b_k(X)$ by dynamic programming in time $O^*(2^n k \log(M))$.

To simplify our proof we raise k to the nearest greater power of 2 namely 2^q such that $k \leq 2^q < 2k$ by introducing additional f_i's that evaluate to 1 at $\varnothing$ and to 0 elsewhere. We define for some given set $N \setminus X$

$$g(s,t,m) = \sum_{l_s + \ldots + l_t = m} \prod_{c=s}^{t} (N \setminus X),\ l_c \geq 0,$$

less formally it is the sum of the products $f_s(S_s) \cdots f_t(S_t)$ where $(S_s, \ldots, S_t)$ are all the $(s - t + 1)$-tuples on $N \setminus X$ constrained by the equation $|S_s| + \ldots + |S_t| = m$. It's clear that $b_k(X) = g(1,k,n)$ and we'll use the following recursion to compute it

$$g(s,t,m) = \sum_{m_0 + m_1 = m} g(s, \lfloor (s+t)/2 \rfloor, m_0) g(\lfloor (s+t)/2 \rfloor + 1, t, m_1)$$

and from our definition of $g(s,t,m)$ we have $g(c,c,m) = \hat{f}^{(m)}(N \setminus X)$ already available from our previous computations. We save time by not having to compute $g(s,t,m)$ for all $s,\ t$ and m but only the $2^{q+1} - 1$ required by $g(1,k,n)$ and that is including itself and the already computed base cases (think of them as the nodes of a binary tree). In the dynamic programming context where in the step of some $g(s,t,m)$'s computation, all other $g(s,t,m')$ required for it are known, the time needed for this operation equals the number of additions times the complexity of each multiplication. The former is $n(n+1)/2$, since m iterates through $1, \ldots, n$ and for each we have m additions. The latter depends on the size of the multiplied integers which are bounded by $n^m M^{t-s+1} m^{t-s}$. The proof by induction on $t - s$ is trivial. It is now clear that the time $T(j)$ required for the step of our dynamic programming algorithm where we compute $g(s,t,m)$ for any $s,\ t$ such that $t - s + 1 = j$ is bounded by

$$O^*((n(n+1)/2) \log(n^n))$$

$$= O^*(n^3 \log(n) + n^2 j \log(M) + n^2 j \log(n^{-1}))$$

$$= O^*(n^3 + n^2 j \log(M)).$$

As we noted before we can think of $g(1,k,n) = g(1,2^q,n)$ and the other g's transitively required for its computation as the nodes of a binary tree. Therefore the total time is

$$T(2^q) + 2^1 T(2^{q-1}) + 2^2 T(2^{q-2}) + \ldots + 2^q T(2^0).$$

The first part of $n^3 + n^2 j \log(M)$, that is, n^3 is constant among all summands above so we have $2(^{q+1}-1)n^3 = 2^{q+1}n^3 = 2^q n^3$. To calculate $n^2 j \log(M)$'s contribution to the total time we note that each summand adds exactly $2^q n^2 \log(M)$ to the sum and there are q such summands. Therefore the time needed to calculate $g(1,k,n)$ is

$$O^*(2^q n^3 + q 2^q n^2 \log(M))$$
$$= O^*(2^q n^3 + 2^q n^2 \log(M))$$
$$= O^*(kn^3 + kn^2 \log(M))$$

We need to compute one $g(1,k,n)$ for each $b_k(X)$, $x \subseteq \mathcal{N}$, so

$$O^*(2^n kn^3 + 2^n kn^2 \log(M))$$
$$= O^*(2^n k \log(M))$$

time for all $b_k(X)$ and thus for $p_k(f)$.

2.4 Finding a Heaviest Partition

We define the same objects as in the previous section. We now want to find a partition whose weight is the maximum out of all partitions' weights for our given set system. We start with an algorithm that computes the greatest weight in time $O^*(2^n kM)$ and use it to construct another one that finds the partition in question, in time $O^*(2^n k^2 M)$.

To make our work easier constraint our function $f_1, \ldots, f_n$ to the range $[0, M]$ down from $[-M, M]$. We define a new set of functions $\{f'_1, \ldots, f'_k\}$ such that $f'_c(S_i) = \beta^{f_c(S_i)}$ where $\beta = k^n + 1$ so that

$$p_k(f') = \sum_S \beta^{f_1(S_1)+\ldots+f_k(S_k)} = \sum_{r=0}^{kM} a_r \beta^r$$

where kM equals the maximum number of different possible weights, r iterates through these weights and a_r is the number of partitions $S = \{S_1, \ldots, S_k\}$ such that $r = f_1(S_1) + \ldots + f_k(S_k)$. It is important to note that the maximum number of partitions, in the case where $\mathcal{F}$ is the powerset of $\mathcal{N}$, is k^n. From that and the definition of a_r it's implied that $a_r \leq k^n < k^n + 1 = \beta$. We now posit that for the largest r that $a_r > 0$, denoted by r',

$$p_k(f') = \sum_{r=0}^{kM} a_r \beta^r < \beta^{r'+1}.$$

To prove it we in turn prove that $\sum_{r=0}^{z} a_r \beta^r < \beta^{z+1}$ by induction. The base case where $\sum_{r=0}^{0} a_0 \beta^0 < \beta^{0+1}$ is obviously true. Now suppose that $\sum_{r=0}^{z-1} a_r \beta^r < \beta^z$ is true. We have that

$$\sum_{r=0}^{z} a_r \beta^r = \sum_{r=0}^{z-1} a_r \beta^r + a_z \beta^z < \beta^z + a_z \beta^z \leq \beta^z + (\beta - 1)\beta^z = \beta^z + \beta^{z+1} - \beta^z = \beta^{z+1},$$

we've consequently proven that $p_k(f') = \sum_{r=0}^{kM} a_r \beta^r < \beta^{r'+1}$. It is now clear that $\log_\beta(p_k(f')) = r'$ the maximum weight. We use the previous section's algorithm to compute $p_k(f')$ and

since f_i' are bounded by β^M we can do it in $\mathcal{O}^*(2^n k \log(\beta^M)) = \mathcal{O}^*(2^n kMn \log(k)) = \mathcal{O}^*(2^n kM)$ time.

Finding a partition for the given max weight W is rather easy. We iterate through the elements of $\mathcal{N}$ and we assign each one to a part (color it in a sense) and then use the above algorithm to calculate the max weight of our set system without taking into account the weight of partitions that disagree with our assignment. We can achieve that by temporarily setting $f_c(S_c) = 0$ for every S_c that doesn't include our element with c being the number of our chosen part, effectively nullifying the weight of partitions that we are not interested in. After assigning and calculating the new possible maximum weight, if it is lower than the old maximum then we assign the element to a different part and repeat the above. When for an assignment the new and old max weights coincide, we lock the element to the part and move to the next element repeating the steps above. the partition we reach to after assigning all the elements to subsets of $\mathcal{N}$ is a maximum weight one. At each step of the iteration through the elements of $\mathcal{N}$ we use the algorithm of the first half of the section k times. Thus the total time is

$$\mathcal{O}^*(k2^n kM + k2^{n-1} kM + \ldots + k2^0 kM) = \mathcal{O}^*(k^2(2^{n+1} - 1)M) = \mathcal{O}^*(2^n k^2 M).$$

3 APPLICATIONS

In contrast to the work in the previous chapter, where we focused on solving problems defined on sets, in this one we focus on graphs. We use the previous results to construct algorithms that compute useful to know graph properties. We conclude with a section on Bayesian model-based clustering, which strays somewhat in content from the rest of the chapter.

3.1 Graph Coloring

We call k-coloring of a graph $G = (V, E)$, $|V| = n$ a mapping $C : V \to \{1, \ldots, k\}$ where for every $v_1, v_2 \in V$ and $C(v_1) = C(v_2)$ we have that $(v_1, v_2) \notin E$. in order to reuse our previous results we let $\mathcal{N} = V$ and $\mathcal{F}$ to be the family of all non empty independent sets of G.

3.1.1 Chromatic Number

We call chromatic number $\chi(G)$ of a graph G the smallest number k for which there exists a k-coloring of G. We now define the problem where for a given graph $G = (V, E)$ and number k we want to know if $\chi(G) \leq k$. This is easily reduced to the counting set covers problem, since if $c_k(\mathcal{F}) > 0$ then there exists some covering $\{S_1, \ldots, S_k\}$ so that we can assign colors $C(v) = \min\{c : v \in S_c\}$ which is a valid at most k-coloring. in addition we provide a alternate way to compute $a(X)$ for all $X \subseteq \mathcal{N}$ for our special case of $\mathcal{F}$.

Let $N(v) = \{v\} \cup \{u \in V : (v, u) \in E\}$ denote the set of v and all its neighbors. We have $a(X) = a(X \cup v) + a(X \cup N(v)) + 1,\ (v \notin X)$ to be true. To make it clearer how this is true we rewrite it as $a(X) - a(X \cup \{v\}) = a(X \cup N(v)) + 1,\ (v \notin X)$. The left hand side of the equation evaluates to the number of independent sets that avoid all of X and at the same time include v. The right hand side of the equation evaluates to the number of independent sets that avoid v and all its neighbors, plus the empty set. If we add v to every set above we reach independent sets that avoid X and obviously include v thus the set of them is a subset of the one we talked about on the left hand side. It is trivial to show it's true the other way around too, therefore since both are a subset of one another, they are equal and so are the left hand side and the right hand side.

Starting from the base $a(V) = 0$ we can use the recursion to compute and store all of $a(X)$ operating on n-bit integers in $O(2^n n)$ time and space. We can then execute the rest of the counting set covers algorithm to see if $c_k(\mathcal{F}) \geq 0$ and thus $\chi(G) \leq k$.

3.1.2 Chromatic polynomial

Given $G = (V, E)$, $|V| = n$ and $k \in \{0, \ldots, n\}$ we want to find the number $P(G; k)$ of k-colorings of G. We will prove that we can do so in $O^*(2^n)$ time and space.

Recall our counting partitions algorithm and the fact that for our definition, $p_r(\mathcal{F})$ (where $\mathcal{F}$ is the family of all independent sets of G) considers partitions consisting of the same subsets in alternate order to be different for the sake of counting them. therefore if we define $p'_r(\mathcal{F})$ to be the number of partitions $S = \{S_1, \ldots, S_k\}$ where the order of S_i doesn't matter, we have that for each such partition there exist $r!$ different reorderings of subsets and thus $p'_r(\mathcal{F})r! = p_r(\mathcal{F})$. We note that the ways we can color each valid r-partition equal the r-permutations, without repetition, of the k available colors and are thus $k!/(k - r)!$ in

number. In conclusion we have

$$P(G;k) = \sum_{r=1}^{k} \frac{k!}{(k-r)!} p'_r(\mathcal{F}) = \sum_{r=1}^{k} \frac{k!}{(k-r)!} \frac{p_r(\mathcal{F})}{r!} = \sum_{r=1}^{k} \binom{k}{r} p_r(\mathcal{F})$$

which requires $O^*(2^n)$ for the computation of p_r.

As a side note the function $P(G;\cdot) : \mathbb{N} \to \mathbb{N}$ is a degree n polynomial so we can compute it at multiple points from the formula above and then use those points to interpolate the polynomial in question and evaluate it at other interesting points such as at -1 which gives us the number of acyclic orientations of G [45].

3.1.3 List Coloring

We will generalize the k-coloring problem by introducing list coloring in which we are given $L(v) \subseteq \{1,\ldots,k\},\ v \in V$ which restricts the colors a vertex v can be assigned to by only allowing those $c \in L(v)$, down from $c \in \{1,\ldots,k\}$ in the k-coloring problem. What we ask is, if given $G = (V,E)$ and list $L : V \to 2^{\{1,\ldots,k\}}$, can G be L-colored?

We note that for those v that $L(v) > d(v)$ we can postpone their coloring till all of their neighbors are colored so that we can simply choose $c \in L(v)$ different from each neighbor's color. Consequently we can limit ourselves to the non-trivial case where $\forall v(v \in V \Rightarrow L(v) \leq d(v))$. The set of all available colors i.e. the union of all the sets in L's image is enumerated as $\{1,\ldots,k\}$. Assume $k \leq n(n-1)$. We define functions $f_1,\ldots,f_k : 2^V \to \{0,1\}$ where $f_c(S_c) = 1$ for S_c independent or empty, and $\forall v(v \in S_c \Rightarrow c \in L(v))$ meaning that it can be colored exclusively by c and if it's not the case then $f_c(S) = 0$. We can now use the above as the input to the Max Weighted Partition algorithm from section 2.4 to find the max weight of $f_1(S_1)+\ldots+f_k(S_k)$ over all $S = \{S_1,\ldots,S_k\}$ partitions of V. The propositions G is L-colorable and $W = k$ imply one another. To prove one way, if G is L-colorable then there exists an L-coloring C such that the sets $S_c = \{v : C(v) = c\}$ are independent or empty with $\forall v(v \in V_c \Rightarrow c \in L(v))$ and therefore form a valid partition which turns all $f_c(S_c)$ to 1 so $W = f_1(S_1) + \ldots + f_k(S_k) = k$. To prove the other way, if $W = k$ then there exists a partition $S = \{S_1,\ldots,S_k\}$ such that $f_c(S_c) = 1$ for all $c \in \{1,\ldots,k\}$. Therefore it's true for all these S_c that they are independent or empty and that $\forall v(v \in S_c \Rightarrow c \in L(v))$, so we can color each S_c monochromatically with c and reach a valid L-coloring. Our algorithm has the same time complexity $O^*(2^n)$ as the max weighted partitions one.

3.1.4 Chromatic Sum

Given a graph $G = (V,E)$ and k colors we want to find a coloring $C : V \to \{1,\ldots,n\}$ that minimizes $\sum_{v\in V} C(v)$. We note here that the coloring that gives the minimum $\max_{v\in V} C(v)$ does not necessarily coincide with the one that minimizes $\sum_{v\in V} C(v)$.

We can solve the Chromatic Sum in $O^*(2^n)$ time by reducing to Max Weighted Partition. We set $\mathcal{F}$ to be $2^{|V|}$ and define function $f_1,\ldots,f_n$ such that $f_c = -c|S|$, if S is independent or empty so that the sum $f_1(S_1)+\ldots+f_k(S_k) = -(1|S_1|+\cdots+k|S_k|)$ is maximized when its absolute value is minimized which is for $\min\sum_{v\in V} C(v)$. In order to discard non-colorable partitions we set $f_c(S_c) = -n^2$ for non-empty non-independent $S_c \in 2^{|V|}$ so that the absolute value of the weight of the partitions including them is larger than every one of those consisting exclusively of independent or empty $S_c \in 2^{|V|}$. The time needed for the above is the same as the one for Max Weigthed Partitions namely $O^*(2^n)$.

3.2 Other Graph Partitioning Problems

For the problems Domatic Number, Chromatic Number, Partitions into Forests where $\mathcal{F}$ consists of all S that $G[S]$ is a forest, Partition Into Perfect Matchings where $G[S]$ is a perfect matching and Bounded Component Spanning Forest where $G[S]$ is connected and $\sum_{v \in S} w(v) \leq B$, we have that we can iterate through k-partitions over $2^{|V|}$ and check for every $S_c \in 2^{|V|}$ of each such partition that $S_c \in \mathcal{F}$ in time polynomial in n, complexity which our $\mathcal{O}^*$ notation allows us to ignore. On the other hand we can't at least at the moment check if a graph is hamiltonian in poly(n) time and thus, in solving the Partition Into Hamiltonian Subgraph problem, we have to enumerate all hamiltonian subgraphs once in $\mathcal{O}^*(2^n)$ time [27] and store them so that we can later determine if $S_c \in \mathcal{F}$ in constant time to determine valid partitions.

3.2.1 Counting Set Packings

Given a set $\mathcal{N}$, a family of subsets of it $\mathcal{F} \subseteq 2^{\mathcal{N}}$ and an integer k, we define a k-packing to be a k-tuple $(S_1, \ldots, S_k)$ over $\mathcal{F}$ such that $\forall i \forall j (i \neq j \Rightarrow S_i \cap S_j = \varnothing)$. We want to find the number of all possible k-packings. We note that if $\mathcal{F} = \varnothing$ then we can use a valid k-packing to construct $k-1$ others by successively substituting S_i's with $\varnothing$.

We will use the Sum Weighted Partitions algorithm from section 2.3 with functions $f_1, \ldots, f_{k_1}$ where $f_1, \ldots, f_k$ are the indicator function of $\mathcal{F}$ and $f_{k+1}(X) = 1$ is a constant function (Please keep in mind that the distinction between k and $k+1$ in the notation is important). Recall $p_{k+1}(\mathcal{F})$ which used to be the number of $k+1$-partitions over $\mathcal{F}$, we now want it to be the number of k-packings over $\mathcal{F}$. For that to be so, $b_{k+1}(X) = g(1, k+1, n)$ should be equal to the number of all k-tuples $\{S_1, \ldots, S_k\}$ on $\mathcal{N} \setminus X$ such that there exists another k-tuple $\{S'_1, \ldots, S'_k\}$ on $\mathcal{N} \setminus X$, for which we have $S_1 \subseteq S'_1, \ldots, S_k \subseteq S'_k$ and $|S'_1| + \ldots + |S'_k| = n$. We note that $\hat{f}_i^{(l)}(\mathcal{N} \setminus X)$ for $i \in \{1, \ldots, k\}$ equals the number of $S_i \in \mathcal{F}$ for which $S_i \cap X = \varnothing$ and $|S| = l$. We also note that $\hat{f}_{k+1}^{(l)}(\mathcal{N} \setminus X)$ equals the number of subsets of $\mathcal{N} \setminus X$ of length l. Now for every k-tuple over $\mathcal{N} \setminus X$, $\{S_1, \ldots, S_k\}$ there exist others over $\mathcal{N} \setminus X$, $\{S'_1, \ldots, S'_k\}$ such that $|S'_1| + \ldots + |S'_k| = n$ and $S_1 \subseteq S'_1, \ldots, S_k \subseteq S'_k$. We have that the set $C = (S'_1 \setminus S_1) \cup \cdots \cup (S'_k \setminus S_k)$ is a subset of $\mathcal{N} \setminus X$. The number of all such k-tuples over $\mathcal{N} \setminus X$ of summed length m is

$$\hat{f}_{k+1}^{(n-m)}(\mathcal{N} \setminus X) \prod_{c=1}^{k} \hat{f}_c^{(l_c)}(\mathcal{N} \setminus X), \text{ where } l_1 + \ldots + l_k = m.$$

In the above $\hat{f}_{k+1}^{(n-m)}(\mathcal{N} \setminus X)$ is the number of all C complementing our k-tuples. If we sum over all the length permutations we get

$$b_{k+1}(X) = g(1, k+1, n) = \sum_{l_1 + \ldots + l_{k+1} = n} \prod_{c=1}^{k+1} (\mathcal{N} \setminus X).$$

3.2.2 Max k-Cut

Given a graph $G(V, E)$ and a weight function $w : E \to \mathbb{N}$ we want to find the k-partition of V that maximizes the sum of all $w(e)$ for e with endpoints in different parts of the partition.

As a side note we can reduce the case where $w : E \to \{1\}$ to the Max k-Colorable Subgraph on the complementary graph. To make it clearer, after solving the max coloring

on the complement, we can use the color classes to form a partition for which, after inverting the complementation of the edges, every element of every part is connected with all elements in foreign parts and the sum of w's is thus maximized.

We propose that we can solve Max k-Cut for $w : E \to [-M, M]$ in $O^*(2^n M)$ time. We define $f_1, \ldots, f_k$ as $f_i(S_i) = -\sum_{e \in E(S_i)} w(e)$ where $E(S_i)$ is the set of the edges with both endpoints in S_i. We now use the Max Weighted Partition algorithm with the above $f_1, \ldots, f_k$. We effectively find the partition that minimizes the absolute value of the $f_1(S_1) + \ldots + f_k(S_k)$ sum and in turn minimizes the intra-part sum of weights over all parts which finally maximizes the sum of weights of edges connecting different parts.

3.3 Bayesian Partition Models

Consider Bayesian model-based clustering [4, 18, 28, 38]. For $y_1, \ldots, y_m$ datapoints, $S = \{S_1, \ldots, S_k\}$ k-partitions of the set of the m indexes and for each such partition associate each part (named cluster) with a function of θ such that for a given j assigned to a cluster with its respective function θ we have that $P(y_j|\theta)$, which ranges from 0 to 1, tells us how suitable the assignment is for y_j. We are also given distributions for θ, $q(\theta)$ and weight functions to evaluate how the importance is distributed among the clusters of a clustering

$$p_c(S_c) = \frac{w_c(S_c)}{\sum_S \prod_c w_c(S_c)}.$$

We define

$$f_c(S_c) = p(S_c) \int \prod_{j \in S_c} P(y_j|\theta_c) q(\theta_c) d\theta_c$$

where we calculate how good the cluster is for some given θ_c, sum over all possible θ_c effectively integrating them out and then multiplying the result with the importance of the cluster. We want to find the optimal clustering which is the one that maximizes the sum $f_1(S_1) + \ldots + f_k(S_k)$ and to approximate such a partition we can use the Max Weighted Partition algorithm from section 2.4. We can't be sure that the result is exactly the optimal one cause Max Weighted Partition was defined on naturals while in this problem we deal with rationals.

4 POLYNOMIAL SPACE

We will prove that if membership in $\mathcal{F}$ can be decided in polynomial space and time then Counting Set Covers and Partitions can both be solved in polynomial space and $\mathcal{O}^*(3^n)$ time.

In the original algorithm of Counting Set Covers the bottleneck in space was storing all $a(X)$ we computed with the z-transform which required $\mathcal{O}^*(2^n)$ space. Since checking for membership in $\mathcal{F}$ costs polynomial time and space, we can compute each $a(X)$ on the fly when needed by checking for every $S_i \subseteq \mathcal{N} \setminus X$ if it is included in $\mathcal{F}$. We therefore have to do $2^{|\mathcal{N}|-|X|}$ membership tests for each $a(X)$ of which there are $\binom{n}{r}$ many, the total number of tests being

$$\sum_{X \subseteq \mathcal{N}} 2^{|\mathcal{N}|-|X|} = \sum_{r=0}^{n} \binom{n}{r} 2^{n-r} = 3^n$$

by the binomial theorem. The cost of each test is given to be polynomial so we ignore it, leading to $\mathcal{O}^*(3^n)$ time. We work similarly on the Counting Set Partitions, effectively substituting the use of the z-transform by membership tests on the fly.

We will now prove that if $f_1, \ldots, f_k$ can be computed in polynomial space and time then Sum and Max Weighted Partitions can be solved in $\mathcal{O}^*(3^n k \log M)$ and $\mathcal{O}^*(3^n k^2 M)$ time respectively and both in space polynomial in nk.

In Sum Weighted Partition we forego the computation and storing of $f_1, \ldots, f_k$ using the z-transform and instead compute them in the fly in time polynomial when needed. This, like in the previous proof, reduces the space requirement from exponential to polynomial and raises the base of the exponential time from 2 to 3. in the case of Max Weighted Partition we know from each original algorithm that it is reducible to Sum Weighted Partitions in under polynomial space and trivial time. We then follow the steps above but this time for M-bit integers and k times for each $X \subseteq \mathcal{N}$ leading to $\mathcal{O}^*(3^n k^2 M)$ time.

The above apply to all the partitioning problems of the previous sections besides Partitioning into Hamiltonian Subgraphs since we can't check if a graph is hamiltonian in polynomial time.

We will now turn to the case where $\mathcal{F}$ is small in comparison to $2^{\mathcal{N}}$ and enumerating it each time may prove faster than testing for membership for all $S_i \subseteq \mathcal{N} \setminus X$.

4.1 Chromatic Number

Given an algorithm [23] that counts all independent subgraphs of a n-vertex graph in $\mathcal{O}(1.2461^n)$ time, we will solve the Chromatic Number in $\mathcal{O}(2.2461^n)$ time and polynomial space.

In this problem $\mathcal{F}$ contains all independent subgraphs of a given $G = (V, E)$. To compute $a(X)$ we remove all vertices of X from V and apply the aforementioned algorithm on the resulting graph, in time $\mathcal{O}(1.2461^{|V|-|X|})$. The total time therefore is

$$\sum_{X \subseteq V} \mathcal{O}(1.2461^{|V|-|X|}) = \sum_{r=0}^{|V|} \binom{|V|}{r} \mathcal{O}(1.2461^{|V|-r}) = \mathcal{O}(2.2461^{|V|})$$

by the binomial theorem.

4.2 Domatic Number

Given a graph $G = (V, E)$, we call dominating set a vertex subset $D \subseteq V$ such that for every vertex in V there exists some vertex in D such that the two of them are one and the same or there is an edge connecting them. We call domatic number $\delta(G)$ of the graph G the largest integer k such that there exists a k-partition of V into dominating subsets. In the Domatic Number problem we are given a graph $G = (V, E)$ and an integer k and we ask if we can partition V into k dominating subsets or equivalently if $\delta(G) \geq k$. What we've proven at the start of this section applies to this problem and so we can solve it in polynomial space and $O^*(3^n)$ time. In this subsection we will provide another way of solving it, this time in $O(2.8718^n)$ time, still in polynomial space and provide a partial proof of it.

We set $\mathcal{F}$ to be the set of minimal dominating sets of G. We call minimal dominating a subset of V from which if we remove any vertex it stops being dominating. Our problem is reduced in constant time and space to Counting Set Packings since given that there exists a k-packing of minimal dominating subsets, we can select vertices missing for it to be a k-partition and add each one to a pack at random, constructing a k-partition of no longer minimal dominating sets. Now we reduce to Sum Weighted Partitions in constant time and space as we did in Section 3.2 but instead of using the z-transform to compute a table for $\hat{f}_c^{(l)}(Y),\ Y \subseteq V$ which would require exponential space, we use the algorithm from [21, Theorem 5.1] for all Y which requires $O(2.8718^n)$ total time and polynomial space. The authors of [22] note that it is not known how representative of the actual run-time the above bound is.

5 APPROXIMATION

We shortly shift the focus to the case where accurate results are not of paramount importance and trading accuracy for speed is desirable. An algorithm for approximating the Chromatic Number of a graph is provided, followed by general guidelines on tackling similar problems.

5.1 Chromatic Number Approximation

We will provide an algorithm that approximates the solution to the Chromatic Number in less time than the one required by our exact algorithm to solve it. Given $\varepsilon \in \mathbb{R},\ \varepsilon > 0$, the chromatic number χ of a graph $G = (V, E),\ |v| = n$, can be approximated by $\overline{\chi}$ where $\chi \leq \overline{\chi} < \chi + \varepsilon\chi + 1$ in time $O(1.2209^n + 2.2461^{e^{-\varepsilon}n})$ and polynomial space.

Begin by finding the largest independent set in our graph and removing its vertices from our graph in time $O(1.2209^n)$ and polynomial space [12]. We repeat this till there are $e^{-\varepsilon}n$ or less vertices in our graph. Let s be the number of the repetitions we did. We now use the exact polynomial space algorithm from section 4.1 on the current graph in time $O(2.2461^{e^{-\varepsilon}n})$ and find its chromatic number χ_0. We set $\overline{\chi} = \chi_0 + s$ and will now bound the maximum error of our approximation. We note that after each repetition in the first part, the graph's chromatic number will have been lowered by 1 if for all minimal colorings one of the colors was used solely by the largest independent set and is therefore no longer needed. In all other cases the chromatic number remains unchanged. Since we have s repetitions and $\overline{\chi} = \chi_0 + s$ it's implied that $\overline{\chi} \geq \chi$, our lower bound. We also have that $\chi_0 \leq \chi$ because a subgraph has the same or lower chromatic number than its supergraph. It is also true that for a given chromatic number χ the size of the largest independent set is minimized in the case where there are χ independent sets of equal size $1/\chi$ that form a partition of the graph. After removing such an independent set in the first step, the resulting graph has $(1 - 1/\chi)n$ vertices and after s repetitions $(1 - 1/\chi)^s n$ vertices. To generalize after s repetitions we have at most $(1 - 1/\chi)^s n$ vertices and it follows that

$$\left(1 - \frac{1}{\chi}\right)^{s-1} n > e^{-\varepsilon}n \Rightarrow \left(1 - \frac{1}{\chi}\right)^{s-1} > e^{-\varepsilon}.$$

The following is also true for all x

$$1 + x \leq e^x \Rightarrow \left(1 - \frac{1}{\chi}\right) \leq e^{-\frac{1}{\chi}} \Rightarrow \left(1 - \frac{1}{\chi}\right)^{s-1} \leq e^{-\frac{s-1}{\chi}},\ \text{for } s \geq 1.$$

From the conjunction of the above we have that

$$e^{-\varepsilon} < e^{-\frac{s-1}{\chi}} \Rightarrow -\varepsilon < -\frac{s-1}{\chi} \Rightarrow \varepsilon > \frac{s-1}{\chi} \Rightarrow \varepsilon\chi > s - 1 \Rightarrow s < \varepsilon\chi + 1$$

which leads us to our upper bound

$$\overline{\chi} < \chi + \varepsilon\chi + 1.$$

5.2 Further Approximations

We can generalize the above to approximate solutions for Minimal Covering problems given that the following are true for $\mathcal{F}$:

1. there is a fast algorithm for finding the largest $S_i \in \mathcal{F}$ and by fast we mean that it requires less time than finding the exact final solution.
2. $\mathcal{F}$ is hereditary which means that $S_i \subset T \in \mathcal{F}$ implies $S_i \in \mathcal{F}$.

For example we can't use the above if $\mathcal{F}$ is the set of induced trees of a graph since subgraphs induced from a tree might not be trees themselves.

6 CONCLUSION

We have studied the results of [1] that provide solutions to the counting covers and partitions problems on sets. This knowledge was subsequently applied to create algorithms for a class of graph coloring and partitioning problems. A brief note on utilizing these in order to create model based clustering algorithms is also included. Alternatives where a low space complexity is of import are given for some of the above, as well as some approximating algorithms based on the same principles as the exact ones we studied.

Regarding the case where our assumption that arithmetic operations have unit cost is invalidated, like when we are dealing with large integers, Michalak *et al.* [53] have provided an $O(3^n)$ algorithm for finding the heaviest partition of a set which is in practice faster than the one we provided. Kemper and Beichl have provided methods to approximate the chromatic polynomial by using probabilistic schemes [54]. An alternate way of coloring a graph is given by Golovnev *et al.* [55] in which the inclusion-exclusion principle is substituted by the Fast Fourier Transform and happens to be faster under certain circumstances. In the case where we are constrained to polynomial space, Gaspers and Lee provide provide us with an algorithm [56] that solves the graph coloring problem in $O(2.2355^n)$ time and as a byproduct they also give us one that runs in $O(1.2330^n)$ time and exponential space. Nederlof *et al.* [57] used the inclusion-exclusion principle to construct a faster branching algorithm that solves the domatic number in polynomial space.

Finding a better compromise between space and time complexity via a third tool that could work as an alternative to the zeta transform and checking for membership schemes could be a focal point of future research. More refined approximation algorithms could be designed that fill the gap between polynomial space approximations and our exponential exact ones. Lastly, a natural question would be on which other interesting set systems could our results be applied and for each one of them are there any special optimizations that could be found.

REFERENCES

[1] A. Björklund, T. Husfeldt and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546-563, 2009.

[2] O. Angelsmark and J. Thapper. Partitioning based algorithms for some colouring problems. *In Recent Advances in Constrains,* Springer LNAI volume 3978, pages 44-58, 2005.

[3] M. H. G. Anthony. Computing chromatic polynomials. *Ars Combinatorica,* 29(C):216-220,1990.

[4] J. D. Banfield and A. E. Raftery. Model-based Gaussian and non Gaussian clustering. *Biometrics,* 49:803-821, 1993.

[5] E. T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Information Processing Letters,* 47(4):203-207, 1993.

[6] E. T. Bax. Algorithms to count paths and cycles. *Information Processing Letters,* 52(5):249-252, 1994.

[7] E. T. Bax and J. Franklin. A finite-difference sieve to count paths and cycles by length. *Information Processing Letters,* 60(4):171-176, 1996.

[8] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *J. Algorithms,* 54(2):168-204, 2005.

[9] N. Biggs. *Algebraic graph theory.* Cambridge University Press, 2nd edition, 1993.

[10] A. Björklund and T. Husfeldt. Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica,* to appear. Prelim. version in *Proc. 33rd ICALP,* Springer LNCS volume 4051, pages 548-559, 2006.

[11] H. L. Bodlænder and D. Kratsch. An exact algorithm for graph coloring with polynomial memory. Technical Report UU-CS-2006-015, Utrecht University, 2006.

[12] J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32:547-556,2004.

[13] J. M. Byskov and D. Eppstein. An algorithm for enumerating maximal bipartite subgraphs. Manuscript, 2004.

[14] D. M. Chickering D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI 1997)*, pages 80-89, 1997.

[15] N. Christofides. An algorithm for the chromatic number of a graph. *Computer J.*, 14:38-39, 1971.

[16] D. Eppstein. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms and Applications*, 7(2):131-140, 2003.

[17] T. Feder and R. Motwani. Worst-case time bounds for coloring and satisfiability problems. *J. Algorithms*, 45(2):192-201, 2002.

[18] M. A. T. Figueiredo and A. K. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381-396, 2002.

[19] F. V. Fomin, S. Gaspers, and A. V. Pyatkin. Finding a minimum feedback vertex set in time $O(1.7548^n)$. In *Proc. 2nd IWPEC*, Springer LNCS volume 4169, pages 184-191, 2006.

[20] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: A simple $O(2^{0.288n})$ Independet Set algorithm. In *Proc. 17th SODA*, pages 18-25, 2005.

[21] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. In *Proc. 16th ISAAC*, Springer LNCS volume 4288, pages 573-582, 2006.

[22] M. Fürer. Faster integer multiplication. In *Proc. 39th STOC*, ACM Press, pages 57-66, 2007.

[23] M. Fürer and S. P. Kasiviswanathan. Algorithms for counting 2-SAT solutions and colorings with applications. In *Proc. 3rd Intl. Conf. on Algorithmic Aspects in Information and Management (AAIM)*, Springer LNCS volume 4508, pages 47-57, 2007.

[24] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, San Fransisco, 1979.

[25] C. Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Computational Complexity*, 9:52-73, 2000.

[26] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19-23, 1993.

[27] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *SIAM J. Appl. Math.*, 10:196-210, 1962.

[28] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264-323, 1999.

[29] R. Kennes. Computational aspects of the Moebius transform of a graph. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:201-223, 1991.

[30] R. M. Karp. Dynamic programming meets the principle of inclusion-exlcusion. *Oper. Res. Lett.*, 1:49-51, 1982.

[31] S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the Traveling Salesman Problem. In *ACM '77: Proceedings of the 1977 annual conference*, ACM Press, pages 294-300, 1977.

[32] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. 3rd ed., Addison-Wesley, 1997.

[33] M. Koivisto. *Sum-Product Algorithms for the Analysis of genetic Risks*. Ph.D. Thesis, University of Helsinki, January 2004.

[34] M. Koivisto. Optimal 2-constraint satisfaction via sum-product algorithms. *Information Processing Letters*, 98:24-28, 2006.

[35] M. Koivisto and K. Sood. Computational aspects of Bayesian partition models. In *International Conference on Machine Learning (ICML 2005)*, pages 433-440, 2005.

[36] E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66-67, 1976.

[37] B. A. Madsen. An algorithm for exact satisfiability analysed with the number of clauses as parameter. *Information Processing Letters*, 97(1):28-30, 2006.

[38] F. A. Quintana and P. L. Iglesias. Bayesian clustering and product partition models. *Journal of the Royal Statistical Society B*, 65(2):557-574, 2003.

[39] I. Razgon. Exact computation of maximum induced forest. In *Proc. 10th SWAT*, Springer LNCS volume 4059, pages 160-171, 2006.

[40] T. Riege and J. Rothe. An exact 2.9416^n algorithm for the three domatic number problem. In *Proc. 30th MFCS*, Springer LNCS volume 3618, pages 733-744, 2005.

[41] T. Riege, J. Rothe, H. Spakowski, and M. Yamamoto. An improved exact algorithm for the domatic number problem. *Information Processing Letters*, 101(3):101-106, 2007.

[42] G. C. Rota. On the foundations of combinatorial theory. I. Theory of Möbius functions. *Z. Wahrscheinlichkeitstheorie und Verw. Gebiete*, 2:340-368, 1964.

[43] H. J. Ryser. *Combinatorial Mathematics.* Number 14 in Carus Math. Monographs. Math. Assoc. America, 1963.

[44] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281-292, 1971.

[45] R. P. Stanley. Acyclic orientations of graphs. *Disc. Math.* 5:171-178, 1973.

[46] S. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398-427, 2001.

[47] H. S. Wilf. *Algorithms and complexity.* Prentice-Hall, 1986.

[48] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348:257-265, 2005.

[49] G. J. Woeginger. Exact algorithms for NP-hard problems: a survey. In *Combinatorial optimization: Eureka, you shrink!*, Springer LNCS volume 2570, pages 185-207, 2003.

[50] G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems. In *Proc. 1st IWPEC*, Springer LNCS volume 3162, pages 281-290, 2004.

[51] F. Yates. The design and analysis of factorial experiments. Technical Communication no. 35 of the Commonwealth Bureau of Soils, 1937.

[52] D. Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3:103-128, 2007. Prelim. version in *Proc. 38th STOC*, ACM Press, pages 681-690, 2006.

[53] T. Michalak, T. Rahwan, E. Elkind, M. Wooldridge and N. R. Jennings. A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, 230:14-50, 2016.

[54] Y. Kemper and I. Beichl. Approximating the Chromatic Polynomial. arXiv:1608.04883 [cs.DM]

[55] A. Golovnev, A. S. Kulikov and I. Mihajlin. Families with Infants: Speeding Up Algorithms for NP-Hard Problems Using FFT. *ACM Transactions on Algorithms (TALG)*, Volume 12 Issue 3, Article No. 35, 2016.

[56] S. Gaspers and E. Lee. Faster Graph Coloring in Polynomial Space. arXiv:1607.06201 [cs.DS]

[57] J. Nederlof, J. M. M. van Rooij and T. C. van Dijk. Inclusion/Exclusion Meets Measure and Conquer. *T.C. Algorithmica*, 69:685, 2014. doi:10.1007/s00453-013-9759-2