SET PARTITIONING VIA INCLUSION–EXCLUSION*

ANDREAS BJÖRKLUND†, THORE HUSFELDT†, AND MIKKO KOIVISTO‡

Key words. Set partition, graph colouring, exact algorithm, zeta transform, inclusion–exclusion

AMS subject classifications. 05C15, 68W01, 68R10, 90C27

Abstract. Given a set N with n elements and a family $\mathcal{F}$ of subsets, we show how to partition N into k such subsets in $2^n n^{O(1)}$ time. We also consider variations of this problem where the subsets may overlap or are weighted, and we solve the decision, counting, summation, and optimisation versions of these problems. Our algorithms are based on the principle of inclusion–exclusion and the zeta transform.

In effect we get exact algorithms in $2^n n^{O(1)}$ time for several well-studied partition problems including Domatic Number, Chromatic Number, Maximum k-Cut, Bin Packing, List Colouring, and the Chromatic Polynomial. We also have applications to Bayesian learning with decision graphs and to model-based data clustering.

If only polynomial space is available, our algorithms run in time $3^n n^{O(1)}$ if membership in $\mathcal{F}$ can be decided in polynomial time. We solve Chromatic Number in $O(2.2461^n)$ time and Domatic Number in $O(2.8718^n)$ time.

Finally, we present a family of polynomial space approximation algorithms that find a number between $\chi(G)$ and $\lceil(1+\epsilon)\chi(G)\rceil$ in time $O(1.2209^n + 2.2461^{e^{-\epsilon}n})$.

1. Introduction. Graph colouring, domatic partitioning, weighted k-cut, and a host of other problems can be viewed as special cases of partitioning an n-set N into subsets $S_1, \ldots, S_k$ from a given family $\mathcal{F}$. More generally, given functions $f_1, \ldots, f_k$ we want to sum the product $f_1(S_1) \cdots f_k(S_k)$ or optimise the sum $f_1(S_1) + \cdots + f_k(S_k)$ over all partitions $(S_1, \ldots, S_k)$ of N.

Ignoring for a moment factors that are polynomial in n, k, and the range of f_c, we solve this problem in time 2^n using two simple ideas. The first idea is to express the problem as an inclusion–exclusion formula over the subsets of N. In its simplest form, it says that N can be covered with k sets from $\mathcal{F}$ if and only if

$$\sum_{X \subseteq N} (-1)^{|X|} a(X)^k > 0\,, \tag{1.1}$$

where $a(X)$ denotes the number of sets from $\mathcal{F}$ not intersecting X. The second idea is to evaluate the summands quickly by first building a table containing all the $a(X)$ using an algorithm known as the fast zeta transform. For our more general applications, we also need some other standard techniques such as dynamic programming, embedding into large integers, and self-reducibility, but even so our arguments remain short, elementary, and self-contained.

1.1. Applications. Perhaps the simplest application of our result is Bin Packing, where we are given a weight $w(x)$ for each $x \in N$ and $\mathcal{F}$ consists of the subsets $S \subseteq N$ satisfying $\sum_{x \in S} w(x) \leq B$.

But typically we consider more constrained partitions. Most notably our results apply to some well-known NP-hard problems on graphs or hypergraphs that ask for a vertex partition into subgraphs that satisfy a given property, such as independence or dominance. Table 1.1 shows some examples.

[1]Previous versions of this paper appeared as [Inclusion–exclusion algorithms for counting set partitions, Proc. 47th FOCS, 2006] by the first two authors, and independently as [An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion, Proc. 47th

Name [23]	Property of $S \in \mathcal{F}$
Domatic Number	S is a dominating set in G
Chromatic Number	S is a independent set in G
Partition into Hamiltonian Subgraphs	$G[S]$ is Hamiltonian
Partition into Forests	$G[S]$ is a forest
Partition into Perfect Matchings	$G[S]$ is a perfect matching
Bounded Component Spanning Forest	$G[S]$ connected, $\sum_{v \in S} w(v) \leq B$

Table 1.1
Some exact partition problems where N are the vertices of a graph G.

The most obvious application is of course Minimum Set Cover and its many variants. However, this may be a misleading example, because in those problems, the set $\mathcal{F}$ is given explicitly as part of the input and is often small compared to n; for example the clauses of a monotone satisfiability problem or the edges of a sparse hypergraph. Our algorithms apply to these problems as well, and become interesting when $\mathcal{F}$ is large compared to n.

The most general formulation of our result replaces the family $\mathcal{F}$ with functions $f_1, \ldots, f_k$, which allows us to consider 'weighted' partitions with the objective function $f_1(S_1) + \cdots + f_k(S_k)$. The applications of this framework include Maximum k-Cut and the graph colouring problems List Colouring and Chromatic Sum.

Typically, we actually solve the related counting version of the problem. For example, we count the number of k-colourings of a graph, which is known as the problem of computing the Chromatic Polynomial. In the weighted case, we compute the sum of the product $f_1(S_1) \cdots f_k(S_k)$ over all partitions; we give applications to Bayesian data clustering and decision graph learning.

1.2. Further Results. We note that (1.1) immediately yields an $|\mathcal{F}|2^n n^{O(1)}$ time, *polynomial space* algorithm. We take a closer look at polynomial space algorithms for Chromatic and Domatic Number. Using the fastest currently known algorithm in the literature for counting independent sets [22] to compute $a(X)$, the total running time to evaluate the sum in (1.1) becomes $O(2.2461^n)$. For Domatic Number, we need a more complicated argument that can be seen as an extension of our main result, together with a recent algorithm to count the number of minimal dominating sets [20] and arrive at total time $O(2.8718^n)$. Both of these algorithms are the fastest polynomial space algorithms known for these problems, in fact they are faster than the best *exponential* space algorithms known prior to this paper.

Finally, we derive a family of exponential-time approximation algorithms based on first removing large independent sets and then applying our ideas on the remaining graph. For instance, we can approximate $\chi(G)$ within a factor 2 in time $O(1.3467^n)$ and polynomial space. The approximability of the chromatic number is very well studied; the best known polynomial time algorithm guarantees only an approximation ratio of $O(n \log^{-3} n \log\log^2 n)$ [25], and $\chi(G)$ is NP-hard to approximate within $n^{1-o(1)}$ [51].

FOCS, 2006] by the third author.

[2]Department of Computer Science, Lund University, Sweden

[3]Helsinki Institute for Information Technology, Basic Research Unit, Helsinki University of Technology and Department of Computer Science, University of Helsinki, Finland

Time $O(c^n)$	Problem	Reference
$c =$2.4423	Find χ	Lawler [35]
2.4151	Find χ	Eppstein [15]
2.4023	Find χ	Byskov [11]
2.3236	Find χ	Björklund and Husfeldt [9]
2.2590	Decide $\chi \leq 5$	Beigel and Eppstein [7]
2.1592	Decide $\chi \leq 5$	Byskov [11]
2.1020	Decide $\chi \leq 5$	Byskov and Eppstein [12]
2.1809	Decide $\chi \leq 6$	*ibid.*
2.9416	Decide $\delta \geq 3$	Riege and Rothe [39]
2.8718	Find δ	Fomin *et al.* [20]
2.6949	Decide $\delta \geq 3$	Riege *et al.* [40]

TABLE 1.2
Previous algorithms for Chromatic Number χ and Domatic Number δ.

Our inclusion–exclusion formulas themselves provide characterizations of well-studied graph numbers. Most notably, for the chromatic polynomial we arrive at

$$P(G;k) = \sum_{r=1}^{k} \binom{k}{r} \left(\sum_{X \subseteq V} (-1)^{|X|} a_r(X) \right), \tag{1.2}$$

where $a_r(X)$ denotes the number of ways to choose r independent sets $S_1, \ldots, S_r \subseteq V \setminus X$, such that $|S_1| + \cdots + |S_r| = n$. To the best knowledge of the authors, these characterizations are new and might be of independent combinatorial interest; in any case, their proofs are elementary.

1.3. Previous Work and Discussion. Previous research on graph partitioning is well characterised by the effort put into the graph colouring problem. This is the set partition problem where N are the vertices of a graph and $\mathcal{F}$ are its independent sets.

A way to solve this problem that goes back at least to Lawler [35], is to use dynamic programming over the subsets of N: Build a table $g(X, m)$ with entries for every $X \subseteq N$ and $m \leq k$. Iterate over the subsets in order of increasing size and use $g(X, m) = \sum_{S \in \mathcal{F}} g(X \setminus S, m-1)$, to check for each $m \leq k$ whether X can be covered by m of the subsets. Clearly, the algorithm's running time is bounded by $|\mathcal{F}|2^n n^{O(1)}$, and it is never worse than within a polynomial factor of $\sum_{S \in \mathcal{F}} 2^{n-|S|} \leq \sum_{\ell=0}^{n} \binom{n}{\ell} 2^\ell = 3^n$. Ingenious ways to enumerate and bound the size of the family $\mathcal{F}$ (corresponding to minimal dominating sets in the case of Domatic Number or to maximal independent sets in the case of Chromatic Number) have resulted in the time bounds $O(2.8718^n)$ for Domatic Number [20] and $O(2.4022^n)$ for Chromatic Number [11]. Reducing these constants towards two has been a perpetual algorithmic challenge (see Table 1.2), and the possibility of ever arriving within a polynomial factor of time 2^n, for example for Chromatic Number, has been a well-known open problem [48].

Our algorithms beat the running time of previous algorithms that decide k-colourability for small values of k. The exceptions are 3- and 4-colourability, which can be decided in time $O(1.3289^n)$ [7] and $O(1.7504^n)$ [11], respectively, well beyond the reach of our constructions.

For polynomial space, the first non-trivial algorithm for finding the chromatic number, by Christofides [14] in 1971, runs in time $n!n^{O(1)}$. Feder and Motwani [16]

gave a randomised linear space algorithm with running time $O\big((\chi/e)^n\big)$, improving Christofides' result for small values of χ. The running time of an algorithm by Angelsmark and Thapper [1] can be given as $O\big((2+\log\chi)^n\big)$, an asymptotic improvement over Christofides' result for all values of χ. Very recently, running times of the form $O(c^n)$ have appeared; Bodlaender and Kratsch [10] achieved $O(5.283^n)$ and Björklund and Husfeldt [9] arrived at $O(8.3203^n)$ and $O(2.4423^n)$. The bound given in the present paper, $O(2.2416^n)$, will improve whenever the running time for counting independent sets is improved, but there is little hope that this approach will ever arrive within a polynomial factor of 2^n because counting independent sets is $\#P$-complete [24, 45]. The existence of such an algorithm remains open.

For Domatic Number, exponential space algorithms that are faster than 3^n have appeared only recently. Fomin *et al.* [20] provided an $O(2.8718^n)$ time algorithm for deciding the domatic number, and Riege *et al.* [40] presented an $O(2.6949^n)$ time, polynomial space algorithm for deciding if the domatic number is at least three. No prior nontrivial polynomial space algorithm for the general problem is known to the authors.

The related counting problem of finding $P(G,k)$, the number of k-colourings of graph G, has been very well studied within Algebraic Graph Theory, where it is known as computing the chromatic polynomial. Anthony [2] surveys and compares previous methods, see also [8, 46]. The *Whitney expansion*,

$$P(G;k) = \sum_{H\subseteq E} (-1)^{|H|} k^{n-r\langle H\rangle}\,, \tag{1.3}$$

where $r\langle H\rangle$ is the rank of the subgraph induced by the edge set H, requires time 2^m. On some instances, a faster way is the *deletion–contraction method*, based on the recurrence

$$P(G;k) = P(G-e,k) + P(G/e,k)\,,$$

where $G-e$ and G/e are constructed by deleting or contracting edge e, which runs within a polynomial factor of $\big(\frac{1}{2}(1+\sqrt{5})\big)^{n+m} = O(1.6180^{n+m})$. Finally, the relation between P and the *Tutte polynomial* $\sum t_{ij}x^iy^j$,

$$P(G;k) = (-1)^{n-1}k\sum_{i=1}^{n-1} t_{i0}(1-k)^i$$

leads to an algorithm that runs within a polynomial factor of $\binom{m}{n-1} < m^n = 2^{n\log m} \leq 4^{n\log n}$. All these algorithms can be seen to run in polynomial space.

The application of inclusion–exclusion to combinatorial optimisation goes back to Ryser's formula for the permanent [42], which remains the most effective way to count the number of matchings in a bipartite graph. The first explicit reference to combinatorial optimisation is for the TSP by Kohn, Gottlieb, and Kohn [30], and a concise paper by Karp [29] applies the idea to a number of problems. Similar ideas were later used by Bax and Franklin [4, 5, 6] and Björklund and Husfeldt [9]. All of these examples consider partitions and other constrained covers from the family $\mathcal{F} = E$ of graph edges. The contribution of the present paper is to observe that inclusion–exclusion can be almost as powerful when the size of the family $\mathcal{F}$ is exponential in the size of the universe. For graph colouring, an earlier paper [9] already tentatively explores this idea.

2. Preliminaries.

2.1. The Principle of Inclusion–Exclusion. We begin with a basic principle of combinatorics. We state both the unweighted and the weighted version [42] and give a proof for completeness.

LEMMA 1. *Let B be a finite set with subsets $A_1, \ldots, A_n \subseteq B$. With the convention that $\bigcap_{i\in\varnothing} A_i = B$, the following holds:*

1. *The number of elements of B which lie in none of the A_i is*

$$\Big|\bigcap_{i=1}^{n} \overline{A_i}\Big| = \sum_{X\subseteq\{1,\ldots,n\}} (-1)^{|X|} \cdot \Big|\bigcap_{i\in X} A_i\Big| .$$

2. *Let $w\colon B \to R$ be a real-valued weight function extended to the domain 2^B by setting $w(A) = \sum_{a\in A} w(a)$. Then*

$$w\Big(\bigcap_{i=1}^{n} \overline{A_i}\Big) = \sum_{X\subseteq\{1,\ldots,n\}} (-1)^{|X|} \cdot w\Big(\bigcap_{i\in X} A_i\Big) .$$

Proof. We analyse the contribution of every element $a \in B$ to both sides of the expression. If a lies in none of the A_i then it contributes $w(a)$ to the left hand side. To the right hand side it contributes $w(a)$ exactly once, namely in the term corresponding to $X = \varnothing$. Conversely, assume that a lies in A_i for all $i \in I \neq \varnothing$. Its contribution to the left hand side is 0. On the right hand side, since a lies in the intersection $\bigcap_{i\in X} A_i$ for every $X \subseteq I$, its total contribution is

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

by the Binomial Theorem. □

2.2. Fast Zeta Transform on Subset Lattices. Let N be an n-element set and R the set of real numbers. The *zeta transform* [41] on the subset lattice $(2^N, \subseteq)$ of N is an operator that maps every function $f\colon 2^N \to R$ into another function $\widehat{f}\colon 2^N \to R$, defined by

$$\widehat{f}(Y) = \sum_{S\subseteq Y} f(S) \quad \text{for } Y \subseteq N .$$

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

The straightforward way to compute the zeta transform evaluates $\widehat{f}(Y)$ anew at every Y, using $O(3^n)$ additions in total. However, this can be improved to only $O(n2^n)$ additions using Yates's method [50], [31, Section 4.3.4]; the resulting algorithm is sometimes called the fast Möbius transform [28, 32], in this paper we term it the *fast zeta transform*.

LEMMA 2. *Let N be a set of n elements. Then the zeta transform on the subset lattice of N, restricted to functions to the integer range $[-M, M]$, can be computed in $O(n2^n)$ additions with $O(n \log M)$-bit integers.*

Proof. We assume $N = \{1, 2, \ldots, n\}$ for convenient notation. Define $g_0, g_1, \ldots, g_n$ at every $Y \subseteq N$ by $g_0(Y) = f(Y)$ and

$$g_i(Y) = \begin{cases} g_{i-1}(Y) + g_{i-1}(Y \setminus \{i\}) , & \text{if } i \in Y , \\ g_{i-1}(Y) , & \text{otherwise} \end{cases} \qquad (i = 1, 2, \ldots, n) .$$

One can show by induction on i that $g_i(Y) = \sum_S f(S)$, where S runs through all subsets of Y such that

$$\{\, j \in S \colon j > i \,\} = \{\, j \in Y \colon j > i \,\}\,.$$

Specifically, we have $g_n = \widehat{f}$. When we evaluate the recursion we store all the values $g_i(Y)$ as they are computed; thus we can compute each g_i from g_{i-1} in $O(2^n)$ additions. We conclude that $\widehat{f}$ can be computed in $O(n2^n)$ additions with integers from the range $[-2^n M, 2^n M]$. □

2.3. Model of Computation. To avoid cumbersome runtime bounds we use the notation O^* to suppress polylogarithmic factors, that is, we write $O^*(\tau)$ when we have $O(\tau \log^d \tau)$ for some constant d in the familiar Landau notation. We write $\log x$ for $\lceil \log_2 x \rceil$.

The model of computation used in this work is the random access machine. We operate on large integers of bit length polynomial in n, so we make the conservative assumption that operations (including comparison) are considered unit-time only for constant-size integers. In this model, two b-bit integers can be added, subtracted, and compared in $O(b)$ time, and multiplied in $O(b \log b \log\log b) = O^*(b)$ time [43], recently improved to $b \log b 2^{O(\log^* b)}$ [21].

3. Results. We begin with an algorithm for counting set covers in Section 3.1. This is the simplest version of our results, but already suffices for computing the chromatic number, see in Proposition 1 of Section 4.1. In Section 3.2, we extend the result to counting set partitions. In Section 3.3 we consider the problem of summing over weighted partitions, where the weight of a partition factorises into a product of the weights of its members. Finally, in Section 3.4 we address the optimisation version of the general problem, finding a partition that maximises the sum of the weights of its members, provided that the weights are small integers.

3.1. Set Cover. A *set system* consists of an n-element set N and a family $\mathcal{F}$ of subsets of N. A *k-cover* is a tuple $(S_1, \ldots, S_k)$ over $\mathcal{F}$ such that

$$S_1 \cup \cdots \cup S_k = N\,, \tag{3.1}$$

possibly with overlap and repetition. This leads to a counting problem:

Counting Set Covers
Input: A set system $(N, \mathcal{F})$ and an integer k.
Output: The number $c_k(\mathcal{F})$ of k-covers.

Lemma 3. *Let $a(X) = \bigl|\{\, S \in \mathcal{F} \colon S \cap X = \varnothing \,\}\bigr|$ denote the number of sets in $\mathcal{F}$ that avoid X. Then*

$$c_k(\mathcal{F}) = \sum_{X \subseteq N} (-1)^{|X|} a(X)^k\,. \tag{3.2}$$

Proof. This is a direct application of Lemma 1, where B is the set of k-tuples $(S_1, \ldots, S_k)$ over $\mathcal{F}$, and $A_i \subseteq B$ are those k-tuples that avoid the singleton $\{i\}$, i.e., $i \notin S_1 \cup \cdots \cup S_k$. Then $c_k(\mathcal{F})$ is exactly the number of k-tuples in B that lie in none of the A_i. Furthermore, $\bigcap_{i \in X} A_i$ contains those k-tuples that avoid all of X, so that $\bigl|\bigcap_{i \in X} A_i\bigr| = a(X)^k$. □

Theorem 1. Counting Set Covers *can be solved in $O^*(2^n)$ time.*

Proof. Observing that

$$a(X) = \sum_{S \subseteq N \setminus X} f(S) = \widehat{f}(N \setminus X)\,,$$

where f is the indicator function of $\mathcal{F}$, we can compute a table containing $a(X)$ for all $X \subseteq N$ using the fast zeta transform. We then raise each entry to the kth power and sum these values according to (3.2). □

3.2. Set Partition. A *k-partition* of a set system is a tuple $(S_1, \ldots, S_k)$ over $\mathcal{F}$ such that

$$S_1 \cup \cdots \cup S_k = N\,, \qquad S_i \cap S_j = \varnothing \quad (i \neq j)\,. \tag{3.3}$$

Counting Set Partitions
Input: A set system $(N, \mathcal{F})$ and an integer k.
Output: The number $p_k(\mathcal{F})$ of k-partitions.

Lemma 4. *Let $a_k(X)$ denote the number of k-tuples $(S_1, \ldots, S_k)$ over $\mathcal{F}$ for which $S_c \cap X = \varnothing$ $(1 \leq c \leq k)$ and*

$$|S_1| + \cdots + |S_k| = n\,. \tag{3.4}$$

Then

$$p_k(\mathcal{F}) = \sum_{X \subseteq N} (-1)^{|X|} a_k(X)\,. \tag{3.5}$$

Proof. Again, we appeal to Lemma 1, where B is the set of k-tuples $(S_1, \ldots, S_k)$ from $\mathcal{F}$ that satisfy (3.4), and $A_i \subseteq B$ are those k-tuples that avoid $\{i\}$. Then $p_k(\mathcal{F})$ counts those choices that lie in none of the A_i, and $\bigl|\bigcap_{i \in X} A_i\bigr| = a_k(X)$. □

Theorem 2. Counting Set Partitions *can be solved in $O^*(2^n)$ time.*

Proof. Write $\widehat{f^{(\ell)}}(Y)$ for the number of sets $S \in \mathcal{F}$ with $|S| = \ell$ and $S \subseteq Y$, and observe that it is the zeta transform of the indicator function

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

Thus, the fast zeta transform computes a table containing $\widehat{f^{(\ell)}}(Y)$ for all ℓ and Y within the stated time bound.

Once these values have been computed, we can evaluate $a_k(X)$ for any fixed $X \subseteq N$ by dynamic programming in time polynomial in k and n as follows. Define $g(j, m)$ to be the number of j-tuples $(S_1, \ldots, S_j)$ for which $S_c \cap X = \varnothing$ $(1 \leq c \leq j)$ and $|S_1| + \cdots + |S_j| = m$, formally

$$g(j, m) = \sum_{\ell_1 + \cdots + \ell_j = m} \prod_{c=1}^{j} \widehat{f^{(\ell_c)}}(N \setminus X)\,,$$

Then $a_k(X) = g(k, n)$, and we can compute it from the recursion

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

observing $g(1, m) = \widehat{f^{(m)}}(N \setminus X)$. Finally, we sum the $a_k(X)$ according to (3.5). □

3.3. Sum of Weighted Partitions. We generalise the results of the previous section by associating each partition with a weight.

As before, let N be a ground set of n elements. If $S_1, \ldots, S_k$ are subsets of a set $Y \subseteq N$, we call $\mathbf{S} = (S_1, \ldots, S_k)$ a *k-tuple on Y*; if, additionally, the members $S_1, \ldots, S_k$ are mutually disjoint and their union is Y, then $\mathbf{S}$ is a *k-partition of Y*. Note that k may well be larger than n, and some of the sets S_c may well be empty; this generality turns out to be useful in some applications.

Let f be a weight function that associates each k-tuple on the ground set N with an integer. Let $p_k(f)$ denote the sum of all weighted k-partitions of N, that is,

$$p_k(f) = \sum_{\mathbf{S}} f(\mathbf{S})\,,$$

where $\mathbf{S} = (S_1, \ldots, S_k)$ runs through all ordered k-partitions of N. We consider the problem of computing $p_k(f)$ in the special case where the weight $f(\mathbf{S})$ factorises into a product $f_1(S_1) \cdots f_k(S_k)$, in other words, f is a tensor product $f_1 \otimes \cdots \otimes f_k$:

Sum Weighted Partitions
Input: An n-element set N and functions $f_1, \ldots, f_k$ from the subsets of N to integers from the range $[-M, M]$.
Output: The value $p_k(f)$ for $f = f_1 \otimes \cdots \otimes f_k$.

We note that if every weight function f_c is simply the indicator of a set system $\mathcal{F}$, then $p_k(f)$ coincides with the $p_k(\mathcal{F})$ introduced in the previous section.

We will show that Sum Weighted Partitions can be solved in time 2^n upto some factors polynomial in n, k, and $\log M$.

Theorem 3. Sum Weighted Partitions *can be solved in $O^*(2^n k \log M)$ time.*

We prove this result in the remainder of this section. We begin with an inclusion–exclusion expression.

Lemma 5. *Let $b_k(X)$ denote the sum of the weights $f(\mathbf{S})$ over all k-tuples $\mathbf{S} = (S_1, \ldots, S_k)$ on $N \setminus X$ such that*

$$|S_1| + \cdots + |S_k| = n\,. \tag{3.6}$$

Then

$$p_k(f) = \sum_{X \subseteq N} (-1)^{|X|} b_k(X)\,.$$

Proof. We appeal to the weighted version of Lemma 1. Let B denote the set of k-tuples $(S_1, \ldots, S_k)$ on N satisfying (3.6), let $A_i \subseteq B$ denote those k-tuples that avoid $\{i\}$, and define $w(\mathbf{S}) = f_1(S_1) \cdots f_k(S_k)$. Then $p_k(f) = w\bigl(\overline{A_1} \cap \cdots \cap \overline{A_k}\bigr)$ and $w\bigl(\bigcap_{i \in X} A_i\bigr) = b_k(X)$. □

It remains to show how to compute $b_k(X)$. Now we make use of the factorisation of f.

Lemma 6. *The values $b_k(X)$ can be computed for all $X \subseteq N$ in $O^*(2^n k \log M)$ total time.*

Proof. Write $\widehat{f}_c^{(\ell)}(Y)$ for the sum of the weights $f_c(S)$ over all $S \subseteq Y$ with $|S| = \ell$. Using fast zeta transform we can compute the values $\widehat{f}_c^{(\ell)}(Y)$ for all c, ℓ, and Y in $O^*(kn2^n \log M) = O^*(2^n k \log M)$ time. Once these values have been computed, we can evaluate $b_k(X)$ for any fixed $X \subseteq N$ by dynamic programming in time polynomial in k and n, and logarithmic in M, as described in the next paragraphs.

To obtain a runtime that is roughly linear in k, as claimed, we compute $b_k(X)$ in a divide-and-conquer manner, instead of the related sequential approach used in the proof of Theorem 2. To this end, assume w.l.o.g. that $k = 2^q$ for some integer q. (Otherwise, if $k < 2^q < 2k$, consider a larger input with $2^q - k$ additional input functions that evaluate to 1 at $\varnothing$, and to 0 elsewhere.) Let $g(s,t,m)$ denote the sum of the weights $f_s(S_s) \cdots f_t(S_t)$ over all $(t-s+1)$-tuples $(S_s, \ldots, S_t)$ on $N \setminus X$ such that $|S_s| + \cdots + |S_t| = m$, formally

$$g(s,t,m) = \sum_{\ell_s + \cdots + \ell_t = m} \prod_{c=s}^{t} \widehat{f}_c^{(\ell_c)}(N \setminus X)\,,$$

where each ℓ_c, for $s \le c \le t$, runs through the integers in $\{0, \ldots, m\}$. Observe that $b_k(X) = g(1,k,n)$. Our algorithm computes $g(1,k,n)$ via the recurrence equation

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

where m_0 and m_1 run through the integers in $\{0, \ldots, m\}$, with the base case

$$g(c,c,m) = \widehat{f}_c^{(m)}(N \setminus X) \quad \text{for } c = 1, \ldots, k\,.$$

Note that while the above recurrence equation holds for any pair (s,t), our algorithm will compute $g(s,t,m)$ only at specific pairs encountered when iteratively halving subranges of $\{1, \ldots, k\}$.

To analyse the runtime, we first bound the sizes of the integers involved in the computations. We show by induction on $t-s$ that the absolute value of $g(s,t,m)$ is bounded above by $n^m M^{t-s+1} m^{t-s}$. For the base case, $g(c,c,m)$, this holds, since the absolute value of $\widehat{f}_c^{(m)}(N \setminus X)$ is at most $n^m M$. For the general case, $g(s,t,m)$, our claim can be verified by induction, as follows. If $m_0 + m_1 = m$, the absolute value of $g(s,c,m_0)g(c+1,t,m_1)$, whenever $s \le c < t$, is bounded by

$$\begin{aligned}\big(n^{m_0} M^{c-s+1} m_0^{c-s}\big)\big(n^{m_1} M^{t-c} m_1^{t-c-1}\big) &\le n^{m_0+m_1} M^{t-s+1} m^{c-s} m^{t-c-1} \\ &= n^m M^{t-s+1} m^{t-s-1}\,.\end{aligned}$$

Thus, the absolute value of $g(s,t,m)$ is at most $n^m M^{t-s+1} m^{t-s}$.

We then complete the runtime analysis. Let $T(j)$ denote the time needed for computing $g(s,t,m)$ for all $m = 0, \ldots, n$ but fixed s and t satisfying $t - s + 1 = j$. By the above analysis,

$$T(j) = O^*\big(n^2 \log\big(n^n M^j n^{j-1}\big)\big) = O^*\big(n^3 + jn^2 \log M\big)\,.$$

(We note that one could save roughly a factor n by computing the convolution using a fast Fourier transform.) Since k is a power of 2, the total runtime is given by

$$\begin{aligned}T(k) + 2 \cdot T(k/2) + 4 \cdot T(k/4) + \cdots + k \cdot T(1) &= O^*\big(kn^3 + (\log k) kn^2 \log M\big) \\ &= O^*\big(kn^2(n + \log M)\big)\,.\end{aligned}$$

Thus we have shown how to compute $b_k(X)$ for *all* $X \subseteq N$ within the stated total time. □

3.4. Finding a Heaviest Partition. Next we turn to an optimisation problem:

MAX WEIGHTED PARTITION
Input: An n-element set N and functions $f_1, \ldots, f_k$ from the subsets of N to integers from the range $[-M, M]$.
Output: A k-partition $(S_1, \ldots, S_k)$ of N that maximises $f_1(S_1) + \cdots + f_k(S_k)$.

We reduce this problem to SUM WEIGHTED PARTITIONS. The embedding technique and self-reducibility argument we use are rather standard; for some previous instantiations see, e.g., Williams [47] and Koivisto [33].

THEOREM 4. MAX WEIGHTED PARTITION *can be solved in* $O^*(2^n k^2 M)$ *time.*

Proof. Assume without loss of generality that the range of the input functions is $\{0, 1, \ldots, M\}$. (For, if this was not the case, we could add M to every value $f_c(S)$ and work within the range $\{0, 1, \ldots, 2M\}$.)

With each input function f_c associate another function f'_c defined by $f'_c(S) = \beta^{f_c(S)}$ for $S \subseteq N$, where β is a suitable number to be specified soon. Let f' denote the tensor product of $f'_1, \ldots, f'_k$, and observe that the sum of the k-partitions of N weighted by the product of the new weights f'_c can be expressed as

$$p_k(f') = \sum_{\mathbf{S}} \beta^{f_1(S_1)+\cdots+f_k(S_k)} = \sum_{r=0}^{kM} \alpha_r \beta^r \,,$$

where α_r is the number of k-partitions of N for which the original weight (i.e., the sum of the original weights) equals r. If we choose β sufficiently large, say $\beta = k^n + 1$, this coefficient representation is unique, and we can deduce the coefficients α_r from the number $p_k(f')$; in particular, we find the largest r for which $\alpha_r > 0$, that is, the maximum weight achieved by any k-partition of N. As the range of each function f'_c is $\{0, 1, \ldots, \beta^M\}$, we can compute $p_k(f')$ in $O^*(2^n kM \log \beta) = O^*(2^n kM)$ time; deducing the coefficients α_r is then much easier.

It remains to show how to find a k-partition that achieves a given weight W when such a partition exists. We use self-reducibility as follows. Think of a partition into k parts as assigning a colour $C(i) \in \{1, \ldots, k\}$ to each element $i \in N$. Take an element i from N. We search for a colour $C(i) \in \{1, 2, \ldots, k\}$ such that the remaining elements in $N \setminus \{i\}$ have a colouring that together with $C(i)$ achieves the weight W. More precisely, for a candidate colour $C(i) = c$ define $\widetilde{f}_c(S) = f_c(S)$ if $i \in S$, and $\widetilde{f}_c(S) = 0$ otherwise; for $j \neq c$ set $\widetilde{f}_j = f_j$. Then compute the maximum weight over all k-partitions of $N \setminus \{i\}$ for the modified weight functions, say $\widetilde{W}$. If $\widetilde{W} = W$, then we may assign $C(i)$ to c and iterate for the remaining elements in $N \setminus \{i\}$ (i.e., find a k-partition of $N \setminus \{i\}$ that achieves the weight W for the modified weight functions $\widetilde{f}_1, \ldots, \widetilde{f}_k$). Note that for at least one assignment $C(i) = c$ we must have $\widetilde{W} = W$. Finally we obtain a required k-partitioning $(S_1, \ldots, S_k)$ by setting $S_c = \{i : C(i) = c\}$. In the first iteration we need to compute the maximum weight at most k times, which takes $O^*(2^n k^2 M)$ time. This is also the overall time complexity as the time bounds for the smaller subproblems decay exponentially fast. □

4. Applications.

4.1. Graph Colouring. A *k-colouring* of a graph $G = (V, E)$, $|V| = n$ is a mapping $V \to \{1, \ldots, k\}$ that gives different values ('colours') to neighbouring vertices. The *chromatic number* $\chi(G)$ is the smallest k for which G admits a k-colouring.

In this section we have $N = V$ and $\mathcal{F} = \mathcal{S}$, the family of nonempty independent sets of G.

Chromatic Number
Input: A graph $G = (V, E)$ and an integer k.
Output: Is $\chi(G) \le k$?

Lemma 7. *$\chi(G) \le k$ if an only if $c_k(\mathcal{S}) > 0$.*

Proof. A k-colouring is a covering with k independent sets, so if it exists, $c_k(\mathcal{S}) > 0$. On the other hand, if $S_1, \ldots, S_k$ cover V (possibly non-distinct and non-disjoint) then $C(v) = \min\{\, c \colon v \in S_c \,\}$ is a colouring of size at most k. □

Proposition 1. Chromatic Number *can be solved in $O\bigl(2^n nk \operatorname{polylog}(nk)\bigr)$ time and $O(2^n n)$ space.* Note that an $O^*(2^n)$ bound is immediate from Theorem 1, but here we give a more careful analysis and a more direct proof.

Proof. Recall that $a(X)$ denotes the number of $S \in \mathcal{S}$ with $S \cap X = \varnothing$, and let $N(v) = \{v\} \cup \{\, u \in V \colon uv \in E \,\}$ denote v and its neighbours.

We will first argue that $a(X)$ satisfies the recurrence

$$a(X) = a\bigl(X \cup \{v\}\bigr) + a\bigl(X \cup N(v)\bigr) + 1\,, \qquad (v \notin X)\,. \tag{4.1}$$

To see this consider the nonempty independent sets S disjoint from X. They can be partitioned into two classes: either $v \in S$ or $v \notin S$. The latter sets are counted in $a\bigl(X \cup \{v\}\bigr)$. It remains to argue that the sets $S \ni v$ are counted in $a\bigl(X \cup N(v)\bigr) + 1$. We will do this by counting the equipotent sets $S \setminus \{v\}$ instead. Since S contains v and is independent, it cannot contain other vertices from $N(v)$. Thus $S \setminus \{v\}$ is disjoint from $N(v)$ and X. Now, either S is the singleton $\{v\}$ itself, accounted for by the '+1' term, or $S \setminus \{v\}$ is a nonempty independent set and therefore counted in $a\bigl(X \cup N(v)\bigr)$.

The recurrence (4.1) gives us an algorithm for $a(X)$, at the bottom of the recursion we have $a(V) = 0$. The operations are on $O(n)$-bit integers, so the entire table can be constructed in $O(2^n n)$ time and space by storing every $a(X)$ as it is computed. Finally, we need to raise each of the $a(X)$ to the kth power as we sum them, amounting to $\log k$ multiplications and additions of nk-bit numbers. □

Chromatic polynomial
Input: A graph G, and an integer $k \in \{0, 1, \ldots, n\}$.
Output: The number of k-colourings of G.

Proposition 2. *The number $P(G; k)$ of k-colourings of an n-vertex graph G can be found in time and space $O^*(2^n)$.*

Proof. Every partition into r non-empty independent sets corresponds to $(k)_r = k(k-1)(k-2)\cdots(k-r+1)$ different k-colourings, so

$$P(G; k) = \sum_{r=1}^{k} \frac{k!}{(k-r)!} \frac{p_r(\mathcal{S})}{r!} = \sum_{r=1}^{k} \binom{k}{r} p_r(\mathcal{S})\,,$$

which can be computed using Theorem 2. □

The function $P(G; \cdot)$ from integers to integers is known to be a degree n polynomial, so we can recover its coefficients by computing $P(G; k)$ at $k = 0, 1, \ldots, n$ and interpolating the unique polynomial through these points. This representation then allows us to evaluate the chromatic polynomial at other points, such as computing $P(G; -1)$, the number of acyclic orientations of G [44].

List colouring is a variant of graph colouring where the admissible colours of each vertex are restricted to a list $L(v) \subseteq Z$. The graph is *L-colourable* if there is a colouring $V \to Z$ such that every $v \in V$ receives a colour from $L(v)$. This generalises k-colouring from the case where every $L(v)$ is $\{1, \ldots, k\}$.

List Colouring
Input: A graph $G = (V, E)$, and a list $L(v)$ for every $v \in V$.
Output: Can G be L-coloured?

Proposition 3. List Colouring *can be solved in $O^*(2^n)$ time.*

Proof. First observe that we can assume that every list $L(v)$ contains at most $d(v)$ colours, so that the total number of colours is at most $2|E|$. Otherwise we could replace every list for which $|L(v)| > d(v)$ by only a single, fresh colour c_v, solve the resulting list colouring problem, and subsequently clean up the result by replacing every c_v by a colour from $L(v)$ that is not used by any of v's neighbours.

We will reduce to Max Weighted Partition. Assume w.l.o.g. that the union of the available colours is $\{1, 2, \ldots, k\}$ with $k \leq n(n-1)$. For each color $c = 1, \ldots, k$ define the function $f_c \colon 2^V \to \{0, 1\}$ by $f_c(S) = 1$ if S is independent (or empty) and $c \in L(v)$ for all $v \in S$; otherwise, $f_c(S) = 0$. Let W denote the maximum value of $f_1(V_1) + \cdots + f_k(V_k)$ over all k-partitions $(V_1, \ldots, V_k)$ of V.

Now the input graph is L-colourable if and only if $W = k$. Namely, if $W = k$, there are k independent (or empty) sets $V_1, \ldots, V_k$ such that their union is V and each V_c only contains vertices v for which c is from $L(v)$. Thus, the sets $V_1, \ldots, V_k$ determine a unique list-colouring (C that satisfies $v \in V_{C(v)}$ for all $v \in V$). On the other hand, if C is a list colouring, then each set $V_c = \{v : C(v) = c\}$ is independent (or empty) and $c \in L(v)$ for all $v \in V_c$, implying $W = k$.

By Theorem 4, the maximum weight W can be computed in $O^*(k^2 2^n) = O^*(2^n)$ time. □

We raise the question of computing in c^n time the *list chromatic number*, which is the smallest k such that the graph can be L-coloured whenever $|L(v)| \geq k$ for every $v \in V$.

Next, we turn to the Chromatic Sum problem, also called Minimum Colour Sum.

Chromatic Sum
Input: A graph $G = (V, E)$.
Output: A colouring $C : V \to \{1, \ldots, n\}$ that minimises $\sum_{v \in V} C(v)$.

The objective for ordinary graph colouring is to minimise $\max_{v \in V} C(v)$. Simple examples like  show that the optimal colour classes for Chromatic Number and Chromatic Sum, or even the number of colours used, are not the same.

Proposition 4. Chromatic Sum *can be solved in time $O^*(2^n)$.*

Proof. We reduce to Max Weighted Partition. Define the functions $f_1, \ldots, f_n$ as

$$f_c(S) = \begin{cases} -c|S|\,, & \text{if } S \text{ is independent or empty,} \\ -n^2\,, & \text{otherwise.} \end{cases}$$

This way, every partition into k independent sets will have value $-(1 \cdot |S_1| + \cdots + k \cdot |S_k|)$, which is larger than any partition that uses a dependent set, and is maximised at a partition that minimises the colour sum of k colours. □

4.2. Other Graph Partitioning Problems. The properties in Table 1.1 are all polynomial time checkable given G, so membership of a given vertex subset in $\mathcal{F}$ can be verified in time polynomial in n, an overhead that vanishes in our notation. The exception is Partition into Hamiltonian Subgraphs, because Hamiltonicity is an NP-hard property. However, we can enumerate all vertex subsets U such that $G[U]$ is Hamiltonian in time $O^*(2^n)$ [26]. Thus we can build a large incidence table for $\mathcal{F}$ beforehand and perform the membership tests in constant time when they are needed.

Another set cover variant is worth pointing out as well: A *k-packing* of a set system $(N, \mathcal{F})$ is a tuple $(S_1, \ldots, S_k)$ over $\mathcal{F}$ such that $S_i \cap S_j = \varnothing$ $(i \neq j)$. This is interesting only if $\mathcal{F}$ does not contain the empty set.

Counting Set Packings
Input: A set system $(N, \mathcal{F})$ and an integer k.
Output: The number of k-packings.

Proposition 5. Counting Set Packings *can be solved in $O^*(2^n)$ time.*

Proof. The problem reduces to Sum Weighted Partitions with $k+1$ input functions, where $f_1, \ldots, f_k$ are the indicator function on $\mathcal{F}$ and f_{k+1} is identically 1. □

We give another example of a *weighted* partitioning problem:

Max k-Cut
Input: A graph $G = (V, E)$ with edge weights $w(e)$ for $e \in E$,
Output: A partition of V into k subsets $V_1, \ldots, V_k$ maximising the sum of the weights $w(e)$ over the edges $e \in E$ whose end points lie in different sets V_i and V_j.

The unweighted version is equivalent to yet another graph colouring problem, Max k-Colourable Subgraph on the complementary graph.

Proposition 6. Max k-Cut *restricted to integer weights in the range $[-M, M]$ can be solved in $O^*(2^n M)$ time.*

Proof. The problem reduces to Max Weighted Partition with k input functions $f_1, \ldots, f_k$, defined by $f_c(S) = -\sum_{e \in E(S)} w(e)$, where $E[S]$ consists of the edges in E whose end points both lie in S. Namely, minimising the total weight of edges within the sets $V_1, \ldots, V_k$ is equivalent to maximising the total weight of edges between these sets. □

4.3. Bayesian Partition Models. First consider Bayesian model-based clustering (see, e.g., [3, 17, 27, 37]). Let $y_1, \ldots, y_m$ be data points, and let $\mathbf{S}$ denote the unknown partition of the index set $\{1, \ldots, m\}$ into k clusters $S_1, \ldots, S_k$. Each cluster S_c is associated with a component model parameterised by θ_c, such that the distribution of y_j, given that j belongs to S_c, is $P(y_j | \theta_c)$. The partition $\mathbf{S}$ and the parameters $\theta = (\theta_1, \ldots, \theta_k)$ are further assigned modular prior distributions, $P(\mathbf{S}) = \prod_c \rho_c(S_c)$ and $P(\theta) = \prod_c q_c(\theta_c)$. (Note that one obtains valid functions $\rho_1, \ldots, \rho_k$, e.g., by dividing any respective, given weight functions $w_1, \ldots, w_k$ by the kth root of the normalisation constant $\sum_{\mathbf{S}} \prod_c w_c(S_c)$.)

We will focus on the problem of computing the so-called marginal likelihood of a clustering model. This quantity, sometimes called the evidence, is defined as the total probability of the data $\mathbf{y}$, and is obtained by integrating out the unknown partition

and the model parameters:

$$P(\mathbf{y}) = \sum_{\mathbf{S}} \prod_{c=1}^{k} f_c(S_c) = p_k(f_1 \otimes \cdots \otimes f_k)\,,$$

where

$$f_c(S_c) = \rho_c(S_c) \int \prod_{j \in S_c} P(y_j|\,\theta_c) q_c(\theta_c) \mathrm{d}\theta_c\,. \tag{4.2}$$

Computing and comparing $P(\mathbf{y})$ for different numbers k is useful, e.g., for selecting a plausible number of clusters. For finding an optimal clustering, one usually considers the corresponding max–sum expression.

Consider then a similar class of models often used in supervised classification and related tasks; now one seeks clusters in a feature space. Partition models (or, equivalently, decision graphs) generalize the popular decision tree models in that they can represent arbitrary (not only tree-structured) partitions of a feature space (e.g., [13]). A probabilistic partition model specifies a conditional distribution of a class variable y given a feature vector x, as follows. The feature space X is partitioned into disjoint subsets $S_1, \ldots, S_k$. Each subset S_c is assigned a simple model parametrised by θ_c, such that the distribution of y, given that x belongs to S_c, is $P(y|\,x, \theta_c)$. In what follows, we will assume that X is a finite set of n elements. We further consider a Bayesian approach [34] where the partition $\mathbf{S}$ and the parameters $\theta = (\theta_1, \ldots, \theta_k)$ are assigned prior distributions, $P(\mathbf{S}) = \prod_c \rho_c(S_c)$ and $P(\theta) = \prod_c q_c(\theta_c)$.

Given m data points $(x_1, y_1), \ldots, (x_m, y_m)$, the marginal likelihood, or the evidence, of a Bayesian partition model is defined as the conditional probability of the observed classifications $\mathbf{y} = (y_1, \ldots, y_m)$ given the respective features $\mathbf{x} = (x_1, \ldots, x_m)$, and is given by

$$P(\mathbf{y}|\,\mathbf{x}) = \sum_{\mathbf{S}} \prod_{c=1}^{k} f_c(S_c) = p_k(f_1 \otimes \cdots \otimes f_k)\,,$$

where

$$f_c(S_c) = \rho_c(S_c) \int \prod_{j=1:x_j \in S_c}^{m} P(y_j|\,x_j, \theta_c) q_c(\theta_c) \mathrm{d}\theta_c\,. \tag{4.3}$$

The quantity $P(\mathbf{y}|\,\mathbf{x})$ is useful, for instance, in feature selection: one computes $P(\mathbf{y}|\,\mathbf{x})$ for different sets of feature variables, x, and selects the one that gives the largest value. If one is interested in the best partition, the sum–product expression is replaced by its max–sum counterpart.

We see that in both cases, in data clustering and partition models, the marginal likelihood can be computed by the algorithm we gave for Sum Weighted Partitions. Here we assume that, as usual [13, 34], the parametric models and the priors ρ_c and q_c are chosen such that the terms (4.2) and (4.3) can be efficiently computed for any given set S_c, and are given as input. Then the inclusion–exclusion algorithm computes the quantities $P(\mathbf{y})$ and $P(\mathbf{y}|\mathbf{x})$ in $O^*(2^m)$ and $O^*(2^n)$ arithmetic operations, respectively. However, it should be noted that the weights are typically rational numbers rather than small integers; it is not clear whether fixed precision computation produces numerically stable results, or whether it is better to operate

with large integers to get accurate results. Likewise, it seems that our algorithm for Max Weighted Partition is not suitable for solving the optimisation versions of these problems, although it may produce good approximations.

5. Polynomial Space. Theorem 5.

1. *If membership in $\mathcal{F}$ can be decided in $n^{O(1)}$ space and time then* Counting Set Covers *and* Counting Set Partitions *can be solved $n^{O(1)}$ space and $O^*(3^n)$ time.*
2. *If $f_1, \ldots, f_k$ can be computed in $n^{O(1)}$ space and time then* Sum Weighted Partitions *and* Max Weighted Partitions *can be solved in $(nk)^{O(1)}$ space and $O^*(3^n k \log M)$ or $O^*(3^n k^2 M)$ time, respectively.*

Proof. We look at Counting Set Covers first. The only part of our constructions that is not in polynomial space are the tables for $a(X)$, which we computed using the fast zeta transform. Instead of this, we need to compute $a(X)$ anew for every term when we evaluate (3.2). To compute $a(X)$ we test every subset $S \subseteq N \setminus X$ for membership in $\mathcal{F}$, amounting to $2^{|N|-|X|}$ membership tests. The total running time is then within a polynomial factor of

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

The argument for Counting Set Partitions is similar. To find $a_k(X)$, the dynamic program needs to evaluate $\widehat{f}^{(\ell)}(N \setminus X)$ for $\ell = 1, \ldots, n$, which again amounts to testing every subset $S \subseteq N \setminus X$ for membership in $\mathcal{F}$.

For Sum Weighted Partitions the dynamic program for $b_k(X)$ needs to evaluate $\widehat{f}_c^{(\ell)}(N \setminus X)$ for $c = 1, \ldots, k$ and $\ell = 1, \ldots, n$. Each of these requires us to evaluate $f_c(S)$ for all $S \subseteq N \setminus X$ with $|S| = \ell$, leading to the same calculation as above.

Finally, Max Weighted Partition reduces to Sum Weighted Partitions in polynomial space. □

For example, this works for all the problems in Table 1.1, except for Partition into Hamiltonian Subgraphs. We also note that the covering and partitioning problems can be solved in polynomial space and time $O^*(2^n|\mathcal{F}|)$ if $\mathcal{F}$ can be enumerated with polynomial delay, which may be faster if $\mathcal{F}$ is small. More interestingly, there are cases where we can enumerate the members of $\mathcal{F}$ faster than checking all subsets. The next two subsections consider such examples.

5.1. Chromatic Number. We spell out the implication for Chromatic Number, turning to a rich literature about counting independent sets. Very recently, continuing a line of improvements, Fürer and Kasiviswanathan [22] showed that the independent sets in a n-vertex graph can be counted in time bounded by $O(1.2461^n)$.

Proposition 7. Chromatic Number *can be solved in $O(2.2461^n)$ time and polynomial space.*

Proof. To evaluate $a(X)$ in (3.2) we construct the graph $G[V \setminus X]$ induced by the complement of X and run the algorithm from [22] in time $O(1.2461^{n-|X|})$. The total running time becomes

$$\sum_{r=0}^{n} \binom{n}{r} O(1.2461^{n-r}) = O(2.2461^n) \,.$$

□

5.2. Domatic Number. A vertex set $D \subseteq V$ *dominates* the graph $G = (V, E)$ if each vertex in V has distance at most one to a vertex in D. The *domatic number* $\delta(G)$ of G is the largest integer k such that there is a partition $V = D_1 \cup \cdots \cup D_k$ into pairwise disjoint dominating subsets.

Domatic Number
Input: A graph $G = (V, E)$, and an integer k.
Output: Is $\delta(G) \geq k$?

A polynomial space algorithm for Domatic Number in $O^*(3^n)$ time is immediate from Theorem 5. To improve this result, we will *pack* the vertices with minimal dominating sets instead of partitioning them into dominating sets. A dominating set is *minimal* if removing any of its vertices destroys the dominance property. Fomin *et al.* [20] observed that the minimal dominating sets can be enumerated faster than by considering all vertex subsets.

Proposition 8. Domatic Number *can be solved in* $O(2.8718^n)$ *time and polynomial space.*

Proof. Set $N = V$ and let $\mathcal{F}$ denote the family of minimal dominating sets of G. We will reduce to Counting Set Packings of Proposition 5. To see that this solves the original problem, a packing with k sets from $\mathcal{F}$ can be extended into a k-partition into (non-minimal) dominating sets by adding $N \setminus \big(S_1 \cup \cdots \cup S_k\big)$ to S_1. Conversely, any k-partition into dominating sets can be shrunk into a k-packing from $\mathcal{F}$.

For the time bound, the algorithm in Proposition 5 reduces to solving an instance of Sum Weighted Partitions where the f_c are the indicator function of $\mathcal{F}$, and as in the proof of Theorem 5, this basically amounts to computing the values $\widehat{f}_c^{(\ell)}(Y)$ for all values of ℓ and Y in polynomial space. Using the enumeration algorithm and notation from [20, proof of Theorem 5.1], the time to compute $\widehat{f}_c^{(\ell)}(Y)$ is $O(\lambda^{n+\alpha_4(n-|Y|)})$ for $\lambda < 1.1487$ and $\alpha_4 = 2.9248$, which leads to the stated bound. (However, as the authors point out, it is unclear exactly how close this bound is from the true running time of their branching algorithm.) □

6. Approximation. Our techniques can be used to create exponential-time approximation algorithms. We present the idea in terms of graph colouring.

Proposition 9. *For every* $\epsilon > 0$*, the chromatic number* χ *of a graph on* n *vertices can be approximated by a value* $\bar{\chi}$ *obeying* $\chi \leq \bar{\chi} \leq \lceil (1+\epsilon)\chi \rceil$ *which can be found in polynomial space and time* $O(1.2209^n + 2.2461^{e^{-\epsilon}n})$.

Proof. Fix some $\epsilon > 0$. We will perform the following operation a number of times:

Find the largest independent set and remove it from the graph. Repeat until the graph has at most $e^{-\epsilon}n$ vertices. Let s be the number of thus removed independent sets. We run the exact algorithm in Proposition 7 for the resulting graph to find its chromatic number χ_0. Our approximation is $\bar{\chi} = \chi_0 + s$.

We need to argue $\bar{\chi}$ is not far from the actual chromatic number. First note that $\bar{\chi} \geq \chi$ since the subgraph obtained after removing an independent set has chromatic number at least $\chi - 1$. Second, $\chi_0 \leq \chi$ since a subgraph cannot have larger chromatic number than its host graph. We note that $s \leq t$ for every integer t obeying

$$(1 - 1/\chi)^t \leq e^{-\epsilon}$$

since every graph with chromatic number χ has an independent set consisting of at least a fraction $1/\chi$ of its vertex set. Furthermore, $(1 - 1/\chi)^t \leq e^{-t/\chi}$ and thus $s \leq \lceil \epsilon\chi \rceil$.

Turning to the running time, we note that the fastest known polynomial space algorithm finding a largest independent set in a graph runs in time $O(1.2209^n)$ [19]. □

The above approximation idea translates to the general case of finding a minimal covering provided $\mathcal{F}$ has the following properties:

1. there is a fast algorithm to find the largest $S \in \mathcal{F}$.
2. $\mathcal{F}$ is hereditary, that is $S \subset T \in \mathcal{F}$ implies $S \in \mathcal{F}$.

An example of an interesting family of sets that is *not* hereditary is given by the induced trees of a graph. On the other hand, the induced forests *are* a hereditary family. In fact, some recent algorithms [38, 18] find a maximum induced forest in time $O(1.7548^n)$, satisfying also the first requirement. Thus our constructions give a good approximation algorithm for finding a small partition into induced forests.

7. Acknowledgments. The first two authors are indebted to Bolette A. Madsen [36] for making them think about these problems. M.K. is grateful to Heikki Mannila for valuable conversations on this work. Reference [30] was unearthed by Ryan Williams. This work was supported in part by the Academy of Finland, Grant 109101 (M.K.).

REFERENCES

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

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

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

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

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

[6] 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.

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

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

[9] 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.

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

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

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

[13] 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.

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

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

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

[17] 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.

[18] 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.

[19] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: A simple $O(2^{0.288n})$ Independent Set algorithm. In *Proc. 17th SODA*, pages 18–25, 2005.
[20] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov. Combinatorial bounds via measure and conquer: Boundings minimal dominating sets and applications. In *Proc. 16th ISAAC*, Springer LNCS volume 4288, pages 573–582, 2006.
[21] M. Fürer. Faster integer multiplication. In *Proc. 39th STOC*, ACM Press, pages 57–66, 2007.
[22] 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.
[23] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.
[24] C. Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Computational Complexity*, 9:52–73, 2000.
[25] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19–23, 1993.
[26] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *SIAM J. Appl. Math.*, 10:196–210, 1962.
[27] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
[28] R. Kennes. Computational aspects of the Moebius transform of a graph. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:201–223, 1991.
[29] R. M. Karp. Dynamic programming meets the principle of inclusion-exclusion. *Oper. Res. Lett.*, 1:49–51, 1982.
[30] 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.
[31] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. 3rd ed., Addison–Wesley, 1997.
[32] M. Koivisto. *Sum–Product Algorithms for the Analysis of Genetic Risks*. Ph.D. Thesis, University of Helsinki, January 2004.
[33] M. Koivisto. Optimal 2-constraint satisfaction via sum–product algorithms. *Information Processing Letters*, 98:24–28, 2006.
[34] M. Koivisto and K. Sood. Computational aspects of Bayesian partition models. In *International Conference on Machine Learning (ICML 2005)*, pages 433–440, 2005.
[35] E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.
[36] B. A. Madsen. An algorithm for exact satisfiability analysed with the number of clauses as parameter. *Information Processing Letters*, 97(1):28–30, 2006.
[37] 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.
[38] I. Razgon. Exact computation of maximum induced forest. In *Proc. 10th SWAT*, Springer LNCS volume 4059, pages 160–171, 2006.
[39] 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.
[40] 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.
[41] G.-C. Rota. On the foundations of combinatorial theory. I. Theory of Möbius functions. *Z. Wahrscheinlichkeitstheorie und Verw. Gebiete*, 2:340–368, 1964.
[42] H. J. Ryser. *Combinatorial Mathematics*. Number 14 in Carus Math. Monographs. Math. Assoc. America, 1963.
[43] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
[44] R. P. Stanley. Acyclic orientations of graphs. *Disc. Math.* 5:171-178, 1973.
[45] S. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.
[46] H. S. Wilf. *Algorithms and complexity*. Prentice–Hall, 1986.
[47] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348:257–265, 2005.
[48] 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.
[49] 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.

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

[51] 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.