Homomorphic Hashing for Sparse Coefficient Extraction

Petteri Kaski[1], Mikko Koivisto[2], and Jesper Nederlof[3]

[1] Helsinki Institute for Information Technology HIIT,
Department of Information and Computer Science, Aalto University, Finland
petteri.kaski@aalto.fi
[2] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Finland
mkhkoivi@cs.helsinki.fi
[3] Utrecht University, Utrecht, The Netherlands
j.nederlof@uu.nl

Abstract. We study classes of Dynamic Programming (DP) algorithms which, due to their algebraic definitions, are closely related to coefficient extraction methods. DP algorithms can easily be modified to exploit sparseness in the DP table through memorization. Coefficient extraction techniques on the other hand are both space-efficient and parallelisable, but no tools have been available to exploit sparseness. We investigate the systematic use of homomorphic hash functions to combine the best of these methods and obtain improved space-efficient algorithms for problems including LINEAR SAT, SET PARTITION and SUBSET SUM. Our algorithms run in time proportional to the number of nonzero entries of the last segment of the DP table, which presents a strict improvement over sparse DP. The last property also gives an improved algorithm for CNF SAT and SET COVER with sparse projections.

1 Introduction

Coefficient extraction can be seen as a general method for designing algorithms, recently in particular in the area of exact algorithms for various NP-hard problems [2,3,13,15,17,24] (cf. [7,26] for an introduction to exact algorithms). The approach of the method is the following (see also [14]):

1. Define a variable (the so-called coefficient) whose value (almost) immediately gives the solution of the problem to be solved,
2. Show that the variable can be expressed by a relatively small formula or circuit over a (cleverly chosen) large algebraic object like a ring or field,
3. Show how to perform operations in the algebraic object relatively efficiently.

In a typical application of the method, the first two steps are derived from an existing Dynamic Programming (DP) algorithm, and the third step deploys a carefully selected algebraic isomorphism, such as the discrete Fourier transform

D.M. Thilikos and G.J. Woeginger (Eds.): IPEC 2012, LNCS 7535, pp. 147–158, 2012.

to extract the desired solution/coefficient. Algorithms based on coefficient extraction have two key advantages over DP algorithms; namely, they are space-efficient and they parallelise well (see, for example, [15]).

Yet, DP has an advantage if the problem instance is *sparse*. By this we mean that the number of candidate/partial solutions that need to be considered during DP is small, that is, most entries in the DP table are not used at all. In such a case we can readily adjust the DP algorithm to take this into account through *memorization* so that both the running time and space usage become proportional to the number of partial solutions considered. Unfortunately, it is difficult to parallelise or lower the space usage of memorization. Coefficient extraction algorithms relying on interpolation of sparse polynomials [16] improve over memorization by scaling proportionally only to the number of *candidate* solutions, but their space usage is still not satisfactory (see also [26]).

This paper aims at obtaining what is essentially the best of both worlds, by investigating the systematic use of homomorphisms to "hash down" circuit-based coefficient extraction algorithms so that the domain of coefficient extraction – and hence the running time – matches or improves that of memorization-based DP algorithms, while providing space-efficiency and efficient parallelisation. The key idea is to take an existing algebraic circuit for coefficient extraction (over a sparsely populated algebraic domain such as a ring or field), and transform the circuit into a circuit over a smaller domain by a homomorphic hash function, and only then perform the actual coefficient extraction. Because the function is homomorphic, by hashing the values at the input gates and evaluating the circuit, the output evaluates to the hash of the original output value. Because the function is a hash function, the coefficient to be extracted collides with other coefficients only with negligible probability in the smaller domain, and coefficient extraction can be successfully used on the new (hashed-down) circuit. We call this approach *homomorphic hashing*.

Our and Previous Results

We study sparse DP/coefficient extraction in three domains: (a) the univariate polynomial ring $\mathbb{Z}[x]$ in Section 3, (b) the group algebra $\mathbb{F}[\mathbb{Z}_2^n]$ where $\mathbb{F}$ is a field of odd characteristic in Section 4 and (c) the Möbius algebra of the subset lattice in Section 5. The subject of sparse DP or coefficient extraction is highly motivated and well-studied [5,6,16,27]. In [16], a sparse polynomial interpolation algorithm using exponential space was already given for (a) and (b); our algorithms improve these to polynomial space. In [13] a polynomial-space algorithm for finding a small multilinear monomial in $\mathbb{F}_2[\mathbb{Z}_2^n]$ was given. In [15] a study of settings (a) and (b) was initiated, but sparsity was not addressed. Our main technical contribution occurs with (c) and hashing down to the "Solomon algebra" of a poset.

Our methods work for general arithmetic circuits similarly as in [13,15,16], and most of our algorithms work for counting variants as well. But, for concreteness, we will work here with specific decision problems. Although we mainly give improvements for sparse variants of these problems, we feel the results will be useful to deal with the general case as well (as we will see in Section 4).

Subset Sum. The Subset Sum problem is the following: given a vector $\boldsymbol{a} = (a_1, \ldots, a_n)$ and integer t, determine whether there exists a subset $X \subseteq [n]$ such that $\sum_{e \in X} a_e = t$. It is known to be solvable $\mathcal{O}^\star(2^{n/2})$ time and $\mathcal{O}^\star(2^{n/4})$ space [11,21], and solving it faster, or even in $\mathcal{O}^\star(1.99^n)$ time and polynomial space are interesting open questions [26]. Recently, a polynomial space algorithm using $\mathcal{O}^\star(t)$ time was given in [15]. Standard sparse DP gives an $\mathcal{O}^\star(S)$ time and $\mathcal{O}^\star(S)$ space algorithm. As a first "warm-up" application of our technique, we improve this to polynomial space as follows. The proofs of claims marked with a "†" are relegated to the full version in order to meet the page limit.

Theorem 1 (†). *Any instance $(\boldsymbol{a}, t)$ of the* Subset Sum *problem can be solved (a) in $\mathcal{O}^\star(S)$ expected time and polynomial space, and (b) in $\mathcal{O}^\star(S^2)$ time and polynomial space, where $S = |\{\sum_{e \in X} a_e : X \subseteq [n]\}|$ is the number of distinct sums.*

Informally stated, our algorithms hash the instances by working modulo randomly chosen prime numbers and apply the algorithm of [15]. While interesting on their own, these results may be useful in resolving the above open questions when combined with other techniques.

Linear Sat. The Linear Sat problem is defined as follows: given a matrix $\boldsymbol{A} \in \mathbb{Z}_2^{n \times m}$, vectors $\boldsymbol{b} \in \mathbb{Z}_2^m$ and $\boldsymbol{\omega} \in \mathbb{N}^n$, and an integer $t = n^{\mathcal{O}(1)}$, determine whether there is a vector $\boldsymbol{x} \in \mathbb{Z}_2^n$ such that $\boldsymbol{x}\boldsymbol{A} = \boldsymbol{b}$ and $\boldsymbol{\omega}\boldsymbol{x}^T \leq t$. Variants of Linear Sat have been studied, perhaps most notably in [10], where approximability was studied; Fixed Parameter Tractability was studied in [1,4]. Here, it was also quoted from [10] that (a variant of) Linear Sat is "as basic as satisfiability".

It can be observed that using the approach from [11], Linear Sat can be solved in $\mathcal{O}(2^{n/2}m)$ time and $\mathcal{O}(2^{n/2}m)$ space. Also, using standard "sparse dynamic programming", it can be solved in $\mathcal{O}^\star(2^{\mathrm{rk}(\boldsymbol{A})})$ time and $\mathcal{O}^\star(2^{\mathrm{rk}(\boldsymbol{A})})$ space, where $\mathrm{rk}(\boldsymbol{A})$ is the rank of $\boldsymbol{A}$. We give algorithms using about the same amount of time but only polynomial space:

Theorem 2. *Every instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{\omega}, t)$ of* Linear Sat *can be solved by algorithms with constant one-sided error probability in (a) $\mathcal{O}^\star(2^{\mathrm{rk}(\boldsymbol{A})})$ time and polynomial space, and (b) $\mathcal{O}^\star(2^{n/2})$ time and polynomial space.*

The first algorithm hashes the input down using a random linear map and afterwards determines the answer using the Walsh-Hadamard transform. The second algorithm uses a Win/Win approach, combining the first algorithm with the fact that an $\boldsymbol{A}$ with high rank can be solved with a complementary algorithm.

Satisfiability. The CNF-Sat problem is defined as follows: given a CNF-formula $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ over n variables, determine whether ϕ is satisfiable. There are many interesting open questions related to this problem, a major one being whether it can be solved in time $\mathcal{O}^\star((2-\epsilon)^n)$ (the 'Strong Exponential Time Hypothesis' [12] states this is not possible), and another being whether satisfying assignments can be counted in time $\mathcal{O}^\star((2-\epsilon)^n)$ for some $\epsilon > 0$ (e.g. [23]).

A *prefix assignment* is an assignment of 0/1 values to the variables $v_1, \ldots, v_i$ for some $1 \leq i \leq n$. A *projection* (*prefix projection*) of a CNF-formula is a subset $\pi \subseteq [m]$ such that there exists an assignment (prefix assignment) of the variables such that for every $1 \leq j \leq m$ it satisfies C_j if and only if $j \in \pi$. An algorithm for CNF-Sat running in time linear in the number of *prefix* projections can be obtained by standard sparse DP. However, it is sensible to ask about complexity of CNF-Sat if the number of projections is small. We give a positive answer:

Theorem 3. *Satisfiability of a formula $\phi = C_1 \wedge \ldots \wedge C_m$ can be determined in $\mathcal{O}^\star(P^2)$ time and $\mathcal{O}^\star(P)$ space, where $P = |\{\pi \subseteq [m] : \pi$ is a projection of $\phi\}|$.*

We are not aware of previous work that studies instances with few projections, but find it a natural parameter. For example, it is easy to see that hitting formulas[1] have only m (and hence the minimum number of) projections, and that formulas having a strong backdoor set[2] of size k have at most $2^k m$ projections. The formula with 2^n (and hence the maximum number of) projections is the one with a singleton clause for every variable. Naturally, there are more interesting cases and upper bounds for special classes of formula's, but to not lose focus from our main contribution we shall not discuss more structural properties of projections.

Underlying Theorem 3 is our main technical contribution (Theorem 15) that enables us to circumvent partial projections and access projections directly, namely homomorphic hashing from the Möbius algebra of the lattice of subsets of $[m]$ to the Solomon algebra of a poset. We think our result opens up a fresh technical perspective that may contribute towards solving the above mentioned and related questions. A full proof of Theorem 15 is given in the full version; we give a specialized, more direct proof of Theorem 3 and another application to Set Cover in Section 5.

2 Notation and Preliminaries

Lower-case boldface characters refer to vectors, while capital boldface letters refer to matrices, $\boldsymbol{I}$ being the identity matrix. The rank of a matrix $\boldsymbol{A}$ is denoted by $\mathrm{rk}(\boldsymbol{A})$. If R and S are sets, and S is finite, denote by R^S the set of all $|S|$-dimensional vectors with values in R, indexed by elements of S, that is, if $\boldsymbol{v} \in R^S$, then for every $e \in S$ we have $v_e \in R$. We denote by $\mathbb{Z}$ and $\mathbb{N}$ the set of integers and non-negative integers, respectively, and by $\mathbb{Z}_p$ the field of integers modulo a prime p. An arbitrary field is denoted by $\mathbb{F}$.

For a logical proposition P, we use Iverson's bracket notation $[P]$ to denote a 1 if P is true and a 0 if P is false. For a function $h : A \to B$ and $b \in B$, the preimage $h^{-1}(b)$ is defined as the set $\{a \in A : h(a) = b\}$. For an integer n and $A \subseteq \{1, \ldots, n\}$, denote by $\boldsymbol{\chi}(A) \in \mathbb{Z}_2^n$ the characteristic vector of A. Sometimes

[1] Every pair of clauses have a conflicting literal [18], also called "semi-complete" [1].

[2] k variables such that each assignment of them leaves a hitting formula (from [25], see also e.g. [8]).

we will state running times of algorithms with the $\mathcal{O}^\star$ notation, which suppresses any factor polynomial in the input size.

For a ring R and a finite set S, we write R^S for the ring consisting of the set R^S (the set of all vectors over R with coordinates indexed by elements of S) equipped with coordinate-wise addition $+$ and multiplication $\circ$ (the *Hadamard product*), that is, for $\boldsymbol{a}, \boldsymbol{b} \in R^S$ and $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{c}$, $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{d}$ we set $a_z + b_z = c_z$ and $a_z b_z = d_z$ for each $z \in S$, where $+$ and the juxtaposition denote addition and multiplication in R, respectively. The inner-product $\boldsymbol{a}, \boldsymbol{b} \in R^S$ is denoted by $\boldsymbol{a}^T \cdot \boldsymbol{b}$. For $\boldsymbol{v} \in R^S$ denote by $\mathrm{supp}(\boldsymbol{v}) \subseteq S$ the *support* of $\boldsymbol{v}$, that is, $\mathrm{supp}(\boldsymbol{v}) = \{z \in S : v_z \neq 0\}$, where 0 is the additive identity element of R. A vector $\boldsymbol{v}$ is called a *singleton* if $|\mathrm{supp}(\boldsymbol{v})| = 1$. We denote by $\langle z \to w \rangle$ the singleton with value w on index z, that is, $\langle z \to w \rangle_y = w[y = z]$ for all $y \in S$.

If R is a ring and $(S, \cdot)$ is a finite semigroup, denote by $R[S]$ the ring consisting of the set R^S equipped with coordinate-wise addition and multiplication defined by the convolution operator $*$, where for $\boldsymbol{a}, \boldsymbol{b} \in R^S$, $\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{c}$ we set $c_z = \sum_{x \cdot y = z} a_x b_y$ for every $z \in S$.

If R, S are rings with operations $(+, *)$ and $(\oplus, \circledast)$ respectively, a *homomorphism from R to S* is a function $h : R \to S$ such that $h(e_1 + e_2) = h(e_1) \oplus h(e_2)$ and $h(e_1 * e_2) = h(e_1) \circledast h(e_2)$ for every $e_1, e_2 \in R$.

Observation 4. *Let R be a ring, and let $(S, \cdot)$ and $(T, \odot)$ be finite semigroups. Suppose $\varphi : S \to T$ such that for every $x, y \in S$ we have $\varphi(x \cdot y) = \varphi(x) \odot \varphi(y)$. Then the function $h : R[S] \to R[T]$ defined by $h : \boldsymbol{a} \mapsto \boldsymbol{b}$ where $b_z = \sum_{y \in \varphi^{-1}(z)} a_y$ for all $z \in T$ is a homomorphism.*

A *circuit* C over a ring R is a labeled directed acyclic graph $D = (V, A)$ where the elements of V are called *gates* and D has a unique sink called the *output gate of C*. All sources of C are called *input gates* and are labeled with elements from R. All gates with non-zero in-degree are labeled as either an *addition* or a *multiplication gate*. (If multiplication in R is not commutative, the in-arcs of each multiplication gate are also ordered.) Every gate g of C can be associated with a ring element in the following natural way: If g is an input gate, we associate the label of g with g. If g is an addition gate we associate the ring element $e_1 + \ldots + e_d$ with g, and if g is a multiplication gate we associate the ring element $e_1 * \ldots * e_d$ with g where $e_1, \ldots, e_d$ are the ring elements associated with the d in-neighbors of g, and $+$ and $*$ are the operations of the ring R.

Suppose the ground set of R is of the type A^B where A, B are sets. Then C is said to have *singleton inputs* if the label of every input-gate of C is a singleton vector of R.

Definition 5. *Let R^1 and R^2 be rings, let $h : R^1 \to R^2$ be a homomorphism, and suppose that C is a circuit over R. Then, the circuit $h(C)$ over R^2* obtained by applying h to C *is defined as the circuit obtained from C by replacing for every input gate the label l by $h(l)$.*

Note that the following is immediate from the definition of a homomorphism:

Observation 6. *Suppose C is a circuit over a ring R^1 with output $v \in R^1$. Then the circuit over R^2 obtained by applying a homomorphism $h : R^1 \to R^2$ to C outputs $h(v) \in R^2$.*

3 Homomorphic Hashing for Subset Sum

In this section we will study the Subset Sum problem and prove Theorem 1. As mentioned in the introduction, it should be noted that this merely serves as an illustration of how similar problems can be tackled as well since the same method applies to the more general sparse polynomial interpolation problem. However, to avoid a repeat of the analysis of [15], we have chosen to restrict ourselves to the Subset Sum problem. Our central contribution over [15] is that we take advantage of sparsity. Given an integer $p \in \mathbb{N}$, let $c^p : \mathbb{N}^n \to \mathbb{N}^p$ be defined by

$$c^p(\boldsymbol{a})_j = \left| \left\{ X \subseteq [n] : \sum_{e \in X} a_e \equiv j \pmod p \right\} \right| \text{ for every } j \in \mathbb{Z}_p \text{ and } \boldsymbol{a} \in \mathbb{N}^n.$$

We also use the shorthand $\boldsymbol{c}(\boldsymbol{a}) = \boldsymbol{c}^\infty(\boldsymbol{a})$. We use the following corollary from [15] and two results on primes:

Corollary 7 (†, [15]). *Given an instance $(\boldsymbol{a}, t)$ of* Subset Sum *and an integer p, $c^p(\boldsymbol{a})_t$ can be computed in $\mathcal{O}^\star(p)$ time and $\mathcal{O}^\star(1)$ space.*

Theorem 8 ([20]). *If $55 < u$, the number of primes at most u is at least $\frac{u}{\ln u + 2}$.*

Lemma 9 (†, Folklore). *There exists an algorithm* `pickprime`(u) *running in* polylog(u) *time that, given integer $u \geq 2$ as input, outputs either a prime chosen uniformly at random from the set of primes at most u or* `notfound`*. Moreover, the probability that the output is* `notfound` *is at most $\frac{1}{e}$.*

We will run a data reduction procedure similar to the one of Claim 2.7 in [9], before applying the algorithm of Corollary 7. The idea of the data reduction procedure is to work modulo a prime of size roughly $|\mathrm{supp}(c(\boldsymbol{a}))|$ or larger:

Lemma 10. *Let $S \geq |\mathrm{supp}(c(\boldsymbol{a}))|$ and let β be an upper bound on the number of bits needed to represent the integers, i.e. $2^\beta > \max\{t, \max_i a_i\}$. Then for sufficiently large β and n, $\mathrm{Prob}_p[c(\boldsymbol{a})_t = c^p(\boldsymbol{a})_t] \geq \frac{1}{2}$, where the probability is taken uniformly over all primes $p \leq S\beta n(\log \beta)(\log n)$.*

Proof. Suppose $c(\boldsymbol{a})_t \neq c^p(\boldsymbol{a})_t$. Then there exists an integer $u \in \mathrm{supp}(c(\boldsymbol{a}))$ such that $u \neq t$ and $u \equiv t \pmod p$. This implies that p is a divisor of $|t - u|$, so let us bound the probability of this event. Since $|t - u| \leq 2^\beta n$, it has at most $\beta + \log n$ distinct prime divisors. Let $\gamma = S\beta n(\log \beta)(\log n)$. By Theorem 8 we have that $\mathrm{Prob}_p\big[p \text{ divides } |t - u|\big]$ is at most

$$\frac{\beta + \log n}{\frac{\gamma}{\log \gamma + 2}} \leq \frac{\beta + \log n}{\frac{\gamma}{3(n + \log \beta)}} \leq \frac{3(n + \log \beta)(\beta + \log n)}{\gamma} \leq \frac{1}{2S}$$

for sufficiently high β and n, where we use that $S \leq 2^n$ in the second inequality. Applying the union bound over the at most S elements of $\mathrm{supp}(c(\boldsymbol{a}))$, the event that there exists a $u \in \mathrm{supp}(c(\boldsymbol{a}))$ with $u \neq t$ and $u \equiv t \pmod{p}$ occurs with probability at most $\frac{1}{2}$. □

Now we give an algorithm for the case where S is known. The proof of Theorem 1, given in the full version, merely adds self-reduction arguments.

Theorem 11. *There exists an algorithm that, given an instance $(\boldsymbol{a}, t)$ of the* Subset Sum *problem and an integer $S \geq |\mathrm{supp}(c(\boldsymbol{a}))|$ as input, outputs a non-negative integer x in $\mathcal{O}^\star(S)$ time and polynomial space such that (i) $x = 0$ implies $c(\boldsymbol{a})_t = 0$ and (ii) $\mathrm{Prob}[c(\boldsymbol{a})_t = x] \geq \frac{1}{4}$.*

Proof. The algorithm is: First, obtain prime $p = \texttt{pickprime}(S\beta n(\log \beta)(\log n))$ using Lemma 9. Second, compute and output $c^p(\boldsymbol{a})_t$ using Corollary 7. Condition (i) holds since $c^p(\boldsymbol{a})_t = 0$ implies $c(\boldsymbol{a})_t = 0$ for any p, t. Moreover, condition (ii) follows from Lemma 10 and Lemma 9 since $\frac{1}{2}(1 - \frac{1}{e}) \geq \frac{1}{4}$. The time and space bounds are met by Corollary 7 because $p = \mathcal{O}^\star(S)$. □

4 Homomorphic Hashing for Linear Satisfiability

In this section we assume that $\mathbb{F}$ is a field of non-even characteristic and that addition and multiplication refer to operations in $\mathbb{F}$. We prove the following general result, having Theorem 2(a) as a special case.

Theorem 12. *There exists a randomized algorithm that, given as input (i) a circuit C with singleton inputs over $\mathbb{F}[\mathbb{Z}_2^n]$, (ii) an integer $S \geq |\mathrm{supp}(\boldsymbol{v})|$, and (iii) an element $\boldsymbol{t} \in \mathbb{Z}_2^n$, outputs the coefficient $v_{\boldsymbol{t}} \in \mathbb{F}$ with probability at least $\frac{1}{2}$, where $\boldsymbol{v} \in \mathbb{F}[\mathbb{Z}_2^n]$ is the output of C. The algorithm uses $\mathcal{O}^\star(S)$ time, $\mathcal{O}^\star(S)$ arithmetic operations in $\mathbb{F}$, and storage for $\mathcal{O}^\star(1)$ bits and elements of $\mathbb{F}$.*

Proof. Consider Algorithm 1. Let us first analyse the complexity of this algorithm: Steps 1 and 2 can be performed in time polynomial in the input. Step 3 also be done in polynomial time since it amounts to relabeling all input gates with $h(\boldsymbol{e})$ where $\boldsymbol{e}$ was the old label. Indeed, we know that $\boldsymbol{e} \in \mathbb{F}[\mathbb{Z}_2^n]$ is a singleton $\langle \boldsymbol{y} \to v \rangle$, so $h(\boldsymbol{e})$ is the singleton $\langle \boldsymbol{y}\boldsymbol{H} \to v \rangle$ and this can be computed in polynomial time. Step 4 takes $\mathcal{O}^\star(S)$ operations and calls to `sub`, so for the complexity bound it remains to show that a call to `sub` runs in polynomial time. Step 5 can be implemented in polynomial time similar to Step 3 since the singleton $\boldsymbol{e} = \langle \boldsymbol{y} \to v \rangle$ is mapped to $(-1)^{\boldsymbol{x}\boldsymbol{y}^T} v$. Finally, the direct evaluation of C_2 uses $|C_2|$ operations in $\mathbb{F}$. Hence the algorithm meets the time bound, and also the space bound is immediate. The fact that `hashZ2` returns $v_{\boldsymbol{t}}$ with probability at least $\frac{1}{2}$ is a direct consequence of the following two claims, where $\boldsymbol{w}$ denotes the output of C_1.

Claim 1 (†). $\mathrm{Prob}_{\boldsymbol{H}}[v_{\boldsymbol{t}} = w_{\boldsymbol{t}\boldsymbol{H}}] \geq \frac{1}{2}$.

Claim 2 (†). *Algorithm* `hashZ2` *returns $w_{\boldsymbol{t}\boldsymbol{H}}$.*

□

Algorithm `hashZ2`
1: Let $s = \lceil \log S \rceil + 1$.
2: Choose a matrix $\boldsymbol{H} \in \mathbb{Z}_2^{s \times n}$ uniformly at random from the set of all $s \times n$ matrices with binary entries.
3: Let $h : \mathbb{F}[\mathbb{Z}_2^n] \to \mathbb{F}[\mathbb{Z}_2^s]$ be the homomorphism defined by $h(\boldsymbol{a}) = \boldsymbol{b}$ where $b_{\boldsymbol{x}} = \sum_{\boldsymbol{y} \in \mathbb{Z}_2^n : \boldsymbol{y}\boldsymbol{H} = \boldsymbol{x}} a_{\boldsymbol{y}}$ for all $\boldsymbol{x} \in \mathbb{Z}_2^s$. Apply h to C to obtain the circuit C_1.
4: **return** $\frac{1}{2^s} \sum_{\boldsymbol{x} \in \mathbb{Z}_2^s} (-1)^{(\boldsymbol{t}\boldsymbol{H})\boldsymbol{x}^T}$ `sub`$(C_1, \boldsymbol{x})$.

Algorithm `sub`$(C_1, \boldsymbol{x})$
5: Let $\varphi : \mathbb{F}[\mathbb{Z}_2^s] \to \mathbb{F}$ be the homomorphism defined by $\varphi(\boldsymbol{w}) = \sum_{\boldsymbol{y} \in \mathbb{Z}_2^s} (-1)^{\boldsymbol{x}\boldsymbol{y}^T} w_{\boldsymbol{y}}$ for all $\boldsymbol{w} \in \mathbb{F}[\mathbb{Z}_2^s]$. Apply φ to C_1 to obtain the circuit C_2.
6: Evaluate C_2 and return the output.

Algorithm 1: Homomorphic hashing for Theorem 12

Proof (of Theorem 2(a)). For $1 \leq i \leq n$ and $0 \leq w \leq t$ denote by $\boldsymbol{A}^{(i)}$ the ith row of $\boldsymbol{A}$ and define $\boldsymbol{f}[i, w] \in \mathbb{Q}[\mathbb{Z}_2^m]$ by

$$\boldsymbol{f}[i,w] = \begin{cases} \langle \boldsymbol{0} \to 1 \rangle & \text{if } i = w = 0, \\ 0 & \text{if } i = 0 \wedge w \neq 0, \\ \boldsymbol{f}[i-1,w] + \boldsymbol{f}[i-1, w-\omega_i] * \left\langle \boldsymbol{A}^{(i)} \to 1 \right\rangle & \text{otherwise.} \end{cases} \quad (1)$$

It is easy to see that for every $1 \leq i \leq n$, $0 \leq w \leq t$, and $\boldsymbol{y} \in \mathbb{Z}_2^m$, the value $f[i,w]_{\boldsymbol{y}}$ is the number of $\boldsymbol{x} \in \mathbb{Z}_2^i$ such that $\tilde{\boldsymbol{\omega}}\boldsymbol{x}^T = w$ and $\boldsymbol{x}\widetilde{\boldsymbol{A}} = \boldsymbol{y}$ where $\tilde{\boldsymbol{\omega}}$ and $\widetilde{\boldsymbol{A}}$ are obtained by truncating $\boldsymbol{\omega}$ and $\boldsymbol{A}$ to the first i rows. Hence, we let C be the circuit implementing (1) and let its output be $\boldsymbol{v} = \sum_{w=0}^{t} \boldsymbol{f}[n, w]$. Thus, $v_{\boldsymbol{b}}$ is the number of $\boldsymbol{x} \in \mathbb{Z}_2^n$ with $\boldsymbol{x}\boldsymbol{A} = \boldsymbol{b}$ and $\boldsymbol{x}\boldsymbol{\omega}^T \leq t$.

Also, $|\mathrm{supp}(\boldsymbol{v})| \leq 2^{\mathrm{rk}(\boldsymbol{A})}$ since any element of the support of $\boldsymbol{v}$ is a sum of rows of $\boldsymbol{A}$ and hence in the row-space of $\boldsymbol{A}$, which has size at most $2^{\mathrm{rk}(\boldsymbol{A})}$. To apply Theorem 12, let $\mathbb{F} = \mathbb{Q}$ and observe that the computations are in fact carried out over integers bounded in absolute value poly-exponentially in n and hence the operations in the base field can also be executed polynomial in n. The theorem follows from Theorem 12. □

To establish Theorem 2(b), let us first see how to exploit a high linear rank of the matrix $\boldsymbol{A}$ in an instance of Linear Sat. By permuting the rows of $\boldsymbol{A}$ as necessary, we can assume that the first $\mathrm{rk}(\boldsymbol{A})$ rows of $\boldsymbol{A}$ are linearly independent. We can now partition $\boldsymbol{x}$ into $\boldsymbol{x} = (\boldsymbol{y}, \boldsymbol{z})$, where $\boldsymbol{y}$ has length $\mathrm{rk}(\boldsymbol{A})$ and $\boldsymbol{z}$ has length $n - \mathrm{rk}(\boldsymbol{A})$. There are $2^{n-\mathrm{rk}(\boldsymbol{A})}$ choices for $\boldsymbol{z}$, each of which by linear independence has at most one corresponding $\boldsymbol{y}$ such that $\boldsymbol{x}\boldsymbol{A} = \boldsymbol{b}$. Given $\boldsymbol{z}$, we can determine the corresponding $\boldsymbol{y}$ (if any) in polynomial time by Gaussian elimination. Thus, we have:

Observation 13. Linear Sat *can be solved in* $\mathcal{O}^\star(2^{n-\mathrm{rk}(\boldsymbol{A})})$ *time and polynomial space.*

This enables a "Win/Win approach" where we distinguish between low and high ranks, and use an appropriate algorithm in each case.

Proof (of Theorem 2(b)). Compute $\mathrm{rk}(\boldsymbol{A})$. If $\mathrm{rk}(\boldsymbol{A}) \geq n/2$, run the algorithm of Observation 13. Otherwise, run the algorithm implied by Theorem 2(a). □

Set Partition. We now give a very similar application to the Set Partition problem: given an integer t and a set family $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = n$, $|U| = m$, determine whether there is a subfamily $\mathcal{P} \subseteq \mathcal{F}$ with $|\mathcal{P}| \leq t$ such that $\bigcup_{S \in \mathcal{P}} S = U$ and $\sum_{S \in \mathcal{P}} |S| = |U|$.

The *incidence matrix* of a set system $(U, \mathcal{F})$ is the $|U| \times |\mathcal{F}|$ matrix $\boldsymbol{A}$ whose entries $A_{e,S} = [e \in S]$ are indexed by $e \in U$ and $S \in \mathcal{F}$.

Theorem 14 (†). *There exist algorithms that given an instance $(U, \mathcal{F}, t)$ of* Set Partition *output the number of set partitions of size at most t with probability at least $\frac{1}{2}$, and use (a) $\mathcal{O}^\star(2^{\mathrm{rk}(\boldsymbol{A})})$ time and polynomial space, and (b) $(2^{\mathrm{rk}(\boldsymbol{A})} + n)m^{\mathcal{O}(1)}$ time and space, where $\boldsymbol{A}$ is the incidence matrix of $(U, \mathcal{F})$.*

5 Homomorphic Hashing for the Union Product

In this section our objective is to mimic the approach of the previous section for $\mathbb{N}[(2^U, \cup)]$, where $(2^U, \cup)$ is the semigroup defined by the set union $\cup$ operation on 2^U, the power set of an n-element set U. The direct attempt to apply a homomorphic hashing function, unfortunately, fails. Indeed, let h be an arbitrary homomorphism from $(2^U, \cup)$ to $(2^V, \cup)$ with $|V| < |U|$. Let $U = \{e_1, e_2, \ldots, e_n\}$ and consider the minimum value $1 \leq j \leq n-1$ with $h(\{e_1, \ldots, e_j\}) = \cup_{i=1}^{j} h(\{e_i\}) = \cup_{i=1}^{j+1} h(\{e_i\}) = h(\{e_1, \ldots, e_{j+1}\})$; in particular, for $X = \{e_1, \ldots, e_j, e_{j+2}, \ldots, e_n\} \neq U$ we have $h(X) = h(U)$, which signals failure since we cannot isolate X from U.

Instead, we use hashing to an algebraic structure based on a poset (the "Solomon algebra" of a poset due to [22]) that is obtained by the technique "Iterative Compression". This gives the following main result. For reasons of space we relegate a detailed proof to the full version; here we will give a simplified version of the proof in the special case of Theorem 3 in this section.

Theorem 15 (†). *Let and $|U| = n$. There are algorithms that, given a circuit C with singleton inputs in $\mathbb{N}[(2^U, \cup)]$ outputting $\boldsymbol{v}$, compute*

(a) a list with v_X for every $X \in \mathrm{supp}(\boldsymbol{v})$ in $\mathcal{O}^\star(|\mathrm{supp}(\boldsymbol{v})|^2 n^{\mathcal{O}(1)})$ time,
(b) v_U in time $\mathcal{O}^\star(2^{(1-\alpha/2)n} n^{\mathcal{O}(1)})$ if $0 < \alpha \leq 1/2$ such that $|\mathrm{supp}(\boldsymbol{v})| \leq 2^{(1-\alpha)n}$.

The above result is stated for simplicity in the unit-cost model, that is, we assume that arithmetic operations on integers take constant time. For the more realistic log-cost model, where such operations are assumed to take time polynomial in the number of bits of the binary representation, we only mention here that our results also hold under some mild technical conditions. Let us first show that Theorem 3(a) indeed is a special case of Theorem 15:

Proof (of Theorem 3). Use the circuit over $\mathbb{N}[(2^{[m]}, \cup)]$ that implements

$$\boldsymbol{f} = (\langle V_1 \to 1\rangle + \langle \bar{V}_1 \to 1\rangle) * (\langle V_2 \to 1\rangle + \langle \bar{V}_2 \to 1\rangle) * \ldots * (\langle V_m \to 1\rangle + \langle \bar{V}_m \to 1\rangle),$$

where $V_i \subseteq [m]$ (respectively, $\bar{V}_i \subseteq [m]$) is the set of all indices of clauses that contain a positive (respectively, negative) literal of the variable v_i. Then use Theorem 15 to determine $f_{[m]}$, the number of satisfying assignments of ϕ. □

Now we proceed with a self-contained proof Theorem 3. Given poset $(P, \leq)$, the *Möbius function* $\mu : P \times P \to \mathbb{N}$ of P is defined for all $x, y \in P$ by

$$\mu(x,y) = \begin{cases} 1 & \text{if } x = y, \\ -\sum_{x \leq z < y} \mu(x,z) & \text{if } x < y, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The *zeta transform* $\boldsymbol{\zeta}$ and *Möbius transform* $\boldsymbol{\mu}$ are the $|P| \times |P|$ matrices defined by $\zeta_{x,y} = [x \leq y]$ and $\mu_{x,y} = \mu(x,y)$ for all $x, y \in P$. For a CNF-formula ϕ denote $\mathrm{supp}(\phi)$ for the set of all projections of ϕ. Recall in Theorem 3 we are given a CNF-Formula $\phi = C^1 \wedge \ldots \wedge C^m$ over n variables. For $i = 1, \ldots, m$ define $\phi_i = C_1 \wedge \ldots \wedge C_i$. Then we have the following easy observations:

1. $\mathrm{supp}(\phi_0) = \{\emptyset\}$,
2. $\mathrm{supp}(\phi_i) \subseteq \mathrm{supp}(\phi_{i-1}) \cup \{X \cup \{i\} : X \in \mathrm{supp}(\phi_{i-1})\}$ for every $i = 1, \ldots, m$,
3. $|\mathrm{supp}(\phi_{i-1})| \leq |\mathrm{supp}(\phi_i)|$ for every $i = 1, \ldots, m$.

Given the above lemma and observations, we will give an algoritm using a technique called *iterative compression* [19]. As we will see, by this technique it is sufficient to solve the following "compression problem":

Lemma 16. *Given a CNF-formula $\phi = C_1 \wedge \ldots \wedge C_m$ and a set family $\mathcal{F} \subseteq 2^{[m]}$ with $\mathrm{supp}(\phi) \subseteq \mathcal{F}$, the set $\mathrm{supp}(\phi)$ can be constructed in $\mathcal{O}^\star(|\mathcal{F}|^2)$ time.*

Proof. In what follows $\boldsymbol{a} \in \{0,1\}^n$ refers to an assignment of values to the n variables in ϕ. Define $\boldsymbol{f} \in \mathbb{N}^{2^{[m]}}$ for all $X \subseteq [m]$ by

$$f_X = |\{\boldsymbol{a} \in \{0,1\}^n : \forall i \in [m] \text{ it holds that } \boldsymbol{a} \text{ satisfies } C_i \text{ iff } i \in X\}|.$$

It is easy to see that $\mathrm{supp}(\boldsymbol{f}) = \mathrm{supp}(\phi)$, so if we know f_X for every $X \in \mathcal{F}$ we can construct $\mathrm{supp}(\phi)$ in $|\mathcal{F}|$ time. Towards this end, first note that for every $Y \subseteq [m]$, $(\boldsymbol{f}\boldsymbol{\zeta})_Y$ equals

$$\sum_{\substack{X \in \mathrm{supp}(\boldsymbol{f}) \\ X \subseteq Y}} f(X) = \sum_{X \subseteq Y} f(X) = |\{\boldsymbol{a} \in \{0,1\}^n \mid \forall i : \boldsymbol{a} \text{ satisfies } C_i \text{ only if } i \in Y\}|.$$

Second, note that the last quantity can be computed in polynomial time: since every clause outside Y must not be satisfied, each such clause forces the variables that occur in it to unique values; any other variables may be assigned to

arbitrary values. That is, the count is 0 if the clauses outside Y force at least one variable to conflicting values, otherwise the count is 2^a where a is the number of variables that occur in none of the clauses outside Y.

Now the algorithm is the following: for every $X \in \mathcal{F}$ compute $(\boldsymbol{f\zeta})_X$ in polynomial time as discussed above. Then we can use algorithm `mobius` as described below to obtain f_X for every $X \in \mathcal{F}$ since it follows that $\boldsymbol{f} = \texttt{mobius}((\mathcal{F}, \subseteq), \boldsymbol{f\zeta})$ from the definition of $\boldsymbol{\mu}$ and the fact that $\boldsymbol{\mu\zeta} = \boldsymbol{I}$. Algorithm `mobius` clearly runs in $\mathcal{O}^\star(|P|^2)$ time, so this procedure meets the claimed time bound.

Algorithm `mobius`$((P, \leq), \boldsymbol{w})$
1: Let $P = \{v_1, v_2 \ldots, v_{|P|}\}$ such that $v_j \leq v_i$ implies $j \leq i$.
2: $\boldsymbol{z} \leftarrow \boldsymbol{w}$.
3: **for** $i = 1, 2, \ldots, |P|$ **do**
4: **for** every $v_j \leq v_i$ **do**
5: $z_i = z_i - z_j$
6: **return** $\boldsymbol{z}$.

□

Proof (of Theorem 3, self-contained). Recall that we already know that $\mathrm{supp}(\phi_0) = \{\emptyset\}$. Now, for $i = 1, \ldots, m$ we set $\mathcal{F} = \mathrm{supp}(\phi_{i-1}) \cup \{X \cup \{i\} : X \in \mathrm{supp}(\phi_{i-1})\}$ and use $\mathcal{F}$ to obtain $\mathrm{supp}(\phi_i)$ using Lemma 16. In the end we are given $\mathrm{supp}(\phi_m)$ and since ϕ_m is exactly the original formula, the input is a yes-instance if and only if $[m] \in \mathrm{supp}(\phi_m)$. The claimed running time follows from Observations 1 and 3 above and the running time of algorithm `mobius`.

□

Set Cover. We will now give an application of Theorem 15(b) to Set Cover: Given a set family $\mathcal{F} \subseteq 2^U$ where $|U| = n$ and an integer k, find a subfamily $\mathcal{C} \subseteq \mathcal{F}$ such that $|\mathcal{C}| = k$ and $\bigcup_{S \in \mathcal{C}} S = U$.

Theorem 17 (†). *Given an instance of* Set Cover*, let* $0 < \alpha \leq 1/2$ *be the largest real such that* $|\{\bigcup_{S \in C} S : \mathcal{C} \subseteq \mathcal{F} \text{ and } |\mathcal{C}| = k\}| \leq 2^{(1-\alpha)n}$*. Then the instance can be solved in* $\mathcal{O}^\star(2^{(1-\alpha/2)n} n^{\mathcal{O}(1)})$ *time.*

Acknowledgements. P.K. is supported by the Academy of Finland, Grants 252083 and 256287. M.K. is supported by the Academy of Finland, Grant 125637. J.N. is supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), project: 'Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms'.

References

1. Alon, N., Gutin, G., Kim, E.J., Szeider, S., Yeo, A.: Solving MAX-r-SAT above a tight lower bound. Algorithmica 61(3), 638–655 (2011)
2. Björklund, A.: Determinant sums for undirected Hamiltonicity. In: FOCS, pp. 173–182. IEEE Computer Society (2010)
3. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. Comput. 39(2), 546–563 (2009)
4. Crowston, R., Gutin, G., Jones, M., Yeo, A.: Lower bound for Max-r-Lin2 and its applications in algorithmics and graph theory. CoRR, abs/1104.1135 (2011)
5. Eppstein, D., Galil, Z., Giancarlo, R., Italiano, G.F.: Sparse dynamic programming I: linear cost functions. J. ACM 39, 519–545 (1992)

6. Eppstein, D., Galil, Z., Giancarlo, R., Italiano, G.F.: Sparse dynamic programming II: convex and concave cost functions. J. ACM 39(3), 546–567 (1992)
7. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms, 1st edn. Springer-Verlag New York, Inc., New York (2010)
8. Gaspers, S., Szeider, S.: Strong backdoors to nested satisfiability. CoRR, abs/1202.4331 (2012)
9. Harnik, D., Naor, M.: On the compressibility of NP instances and cryptographic applications. SIAM Journal on Computing 39(5), 1667–1713 (2010)
10. Håstad, J.: Some optimal inapproximability results. J. ACM 48, 798–859 (2001)
11. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM 21, 277–292 (1974)
12. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. 62(2), 367–375 (2001)
13. Koutis, I., Williams, R.: Limits and Applications of Group Algebras for Parameterized Problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 653–664. Springer, Heidelberg (2009)
14. Lipton, R.J.: Beating Bellman for the knapsack problem (2010), `http://rjlipton.wordpress.com/2010/03/03/beating-bellman-for-the-knapsack-problem/`, `http://rjlipton.wordpress.com`
15. Lokshtanov, D., Nederlof, J.: Saving space by algebraization. In: Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, pp. 321–330. ACM, New York (2010)
16. Mansour, Y.: Randomized interpolation and approximation of sparse polynomials. SIAM J. Comput. 24(2), 357–368 (1995)
17. Nederlof, J.: Fast Polynomial-Space Algorithms Using Möbius Inversion: Improving on Steiner Tree and Related Problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 713–725. Springer, Heidelberg (2009)
18. Nishimura, N., Ragde, P., Szeider, S.: Solving #sat using vertex covers. Acta Inf. 44(7), 509–523 (2007)
19. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. 32(4), 299–301 (2004)
20. Rosser, B.: Explicit bounds for some functions of prime numbers. American Journal of Mathematics 63(1), 211–232 (1941)
21. Schroeppel, R., Shamir, A.: A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM J. Comput. 10(3), 456–464 (1981)
22. Solomon, L.: The burnside algebra of a finite group. Journal of Combinatorial Theory 2(4), 603–615 (1967)
23. Traxler, P.: Exponential Time Complexity of SAT and Related Problems. PhD thesis, ETH Zürich (2010)
24. Williams, R.: Finding paths of length k in $\mathcal{O}^\star(2^k)$ time. Inf. Process. Lett. 109(6), 315–318 (2009)
25. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: IJCAI, pp. 1173–1178. Morgan Kaufmann (2003)
26. Woeginger, G.J.: Open problems around exact algorithms. Discrete Applied Mathematics 156(3), 397–405 (2008)
27. Zippel, R.: Probabilistic Algorithms for Sparse Polynomials. In: Ng, E.W. (ed.) EUROSAM 1979 and ISSAC 1979. LNCS, vol. 72, pp. 216–226. Springer, Heidelberg (1979)