LINEAR AND BILINEAR TRANSFORMATIONS FOR MODERATELY EXPONENTIAL ALGORITHMS (LECTURE NOTES FOR AGAPE 09)

PETTERI KASKI

1. Introduction

Many basic tasks in algebra can be reduced to the problem of evaluating a set of linear or bilinear forms over a ring R (for example, over the integers).

In the linear case, the task is to evaluate

$$y_i = \sum_{j=1}^{n} a_{ij} x_j \quad \text{for } 1 \leq i \leq m, \tag{1}$$

where the a_{ij} are coefficients in R, and the x_j are elements of R given as input.

In the bilinear case, the task is to evaluate

$$z_i = \sum_{j=1}^{n} \sum_{k=1}^{s} b_{ijk} x_j y_k \quad \text{for } 1 \leq i \leq m, \tag{2}$$

where the b_{ijk} are coefficients in R, and the x_j and y_k are elements of R given as input.

Here the coefficients a_{ij} and b_{ijk} are understood to be static (that is, part of the problem definition) and not part of the formal input. Put otherwise, (1) is essentially the task of multiplying a given vector with an implicit $m \times n$ matrix, and (2) asks us to evaluate a bilinear product of given two vectors. For example, taking the average $\frac{1}{n} \sum_{j=1}^{n} x_j$ reduces to (1) with $m = 1$, and the inner product $\sum_{j=1}^{n} x_j y_j$ reduces (2) with $m = 1$ and $n = s$.

Exercise 1. *Express the task of multiplying two complex numbers, $x_1 + x_2\Im$ and $y_1 + y_2\Im$, as the task of evaluating a set of two bilinear forms.*

Exercise 2. *Express the task of multiplying two polynomials of degrees p and q, respectively, as the task of evaluating a set of $p + q + 1$ bilinear forms.*

Exercise 3. *Express the task of multiplying two matrices of sizes $p \times q$ and $q \times r$, respectively, as the task of evaluating a set of $p \cdot r$ bilinear forms.*

In this lecture we study algorithms based on "faster-than-obvious" evaluation strategies for specific linear and bilinear transformations. Our measure of efficiency in this setting is the number of basic arithmetic operations in the ring R.

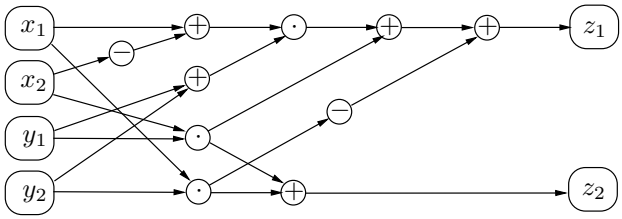
Exercise 4. *The obvious way to multiply two complex numbers requires four real multiplications:*

$$(x_1 + x_2\Im)(y_1 + y_2\Im) = x_1 y_1 - x_2 y_2 + (x_1 y_2 + y_1 x_2)\Im.$$

Show that three real multiplications suffice.

In many cases it is possible to view an evaluation strategy as an *R-arithmetic circuit* that transforms a given input into the desired output via a circuit of (i) arithmetic gates (addition, negation, multiplication) and (ii) constant gates taking values in R. We assume that all arithmetic gates have a fan-in of at most two.

Example 5. *A real arithmetic circuit for multiplying two complex numbers with three multiplications.*

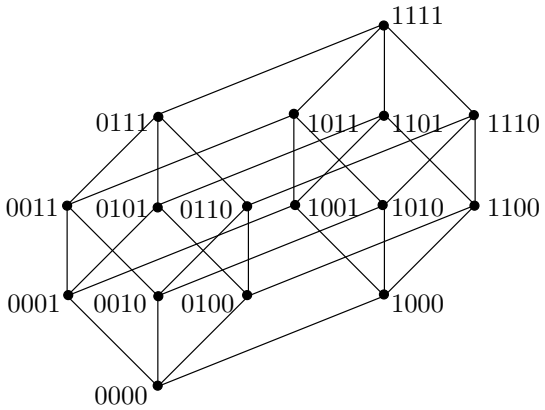


Note that multiplication in an arbitrary ring R need not be commutative, in which case one must explicitly indicate the left and right inputs to a multiplication gate.

2. Two examples of fast evaluation

2.1. **Yates's algorithm.** The n-dimensional binary hypercube (the n-cube) is the graph whose vertices are the binary strings $s = (s_1, s_2, \ldots, s_n) \in \{0,1\}^n$, and any two vertices are joined by an edge if and only if they differ in exactly one position.

Example 6. *The n-cube for $n = 4$.*



Consider the following task. Let $k : \{0,1\}^2 \to R$ and $x : \{0,1\}^n \to R$ be given as input. We must output the function $y : \{0,1\}^n \to R$, defined by

$$y(t) \stackrel{\text{def}}{=} \sum_{s \in \{0,1\}^n} \left(\prod_{i=1}^{n} k(t_i, s_i) \right) x(s) \quad \text{for } t \in \{0,1\}^n. \tag{3}$$

Observe that (3) in fact asks us to evaluate a set of 2^n linear forms, one for each vertex t of the n-cube. Thus, we are looking at an instance $y(t) = \sum_s a(t,s)x(s)$ of (1), where the coefficients $a(t,s) = \prod_{i=1}^{n} k(t_i, s_i)$ are determined by the auxiliary input k.

A direct evaluation of (3) takes $O(4^n n)$ ring operations.

To arrive at more efficient evaluation, one possibility is to view $\prod_{i=1}^{n} k(t_i, s_i)$ in (3) as the "weight" of a "walk" from s to t in the n-cube, where each step $s_i \mapsto t_i$ contributes the weight $k(t_i, s_i)$. In particular, we can view $y(t)$ as the weighted sum of "messages" $x(s)$ transmitted along walks to t. A direct evaluation of (3) corresponds to considering each individual walk separately, but it turns out that the walks can be processed in aggregate using dynamic programming.

Let us make more precise the notion of a "walk" in this context. Let s and t be any two vertices of the n-cube. The *walk* from s to t is a sequence of n steps, where step $i = 1, 2, \ldots, n$ consists of the rule $s_i \mapsto t_i$ applied to position i.

Example 7. *The walk from* $s = (0, 1, 1, 0)$ *to* $t = (1, 1, 0, 0)$ *is* $(0 \mapsto 1, 1 \mapsto 1, 1 \mapsto 0, 0 \mapsto 0)$. *The sequence of vertices visited by the walk appears below.*

$$\begin{array}{llllllll} 0 & (0 \mapsto 1) & 1 & & 1 & & 1 & & 1 \\ 1 & & 1 & (1 \mapsto 1) & 1 & & 1 & & 1 \\ 1 & & 1 & & 1 & (1 \mapsto 0) & 0 & & 0 \\ 0 & & 0 & & 0 & & 0 & (0 \mapsto 0) & 0 \end{array}$$

Observe that a walk uniquely determines the sequence of vertices it visits.

Exercise 8. *Conclude that for any vertex* $u \in \{0,1\}^n$ *there are exactly* 2^n *walks that are at* u *after exactly* j *steps,* $0 \le j \le n$.

The intuition in the following algorithm is that $z_j(u)$ contains an aggregate of the walks that are *at* u *after exactly* j *steps*. In particular, these walks originate from the vertices $(s_1, \ldots, s_j, u_{j+1}, \ldots, u_n)$ for $s_1, \ldots, s_j \in \{0,1\}$.

The algorithm proceeds in n rounds. First, for every $s \in \{0,1\}^n$, set

$$z_0(s) \stackrel{\text{def}}{=} x(s). \tag{4}$$

In round $j = 1, 2, \ldots, n$, for every $u \in \{0,1\}^n$, set

$$\begin{aligned} &z_j(u_1, \ldots, u_n) \stackrel{\text{def}}{=} \\ &\quad k(u_j, 0) \cdot z_{j-1}(u_1, \ldots, u_{j-1}, 0, u_{j+1}, \ldots, u_n) \\ &\quad + k(u_j, 1) \cdot z_{j-1}(u_1, \ldots, u_{j-1}, 1, u_{j+1}, \ldots, u_n). \end{aligned} \tag{5}$$

Lemma 9. *For all* $j = 0, 1, \ldots, n$ *and* $u \in \{0,1\}^n$ *it holds that*

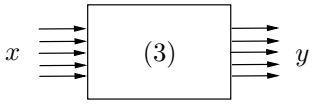
$$z_j(u_1, \ldots, u_n) = \sum_{s_1, \ldots, s_j \in \{0,1\}} \left(\prod_{i=1}^{j} k(u_i, s_i) \right) x(s_1, \ldots, s_j, u_{j+1}, \ldots, u_n)\,. \tag{6}$$

Proof. By induction on j. □

Exercise 10. *Give a full proof for Lemma 9.*

In particular, (3) and (6) imply that $z_n(t) = y(t)$ for all $t \in \{0,1\}^n$. Thus, *Yates's algorithm* given by (4) and (5) evaluates (3) in $O(2^n n)$ ring operations.

Observe that we may think of (4) and (5) as a specification of an arithmetic circuit with inputs for x and outputs for $y = z_n$.



Exercise 11. *Draw Yates's algorithm as an arithmetic circuit when $n = 3$ and $k(0,0) = 1$, $k(1,0) = 1$, $k(0,1) = 0$, $k(1,1) = 1$.*

Exercise 12. *Let A and B be $m \times n$ and $p \times q$ matrices, respectively. The Kronecker product $A \otimes B$ is the $mp \times nq$ matrix defined by*

$$A \otimes B \stackrel{\text{def}}{=} \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}.$$

Express (3) *as a matrix–vector product, where the matrix is obtained via Kronecker products. Using this representation, design a recursive version of Yates's algorithm.*

Exercise 13. *Extend Yates's algorithm to evaluate sums of the form*

$$y(t_1, \ldots, t_n) = \sum_{s_1=0}^{m_1-1} \sum_{s_2=0}^{m_2-1} \cdots \sum_{s_n=0}^{m_n-1} k_1(t_1, \ldots, t_n, s_1) k_2(t_2, \ldots, t_n, s_2) \cdots k_n(t_n, s_n) x(s_1, \ldots, s_n)$$

where $t_i = 0, 1, \ldots, m_i - 1$ for $i = 1, 2, \ldots n$.

Exercise 14. *Consider the one-dimensional discrete Fourier transform for a sequence $x(0), x(1), \ldots, x(2^n - 1)$ of length 2^n:*

$$y(t) = \sum_{s=0}^{2^n-1} \exp\left(\frac{2\pi \Im st}{2^n}\right) x(s) \quad \text{for } t = 0, 1, \ldots, 2^n - 1.$$

Reduce to the form in the previous exercise.

2.2. **Strassen's algorithm.** How many multiplications does it take to multiply two 2×2 matrices? A direct evaluation takes 8 multiplications:

$$X = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad Y = \begin{bmatrix} e & f \\ g & h \end{bmatrix}, \quad XY = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}.$$

However, as was discovered by Strassen in 1968, 7 multiplications suffice:

$$XY = \begin{bmatrix} q_5 + q_4 - q_2 + q_6 & q_1 + q_2 \\ q_3 + q_4 & q_1 + q_5 - q_3 - q_7 \end{bmatrix},$$

where

$$\begin{aligned} q_1 &= a(f - h), & q_5 &= (a + d)(e + h), \\ q_2 &= (a + b)h, & q_6 &= (b - d)(g + h), \\ q_3 &= (c + d)e, & q_7 &= (a - c)(e + f), \\ q_4 &= d(g - e). \end{aligned}$$

Exercise 15. *Express Strassen's formula as an arithmetic circuit with three levels of gates (addition, multiplication, addition).*

Now, suppose we want to multiply two $n \times n$ matrices, X and Y. Assuming n is even (if not, insert a row and column of 0s to both matrices), we can partition X and Y each into four $\lceil n/2 \rceil \times \lceil n/2 \rceil$ submatrices

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

whereby the product XY can be expressed in similarly partitioned form as

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}.$$

Déjà vu? Indeed, Strassen's discovery tells us how to compute the matrix product XY using only 7 multiplications of $\lceil n/2 \rceil \times \lceil n/2 \rceil$ matrices! By recursion, we obtain that two $n \times n$ matrices can be multiplied in

$$T(n) = 7\,T(\lceil n/2 \rceil) + O(n^2) \tag{7}$$

arithmetic operations, that is, $T(n) = O(n^{\log_2 7}) = O(n^{2.81})$. Computing directly from the definition takes $\Theta(n^3)$ operations.

Can one do better than $O(n^{2.81})$? Put otherwise, for which exponents α do there exist algorithms such that one can multiply $n \times n$ matrices in time $O(n^\alpha)$? Obviously $\alpha \geq 2$. Let ω be the greatest lower bound for such exponents α. We say that ω is the *exponent of matrix multiplication*. Currently the best upper bound is $\omega \leq 2.376$ (see §5.3), but many conjecture that $\omega = 2$.

3. Transformations on the subset lattice

Let $U = \{1, 2, \ldots, n\}$ and denote by 2^U the set of all subsets of U. The basic set operations ($A \cup B$, $A \cap B$, $A \setminus B$, ...) on 2^U induce natural bilinear transformations on functions $f : 2^U \to R$. We study two such transformations, the *union product* and the *disjoint union product*.

3.1. **The union product.** Let $f : 2^U \to R$ and $g : 2^U \to R$. Define the *union product* $f \cup g : 2^U \to R$ by

$$(f \cup g)(S) \stackrel{\text{def}}{=} \sum_{A \cup B = S} f(A)g(B) \quad \text{for } S \subseteq U, \tag{8}$$

where the sum is understood to range over all pairs $A, B \subseteq U$ such that $A \cup B = S$.

Observe that (8) is an instance of (2).

Exercise 16. $(f \cup g) \cup h = f \cup (g \cup h)$.

Exercise 17. $(f_1 \cup f_2 \cup \cdots \cup f_k)(S) = \sum_{A_1 \cup A_2 \cup \cdots \cup A_k = S} f_1(A_1) f_2(A_2) \cdots f_k(A_k)$.

Exercise 18. *For a given $S \subseteq U$, how many pairs (A, B) are there such that $A \cup B = S$? Given f and g as input, how many arithmetic operations does it take to compute $f \cup g$ directly from the definition?*

3.2. **Moebius inversion.** The principle of *Moebius inversion* states that any finite partially ordered set $(X, \leq)$ has a pair of mutually inverse linear transformations, the *zeta transform* and the *Moebius transform*, for manipulating functions $f : X \to R$. In particular, this applies to $(2^U, \subseteq)$, and will reduce $f \cup g$ into a pointwise product amenable for fast evaluation.

Lemma 19. *A finite nonempty set has equally many subsets of even and odd size.*

Proof. Let $x \in U$. Partition the subsets of U into pairs by associating $S \subseteq U \setminus \{x\}$ with $S \cup \{x\}$. Exactly one set in each pair has even (odd) size. □

Exercise 20. *Give an algebraic proof for Lemma 19. Hint: the Binomial Theorem $(x + y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i}$.*

Let $f : 2^U \to R$. Define the *zeta transform* $f\zeta : 2^U \to R$ by

$$(f\zeta)(T) \stackrel{\text{def}}{=} \sum_{S \subseteq T} f(S) \quad \text{for } T \subseteq U. \tag{9}$$

Define the *Moebius transform* $f\mu : 2^U \to R$ by

$$(f\mu)(T) \stackrel{\text{def}}{=} \sum_{S \subseteq T} (-1)^{|S|+|T|} f(S) \quad \text{for } T \subseteq U. \tag{10}$$

Observe that both (9) and (10) are examples of (1). Also observe that if we are working over an abstract ring R, then "1" in (10) refers to the multiplicative identity element of R.

Exercise 21. *Given f as input, how many arithmetic operations does it take to compute $f\zeta$ directly from the definition?*

For a logical proposition P, we use Iverson's bracket notation $[P]$ as a shorthand for 1 when P is true, and for 0 when P is false. This notation will be convenient when simplifying nested sums, such as in the proof of the following lemma.

Lemma 22. *The zeta- and Moebius transforms on $(2^U, \subseteq)$ are mutual inverses.*

Proof. We show that $f\zeta\mu = f$ and leave $f\mu\zeta = f$ as Exercise 23. Consider an arbitrary $S \subseteq U$. We have

$$\begin{aligned}
(f\zeta\mu)(S) &= \sum_{Q \subseteq S} (-1)^{|Q|+|S|} (f\zeta)(Q) && \text{(definition)} \\
&= \sum_{Q \subseteq S} (-1)^{|Q|+|S|} \sum_{P \subseteq Q} f(P) && \text{(definition)} \\
&= \sum_{Q} [Q \subseteq S] (-1)^{|Q|+|S|} \sum_{P} [P \subseteq Q] f(P) && \text{(to brackets)} \\
&= \sum_{Q,P} [Q \subseteq S] (-1)^{|Q|+|S|} [P \subseteq Q] f(P) && \text{(expand)} \\
&= (-1)^{|S|} \sum_{P} f(P) \sum_{Q} (-1)^{|Q|} [P \subseteq Q][Q \subseteq S] && \text{(collect)} \\
&= (-1)^{|S|} \sum_{P} f(P) \sum_{Q} (-1)^{|Q|} [P \subseteq Q \subseteq S] && \text{(simplify brackets)} \\
&= (-1)^{|S|} \sum_{P} f(P) [P \subseteq S] \sum_{X \subseteq S \setminus P} (-1)^{|P|+|X|} && (X \stackrel{\text{def}}{=} Q \setminus P) \\
&= (-1)^{|S|} \sum_{P} f(P) (-1)^{|P|} [P = S] && \text{(Lemma 19)} \\
&= f(S)\,. && \text{(simplify sum)}
\end{aligned}$$

□

Exercise 23. *Show that $f\mu\zeta = f$.*

Exercise 24. *What do the matrices associated with ζ and μ look like if we consider the subsets indexing the rows and columns of the matrices in lexicographic order? Construct the matrices for small values of n. By looking at the matrices, can you devise a fast recursion for computing $f\zeta$ and $f\mu$ given f as input?*

Exercise 25. *Generalize the zeta transform to functions $f : X \to R$, where X is an arbitrary finite set equipped with a partial order $\leq$. Prove that the zeta transform on X is invertible. Derive an explicit form for the Moebius transform on your favourite partially ordered set, say, the set of positive divisors of the integer n, partially ordered by divisibility.*

Define the *pointwise product* $f \cdot g : 2^U \to R$ by $(f \cdot g)(S) \stackrel{\text{def}}{=} f(S)g(S)$ for $S \subseteq U$.

Lemma 26. $(f \cup g)\zeta = (f\zeta) \cdot (g\zeta)$.

Proof. For a $T \subseteq U$, we have

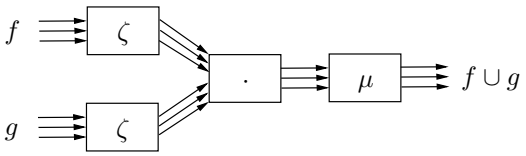
$$\begin{aligned}
((f \cup g)\zeta)(T) &= \sum_{S \subseteq T} (f \cup g)(S) && \text{(definition)}\\
&= \sum_{S \subseteq T} \sum_{A \cup B = S} f(A)g(B) && \text{(definition)}\\
&= \sum_{S} [S \subseteq T] \sum_{A,B} [A \cup B = S] f(A)g(B) && \text{(to brackets)}\\
&= \sum_{S,A,B} [S \subseteq T][A \cup B = S] f(A)g(B) && \text{(expand)}\\
&= \sum_{A,B} f(A)g(B) \sum_{S} [S \subseteq T][A \cup B = S] && \text{(collect)}\\
&= \sum_{A,B} f(A)g(B) \sum_{S} [A \cup B = S \subseteq T] && \text{(simplify brackets)}\\
&= \sum_{A,B} f(A)g(B)[A \cup B \subseteq T] && \text{(simplify sum)}\\
&= \sum_{A,B} f(A)g(B)[A \subseteq T][B \subseteq T] && \text{(simplify brackets)}\\
&= \sum_{A} [A \subseteq T] f(A) \sum_{B} [B \subseteq T] g(B) && \text{(collect)}\\
&= \sum_{A \subseteq T} f(A) \sum_{B \subseteq T} g(B) && \text{(to sums)}\\
&= (f\zeta)(T)(g\zeta)(T)\,. && \text{(definition)}
\end{aligned}$$

□

By Moebius inversion, we can recover the union product by taking the Moebius transform on both sides:

$$f \cup g = ((f\zeta) \cdot (g\zeta))\mu\,. \tag{11}$$

Again we may think of (11) as a circuit specification:



Exercise 27. $(f_1 \cup f_2 \cup \cdots \cup f_k)\zeta = (f_1\zeta) \cdot (f_2\zeta) \cdot \ldots \cdot (f_k\zeta)$.

Exercise 28. *How many operations does it take to directly evaluate the right-hand side of* $\underbrace{f \cup f \cup \cdots \cup f}_{k \text{ terms}} = (f\zeta)^{\cdot k}\mu$ *? What if we only want* $(f \cup f \cup \cdots \cup f)(U)$*?*

3.3. **Disjoint union product (subset convolution).** Let us consider a variant of the union product where we require the unions $A \cup B = S$ to be disjoint, that is, that $A \cap B = \emptyset$ must also hold.

For brevity, let us write $A \mathbin{\dot\cup} B = S$ as a shorthand for $A \cup B = S$ and $A \cap B = \emptyset$. More generally, let us write $A_1 \mathbin{\dot\cup} A_2 \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} A_k = S$ for $A_1 \cup A_2 \cup \cdots \cup A_k = S$ and $A_i \cap A_j = \emptyset$ for all $1 \leq i < j \leq k$.

Define the *disjoint union product* $f \mathbin{\dot\cup} g : 2^U \to R$ by

$$(f \mathbin{\dot\cup} g)(S) \stackrel{\text{def}}{=} \sum_{A \mathbin{\dot\cup} B = S} f(A)g(B) \quad \text{for } S \subseteq U. \tag{12}$$

The union product is also called the *subset convolution* because of the "convolution-like" equivalent form

$$(f \mathbin{\dot\cup} g)(S) = \sum_{T \subseteq S} f(T)g(S \setminus T)\,.$$

Again observe that (12) is an instance of (2).

Exercise 29. $(f \mathbin{\dot\cup} g) \mathbin{\dot\cup} h = f \mathbin{\dot\cup} (g \mathbin{\dot\cup} h)$.

Exercise 30. $(f_1 \mathbin{\dot\cup} f_2 \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} f_k)(S) = \sum_{A_1 \mathbin{\dot\cup} A_2 \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} A_k = S} f_1(A_1)f_2(A_2)\cdots f_k(A_k)$.

Exercise 31. *Given* f *and* g *as input, how many arithmetic operations does it take to compute* $f \mathbin{\dot\cup} g$ *directly using the convolution form?*

Let us now reduce $f \mathbin{\dot\cup} g$ via "polynomial extension" to the union product. The following fact will provide the crux of the argument.

Lemma 32. *For* $A, B, S \subseteq U$ *we have* $A \cup B = S$ *and* $A \cap B = \emptyset$ *if and only if* $A \cup B = S$ *and* $|A| + |B| = |S|$.

Let w be a polynomial indeterminate, and denote by $R^{\langle w \rangle}$ the associated univariate polynomial ring with coefficients in R. For a polynomial $p = \sum_i a_i w^i \in R^{\langle w \rangle}$, denote by $\{w^j\}p$ the coefficient of the monomial w^j in p, that is, $\{w^j\}p = a_j$.

Exercise 33. *Can we use an* R*-arithmetic circuit to simulate an* $R^{\langle w \rangle}$*-arithmetic circuit? What if we introduce an* $R^{\langle w \rangle}$*-arithmetic gate* $\{w^j\}$ *that takes as input a polynomial and outputs (as a polynomial of degree zero) the coefficient of the monomial* w^j *for a constant* j*?*

Exercise 34. *Can we replace computations with explicit polynomials by computations with evaluations of such polynomials in sufficiently many distinct points* $w = w_0, w_1, \ldots, w_d$ *to enable recovery of the polynomial coefficients (as necessary) via interpolation? Can we interpolate in an arbitrary ring?*

Let us now proceed with the reduction. Suppose we are given $f : 2^U \to R$ and $g : 2^U \to R$ as input. Define $f^{\langle w \rangle} : 2^U \to R^{\langle w \rangle}$ and $g^{\langle w \rangle} : 2^U \to R^{\langle w \rangle}$ by $f^{\langle w \rangle}(S) = f(S)w^{|S|}$ and $g^{\langle w \rangle}(S) = g(S)w^{|S|}$ for $S \subseteq U$.

Note in the following equality that the union product on the right-hand side is evaluated in $R^{\langle w \rangle}$.

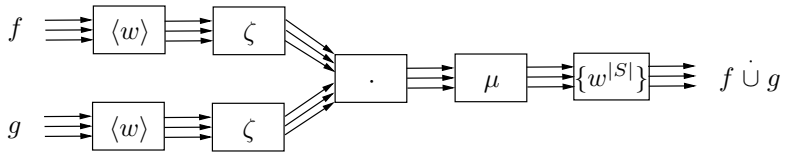
Lemma 35. $(f \mathbin{\dot\cup} g)(S) = \{w^{|S|}\}(f^{\langle w \rangle} \cup g^{\langle w \rangle})(S)$.

Proof.

$$\begin{aligned}
\{w^{|S|}\}(f^{\langle w \rangle} \cup g^{\langle w \rangle})(S) &= \{w^{|S|}\} \sum_{A \cup B = S} f^{\langle w \rangle}(A) g^{\langle w \rangle}(B) && \text{(definition)} \\
&= \{w^{|S|}\} \sum_{A \cup B = S} f(A) w^{|A|} g(B) w^{|B|} && \text{(definition)} \\
&= \sum_{A \cup B = S} \{w^{|S|}\} f(A) w^{|A|} g(B) w^{|B|} && \text{(linearity)} \\
&= \sum_{A \cup B = S} \{w^{|S|}\} f(A) g(B) w^{|A|+|B|} && \text{(collect)} \\
&= \sum_{A,B} [A \cup B = S] \{w^{|S|}\} f(A) g(B) w^{|A|+|B|} && \text{(to brackets)} \\
&= \sum_{A,B} [A \cup B = S][|A| + |B| = |S|] f(A) g(B) && \text{(definition)} \\
&= \sum_{A,B} [A \cup B = S][A \cap B = \emptyset] f(A) g(B) && \text{(Lemma 32)} \\
&= \sum_{\substack{A \cup B = S \\ A \cap B = \emptyset}} f(A) g(B) && \text{(to sums)} \\
&= \sum_{A \dot\cup B = S} f(A) g(B) && \text{(definition)} \\
&= (f \mathbin{\dot\cup} g)(S)\,. && \text{(definition)}
\end{aligned}$$

□

Again we may view the result as a circuit specification:



Exercise 36. $(f_1 \mathbin{\dot\cup} f_2 \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} f_k)(S) = \{w^{|S|}\}(f_1^{\langle w \rangle} \cup f_2^{\langle w \rangle} \cup \cdots \cup f_k^{\langle w \rangle})(S)$.

Exercise 37. *Consider the disjoint union product. Suppose we relax $A \cup B = S$ to $A, B \subseteq S$, but still require that $A \cap B = \emptyset$. Study the resulting "disjoint packing" product. Can you reduce the disjoint packing product to the disjoint union product?*

4. Two examples of algebraization

We consider two examples of algebraization and associated time–space tradeoffs enabled by fast evaluation.

4.1. **Graph coloring.** Let G be an undirected loopless graph with vertex set V. A set $I \subseteq V$ is *independent* in G if no two vertices in I are joined by an edge of G. We say that G is *k-colorable* if there exist independent sets $I_1, I_2, \ldots, I_k$ such that $I_1 \mathbin{\dot\cup} I_2 \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} I_k = V$. Such an ordered tuple $(I_1, I_2, \ldots, I_k)$ is a (proper) *k-coloring* of G. Indeed, the intuition is that the vertices in I_i have "color" i; a coloring is proper if and only if the ends of every edge have distinct colors.

The *k-coloring problem* asks, given a graph G and a positive integer k as input, whether G has a proper k-coloring.

4.2. **Algebraizing graph coloring.** Denote by $\mathbb{Z}$ the ring of integers. Let $f : 2^V \to \mathbb{Z}$ be the indicator function for independent sets in G, that is, $f(I) = [I \text{ is independent in } G]$ for $I \subseteq V$. Note that f is implicitly defined by the input G.

The number of distinct proper k-colorings of G is, by an iterated application of the disjoint union product,

$$\sum_{I_1 \dot\cup I_2 \dot\cup \cdots \dot\cup I_k = V} f(I_1) f(I_2) \cdots f(I_k) = \{w^{|V|}\}((f^{\langle w \rangle}\zeta)^{\cdot k}\mu)(V)\,. \tag{13}$$

Assuming that $|V| = n$ and $k = O(n)$, a direct evaluation of the right-hand side of (13) requires $O^*(3^n)$ time and $O^*(1)$ space, where $O^*(\cdot)$ hides a multiplicative factor polynomial in n.

4.3. **A time-space tradeoff via fast Moebius inversion.** The evaluation of product forms such as (13) can be expedited if sufficient space is available.

Lemma 38. *The zeta transform on the subset lattice can be computed in $O(2^n n)$ ring operations given space for $O(2^n)$ ring elements.*

Proof. Recall that we assume $U = \{1, 2, \ldots, n\}$. Identify $S \subseteq U$ with the binary string $s = (s_1, s_2, \ldots, s_n) \in \{0,1\}^n$ by $i \in S$ if and only if $s_i = 1$ for $1 \leq i \leq n$. In particular, we have $[S \subseteq T] = \prod_{i=1}^n [s_i \leq t_i]$. Thus, Yates's algorithm with $x(s) \leftarrow f(S)$ and $k(b,a) \leftarrow [a \leq b]$ in (3) uses $O(2^n n)$ ring operations to compute $(f\zeta)(T) = y(t)$ for all $T \subseteq U$. Storage for $O(2^n)$ ring elements suffices because z_j depends only on z_{j-1} in (5). □

Exercise 39. *Show that the Moebius transform admits a similar tradeoff.*

Theorem 40. *The union product can be computed in $O(2^n n)$ ring operations given space for $O(2^n)$ ring elements.*

Proof. We take advantage of (11) and Lemma 38. Given f and g as input, compute $f\zeta$ and $g\zeta$, take the pointwise product $(f\zeta) \cdot (g\zeta)$, and finally compute the Moebius transform $((f\zeta) \cdot (g\zeta))\mu$ to obtain $f \cup g$. Each of the three steps takes $O(2^n n)$ operations. □

Exercise 41. *Show that the disjoint union product can be computed in $O(2^n n^2)$ ring operations given space for $O(2^n)$ ring elements.*

Thus, given $O^*(2^n)$ space, we can solve graph coloring in $O^*(2^n)$ time by evaluating the right-hand side of (13).

Exercise 42. *Can you give other examples of natural "partitioning problems" that can be solved using product forms such as* (13)*?*

Exercise 43. *Algebraize the task of counting connected spanning subgraphs of a given graph with n vertices. Develop an algorithm with $O^*(2^n)$ running time. Hint: A graph with e edges has exactly 2^e spanning subgraphs, each of which partitions into one or more connected components.*

4.4. **Maximum satisfiability (MAX-SAT).** Let $x_1, x_2, \ldots, x_n$ be variables taking values in $\{0,1\}$. A *literal* is a variable x_i or its negation $\bar{x}_i$. A positive literal x_i (respectively, a negative literal $\bar{x}_i$) is *satisfied* by an assignment of values to the variables if $x_i = 1$ (respectively, $x_i = 0$). A *clause* of length k (a *k-clause*) is a set of k literals. A clause is *satisfied* if at least one of its literals is satisfied.

The *k-satisfiability* problem (k-SAT) asks, given a collection of k-clauses as input, whether there exists an assignment of values to the variables such that all input clauses are satisfied. The *maximum k-satisfiability* problem (MAX-k-SAT) asks, given a collection of k-clauses as input, for the maximum number of input clauses that can be satisfied by an assignment of values to the variables.

4.5. **Algebraizing MAX-SAT.** Suppose a collection of m clauses over n variables has been given as input. Let us view an assignment of values to the variables $x_1, x_2, \ldots, x_n$ as an n-tuple $t = (t_1, t_2, \ldots, t_n) \in \{0,1\}^n$, where t_i is the value assigned to x_i for $1 \le i \le n$.

For an assignment $t \in \{0,1\}^n$, denote by $N(t)$ the number of input clauses satisfied by t. Introduce the generating function

$$G(w) = \sum_{t \in \{0,1\}^n} w^{N(t)}\,. \tag{14}$$

Observe that $G(w)$ is a polynomial of degree at most m with nonnegative integer coefficients that sum to 2^n. In particular, the degree of $G(w)$ is the maximum number of input clauses that can be satisfied by an assignment.

4.6. **A time-space tradeoff via fast matrix multiplication.** We now restrict to the case $k = 2$ (MAX-2-SAT). Assume that 3 divides n (if not, insert new variables). Partition the variables arbitrarily into three types A, B, C so that there are exactly $n/3$ variables of each type. Partition the input clauses into types A, B, C so that a clause of type T does **not** contain a variable of type T. Because $k = 2$, such a partition of the input clauses always exists.

We can now split an assignment $t \in \{0,1\}^n$ into three sub-assignments $a, b, c \in \{0,1\}^{n/3}$ for the variables of each type. The number of satisfied input clauses splits accordingly into

$$N(t) = N_C(a,b) + N_B(a,c) + N_A(b,c) \tag{15}$$

where N_C, N_B, and N_A count the number of satisfied input clauses of each respective type. In particular, N_T is independent of the sub-assignment to variables of type T.

We proceed to split $G(w)$ using (15) and recover a matrix product. Let $N_C^{\langle w \rangle}$, $N_B^{\langle w \rangle}$, $N_A^{\langle w \rangle}$ be matrices of size $2^{n/3} \times 2^{n/3}$ with entries defined for $a, b, c \in \{0,1\}^{n/3}$ by

$$N_C^{\langle w \rangle}(a,b) \stackrel{\text{def}}{=} w^{N_C(a,b)}, \quad N_B^{\langle w \rangle}(a,c) \stackrel{\text{def}}{=} w^{N_B(a,c)}, \quad N_A^{\langle w \rangle}(b,c) \stackrel{\text{def}}{=} w^{N_A(b,c)}.$$

We now have

$$\begin{aligned} G(w) &= \sum_{a,b,c} w^{N_C(a,b)+N_B(a,c)+N_A(b,c)} && \text{(split to types)} \\ &= \sum_{a,b,c} w^{N_C(a,b)} w^{N_B(a,c)} w^{N_A(b,c)} && \text{(expand)} \\ &= \sum_{a,c} w^{N_B(a,c)} \sum_{b} w^{N_C(a,b)} w^{N_A(b,c)} && \text{(collect)} \\ &= \sum_{a,c} N_B^{\langle w \rangle}(a,c) \sum_{b} N_C^{\langle w \rangle}(a,b) N_A^{\langle w \rangle}(b,c) && \text{(in matrix form)} \\ &= \sum_{a,c} N_B^{\langle w \rangle}(a,c) \big(N_C^{\langle w \rangle} N_A^{\langle w \rangle}\big)(a,c). && \text{(to matrix product)} \end{aligned} \tag{16}$$

A trivial algorithm for MAX-2-SAT runs in $O^*(2^n)$ time and $O^*(1)$ space. Using (16), we can now trade space for time by first constructing the matrices $N_C^{\langle w \rangle}$, $N_B^{\langle w \rangle}$, $N_A^{\langle w \rangle}$ and then using fast matrix multiplication to determine the product $N_C^{\langle w \rangle} N_A^{\langle w \rangle}$ in $O((2^{n/3})^{\omega+\epsilon})$ ring operations for any fixed $\epsilon > 0$, which leads to $O^*(2^{(\omega+\epsilon)n/3})$ time and $O^*(2^{2n/3})$ space for MAX-2-SAT.

Exercise 44. *Observe that one can carry out the computations in* (16) *with evaluations of the generating function $G(w)$ at $m+1$ distinct points $w = w_0, w_1, \ldots, w_m$, and then recover $G(w)$ via interpolation.*

5. Further exercises and remarks

5.1. **Transforms on the subset lattice.** The following exercises develop and relate to each other some further transformations on the subset lattice.

Define the *up-zeta transform* $f\zeta' : 2^U \to R$ by

$$f\zeta'(T) \stackrel{\text{def}}{=} \sum_{T \subseteq S} f(S) \quad \text{for } T \subseteq U.$$

Define the *up-Moebius transform* $f\mu' : 2^U \to R$ by

$$f\mu'(T) \stackrel{\text{def}}{=} \sum_{T \subseteq S} (-1)^{|S|+|T|} f(S) \quad \text{for } T \subseteq U.$$

Exercise 45. *Observe that in matrix form we obtain ζ' from ζ by taking the transpose. Similarly for μ' and μ.*

Define the *complement transform* $f\kappa : 2^U \to R$ by

$$(f\kappa)(S) \stackrel{\text{def}}{=} f(U \setminus S) \quad \text{for } S \subseteq U.$$

Define the *odd-negation transform* $f\sigma : 2^U \to R$ by

$$(f\sigma)(S) \stackrel{\text{def}}{=} (-1)^{|S|} f(S) \quad \text{for } S \subseteq U.$$

Exercise 46. $\zeta' = \kappa\zeta\kappa$, $\mu' = \kappa\mu\kappa$, $\mu = \sigma\zeta\sigma$, $\zeta = \sigma\mu\sigma$.

Exercise 47. $\zeta'\mu = \zeta\kappa\sigma$, $\zeta\mu' = \zeta'\sigma\kappa$, $\mu'\zeta = \mu\sigma\kappa$, $\mu\zeta' = \mu'\kappa\sigma$.

Exercise 48. *Present a natural definition for $\delta \stackrel{\text{def}}{=} \zeta\kappa$. Show that $\delta = \zeta'\sigma\zeta$.*

Exercise 49. *Define the intersection product* $f \cap g : 2^U \to R$ *by* $(f \cap g)(S) \stackrel{\text{def}}{=} \sum_{A \cap B = S} f(A)g(B)$ *for* $S \subseteq U$*. Show that* $(f \cap g)\kappa = (f\kappa) \cup (f\kappa)$*. Take the up-zeta transform of* $f \cap g$ *and simplify to a pointwise product. Establish* $(f \cap g) \cap h = f \cap (g \cap h)$*.*

Exercise 50. *Define the difference product* $f \setminus g : 2^U \to R$ *by* $(f \setminus g)(S) \stackrel{\text{def}}{=} \sum_{A \setminus B = S} f(A)g(B)$ *for* $S \subseteq U$*. Show that* $f \setminus g = f \cap (g\kappa)$*.*

Exercise 51. *Define the Walsh–Hadamard transform (the Fourier transform on the n-cube)* $f\phi : 2^U \to R$ *by* $(f\phi)(S) \stackrel{\text{def}}{=} \sum_T (-1)^{|S \cap T|} f(T)$ *for* $S \subseteq U$*. Show that* $f\phi^2 = 2^n f$*. Is* ϕ *invertible if* R *is an arbitrary ring?*

Exercise 52. *For* $A, B \subseteq U$*, define the symmetric difference* $A \oplus B \stackrel{\text{def}}{=} (A \setminus B) \cup (B \setminus A)$*. Define the symmetric difference product* $f \oplus g : 2^U \to R$ *by* $(f \oplus g)(S) \stackrel{\text{def}}{=} \sum_{A \oplus B = S} f(A)g(B)$ *for* $S \subseteq U$*. Show that* $(f \oplus g)\phi = (f\phi) \cdot (g\phi)$*. Establish* $(f \oplus g) \oplus h = f \oplus (g \oplus h)$*.*

Exercise 53. *Reduce the disjoint union product to the symmetric difference product.*

Exercise 54. *Let* $f : 2^U \to R$ *and* $k : 2^U \to R$*. Define the k-intersection transform* $f\tau_k^\cap : 2^U \to R$ *by* $(f\tau_k^\cap)(S) = \sum_T k(S \cap T)f(T)$ *for* $S \subseteq U$*. Show that* $f\tau_k^\cap = ((k\mu) \cdot (f\zeta'))\zeta$*.*

Exercise 55. *Observe that* ϕ *reduces to* $\tau_k^\cap$ *for a specific* k*. Simplify the pointwise product form of the k-intersection product when* k *is* 1 *on sets of size* j*, and* 0 *elsewhere.*

Exercise 56. *Define similar k-union, k-difference, and k-symmetric difference transforms. Reduce each to a pointwise product form.*

Exercise 57. *Are all the transforms in this section computable in* $O^*(2^n)$ *ring operations given space for* $O^*(2^n)$ *ring elements?*

5.2. **Trimming.** This section investigates "trimming" of transforms on 2^U when the input and output are restricted to subsets of 2^U.

Our first objective is a "closure trimming lemma" for fast Moebius inversion.

Let $\mathcal{F} \subseteq 2^U$. Denote by $\uparrow\mathcal{F}$ the *up-closure* of $\mathcal{F}$, that is, the set consisting of the sets in $\mathcal{F}$ and all their supersets in 2^U. Denote by $\downarrow\mathcal{F}$ the *down-closure* of $\mathcal{F}$, that is, the set consisting of the sets in $\mathcal{F}$ and all their subsets in 2^U. Define the *elementwise complement* of $\mathcal{F}$ by $U\setminus\mathcal{F} \stackrel{\text{def}}{=} \{U\setminus S : S \in \mathcal{F}\}$.

Exercise 58. $U\setminus\uparrow\mathcal{F} = \downarrow U\setminus\mathcal{F}, \quad U\setminus\downarrow\mathcal{F} = \uparrow U\setminus\mathcal{F}$*.*

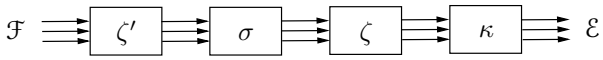
Lemma 59. *There exist algorithms that, given* $\mathcal{E} \subseteq 2^U$ *and* $\mathcal{F} \subseteq 2^U$ *as input, construct an* R*-arithmetic circuit with input gates for* $f : \mathcal{F} \to R$ *and output gates that evaluate to any of* $f\zeta : \mathcal{E} \to R$*,* $f\zeta' : \mathcal{E} \to R$*,* $f\mu : \mathcal{E} \to R$*, or* $f\mu' : \mathcal{E} \to R$*, with construction time any of* $O^*(|\uparrow\mathcal{E}| + |\uparrow\mathcal{F}|)$*,* $O^*(|\uparrow\mathcal{E}| + |\downarrow\mathcal{F}|)$*,* $O^*(|\downarrow\mathcal{E}| + |\uparrow\mathcal{F}|)$*, or* $O^*(|\downarrow\mathcal{E}| + |\downarrow\mathcal{F}|)$*.*

Proof. Because $\mu = \sigma\zeta\sigma$ and $\mu' = \sigma\zeta'\sigma$, it suffices to consider ζ and ζ' only. Because $\zeta' = \kappa\zeta\kappa$, it suffices to consider ζ only. To evaluate $f\zeta : \mathcal{E} \to R$ it suffices to consider $f : \downarrow\mathcal{E} \to R$ because f outside $\downarrow\mathcal{E}$ does not affect $f\zeta$ in $\mathcal{E}$.

Time $O^*(|{\downarrow}\mathcal{E}| + |\mathcal{F}|)$. Consider Yates's algorithm with $k(0,0) = 1$, $k(0,1) = 0$, $k(1,0) = 1$, and $k(1,1) = 1$, that is, the output is the zeta transform of the input. In this case any walk in the n-cube that has a $1 \mapsto 0$ step has weight 0 because $k(0,1) = 0$. Thus, such walks may be discarded. Furthermore, all walks to vertices in $\mathcal{E}$ that do not have a $1 \mapsto 0$ step traverse only vertices in ${\downarrow}\mathcal{E}$. Thus, Yates's algorithm in fact specifies an R-arithmetic circuit of size $O(n|{\downarrow}\mathcal{E}|)$ with inputs and outputs indexed by ${\downarrow}\mathcal{E}$. The construction is completed by connecting the inputs in $\mathcal{F}$ with corresponding input gates in the zeta circuit; any inputs to the zeta circuit not in $\mathcal{F}$ are forced to 0.

Time $O^*(|\mathcal{E}| + |{\uparrow}\mathcal{F}|)$. Observe that $f\zeta$ vanishes outside ${\uparrow}\mathcal{F}$. Furthermore, all walks to ${\uparrow}\mathcal{F}$ from $\mathcal{F}$ stay within ${\uparrow}\mathcal{F}$. Restrict Yates to ${\uparrow}\mathcal{F}$ and output 0 outside ${\uparrow}\mathcal{F}$.

Time $O^*(|{\uparrow}\mathcal{E}| + |{\downarrow}\mathcal{F}|)$. Observe that $\zeta = \zeta'\sigma\zeta\kappa$. Let us develop the right-hand side into a sequential circuit by "meeting in the middle."



Starting from the left, given the inputs at $\mathcal{F}$, we have that the output of ζ' vanishes outside ${\downarrow}\mathcal{F}$. Starting from the right, the outputs at $\mathcal{E}$ require input at $U\backslash\mathcal{E}$ for κ. Thus, we require input for ζ at ${\downarrow}U\backslash\mathcal{E}$ to evaluate the outputs at $U\backslash\mathcal{E}$. The odd-negation layer σ thus requires input and output at ${\downarrow}U\backslash\mathcal{E}$. We now connect the outputs of the left part with the corresponding inputs of the right part at $({\downarrow}\mathcal{F}) \cap ({\downarrow}U\backslash\mathcal{E})$, and force any remaining inputs of the right part to 0. Observe that ${\downarrow}U\backslash\mathcal{E} = U\backslash{\uparrow}\mathcal{E}$ and that $|U\backslash{\uparrow}\mathcal{E}| = |{\uparrow}\mathcal{E}|$. □

Exercise 60. *Open problem: Can one evaluate $f\zeta : \mathcal{E} \to R$ for given $\mathcal{E}, \mathcal{F} \subseteq 2^U$ and $f : \mathcal{F} \to R$ using an R-arithmetic circuit of size $O^*(|{\uparrow}\mathcal{E}| + |\mathcal{F}|)$ or $O^*(|\mathcal{E}| + |{\downarrow}\mathcal{F}|)$?*

Let us sketch an application of Lemma 59 in counting long paths in graphs. Recalling Exercise 49, consider the intersection product $f \cap g : \mathcal{E} \to R$ for given $f : \mathcal{F} \to R$ and $g : \mathcal{G} \to R$. We have

$$f \cap g = ((f\zeta') \cdot (g\zeta'))\mu'.$$

Observe that $(f\zeta') \cdot (g\zeta')$ vanishes outside $({\downarrow}\mathcal{F}) \cap ({\downarrow}\mathcal{G})$. Thus, Lemma 59 implies that we can evaluate $f \cap g$ in $O^*(|{\downarrow}\mathcal{E}| + |{\downarrow}\mathcal{F}| + |{\downarrow}\mathcal{G}|)$ ring operations, given space for $O^*(|{\downarrow}\mathcal{E}| + |{\downarrow}\mathcal{F}| + |{\downarrow}\mathcal{G}|)$ ring elements. In particular, letting $\mathcal{E} = \{\emptyset\}$, we can evaluate the sum

$$(f \cap g)(\emptyset) = \sum_{\substack{A \in \mathcal{F}, B \in \mathcal{G} \\ A \cap B = \emptyset}} f(A)g(B) \tag{17}$$

in $O^*(|{\downarrow}\mathcal{F}| + |{\downarrow}\mathcal{G}|)$ ring operations, given space for $O^*(|{\downarrow}\mathcal{F}| + |{\downarrow}\mathcal{G}|)$ ring elements.

Now consider an undirected graph G with vertex set V, $|V| = n$. Suppose we want to compute the number of (s,t)-paths of length k for given G, $1 \leq k \leq n-1$, and distinct $s, t \in V$. (The length of a path is the number of edges in it.) For simplicity, we assume that k is even.

Observe that every (s,t)-path of even length k has a unique midpoint $v \in V \setminus \{s,t\}$ that splits the path into two halves of equal length; that is, into an (s,v)-path and a (v,t)-path, both of length $k/2$.

We proceed to "count in halves" via (17). Let $v \in V \setminus \{s,t\}$ and $U = V \setminus \{v\}$. Let $f_v(A)$ be the number of (s,v)-paths in G with vertex set $A \cup \{v\}$, and let $g_v(B)$ be the number of (v,t)-paths in G with vertex set $B \cup \{v\}$.

Exercise 61. *Design a dynamic programming algorithm that computes* $f_v : \binom{U}{k/2} \to \mathbb{Z}$ *and* $g_v : \binom{U}{k/2} \to \mathbb{Z}$ *in time* $O^*(\binom{n}{k/2})$.

It now follows from (17) that we can count in time and space $O^*(\binom{n}{k/2})$ the number of (s,t)-paths of length k in G, that is,

$$\sum_{v \in V \setminus \{s,t\}} \sum_{\substack{A,B \in \binom{V \setminus \{v\}}{k/2} \\ A \cap B = \emptyset}} f_v(A) g_v(B).$$

5.3. **Bibliography.** Donald Knuth's *The Art of Computer Programming* (Volume 2: Seminumerical Algorithms, 3rd ed., Addison–Wesley, 1998) gives a comprehensive introduction to arithmetic algorithms. More specialized texts include *Computational Complexity of Algebraic and Numeric Problems* by A. Borodin and I. Munro (Elsevier, 1975), *Polynomial and Matrix Computations* by D. Bini and V. Y. Pan (Volume 1: Fundamental algorithms, Birkhäuser, 1994), and *Algebraic Complexity Theory* by P. Bürgisser, M. Clausen, and M. Amin Shokrollahi (Springer, 1997).

Strassen's algorithm appears in [V. Strassen, *Numer. Math.* 13 (1969) 354–356]. Yates's algorithm is due to F. Yates [*The Design and Analysis of Factorial Experiments*, Imperial Bureau of Soil Sciences, Harpenden 1937]. Currently the asymptotically fastest algorithm for matrix multiplication is due to D. Coppersmith and S. Winograd [*J. Symbolic Comp.* 9 (1990) 251–280].

An introductory text covering basic combinatorial techniques such as Moebius inversion is *A Course in Combinatorics* by J. H. van Lint and R. M. Wilson (Cambridge University Press, 1992).

The union product (in a dual form) together with fast Moebius inversion was introduced in [R. Kennes, *IEEE Transactions on Systems, Man, and Cybernetics* 22 (1991) 201–223]. The disjoint union product was introduced in A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto [Proc. 39th ACM Symposium on Theory of Computing, 2007, 67–74].

A. Björklund, T. Husfeldt, and M. Koivisto [*SIAM J. Comput.*, to appear; see also Proc. 47th IEEE Symposium on Foundations of Computer Science, 2006, 575–582 and 583–590] discovered that graph coloring and other partitioning problems can be solved in time $O^*(2^n)$ using inclusion–exclusion. Algebraization of MAX-2-CSP via matrix multiplication is due to R. Williams [*Theoret. Comput.Sci.* 348 (2005) 357–365] and M. Koivisto [*Inform. Proc. Lett.* 98 (2006) 22–24].

The closure trimming lemma (Lemma 59) is an unpublished extension of results in A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto [Proc. 25th Annual Symposium on Theoretical Aspects of Computer Science, 2008, 85–96]. The k-intersection transform in Exercise 54 is an unpublished slightly stronger version of results in A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto [arXiv:0809.2489]. The application to counting paths in graphs is from A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto [arXiv:0904.3093]; see [arXiv:0904.3251] for an application to evaluation of permanents.

Helsinki Institute for Information Technology HIIT, University of Helsinki, Department of Computer Science, P.O. Box 68, 00014 University of Helsinki, Finland

E-mail address: petteri.kaski@cs.helsinki.fi