Finding Optimal Formulae for Bilinear Maps

Razvan Barbulescu, Jérémie Detrey, Nicolas Estibals, Paul Zimmermann

Finding Optimal Formulae for Bilinear Maps

Razvan Barbulescu, Jérémie Detrey, Nicolas Estibals, and Paul Zimmermann

CARAMEL project-team, LORIA, Université de Lorraine / INRIA / CNRS,
Campus Scientifique, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France

Abstract. We describe a unified framework to search for optimal formulae evaluating bilinear — or quadratic — maps. This framework applies to polynomial multiplication and squaring, finite field arithmetic, matrix multiplication, etc. We then propose a new algorithm to solve problems in this unified framework. With an implementation of this algorithm, we prove the optimality of various published upper bounds, and find improved upper bounds.

Keywords: optimal algorithms, polynomial multiplication and squaring, finite field arithmetic, tensor rank, bilinear map, bilinear rank

1 Introduction

This article studies the *bilinear rank problem*. Given a field K, this problem naturally arises when considering the computational complexity of formulae (or algorithms) for evaluating bilinear maps over K [4, Ch. 14]. For instance, typical bilinear maps include — but are not limited to — the multiplication of two n-term polynomials of $K[X]$, or of two $r \times r$ square matrices of $K^{r\times r}$.

Note that, given an algorithm for computing a bilinear map over K, this algorithm can be naturally extended to compute the same bilinear map over any larger K-algebra $\mathbb{K}$. For instance, given a formula for computing the product of two n-term polynomials over $K = \mathbb{F}_2$, the exact same formula can also be used to compute the product of two n-term polynomials over any field extension $\mathbb{K} = \mathbb{F}_{2^m}$, or even the product of two n-term polynomials of $\mathbb{F}_2^{r\times r}[X]$, *i.e.*, polynomials having binary $r \times r$ matrices as coefficients. A nice illustration of the latter is given by Albrecht in [1], where the multiplication of large $r \times r$ matrices over $\mathbb{F}_{2^n}$ is implemented by representing $\mathbb{F}_{2^n}^{r\times r}$ as $\mathbb{F}_2^{r\times r}[X]/\langle f(X)\cdot\mathbb{F}_2^{r\times r}[X]\rangle$, with f an irreducible polynomial over $\mathbb{F}_2$ of degree n, and then using Montgomery's n-term polynomial multiplication formulae [16].

Therefore, when dealing with such formulae, a crucial distinction is made between

- a *full multiplication* $a\cdot b$ of two elements a and b derived from the inputs of the bilinear map, and thus possibly living in the larger algebra $\mathbb{K}$; and
- an *addition* $a + b$ or a *scalar multiplication* λa, where λ is given by the bilinear map only, and thus belongs to the smaller coefficient field K.

Intuitively, the first one is expected to have a much higher computational cost than the last two. Table 3 in [1] shows that the time needed to multiply two matrices over fields of characteristic 2 is proportional to the number of full multiplications in the formulae used for this purpose. A sensible way to optimize the overall complexity of the formula at hand is therefore to minimize the number of these full multiplications, which corresponds to solving a particular instance of the bilinear rank problem.

Most of the fast algorithms for polynomial or integer multiplication can be expressed in terms of the bilinear rank problem, for example Karatsuba's algorithm [15] and Toom's algorithm [20]. But this applies to other problems as well, like the middle product [13], or matrix multiplication [3,11,19].

Definition of the problem. Let K be a field. Given an $n \times m$ *bilinear map* $\phi : K^n \times K^m \to K^\ell$, the bilinear rank problem consists in finding formulae for evaluating ϕ involving a minimal number k of full multiplications. This optimal k is called the *bilinear rank* of the map ϕ.

More formally, consider $\mathcal{B}(K^n, K^m; K)$ the set of $n \times m$ *bilinear forms* from $K^n \times K^m$ to K. Each such bilinear form γ can be written as $\gamma : \mathbf{a}, \mathbf{b} \mapsto \sum_{i,j} \gamma_{i,j} a_i b_j$, with the two input vectors $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1}) \in K^n$, $\mathbf{b} = (b_0, b_1, \ldots, b_{m-1}) \in K^m$, and the coefficients $\gamma_{i,j} \in K$. As γ is uniquely determined by the $\gamma_{i,j}$'s, we can represent $\mathcal{B}(K^n, K^m; K)$ as the vector space of dimension nm over K, which we will denote by V in the following. The representation of the bilinear form γ in V is therefore the nm-dimensional vector $\gamma = (\gamma_{0,0}, \gamma_{0,1}, \ldots, \gamma_{n-1,m-1})$, and a bilinear map ϕ from $K^n \times K^m$ to K^ℓ can then be represented as a tuple of ℓ vectors of V: $\phi = (\phi_0, \phi_1, \ldots, \phi_{\ell-1}) \in V^\ell$.

Consider now an $n \times m$ bilinear form γ. We say that γ is *of rank* 1 if there exist two linear forms $\alpha : K^n \to K, \mathbf{a} \mapsto \sum_i \alpha_i a_i$ and $\beta : K^m \to K, \mathbf{b} \mapsto \sum_j \beta_j b_j$ such that $\gamma(\mathbf{a}, \mathbf{b}) = \alpha(\mathbf{a}) \cdot \beta(\mathbf{b})$ for all $\mathbf{a} \in K^n$ and $\mathbf{b} \in K^m$.

Rank-1 bilinear forms are particularly relevant in the context of the bilinear rank problem: indeed, each such form can be evaluated using only one full multiplication, as evaluating the two linear forms α and β at $\mathbf{a}$ and $\mathbf{b}$, respectively, only requires scalar multiplications and additions. Furthermore, since the maps $e_{i,j} : \mathbf{a}, \mathbf{b} \mapsto a_i b_j$ are all rank-1 bilinear forms, any bilinear form can then be written as a linear combination of rank-1 bilinear forms: $\gamma = \sum_{i,j} \gamma_{i,j} e_{i,j}$. Therefore, one can ask the question: What is the minimal number of rank-1 bilinear forms necessary to evaluate a given bilinear form? A given bilinear map?

In the case of a (single) bilinear form γ, the answer is easy: the bilinear rank is given by the actual rank of the $n \times m$ matrix $(\gamma_{i,j})_{0 \le i < n, 0 \le j < m}$ formed by the coefficients of γ. However, the minimal number of full multiplications is much more difficult to compute when evaluating $\ell \ge 2$ bilinear forms simultaneously, such as is the case with bilinear maps: this is the aforementioned *bilinear rank problem*, formalized below.

Definition 1 (Bilinear rank problem [4, Ch. 14]). *Using the above notations, and given a finite* generator set $\mathcal{G} \subset V$ *composed of rank-1 $n \times m$ bilinear forms, along with a finite* target set *of ℓ bilinear forms $\mathcal{T} = \{t_0, t_1, \ldots, t_{\ell-1}\} \subseteq \mathrm{Span}(\mathcal{G})$, the* bilinear rank problem *consists in generating all the elements of $\mathcal{T}$ by K-linear combinations of a minimal number k of elements of $\mathcal{G}$ or, alternatively, to find all solutions for that optimal k.*

Without loss of generality, the target vectors t_i can further be assumed to be linearly independent, as reconstructing an extra target vector by linear combination would only require scalar multiplications and additions.

Note that this problem can be adapted to also encompass the case of sets of *quadratic* forms. Indeed, any quadratic form σ from K^n to K, $\sigma : \mathbf{a} \mapsto \sum_{0 \le i \le j < n} \sigma_{i,j} a_i a_j$, is uniquely represented by its coefficients $\sigma_{i,j} \in K$. The set of n-ary quadratic forms $\mathcal{Q}(K^n; K)$ can thus be seen as a vector space of dimension $n(n+1)/2$ over K. It suffices then to take V to be this vector space, and to define the rank-1 quadratic forms to be the quadratic forms σ for which there exist two linear forms over K^n, $\alpha : \mathbf{a} \mapsto \sum_i \alpha_i a_i$ and $\alpha' : \mathbf{a} \mapsto \sum_i \alpha'_i a_i$, such that $\sigma(\mathbf{a}) = \alpha(\mathbf{a}) \cdot \alpha'(\mathbf{a})$ for all $\mathbf{a} \in K^n$.

In fact, any $n \times m$ bilinear form can be seen as an $(n+m)$-ary quadratic form σ such that no product $a_i a_j$ occurs between the first n or between the last m input variables: $\sigma_{i,j} = 0$ for all $0 \le i \le j < n$ and for all $n \le i \le j < n + m$. Hence, computing the rank of a set of quadratic forms is more general than computing the bilinear rank.

Also, the bilinear rank problem is NP-hard. Indeed, Bürgisser *et al.* show in [4, Sec. 14.2] its equivalence to the decomposition of an order-3 tensor, known to be NP-hard [14].

Notation. Note that in the rest of this document, by an abuse of notation, we will omit writing the input vectors of the considered bilinear or quadratic forms, as they will always be $\mathbf{a}, \mathbf{b}$ or $\mathbf{a}$, respectively. Hence, we will simply write $\gamma = \sum_{i,j} \gamma_{i,j} a_i b_j$ or $\sigma = \sum_{i \leq j} \sigma_{i,j} a_i a_j$ to refer to the corresponding forms in the following. This amounts to implicitly considering the a_i's and b_j's as formal variables over K.

Related results. Several authors have considered special instances of the bilinear rank problem. We do not claim an exhaustive state-of-the-art here, but we mention the main results related to our work.

For polynomial multiplication, evaluation–interpolation algorithms like Karatsuba's and Toom's algorithms [15,20] are special cases of the problem, where the only full multiplications involved correspond to evaluations of the product of the two polynomials at different points.

Even though the bilinear rank problem was already well known in the algebraic complexity community [4], until only recently, all the formulae used in the computer arithmetic community were based on this evaluation–interpolation scheme. In [16], instead of using only the evaluation products, *i.e.*, $(\sum_i \kappa^i a_i)(\sum_j \kappa^j b_j)$ for some $\kappa \in K$, Montgomery considered other rank-1 bilinear forms. This allowed him to find formulae with less products for the 5-, 6-, and 7-term polynomial multiplications, which do not fit in any evaluation–interpolation scheme. Montgomery however restricted the exploration to the set of "symmetric" generators of the form $(\sum_i \alpha_i a_i)(\sum_j \alpha_j b_j)$. Our work shows how to improve Montgomery's exploration and proves that the formulae in [16] are optimal for the 5-term polynomial multiplication.

In [9], Chung and Hasan propose asymmetric squaring formulae over $\mathbb{Q}$, found using an exhaustive search method similar to that of Montgomery, but starting from an ad-hoc set of rank-1 generators. In [12], Fan and Hasan use Montgomery's formulae and the Chinese Remainder Theorem together with a short product for the high degree terms to improve the bounds over $\mathbb{F}_2$; those results were further improved by Cenk and Özbudak [5]. In [18], using both exhaustive and heuristic search algorithms, Oseledets considers the multiplication of n-term polynomials, and also their product modulo X^n — which corresponds to Mulders' short product [17] — and modulo $X^n + 1$. Some of the linear algebra routines he proposed are a building block in our method.

Multiplication in finite field extensions $\mathbb{F}_{q^n}$ for $n \geq 2$ reduces to polynomial multiplication modulo a degree-n irreducible polynomial f over $\mathbb{F}_q$. Thus, one could first compute a full product of two n-term polynomials, then reduce it modulo f using only scalar multiplications; however it is sometimes faster to directly compute the n terms of the product modulo f [7].

Fast matrix multiplication is another application of the bilinear rank problem, with the smallest unsolved problem being the multiplication of two 3×3 matrices (for two square matrices): we know that at least 19 products are needed, and that 23 are enough, but the minimal rank is still unknown [3,11].

Contributions. The contributions of the article are the following. First, we present the bilinear rank problem, as it is known to the algebraic complexity community [4], showing

why it is pertinent to the computer algebra and computer arithmetic communities. Following Montgomery [16], we see this problem in the light of linear algebra, which enables one to search for optimal formulae for a large set of applications such as polynomial multiplication or squaring, matrix multiplication, etc. Second, we propose a new algorithm to solve the bilinear rank problem; this algorithm is faster than Montgomery's exploration algorithm [16] and is an improvement on Oseledets' heuristic Minbas [18], which might not find the minimal rank. Last, using this new algorithm we prove the optimality of several known formulae, and exhibit new bounds for some bilinear maps.

Roadmap. The article is organized as follows. After detailing a few instances of the bilinear rank problem in Section 2, we show in Section 3 how this problem can be translated into a linear algebra problem. Section 4 gives an efficient algorithm solving this linear algebra problem. Finally Section 5 presents experimental results proving that some bounds from the literature are optimal, along with an improved bound for the multiplication in $\mathbb{F}_{3^5}$.

2 Some Instances of the Bilinear Rank Problem

Throughout the rest of this paper, we will use the following running example:

Example 1 (2 × 3-term polynomial product). We want to multiply $A = a_1X + a_0$ by $B = b_2X^2 + b_1X + b_0$ in $K[X]$. The naive algorithm requires 6 products, while only 5 products are necessary when using Karatsuba's trick:

$$A \cdot B = g_3X^3 + (g_1 + g_2)X^2 + (g_4 - g_2 - g_0)X + g_0,$$

where $g_0 = a_0b_0$, $g_1 = a_0b_2$, $g_2 = a_1b_1$, $g_3 = a_1b_2$, and $g_4 = (a_0 + a_1)(b_0 + b_1)$. Since $A \cdot B$ has 4 terms, if the base field K contains at least 3 elements, evaluating $A \cdot B$ at those elements and at infinity — *i.e.*, the a_1b_2 product — enables to recover $A \cdot B$ by Lagrange interpolation. However, if $K = \mathbb{F}_2$, are 5 products optimal?

To illustrate the generality of the proposed framework, we give a few more examples:

Example 2 (3-term polynomial squaring [9]). We want to square the polynomial $A = a_2X^2 + a_1X + a_0 \in K[X]$. For this quadratic map, we have $n = 3$, the generator set $\mathcal{G}$ consists of the 28 products

$$\begin{array}{l}
a_0^2,\ a_0a_1,\ a_0a_2,\ a_0(a_0 + a_1),\ a_0(a_0 + a_2),\ a_0(a_1 + a_2),\ a_0(a_0 + a_1 + a_2),\\
a_1^2,\ a_1a_2,\ a_1(a_0 + a_1),\ a_1(a_0 + a_2),\ a_1(a_1 + a_2),\ a_1(a_0 + a_1 + a_2),\\
a_2^2,\ a_2(a_0 + a_1),\ a_2(a_0 + a_2),\ a_2(a_1 + a_2),\ a_2(a_0 + a_1 + a_2),\\
(a_0 + a_1)^2,\ (a_0 + a_1)(a_0 + a_2),\ (a_0 + a_1)(a_1 + a_2),\ (a_0 + a_1)(a_0 + a_1 + a_2),\\
(a_0 + a_2)^2,\ (a_0 + a_2)(a_1 + a_2),\ (a_0 + a_2)(a_0 + a_1 + a_2),\\
(a_1 + a_2)^2,\ (a_1 + a_2)(a_0 + a_1 + a_2),\\
(a_0 + a_1 + a_2)^2,
\end{array}$$

and the target set is

$$\mathcal{T} = \{t_0 := a_0^2,\ t_1 := 2a_0a_1,\ t_2 := a_1^2 + 2a_0a_2,\ t_3 := 2a_1a_2,\ t_4 := a_2^2\},$$

which are the 5 quadratic forms corresponding to the coefficients of A^2.

Example 3 (Middle product [13]). Assume we only want to compute the degree-1 and -2 coefficients of the product $A \cdot B$ from Example 1. We have $n = 2$, $m = 3$, and $\mathcal{G}$ is the set of all rank-1 bilinear forms of the form $(\alpha_0 a_0 + \alpha_1 a_1)(\beta_0 b_0 + \beta_1 b_1 + \beta_2 b_2)$ for $\alpha_i, \beta_j \in K$. The target set is

$$\mathcal{T} = \{t_0 := a_0 b_1 + a_1 b_0,\ t_1 := a_0 b_2 + a_1 b_1\}.$$

A solution with $k = 3$ considers the subset

$$\{g_0 := a_0(b_1 + b_2),\ g_1 := a_1(b_0 + b_1),\ g_2 := (a_1 - a_0)b_1\} \subseteq \mathcal{G},$$

which gives the formulae $t_0 = g_1 - g_2$ and $t_1 = g_0 + g_2$.

Example 4 (3×3 matrix multiplication). Here, $n = m = 9$, and we consider the product $A \cdot B$ over $K^{3\times 3}$ of the two matrices

$$A = \begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} b_0 & b_1 & b_2 \\ b_3 & b_4 & b_5 \\ b_6 & b_7 & b_8 \end{pmatrix}.$$

The target set $\mathcal{T}$ consists of the 9 bilinear forms $\{a_0b_0 + a_1b_3 + a_2b_6, ..., a_6b_2 + a_7b_5 + a_8b_8\}$, and the generator set $\mathcal{G}$ consists of the non-zero rank-1 bilinear forms $(\alpha_0 a_0 + \cdots + \alpha_8 a_8)(\beta_0 b_0 + \cdots + \beta_8 b_8)$ for $\alpha_i, \beta_j \in K$, *i.e.*, $(2^9 - 1)^2 = 261\,121$ forms for $K = \mathbb{F}_2$.

3 From Bilinear Applications to Linear Algebra

We focus in the rest of the paper on bilinear maps, the case of quadratic maps being similar. Recall that we denote by V the vector space of dimension nm over K isomorphic to the space of $n \times m$ bilinear forms $\mathcal{B}(K^n, K^m; K)$. Thus, each element of $\mathcal{T}$ and $\mathcal{G}$ (see Definition 1) being such a bilinear form, it can be written as a row vector of dimension nm, where the $(im + j)$-th column corresponds to the coefficient of the $a_i b_j$ term.

Example 1 (Cont'd). For our running example of the 2×3-term polynomial multiplication over $K = \mathbb{F}_2$, the canonical basis of the vector space V is $(a_0b_0, a_0b_1, a_0b_2, a_1b_0, a_1b_1, a_1b_2)$. The target set is $\mathcal{T} = \{a_0b_0, a_0b_1 + a_1b_0, a_0b_2 + a_1b_1, a_1b_2\}$, and the set of generators $\mathcal{G}$ consists of the 21 products:

$$\begin{aligned} \mathcal{G} = \{\ & a_0b_0, && a_1b_0, && (a_0 + a_1)b_0, \\ & a_0b_1, && a_1b_1, && (a_0 + a_1)b_1, \\ & a_0(b_0 + b_1), && a_1(b_0 + b_1), && (a_0 + a_1)(b_0 + b_1), \\ & a_0b_2, && a_1b_2, && (a_0 + a_1)b_2, \\ & a_0(b_0 + b_2), && a_1(b_0 + b_2), && (a_0 + a_1)(b_0 + b_2), \\ & a_0(b_1 + b_2), && a_1(b_1 + b_2), && (a_0 + a_1)(b_1 + b_2), \\ & a_0(b_0 + b_1 + b_2), && a_1(b_0 + b_1 + b_2), && (a_0 + a_1)(b_0 + b_1 + b_2)\ \}. \end{aligned}$$

We can then rewrite $\mathcal{T}$ as the following matrix of 4 row vectors:

$$\begin{array}{c} \begin{matrix} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{matrix} \\ \mathcal{T} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} a_0b_0 \\ a_0b_1 + a_1b_0 \\ a_0b_2 + a_1b_1 \\ a_1b_2 \end{array}$$

The same applies to the set of generators $\mathcal{G}$, which gives the 21×6 matrix:

$$\mathcal{G} = \begin{array}{c} \begin{matrix} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{array}{l} a_0b_0 \\ a_1b_0 \\ (a_0+a_1)b_0 \\ \vdots \\ (a_0+a_1)(b_0+b_1+b_2) \end{array}$$

The bilinear rank problem is then stated as follows: For a given integer k, we want to find the subsets $\mathcal{W}$ of k elements of $\mathcal{G}$ such that the subspace W spanned by $\mathcal{W}$ contains $\mathcal{T}$. In linear algebra terms, we want to find the subsets of k rows of $\mathcal{G}$ whose K-linear span contains all the row vectors of $\mathcal{T}$, *i.e.*, contains the subspace $T = \mathrm{Span}(\mathcal{T})$.

4 Solving the Linear Algebra Problem

We recall the notations from Definition 1: V is the vector space of dimension nm spanned by the a_ib_j products, $\mathcal{G} \subset V$ is a finite set of generators, $\mathcal{T} \subseteq \mathrm{Span}(\mathcal{G})$ is the set of target vectors, and $T = \mathrm{Span}(\mathcal{T})$ is the corresponding target space spanned by the target vectors, which has dimension ℓ.

In the previous section, we reduced our problem of finding all formulae with k full multiplications for evaluating a given bilinear map to a linear algebra problem: given a finite subset $\mathcal{G}$ of a K-vector space V and a target subspace $T = \mathrm{Span}(\mathcal{T})$ of $\mathrm{Span}(\mathcal{G})$, we want to find all the rank-k subspaces W of V containing T and which can be generated by elements of $\mathcal{G}$ only (*i.e.*, $\mathrm{Span}(W \cap \mathcal{G}) = W$). One should note that this linear algebra problem is more general than the original problem of optimizing the computation of bilinear maps.

4.1 Naive Algorithm

A trivial approach to the linear algebra problem is to enumerate all the $\binom{\#\mathcal{G}}{k}$ subsets $\mathcal{W} = \{g_1, \ldots, g_k\}$ of $\mathcal{G}$ and, for each of them, test if its span covers T. The most efficient way to test this inclusion of spaces is to construct $W = \mathrm{Span}(\mathcal{W})$, represented by its basis as a matrix in row echelon form, and then test if each vector $t \in \mathcal{T}$ reduces to 0 against this matrix.

For simplicity's sake, in this paper we focus on the *combinatorial complexity* of the algorithms, considering that all matrix operations (*e.g.*, computing the dimension of a vector space, checking if a vector is in a vector space, computing the row-echelon form of a matrix) have constant complexity. The complexity of the naive algorithm is thus

$$\binom{\#\mathcal{G}}{k}.$$

Example 1 (Cont'd). For the 2×3-term polynomial multiplication, $\#\mathcal{G} = 21$, $k = 5$. We therefore have to test 20 349 subsets $\mathcal{W}$.

4.2 Improved Algorithm

The main drawback of the naive algorithm is that different subsets of generators $\mathcal{W}_i$ may be linearly dependent and span the same vector space $W = \mathrm{Span}(\mathcal{W}_i)$. For instance, for

the multiplication of two 3-term polynomials over $\mathbb{Q}$, Montgomery [16] gives the following family of solutions:

$$\begin{aligned}(a_0+a_1X+a_2X^2)(b_0+b_1X+b_2X^2) &= a_0b_0(C+1-X-X^2) \\ &+a_1b_1(C-X+X^2-X^3)+a_2b_2(C-X^2-X^3+X^4) \\ &+(a_0+a_1)(b_0+b_1)(-C+X)+(a_0+a_2)(b_0+b_2)(-C+X^2) \\ &+(a_1+a_2)(b_1+b_2)(-C+X^3)+(a_0+a_1+a_2)(b_0+b_1+b_2)C.\end{aligned} \tag{1}$$

Taking different values for the arbitrary polynomial $C \in \mathbb{Q}[X]$, we see that any subset of 6 out of the 7 generators $\mathcal{G} = \{a_0b_0, a_1b_1, a_2b_2, (a_0+a_1)(b_0+b_1), (a_0+a_2)(b_0+b_2), (a_1+a_2)(b_1+b_2), (a_0+a_1+a_2)(b_0+b_1+b_2)\}$ yields a solution. However, these 7 solutions correspond to the same vector space W, since the 7 generators are linearly dependent:

$$\begin{aligned}(a_0+a_1+a_2)(b_0+b_1+b_2) &= (a_0+a_1)(b_0+b_1)+(a_0+a_2)(b_0+b_2) \\ &+(a_1+a_2)(b_1+b_2)-a_0b_0-a_1b_1-a_2b_2.\end{aligned}$$

We propose an algorithm that takes advantage of this redundancy by looking directly for the subspaces W — instead of the subsets of generators — that cover our target space T. More formally, we search all the subspaces W of V such that:

- (i) $T \subset W$, *i.e.*, the target space T is covered by W;
- (ii) $\mathrm{Span}(W \cap \mathcal{G}) = W$, *i.e.*, W is spanned by elements of $\mathcal{G}$; and
- (iii) $\dim W = k$, *i.e.*, only k generators are needed.

A first remark is that from (i), the target space T is contained in each solution space W. Thus we should look for all the W's by extending this vector space. Unfortunately, there are a lot of spaces above T, possibly more than there are subsets $\mathcal{W}$ of $\mathcal{G}$. Instead, we might consider only those spaces W which are obtained by adding generators to T. This technique has already been used by Oseledets [18] in some heuristic algorithms. In order to use this idea without losing the exhaustiveness of our algorithm, we introduce a new condition:

$$(ii') \quad \exists \mathcal{W}' \subset \mathcal{G} \text{ such that } W = T \oplus \mathrm{Span}(\mathcal{W}').$$

Lemma 1. *All subspaces W that verify (i) and (ii) also verify (ii').*

Proof. Let W a subspace verifying (i) and (ii), the result follows directly from Proposition 1 with $\mathcal{H}$ being $W \cap \mathcal{G}$, and $\mathcal{F}$ being the free family $\mathcal{T} \subset W$. □

Proposition 1 ([2, Prop. 3.15]). *Let W be a vector space over a field K spanned by a finite set of generators $\mathcal{H}$. Let $\mathcal{F}$ be a free family of W. Then there exists a subset $\mathcal{J}$ of $\mathcal{H}$ such that $\mathcal{F} \cup \mathcal{J}$ is a basis of W.*

Proof. If $\mathcal{F}$ generates W then we take $\mathcal{J} = \emptyset$.

Let now assume that $\mathcal{F}$ does not span W. If we had $\mathcal{H} \subset \mathrm{Span}(\mathcal{F})$, then it would follow that $W = \mathrm{Span}(\mathcal{H}) \subset \mathrm{Span}(\mathcal{F})$, which contradicts the assumption. Therefore, there exists $h \in \mathcal{H}$ such that $h \notin \mathrm{Span}(\mathcal{F})$, and thus the family $\mathcal{F}' := \mathcal{F} \cup \{h\}$ is also free. We then iterate the process with $\mathcal{F}'$, until it eventually spans the whole vector space W. Since one cannot choose the same element $h \in \mathcal{H}$ twice and since $\mathcal{H}$ is finite, the process terminates. The set $\mathcal{J}$ is then composed of all the generators h added to $\mathcal{F}$ when constructing $\mathcal{F}'$. □

Algorithm 1 Find all the subspaces of dimension k generated by elements of $\mathcal{G}$ only and that contain the target space T.

Input: A vector space V, a finite set $\mathcal{G}$ of elements of V, a subspace T of V included in $\mathrm{Span}(\mathcal{G})$, and an integer k such that $\dim T \leq k \leq \mathrm{rk}(\mathcal{G})$.
Output: The set $\mathcal{S}$ of all subspaces W of V such that $T \subset W$, $\mathrm{Span}(W \cap \mathcal{G}) = W$, and $\dim W = k$.

```
1. S ← ∅
2. procedure expand_subspace(W)
3.     if dim W = k and rk(W ∩ G) = k then
4.         S ← S ∪ {W}
5.     else if dim W < k then
6.         for each g ∈ G \ W do
7.             expand_subspace(W ⊕ Span(g))
8. end procedure
9. expand_subspace(T)
10. return S
```

Thanks to Lemma 1, finding all subspaces W of V verifying conditions (i), (ii), and (iii) can be achieved by first enumerating all subsets $\mathcal{W}'$ of $\mathcal{G}$ such that $W = T \oplus \mathrm{Span}(\mathcal{W}')$ verifies conditions (i), (ii'), and (iii), and then keeping only those for which W also verifies condition (ii). This method is implemented in Algorithm 1, which is proven correct in Theorem 1.

Theorem 1 (Correctness of Algorithm 1). *For any given k, Algorithm 1 returns all the subspaces W of V verifying conditions (i), (ii) and (iii). In particular, when Algorithm 1 fails to find any solutions for all k under a bound k_0, then no solutions exist below this bound, and the bilinear rank is greater or equal to k_0.*

Proof. Let W be a subspace of V verifying (i), (ii) and (iii). We first prove that W is included in the output of Algorithm 1. By Lemma 1, W also satisfies (ii'). Thus there is a subset $\mathcal{W}' = \{g_1, ..., g_{k'}\}$ of $\mathcal{G}$ such that $W = T \oplus \mathrm{Span}(g_1, ..., g_{k'})$, and we can choose $g_1, ..., g_{k'}$ to be linearly independent. Therefore, Algorithm 1 will, at some point in the enumeration, consider the candidate subspace W. Since $\dim W = k$ and, by condition (ii) we have $\mathrm{rk}(W \cap \mathcal{G}) = \dim W = k$, Algorithm 1 will include W in $\mathcal{S}$.

Conversely, let W in the output of Algorithm 1; let us show that conditions (i), (ii), and (iii) are fulfilled. Condition (i) is trivially verified, since Algorithm 1 constructs W by expanding the initial subspace T. Furthermore, since W is in $\mathcal{S}$, we have $\dim W = k$, thus (iii) is also verified. Finally, we have $\mathrm{rk}(W \cap \mathcal{G}) = k$, which implies $\mathrm{Span}(W \cap \mathcal{G}) = W$, and thus W also fulfills condition (ii). □

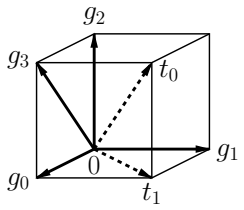
Example 1 (Cont'd). For our running example, Algorithm 1 immediately shows that no solutions exist with $k = 4$ over $\mathbb{F}_2$. Indeed, we have $\dim T = 4$, thus $\dim W = 4$ at the very first call of `expand_subspace`, but the rank of $W \cap \mathcal{G}$ is only 3.

Example 5 (Step-by-step illustration of Algorithm 1). Take $K = \mathbb{Q}$, $V = \mathbb{Q}^3$, $\mathcal{T} = \{t_0, t_1\}$ with $t_0 = (1, 1, 1)$ and $t_1 = (1, 1, 0)$; and $\mathcal{G} = \{g_0, g_1, g_2, g_3\}$ with $g_0 = (1, 0, 0)$, $g_1 = (0, 1, 0)$, $g_2 = (0, 0, 1)$, and $g_3 = (1, 0, 1)$, as depicted in Figure 1. In this example, for clarity's sake, we denote the recursive depth of the current call by a parenthesized superscript: *e.g.*, $W^{(i)}$ represents the subspace W at recursive depth i, starting at $i = 0$.

Let us first consider the case $k = 2$. The algorithm initializes $W^{(0)}$ to $T = \mathrm{Span}(t_0, t_1)$, which is already of dimension 2. It then computes $W \cap \mathcal{G} = \{g_2\}$, which is of rank 1. Therefore, there are no solutions for $k = 2$.

Consider now the case $k = 3$. Starting again with $W^{(0)} = T$, since $\dim T = 2$, the algorithm will then have to expand $W^{(0)}$ by adding a linear independent vector g from

Fig. 1. The sets $\mathcal{G}$ and $\mathcal{T}$ from Example 5.



$\mathcal{G} \setminus W^{(0)} = \{g_0, g_1, g_3\}$. Taking $g = g_0$ and adding it to $W^{(0)}$, the algorithm recurses down with $W^{(1)} = T \oplus \mathrm{Span}(g_0)$. We have $W^{(1)} \cap \mathcal{G} = \{g_0, g_1, g_2, g_3\}$, which has rank 3, meaning that this $W^{(1)}$ is a solution subspace, which is then added to the set $\mathcal{S}$. The algorithm can now reiterate the process for the subspaces $W^{(0)} \oplus \mathrm{Span}(g_1)$ and $W^{(0)} \oplus \mathrm{Span}(g_3)$, finding both of them to also be solutions.

Note that there is no point to try the algorithm for $k > 3$: since $\dim V = 3$, the algorithm will never find subspaces W of V of dimension k.

Remark also that this simple example illustrates *rank leaps* for $W \cap \mathcal{G}$: if $\mathrm{rk}(W^{(0)} \cap \mathcal{G}) = 1$, when taking $W^{(1)} = W^{(0)} \oplus \mathrm{Span}(g_0)$, the rank of $W^{(1)} \cap \mathcal{G}$ jumps directly to 3. In fact, since T cannot usually be generated by elements of $\mathcal{G}$ only, we have $\dim W^{(0)} > \mathrm{rk}(W^{(0)} \cap \mathcal{G})$ at the beginning of the algorithm. Therefore, since $\dim W$ is only incremented by 1 at each recursive call, rank leaps between $W^{(i)} \cap \mathcal{G}$ and $W^{(i+1)} \cap \mathcal{G}$ gradually close the gap between $\dim W$ and $\mathrm{rk}(W \cap \mathcal{G})$.

Example 6 (Evaluation–interpolation schemes). In the case of the n-term polynomial multiplication over a field K, evaluating the resulting polynomial at a point $\kappa \in K$ only involves a linear combination of the target coefficients t_i. Therefore, the rank-1 bilinear forms $(\sum_i \kappa^i a_i)(\sum_j \kappa^j b_j)$ corresponding to the evaluation of the two input polynomials at κ are also in T. This is also the case when evaluating "at infinity", *i.e.*, considering the leading coefficient $a_{n-1}b_{n-1}$. Therefore, since these rank-1 bilinear forms are linearly independent (by Vandermonde determinant), if $\#K \geq 2n - 2$, then we have $\mathrm{rk}(T \cap \mathcal{G}) = 2n - 1 = \ell$, and T itself is a solution subspace.

4.3 Implementation Issues

Algorithm 1 relies extensively on operations on vector spaces, such as computing the rank of the intersection $W \cap \mathcal{G}$ (line 3), or excluding elements of W from $\mathcal{G}$ (line 6). In order to perform these operations efficiently, each subspace W is represented in the algorithm by the $r \times nm$ matrix of its basis in row echelon form, where $r = \dim W$. Note that we also denote by W the matrix in row echelon form corresponding to the subspace W.

Computing $W_{\mathcal{G}} = \mathrm{Span}(W \cap \mathcal{G})$ is done as follows: first reduce every vector $g \in \mathcal{G}$ by the matrix W; if g reduces to the null vector, then $g \in W$. For each such g, reduce it by the current matrix $W_{\mathcal{G}}$ (initially empty), and if the reduced vector is not null, add it to $W_{\mathcal{G}}$. Once $W_{\mathcal{G}}$ is constructed, the rank of $W \cap \mathcal{G}$ is then given by the number of rows of the matrix.

Enumerating all the generators $g \in \mathcal{G} \setminus W$ (line 6) can be achieved using a similar technique. However, it is more efficient to maintain the set $\mathcal{H}$ of all generators $g \in \mathcal{G}$ reduced by W, and pick only from these reduced generators for the recursive calls. Indeed, it might very well be the case that two generators g and $g' \in \mathcal{G}$ are in fact the same vector after reduction by W. In that case, the two subspaces $W \oplus \mathrm{Span}(g)$ and $W \oplus \mathrm{Span}(g')$

are identical, and recursing down the algorithm for both of them would just be a waste of time. Maintaining a set $\mathcal{H}$ of reduced generators prevents this to happen, as g and g' will then correspond to a single reduced generator in $\mathcal{H}$. Furthermore, at recursion depth i, since all elements g of $\mathcal{H}^{(i)}$ are reduced by $W^{(i)}$, when recursing down the algorithm:

- constructing $W^{(i+1)} = W^{(i)} \oplus \mathrm{Span}(g)$ only requires inserting the row vector g into the matrix $W^{(i)}$; and
- reducing each element $h \in \mathcal{H}^{(i)}$ by $W^{(i+1)}$ to construct $\mathcal{H}^{(i+1)}$ only requires reducing h by $\mathrm{Span}(g)$, as it is already reduced by $W^{(i)}$.

Example 1 (Cont'd). On our running example, $T = \mathrm{Span}(\mathcal{T})$ is the 4×6 matrix — which is already in row echelon form — and $\mathcal{G}$ consists of the 21 generators from Section 3. Let us follow the execution of the algorithm in the case $k = 5$.

We first construct $\mathcal{H}^{(0)}$ by reducing all generators of $\mathcal{G}$ by $W^{(0)} = T$. Only 3 unique reduced generators remain:

$$\mathcal{H}^{(0)} = \begin{array}{c} \begin{matrix} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{l} a_1b_0 \\ a_1b_1 \\ a_1(b_0 + b_1) \end{array}$$

Since $\dim W^{(0)} = 4 < k$, these generators in $\mathcal{H}^{(0)}$ are then enumerated, and the procedure `expand_subspace` is called recursively for each of them.

For instance, when taking $g = a_1b_0$, the subspace $W^{(1)}$ is constructed by inserting g into the matrix $W^{(0)}$:

$$W^{(1)} = \begin{array}{c} \begin{matrix} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} a_0b_0 \\ a_0b_1 + a_1b_0 \\ a_0b_2 + a_1b_1 \\ g = a_1b_0 \\ a_1b_2 \end{array}$$

Computing $W^{(1)} \cap \mathcal{G}$ yields a set of 9 generators and of rank 5; $W^{(1)}$ is thus a solution subspace.

Finally, continuing the enumeration for $g = a_1b_1$ then $g = a_1(b_0 + b_1)$, the algorithm will find two other solution subspaces of rank 5.

4.4 From Solution Subspaces back to Formulae

Given a solution subspace W verifying conditions (i), (ii), and (iii), we still need to retrieve the corresponding formulae. Note that since $T \subset W$, each formula corresponds to a basis of W using only generators from $\mathcal{G}$. A simple solution is thus to first compute the set $W \cap \mathcal{G}$, then enumerate all subsets of k linearly independent vectors in $W \cap \mathcal{G}$.

For every basis of W, obtaining the corresponding formula is then just a matter of finding the coordinates of the vectors of $\mathcal{T}$ in this basis, which will give us the corresponding linear combinations of generators required to compute them.

Example 1 (Cont'd). In our running example, each of the 3 solution subspaces yields 54 different 5-subsets of $\mathcal{G}$ that span T. We thus find a total of 162 formulae for evaluating 2×3-term polynomial multiplications over binary fields.

4.5 Complexity Analysis

Algorithm 1 allows us to drastically reduce the number of operations required to find all the formulae for evaluating a given bilinear map thanks to the two following key ideas:

1. searching for subspaces W of $\mathrm{Span}(\mathcal{G})$ instead of subsets $\mathcal{W}$ of $\mathcal{G}$; and
2. constructing these subspaces starting from T instead of $\{0\}$.

The algorithm consists in expanding T to a vector space of dimension k by adding $k - \ell$ vectors from $\mathcal{G}$, where $\ell = \dim T$. The combinatorial complexity of the algorithm is thus bounded by:

$$\binom{\#\mathcal{G}}{k-\ell}.$$

Note however that this complexity bound does not reflect the gain due to the first idea — searching for subspaces instead of subsets — since that gain depends on the particular problem to be solved, and cannot be expressed simply. In Algorithm 1, once say $g_1, g_2, \ldots$ have been added to the initial $W^{(0)} = T$, this idea will prune the remaining set of independent generators $\mathcal{G} \backslash W$ (without this idea, instead of considering all $g \in \mathcal{G} \backslash W$ at line 6 of Algorithm 1, we would consider all $g \in \mathcal{G}$, which would be very similar to the naive algorithm). Note however that Algorithm 1 does not guarantee that all subspaces W explored and found to be solutions are different. Duplicates have to be detected and removed upon adding solution subspaces to the set $\mathcal{S}$.

In practice, this combinatorial complexity is usually not attained, as can be seen in the experimental results reported in Section 5.

Example 1 (Cont'd). For the 2×3-term polynomial multiplication over $\mathbb{F}_2$, with $\#\mathcal{G} = 21$, $k = 5$, and $\ell = 4$, the complexity estimate predicts 21 subspaces W to consider, whereas in practice, only 3 of these subspaces need be explored by the algorithm. This has to be compared with the 20 349 subsets explored by the naive algorithm.

4.6 Special Case of Symmetric Bilinear Maps

An $n \times n$ bilinear form $\gamma : \mathbf{a}, \mathbf{b} \mapsto \sum_{i,j} \gamma_{i,j} a_i b_j$ is said to be *symmetric* if $\gamma_{i,j} = \gamma_{j,i}$ for all $0 \leq i, j < n$. Similarly, an $n \times n$ bilinear map ϕ is said to be *symmetric* if the bilinear forms corresponding to its coefficients are all symmetric. For instance, it is the case for the n-term polynomial multiplication.

Example 7 (3-term polynomial multiplication). Given $A = a_2X^2 + a_1X + a_0$ and $B = b_2X^2 + b_1X + b_0 \in K[X]$, the target set for the product $A \cdot B$ is

$$\mathcal{T} = \{a_0b_0, a_0b_1 + a_1b_0, a_0b_2 + a_1b_1 + a_2b_0, a_1b_2 + a_2b_1, a_2b_2\},$$

all of whose bilinear forms are symmetric.

In the case where the target set $\mathcal{T}$ is composed only of symmetric forms, an idea to accelerate the search for formulae is to restrict $\mathcal{G}$ to be the set of rank-1 symmetric bilinear forms, which are of the form $(\sum_i \alpha_i a_i)(\sum_j \alpha_j b_j)$. Indeed, while there are $(\#K)^{n^2} - 1$ non-zero bilinear forms of rank 1 in $\mathcal{B}(K^n, K^n; K)$, only $(\#K)^n - 1$ of them are symmetric, which yields a huge speedup.

However, the symmetric rank-1 forms alone may fail to produce optimal formulae for symmetric maps: for instance, $a_0b_1 + a_1b_0$ cannot be computed with 2 symmetric products over $\mathbb{F}_2$. (For an introduction on the topic of decomposing symmetric maps as sums of symmetric tensors see for instance [10].) Therefore, restricting to symmetric forms has to be considered as a heuristic, and cannot be used to prove lower bounds on the bilinear rank of symmetric maps.

5 Experimental Results

We present here some results obtained for various instances of the bilinear rank problem using a multi-threaded C implementation of Algorithm 1. The reported timings correspond to the total execution time on a single core of a 2.2 GHz Xeon L5640. We reproduce or improve some known results [7,9,16] but also find the complete list of solutions for each problem or prove lower bounds thanks to Theorem 1.

In all the following tables, only one value for k is reported for each problem. It is the smallest possible value for which solutions were found or, if no solutions were found, it is the largest value which was tried with our algorithm. In both cases, it means that there are no solutions for smaller values of k. Rows without a value for k were deemed too computationally expensive and were not tried at all with our implementation.

Under the heading "# of tests" are reported the number of subspaces W for which the algorithm had to check whether $\mathrm{rk}(W \cap \mathcal{G}) = k$ or not. It corresponds to the number of leaves in the tree of recursive calls. The column "# of solutions" reports the number $\#\mathcal{S}$ of solution subspaces returned by Algorithm 1. Finally, we indicate in the column "# of formulae" the corresponding number of formulae, or ∞ when this number is really large.

Finally, where applicable and when an exhaustive search using all rank-1 bilinear forms for $\mathcal{G}$ was too expensive, we enumerated subspaces using only symmetric generators. This is indicated by a "(Sym.)" mention next to the cardinality of $\mathcal{G}$.

Thus, Table 1 gives experimental results for $n \times m$-term polynomial multiplication over $\mathbb{F}_2$. For instance, the 6×6 row indicates that there are no formulae with only $k = 14$ full multiplications over $\mathbb{F}_2$; however, symmetric formulae exist for $k = 17$. Similarly, the 5×5 row shows that Montgomery's bound of $k = 13$ is optimal over $\mathbb{F}_2$ [16]. Table 2 gives similar results over $\mathbb{F}_3$. For example, the 5×5 row with no solution for $k = 11$ proves that the bound $M_3(5) = 12$ from [6, Table 2] is optimal.

Table 3 corresponds to small extensions of $\mathbb{F}_2$ or $\mathbb{F}_3$, where we consider the multiplication of two elements in polynomial basis. This is relevant for example for multiplication of matrices over $\mathbb{F}_{2^e}$ [1]. In particular the rows $\mathbb{F}_{2^4}$ with $k = 9$ and $\mathbb{F}_{3^4}$ with $k = 6$ confirm the values from [7, Table 1].

Furthermore, according to the literature, the best known formula for computing the product of two elements over the finite field $\mathbb{F}_{3^5}$ uses 12 full multiplications [7]. As indicated in Table 3, our algorithm found 121 symmetric formulae using only 11 full multiplications. One such formula is given in Algorithm 2.

Finally, Table 4 considers the product of two n-term polynomials modulo X^n and $X^n - 1$ over $\mathbb{F}_2$ — as in [18] — and also over $\mathbb{F}_3$. This proves the tensor rank from [18] is the optimal one for $n = 2, 3, 4$ over $\mathbb{F}_2$ modulo both X^n and $X^n - 1$. One should note that over $\mathbb{F}_3$, computing a product modulo $X^4 - 1$ can be done with the same number of products (5) than modulo $X^3 - 1$.

6 Conclusion

Long considered to be "magical", Karatsuba's and Strassen's formulae can be understood in a unified framework: the bilinear rank problem. Small instances of this NP-hard problem can be tackled by exhaustive search methods, as proposed by Montgomery [16]. Our improved algorithm proved that many known formulae from the literature, some of which discovered by ad-hoc methods, are optimal. Not relying on heuristic methods, the algorithm we presented here finds *all* the formulae using the optimal number of full multiplications, and gives lower bounds. For example, for the product of 3-term polynomials

Table 1. Experimental results for $n \times m$ polynomial multiplication over $\mathbb{F}_2$.

$n \times m$	$\#\mathcal{G}$	k	# of tests	# of solutions	# of formulae	Calculation time [s]
2×2	9	3	1	1	1	0.00
3×2	21	5	2	3	162	0.00
3×3	49	6	9	3	9	0.00
4×2	45	6	5	4	108	0.00
4×3	105	8	700	33	423	0.00
4×4	225	9	$6.60 \cdot 10^3$	4	4	0.03
5×2	93	8	56	28	790 272	0.00
5×3	217	10	$1.46 \cdot 10^5$	366	48 195	0.51
5×4	465	12	$3.13 \cdot 10^8$	4 113	66 153	$2.86 \cdot 10^3$
5×5	961	13	$9.65 \cdot 10^9$	27	27	$2.28 \cdot 10^5$
6×2	189	9	250	64	1 404 928	0.00
6×3	441	11	$2.05 \cdot 10^6$	3	243	11.5
6×4	945	13	$7.69 \cdot 10^9$	9	15	$1.62 \cdot 10^5$
6×5	1 953	14	$2.01 \cdot 10^{11}$	—	—	$9.97 \cdot 10^6$
6×6	3 969	14	$4.37 \cdot 10^9$	—	—	$6.03 \cdot 10^5$
	(Sym.) 63	17	$8.08 \cdot 10^6$	6	54	17.7
7×2	381	11	$9.14 \cdot 10^3$	960	∞	0.07
7×3	889	13	$2.52 \cdot 10^9$	87	63 423	$3.66 \cdot 10^4$
7×4	1 905	14	$1.47 \cdot 10^{11}$	—	—	$6.34 \cdot 10^6$
7×7	(Sym.) 127	22	$3.38 \cdot 10^{12}$	2 618	19 550	$1.59 \cdot 10^7$
8×2	765	12	$7.80 \cdot 10^4$	4 096	∞	0.75
8×3	1 785	14	$5.27 \cdot 10^{10}$	—	—	—

Table 2. Experimental results for $n \times m$ polynomial multiplication over $\mathbb{F}_3$.

$n \times m$	$\#\mathcal{G}$	k	# of tests	# of solutions	# of formulae	Calculation time [s]
2×2	16	3	1	1	4	0.00
3×2	52	4	1	1	1	0.00
3×3	169	6	24	22	1 493	0.00
4×2	160	6	9	13	38 880	0.00
4×3	520	7	164	12	48	0.00
4×4	1 600	9	$4.11 \cdot 10^5$	726	50 640	14.9
5×2	484	7	24	36	93 312	0.00
5×3	1 573	9	$2.81 \cdot 10^5$	1 116	94 629	9.33
5×4	4 840	10	$4.75 \cdot 10^6$	48	768	$1.01 \cdot 10^3$
5×5	14 641	11	$4.89 \cdot 10^7$	—	—	$4.02 \cdot 10^4$
	(Sym.) 121	12	$3.93 \cdot 10^4$	31	6 460	0.14
6×2	1 456	8	69	81	104 976	0.01
6×3	4 732	10	$3.24 \cdot 10^6$	240	4 272	566
6×4	14 560	11	$4.55 \cdot 10^7$	—	—	$3.31 \cdot 10^4$
6×5	44 044	12	$4.58 \cdot 10^8$	—	—	$1.31 \cdot 10^6$
6×6	(Sym.) 364	15	$2.37 \cdot 10^8$	4	1 024	$3.79 \cdot 10^3$
7×2	4 372	10	$2.27 \cdot 10^4$	10 530	∞	2.84
7×3	14 209	11	$3.15 \cdot 10^7$	—	—	$1.84 \cdot 10^4$
7×4	43 720	12	$4.16 \cdot 10^8$	—	—	$1.03 \cdot 10^6$
7×7	(Sym.) 1 093	17	$2.69 \cdot 10^{10}$	—	—	$1.50 \cdot 10^6$
8×2	13 120	11	$2.01 \cdot 10^5$	85 293	∞	53.6
8×3	42 640	12	$2.90 \cdot 10^8$	—	—	$5.46 \cdot 10^5$

Table 3. Experimental results for multiplication over small extensions of $\mathbb{F}_2$ and $\mathbb{F}_3$.

Finite field	$\#\mathcal{G}$	k	# of tests	# of solutions	# of formulae	Calculation time [s]
$\mathbb{F}_{2^2}$	9	3	3	3	3	0.00
$\mathbb{F}_{2^3}$	49	6	$7.03 \cdot 10^3$	105	147	0.01
$\mathbb{F}_{2^4}$	225	9	$2.57 \cdot 10^9$	2 025	2 025	$1.13 \cdot 10^4$
$\mathbb{F}_{2^5}$	961	9	$3.10 \cdot 10^{10}$	—	—	$8.11 \cdot 10^5$
	(Sym.) 31	13	$3.49 \cdot 10^6$	2 015	2 015	6.24
$\mathbb{F}_{2^6}$	(Sym.) 63	15	$2.21 \cdot 10^{10}$	21	21	$6.63 \cdot 10^4$
$\mathbb{F}_{2^7}$	(Sym.) 127	15	$1.34 \cdot 10^{12}$	—	—	$6.17 \cdot 10^6$
$\mathbb{F}_{3^2}$	16	3	3	4	16	0.00
$\mathbb{F}_{3^3}$	169	6	$2.42 \cdot 10^5$	11 843	105 963	1.08
$\mathbb{F}_{3^4}$	1 600	8	$2.27 \cdot 10^{11}$	—	—	$1.08 \cdot 10^7$
	(Sym.) 40	9	$1.10 \cdot 10^5$	234	615 240	0.45
$\mathbb{F}_{3^5}$	(Sym.) 121	11	$2.66 \cdot 10^9$	121	121	$1.45 \cdot 10^4$
$\mathbb{F}_{3^6}$	(Sym.) 364	12	$3.01 \cdot 10^{12}$	—	—	$4.50 \cdot 10^7$

Table 4. Experimental results for the multiplication of two n-term polynomials in $\mathbb{F}_p[X]/(X^n)$ and $\mathbb{F}_p[X]/(X^n - 1)$, with $p = 2$ and 3.

Ring	n	$\#\mathcal{G}$	k	# of tests	# of solutions	# of formulae	Calculation time [s]
$\mathbb{F}_2[X]/(X^n)$	2	9	3	3	3	10	0.00
	3	49	5	590	12	40	0.00
	4	225	8	$5.17 \cdot 10^7$	1 440	9 248	230
	5	961	9	$2.66 \cdot 10^{10}$	—	—	$6.70 \cdot 10^5$
		(Sym.) 31	11	$3.64 \cdot 10^5$	112	736	0.48
	6	(Sym.) 63	14	$2.63 \cdot 10^9$	384	2 816	$7.66 \cdot 10^3$
	7	(Sym.) 127	15	$1.16 \cdot 10^{12}$	—	—	$5.46 \cdot 10^6$
$\mathbb{F}_2[X]/(X^n - 1)$	2	9	3	3	3	10	0.00
	3	49	4	21	3	3	0.00
	4	225	8	$2.69 \cdot 10^7$	1 440	9 248	124
	5	961	9	$1.39 \cdot 10^{10}$	—	—	$3.65 \cdot 10^5$
		(Sym.) 31	10	$7.46 \cdot 10^4$	25	25	0.09
	6	(Sym.) 63	12	$2.33 \cdot 10^7$	31	148	50.0
	7	(Sym.) 127	13	$2.55 \cdot 10^9$	1	49	$1.24 \cdot 10^4$
$\mathbb{F}_3[X]/(X^n)$	2	16	3	4	4	39	0.00
	3	169	5	$7.94 \cdot 10^3$	90	1 539	0.07
	4	1 600	7	$5.54 \cdot 10^8$	—	—	$3.22 \cdot 10^4$
		(Sym.) 40	8	$3.17 \cdot 10^5$	252	40 095	0.14
	5	(Sym.) 121	10	$1.45 \cdot 10^8$	243	13 122	$2.28 \cdot 10^3$
	6	(Sym.) 364	11	$4.79 \cdot 10^{10}$	—	—	$8.22 \cdot 10^5$
$\mathbb{F}_3[X]/(X^n - 1)$	2	16	2	1	1	1	0.00
	3	169	5	$4.45 \cdot 10^3$	90	1 539	0.04
	4	1 600	5	767	4	16	0.07
	5	(Sym.) 121	10	$8.74 \cdot 10^7$	234	615 240	$1.39 \cdot 10^3$
	6	(Sym.) 364	10	$3.37 \cdot 10^8$	9	2 025	$1.68 \cdot 10^4$

Algorithm 2 Multiplication over $\mathbb{F}_{3^5} \cong \mathbb{F}_3[X]/(X^5 - X + 1)$.

Input: $A = a_4X^4 + a_3X^3 + a_2X^2 + a_1X + a_0 \in \mathbb{F}_{3^5}$ and $B = b_4X^4 + b_3X^3 + b_2X^2 + b_1X + b_0 \in \mathbb{F}_{3^5}$.
Output: $A \cdot B \bmod (X^5 - X + 1) = t_4X^4 + t_3X^3 + t_2X^2 + t_1X + t_0 \in \mathbb{F}_{3^5}$.

1. $g_0 \leftarrow a_4b_4$
2. $g_1 \leftarrow (a_0 + a_2)(b_0 + b_2)$
3. $g_2 \leftarrow (a_0 - a_3)(b_0 - b_3)$
4. $g_3 \leftarrow (a_1 - a_4)(b_1 - b_4)$
5. $g_4 \leftarrow (a_3 - a_4)(b_3 - b_4)$
6. $g_5 \leftarrow (a_0 + a_1 - a_2)(b_0 + b_1 - b_2)$
7. $g_6 \leftarrow (a_1 - a_3 + a_4)(b_1 - b_3 + b_4)$
8. $g_7 \leftarrow (a_2 + a_3 - a_4)(b_2 + b_3 - b_4)$
9. $g_8 \leftarrow (a_0 + a_1 - a_2 - a_3)(b_0 + b_1 - b_2 - b_3)$
10. $g_9 \leftarrow (a_0 + a_1 - a_3 + a_4)(b_0 + b_1 - b_3 + b_4)$
11. $g_{10} \leftarrow (a_1 + a_2 - a_3 - a_4)(b_1 + b_2 - b_3 - b_4)$
12. $t_0 \leftarrow g_0 + g_2 - g_3 - g_4 + g_5 + g_6 - g_8$
13. $t_1 \leftarrow g_1 + g_2 + g_3 + g_4 - g_5 - g_8 + g_{10}$
14. $t_2 \leftarrow g_1 - g_2 - g_3 - g_5 - g_6 - g_7 + g_8$
15. $t_3 \leftarrow -g_0 - g_3 + g_4 + g_5 - g_7 - g_8 + g_{10}$
16. $t_4 \leftarrow -g_1 + g_2 - g_4 + g_5 + g_6 + g_7 + g_8 + g_9 - g_{10}$
17. **return** $t_4X^4 + t_3X^3 + t_2X^2 + t_1X + t_0$

over $\mathbb{F}_2$, in addition to the 7 possible formulae from Eq. (1) already found by Montgomery in [16], we found two new asymmetric formulae, the first one using the generators

$$g_1 := a_0b_0,\ g_2 := a_2b_2,\ g_3 := (a_0 + a_1)(b_0 + b_2),\ g_4 := (a_0 + a_2)(b_1 + b_2),$$
$$g_5 := (a_1 + a_2)(b_0 + b_1),\ g_6 := (a_0 + a_1 + a_2)(b_0 + b_1 + b_2),$$

with $(a_0 + a_1X + a_2X^2)(b_0 + b_1X + b_2X^2)$ being equal to:

$$g_1 + (g_2 + g_3 + g_5 + g_6)X + (g_3 + g_4 + g_6)X^2 + (g_1 + g_4 + g_5 + g_6)X^3 + g_2X^4,$$

and the second one using the generators

$$a_0b_0, a_2b_2, (a_0+a_1)(b_1+b_2), (a_0+a_2)(b_0+b_1), (a_1+a_2)(b_0+b_2), (a_0+a_1+a_2)(b_0+b_1+b_2).$$

It would be interesting to give a simple mathematical explanation for these new formulae.

We were also able to improve the bound from [8] for the multiplication in $\mathbb{F}_{3^5}$ from 12 to 11 multiplications, using again a completely generic method.

If one wants to go further, one has to either speed up the computations or to use more heuristic or proven techniques borrowed to the algebraic complexity community: restricting the set $\mathcal{G}$ of products, *e.g.*, to symmetric products for polynomial multiplication, enlarging the target set $\mathcal{T}$ by imposing that some products have to occur in the formula or using the symmetries of the problem.

Acknowledgements. The authors would like to thank Marc Mezzarobba for the interesting and fruitful discussions we had on the subject, especially for his knowledge of the many results on this topic.

References

1. Albrecht, M.R.: The M4RIE library for dense linear algebra over small fields with even characteristic. `http://arxiv.org/abs/1111.6900` (2011), preprint
2. Artin, M.: Algebra. Prentice-Hall, Inc. (1991)
3. Bläser, M.: On the complexity of the multiplication of matrices of small formats. Journal of Complexity 19, 43–60 (2003)
4. Bürgisser, P., Clausen, M., Shokrollahi, M.: Algebraic complexity theory, vol. 315. Springer Verlag (1997)
5. Cenk, M., Ozbudak, F.: Improved polynomial multiplication formulas over $\mathbb{F}_2$ using Chinese remainder theorem. IEEE Trans. Comput. 58(4), 572–576 (2009)

6. Cenk, M., Özbudak, F.: Efficient multiplication in $\mathbb{F}_{3^{\ell m}}$, $m \geq 1$ and $5 \leq \ell \leq 18$. In: Vaudenay, S. (ed.) Proc. AFRICACRYPT 2008. Lecture Notes in Comput. Sci., vol. 5023, pp. 406–414 (2008)
7. Cenk, M., Özbudak, F.: On multiplication in finite fields. J. Complexity 26, 172–186 (2010)
8. Cenk, M., Özbudak, F.: Multiplication of polynomials modulo x^n. Theoret. Comput. Sci. 412, 3451–3462 (2011)
9. Chung, J., Hasan, M.A.: Asymmetric squaring formulae. In: Kornerup, P., Muller, J.M. (eds.) Proc. ARITH 18. pp. 113–122 (2007)
10. Comon, P., Golub, G., Lim, L., Mourrain, B.: Symmetric tensors and symmetric tensor rank. SIAM J. Matrix Anal. & Appl. 30(3), 1254–1279 (2008)
11. Courtois, N.T., Bard, G.V., Hulme, D.: A new general-purpose method to multiply 3×3 matrices using only 23 multiplications. `http://arxiv.org/abs/1108.2830` (2011), preprint
12. Fan, H., Hasan, A.: Comments on five, six, and seven-term Karatsuba-like formulae. IEEE Trans. Comput. 56(5), 716–717 (2007)
13. Hanrot, G., Quercia, M., Zimmermann, P.: The middle product algorithm, I. Speeding up the division and square root of power series. Appl. Algebra Engrg. Comm. Comput. 14(6), 415–438 (2004)
14. Hastad, J.: Tensor rank is NP-complete. J. Algorithms 11(4), 644–654 (1990)
15. Karatsuba, A.A., Ofman, Y.: Multiplication of multi-digit numbers on automata (in Russian). Doklady Akad. Nauk SSSR 145(2), 293–294 (1962), translation in *Soviet Physics-Doklady* 7, 595–596 (1963)
16. Montgomery, P.: Five, six, and seven-term Karatsuba-like formulae. IEEE Trans. Comput. 54(3), 362–369 (2005)
17. Mulders, T.: On short multiplications and divisions. Appl. Algebra Engrg. Comm. Comput. 11(1), 69–88 (2000)
18. Oseledets, I.: Optimal Karatsuba-like formulae for certain bilinear forms in GF(2). Linear Algebra and its Applications 429, 2052–2066 (2008)
19. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik 13(4), 354–356 (1969)
20. Toom, A.: The complexity of a scheme of functional elements realizing the multiplication of integers. In: Soviet Mathematics Doklady. vol. 3, pp. 714–716 (1963)

Appendix

We compare here Algorithm 1 to Oseledets' work. The main similarity is that both methods use a linear algebra setting and search for solution spaces W rather than solution sets $\mathcal{W}$ of generators. In terms of differences, we note that Algorithm 1 is proven, making it a tool for proving lower bounds (cf. Section 5). Oseledets' work presents a main algorithm called Minbas, a subroutine called Rank-1 and some more tricks.

Minbas starts by computing the rank of all $\#K^{\dim T} - 1$ elements of T (denoted C in [18]), where the rank of a bilinear form is the minimal k such that $t = g_1 + g_2 + \cdots + g_k$ with $g_i \in \mathcal{G}$. Minbas continues by constructing a basis of T by adding in order linearly independent elements of rank 1, 2 and so on. In our setting, the rank-1 elements from [18] are elements of the target space which are also in the set $\mathcal{G}$ of generators. Since Algorithm 1 starts by $W = T$, those forms are automatically included in our algorithm. From this point on, the two algorithms diverge since Algorithm 1 explores spaces W of higher dimension whereas Minbas adds forms of rank 2 or more.

The second algorithm of Oseledets' work is called Rank-1 and can be used as a subroutine in algorithms solving the bilinear rank problem. Given a candidate space W of dimension k lying above T, the Rank-1 procedure tests if W contains a basis made out of elements in $\mathcal{G}$ by writing the lines of W and $\mathcal{G}$ in row echelon form. In our setting this is equivalent to the test $\mathrm{rk}(W \cap \mathcal{G}) = k$.

Without giving a general description of the method, nor analyzing its correctness, Oseledets implements the idea which we presented in Section 4.2. For example, in the problem of 5-term polynomial multiplication we have $\dim T = 9$ and we search solution spaces of dimension $k = 13$; Oseledets notes that "we have to add 3 [there is a typo here, read 4] matrices to the current basis. There are $31 - 3 = 28$ possible symmetric rank-1 matrices to add". The implicit idea is that we restrict our search to spaces which can be

written as $W = T \oplus \langle g_1, ..., g_{k-\dim T} \rangle$. One recognizes the idea of replacing condition (ii) by (ii') in Section 4.2. Oseledets did not realize this change still performs an exhaustive search, as demonstrated by Theorem 1, since he says on page 2061 "Partial exhaustive search – search only for the complement space C of R".