Fast, Algebraic Multivariate Multipoint Evaluation in Small Characteristic and Applications

Vishwas Bhargava[*]
Computer Science, Rutgers University
USA
vishwas1384@gmail.com

Sumanta Ghosh[†]
Computing and Mathematical Sciences, Caltech
USA
besusumanta@gmail.com

Mrinal Kumar
Computer Science & Engineering, IIT Bombay
India
mrinal@cse.iitb.ac.in

Chandra Kanta Mohapatra
Computer Science & Engineering, IIT Bombay
India
ckm@cse.iitb.ac.in

ABSTRACT

Multipoint evaluation is the computational task of evaluating a polynomial given as a list of coefficients at a given set of inputs. Besides being a natural and fundamental question in computer algebra on its own, fast algorithms for this problem are also closely related to fast algorithms for other natural algebraic questions like polynomial factorization and modular composition. And while *nearly linear time* algorithms have been known for the univariate instance of multipoint evaluation for close to five decades due to a work of Borodin and Moenck, fast algorithms for the multivariate version have been much harder to come by. In a significant improvement to the state of art for this problem, Umans and Kedlaya & Umans gave nearly linear time algorithms for this problem over field of small characteristic and over all finite fields respectively, provided that the number of variables n is at most $d^{o(1)}$ where the degree of the input polynomial in every variable is less than d. They also stated the question of designing fast algorithms for the large variable case (i.e. $n \notin d^{o(1)}$) as an open problem.

In this work, we show that there is a deterministic algorithm for multivariate multipoint evaluation over a field $\mathbb{F}_q$ of characteristic p which evaluates an n-variate polynomial of degree less than d in each variable on N inputs in time

$$\left((N + d^n)^{1+o(1)} \text{poly}(\log q, d, n, p)\right)$$

provided that p is at most $d^{o(1)}$, and q is at most $(\exp(\cdots(\exp(d))))$, where the height of this tower of exponentials is fixed. When the number of variables is large (e.g. $n \notin d^{o(1)}$), this is the first nearly linear time algorithm for this problem over any (large enough) field.

Our algorithm is based on elementary algebraic ideas and this algebraic structure naturally leads to the following two independently interesting applications.

We show that there is an *algebraic* data structure for univariate polynomial evaluation with nearly linear space complexity and sublinear time complexity over finite fields of small characteristic and quasipolynomially bounded size. This provides a counterexample to a conjecture of Miltersen who conjectured that over small finite fields, any algebraic data structure for polynomial evaluation using polynomial space must have linear query complexity.

We also show that over finite fields of small characteristic and quasipolynomially bounded size, Vandermonde matrices are not rigid enough to yield size-depth tradeoffs for linear circuits via the current quantitative bounds in Valiant's program. More precisely, for every fixed prime p, we show that for every constant $\epsilon > 0$, and large enough n, the rank of any $n \times n$ Vandermonde matrix V over the field $\mathbb{F}_{p^a}$ can be reduced to $\left(n/\exp(\Omega(\text{poly}(\epsilon)\sqrt{\log n}))\right)$ by changing at most $n^{\Theta(\epsilon)}$ entries in every row of V, provided $a \leq \text{poly}(\log n)$. Prior to this work, similar upper bounds on rigidity were known only for special Vandermonde matrices. For instance, the Discrete Fourier Transform matrices and Vandermonde matrices with generators in a geometric progression.

CCS CONCEPTS

• **Theory of computation** → **Algebraic complexity theory**.

KEYWORDS

Multipoint evaluation, Matrix rigidity, Modular composition, Polynomial evaluation, Data structures, Algebraic circuits, Vandermonde matrix

ACM Reference Format:
Vishwas Bhargava, Sumanta Ghosh, Mrinal Kumar, and Chandra Kanta Mohapatra. 2022. Fast, Algebraic Multivariate Multipoint Evaluation in Small Characteristic and Applications. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3519935.3519968

1 INTRODUCTION

We study the question of designing fast algorithms for the following very natural and fundamental computational task.

[*]Research supported in part by the Simons Collaboration on Algorithms and Geometry and NSF grant CCF-1909683
[†]A part of this work was done during a postdoctoral stay at the Department of Computer Science & Engineering, IIT Bombay, Mumbai, India.

Question 1.1 (Multipoint Evaluation)**.** *Given the coefficient vector of an n-variate polynomial f of degree at most $d-1$ in each variable over a field $\mathbb{F}$ and a set of points $\{\boldsymbol{\alpha}_i : i \in [N]\}$ in $\mathbb{F}^n$, output $f(\boldsymbol{\alpha}_i)$ for each $i \in [N]$.*

Besides being a natural and fundamental question in computer algebra on its own, fast algorithms for this problem is also closely related to fast algorithms for other natural algebraic questions like polynomial factorization and modular composition [10].

The input for this question can be specified by $(d^n + Nn)$ elements of $\mathbb{F}$ and clearly, there is a simple algorithm for this task which needs roughly $((d^n \cdot N)\mathrm{poly}(n,d))$ arithmetic operations over $\mathbb{F}$: just evaluate f on $\boldsymbol{\alpha}_i$ for every i iteratively.

Thus for $N = d^n$, the number of field operations needed by this algorithm is roughly quadratic in the input size. While *nearly linear time*[1] algorithms have been known for the univariate instance of multipoint evaluation [5] for close to five decades, fast algorithms for the multivariate version have been much harder to come by. In a significant improvement to the state of art for this problem, Umans [15] and Kedlaya & Umans [10] gave nearly linear time algorithms for this problem over fields of small characteristic and over all finite fields respectively, provided that the number of variables n is at most $d^{o(1)}$ where the degree of the input polynomial in every variable is less than d. They also stated the question of designing fast algorithms for the large variable case (i.e. $n \notin d^{o(1)}$) as an open problem. In this work, we make some concrete progress towards this question over finite fields of small characteristic (and not too large size). We also show two independently interesting applications of our algorithm. The first is to an upper bound for algebraic data structures for univariate polynomial evaluation over finite fields and second is to an upper bound on the rigidity of Vandermonde matrices over fields of small characteristic. Before stating our results, we start with a brief outline of each of these problems and discuss some of the prior work and interesting open questions. We state our results in Section 2.

1.1 Algorithms for Multivariate Multipoint Evaluation

For the case of univariate polynomials and $N = d$, Borodin and Moenck [5] showed that multipoint evaluation can be solved in $O(d\,\mathrm{poly}(\log d))$ field operations via a clever use of the Fast Fourier Transform (FFT).

For multivariate polynomials, when the evaluation points of interest are densely packed in a product set in $\mathbb{F}^n$, FFT based ideas naturally generalize to multivariate multipoint evaluation yielding a nearly linear time algorithm. However, if the evaluation points are arbitrary and the underlying field is sufficiently large [2], and in particular not packed densely in a product set, the question of designing algorithms for multipoint evaluation that are significantly faster than the straightforward quadratic time algorithm appears to be substantially harder. In fact, the first significant progress in this direction was achieved nearly three decades after the work of Borodin and Moenck by Nüsken and Ziegler [14] who showed that for $n = 2$ and $N = d^2$, multipoint evaluation can be solved in most $O(d^{\omega_2/2+1})$ operations, where ω_2 is the exponent for multiplying a $d \times d$ and a $d \times d^2$ matrix. The algorithm in [14] also generalizes to give an algorithm for general n that requires $O(d^{\omega_2/2(n-1)+1})$ field operations.[3] Two significant milestones in this line of research are the results of Umans [15] and Kedlaya & Umans [10] who designed nearly linear time algorithms for this problem for fields of small characteristic and over all finite fields respectively, provided the number of variables n is at most $d^{o(1)}$. We now discuss these results in a bit more detail.

Umans [15] gave an algorithm for multipoint evaluation over finite fields of small characteristic. More precisely, the algorithm in [15] solves multipoint evaluation in time $O((N + d^n)(n^2 p)^n) \cdot \mathrm{poly}(d, n, p, \log N)$ over a finite field $\mathbb{F}$ of characteristic p. Thus, when p and n are $d^{o(1)}$, the running time can be upper bounded by $(N + d^n)^{1+\delta}$ for every constant $\delta > 0$ and d, N sufficiently large. In addition to its impressive running time, the algorithm of Umans [15] is also algebraic, i.e. it only requires algebraic operations over the underlying field. With multipoint evaluation naturally being an algebraic computational problem, an algebraic algorithm for it has some inherent aesthetic appeal. The results in [15], while being remarkable has two potential avenues for improvement, namely, a generalization to other fields and to the case when the number of variables is not $d^{o(1)}$.

In [10], Kedlaya & Umans addressed the first of these issues. They showed that multipoint evaluation can be solved in nearly linear time over *all* finite fields. More precisely, for every $\delta > 0$, their algorithm for multipoint evaluation has running time $(N + d^n)^{1+\delta} \log^{1+o(1)} q$ over any finite field $\mathbb{F}$ of size q, provided d is sufficiently large and $n = d^{o(1)}$. Quite surprisingly, the algorithm in [10] is not algebraic. It goes via lifting the problem instance from the finite field $\mathbb{F}$ to an instance over $\mathbb{Z}$ and then relies on an extremely clever and unusual application of the Chinese Remainder Theorem to reduce the instance over $\mathbb{Z}$ back to instances over small finite fields. Intuitively, the gain in the entire process comes from the fact that in the reduced instances obtained over small finite fields, the evaluation points of interests are quite densely packed together inside a small product set and a standard application of the multidimensional FFT can be used to solve these small field instances quite fast. Another closely related result is a recent work of Björklund, Kaski and Williams [4] who (among other results) give an algorithm for multivariate multipoint evaluation but their time complexity depending polynomially on the field size (and not polynomially on the logarithm of the field size).

In addition to these algorithms for multivariate multipoint evaluation, Umans [15] and Kedlaya & Umans [10] also show that these fast algorithms lead to significantly faster than previously known algorithms for many other natural algebraic problems. This includes the questions of modular composition where the input consists of three univariate polynomials $f, g, h \in \mathbb{F}[X]$ of degree less than d each and the goal is to output $(f(g(X)) \mod h(X))$. In addition to

[1]Throughout this paper, we use the phrase "nearly linear time" to refer to algorithms such that for all sufficiently large m, they run in time $m^{1+o(1)}$ on inputs of size m.
[2]Over small fields, for instance if $|\mathbb{F}| \le d^{1+o(1)}$ or $|\mathbb{F}|^n \le N^{1+o(1)}$, a standard application of multidimensional Fast Fourier Transform which just evaluates the polynomial at all points in $\mathbb{F}^n$ and looks up the values at the N input points works in nearly linear time. So, throughout the discussion on multipoint evaluation, we assume that $\mathbb{F}$ is large enough.

[3]The results in both [5] and [14] work for arbitrary N, but for simplicity have been stated for $N = d$ and $N = d^2$ respectively here.

being interesting on its own, faster algorithms for modular composition over finite fields are known to directly imply faster algorithms for univariate polynomial factorization over such fields. Indeed, using their nearly linear time algorithm for multipoint evaluation , Umans [15] and Kedlaya & Umans [10] obtain the currently fastest known algorithms for univariate polynomial factorization over finite fields. We refer the reader to [10] for a detailed discussion of these connections and implications.

In spite of the significant progress on the question of algorithms for multipoint evaluation in [15] and [10], some very natural related questions continue to remain open. For instance, we still do not have nearly linear time algorithms for multipoint evaluation when the number of variables is large, e.g. $n \notin d^{o(1)}$ over any (large enough) finite field, or when the field is not finite. Since multipoint evaluation is quite naturally an algebraic computational problem, it would also be quite interesting to have a nearly linear size arithmetic circuits over the underlying field for this problem even if such a circuit cannot be efficiently constructed. Currently, small circuits of this kind are only known over finite fields of small characteristic due to the results in [15]. The algorithm in [10] does not seem to yield such a circuit since it is not algebraic over the underlying field.

1.2 Data Structures for Polynomial Evaluation

One particular implication of the results in [10] is towards the question of constructing efficient data structures for polynomial evaluation over finite fields. The *data* here is a univariate polynomial $f \in \mathbb{F}[X]$ of degree less than n over a finite field $\mathbb{F}$. The goal is to process this data and store it in a way that we can support fast polynomial evaluation queries, i.e. queries of the form: given an $\alpha \in \mathbb{F}$ output $f(\alpha)$. The two resources of interest here are the space required to store the data and the number of locations [4] accessed for every query, i.e the query complexity. There are two very natural solutions to this problem.

- We can store the coefficient vector of the polynomial f in the memory and for each query $\alpha \in \mathbb{F}$, we can read the whole memory to recover the coefficient vector of f and hence compute $f(\alpha)$. Thus, the space complexity and the query complexity of this data structure are both $(O(n \log q))$ bits, with clearly the space requirement being the best that we can hope for.
- The second natural data structure for this problem just stores the evaluation of f on all $\alpha \in \mathbb{F}$ in the memory, and on any query, can just read off the relevant value. Thus, the space complexity here is $O(q \log q)$ bits, but the query complexity is $O(\log q)$ bits (which is the best that we can hope for). For q being much larger than n the space requirement here is significantly larger than that in the first solution.

Using their algorithm for multipoint evaluation in [10], Kedlaya & Umans construct a data structure for this problem with space complexity $n^{1+\delta} \log^{1+o(1)} q$ and query complexity $\mathrm{poly}(\log n) \cdot \log^{1+o(1)} q$ for all $\delta > 0$ and sufficiently large n. Thus, the space needed is quite close to optimal, and the query complexity is within a $\mathrm{poly}(\log n)$ factor of the optimal. Quite surprisingly, this data structure is not algebraic since it relies on the multipoint evaluation algorithm in [10] which in turn relies on non-algebraic modular arithmetic. We also note that while the algorithm for multipoint evaluation over fields of small characteristic in [15] is algebraic, to the best of our knowledge, it does not immediately yield a data structure for polynomial evaluation. We remark that while the discussion here has been focused on data structures for univariate polynomial evaluation, the ideas in [10] continue to work as it is even for the multivariate version of this problem and gives quantitatively similar results there. In fact, their solution to the univariate problem goes via a reduction to the multivariate case!

In a recent work, Björklund, Kaski and Williams [4] also prove new data structures upper bounds for polynomial evaluations for multivariate polynomials over finite fields. These data structures are algebraic and are based on some very neat geometric ideas closely related to the notion of Kakeya sets over finite fields. Their construction can be viewed as giving a tradeoff in the space and query complexities but at least one of these parameters always appears to have polynomial dependence on the size of the underlying finite field. This is in contrast to the results in [10] where the query complexity depends nearly linearly on $\log q$ which is more desirable for this problem.

A very natural open question in this line of research is to obtain an algebraic data structure for this problem which matches the space and query complexity of the results in [10]. Currently, we do not have an algebraic data structure for this problem over with even polynomial space and sublinear query complexity over any sufficiently large field. In fact, Miltersen [13] showed that for algebraic data structures over finite fields of size exponential in n, if the space used is $\mathrm{poly}(n)$, then the trivial data structure obtained by storing the given polynomial as a list of coefficients and reading off everything in the memory on every query is essentially the best we can do. Miltersen also conjectured a similar lower bound to hold over smaller fields. Thus, over smaller finite fields (for instance, finite fields of size $\mathrm{poly}(n)$), either proving a lower bound similar to that in [13] , or constructing *algebraic* data structures for polynomial evaluation with perform guarantees similar to those in [10] are extremely interesting open problems. For the later goal, it would be a good start to even have an algebraic data structure that does significantly better than the trivial solution of storing the coefficient vector of the given polynomial.

1.3 Non-Rigidity of Vandermonde Matrices

An application of our results for multipoint evaluation is towards upper bounds for the rigidity of Vandermonde matrices. In this section, we give a brief overview of matrix rigidity.

Let $\mathbb{F}$ be any field. An $n \times n$ matrix M over $\mathbb{F}$ is said to be (r, s) rigid for some parameters $r, s \in \mathbb{N}$ if M cannot be written as a sum of $n \times n$ matrices of rank at most r and sparsity at most s. In other words, the rank of M cannot be reduced to less than or equal to r by changing at most s of its entries. This notion was defined by Valiant [16] who showed that if the linear transformation given by M can be computed by an arithmetic circuit of size $O(n)$ and depth $O(\log n)$, then M is not $(O(n/\log\log n), O(n^{1+\epsilon}))$ rigid for any $\epsilon > 0$.

[4]This can be measured in terms of the cells accessed where each cell contains an element over the underlying field. This is an instance of the cell probe model and is quite natural in the context of algebraic data structures for algebraic problems. Alternatively, we can also measure the space and query complexity in terms of the number of bits stored and accessed respectively.

For brevity, we say that a family of matrices is *Valiant rigid* if it is $(O(n/\log\log n), O(n^{1+\epsilon}))$ rigid for some $\epsilon > 0$.

Even though the question of provable rigidity lower bounds for explicit matrix families has remained elusive, there has been a steady accumulation of various families of explicit matrices that are suspected to be rigid. For instance, Hadamard Matrices, Design Matrices, the Discrete Fourier Transform (DFT) matrices and various Vandermonde Matrices have all been suspected to be rigid with varying parameters at various points in time. For some of these cases, we even have rigidity lower bounds either for special cases or with parameters weaker than what is needed for Valiant's connection to arithmetic circuit lower bounds. However, quite surprisingly Alman & Williams [2] showed that Hadamard matrices are not Valiant rigid over $\mathbb{Q}$. This result was succeeded by a sequence of recent results all showing that many more families of matrices suspected to be highly rigid are in fact not Valiant rigid. This includes the work of Dvir & Edelman [6], the results of Dvir & Liu [7], those of Alman [1] and Kivva [11].

This list of suspected to be highly rigid that have since been proven innocent includes families like Hadamard Matrices [2], Discrete Fourier Transform (DFT) Matrices, Circulant and Toeplitz matrices [7] and any family of matrices that can be expressed as a Kronecker product of small matrices [1, 11].

However, a notable family of matrices missing from this list is that of Vandermonde matrices. Special cases of Vandermonde matrices, for instance the DFT matrices, are known to be not be Valiant rigid, and in fact this result extends to the case of all Vandermonde matrices where the *generators* are in geometric progression.[5] However, the case of Vandermonde matrices with arbitrary generators is still not well understood.[6]

2 OUR RESULTS

We now state our results formally and try to place them in the context of prior work.

2.1 Algorithms for Multivariate Multipoint Evaluation

Our main result is a fast algebraic algorithm for multipoint evaluation over fields of small characteristic. We state this result informally here, and refer the reader to Theorem 7.1 for a formal statement.

Theorem 2.1 (Informal)**.** *Over a field $\mathbb{F}_{p^a}$ of characteristic p, there is a deterministic algorithm which evaluates a given n-variate polynomial of degree less than d in each variable on N inputs in time*

$$\left((N + d^n)^{1+o(1)} \cdot \mathit{poly}(a, d, n, p)\right),$$

provided that p is at most $d^{o(1)}$ and a is at most $(\exp(\cdots(\exp(d))))$, where the height of this tower of exponentials is fixed.

A few remarks are in order.

[5]An $n \times n$ Vandermonde matrix over a field $\mathbb{F}$ is specified by a list of n field elements $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ in $\mathbb{F}$ that we call generators. The rows and columns are indexed by $\{0, 1, \ldots, n-1\}$ and the (i, j)th entry of the matrix equals $\alpha_i{}^j$.

[6]Lokam [12] shows that $n \times n$ Vandermonde matrices with algebraically independent generators are at least $(\sqrt{n}, \Omega(n^2))$ rigid. This bound, however, is not sufficient for Valiant's program.

Remark 2.2. *Throughout this paper, we assume that we are given a description of the field $\mathbb{F}_{q=p^a}$ as a part of the input. For instance, we are given an irreducible polynomial $v(Y) \in \mathbb{F}_p[Y]$ of degree equal to $\log_p q$ and $\mathbb{F}_q \equiv \mathbb{F}_p[Y]/\langle v(Y)\rangle$.* ⌟

Remark 2.3. *Our algorithms for Theorem 2.1 can be viewed as naturally giving an arithmetic circuit of nearly linear size for multivariate multipoint evaluation over the underlying finite field $\mathbb{F}_{p^a}$. Throughout this paper, this is what we mean when we say we have an "algebraic" algorithm. Moreover, given a description of $\mathbb{F}_q$ as in Remark 2.2, we can use the algorithm in Theorem 2.1 to output such a circuit for multipoint evaluation in nearly linear time.* ⌟

As alluded to in the introduction, when the number of variables is large (e.g. $n \notin d^{o(1)}$), this is the first nearly linear time algorithm for this problem over any sufficiently large field. Prior to this work, the fastest known algorithms for multivariate multipoint evaluation are due to the results of Umans [15] and Kedlaya & Umans [10] who give nearly linear time algorithms for this problem over finite fields of small characteristic and all finite fields respectively when the number of variables n is at most $d^{o(1)}$. Theorem 2.1 answers an open question of Kedlaya & Umans [10] over the fields where it applies.

By a direct connection between the complexity of multipoint evaluation and modular composition shown by Kedlaya & Umans [10], Theorem 2.1 implies a nearly linear time algorithm for modular composition even when the number of variables n is not less than $d^{o(1)}$. In [10], such an algorithm was obtained when $n < d^{o(1)}$ (over all finite fields). More precisely, we have the following corollary.

Corollary 2.4 (Informal)**.** *Let $\mathbb{F}_{p^a}$ be a field of characteristic p. Then, there is an algorithm that on input an n-variate polynomial $f(X_1, X_2, \ldots, X_n)$ of individual degree less than d and univariate polynomials $g_1(X), \ldots, g_n(X)$ and $h(X)$ in $\mathbb{F}_{p^a}[X]$ with degree less than N, outputs the polynomial*

$$f(g_1(X), g_2(X), \ldots, g_n(X)) \mod h(X)$$

in time

$$(d^n + N)^{1+o(1)} \cdot \mathit{poly}(a, d, n, p),$$

provided that p is at most $d^{o(1)}$ and a is at most $(\exp(\cdots(\exp(d))))$, where the height of this tower of exponentials is fixed.

Our algorithm is based on elementary algebraic ingredients. One of these ingredients is the basic fact that the restriction of a low degree multivariate polynomial to a low degree curve is a low degree univariate polynomial! We use this fact together with some other algebraic tools, e.g. univariate polynomial interpolation (with multiplicities), structure of finite fields, and multidimensional FFT for our algorithm. We describe an overview of the main ideas in the proof in Section 3. We also note that even though the algorithm in [15] is algebraic, it appears to be based on ideas very different from those in this paper. In particular, Umans relies on a clever reduction from the multivariate problem to the univariate problem by working over appropriate extension of the underlying field. This is then combined with the classical univariate multipoint evaluation algorithm to complete the picture. Our algorithm, on the other hand, does not involve a global reduction from the multivariate set up to the univariate set up, and crucially relies on more local properties of low degree multivariate polynomials.

Another related prior work is a result of Björklund, Kaski and Williams [4], who give a data structure (and an algorithm) for multipoint evaluation and some very interesting consequences to fast algorithms for problems in #¶. We note that at a high level, the structure of our algorithm is similar to that of the algorithm of Björklund, Kaski and Williams [4]. However, the technical details and quantitative bounds achieved are different. One major difference is that the time complexity of the algorithm in [4] depends *polynomially* on the field size. Thus strictly speaking, with the field size growing, this algorithm is not polynomial time in the input size. On the other hand, the time complexity of the algorithms in the works of Umans [15], Kedlaya & Umans [10] and that in Theorem 2.1 depends polynomially in the logarithm of the field size, as is more desirable. We discuss the similarities and differences in the high level structure of the algorithm in [4] and that in Theorem 2.1 in a little more detail in Section 3.

2.2 Data Structures for Polynomial Evaluation

As an interesting application of our ideas in Theorem 2.1, we get the following upper bound for data structure for polynomial evaluation.

Theorem 2.5 (Informal)**.** *Let p be a fixed prime. Then, for all sufficiently large $n \in \mathbb{N}$ and all fields $\mathbb{F}_{p^a}$ with $a \leq poly(\log n)$, there is an algebraic data structure for polynomial evaluation for univariate polynomials of degree less than n over $\mathbb{F}_{p^a}$ that has space complexity at most $n^{1+o(1)}$ and query complexity at most $n^{o(1)}$.*

A more precise version of Theorem 2.5 can be found in the full version [3]. We remark that by an algebraic data structure, we mean that there is an algebraic algorithm (in the spirit of Remark 2.3) over $\mathbb{F}_{p^a}$ that, when given the coefficients of a univariate polynomial f of degree at most n as input outputs the data structure $\mathcal{D}_f$ in time $n^{1+o(1)}$ and another algebraic algorithm which when given an $\alpha \in \mathbb{F}_{p^a}$ and query access to $\mathcal{D}_f$ outputs $f(\alpha)$ in time $n^{o(1)}$. In other words, there is an arithmetic circuit C_1 over $\mathbb{F}_{p^a}$ with $n^{1+o(1)}$ outputs that when given the coefficients of f as input, outputs $\mathcal{D}_f$ and an arithmetic circuit C_2 with $n^{o(1)}$ inputs satisfying the following: for every $\alpha \in \mathbb{F}_{p^a}$, there is a subset $S(\alpha)$ of cells in $\mathcal{D}_f$ such that on input α and $\mathcal{D}_f|_{S(\alpha)}$, C_2 outputs $f(\alpha)$.

As alluded to in the introduction, Miltersen [13] showed that over finite fields that are exponentially large (in the degree parameter n), any algebraic data structure for polynomial evaluation with space complexity poly(n) must have query complexity $\Omega(n)$. He also conjectured that the lower bound continues to hold over smaller fields.[7] Theorem 2.5 provides a counterexample to this conjecture when the underlying field has small characteristic and is quasipolynomially bounded in size.

The data structure of Kedlaya & Umans [10] outperforms the space and query complexities of the data structure in Theorem 2.5. However, their construction is not algebraic; essentially because their algorithm for multipoint evaluation is not algebraic.[8] However, their construction works over all finite fields, while we require fields of small characteristic that are quasipolynomially bounded in size. Umans' [15] algorithm for multipoint evaluation on the other hand is algebraic, although to the best of our knowledge, this is not known to give a data structure for polynomial evaluation. Finally, we note that for the algebraic data structure in the work of Björklund, Kaski and Williams [4], either the query complexity or the space complexity has polynomial dependence on the field size and thus even over fields of polynomial size it does not appear to give nearly linear space complexity or sublinear query complexity. However, the results in [4] are stated for multivariate polynomials and it is not clear to us if for the special case of univariate polynomial one can somehow bypass this polynomial dependence on field size by a careful modification of their construction.

2.3 Upper Bound on the Rigidity of Vandermonde Matrices

As the second application of the ideas in Theorem 2.1, we show the following upper bound on the rigidity of general Vandermonde matrices.

Theorem 2.6 (Informal)**.** *Let p be a fixed prime. Then, for all constants ϵ with $0 < \epsilon < 0.01$ and for all sufficiently large n, if V is an $n \times n$ Vandermonde matrix over the field $\mathbb{F}_{p^a}$ for $a \leq poly(\log n)$, then the rank of V can be reduced to $\frac{n}{\exp(\Omega(\epsilon^7 \log^{0.5} n))}$ by changing at most $n^{1+\Theta(\epsilon)}$ entries of V.*

For a more formal version of Theorem 2.6, we refer to the full version of our paper [3]. Theorem 2.6 extends the list of natural families of matrices that were considered potential explicit candidates for rigidity but turn out to not be rigid enough for Valiant's program [16] of obtaining size-depth tradeoffs for linear arithmetic circuits via rigidity. Prior to this work, such upper bounds on rigidity were only known for special Vandermonde matrices, for instance, the Discrete Fourier transform matrix and Vandermonde matrices with generators in geometric progression [7].

Our proof of Theorem 2.6 crucially relies on the results in [7] and combines these ideas with ideas in the proof of Theorem 2.1. We discuss these in more details in the next section.

3 AN OVERVIEW OF THE PROOFS

In this section we describe some detail, the main high level ideas of our proofs. We begin with a detailed overview of our algorithms for multipoint evaluation. We have three algorithms (Section 5, Section 6 and Section 7) starting with the simplest one and each subsequent algorithm building upon the previous one with some new ideas. We start with the simplest one here.

3.1 A Simple Algorithm for Multipoint Evaluation

We start with some necessary notation. Let p be a prime and $\mathbb{F}_q$ be a finite field with $q = p^a$. Let $f \in \mathbb{F}_q[\mathbf{x}]$ be an n-variate polynomial of degree at most $d-1$ in every variable and for $i = 1, 2, \ldots, N$ let $\boldsymbol{\alpha}_i \in \mathbb{F}_q^n$ be points. The goal is to output the value of f at each of

[7]We note that Miltersen did not precisely quantify what *smaller* fields mean, but the case when the field size is a large polynomial in the degree parameter n is a natural setting, since the trivial data structures in this case do not have both nearly linear space and sublinear query complexity. Theorem 2.5 provides such a construction when the underlying field additionally has a small characteristic.

[8]This is also the reason why the data structure in [10] does not give a counterexample to Miltersen's conjecture.

these points $\boldsymbol{\alpha}_i$. As is customary, we assume that the field $\mathbb{F}_q$ is given as $\mathbb{F}_p[Y]/\langle v(Y)\rangle$ for some degree a irreducible polynomial $v(Y) \in \mathbb{F}_p[Y]$. In Observation 4.2, we observe that given the irreducible polynomial $v(Y) \in \mathbb{F}_p[Y]$ such that $\mathbb{F}_q = \mathbb{F}_p[Y]/\langle v(Y)\rangle$ and any $u \in \mathbb{F}_q$, we can efficiently compute the coefficients of the univariate polynomial over $F_p[Y]$ corresponding to u via arithmetic operations over $\mathbb{F}_q$. Therefore, for the rest of this discussion, we assume that every field element (in the coefficients of f and the coordinates of $\boldsymbol{\alpha}_i$) are explicitly given to univariate polynomials of degree at most $a-1$ in $\mathbb{F}_p[Y]$.

We start with a discussion of the simplest version of our algorithm before elaborating on the other ideas needed for further improvements. The formal guarantees for this version can be found in Theorem 5.1. The algorithm can be thought to have two phases, the preprocessing phase and the local computation phase.

Preprocessing Phase. We start with a description of the preprocessing phase.

- **A Subfield of Appropriate Size:** As the first step of the algorithm, we compute a natural number b such that $p^{b-1} \leq adn \leq p^b$. For the ease of this discussion, let us assume that b divides a, and thus $\mathbb{F}_{p^b}$ is a subfield of $\mathbb{F}_q = \mathbb{F}_{p^a}$. If b does not divide a, then we work in a field $\mathbb{F}_{p^c}$ that is a common extension of $\mathbb{F}_{p^a}$ and $\mathbb{F}_{p^b}$.
- **Evaluating f on $\mathbb{F}^n_{p^b}$:** We now use the standard multidimensional Fast Fourier Transform algorithm to evaluate f on all of $\mathbb{F}^n_{p^b}$. This algorithm runs in quasilinear time in the input size, i.e. $\tilde{O}(d^n + (p^{bn}))$, where $\tilde{O}$ hides $\mathrm{poly}(d,n,p,b)$ factors. From our choice of b, we note that this quantity is at most $\tilde{O}((padn)^n)$.

Local Computation Phase: We now describe the local computation phase.

- **A Low Degree Curve through $\boldsymbol{\alpha}_i$:** Once we have the evaluation of f on all points in $\mathbb{F}^n_{p^b}$, we initiate some *local* computation at each $\boldsymbol{\alpha}_i$. This local computation would run in time $(adn)^c$ for some fixed constant c, thereby giving an upper bound of $\tilde{O}\left((pad)^n + N(adn)^{O(1)}\right)$ on the total running time. To describe this local computation, let us focus on a point $\boldsymbol{\alpha}_i$. Since the field elements of $\mathbb{F}_q$ are represented as univariate polynomials of degree at most $(a-1)$ in $\mathbb{F}_p[Y]$, we get that for every $\boldsymbol{\alpha}_i \in \mathbb{F}^n_q$, there exist vectors $\boldsymbol{\alpha}_{i,0}, \boldsymbol{\alpha}_{i,1}, \ldots, \boldsymbol{\alpha}_{i,a-1}$ in $\mathbb{F}^n_p$ such that
$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}Y + \cdots + \boldsymbol{\alpha}_{i,a-1}Y^{a-1}\,.$$
Let us now consider the curve $\mathrm{g}(t) \in \mathbb{F}^n_p[t]$ defined as
$$\mathrm{g}_i(t) = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}t + \cdots + \boldsymbol{\alpha}_{i,a-1}t^{a-1}\,.$$
We are interested in some simple properties of this curve. The first such property is that it passes through the point $\boldsymbol{\alpha}_i$, since $\boldsymbol{\alpha}_i = \mathrm{g}_i(Y)$ (recall that Y is an element of $\mathbb{F}_q = \mathbb{F}_p[Y]/\langle v(Y)\rangle$ here). The second property is that this curve contains *a lot* of points in the $\mathbb{F}^n_{p^b}$. In particular, note that for every $\gamma \in \mathbb{F}_{p^b}$, $\mathrm{g}_i(\gamma) \in \mathbb{F}^n_{p^b}$. Thus, there are at least p^b points on $\mathrm{g}_i(t)$ in $\mathbb{F}^n_{p^b}$ (counted with multiplicities).
- **Restriction of f to $\mathrm{g}_i(t)$:** We now look at the univariate polynomial $h_i(t)$ obtained by restricting the n-variate polynomial f to the curve $\mathrm{g}_i(t)$. Thus, if $\mathrm{g}_i(t) = (g_{i,0}(t) \ldots g_{i,n-1}(t))$ for some univariate polynomials $g_{i,j}(t)$ of degree at most $a-1$, then $h_i(t)$ is equal to the polynomial $f(g_{i,0}(t), \ldots, g_{i,n-1}(t))$. Clearly, the degree of h_i is at most $a(d-1)n < adn$. From our previous discussion, we know that $h_i(Y) = f(\boldsymbol{\alpha}_i)$. Moreover, we have already evaluated f on all of $\mathbb{F}^n_{p^b}$ and thus, we know the value of $h_i(\gamma)$ for all $\gamma \in \mathbb{F}_{p^b}$. Note that these are at least p^b many inputs on which the value of $h_i(t)$ is correctly known to us. Also, from our choice of b, we know that $p^b > adn > \deg(h_i)$. Thus, we can recover the polynomial h_i completely using univariate polynomial interpolation in time at most $\mathrm{poly}(a,d,n,p)$, and thus can output $h_i(Y) = f(\boldsymbol{\alpha}_i)$ in time $\mathrm{poly}(a,d,n,p)$. Iterating this local computation for every $i \in \{0,1,\ldots,N-1\}$, we can compute the value of f at $\boldsymbol{\alpha}_i$ for each such i.

Correctness and Running Time: The correctness of the algorithm immediately follows from the outline above. Essentially, we set things up in a way that to compute $f(\boldsymbol{\alpha}_i)$ it suffices to evaluate the univariate polynomial h_i at input $Y \in \mathbb{F}_q$. Moreover, from the preprocessing phase, we already have the value of f on $\mathbb{F}^n_{p^b}$ and this in turn gives us the evaluation of $h_i(t)$ on $p^b > adn > \deg(h_i)$ distinct inputs. Thus, by standard univariate polynomial interpolation, we recover h_i and hence $h_i(Y) = f(\boldsymbol{\alpha}_i)$ correctly.

The time complexity of the preprocessing phase is dominated by the step where we evaluate f on $\mathbb{F}^n_{p^b}$. This can be upper bounded by $\tilde{O}((padn)^n)$ using the standard multidimensional FFT algorithm. In the local computation phase, the computation at each input point $\boldsymbol{\alpha}_i$ involves constructing the curve $\mathrm{g}_i(t)$, constructing the set $\{(\gamma, h_i(\gamma)) : \gamma \in \mathbb{F}_{p^b}\}$, using the evaluation of h_i on these p^b inputs to recover h_i uniquely via interpolation and then computing $h_i(Y)$. For every $\gamma \in \mathbb{F}_{p^b}$, $\mathrm{g}_i(\gamma) \in \mathbb{F}^n_{p^b}$ can be done in time at most $\mathrm{poly}(a,d,n,p)$. So, the total time complexity of this phase is at most $(N \cdot \mathrm{poly}(a,d,n,p))$, and hence the total running time of the algorithm is $\tilde{O}(N + (padn)^n)$.

3.2 Towards Faster Multipoint Evaluation

The algorithm outlined in the previous section achieves a $O(Nn + d^n)^{1+o(1)}$ when $apn = d^{o(1)}$. We now try to modify it so that it continues to be nearly linear time even when the number of variables n and the degree of underlying field a are not less than $d^{o(1)}$. The factor of p^n appears to be inherent to our approach and seems difficult to get rid of, and this leads to the restriction of working over fields of small characteristic for all our results in this paper.

Before proceeding further, we remark that the basic intuition underlying all of our subsequent algorithms are essentially the same as those in the simple algorithm outlined in this section. For each of the further improvements, we modify certain aspects of this algorithm using a few more technical (and yet simple) ideas on top of the ones already discussed in Section 3.1.

Handling Large Number of Variables. The factor of n^n in the running time appears in the preprocessing phase of the algorithm in Section 3.1. The necessity for this stems from the fact that the univariate polynomial $h_i(t)$ obtained by restricting f to the curve $\mathbf{g}_i(t)$ through $\boldsymbol{\alpha}_i$ can have degree as large as $a(d-1)n$. Thus, for interpolating $h_i(t)$ from its evaluations, we need its value on at least $a(d-1)n+1$ distinct inputs. Thus, we need p^b to be at least $a(d-1)n+1$.

However, we note that if we have access to not just the evaluations of $h_i(t)$, but also to the evaluations of its derivatives up to order $n-1$ at each of these inputs in $\mathbb{F}_{p^b}$, then $h_i(t)$ can be uniquely from this information provided p^b is at least $\deg(h_i(t))/n$, i.e. $(a(d-1)n+1)/n \leq ad$ (see Lemma 4.9 for a formal statement). Thus, with observation at hand, we now choose b such that $p^{b-1} \leq ad \leq p^b$. Moreover, for the local computation, we now need not only the evaluation of h_i on all points in $\mathbb{F}_{p^b}$ but also the evaluations of all derivatives of $h_i(t)$ of order at most $n-1$ on all these points. A natural way of ensuring that the evaluations of these derivatives of $h_i(t)$ are available in the local computation phase is to compute not just the evaluation of f but also of *all* its partial derivatives of up to n on all of $\mathbb{F}_{p^b}^n$. Together with the chain rule of partial derivatives, we can use the evaluations of these partial derivatives of f and the identity $h_i(t) = f \circ \mathbf{g}_i(t)$ to obtain the evaluations of $h_i(t)$ and all its derivatives of order at most $n-1$ on all inputs in $\mathbb{F}_{p^b}$. This ensures that h_i can once again be correctly and uniquely recovered given this information via a standard instance of Hermite Interpolation, which in turn ensures the correctness of the algorithm.

To see the effect on the running time, note that in the preprocessing phase, we now need to evaluate not just f but all its partial derivatives of order at most $n-1$ on all of $\mathbb{F}_{p^b}^n$. Thus, there are now roughly $\binom{n+n}{n} \leq 4^n$ polynomials to work with in this phase. So, given the coefficients of f, we first obtain the coefficients of all these derivatives, and then evaluate these polynomials on $\mathbb{F}_{p^b}^n$ using a multidimensional FFT algorithm again. Also, the coefficient representation of any fixed derivative of order up to $n-1$ can be computed from the coefficients of f in $\tilde{O}(d^n)$ time (see Lemma 4.6). Thus, the total time complexity of the preprocessing phase in this new algorithm can be upper bounded by $\tilde{O}((adp)^n 4^n)$.

Once we have this stronger guarantee from the preprocessing phase, we get to doing some local computation at each point $\boldsymbol{\alpha}_i$. Now, instead of recovering h_i via a standard univariate polynomial interpolation, we have to rely on a standard Hermite interpolation for this. In particular, we need access to the evaluation of all derivatives of $h_i(t)$ of order at most $n-1$ on all inputs $\gamma \in \mathbb{F}_{p^b}$. This can be done via an application of chain rule of derivatives and the fact that we have evaluations of *all* partial derivatives of f of order at most $n-1$ on all points in $\mathbb{F}_{p^b}^n$. The time taken for this computation at each $\gamma \in \mathbb{F}_{p^b}$ turns out to be about $O(4^n \mathrm{poly}(d,n,a,p))$. Thus, the total time taken for local computation at all the input points can be upper bounded by roughly $O(N4^n \mathrm{poly}(d,n,a,p))$.

Thus, the total time complexity of this modified algorithm is $\tilde{O}((N+(adp)^n)4^n)$. In other words, we have managed to remove the factor of n^n present in the algorithm in Section 3.1 and replace it by 4^n. An algorithm based on this improvement is described in Section 6.

Handling Larger Fields. We now discuss the improvement in the dependence on the parameter a, which is the degree of the extension of $\mathbb{F}_p$ where the input points lie. In the local computation step at each point, the curve $\mathbf{g}_i(t)$ through $\boldsymbol{\alpha}_i$ has degree $a-1$ in the worst case, since we view the field elements in $\mathbb{F}_{p^a}$ as univariate polynomials of degree at most $a-1$ with coefficients in $\mathbb{F}_p$. Therefore, the restriction of f to such a curve, namely the polynomial $h_i(t)$ can have degree $(a-1)\deg(f)$ in the worst case. This forces us to choose the parameter b such that p^b is at least $\deg(h_i)$, thereby leading to a factor of a^n in the running time. Note that if we had the additional promise that the point $\boldsymbol{\alpha}_i$ was in an extension $\mathbb{F}_{p^{\tilde{a}}}$ of $\mathbb{F}_p$ for some $\tilde{a} < a$, then the curve $\mathbf{g}_i$ would be of degree at most $(\tilde{a}-1) < (a-1)$ and hence the polynomial h_i would have degree at most $(\tilde{a}-1)\deg(f)$. More generally, if all the input points $\boldsymbol{\alpha}_i$ were promised to be in $\mathbb{F}_{p^{\tilde{a}}}^n$, we can improve the factor a^n to $(\tilde{a})^n$ in the running time by choosing b such that p^b is larger than $\tilde{a}dn$ (in fact, we only need $p^b \geq (\tilde{a}d)$ if we are working with multiplicities). We also note that for every $\tilde{a} \in \mathbb{N}$ the curve $\mathbf{g}_i(t)$ takes a value in $\mathbb{F}_{p^{\tilde{a}}}^n$ whenever t is set to a value in $\mathbb{F}_{p^{\tilde{a}}}$. As a consequence, the curve $\mathbf{g}_i$ contains at least $p^{\tilde{a}}$ points in $\mathbb{F}_{p^{\tilde{a}}}^n$. With these observations in hand, we now elaborate on the idea for reducing the a^n factor in the running time. For simplicity of exposition, we outline our ideas in the setting of the algorithm discussed in Section 3.1. In particular, derivative based improvements are not involved.

Let a' be such that $p^{a'} > adn \geq p^{a'-1}$. Now, instead of recovering h_i directly from its values on $\mathbb{F}_{p^b}$, we try to recover h_i in two steps. In the first step, we try to obtain the values of $h_i(\gamma)$ for every $\gamma \in \mathbb{F}_{p^{a'}}$ using the information we have from the preprocessing phase. Assuming that we can do this, we can again obtain h_i by interpolation and compute $h_i(Y) = f(\boldsymbol{\alpha}_i)$.

Now, to compute $h_i(\gamma)$ for $\gamma \in \mathbb{F}_{p^{a'}}$, we note that $h_i(\gamma)$ equals $f \circ \mathbf{g}_i(\gamma)$, thus it would be sufficient if we had the evaluation of f on the point set $\{\mathbf{g}_i(\gamma) : \gamma \in \mathbb{F}_{p^{a'}}\}$. This seems like the problem we had started with, but with one key difference: the points $\{\mathbf{g}_i(\gamma) : \gamma \in \mathbb{F}_{p^{a'}}\}$ are all in $\mathbb{F}_{p^{a'}}^n$ with $a' = \Theta(\log adn)$! Thus, the degree of the extension where these points lie is significantly reduced. In essence, this discussion gives us a reduction from the problem of evaluating f on N points in $\mathbb{F}_{p^a}^n$ to evaluating f on $N \cdot adn$ points in $\mathbb{F}_{p^{a'}}^n$, with $a' = \Theta(\log adn)$. Thus, we have another instance of multipoint evaluation with a multiplicatively larger point set in an extension of $\mathbb{F}_p$ of degree logarithmic in adn. If we now apply the algorithm discussed in Section 3.1, we get a running time of roughly $\tilde{O}(Nadn + (pdn\log(adn))^n)$. Thus, in the running time, the factor a^n has been replaced by $\log^n a$ at the cost of N being replaced by $Nadn$. In fact, we can continue this process ℓ times, and in each step we end up with an instance of multipoint evaluation with the size of the point set being increased by a multiplicative factor, with the gain being that we have a substantial reduction in the degree of the field extension that the points live in.

Comparison with the Techniques of Björklund, Kaski and Williams [4]. Now that we have an overview of the algorithms for multipoint evaluation in this paper, we can elaborate on the similarities they share with the algorithms in [4]. At a high level, the similarities are significant. In particular, both the algorithms have a preprocessing phase where the polynomial is on a product set using multidimensional FFT. This is followed by a local computation step, where the value of the polynomial at any specific input of interest is deduced from the already computed data by working with the restriction of the multivariate polynomial to an appropriate curve. In spite of these similarities in the high level outline, the quantitative details of these algorithms are different. One salient difference is that the time complexity of the algorithm in [4], depends polynomially on the size of the underlying field, whereas in our algorithm outlined above, this dependence is polynomial in logarithm of the field size as long as the size of the field is bounded by a tower function of fixed height in the degree parameter d. This difference stems from technical differences in the precise product set used in the preprocessing phase and the sets of curves utilized in the local computation phase. In particular, the degree of the curves in the local computation phase of our algorithms depends polynomially on $\log |\mathbb{F}|$, where as the degree of the curves used in [4] depends polynomially on $|\mathbb{F}|$. Additionally, algorithms in [4] rely on the assumption that the total degree of the polynomial divides $|\mathbb{F}^*| - 1$, whereas we do not need any such divisibility condition.

3.3 Data Structure for Polynomial Evaluation

The multipoint evaluation algorithm in Theorem 2.1 is naturally conducive to obtaining data structures for polynomial evaluation. Essentially, the evaluation of the polynomial in a fixed grid (independent of the N points of interest in the input) gives us the data structure, and the local computation at each input point of interest which requires access to some of the information computed in the preprocessing phase constitutes the query phase of the data structure. We discuss this in some more detail now.

Let $f(X) \in \mathbb{F}_{p^a}[X]$ be a univariate polynomial of degree at most n. We start by picking parameters d, m such that d^m is at least n. For any such choice of d and n, there is clearly an m-variate polynomial $F(Z_0, Z_1, \dots, Z_{m-1})$ such that $F(X, X^d, X^{d^2}, \dots, X^{d^{m-1}}) = f(X)$. In other words, the image of F under the Kronecker substitution equals f. Now, as in the multipoint evaluation algorithms, we pick the smallest integer b such that $p^b > adm$ and evaluate F on $\mathbb{F}_{p^b}^m$ and store these points along with the value of F on these inputs in the memory. This forms the memory content of our data structure. Thus, the memory can be thought of having $p^{bm} \leq (padm)^m$ cells, each containing a pair $(\mathbf{c}, F(\mathbf{c}))$ for $\mathbf{c} \in \mathbb{F}_{p^b}^m$.

To get a sense of the parameters, let us set $d = n^{1/\log\log n}$ and $m = \log\log n$. Clearly, the constraint $d^m \geq n$ is met in this case. For this choice of parameter and for p being a constant and $a \leq \mathrm{poly}(\log n)$, we get that the space complexity is at most $n^{1+o(1)}$ and the query complexity is at most $n^{o(1)}$. The complete details can be found in the full version [3].

3.4 Rigidity of Vandermonde Matrices

The connection between rigidity of Vandermonde matrices and multipoint evaluation is also quite natural. Consider a Vandermonde matrix V_n with generators $\alpha_0, \dots, \alpha_{n-1}$ and for every $i, j \in \{0, 1, \dots, n-1\}$, the (i, j)th entry of V_n is α_i^j. Now, for any univariate polynomial f of degree at most $n-1$, the coefficients of f, together with the set $\{\alpha_i : i \in \{0, 1, \dots, n-1\}\}$ of generators form an instance of (univariate) multipoint evaluation. Moreover, for any choice of the generators $\{\alpha_i : i \in \{0, 1, \dots, n-1\}\}$, the algorithm for multipoint evaluation, e.g Theorem 2.1 can naturally be interpreted as a circuit for computing the linear transform given by the matrix V_n. Furthermore, if this linear circuit is structured enough, we could, in principle hope to get a decomposition of V_n as a sum of a sparse and a low rank matrix from this linear circuit, for instance, along the lines of the combinatorial argument of Valiant [16]. Our proof of Theorem 2.6 is along this outline. We now describe these ideas in a bit more detail.

Given a univariate polynomial f of degree $n-1$ and inputs $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$, let F be an m- variate polynomial of degree d such that $(n = d^m)$[9] as described in Section 3.3. Moreover, for $i \in \{0, 1, \dots, n-1\}$, let $\boldsymbol{\alpha}_i = (\alpha_i, \alpha_i^d, \dots, \alpha_i^{d^{m-1}})$. Now, as discussed in Section 3.3, $f(\alpha_i) = F(\boldsymbol{\alpha}_i)$. Let $\tilde{V}$ be the $n \times n$ matrix where the rows are indexed by $\{0, 1, \dots, n-1\}$ and the columns are indexed by all m- variate monomials of individual degree at most $d-1$. We use the fact that $d^m = n$ here. From the above set up, it immediately follows that the coefficient vectors of f and F are equal to each other (with the coordinate indices having slightly different semantics) and the matrices V_n and $\tilde{V}$ are equal to each other.

We now observe that the algorithm for multipoint evaluation described in Section 3.1 gives a natural decomposition of $\tilde{V}$ (and hence V_n) as a product of a matrix A of row sparsity at most adm and a $p^{bm} \times d^m$ matrix B with b being the smallest integer such that $p^b > adm$. The rows of B are indexed by all elements of $\mathbb{F}_{p^b}^m$ and the columns are indexed by all m-variate monomials of individual degree at most $d-1$, and the $(\boldsymbol{\alpha}, \mathbf{e})$ entry of B equals $\boldsymbol{\alpha}^{\mathbf{e}}$. Intuitively, the matrix B corresponds to the preprocessing phase of the algorithm and the matrix A corresponds to the local computation. At this point, we use an upper bound of [7] on the rigidity of Discrete Fourier Transform matrices over finite fields and the inherent Kronecker product structure of the matrix B to obtain an upper bound on the rigidity of B. Finally, we observe that that matrix $V_n = \tilde{V} = A \cdot B$ obtained by multiplying a sufficiently non-rigid matrix B with a row sparse matrix A continues to be non-rigid with an interesting regime of parameters. This essentially completes the proof. For more details, we refer the reader to look at the full version [3] of this result.

Organization of the Paper. The rest of the paper is organized as follows. We start with the preliminaries section in Section 4. We then present the most basic version of our algorithm in Section 5 followed by the improved versions for larger number of variables and larger size fields in Section 6 and Section 7 respectively. Due to space constraints, we skip some of the details in this version of the paper, including the proofs of many of the claims and the analyses of some of the algorithms. We refer the interested reader to the full version [3] for these missing details.

4 PRELIMINARIES

We use $\mathbb{N}$ to denote the set of natural numbers $\{0, 1, 2, \dots\}$, $\mathbb{F}$ to denote a general field. For any positive integer N, $[N]$ denotes the set $\{1, 2, \dots, N\}$. By $\mathbf{x}$ and $\mathbf{z}$, we denote the variable tuples $(X_1, \dots, X_n)$ and $(Z_1, \dots, Z_n)$, respectively. For any $\mathbf{e} = (e_1, \dots, e_n) \in \mathbb{N}^n$, $\mathbf{x}^{\mathbf{e}}$ denotes the monomial $\prod_{i=1}^n X_i^{e_i}$. By $|\mathbf{e}|_1$, we denote the sum $e_1 + \cdots + e_n$.

[9]For simplicity, let us assume that such a choice of integers d, m exist.

For every positive integer k, $k!$ denotes $\prod_{i=1}^{k} i$. For $k = 0$, $k!$ is defined as 1. For two non-negative integer i and k with $k \geq i$, $\binom{k}{i}$ denotes $\frac{k!}{i!(k-i)!}$. For $k < i$, $\binom{k}{i} = 0$. For non-negative integer $i_1, \ldots, i_s$ with $i_1 + \cdots + i_s = k$, $\binom{k}{i_1,\ldots,i_s} = \frac{k!}{i_1!\cdots i_s!}$. For $\mathbf{a} = (a_1, \ldots, a_n), \mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$, $\binom{\mathbf{a}}{\mathbf{b}} = \prod_{i=1}^{n} \binom{a_i}{b_i}$, and $\binom{\mathbf{a}+\mathbf{b}}{\mathbf{a},\mathbf{b}} = \prod_{i=1}^{n} \binom{a_i+b_i}{a_i,b_i}$.

We say that a function $\psi : \mathbb{N} \to \mathbb{N}$ is polynomially bounded, or denoted by $\psi(n) \leq \text{poly}(n)$, if there exists a constant c such that for all large enough $n \in \mathbb{N}$, $\psi(n) \leq n^c$.

Suppose that p be a positive integer greater than 1. Then for any non-negative integer c, $\log_p^{\circ c}(n)$ denotes the c-times composition of logarithm function with itself, with respect to base p. For example, $\log_p^{\circ 2}(n) = \log_p \log_p(n)$. By $\log_p^{\star}(n)$, denotes the smallest non-negative integer c such that $\log_p^{\circ c}(n) \leq 1$. For $p = 2$, we may omit the subscript p in $\log_p(n)$, $\log_p^{\circ c}(n)$ and $\log_p^{\star}(n)$.

4.1 Some Facts about Finite Fields

Suppose that p is a prime and $q = p^a$ for some positive integer a. Then there exists an *unique* finite field of size q. In other words, all the finite fields of size q are *isomorphic* to each other. We use $\mathbb{F}_q$ to denote the finite field of size q, and p is called the characteristic of $\mathbb{F}_q$. For any finite field $\mathbb{F}_q$, $\mathbb{F}_q^*$ represents the multiplicative cyclic group after discarding the field element 0. For any irreducible polynomial $v(Y)$ over $\mathbb{F}_q$, the quotient ring $\mathbb{F}_q[Y]/\langle v(Y)\rangle$ forms a larger field over $\mathbb{F}_q$ of size q^b where b is the degree of $v(Y)$. The next lemma describes that we can efficiently construct such larger fields over $\mathbb{F}_q$, when the characteristic of the field is small.

Lemma 4.1. *Let p be a prime and $q = p^a$ for some positive integer a. Then, for any positive integer b, the field $\mathbb{F}_{q^b}$ can be constructed as $\mathbb{F}_q[Y]/\langle v(Y)\rangle$, where $v(Y)$ is degree b irreducible polynomial over $\mathbb{F}_q$, in poly(a, b, p) $\mathbb{F}_q$-operations. Furthermore, all the basic operations in $\mathbb{F}_{q^b}$ can be done in poly(b) $\mathbb{F}_q$-operations.*

Fix a field F_q of characteristic p. In the standard algebraic model over $\mathbb{F}_q$, the basic operations are addition, subtraction, multiplication, and division of elements in $\mathbb{F}_q$. Let $\mathbb{F}_q = \mathbb{F}_p[X]/\langle g(X)\rangle$ where $q = p^a$ and $g(X)$ is a degree a irreducible polynomial over $\mathbb{F}_p$. Then for any element $\alpha \in \mathbb{F}_q$, consider its canonical representation $\alpha = \alpha_0 + \alpha_1 X + \ldots + \alpha_{a-1}X^{a-1}$ where $\alpha_i \in \mathbb{F}_p$. Note that it is not clear how to extract α_i's from α using the algebraic operations over $\mathbb{F}_q$. We show that this is possible if p is small.

Observation 4.2. *Let p be prime and $q = p^a$ for some positive integer a. Let $\mathbb{F}_q = \mathbb{F}_p[X]/\langle g(X)\rangle$ where $g(X)$ is a degree a irreducible polynomial over $\mathbb{F}_p$. Let $\alpha \in \mathbb{F}_q$ and $\alpha = \alpha_0 + \alpha_1 X + \cdots + \alpha_{a-1}X^{a-1}$ where $\alpha_i \in \mathbb{F}_p$. Then, given blackbox access to α and $\mathbb{F}_q$-operations, $\alpha_0, \alpha_1, \ldots, \alpha_{a-1}$ can be computed in poly$(a, \log p)$ $\mathbb{F}_q$-operations.*

Thus, for the rest of our paper, we consider that the extraction of the $\mathbb{F}_p$-coefficients from elements in $\mathbb{F}_q$ as an algebraic operation.

Suppose that $\mathbb{F}_{q_1}$ and $\mathbb{F}_{q_2}$ are two finite fields of characteristic p such that $\mathbb{F}_{q_1}$ is a subfield of $\mathbb{F}_{q_2}$. Then $\mathbb{F}_{q_2}$ forms a vector space over $\mathbb{F}_{q_1}$. A subset $\{\beta_1, \beta_2, \ldots, \beta_k\}$ of $\mathbb{F}_{q_2}$ is called an $\mathbb{F}_{q_1}$-basis if every element of $\alpha \in F_{q_2}$ is a unique linear combination of β_i's over $\mathbb{F}_{q_1}$.

Lemma 4.3. *Let p be a prime and $q = p^a$ for some positive integer a. Let b be a positive integer and $\mathbb{F}_{q^b} = \mathbb{F}_q[Y]/\langle v(Y)\rangle$ for some degree b irreducible polynomial $v(Y)$ over $\mathbb{F}_q$. Then, the following holds:*

1. *The field $\mathbb{F}_{q^b}$ contains the subfield $\mathbb{F}_{p^b}$. Furthermore, all the elements of $\mathbb{F}_{p^b}$ can be computed in $p^b \cdot$ poly(a, b, p) $\mathbb{F}_q$-operations.*
2. *In poly(a, b, p) $\mathbb{F}_q$-operations, an element $\beta \in \mathbb{F}_{q^b}$ can be computed such that $\{1, \beta, \ldots, \beta^{b-1}\}$ forms an $\mathbb{F}_p$-basis for $\mathbb{F}_{p^b}$. Moreover, given any element $\alpha \in \mathbb{F}_{p^b}$, the $\mathbb{F}_p$-linear combination of α in the basis $\{1, \beta, \ldots, \beta^{b-1}\}$ can be computed in poly(b) $\mathbb{F}_q$-operations.*

4.2 Hasse Derivatives

In this section, we briefly discuss the notion of Hasse derivatives that plays a crucial role in our results.

Definition 4.4 (Hasse derivative)**.** *Let $f(\mathbf{x})$ be an n-variate polynomial over a field $\mathbb{F}$. Let $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$. Then, the Hasse derivative of f with respect to the monomial $\mathbf{x}^{\mathbf{e}}$ is the coefficient of $\mathbf{z}^{\mathbf{e}}$ in the polynomial $f(\mathbf{x} + \mathbf{z}) \in (\mathbb{F}[\mathbf{x}])[\mathbf{z}]$.* ⌟

Notations. Suppose that $f(\mathbf{x})$ be an n-variate polynomial over a field $\mathbb{F}$. Let $\mathbf{b} \in \mathbb{N}^n$. Then, $\overline{\partial}_{\mathbf{b}}(f)$ denotes the Hasse derivative of $f(\mathbf{x})$ with respect to the monomial $\mathbf{x}^{\mathbf{b}}$. For any non-negative integer k, $\overline{\partial}^{\leq k}(f)$ is defined as

$$\overline{\partial}^{\leq k}(f) = \left\{ \overline{\partial}_{\mathbf{b}}(f) \;\middle|\; \mathbf{b} \in \mathbb{N}^n \text{ s.t. } |\mathbf{b}|_1 \leq k \right\},$$

For a univariate polynomial $h(t)$ over $\mathbb{F}$ and a non-negative integer k, $h^{(k)}(t)$ denotes the Hasse derivative of $h(t)$ with respect to the monomial t^k, that is, $\text{Coeff}_{Z^k}(h(t + Z))$.

Next, we mention some useful properties of Hasse derivatives.

Proposition 4.5. *Let $f(\mathbf{x})$ be an n-variate polynomial over $\mathbb{F}$. Let $\mathbf{a}, \mathbf{b} \in \mathbb{N}^n$. Then,*

1. $\overline{\partial}_{\mathbf{a}}(f) = \sum_{\mathbf{e} \in \mathbb{N}^n} \binom{\mathbf{e}}{\mathbf{a}} \text{Coeff}_{\mathbf{x}^{\mathbf{e}}}(f)\mathbf{x}^{\mathbf{e}-\mathbf{a}}$.
2. $\overline{\partial}_{\mathbf{a}}\overline{\partial}_{\mathbf{b}}(f) = \binom{\mathbf{a}+\mathbf{b}}{\mathbf{a},\mathbf{b}}\overline{\partial}_{\mathbf{a}+\mathbf{b}}(f)$.

For proof one can see [8, Appendix C]. The following lemma describes the cost of computing Hasse derivatives.

Lemma 4.6. *Let p be a prime and $q = p^a$ for some positive integer a. Let $f(\mathbf{x})$ be an n-variate polynomial over $\mathbb{F}_q$ with individual degree less than d. Let $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$. Then, given $f(\mathbf{x})$ and $\mathbf{b}$ as input, Algorithm 1 outputs $\overline{\partial}_{\mathbf{b}}(f)$ in*

$$d^n \cdot poly(n) + poly(b, d)$$

$\mathbb{F}_q$-operations, where $b = \max_{i \in [n]} b_i$.

Proof. We first describe the algorithm and then argue about its correctness and running time.

Algorithm 1 Computing Hasse derivative

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d and $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$.
Output: $\overline{\partial}_{\mathbf{b}}(f)$.

```
1:  Let $b$ be $\max_{i \in [n]} b_i$.
2:  Let $D$ be an $(b+1) \times d$ array.
3:  for $j \leftarrow 0$ to $d-1$ do
4:      for $i \leftarrow 0$ to $b$ do
5:          if $i = j$ then
6:              $D_{i,j} \leftarrow 1$.
7:          else if $i > j$ then
8:              $D_{i,j} \leftarrow 0$.
9:          else if $i = 0$ then
10:             $D_{0,j} \leftarrow 1$.
11:         else
12:             $D_{i,j} = D_{i-1,j-1} + D_{i,j-1}$
13: for $\mathbf{e} \in \{0, 1, \ldots, d-1\}^n$ do
14:     Let $\mathbf{e} = (e_1, \ldots, e_n)$.
15:     $c_{\mathbf{e}} \leftarrow \mathrm{Coeff}_{\mathbf{x}^{\mathbf{e}}}(f) \cdot \prod_{i=1}^{n} D_{b_i, e_i}$.
16: Output $\sum_{\mathbf{e} \in \{0,1,\ldots,d-1\}^n} c_{\mathbf{e}} \mathbf{x}^{\mathbf{e}-\mathbf{b}}$.
```

In Algorithm 1, for all $i \in \{0, 1, \ldots, b\}$ and $j \in \{0, 1, \ldots, d-1\}$, the (i, j)th entry of array D

$$D_{i,j} = \binom{j}{i} \bmod p.$$

For this, we note that the arithmetic in Line 15 of the algorithm is happening over the underlying field $\mathbb{F}_q$. This combined with Proposition 4.5 implies that the Algorithm 1 computes $\overline{\partial}_{\mathbf{b}}(f)$.

To compute the array D, we are performing $d(b+1)$ $\mathbb{F}_p$-operations. Computing all $c_{\mathbf{e}}$'s for $\mathbf{e} \in \{0, 1, \ldots, d-1\}^n$ takes $d^n \cdot (n+1)$ $\mathbb{F}_q$-operations. Therefore, Algorithm 1 runs in our desired time complexity. □

4.3 Univariate Polynomial Evaluation and Interpolation

The two simplest but most important ways of representing an univariate polynomial of degree less than d are either by giving the list of its coefficients, or by giving its evaluations at d distinct points. In this section, we discuss about the cost of changing between these two representations. First, we mention the cost of polynomial evaluation, that is, going from the list of coefficients to the list of evaluations.

Lemma 4.7 (Evaluation)**.** *Let $f(x)$ be a degree d polynomial over $\mathbb{F}$. Let $\alpha_1, \alpha_2, \ldots, \alpha_N$ be N distinct elements from $\mathbb{F}$. Then, $f(\alpha_i)$ for all $i \in [N]$ can be computed in $O(Nd)$ $\mathbb{F}$-operations.*

For each $i \in [N]$, using Horner's rule, one can compute $f(\alpha_i)$ with $d-1$ additions and $d-1$ multiplications over $\mathbb{F}$. Therefore, the total cost of computing $f(\alpha_i)$ for all $i \in [N]$ is $O(Nd)$ operations. For more details see [9, Section 5.2]. Next, we discuss the cost of polynomial interpolation where we go from the list of evaluations to the list of coefficients.

Lemma 4.8 (Interpolation)**.** *Let $f(x)$ be a degree d polynomial over $\mathbb{F}$. Let $\alpha_0, \alpha_1, \ldots, \alpha_d$ be $(d+1)$ distinct elements from $\mathbb{F}$. Let $\beta_i = f(\alpha_i)$ for all $i \in \{0, 1, \ldots, d\}$. Then, given (α_i, β_i) for all $i \in \{0, 1, \ldots, d\}$, $f(x)$ can be computed in $O(d^2)$ $\mathbb{F}$-operations.*

For proof see [9, Section 5.2]. The following lemma gives a stronger version of univariate polynomial interpolation, known as Hermite interpolation. Here, the number of evaluation points can be less than d, but evaluations of Hasse derivatives of the polynomial up to certain order is available.

Lemma 4.9 (Hermite interpolation)**.** *Let $f(x)$ be a degree d univariate polynomial over a field $\mathbb{F}$ and $e_1, \ldots, e_m$ be m positive integers such that $e_1 + \cdots + e_m$ is greater than d. Let $\alpha_1, \ldots, \alpha_m$ be m distinct elements from $\mathbb{F}$. For all $i \in [m]$ and $j \in [e_j]$, let $f^{(j-1)}(\alpha_i) = \beta_{ij}$. Then given $(\alpha_i, j, \beta_{ij})$ for all $i \in [m]$ and $j \in [e_j]$, $f(x)$ can be computed in $O(d^2)$ $\mathbb{F}$-operations.*

For proof see [9, Section 5.6]. We also remark that while there are nearly linear time algorithms for all of the above operations (multipoint evaluation, interpolation and Hermite interpolation) based on the Fast Fourier transform. However, for our applications in this paper, the above stated more naive bounds suffice.

4.4 Multidimensional Fast Fourier Transform

We crucially rely on the following lemma,

Lemma 4.10. *Let $\mathbb{F}$ be a finite field and let $\tilde{\mathbb{F}}$ be a subfield of $\mathbb{F}$. Then, there is a deterministic algorithm that takes as input an n-variate polynomial $f \in \mathbb{F}[\mathbf{x}]$ of degree at most $d-1$ in each variable as a list of coefficients, and in at most $(d^n + |\tilde{\mathbb{F}}|^n) \cdot \mathrm{poly}(n, d, \log |\mathbb{F}|)$ operations over the field $\mathbb{F}$, it outputs the evaluation of f for all $\boldsymbol{\alpha} \in \tilde{\mathbb{F}}^n$.*

5 A SIMPLE ALGORITHM FOR MULTIPOINT EVALUATION

We start with our first and simplest algorithm for multipoint evaluation . The algorithm gives an inferior time complexity to what is claimed in Theorem 2.1, but contains some of the main ideas. Subsequently, in Section 6 and Section 7, we build upon this algorithm to eventually prove Theorem 2.1. Our main theorem for this section is the following.

Theorem 5.1. *Let p be a prime and $q = p^a$ for some positive integer a. There is a deterministic algorithm such that on input an n-variate polynomial $f(\mathbf{x})$ over $\mathbb{F}_q$ with individual degree less than d and points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$, it outputs $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in time*

$$(N + (adnp)^n) \cdot \mathrm{poly}(a, d, n, p)\,.$$

5.1 A Description of the Algorithm

We start with a description of the algorithm, followed by its analysis. We recall again that through all the algorithms in this and subsequent sections, we assume that the underlying field $\mathbb{F}_q$ is given to us via an irreducible polynomial of appropriate degree over the prime subfield. Moreover, from Observation 4.2, we also assume without loss of generality that for every input field element, we have access to its representation as a polynomial of appropriate degree over the prime subfield. For a polynomial map $\mathbf{g}(t) = (g_1(t), g_2(t), \ldots, g_n(t))$ and an n-variate polynomial f, we use $f(\mathbf{g}(t))$ to denote the univariate polynomial $f(g_1(t), \ldots, g_n(t))$.

Algorithm 2 Efficient Multivariate Multipoint Evaluation

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d and N distinct points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$.
Output: $f(\boldsymbol{\alpha}_1), f(\boldsymbol{\alpha}_2), \ldots, f(\boldsymbol{\alpha}_N)$.

```
1: Let $p$ be the characteristic of $\mathbb{F}_q$ and $q = p^a$.
2: Let $v_0(Y_0)$ be an irreducible polynomial in $\mathbb{F}_p[Y_0]$ of degree $a$ and
       $\mathbb{F}_q = \mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle$.
3: Let $b$ be the smallest integer such that $p^b > adn$.
4: Compute an irreducible polynomial $v_1(Y_1)$ in $\mathbb{F}_q[Y_1]$ of degree $b$ and
       $\mathbb{F}_{q^b} = \mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$. (Lemma 4.1)
5: Compute the subfield $\mathbb{F}_{p^b}$ of $\mathbb{F}_{q^b}$. (Lemma 4.3)
6: Evaluate $f(\mathbf{x})$ over the grid $\mathbb{F}_{p^b}^n$. (Lemma 4.10)
7: for all $i \in [N]$ do
8:     Let $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}Y_0 + \cdots + \boldsymbol{\alpha}_{i,a-1}Y_0^{a-1}$, where $\boldsymbol{\alpha}_{i,j} \in \mathbb{F}_p^n$.
9:     Let $\mathbf{g}_i(t)$ be the curve defined as $\boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}t + \cdots + \boldsymbol{\alpha}_{i,a-1}t^{a-1}$.
10:    Compute the set $P_i = \{(\gamma, \mathbf{g}_i(\gamma)) \mid \gamma \in \mathbb{F}_{p^b}\}$. (Lemma 4.7)
11:    Compute the set $E_i = \{(\gamma, f(\boldsymbol{\gamma}')) \mid (\gamma, \boldsymbol{\gamma}') \in P_i\}$ from the evaluations of $f(\mathbf{x})$ over $\mathbb{F}_{p^b}^n$.
12:    Let $h_i(t)$ be the univariate polynomial defined as $f(\mathbf{g}_i(t))$.
13:    Using $E_i$, interpolate $h_i(t)$. (Lemma 4.8)
14:    Output $h_i(Y_0)$ as $f(\boldsymbol{\alpha}_i)$. (Lemma 4.7)
```

5.2 Analysis of Algorithm 2

Proof of Theorem 5.1. Proof of correctness can be found in the full version [3] of our result.

Time Complexity of Algorithm 2. From Lemma 4.1, the field $\mathbb{F}_{q^b}$ can be constructed as $\mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$ for some degree b irreducible polynomial $v_1(Y_1)$ over $\mathbb{F}_q$ in poly(a, b, p) many $\mathbb{F}_q$-operations. Also, all the basic operations in the field $\mathbb{F}_{q^b} = \mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$ can be done using poly(b) $\mathbb{F}_q$-operations. Applying Lemma 4.3, the cost of computing all the elements of the subfield $\mathbb{F}_{p^b}$ (of $\mathbb{F}_{q^b}$) is $p^b \cdot \text{poly}(a, b, p)$ $\mathbb{F}_q$-operations. Using Lemma 4.10, we can evaluate $f(\mathbf{x})$ over the grid $\mathbb{F}_{p^b}^n$ in

$$(d^n + p^{bn}) \cdot \text{poly}(a, b, d, n, p)$$

$\mathbb{F}_q$-operations. For all $i \in [N]$, using Lemma 4.7, the cost of computing the set $P_i = \{(\gamma, \mathbf{g}_i(\gamma)) \mid \gamma \in \mathbb{F}_{p^b}\}$ is $p^b \cdot \text{poly}(a, b, n)$ $\mathbb{F}_q$-operations. Using the set E_i, Lemma 4.8 ensures that $h_i(t)$ can be interpolated using poly(a, b, d, n) $\mathbb{F}_q$-operations. Finally, $h(Y_0)$ can be computed in poly(a, d, n) many $\mathbb{F}_q$-operations. Since $adn < p^b \le adnp$, the above discussion implies that that Algorithm 2 performs

$$(N + (adnp)^n) \cdot \text{poly}(a, d, n, p)$$

$\mathbb{F}_q$-operations. □

6 MULTIPOINT EVALUATION FOR LARGE NUMBER OF VARIABLES

In this section, we append the overall structure of Algorithm 2 with some more ideas to improve the dependence of the running time on n. In particular, the goal is to reduce the n^n factor in the running time of Theorem 5.1 to a factor of the form $\exp(O(n))$. The main result of this section is the following theorem.

Theorem 6.1. *Let p be a prime and $q = p^a$ for some positive integer a. There is a deterministic algorithm such that on input an n-variate polynomial $f(\mathbf{x})$ over $\mathbb{F}_q$ with individual degree less than d and points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$, it outputs $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in time*

$$(N + (adp)^n) \cdot 4^n \cdot poly(a, d, n, p).$$

A useful additional ingredient in the proof of this theorem is the following lemma.

Lemma 6.2. *Let $f(\mathbf{x})$ be an n-variate degree d polynomial over a field $\mathbb{F}$, $\mathbf{g}(t) = (g_1, \ldots, g_n)$ where $g_i \in \mathbb{F}[t]$, and $h(t) = f(\mathbf{g}(t))$. For all $i \in [n]$, let $g_i(t + Z) = g_i(t) + Z\tilde{g}_i(t, Z)$ for some $\tilde{g}_i \in \mathbb{F}[t, Z]$. Let $\tilde{\mathbf{g}}(t, Z) = (\tilde{g}_1, \ldots, \tilde{g}_n)$, and for all $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$, $\tilde{\mathbf{g}}_{\mathbf{e}} = \prod_{i=1}^n \tilde{g}_i^{e_i}$. For any $\ell \in \mathbb{N}$, let*

$$h_\ell(t, Z) = \sum_{i=0}^{\ell} Z^i \sum_{\mathbf{e} \in \mathbb{N}^n : |\mathbf{e}|_1 = i} \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}(t)) \cdot \tilde{\mathbf{g}}_{\mathbf{e}}(t, Z).$$

Then, for every $k \in \mathbb{N}$ with $k \le \ell$, $h^{(k)}(t) = \text{Coeff}_{Z^k}(h_\ell)$.

6.1 A Description of the Algorithm

We start by describing the algorithm.

Algorithm 3 Efficient multivariate polynomial evaluation with large number of variables

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d and N points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$.
Output: $f(\boldsymbol{\alpha}_1), f(\boldsymbol{\alpha}_2), \ldots, f(\boldsymbol{\alpha}_N)$.

```
1: Let $p$ be the characteristic of $\mathbb{F}_q$ and $q = p^a$.
2: Let $v_0(Y_0)$ be an irreducible polynomial in $\mathbb{F}_p[Y_0]$ of degree $a$ and
       $\mathbb{F}_q = \mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle$.
3: Let $b$ the smallest positive integer such that $p^b > ad$.
4: Compute an irreducible polynomial $v_1(Y_1)$ in $\mathbb{F}_q[Y_1]$ of degree $b$ and
       $\mathbb{F}_{q^b} = \mathbb{F}_q[Y_1]/\langle v_1(Y_1)\rangle$. (Lemma 4.1)
5: Compute the subfield $\mathbb{F}_{p^b}$ of $\mathbb{F}_{q^b}$. (Lemma 4.3)
6: Compute the set $\overline{\partial}^{<n}(f)$. (Lemma 4.6)
7: Evaluate all the polynomials in $\overline{\partial}^{<n}(f)$ over the grid $\mathbb{F}_{p^b}^n$. (Lemma 4.10)
8: for all $i \in [N]$ do
9:     Let $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}Y_0 + \cdots + \boldsymbol{\alpha}_{i,a-1}Y_0^{a-1}$, where $\boldsymbol{\alpha}_{i,j} \in \mathbb{F}_p^n$.
10:    Let $\mathbf{g}_i(t)$ be the curve defined as $\boldsymbol{\alpha}_{i,0} + \boldsymbol{\alpha}_{i,1}t + \cdots + \boldsymbol{\alpha}_{i,a-1}t^{a-1}$.
11:    Let $h_i(t) = f(\mathbf{g}_i(t))$.
12:    Let $E_i = \{(\gamma, h_i^{(0)}(\gamma), h_i^{(1)}(\gamma), \ldots, h_i^{(n-1)}(\gamma) \mid \gamma \in \mathbb{F}_{p^b}\}$.
13:    Invoke the function Evaluate Derivatives A with input $\mathbf{g}_i(t)$ and compute the set $E_i$.
14:    Using $E_i$, interpolate $h_i(t)$. (Lemma 4.9)
15:    Output $h_i(Y_0)$ as $f(\boldsymbol{\alpha}_i)$. (Lemma 4.7)
```

We now describe the function Evaluate Derivatives A invoked above. We follow the same notation as in Algorithm 3 including the local variable names.

Algorithm 4 Function to generate data for Hermite Interpolation

```
1: function EVALUATE DERIVATIVES A ($\mathbf{g}(t)$)
2:     Let $\mathbf{g}(t) = (g_1, \ldots, g_n)$.
3:     For all $i \in [n]$, let $g_i(t+Z) = g_i(t) + Z\tilde{g}_i(t,Z)$ and $\tilde{\mathbf{g}}(t,Z) = (\tilde{g}_1(t,Z), \ldots, \tilde{g}_n(t,Z))$.
4:     Compute $\tilde{g}_i(t,Z)$ for all $i \in [n]$. (Lemma 4.6)
5:     For all $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$, let $\tilde{\mathbf{g}}_{\mathbf{e}} = \prod_{i=1}^n \tilde{g}_i^{e_i}$.
6:     Compute the set of polynomials $\{\tilde{\mathbf{g}}_{\mathbf{e}}(t,Z) \mid |\mathbf{e}|_1 < n\}$. (Polynomial multiplication)
7:     $P \leftarrow \emptyset$.
8:     for all $\gamma \in \mathbb{F}_{p^b}$ do
9:         Using evaluations of polynomials in $\overline{\partial}^{<n}(f)$ over $\mathbb{F}_{p^b}^n$, compute the polynomial
               $h_\gamma(Z) = \sum_{i=0}^{n-1} Z^i \sum_{\mathbf{e}\in\mathbb{N}^n:|e|_1=i} \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}(\gamma))\tilde{\mathbf{g}}_{\mathbf{e}}(\gamma, Z).$
10:        For all $i \in \{0,1,\ldots,n-1\}$, extract $\mathrm{Coeff}_{Z^i}(h_\gamma)$.
11:        $P \leftarrow P \cup \{(\gamma, \mathrm{Coeff}_{Z^0}(h_\gamma), \mathrm{Coeff}_{Z^1}(h_\gamma), \ldots, \mathrm{Coeff}_{Z^{n-1}}(h_\gamma))\}$.
12:    return $P$.
```

Please refer to the full version [3] for a detailed analysis of Algorithm 3 and proof of Theorem 6.1.

7 MULTIPOINT EVALUATION WITH IMPROVED FIELD DEPENDENCE

In this section, we build on the ideas in Algorithm Theorem 6.1 to improve the dependence on the field size. Our main theorem, which is a formal statement of our main result Theorem 2.1 stated in the introduction.

Theorem 7.1. *Let p be a prime and $q = p^a$ for some positive integer a. There is a deterministic algorithm such that on input an n-variate polynomial $f(\mathbf{x})$ over $\mathbb{F}_q$ with individual degree less than d, points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$ and a non-negative integer $\ell \le \log_p^\star(a)$, it outputs $f(\boldsymbol{\alpha}_i)$ for all $i \in [N]$ in time*

$$\left(N \cdot \left(2dp\log_p(dp)\right)^\ell + \left(2rdp\log_p(dp)\right)^n\right) \cdot O(\ell+1)^n \cdot poly(a,d,n,p),$$

where $r = \max\{2, \log_p^{\circ \ell}(a)\}$.

7.1 A Description of the Algorithm

We start by describing the algorithm.

Algorithm 5 Efficient multivariate polynomial evaluation over large fields

Input: An n-variate polynomial $f(\mathbf{x}) \in \mathbb{F}_q[\mathbf{x}]$ with individual degree less than d, N points $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ from $\mathbb{F}_q^n$, and a non-negative integer $\ell \le \log_p^\star(a)$ where $q = p^a$ and p is the characteristic of $\mathbb{F}_q$.

Output: $f(\boldsymbol{\alpha}_1), \ldots, f(\boldsymbol{\alpha}_N)$.

```
1: Let $v_0(Y_0)$ be an irreducible polynomial in $\mathbb{F}_p[Y_0]$ of degree $a$ and $\mathbb{F}_q = \mathbb{F}_p[Y_0]/\langle v_0(Y_0)\rangle$.
2: $\mathsf{Points}_0 \leftarrow \{\boldsymbol{\alpha}_i \mid i \in [N]\}$, $a_0 \leftarrow a$, and $q_0 \leftarrow p^a$.
3: POLYNOMIAL EVALUATION(0).    (Recursive call)
4: Output $\mathsf{Eval}_{0,\mathbf{0}}$.
5: function POLYNOMIAL EVALUATION($i$)
6:     Let $a_{i+1}$ be the smallest positive integer such that $p^{a_{i+1}} > a_i d$, and $q_{i+1} \leftarrow q^{a_{i+1}}$.
7:     Compute an irreducible polynomial $v_{i+1}(Y_{i+1})$ over $\mathbb{F}_q$ of degree $a_{i+1}$ and
           $\mathbb{F}_{q_{i+1}} = \mathbb{F}_q[Y_{i+1}]/\langle v_{i+1}(Y_{i+1})\rangle$. (Lemma 4.1)
8:     Compute the subfield $\mathbb{F}_{p^{a_{i+1}}}$ of $\mathbb{F}_{q_{i+1}}$. (Lemma 4.3)
9:     Compute an element $\beta_i$ in $\mathbb{F}_{q_i}$ s.t. $\{1, \beta_i, \ldots, \beta_i^{a_i-1}\}$ forms an $\mathbb{F}_p$-basis for $\mathbb{F}_{p^{a_i}}$. (Lemma 4.3)
10:    $\mathsf{Points}_{i+1} \leftarrow \emptyset$.
11:    for all $\boldsymbol{\alpha} \in \mathsf{Points}_i$ do
12:        Let $\boldsymbol{\alpha} = \boldsymbol{\alpha}_0 + \boldsymbol{\alpha}_1\beta_i + \cdots + \boldsymbol{\alpha}_{a_i-1}\beta_i^{a_i-1}$, where $\boldsymbol{\alpha}_j \in \mathbb{F}_p^n$.
13:        Compute $\boldsymbol{\alpha}_0, \ldots, \boldsymbol{\alpha}_{a_i-1}$. (Lemma 4.3)
14:        Let $\mathbf{g}_{\boldsymbol{\alpha}}(t)$ be the curve defined as $\boldsymbol{\alpha}_0 + \boldsymbol{\alpha}_1 t + \cdots + \boldsymbol{\alpha}_{a_i-1}t^{a_i-1}$.
15:        $P_{\boldsymbol{\alpha}} \leftarrow \{\mathbf{g}_{\boldsymbol{\alpha}}(\gamma) \mid \gamma \in \mathbb{F}_{p^{a_{i+1}}}\}$ (Lemma 4.7), and $\mathsf{Points}_{i+1} \leftarrow \mathsf{Points}_{i+1} \cup P_{\boldsymbol{\alpha}}$.
16:    if $i < \ell$ then
17:        POLYNOMIAL EVALUATION($i+1$).
18:    else
19:        Compute all the polynomials in $\overline{\partial}^{\le(\ell+1)(n-1)}(f)$. (Lemma 4.6)
20:        Evaluate all the polynomials in $\overline{\partial}^{\le(\ell+1)(n-1)}(f)$ over the grid $\mathbb{F}_{p^{a_{\ell+1}}}^n$. (Lemma 4.10)
21:        Observe that $\mathsf{Points}_{\ell+1}$ is a subset of $\mathbb{F}_{p^{a_{\ell+1}}}^n$.
22:        For all $\mathbf{e} \in \mathbb{N}^n$ with $|\mathbf{e}|_1 \le (\ell+1)(n-1)$, $\mathsf{Eval}_{\ell+1,\mathbf{e}} = \{(\boldsymbol{\alpha}, \overline{\partial}_{\mathbf{e}}(f)(\boldsymbol{\alpha})) \mid \boldsymbol{\alpha} \in \mathbb{F}_{p^{a_{\ell+1}}}^n\}$.
23:    for all $\mathbf{e} \in \mathbb{N}^n$ s.t. $|\mathbf{e}|_1 \le i(n-1)$ do
24:        $\mathsf{Eval}_{i,\mathbf{e}} \leftarrow \emptyset$.
25:        for all $\boldsymbol{\alpha} \in \mathsf{Points}_i$ do
26:            Let $h_{\mathbf{e},\boldsymbol{\alpha}}(t) = \overline{\partial}_{\mathbf{e}}(f)(\mathbf{g}_{\boldsymbol{\alpha}}(t))$.
27:            Let $E_{\mathbf{e},\boldsymbol{\alpha}} = \{(\gamma, h_{\mathbf{e},\boldsymbol{\alpha}}^{(0)}(\gamma), \ldots, h_{\mathbf{e},\boldsymbol{\alpha}}^{(n-1)}(\gamma)) \mid \gamma \in \mathbb{F}_{p^{a_{i+1}}}\}$.
28:            Using EVALUATE DERIVATIVES B with input $(\mathbf{g}_{\boldsymbol{\alpha}}(t), i, \mathbf{e})$, compute $E_{\mathbf{e},\boldsymbol{\alpha}}$.
29:            Using $E_{\mathbf{e},\boldsymbol{\alpha}}$, interpolate $h_{\mathbf{e},\boldsymbol{\alpha}}(t)$. (Lemma 4.9)
30:            $\mathsf{Eval}_{i,\mathbf{e}} \leftarrow \mathsf{Eval}_{i,\mathbf{e}} \cup \{(\boldsymbol{\alpha}, h_{\mathbf{e},\boldsymbol{\alpha}}(\beta_i))\}$. (Lemma 4.7)
```

Algorithm 6 Evaluating Hasse derivatives for Algorithm 5

```
1:  function EVALUATE DERIVATIVES B ($\mathbf{g}(t)$, $k$, $\mathbf{e}$)
2:      Let $\mathbf{g}(t) = (g_1, \ldots, g_n)$.
3:      Let $\mathbf{e} = (e_1, \ldots, e_n)$.
4:      Let $g_i(t+Z) = g_i(t) + Z\tilde{g}_i(t,Z)$, for all $i \in [n]$, and $\tilde{\mathbf{g}}(t,Z) = (\tilde{g}_1(t,Z), \ldots, \tilde{g}_n(t,Z))$.
5:      Compute $\tilde{g}_i(t,Z)$ for all $i \in [n]$. (Lemma 4.6)
6:      For all $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$, let $\tilde{\mathbf{g}}_{\mathbf{b}} = \prod_{i=1}^{n} \tilde{g}_i^{b_i}$.
7:      Compute the set of polynomials $\{\tilde{\mathbf{g}}_{\mathbf{b}}(t,Z) \mid |\mathbf{b}|_1 < n\}$. (Polynomial multiplication)
8:      Let $D$ be an $((k+1)(n-1)+1) \times n$ array such that,
            $D_{i,j} = \binom{j}{i} \bmod p$, where $i \in \{0,1,\ldots,n-1\}, j \in \{0,1,\ldots,(k+1)(n-1)\}$
9:      Like Algorithm 1, we can compute $D$ using $\mathbb{F}_p$-operations.
10:     For $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$ such that $|\mathbf{b}|_1 < n$,
            $c_{\mathbf{b}} \leftarrow \prod_{i=1}^{n} D_{e_i+b_i, b_i}$.
11:     $P \leftarrow \emptyset$.
12:     for all $\gamma \in \mathbb{F}_{p^{a_{k+1}}}$ do
13:         Using evaluations of polynomials in $\overline{\partial}^{\le (k+1)(n-1)}(f)$ over Points$_{k+1}$, compute
                $h_\gamma(Z) = \sum_{i=0}^{n-1} Z^i \sum_{\mathbf{b} \in \mathbb{N}^n : |\mathbf{b}|_1 = i} c_{\mathbf{b}} \overline{\partial}_{\mathbf{e}+\mathbf{b}}(f)(\mathbf{g}(\gamma)) \tilde{\mathbf{g}}_{\mathbf{b}}(\gamma, Z)$.
14:         For all $i \in \{0,1,\ldots,n-1\}$, extract Coeff$_{Z^i}(h_\gamma)$.
15:         $P \leftarrow P \cup \{(\gamma, \mathrm{Coeff}_{Z^0}(h_\gamma), \mathrm{Coeff}_{Z^1}(h_\gamma), \ldots, \mathrm{Coeff}_{Z^{n-1}}(h_\gamma))\}$.
16:     return $P$.
```

Please refer to the full version [3] for a detailed analysis of Algorithm 5 and proof of Theorem 7.1.

ACKNOWLEDGMENTS

Mrinal thanks Swastik Kopparty for introducing him to the work of Kedlaya & Umans [10] and the question of multipoint evaluation and numerous invaluable discussions.

We thank Zeev Dvir and Allen Liu for answering our questions regarding the results in [7] and for allowing us to include a proof sketch in the full version [3]. We also thank Ben Lund for helpful discussions and references on the finite fields Kakeya problem and for pointing us to the relevant literature on Furstenberg sets, both of which indirectly played a role in some of the ideas in this paper.

Finally, we thank Prerona Chatterjee, Prahladh Harsha, Ramprasad Saptharishi and Aparna Shankar for sitting through a (pretty sketchy) presentation of earlier versions of some of the proofs in this paper and for much encouragement.

REFERENCES

[1] Josh Alman. 2021. Kronecker products, low-depth circuits, and matrix rigidity. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. ACM, 772–785. https://doi.org/10.1145/3406325.3451008

[2] Josh Alman and R. Ryan Williams. 2017. Probabilistic rank and matrix rigidity. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)* (Montreal, QC, Canada, 19–23 June). 641–652. https://doi.org/10.1145/3055399.3055484 arXiv:1611.05558

[3] Vishwas Bhargava, Sumanta Ghosh, Mrinal Kumar, and Chandra Kanta Mohapatra. 2021. Fast, Algebraic Multivariate Multipoint Evaluation in Small Characteristic and Applications. *CoRR* abs/2111.07572 (2021). arXiv:2111.07572 https://arxiv.org/abs/2111.07572

[4] Andreas Björklund, Petteri Kaski, and Ryan Williams. 2019. Generalized Kakeya sets for polynomial evaluation and faster computation of fermionants. *Algorithmica* 81, 10 (2019), 4010–4028. https://doi.org/10.1007/s00453-018-0513-7

[5] A. Borodin and R. Moenck. 1974. Fast modular transforms. *J. Comput. System Sci.* 8, 3 (1974), 366–386. https://doi.org/10.1016/S0022-0000(74)80029-2

[6] Zeev Dvir and Benjamin L. Edelman. 2019. Matrix Rigidity and the Croot-Lev-Pach Lemma. *Theory Comput.* 15, 8 (2019), 1–7. https://doi.org/10.4086/toc.2019.v015a008 arXiv:1708.01646

[7] Zeev Dvir and Allen Liu. 2020. Fourier and Circulant Matrices are Not Rigid. *Theory of Computing* 16, 20 (2020), 1–48. https://doi.org/10.4086/toc.2020.v016a020

[8] Michael Forbes. 2014. *Polynomial Identity Testing of Read-Once Oblivious Algebraic Branching Programs.* Ph. D. Dissertation. Massachusetts Institute of Technology. http://hdl.handle.net/1721.1/89843

[9] Joachim Von Zur Gathen and Jurgen Gerhard. 2003. *Modern Computer Algebra.* Cambridge University Press, New York, NY, USA.

[10] Kiran S. Kedlaya and Christopher Umans. 2011. Fast Polynomial Factorization and Modular Composition. *SIAM J. Comput.* 40, 6 (2011), 1767–1802. https://doi.org/10.1137/08073408X

[11] Bohdan Kivva. 2021. Improved Upper Bounds for the Rigidity of Kronecker Products. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 202)*, Filippo Bonchi and Simon J. Puglisi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 68:1–68:18. https://doi.org/10.4230/LIPIcs.MFCS.2021.68

[12] Satyanarayana V. Lokam. 2000. On the rigidity of Vandermonde matrices. *Theor. Comput. Sci.* 237, 1-2 (2000), 477–483. https://doi.org/10.1016/S0304-3975(00)00008-6

[13] Peter Bro Miltersen. 1995. On the Cell Probe Complexity of Polynomial Evaluation. *Theor. Comput. Sci.* 143, 1 (May 1995), 167–174. https://doi.org/10.1016/0304-3975(95)80032-5

[14] Michael Nüsken and Martin Ziegler. 2004. Fast Multipoint Evaluation of Bivariate Polynomials. In *Algorithms – ESA 2004*, Susanne Albers and Tomasz Radzik (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 544–555.

[15] Christopher Umans. 2008. Fast polynomial factorization and modular composition in small characteristic. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, Cynthia Dwork (Ed.). ACM, 481–490. https://doi.org/10.1145/1374376.1374445

[16] Leslie G. Valiant. 1977. Graph-Theoretic Arguments in Low-Level Complexity. In *Proc. 6th Symposium of Mathematical Foundations of Computer Science (MFCS)* (Tatranska Lomnica, Czechoslovakia, 5–9 Sept.) *(LNCS, Vol. 53)*, Jozef Gruska (Ed.). Springer, 162–176. https://doi.org/10.1007/3-540-08353-7_135