Fast Multivariate Multipoint Evaluation

Based on joint works with various subsets of

V Bhargava, S Ghosh, Z Guo, P Harsha, S Herdade, C K Mohapatra, R Saptharishi, C Umans

Multipoint evaluation

Input

- An m-variate polynomial f with degree at most (d-1) in each variable over a field **K**, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}^{\mathbf{m}}$

Output

- Evaluation of f on $\alpha_1, \alpha_2, \ldots, \alpha_N$

Multipoint evaluation

Input

- An m-variate polynomial f with degree at most (d-1) in each variable over a field **K**, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}^{\mathbf{m}}$

Output

- Evaluation of f on $\alpha_1, \alpha_2, \ldots, \alpha_N$

Input: $(d^m + Nm)$ field elements

Multipoint evaluation

Multipoint evaluation

Naïve algorithm

Multipoint evaluation

Naïve algorithm

For i = 1 to N:

Evaluate f on α_i

Multipoint evaluation

Naïve algorithm

For i = 1 to N:

Evaluate f on α_i

Roughly (Nmd^m) field operations in total

Multipoint evaluation

Naïve algorithm

For i = 1 to N:

Evaluate f on α_i

Roughly (Nmd^m) field operations in total

When $N = d^m$, quadratic in the input size

Multipoint evaluation

Naïve algorithm

For i = 1 to N:

Evaluate f on α_i

Roughly (Nmd^m) field operations in total

When $N = d^m$, quadratic in the input size

Can we do this faster ?

Multipoint evaluation

Naïve algorithm

For i = 1 to N:

Evaluate f on α_i

Roughly (Nmd^m) field operations in total

When $N = d^m$, quadratic in the input size

Can we do this faster ?

In particular, is there an algorithm that runs in linear time in the input size ?

Multipoint evaluation <u>over infinite fields</u>

Multipoint evaluation <u>over infinite fields</u>

Seeking a nearly linear time algorithm over finite fields is reasonable, since the output description is nearly linear in the input description

Multipoint evaluation over infinite fields

Seeking a nearly linear time algorithm over finite fields is reasonable, since the output description is nearly linear in the input description

No longer true over infinite fields!

Multipoint evaluation over infinite fields

Seeking a nearly linear time algorithm over finite fields is reasonable, since the output description is nearly linear in the input description

No longer true over infinite fields!

Evaluate $f(x) = x^d$ at 1, 2, …., d

Multipoint evaluation over infinite fields

Seeking a nearly linear time algorithm over finite fields is reasonable, since the output description is nearly linear in the input description

No longer true over infinite fields!

Evaluate $f(x) = x^d$ at 1, 2, …., d

Total output size is at least quadratic in d

Multipoint evaluation over infinite fields

Seeking a nearly linear time algorithm over finite fields is reasonable, since the output description is nearly linear in the input description

No longer true over infinite fields!

Evaluate $f(x) = x^d$ at 1, 2, …., d

Total output size is at least quadratic in d

Various versions: nearly linear time in the output size, input points with small absolute value, computing approximations of the evaluations, or count field operations only

Approximate multipoint evaluation (rationals/reals/complexes)

Input

- An m-variate polynomial f with degree at most (d-1) in each variable over rational numbers, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in Q^m$, from the unit cube
- Accuracy parameter t

Output

- Rational numbers $\beta_1, \beta_2, \ldots, \beta_N$ such that $|f(\alpha_i) - \beta_i| < 1/2^t$

Why do we care ?

Why do we care ?

- A very basic and natural algorithmic question in computational algebra

Why do we care ?

- A very basic and natural algorithmic question in computational algebra
- Many direct and natural applications – fast modular composition, univariate polynomial factorization over finite fields, generating irreducible polynomials, computing minimal polynomials, data structures for polynomial evaluation, ….

Why do we care ?

- A very basic and natural algorithmic question in computational algebra
- Many direct and natural applications – fast modular composition, univariate polynomial factorization over finite fields, generating irreducible polynomials, computing minimal polynomials, data structures for polynomial evaluation, ...

- Current fastest algorithms for all these problems go via fast multipoint evaluation

Faster-than-trivial multipoint evaluation

What do we know ?

Faster-than-trivial multipoint evaluation

What do we know ?

- Depends on the number of variables

Faster-than-trivial multipoint evaluation

What do we know ?

- Depends on the number of variables
- For the univariate case ($m = 1$)....pretty good understanding of the problem over all fields

Faster-than-trivial multipoint evaluation

What do we know ?

- Depends on the number of variables
- For the univariate case ($m = 1$)….pretty good understanding of the problem over all fields
- In particular, nearly linear time algorithms known

Faster-than-trivial multipoint evaluation

What do we know ?

- Depends on the number of variables
- For the univariate case ($m = 1$)….pretty good understanding of the problem over all fields
- In particular, nearly linear time algorithms known
- For the multivariate case ($m > 1$)…much less understood

Faster-than-trivial multipoint evaluation

What do we know ?

- Depends on the number of variables
- For the univariate case (m = 1)....pretty good understanding of the problem over all fields
- In particular, nearly linear time algorithms known
- For the multivariate case (m > 1)...much less understood

Multipoint evaluation: the univariate case

Multipoint evaluation: the univariate case

Input

- A univariate polynomial f with degree (d-1) over a field **K**, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$

Multipoint evaluation: the univariate case

Input

- A univariate polynomial f with degree (d-1) over a field **K**, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$

Output

- Evaluation of f on $\alpha_1, \alpha_2, \ldots, \alpha_N$

Multipoint evaluation: the univariate case

Input

- A univariate polynomial f with degree (d-1) over a field **K**, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$

Output

- Evaluation of f on $\alpha_1, \alpha_2, \ldots, \alpha_N$

Input is specified via $(N + d)$ field elements

Multipoint evaluation: the univariate case

Multipoint evaluation: the univariate case

For structured set of input points

Multipoint evaluation: the univariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ are all roots of unity of order N

Multipoint evaluation: the univariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ are all roots of unity of order N
- an algorithm with $(N + d)^{1+o(1)}$ field operations using Fast Fourier Transform

Multipoint evaluation: the univariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ are all roots of unity of order N
- an algorithm with $(N + d)^{1+o(1)}$ field operations using Fast Fourier Transform

For an arbitrary set of input points

Multipoint evaluation: the univariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ are all roots of unity of order N
- an algorithm with $(N + d)^{1+o(1)}$ field operations using Fast Fourier Transform

For an arbitrary set of input points

- **[Borodin-Moenck, 1974]** An algorithm with $(N + d)^{1+o(1)}$ field operations

Multipoint evaluation: the univariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ are all roots of unity of order N
- an algorithm with $(N + d)^{1+o(1)}$ field operations using Fast Fourier Transform

For an arbitrary set of input points

- **[Borodin-Moenck, 1974]** An algorithm with $(N + d)^{1+o(1)}$ field operations
- a very clever and neat application of FFT

Multipoint evaluation: the univariate case

For an arbitrary set of input points

- **[Moroz, 2019]** An nearly linear time algorithm for approximate univariate MME
- Based on known algorithms for approximate FFT + beautiful geometric ideas

Multipoint evaluation: the multivariate case

Multipoint evaluation: the multivariate case

For structured set of input points

Multipoint evaluation: the multivariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ form a product set, i.e., $\{\alpha_1, \alpha_2, \ldots, \alpha_N\} = S_1 \times S_2 \times \cdots \times S_m$, for $S_i \subseteq \mathbf{K}$

Multipoint evaluation: the multivariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ form a product set, i.e., $\{\alpha_1, \alpha_2, \ldots, \alpha_N\} = S_1 \times S_2 \times \cdots \times S_m$, for $S_i \subseteq \mathbf{K}$
- an easy nearly linear time algorithm – induction on the number of variables

Multipoint evaluation: the multivariate case

For structured set of input points

- when $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}$ form a product set, i.e., $\{\alpha_1, \alpha_2, \ldots, \alpha_N\} = S_1 \times S_2 \times \cdots \times S_m$, for $S_i \subseteq \mathbf{K}$
- an easy nearly linear time algorithm – induction on the number of variables
- uses the univariate case as the base case

Multipoint evaluation: the multivariate case

For an arbitrary set of input points

Multipoint evaluation: the multivariate case

For an arbitrary set of input points

- no non-trivial algorithm known till relatively recently (even for the bivariate case)

Multipoint evaluation: the multivariate case

For an arbitrary set of input points

- no non-trivial algorithm known till relatively recently (even for the bivariate case)
- Nusken-Ziegler designed a slightly faster (though far from linear time) algorithm in 2004

Multipoint evaluation: the multivariate case

For an arbitrary set of input points

- no non-trivial algorithm known till relatively recently (even for the bivariate case)
- Nusken-Ziegler designed a slightly faster (though far from linear time) algorithm in 2004
- based on faster rectangular matrix multiplication

The multivariate case: more recent progress

The multivariate case: more recent progress

[Umans, 2008]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. char(**K**) is less than $d^{o(1)}$
2. number of variables (m) is less than $d^{o(1)}$

The multivariate case: more recent progress

[Umans, 2008]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. char(**K**) is less than $d^{o(1)}$
2. number of variables (m) is less than $d^{o(1)}$

[Kedlaya, Umans, 2008]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. **K** is any finite field
2. number of variables (m) is less than $d^{o(1)}$

The multivariate case: more recent progress

[Bjorklund, Kaski, Williams, 2019]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. |**K**| is small
2. |**K**|-1 has small divisors

The multivariate case: more recent progress

[Bjorklund, Kaski, Williams, 2019]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. |**K**| is small
2. |**K**|-1 has small divisors

Not a polynomial time algorithm, since the running time depends polynomially (and not polylogarithmically) on the field size

Nevertheless, happens to be very useful for one of our results

Multivariate multipoint evaluation

In particular

Multivariate multipoint evaluation

In particular

No nearly linear time algorithm for multivariate multipoint evaluation when

- number of variables (m) is not less than $d^{o(1)}$, over any (sufficiently large) field

Multivariate multipoint evaluation

In particular

No nearly linear time algorithm for multivariate multipoint evaluation when

- number of variables (m) is not less than $d^{o(1)}$, over any (sufficiently large) field

This is the question that we study in our work and focus of rest of the talk.

Our results

Our results

[Bhargava, Ghosh, K., Mohapatra, 2021]

A nearly linear time algorithm for multivariate multipoint evaluation when

Our results

[Bhargava, Ghosh, K., Mohapatra, 2021]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. char(**K**) is less than $d^{o(1)}$
2. **K** is of size at most exp(exp(exp(…exp(d)))) (tower of fixed height)

Our results

[Bhargava, Ghosh, K., Mohapatra, 2021]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. char(**K**) is less than $d^{o(1)}$
2. **K** is of size at most exp(exp(exp(…exp(d)))) (tower of fixed height)
3. ~~number of variables (m) is less than $d^{o(1)}$~~

Our results

[Bhargava, Ghosh, K., Mohapatra, 2021]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. char(**K**) is less than $d^{o(1)}$
2. **K** is of size at most exp(exp(exp(…exp(d)))) (tower of fixed height)
3. ~~number of variables (m) is less than $d^{o(1)}$~~

[Bhargava, Ghosh, Guo, K., Umans, 2022]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. **K** is any finite field

Our results

[Bhargava, Ghosh, K., Mohapatra, 2021]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. char(**K**) is less than $d^{o(1)}$
2. **K** is of size at most exp(exp(exp(…exp(d)))) (tower of fixed height)
3. ~~number of variables (m) is less than $d^{o(1)}$~~

[Bhargava, Ghosh, Guo, K., Umans, 2022]

A nearly linear time algorithm for multivariate multipoint evaluation when

1. **K** is any finite field
2. ~~number of variables (m) is less than $d^{o(1)}$~~

(degree d is asymptotically growing)

Our results

[Ghosh, Harsha, Herdade, K, Saptharishi, 2023]

A nearly linear time algorithm for approximate multivariate multipoint evaluation.

- Running time is $((Nm + d^m)t)^{1+o(1)}$

(degree d is asymptotically growing)

Multivariate multipoint evaluation

In particular

No nearly linear time algorithm for multivariate multipoint evaluation when

- number of variables (m) is not less than $d^{o(1)}$, over any (sufficiently large) field

Multivariate multipoint evaluation

In particular

No nearly linear time algorithm for multivariate multipoint evaluation when

- number of variables (m) is not less than $d^{o(1)}$, over any (sufficiently large) field

Our results

Multivariate multipoint evaluation

In particular

No nearly linear time algorithm for multivariate multipoint evaluation when

- number of variables (m) is not less than $d^{o(1)}$, over any (sufficiently large) field

Our results

Nearly linear time algorithm for multivariate multipoint evaluation over all finite fields, for growing d, and all m

Multivariate multipoint evaluation

In particular

No nearly linear time algorithm for multivariate multipoint evaluation when

- number of variables (m) is not less than $d^{o(1)}$, over any (sufficiently large) field

Our results

Nearly linear time algorithm for multivariate multipoint evaluation over all finite fields, for growing d, and all m

Nearly linear time algorithm for approximate multivariate multipoint evaluation over rationals, reals, complex numbers, for growing d, and all m

In summary

	Field Size	Characteristic	Number of variables	Algebraic vs non-algebraic
Umans	Finite	char(K) < $d^{o(1)}$	m < $d^{o(1)}$	Algebraic
Kedlaya-Umans	Finite	All finite fields	m < $d^{o(1)}$	Non-algebraic
Bhargava-Ghosh-K-Mohapatra	Not-too-large	char(K) < $d^{o(1)}$	No constraint	Algebraic
Bhargava-Ghosh-Guo-K-Umans	Finite	All finite fields	No constraint	Non-algebraic
Ghosh-Harsha-Herdade-K-Saptharishi	Infinite	Rationals, Reals, Complex numbers	No constraint	Non-algebraic (approximate MME)

Applications ?

Applications ?

- Faster algorithms for problems like modular composition in newer regime of parameters via known blackbox connections

Applications ?

- Faster algorithms for problems like modular composition in newer regime of parameters via known blackbox connections

- Two non-blackbox applications from the algorithm over finite fields of small characteristic – algebraic data structures for polynomial evaluation, upper bounds on the rigidity of Vandermonde matrices

Data structures for polynomial evaluation

Data structures for polynomial evaluation

- **K** – finite field of size q

Data structures for polynomial evaluation

- **K** – finite field of size q
- Data: (univariate) polynomial f in K[x] of degree < d

Data structures for polynomial evaluation

- **K** – finite field of size q
- Data: (univariate) polynomial f in K[x] of degree < d
- Goal: to process and store this data in memory, so as to support fast polynomial evaluation queries

Data structures for polynomial evaluation

- **K** – finite field of size q
- Data: (univariate) polynomial f in K[x] of degree < d
- Goal: to process and store this data in memory, so as to support fast polynomial evaluation queries
- Resources
  - Space complexity: amount of memory needed for storage in the worst case

Data structures for polynomial evaluation

- **K** – finite field of size q
- Data: (univariate) polynomial f in K[x] of degree < d
- Goal: to process and store this data in memory, so as to support fast polynomial evaluation queries
- Resources
  - Space complexity: amount of memory needed for storage in the worst case
  - Query complexity: number of cells/bits in the memory needed to access queries in the worst case

Two simple constructions

The *first* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1} x^{d-1}$

Two simple constructions

The *first* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store all the coefficients of f

Two simple constructions

The *first* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store all the coefficients of f
- Query: on queried for any input a in **K,** read all the coefficients and do the evaluation

Two simple constructions

The *first* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store all the coefficients of f
- Query: on queried for any input a in **K,** read all the coefficients and do the evaluation

- Space complexity: $(d \cdot \log q)$ bits

Two simple constructions

The *first* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store all the coefficients of f
- Query: on queried for any input a in **K,** read all the coefficients and do the evaluation

- Space complexity: $(d \cdot \log q)$ bits
- Query complexity: $(d \cdot \log q)$ bits

Two simple constructions

The *first* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store all the coefficients of f
- Query: on queried for any input a in **K,** read all the coefficients and do the evaluation

- Space complexity: $(d \cdot \log q)$ bits - optimal
- Query complexity: $(d \cdot \log q)$ bits - not great

Two simple constructions

The *second* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$

Two simple constructions

The *second* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store the value of f on all inputs in **K**

Two simple constructions

The *second* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store the value of f on all inputs in **K**
- Query: on queried for any input a in **K,** read the appropriate memory locations

Two simple constructions

The *second* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store the value of f on all inputs in **K**
- Query: on queried for any input a in **K,** read the appropriate memory locations

- Space complexity: $(q \cdot \log q)$ bits

Two simple constructions

The *second* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store the value of f on all inputs in **K**
- Query: on queried for any input a in **K,** read the appropriate memory locations

- Space complexity: $(q \cdot \log q)$ bits
- Query complexity: $(\log q)$ bits

Two simple constructions

The *second* construction

- Data: $f(x) = f_0 + f_1 \cdot x + \cdots + f_{d-1}x^{d-1}$
- Memory: store the value of f on all inputs in **K**
- Query: on queried for any input a in **K,** read the appropriate memory locations

- Space complexity: $(q \cdot \log q)$ bits - not great
- Query complexity: $(\log q)$ bits - optimal

A natural question

Is there a construction that achieves the best of both these worlds ?

Space complexity: $(\mathrm{d} \cdot \log \mathrm{q})^{1+o(1)}$

Query complexity: $(\log \mathrm{q})^{1+o(1)}$

A natural question

Is there a construction that achieves the best of both these worlds ?

Space complexity: $(d \cdot \log q)^{1+o(1)}$

Query complexity: $(\log q)^{1+o(1)}$

[Kedlaya-Umans, 2008] Sort of!

Space complexity: $(d \cdot \log q)^{1+o(1)}$

Query complexity: $\mathrm{poly}(\log d) \cdot (\log q)^{1+o(1)}$

A natural question

Is there a construction that achieves the best of both these worlds ?

Space complexity: $(\mathrm{d} \cdot \log \mathrm{q})^{1+o(1)}$

Query complexity: $(\log \mathrm{q})^{1+o(1)}$

[Kedlaya-Umans, 2008] Sort of!

Space complexity: $(\mathrm{d} \cdot \log \mathrm{q})^{1+o(1)}$

Query complexity: $\mathrm{poly}(\log \mathrm{d}) \cdot (\log \mathrm{q})^{1+o(1)}$

So, this *almost* answers this question!

Algebraic vs non-algebraic

Algebraic vs non-algebraic

Algebraic algorithms: basic operations are arithmetic operations (+, *) over the underlying field

Algebraic vs non-algebraic

Algebraic algorithms: basic operations are arithmetic operations (+, *) over the underlying field

Algebraic data structures: all the associated underlying algorithms are algebraic

Algebraic vs non-algebraic

Algebraic algorithms: basic operations are arithmetic operations (+, *) over the underlying field

Algebraic data structures: all the associated underlying algorithms are algebraic

- naïve algorithm for multipoint evaluation, Fast Fourier Transform, both the simple data structures for polynomial evaluation are algebraic

Algebraic vs non-algebraic

Algebraic algorithms: basic operations are arithmetic operations (+, *) over the underlying field

Algebraic data structures: all the associated underlying algorithms are algebraic

- naïve algorithm for multipoint evaluation, Fast Fourier Transform, both the simple data structures for polynomial evaluation are algebraic
- algorithm of Kedlaya-Umans for multipoint evaluation is non-algebraic - uses things like bit operations, lifts the problem from the underlying field to over integers

Algebraic vs non-algebraic

Algebraic algorithms: basic operations are arithmetic operations (+, *) over the underlying field

Algebraic data structures: all the associated underlying algorithms are algebraic

- naïve algorithm for multipoint evaluation, Fast Fourier Transform, both the simple data structures for polynomial evaluation are algebraic
- algorithm of Kedlaya-Umans for multipoint evaluation is non-algebraic - uses things like bit operations, lifts the problem from the underlying field to over integers
- algebraic algorithms might be more aesthetic, could be useful when working with arithmetic models of computation

Algebraic data structures for polynomial evaluation

Algebraic data structures for polynomial evaluation

Ideally, it would be nice to have algebraic data structures for polynomial evaluation, with Kedlaya-Umans like performance guarantees

Algebraic data structures for polynomial evaluation

Ideally, it would be nice to have algebraic data structures for polynomial evaluation, with Kedlaya-Umans like performance guarantees

Is that even possible ?

Algebraic data structures for polynomial evaluation

Ideally, it would be nice to have algebraic data structures for polynomial evaluation, with Kedlaya-Umans like performance guarantees

Is that even possible ?

[Miltersen, 1995]

If $q > \exp(d)$, then essentially no algebraic data structure for polynomial evaluation better than the trivial solution of storing all coefficients.

Algebraic data structures for polynomial evaluation

Ideally, it would be nice to have algebraic data structures for polynomial evaluation, with Kedlaya-Umans like performance guarantees

Is that even possible ?

[Miltersen, 1995]

If $q > \exp(d)$, then essentially no algebraic data structure for polynomial evaluation better than the trivial solution of storing all coefficients.

Conjectured that the same should hold even for smaller fields.

A corollary

[Bhargava, Ghosh, K., Mohapatra, 2021]

A data structure for polynomial evaluation with nearly linear space and sublinear query complexity, provided

1. char of the field is small
2. $q <$ quasipoly(d)

(Input size: $d \cdot \log q$ bits)

Matrix Rigidity

Matrix Rigidity

(r,s)-rigid matrices

$V \neq L + S$, where $\text{rank}(L) < r$, $\text{sparsity}(S) < s$

Matrix Rigidity

(r,s)-rigid matrices

$V \neq L + S$, where rank(L) < r, sparsity(S) < s

[Valiant, 1977]

Explicit construction of sufficiently rigid matrices implies new lower bounds in algebraic complexity

Matrix Rigidity

(r,s)-rigid matrices

$V \neq L + S$, where $\mathrm{rank}(L) < r$, $\mathrm{sparsity}(S) < s$

[Valiant, 1977]

Explicit construction of sufficiently rigid matrices implies new lower bounds in algebraic complexity

Many popular candidates – Hadamard, Discrete Fourier Transform, Vandermonde matrices

Matrix Rigidity

(r,s)-rigid matrices

$V \neq L + S$, where $\text{rank}(L) < r$, $\text{sparsity}(S) < s$

[Valiant, 1977]

Explicit construction of sufficiently rigid matrices implies new lower bounds in algebraic complexity

Many popular candidates – Hadamard, Discrete Fourier Transform, Vandermonde matrices

No sufficiently strong lower bounds

Non-rigidity of popular candidates

Non-rigidity of popular candidates

[Alman, Williams, 2016]

Hadamard matrices are not sufficiently rigid

Non-rigidity of popular candidates

[Alman, Williams, 2016]

Hadamard matrices are not sufficiently rigid

[Dvir, Liu, 2019]

Discrete Fourier Transform matrices are not sufficiently rigid

Non-rigidity of popular candidates

[Alman, Williams, 2016]

Hadamard matrices are not sufficiently rigid

[Dvir, Liu, 2019]

Discrete Fourier Transform matrices are not sufficiently rigid

And, some more – Alman, Dvir-Edelman, Kivva….

Non-rigidity of popular candidates

[Alman, Williams, 2016]

Hadamard matrices are not sufficiently rigid

[Dvir, Liu, 2019]

Discrete Fourier Transform matrices are not sufficiently rigid

And, some more – Alman, Dvir-Edelman, Kivva….

Rigidity upper bounds for general Vandermonde matrices remains open

A corollary

[Bhargava, Ghosh, K., Mohapatra, 2021]

Vandermonde matrices are not sufficiently rigid, when

1. char of the field is small
2. q < quasipoly(dimension)

An outline of the algorithm

An outline of the algorithm

Theorem

A nearly linear time algorithm for multivariate multipoint evaluation when

1. char(**K**) is less than $d^{o(1)}$
2. **K** is of size at most exp(exp(exp(…exp(d)))) (tower of fixed height)

An outline of the algorithm

Input

- An m-variate polynomial f with degree at most (d-1) in each variable over a field ***K***, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}^{\mathbf{m}}$

An outline of the algorithm

Input

- An m-variate polynomial f with degree at most (d-1) in each variable over a field ***K***, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}^{\mathbf{m}}$

Two phases of the algorithm

An outline of the algorithm

Input

- An m-variate polynomial f with degree at most (d-1) in each variable over a field ***K***, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}^{\mathbf{m}}$

Two phases of the algorithm

- Preprocessing phase: independent of the evaluation points $\alpha_1, \alpha_2, \ldots, \alpha_N$

An outline of the algorithm

Input

- An m-variate polynomial f with degree at most (d-1) in each variable over a field ***K***, as a list of coefficients
- N points $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbf{K}^m$

Two phases of the algorithm

- Preprocessing phase: independent of the evaluation points $\alpha_1, \alpha_2, \ldots, \alpha_N$
- Local computation phase: depend on $\alpha_1, \alpha_2, \ldots, \alpha_N$, and earlier computation

An outline of the algorithm

Preprocessing phase

An outline of the algorithm

Preprocessing phase

1. Construct a set $S \subseteq \mathbf{K}^m$ such that

An outline of the algorithm

Preprocessing phase

1. Construct a set $S \subseteq \mathbf{K}^m$ such that
   - $|S|$ is not too large (comparable to the input size)

An outline of the algorithm

Preprocessing phase

1. Construct a set $S \subseteq \mathbf{K}^m$ such that
   - $|S|$ is not too large (comparable to the input size)
   - S is a product set

An outline of the algorithm

Preprocessing phase

1. Construct a set $S \subseteq \mathbf{K}^m$ such that
   - $|S|$ is not too large (comparable to the input size)
   - S is a product set
   - For every $\alpha \in \mathbf{K}^m$, there is a low degree curve C_α through α which has large intersection with S

An outline of the algorithm

Preprocessing phase

1. Construct a set $S \subseteq \mathbf{K}^m$ such that
   - $|S|$ is not too large (comparable to the input size)
   - S is a product set
   - For every $\alpha \in \mathbf{K}^m$, there is a low degree curve C_α through α which has large intersection with S

2. Evaluate f on all points of S

An outline of the algorithm

Local computation

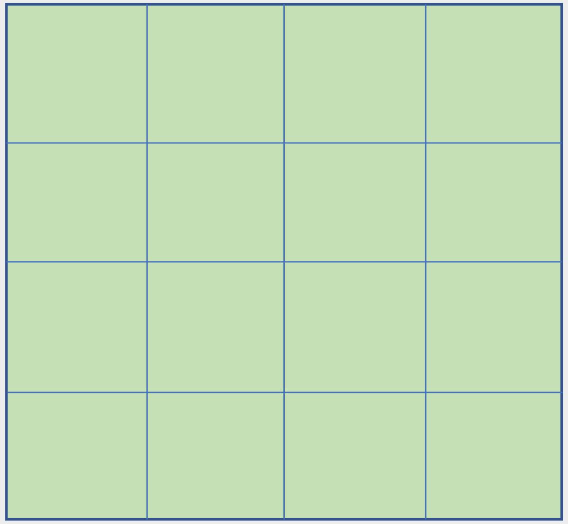
An outline of the algorithm

Local computation

- have an $\alpha \in \mathbf{K}^m$, want to compute $f(\alpha)$ fast, using info from the previous step

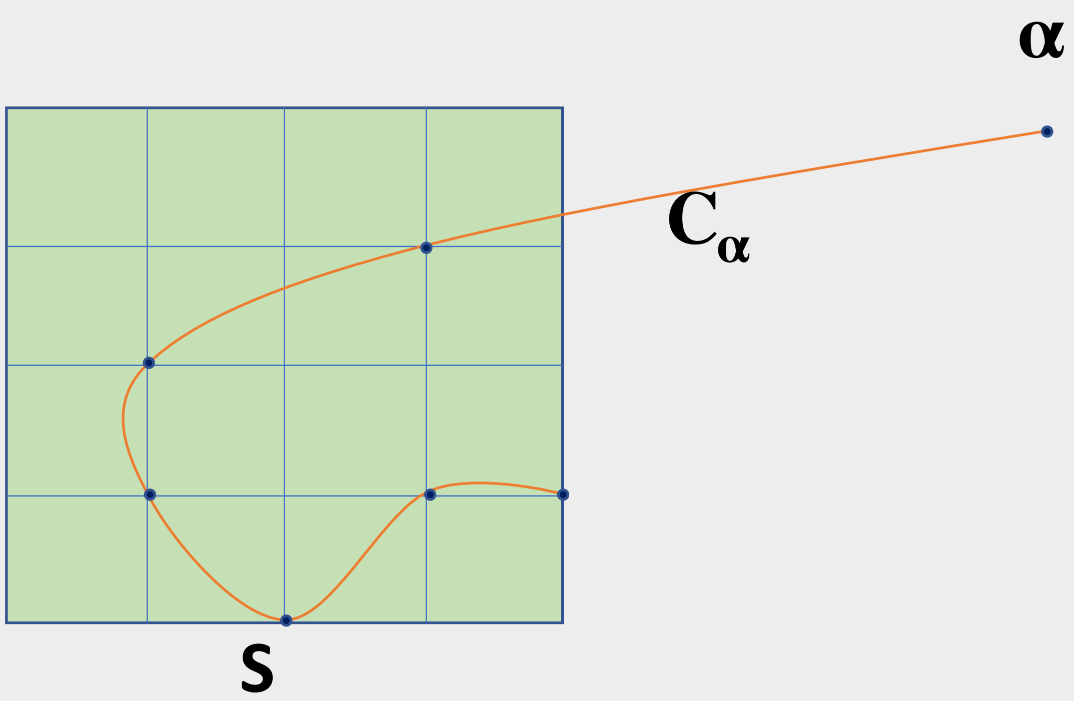
An outline of the algorithm

α



S

An outline of the algorithm



An outline of the algorithm

Local computation

- have an $\alpha \in \mathbf{K}^m$, want to compute $f(\alpha)$ fast, using info from the previous step
- let $C_\alpha(y) = (r_{\alpha,1}(y), r_{\alpha,2}(y), \ \ldots, r_{\alpha,m}(y))$ be the low degree curve through α, with large intersection with S; each $r_{\alpha,i}(y)$ is a low degree polynomial

An outline of the algorithm

Local computation

- have an $\alpha \in \mathbf{K}^m$, want to compute $f(\alpha)$ fast, using info from the previous step
- let $C_\alpha(y) = \left(r_{\alpha,1}(y), r_{\alpha,2}(y), \ \ldots, r_{\alpha,m}(y)\right)$ be the low degree curve through α, with large intersection with S; each $r_{\alpha,i}(y)$ is a low degree polynomial
- $C_\alpha(y)$ passes through α, i.e. there exists $u \in \mathbf{K}$, such that $C_\alpha(u) = \alpha$

An outline of the algorithm

Local computation

- have an $\alpha \in \mathbf{K}^m$, want to compute $f(\alpha)$ fast, using info from the previous step
- let $C_\alpha(y) = (r_{\alpha,1}(y), r_{\alpha,2}(y), \ \ldots, r_{\alpha,m}(y))$ be the low degree curve through α, with large intersection with S; each $r_{\alpha,i}(y)$ is a low degree polynomial
- $C_\alpha(y)$ passes through α, i.e. there exists $u \in \mathbf{K}$, such that $C_\alpha(u) = \alpha$
- let $g(t) = f(r_{\alpha,1}(y), r_{\alpha,2}(y), \ \ldots, r_{\alpha,m}(y))$ be the restriction of the polynomial f on the curve $C_\alpha(y)$

An outline of the algorithm

Local computation

- have an $\alpha \in \mathbf{K}^m$, want to compute $f(\alpha)$ fast, using info from the previous step
- let $C_\alpha(y) = (r_{\alpha,1}(y), r_{\alpha,2}(y), \ldots, r_{\alpha,m}(y))$ be the low degree curve through α, with large intersection with S; each $r_{\alpha,i}(y)$ is a low degree polynomial
- $C_\alpha(y)$ passes through α, i.e. there exists $u \in \mathbf{K}$, such that $C_\alpha(u) = \alpha$
- let $g(t) = f(r_{\alpha,1}(y), r_{\alpha,2}(y), \ldots, r_{\alpha,m}(y))$ be the restriction of the polynomial f on the curve $C_\alpha(y)$
- g is univariate of degree at most $(\deg(C_\alpha) \cdot dm)$

An outline of the algorithm

Local computation

- have an $\alpha \in \mathbf{K}^m$, want to compute $f(\alpha)$ fast, using info from the previous step
- let $C_\alpha(y) = \left(r_{\alpha,1}(y), r_{\alpha,2}(y), \ldots, r_{\alpha,m}(y)\right)$ be the low degree curve through α, with large intersection with S; each $r_{\alpha,i}(y)$ is a low degree polynomial
- $C_\alpha(y)$ passes through α, i.e. there exists $u \in \mathbf{K}$, such that $C_\alpha(u) = \alpha$
- let $g(t) = f(r_{\alpha,1}(y), r_{\alpha,2}(y), \ldots, r_{\alpha,m}(y))$ be the restriction of the polynomial f on the curve $C_\alpha(y)$
- g is univariate of degree at most $(\deg(C_\alpha) \cdot dm)$
- if we can efficiently get our hands on g, we can set t = u, to get $f(\alpha)$

An outline of the algorithm

Local computation

An outline of the algorithm

Local computation

- for each of the N input points, we only have sublinear [$(N + d^m)^{o(1)}$] time

An outline of the algorithm

Local computation

- for each of the N input points, we only have sublinear [$\left(N + d^m\right)^{o(1)}$] time
- from properties of S, we have that C_α intersects S at many points, and we have value of f at all points in S

An outline of the algorithm

Local computation

- for each of the N input points, we only have sublinear [$\left(N + d^m\right)^{o(1)}$] time
- from properties of S, we have that C_α intersects S at many points, and we have value of f at all points in S
- let $v \in \mathbf{K}$ be such that $C_\alpha(v) = \left(r_{\alpha,1}(v), r_{\alpha,2}(v), \ \ldots, r_{\alpha,m}(v)\right)$ be in S

An outline of the algorithm

Local computation

- for each of the N input points, we only have sublinear [$\left(\mathrm{N}+\mathrm{d}^{\mathrm{m}}\right)^{o(1)}$] time
- from properties of S, we have that C_{α} intersects S at many points, and we have value of f at all points in S
- let $\mathrm{v} \in \mathbf{K}$ be such that $\mathrm{C}_{\alpha}(\mathrm{v})=\left(\mathrm{r}_{\alpha,1}(\mathrm{v}), \mathrm{r}_{\alpha,2}(\mathrm{v}), \ldots, \mathrm{r}_{\alpha,\mathrm{m}}(\mathrm{v})\right)$ be in S
- from the preprocessing phase, we have already computed $\mathrm{g}(\mathrm{v})=\mathrm{f}(\mathrm{r}_{\alpha,1}(\mathrm{v}), \mathrm{r}_{\alpha,2}(\mathrm{v}), \ldots, \mathrm{r}_{\alpha,\mathrm{m}}(\mathrm{v}))$

An outline of the algorithm

Local computation

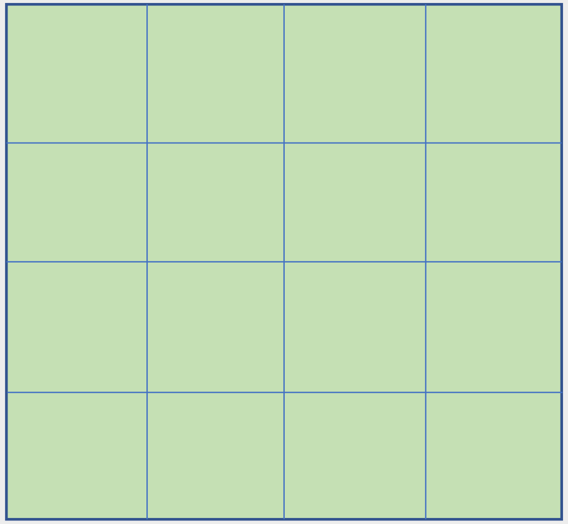
- for each of the N input points, we only have sublinear [$\left(N + d^m\right)^{o(1)}$] time
- from properties of S, we have that C_α intersects S at many points, and we have value of f at all points in S
- let $v \in \mathbf{K}$ be such that $C_\alpha(v) = \left(r_{\alpha,1}(v), r_{\alpha,2}(v), \ldots, r_{\alpha,m}(v)\right)$ be in S
- from the preprocessing phase, we have already computed $g(v) = f(r_{\alpha,1}(v), r_{\alpha,2}(v), \ldots, r_{\alpha,m}(v))$
- so, if $\left|C_\alpha \cap S\right| > \deg\left(g\right)$, can recover the polynomial g via interpolation

An outline of the algorithm

Local computation

- for each of the N input points, we only have sublinear [$(N + d^m)^{o(1)}$] time
- from properties of S, we have that C_α intersects S at many points, and we have value of f at all points in S
- let $v \in \mathbf{K}$ be such that $C_\alpha(v) = (r_{\alpha,1}(v), r_{\alpha,2}(v), \ \ldots, r_{\alpha,m}(v))$ be in S
- from the preprocessing phase, we have already computed $g(v) = f(r_{\alpha,1}(v), r_{\alpha,2}(v), \ \ldots, r_{\alpha,m}(v))$
- so, if $|C_\alpha \cap S| > \deg(g)$, can recover the polynomial g via interpolation
- once, we have g, can recover $g(u) = \ f(\alpha)$

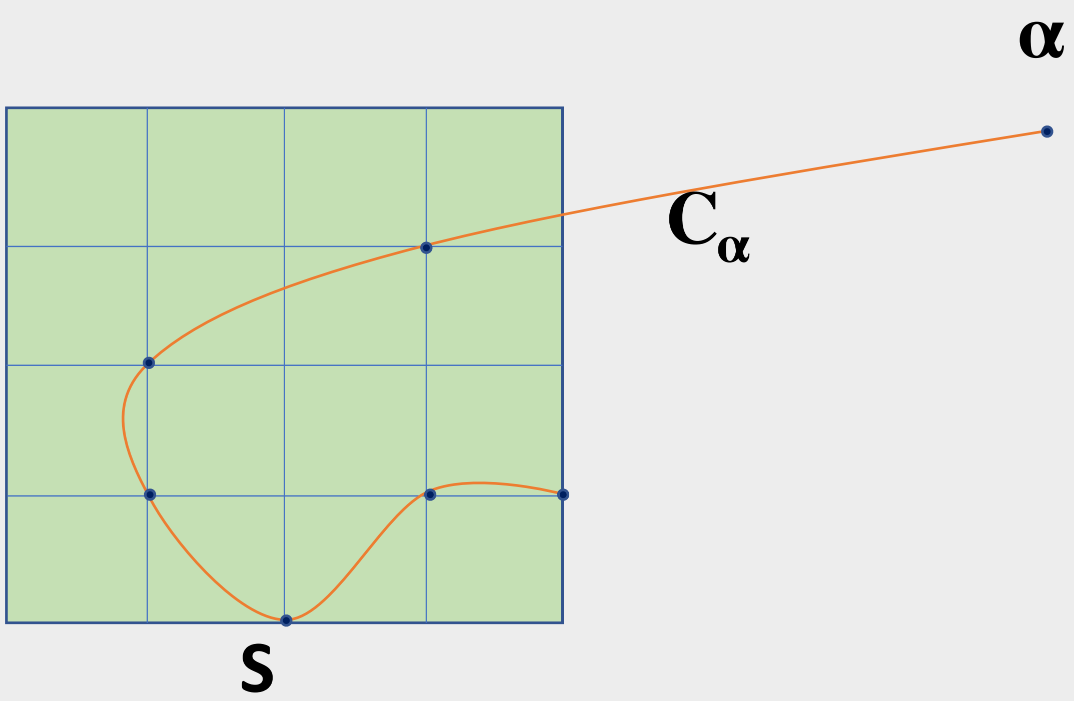
An outline of the algorithm



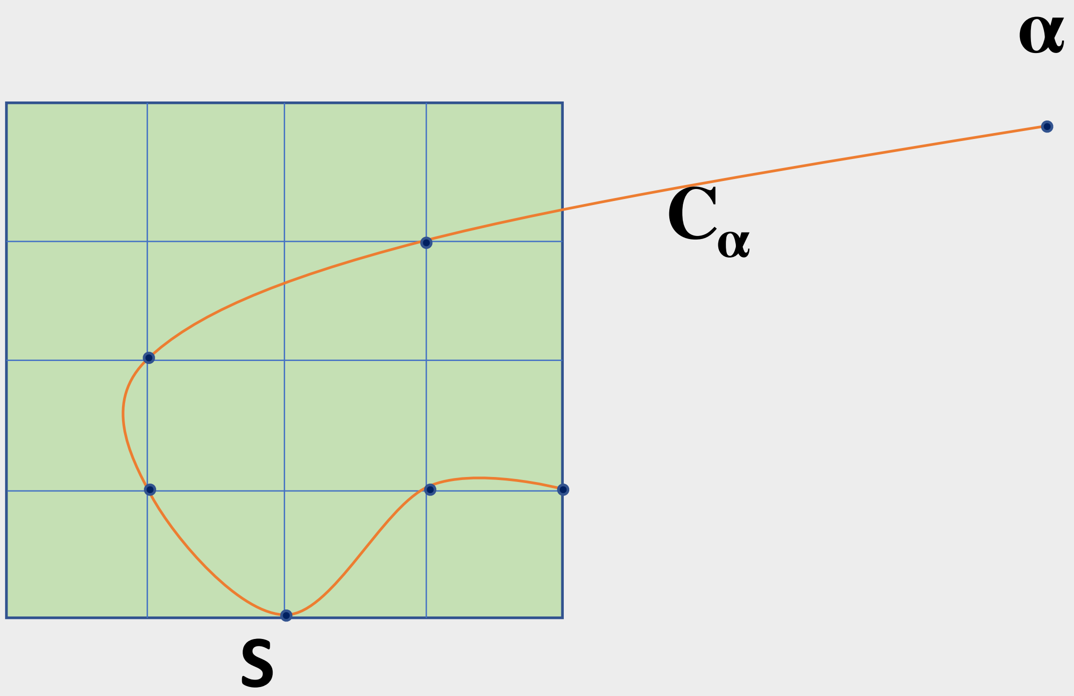
S

α

An outline of the algorithm



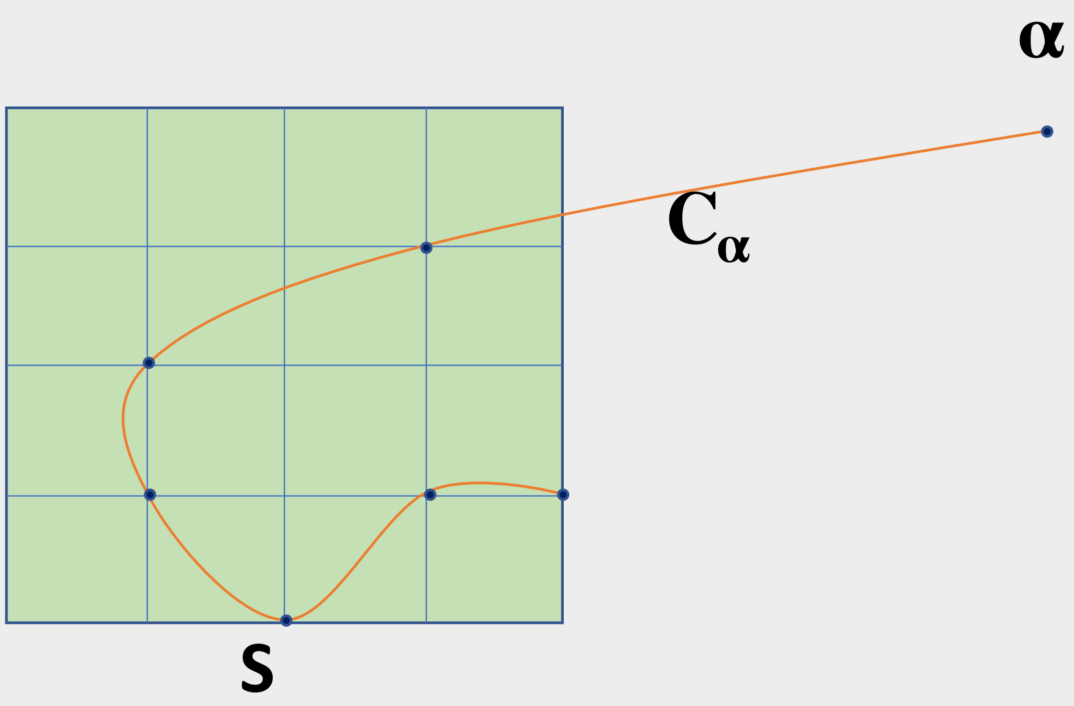
An outline of the algorithm



Want

$$|C_\alpha \cap S| > \deg(C_\alpha) \cdot dm$$

An outline of the algorithm



Want

$$|C_\alpha \cap S| > \deg(C_\alpha)\cdot dm$$

- $|S| < \left(pdm \cdot \log_p |\mathbf{K}|\right)^m$
- $\deg(C_\alpha) < \log_p |\mathbf{K}|$
- $|C_\alpha \cap S| > \log_p |\mathbf{K}| \cdot dm > \deg(C_\alpha)\cdot dm$

The mysterious set S

The mysterious set S

- b in **N** be such that $p^{b-1} \leq dm \cdot \log_p |\mathbf{K}| \leq p^b$

The mysterious set S

- b in **N** be such that $p^{b-1} \leq dm \cdot \log_p |\mathbf{K}| \leq p^b$
- $S = \mathbf{F}_{p^b}^m$

The mysterious set S

- b in **N** be such that $p^{b-1} \leq dm \cdot \log_p |\mathbf{K}| \leq p^b$
- $S = \mathbf{F}_{p^b}^m$
- clearly, $|S| < \left(pdm \cdot \log_{\boldsymbol{p}} |\boldsymbol{K}|\right)^{\mathbf{m}}$

The mysterious set S

- b in **N** be such that $p^{b-1} \leq dm \cdot \log_p |\mathbf{K}| \leq p^b$
- $S = \mathbf{F}_{p^b}^m$
- clearly, $|S| < \left(pdm \cdot \log_{p} |\boldsymbol{K}|\right)^{\mathbf{m}}$
- S can be constructed ‘efficiently’ – linear time in its size

The mysterious set S

- b in **N** be such that $p^{b-1} \leq dm \cdot \log_p |\mathbf{K}| \leq p^b$
- $S = \mathbf{F}_{p^b}^m$
- clearly, $|S| < \left(pdm \cdot \log_{p} |\boldsymbol{K}|\right)^{\mathbf{m}}$
- S can be constructed ‘efficiently’ – linear time in its size
- product set, so, f can be evaluated on S in nearly linear time

The mysterious set S

- b in **N** be such that $p^{b-1} \leq dm \cdot \log_p |\mathbf{K}| \leq p^b$
- $S = \mathbf{F}_{p^b}^m$
- clearly, $|S| < \left(pdm \cdot \log_p |\boldsymbol{K}|\right)^{\mathbf{m}}$
- S can be constructed 'efficiently' – linear time in its size
- product set, so, f can be evaluated on S in nearly linear time
- running time of the first phase - nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_p |\boldsymbol{K}|\right)^{\mathbf{m}}$

The mysterious set S

- b in **N** be such that $p^{b-1} \le dm \cdot \log_p |\mathbf{K}| \le p^b$
- $S = \mathbf{F}_{p^b}^m$
- clearly, $|S| < \left(pdm \cdot \log_p |\boldsymbol{K}|\right)^{\mathbf{m}}$
- S can be constructed 'efficiently' – linear time in its size
- product set, so, f can be evaluated on S in nearly linear time
- running time of the first phase - nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_p |\boldsymbol{K}|\right)^{\mathbf{m}}$
- has the mysterious curve property needed for subsequent step

The curve property

- let b = 1, m = 2.

The curve property

- let b = 1, m = 2. So, $\mathrm{S} = \mathbf{F}_{\mathrm{p}}^{2}$

The curve property

- let b = 1, m = 2. So, $S = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$

The curve property

- let b = 1, m = 2. So, $\mathrm{S} = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$
- $\alpha = (\alpha_0,\ \alpha_1) \in \mathbf{K}^2$

The curve property

- let b = 1, m = 2. So, $S = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$
- $\alpha = (\alpha_0,\ \alpha_1) \in \mathbf{K}^2$
- $(\alpha_0,\ \alpha_1) = \ (a_{0,0} + a_{0,1}z,\ a_{1,0} + a_{1,1}z) = (a_{0,0},\ a_{1,0}) + z(a_{0,1}, a_{1,1})$
  $(a_{i,j} \ \in \mathbf{F}_p)$

The curve property

- let b = 1, m = 2. So, $S = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$
- $\alpha = (\alpha_0,\ \alpha_1) \in \mathbf{K}^2$
- $(\alpha_0,\ \alpha_1) = \ (a_{0,0} + a_{0,1}z,\ a_{1,0} + a_{1,1}z) = (a_{0,0},\ a_{1,0}) + z(a_{0,1}, a_{1,1})$
  $(a_{i,j} \ \in \mathbf{F}_p)$

- $C_\alpha(t) := A_0 + t \cdot A_1$, where $A_i = (a_{0,i},\ a_{1,i}) \in \mathbf{F}_p^2$

The curve property

- let b = 1, m = 2. So, $S = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$
- $\alpha = (\alpha_0,\ \alpha_1) \in \mathbf{K}^2$
- $(\alpha_0,\ \alpha_1) = (a_{0,0} + a_{0,1}z,\ a_{1,0} + a_{1,1}z) = (a_{0,0},\ a_{1,0}) + z(a_{0,1}, a_{1,1})$ $(a_{i,j} \in \mathbf{F}_p)$

- $C_\alpha(t) := A_0 + t \cdot A_1$, where $A_i = (a_{0,i},\ a_{1,i}) \in \mathbf{F}_p^2$
- $C_\alpha(z) = \alpha$

The curve property

- let b = 1, m = 2. So, $S = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$
- $\alpha = (\alpha_0,\ \alpha_1) \in \mathbf{K}^2$
- $(\alpha_0,\ \alpha_1) = \ (a_{0,0} + a_{0,1}z,\ a_{1,0} + a_{1,1}z) = (a_{0,0},\ a_{1,0}) + z(a_{0,1}, a_{1,1})$ $(a_{i,j}\ \in \mathbf{F}_p)$

- $C_\alpha(t) := A_0 + t \cdot A_1$, where $A_i = (a_{0,i},\ a_{1,i}) \in \mathbf{F}_p^2$
- $C_\alpha(z) = \alpha$ (the curve passes through α)

The curve property

- let b = 1, m = 2. So, $S = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$
- $\alpha = (\alpha_0,\ \alpha_1) \in \mathbf{K}^2$
- $(\alpha_0,\ \alpha_1) = (a_{0,0} + a_{0,1}z,\ a_{1,0} + a_{1,1}z) = (a_{0,0},\ a_{1,0}) + z(a_{0,1}, a_{1,1})$ $(a_{i,j} \in \mathbf{F}_p)$

- $C_\alpha(t) := A_0 + t \cdot A_1$, where $A_i = (a_{0,i},\ a_{1,i}) \in \mathbf{F}_p^2$
- $C_\alpha(z) = \alpha$ (the curve passes through α)
- For every $u \in \mathbf{F}_p$, $C_\alpha(u) := A_0 + u \cdot A_1 \in \mathbf{F}_p^2 = S$

The curve property

- let b = 1, m = 2. So, $S = \mathbf{F}_p^2$
- let $\mathbf{K} = \mathbf{F}_{p^2} = \mathbf{F}_p[z]/I(z)$, where I is an irreducible polynomial of degree 2 in $F_p[z]$
- $\alpha = (\alpha_0,\ \alpha_1) \in \mathbf{K}^2$
- $(\alpha_0,\ \alpha_1) = \ (a_{0,0} + a_{0,1}z,\ a_{1,0} + a_{1,1}z) = (a_{0,0},\ a_{1,0}) + z(a_{0,1}, a_{1,1})$ $\quad (a_{i,j} \in \mathbf{F}_p)$

- $C_\alpha(t) := A_0 + t \cdot A_1$, where $A_i = (a_{0,i},\ a_{1,i}) \in \mathbf{F}_p^2$
- $C_\alpha(z) = \alpha$ (the curve passes through α)
- For every $u \in \mathbf{F}_p$, $C_\alpha(u) := A_0 + u \cdot A_1 \in \mathbf{F}_p^2 = S$ (large intersection with S)

Running time

Running time

- running time of the first phase – nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_{\mathbf{p}}|\mathbf{K}|\right)^{\mathbf{m}}$

Running time

- running time of the first phase – nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_{\mathbf{p}} \left|\mathbf{K}\right|\right)^{\mathbf{m}}$
- N iterations of univariate polynomial interpolation for degree $\log_{\mathbf{p}} \left|\mathbf{K}\right| \cdot dm$ + finding the curves at each input

Running time

- running time of the first phase – nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_p |\mathbf{K}|\right)^{\mathbf{m}}$
- N iterations of univariate polynomial interpolation for degree $\log_{\mathbf{p}} |\mathbf{K}| \cdot dm$ + finding the curves at each input
- $N \cdot poly(\log_{\mathbf{p}} |\mathbf{K}| \cdot dm)$ time

Running time

- running time of the first phase – nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_p |\mathbf{K}|\right)^{\mathbf{m}}$
- N iterations of univariate polynomial interpolation for degree $\log_p |\mathbf{K}| \cdot dm$ + finding the curves at each input
- $N \cdot \text{poly}(\log_p |\mathbf{K}| \cdot dm)$ time
- total running time : $(N + \left(pdm \cdot \log_p |\mathbf{K}|\right)^{\mathbf{m}}) \cdot \text{poly}(\log_p |\mathbf{K}| \cdot dm)$

Running time

- running time of the first phase – nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_p |K|\right)^m$
- N iterations of univariate polynomial interpolation for degree $\log_p |K| \cdot dm$ + finding the curves at each input
- $N \cdot \text{poly}(\log_p |K| \cdot dm)$ time
- total running time : $(N + \left(pdm \cdot \log_p |K|\right)^m) \cdot \text{poly}(\log_p |K| \cdot dm)$
- for small p, m, $\log_p |K|$, this is nearly linear time in $(d^m + N)$

Running time

- running time of the first phase – nearly linear in $(d^m + |S|) \sim \left(pdm \cdot \log_p |\mathbf{K}|\right)^{\mathbf{m}}$
- N iterations of univariate polynomial interpolation for degree $\log_p |\mathbf{K}| \cdot dm$ + finding the curves at each input
- $N \cdot poly(\log_p |\mathbf{K}| \cdot dm)$ time
- total running time : $(N + \left(pdm \cdot \log_p |\mathbf{K}|\right)^{\mathbf{m}}) \cdot poly(\log_{\mathbf{p}} |\mathbf{K}| \cdot dm)$
- for small p, m, $\log_p |\mathbf{K}|$, this is nearly linear time in $(d^m + N)$
- Well…what about large m, large fields ?

A few more ideas

- Well…what about large m, large fields ?

A few more ideas

- Well…what about large m, large fields ?
- the bottleneck is the size of S

A few more ideas

- Well…what about large m, large fields ?
- the bottleneck is the size of S
- if we could work with a smaller set S, then…..

A few more ideas

- Well…what about large m, large fields ?
- the bottleneck is the size of S
- if we could work with a smaller set S, then…..
- to continue the local decoding step, will need to ensure that we have sufficient information for univariate interpolation along the curve at each point

A few more ideas

- Well…what about large m, large fields ?
- the bottleneck is the size of S
- if we could work with a smaller set S, then…..
- to continue the local decoding step, will need to ensure that we have sufficient information for univariate interpolation along the curve at each point
- here, we work with a smaller set S

A few more ideas

- Well…what about large m, large fields ?
- the bottleneck is the size of S
- if we could work with a smaller set S, then…..
- to continue the local decoding step, will need to ensure that we have sufficient information for univariate interpolation along the curve at each point
- here, we work with a smaller set S
- leads to reduced intersection between the curves and the set S

A few more ideas

- Well…what about large m, large fields ?
- the bottleneck is the size of S
- if we could work with a smaller set S, then…..
- to continue the local decoding step, will need to ensure that we have sufficient information for univariate interpolation along the curve at each point
- here, we work with a smaller set S
- leads to reduced intersection between the curves and the set S
- to compensate, need stronger preprocessing phase, and a more complicated local computation step

A few more ideas

Dealing with large number of variables

A few more ideas

Dealing with large number of variables

- method of multiplicities

A few more ideas

Dealing with large number of variables

- method of multiplicities
- evaluate f, and all its partial derivatives of order at most m, on all points of S

A few more ideas

Dealing with large number of variables

- method of multiplicities
- evaluate f, and all its partial derivatives of order at most m, on all points of S
- this additional information lets us proceed with a smaller set S

  ($|S| < \left(pd \cdot \log_p |K|\right)^m$)

A few more ideas

Dealing with large number of variables

- method of multiplicities
- evaluate f, and all its partial derivatives of order at most m, on all points of S
- this additional information lets us proceed with a smaller set S

  ($|S| < \left(pd \cdot \log_{\mathbf{p}} |\mathbf{K}|\right)^{\mathbf{m}}$)
- instead of constructing univariate polynomials from just evaluations, we now construct them from their evaluations and the evaluations of their derivatives

A few more ideas

Dealing with large number of variables

- method of multiplicities
- evaluate f, and all its partial derivatives of order at most m, on all points of S
- this additional information lets us proceed with a smaller set S

  $(|S| < (pd \cdot \log_p |K|)^m)$
- instead of constructing univariate polynomials from just evaluations, we now construct them from their evaluations and the evaluations of their derivatives
- running time - $(N + (pd \cdot \log_p |K|)^m) \cdot poly(\log_p |K| \cdot dm)$

A few more ideas

Dealing with large fields

A few more ideas

Dealing with large fields

- the degree of the curve through an input point depends on the degree of the field extension that the point lies in

A few more ideas

Dealing with large fields

- the degree of the curve through an input point depends on the degree of the field extension that the point lies in
- on each C_α, there are many points β that lie in much lower degree extensions

A few more ideas

Dealing with large fields

- the degree of the curve through an input point depends on the degree of the field extension that the point lies in
- on each C_α, there are many points β that lie in much lower degree extensions
- so, the value of f is *easier* to *decode* on such points

A few more ideas

Dealing with large fields

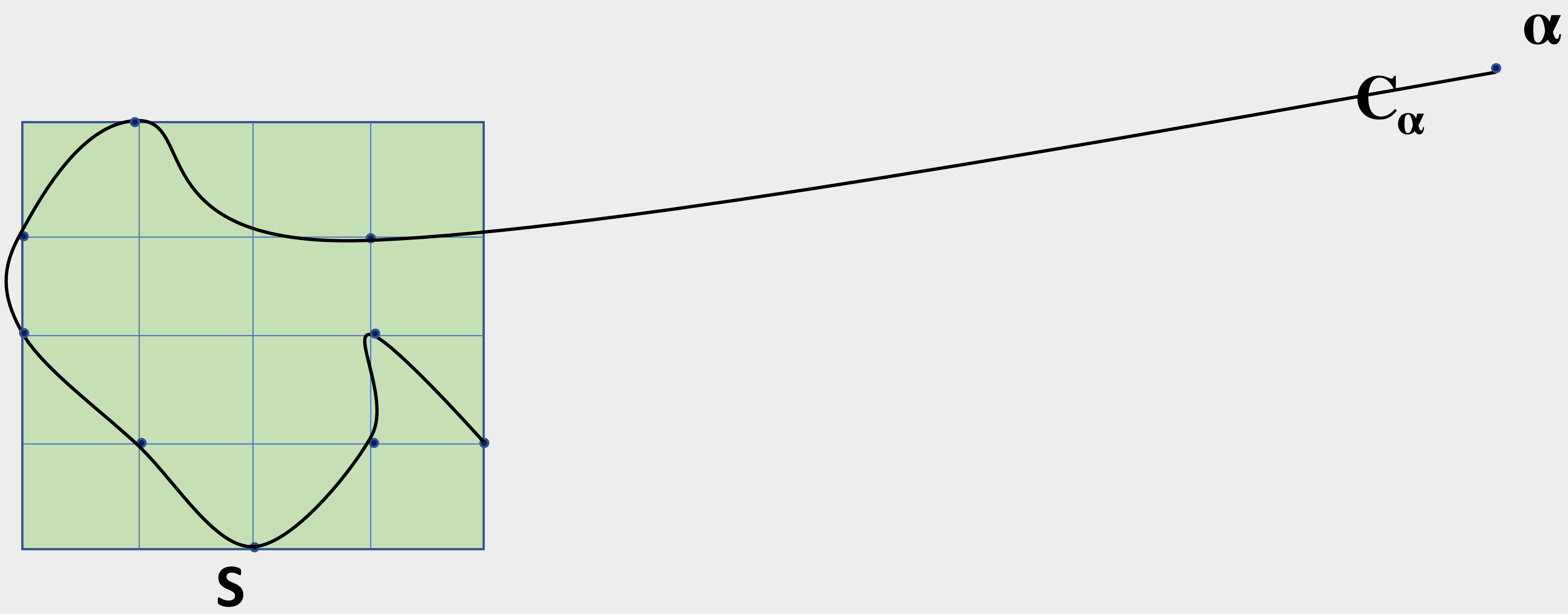
- the degree of the curve through an input point depends on the degree of the field extension that the point lies in
- on each C_α, there are many points β that lie in much lower degree extensions
- so, the value of f is *easier* to *decode* on such points
- instead of computing the restriction of f on C_α, by looking at the values of f on $C_\alpha \cap S$, we first compute f on easier points of C_α

A few more ideas

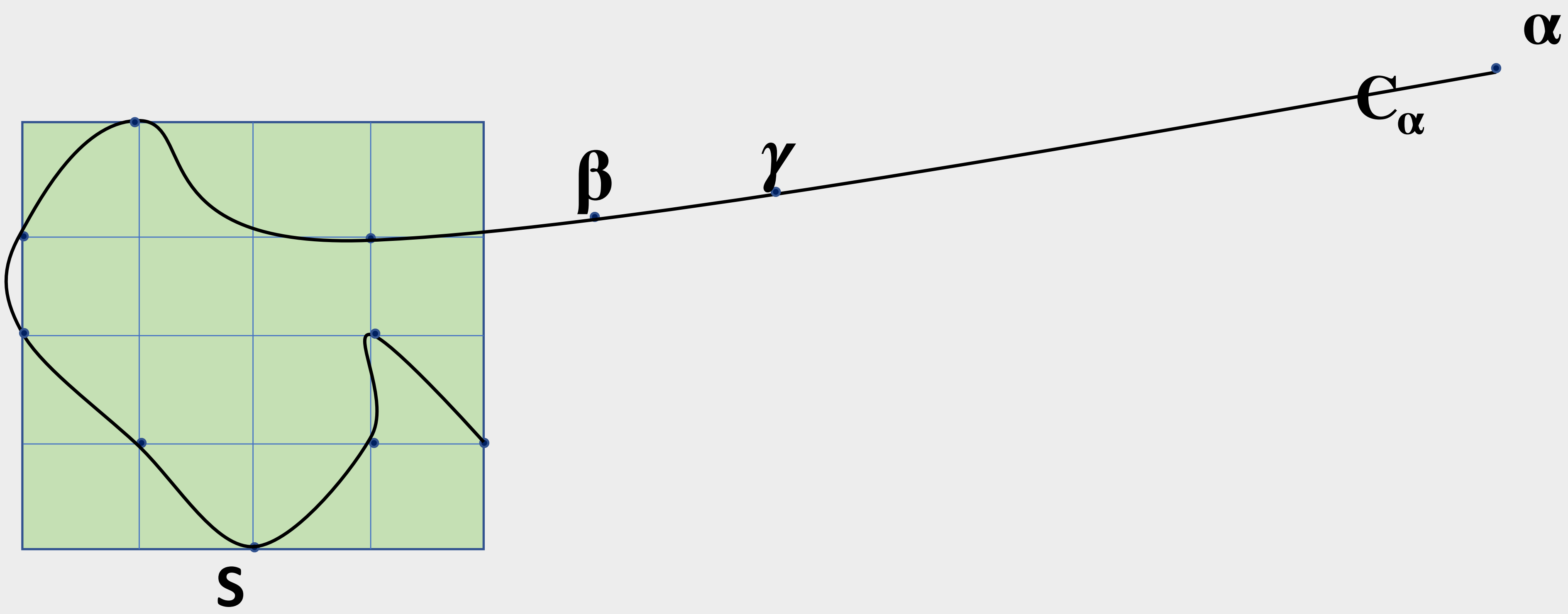
Dealing with large fields

- the degree of the curve through an input point depends on the degree of the field extension that the point lies in
- on each C_α, there are many points β that lie in much lower degree extensions
- so, the value of f is *easier* to *decode* on such points
- instead of computing the restriction of f on C_α, by looking at the values of f on $C_\alpha \cap S$, we first compute f on easier points of C_α
- then, use this additional info, together with values of f on S to do interpolation

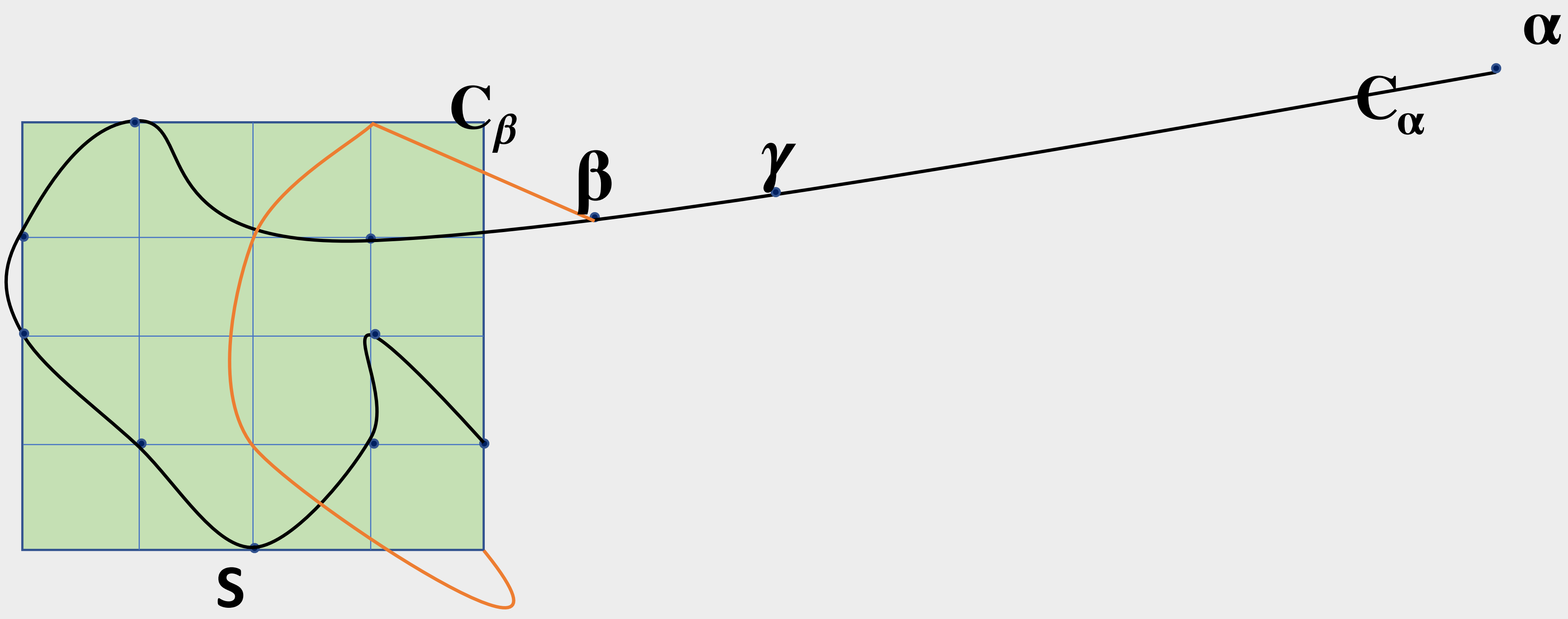
The final inaccurate picture



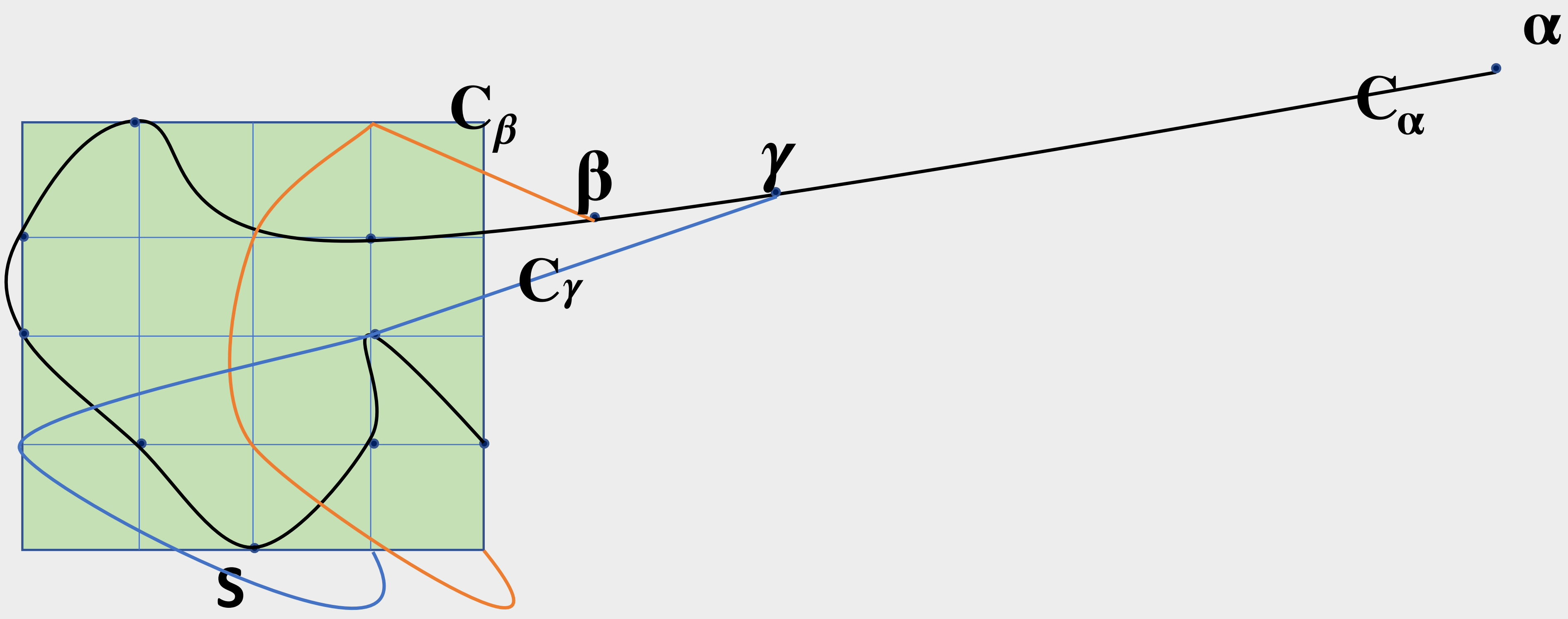
The final inaccurate picture



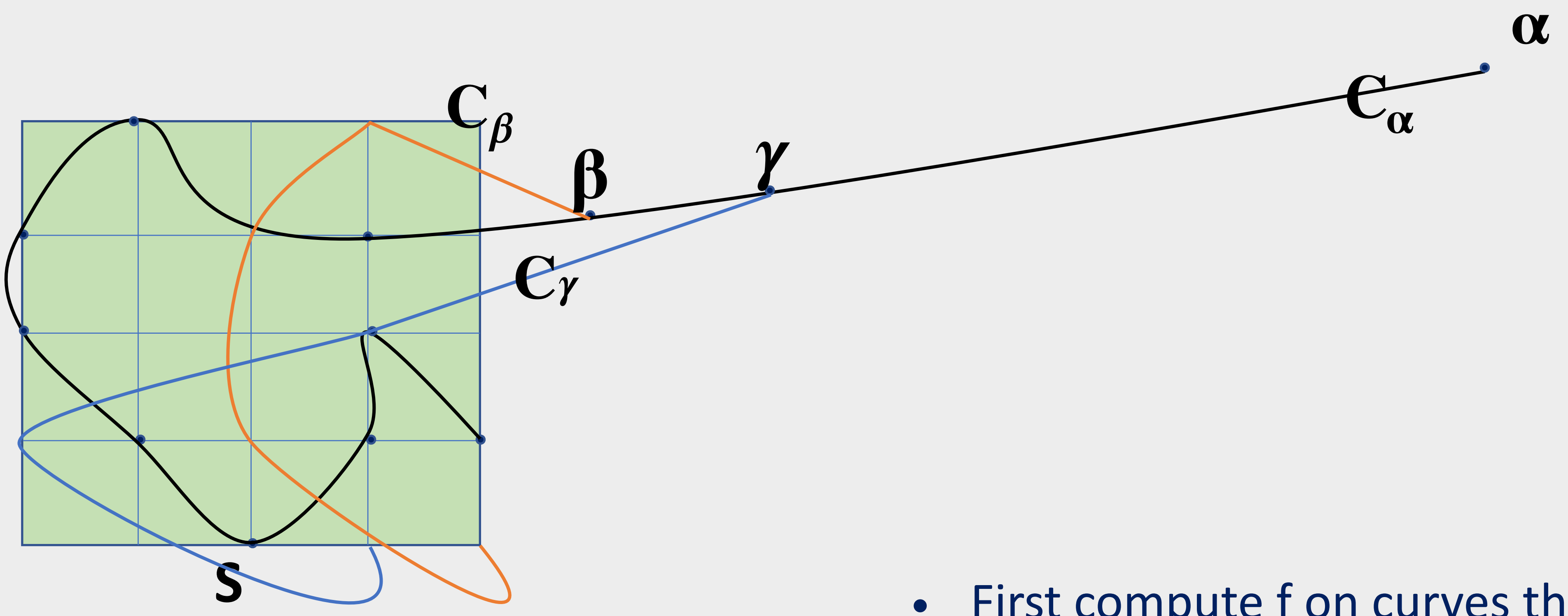
The final inaccurate picture



The final inaccurate picture



The final inaccurate picture



- First compute f on curves through simpler points $\beta,\ \gamma$ using the previous algorithm
- Then, use the values of f on S, and curves through $\beta,\ \gamma$ to compute f on C_α

Multipoint evaluation over all finite fields

Multipoint evaluation over all finite fields

- two different algorithms

Multipoint evaluation over all finite fields

- two different algorithms
- both rely on ideas from the previous algorithm + approach of Kedlaya-Umans + some more ideas (primes in an AP, algorithm of BKW2019)

Multipoint evaluation over all finite fields

- two different algorithms
- both rely on ideas from the previous algorithm + approach of Kedlaya-Umans + some more ideas (primes in an AP, algorithm of BKW2019)
- one completely elementary, but slightly technical to describe, requires the field to be not-too-large

Multipoint evaluation over all finite fields

- two different algorithms
- both rely on ideas from the previous algorithm + approach of Kedlaya-Umans + some more ideas (primes in an AP, algorithm of BKW2019)
- one completely elementary, but slightly technical to describe, requires the field to be not-too-large
- one simpler and shorter to describe, but not entirely elementary

Multipoint evaluation over all finite fields

- two different algorithms
- both rely on ideas from the previous algorithm + approach of Kedlaya-Umans + some more ideas (primes in an AP, algorithm of BKW2019)
- one completely elementary, but slightly technical to describe, requires the field to be not-too-large
- one simpler and shorter to describe, but not entirely elementary
- crucially uses a result of Bombieri-Vinogradov about the density of primes in an arithmetic progression

Multipoint evaluation over all finite fields

- two different algorithms
- both rely on ideas from the previous algorithm + approach of Kedlaya-Umans + some more ideas (primes in an AP, algorithm of BKW2019)
- one completely elementary, but slightly technical to describe, requires the field to be not-too-large
- one simpler and shorter to describe, but not entirely elementary
- crucially uses a result of Bombieri-Vinogradov about the density of primes in an arithmetic progression
- essentially, both improve some of the bottlenecks in Kedlaya-Umans using ideas from the small characteristic case and BKW19 in slightly different ways

Open Questions

- An algebraic algorithm over finite fields ?
- An algorithm (or an algebraic circuit) over infinite fields (complex numbers) ?
- More applications ?
- What about faster algorithms for other related problems ? e.g. multivariate interpolation ?
- What about the case of constant d ? e.g. multilinear polynomials ?

Thank You!