Fast Multivariate Multipoint Evaluation

Based on joint works with

Vishwas Bhargava, Sumanta Ghosh, Zeyu Guo, Chandra Kanta Mohapatra, Chris 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 $(\mathrm{Nmd^m})$ field operations in total

When $\mathrm{N} = \mathrm{d^m}$, 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 $(\mathrm{Nmd}^{\mathrm{m}})$ field operations in total

When $\mathrm{N} = \mathrm{d}^{\mathrm{m}},$ 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 ?

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 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. $|\mathbf{K}|$ is small
2. $|\mathbf{K}|$-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. $|\mathbf{K}|$ is small
2. $|\mathbf{K}|$-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 $\mathrm{d}^{o(1)}$
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)

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

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

In summary

	Field Size	Characteristic	Number of variables	Algebraic vs non-algebraic
Umans	Finite	char(K) <	m <	Algebraic
Kedlaya-Umans	Finite	All finite fields	m <	Non-algebraic
Bhargava-Ghosh-K-Mohapatra	Not-too-large	char(K) <	No constraint	Algebraic
Bhargava-Ghosh-Guo-K-Umans	Finite	All finite fields	No constraint	Non-algebraic

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 $\mathrm{S} \subseteq \mathbf{K}^{\mathrm{m}}$ 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

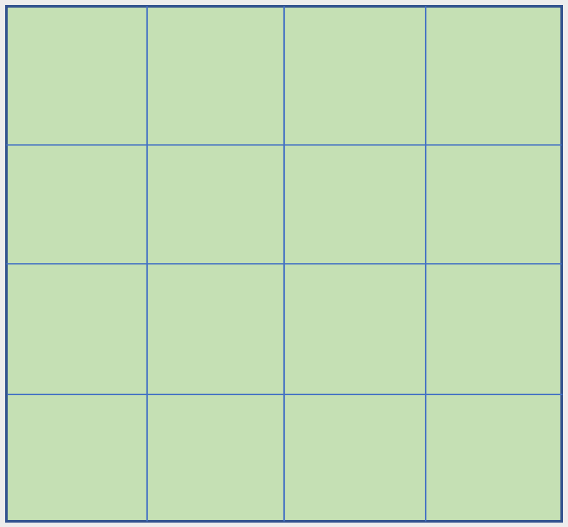
An outline of the algorithm

Local computation

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

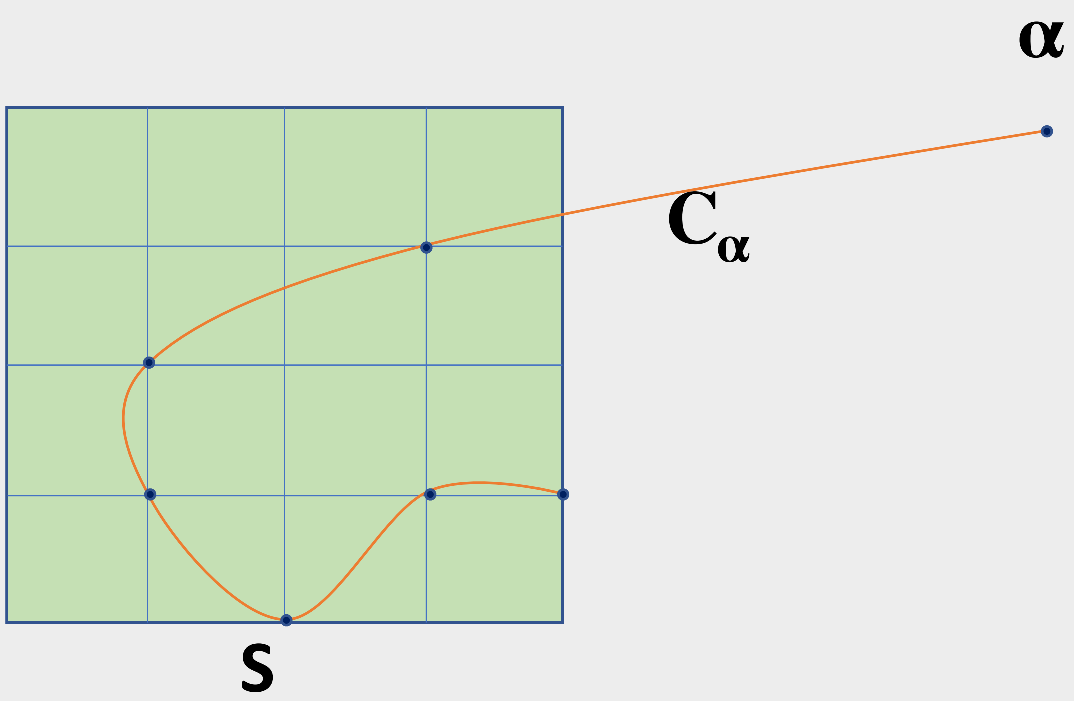
An outline of the algorithm

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 $C_\alpha(y) = (r_{\alpha,1}(y), r_{\alpha,2}(y), \ldots, r_{\alpha,m}(y))$ 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 $C_\alpha(u) = \alpha$

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 $C_\alpha(y)$

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 [$\left(N + d^m\right)^{o(1)}$] 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 [$\left(N + d^m\right)^{o(1)}$] 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 $g(v) = f(r_{\alpha,1}(v), r_{\alpha,2}(v), \ \ldots, r_{\alpha,m}(v))$

An outline of the algorithm

Local computation

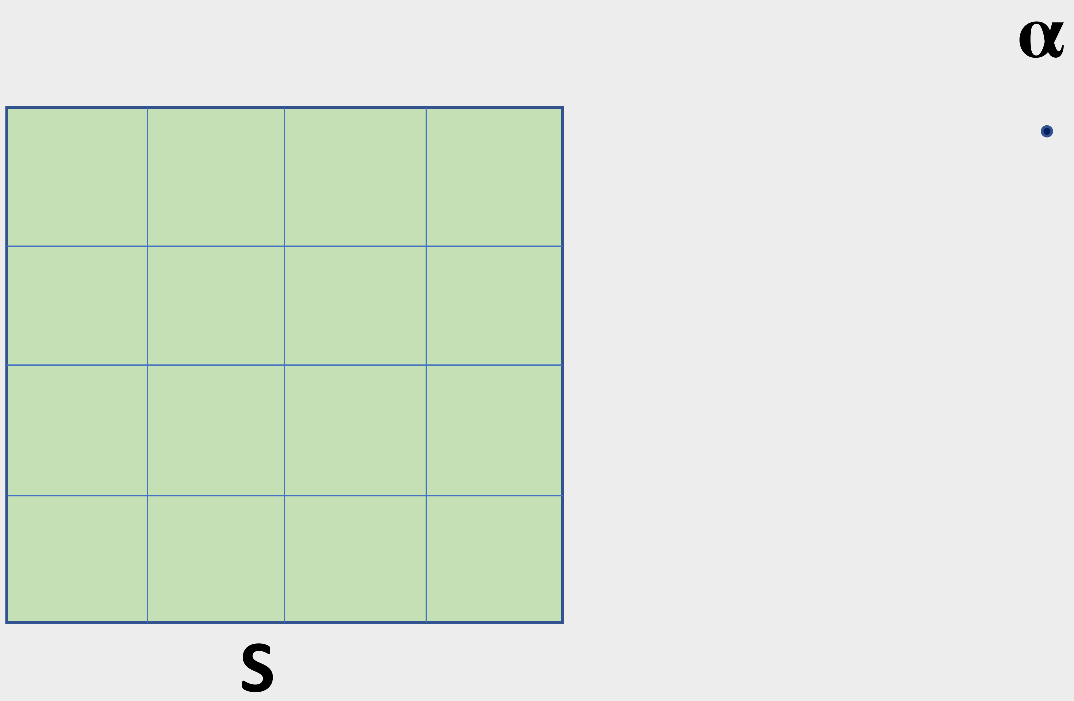
- for each of the N input points, we only have sublinear [$\left(N + d^m\right)^{o(1)}$] time
- from properties of S, we have that C_α intersects S at many points, and we have value of f at all points in S
- let $v \in \mathbf{K}$ be such that $C_\alpha(v) = \left(r_{\alpha,1}(v), r_{\alpha,2}(v), \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

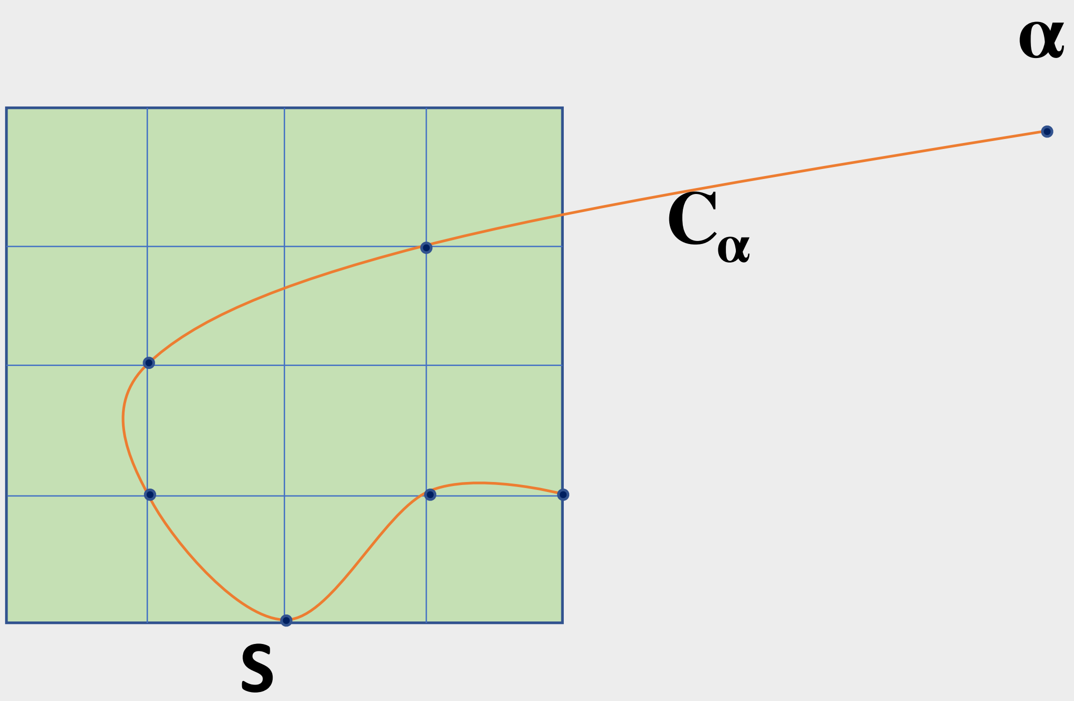
Local computation

- 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 $C_\alpha(v) = \left(r_{\alpha,1}(v), r_{\alpha,2}(v), \ \ldots, r_{\alpha,m}(v)\right)$ 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 $\left|C_\alpha \cap S\right| > \deg(g)$, can recover the polynomial g via interpolation
- once, we have g, can recover $g(u) = f(\alpha)$

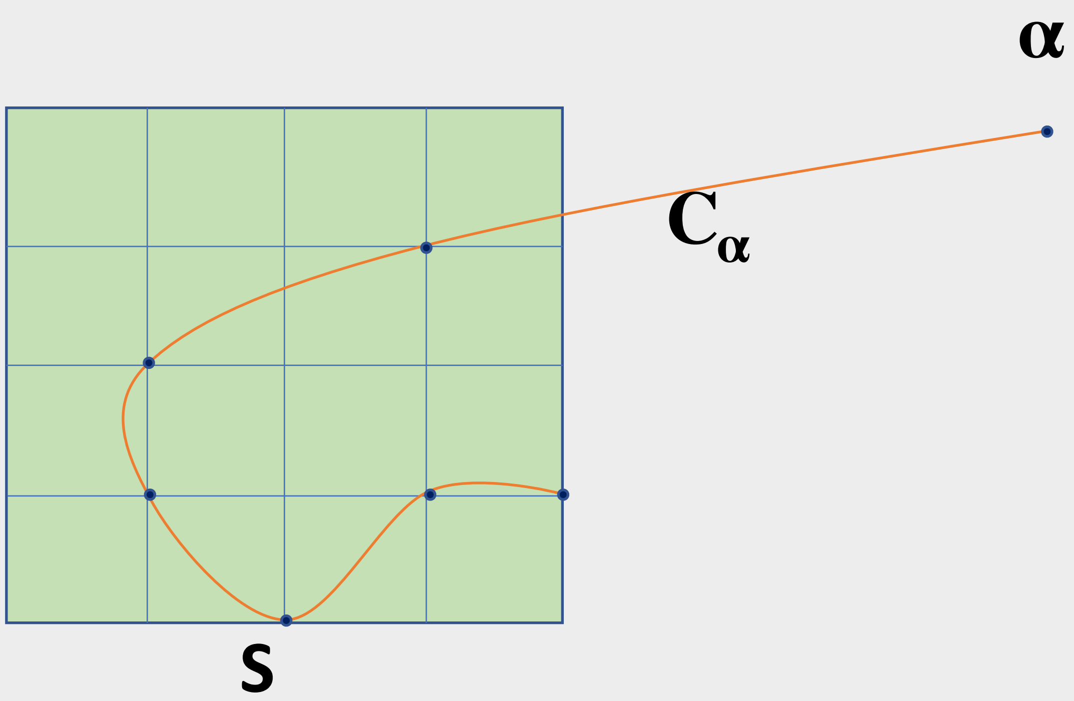
An outline of the algorithm



An outline of the algorithm



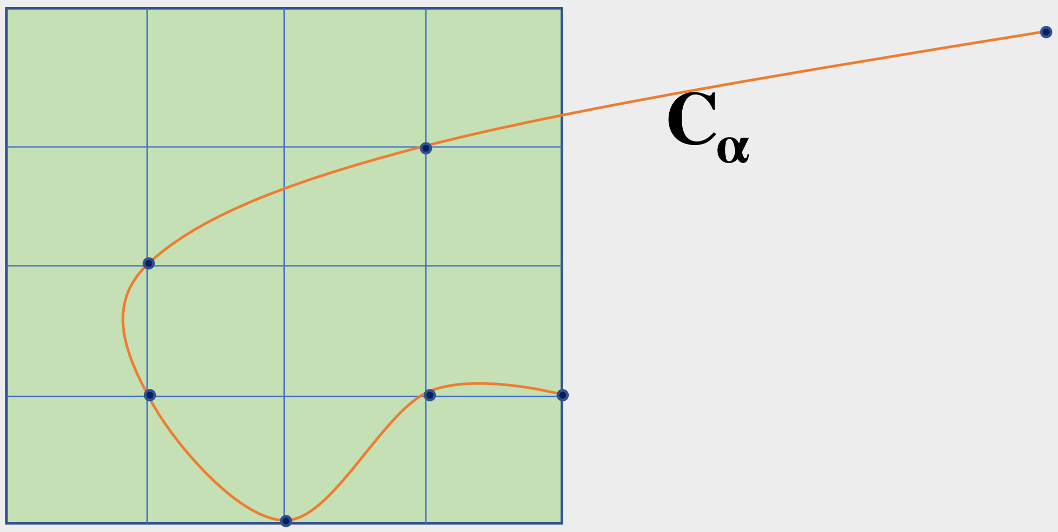
An outline of the algorithm



Want

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

An outline of the algorithm



Want

$$|\mathbf{C}_\alpha \cap S| > \mathbf{deg}(\mathbf{C}_\alpha) \cdot \mathbf{dm}$$

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

The mysterious set S

- ends up being a vector space over a subfield of appropriate size
- requires the characteristic of the underlying field to be small, else, unclear if such a set exists
- curve property follows from structure of field extensions

Running time

Running time

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

N iterations of univariate polynomial interpolation for degree $\log_{\mathbf{p}} |\mathbf{K}| \cdot dm$ + finding the curves at each input

overall running time : $(N + \left(pdm \cdot \log_{\mathbf{p}} |\mathbf{K}|\right)^{\mathbf{m}}) \cdot poly(\log_{\mathbf{p}} |\mathbf{K}| \cdot dm)$

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

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

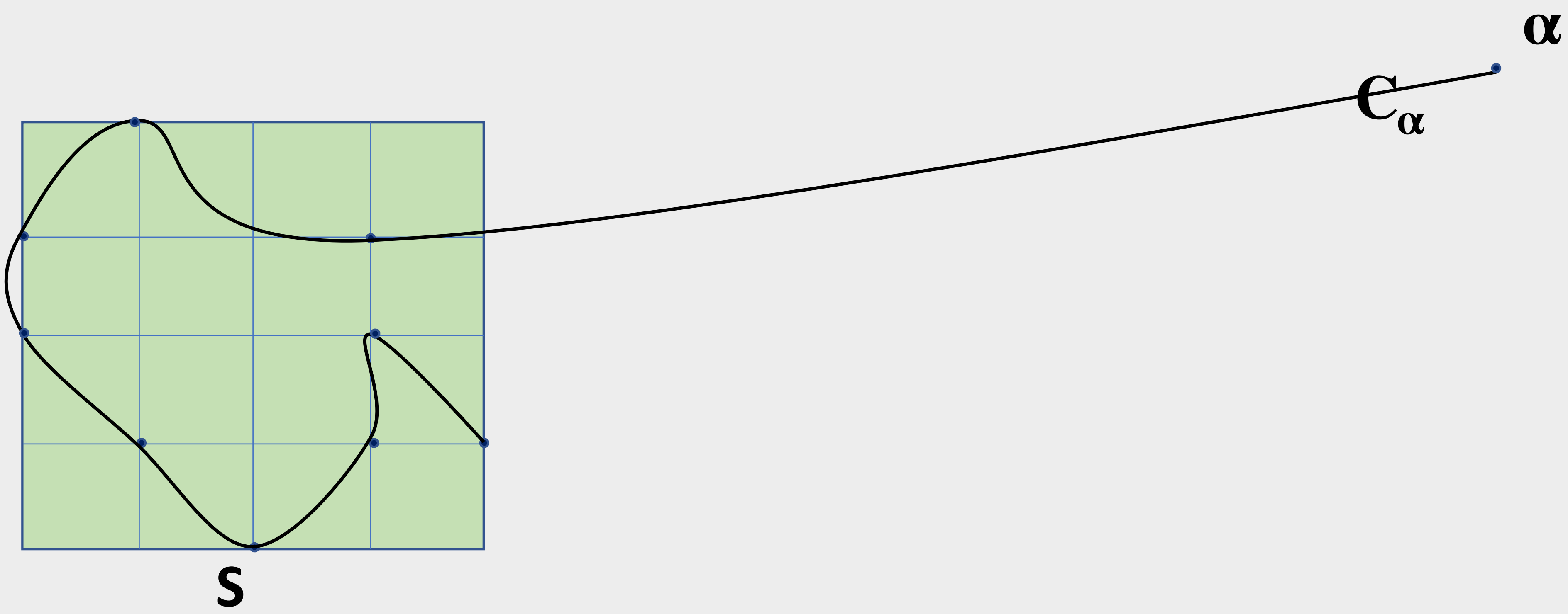
- the degree of the curve through an input point depends on the degree of the field extension that the point lies in
- on each C_α, there are many points β that lie in much lower degree extensions
- so, the value of f is *easier* to *decode* on such points
- instead of computing the restriction of f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

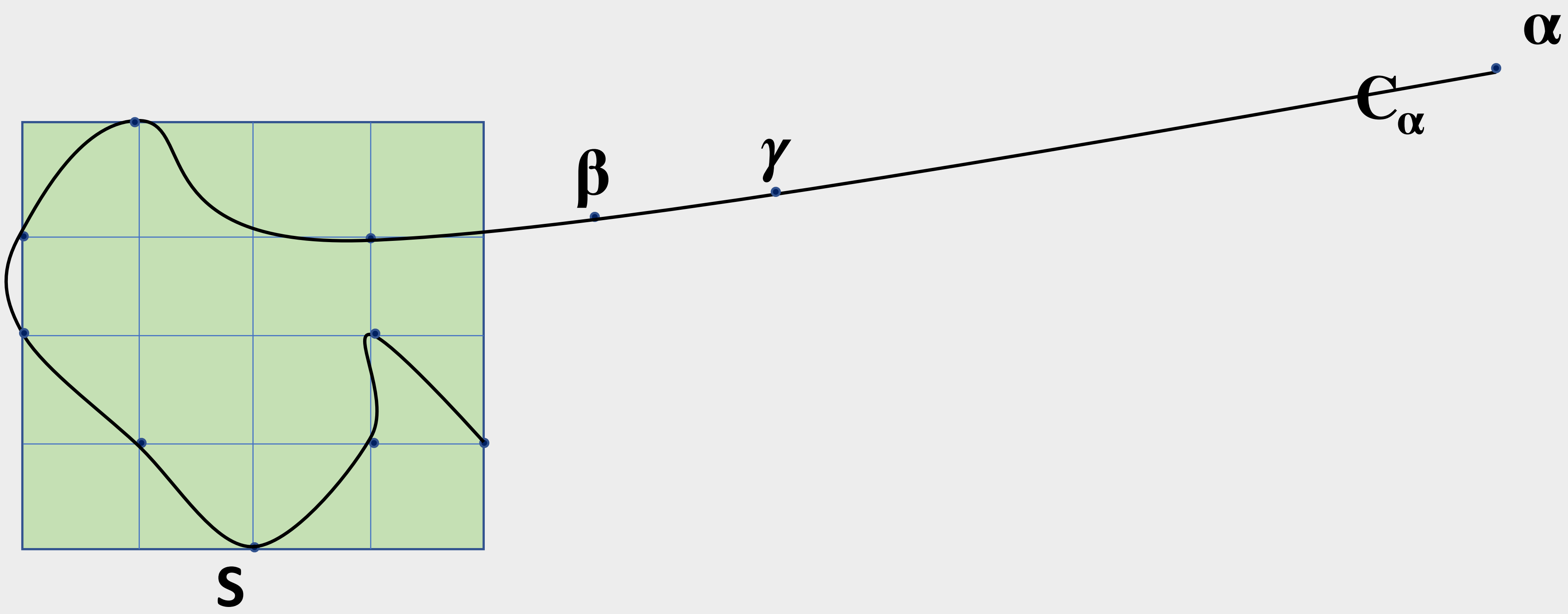
Dealing with large fields

- 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

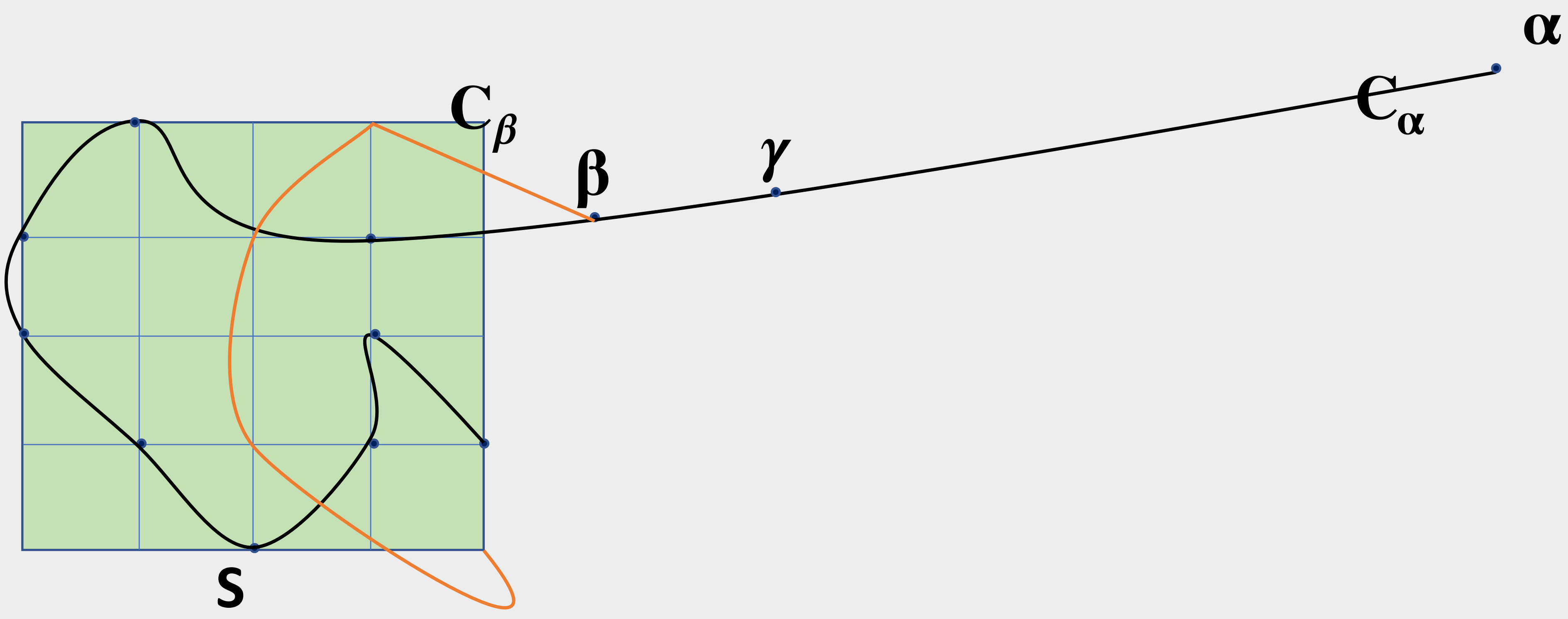
The final inaccurate picture



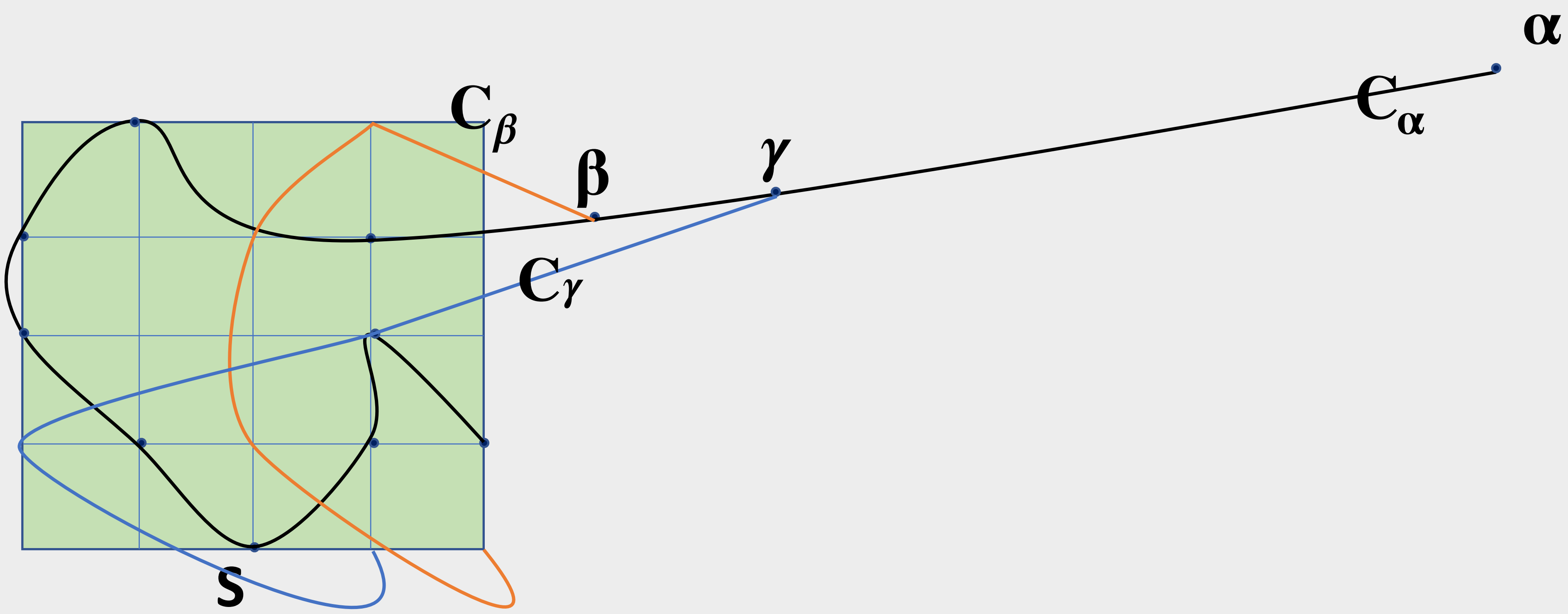
The final inaccurate picture



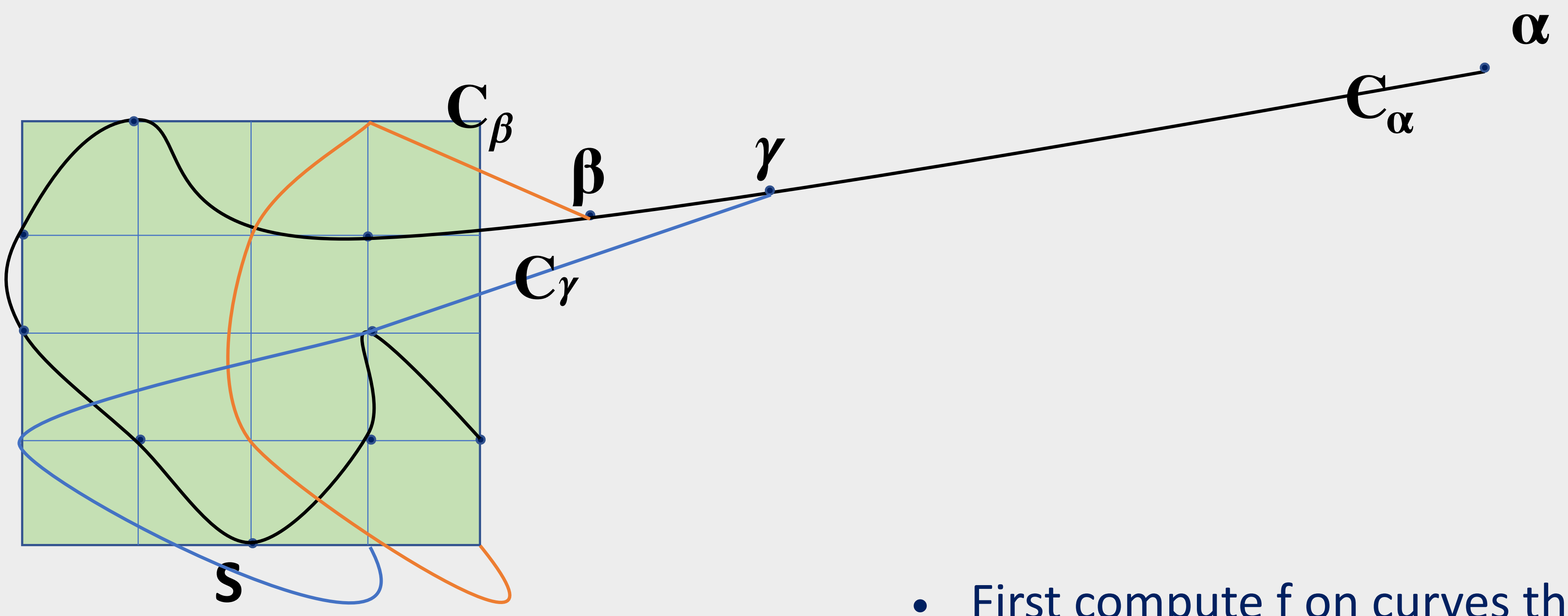
The final inaccurate picture



The final inaccurate picture



The final inaccurate picture



- First compute f on curves through simpler points β, γ using the previous algorithm
- Then, use the values of f on S, and curves through β, γ 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!