UC San Diego

Explicit Newton strata via p-adic cohomology

Jack J Garzella with Ryan Batubara, Yongyuan (Steve) Huang, and Maximus Mellberg

Zeta functions of projective hypersurfaces

- We study smooth projective hypersurfaces
    - example: smooth plane curve
- We compute their zeta functions (i.e. L-polynomials, point counts)
- We use the method of *controlled reduction* due to Costa-Harvey and Costa-Harvey-Kedlaya

$$Z(X,t) = \exp\left(\sum_{n=1}^{\infty} \#X(\mathbb{F}_{q^n}) \frac{t^n}{n}\right)$$

Our results: Zeta functions of projective hypersurfaces

- Zeta functions of quintic curves ($p < 100$)
    - competitive with MAGMA on a single CPU core
- Zeta functions of quartic K3 surfaces ($p < 100$)
    - less than 1 minute on a single core
    - less than 10 seconds on a M3 Mac using 10 cores
- Zeta functions of cubic fourfolds (uses GPU)
    - 2-4 hours in characteristic $p = 7, 11$
    - Competitive with [CHK] in characteristic $p=13$
    - New examples of cubic fourfolds (height 2 examples in $p=7, 11$)

Improvements to the SOTA

1. Hardware usage – Julia + GPU or multi-core > C + single-core
2. Software design – modular components for linear algebra, polynomial recurrences, etc.
3. Algorithmic improvements – the abstract reduction problem

We only have 20 minutes

1. ~~Hardware usage – Julia + GPU or multi-core > C + single-core~~
2. ~~Software design – modular components for linear algebra, polynomial recurrences, etc.~~
3. Algorithmic improvements – the abstract reduction problem

Algorithmic Improvements

The Abstract Reduction Problem

- Encodes a process called “pole reduction”
- Is elementary: requires no understanding of zeta functions nor cohomology
- Is abstracted from controlled reduction work of Costa-Harvey and Costa-Harvey-Kedlaya

If you’re curious how this relates to zeta functions and cohomology, please ask me after

Setup

We will consider elements $u = (u^{(1)}, \ldots, u^{(n)}) \in \mathbb{N}^n$ (where the natural numbers contain zero)

Let $|\cdot|$ be the taxicab norm

We start with a finite set $U \subseteq \mathbb{N}^n$

A picture of U

n=2

Setup

We will consider elements $u = (u^{(1)}, \ldots, u^{(n)}) \in \mathbb{N}^n$ (where the natural numbers contain zero)

Let $|\cdot|$ be the taxicab norm

We start with a finite set $U \subseteq \mathbb{N}^n$

We also have $N \in \mathbb{N}$, called the *target norm*. We also fix $d \in \mathbb{N}$, the *degree*

All $u \in U$ have $|u| = N + md$ for some $m \in \mathbb{N}$

A picture of N

n=2, N=2, d=2

Setup, continued

For u with $N < |u|$ and $k \in \mathbb{N}$, we also have $V(u,k) \subseteq \mathbb{N}^n$, called the *direction choice collection*, which is subject to three conditions

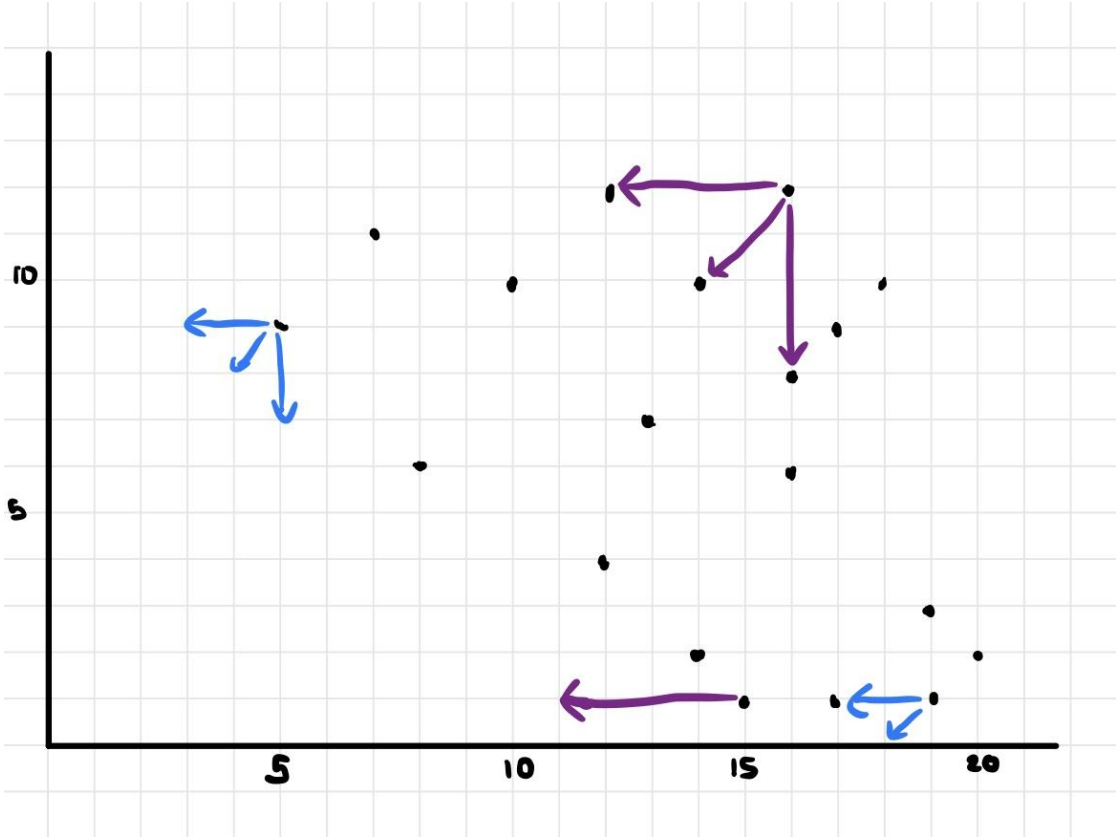
- for $v \in V(u,k)$, $|v| = d$
- $V(u,1)$ is nonempty
- $V(u,k) \subseteq V_{all}(u,k)$

A picture of $V_{all}(u,k)$

n=2, N=2, d=2

Blue: $V_{all}(u,1)$

Purple: $V_{all}(u,2)$



A picture of $V_{all}(u,k)$

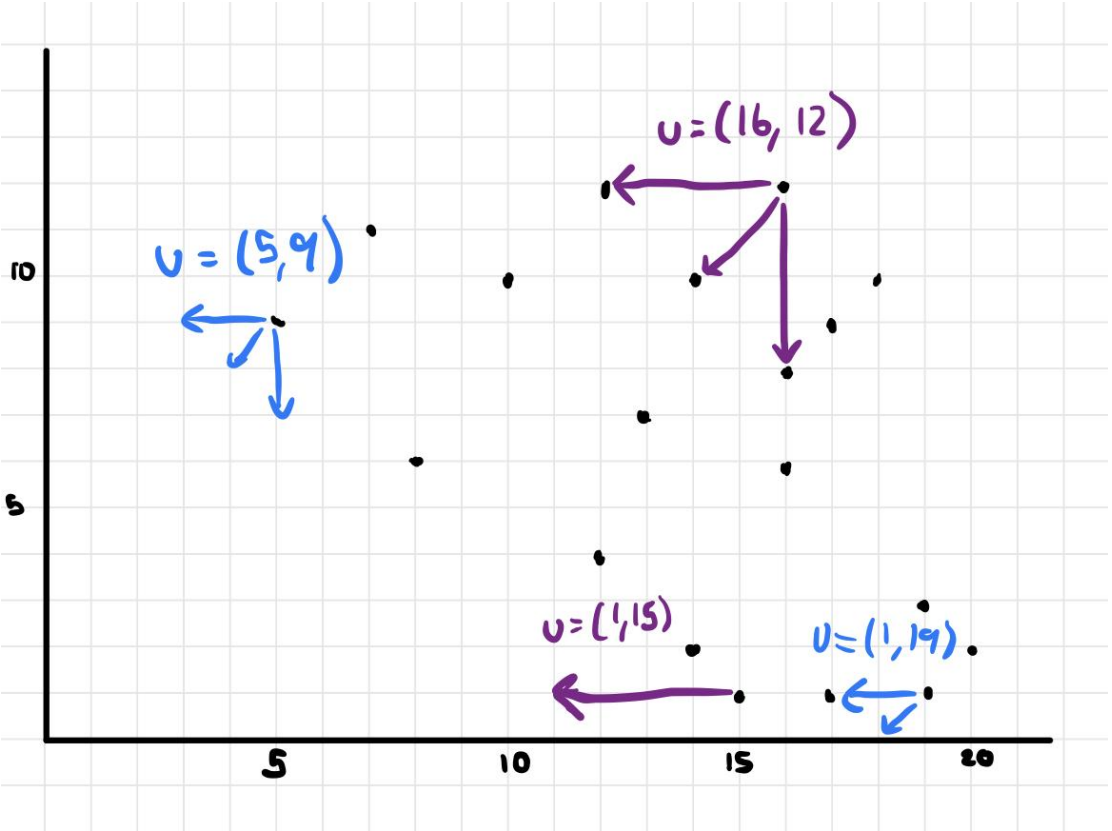
n=2, N=2, d=2

$V_{all}((5,9),1) = \{(2,0),(1,1),(0,2)\}$

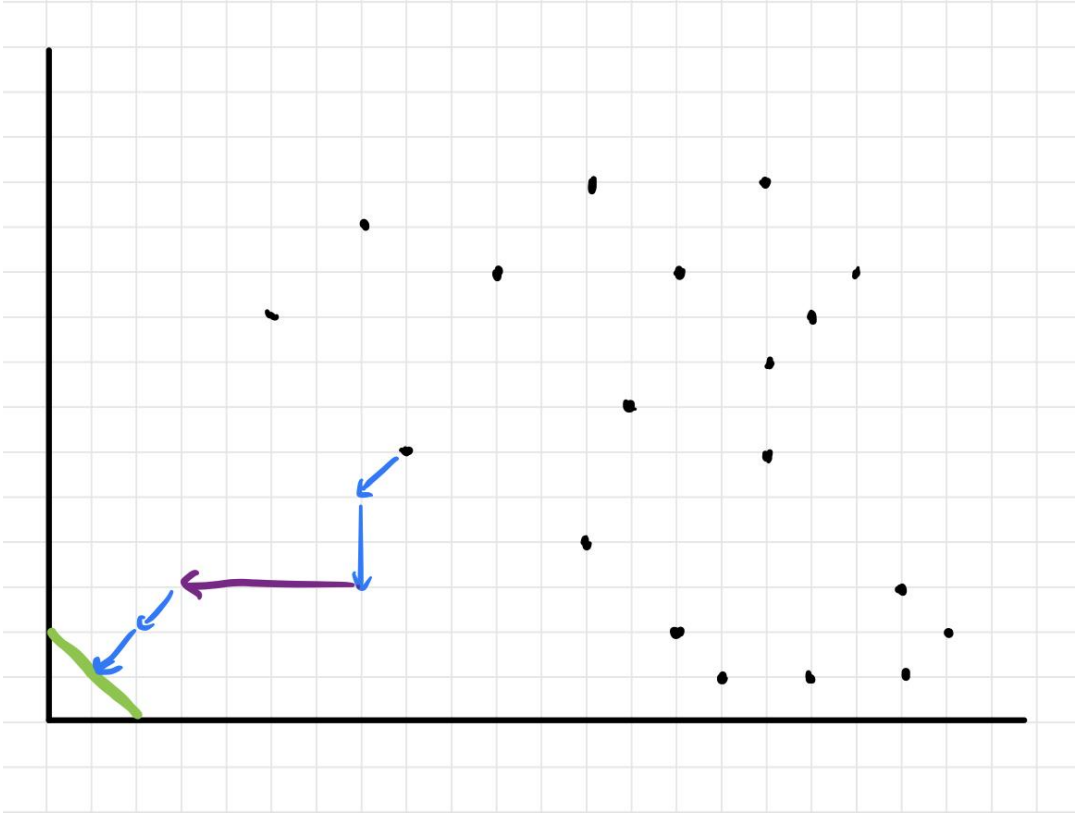
$V_{all}((1,19),1) = \{(2,0),(1,1)\}$

$V_{all}((16,12),2) = \{(2,0),(1,1),(0,2)\}$

$V_{all}((1,15),2) = \{(2,0)\}$



A picture of reduction



Cost Constants

We fix constants $D << T << M$.

D corresponds to one vector addition

T corresponds to a 6-8 matrix additions

M corresponds to one matrix-vector multiplication and one matrix addition

In practice, memory overhead dominates all cases, so we use this three-tiered abstraction instead of counting operations.

The Abstract Reduction Problem

Given, a U, $V(u,k)$, N, d, subject to the conditions in the setup, the *abstract reduction problem* is a one-player game where the player may make one of two moves, each of which has a cost

The game is won when ever $u \in U$ has $|u| = N$

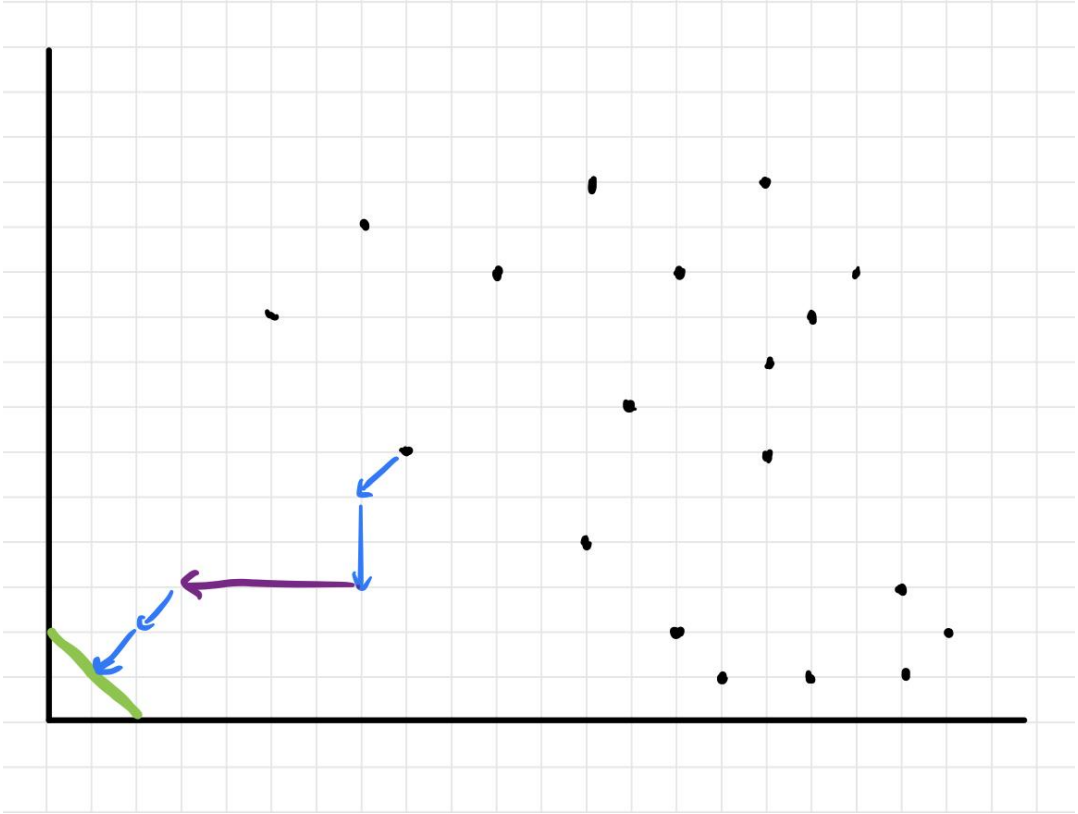
The goal is to win the game while minimizing the cost

Possible moves:

1. **Reduction chunk of size k**
   - Choose a k and a $v \in V(u,k)$. Replace $u \in U$ by $u - kv$ for a cost of $Mk + T$
2. **Combine like terms**
   - Remove all duplicates in U for a cost of D per duplicate

A program which for any possible U runs and wins the game is called a *reduction policy*

A picture of reduction



Basic strategy

Possible moves:

1. **Reduction chunk of size k**
   - Choose a k and a $v \in V(u,k)$. Replace $u \in U$ by $u - kv$ for a cost of $Mk + T$
2. **Combine like terms**
   - Remove all duplicates in U for a cost of D per duplicate

Any reduction policy needs to balance the savings of combining like terms with the savings of doing longer reduction chunks

Our contributions

In previous work, this “abstract reduction problem” is an implementation detail only

However, it can have a big effect on performance, especially when memory is tight

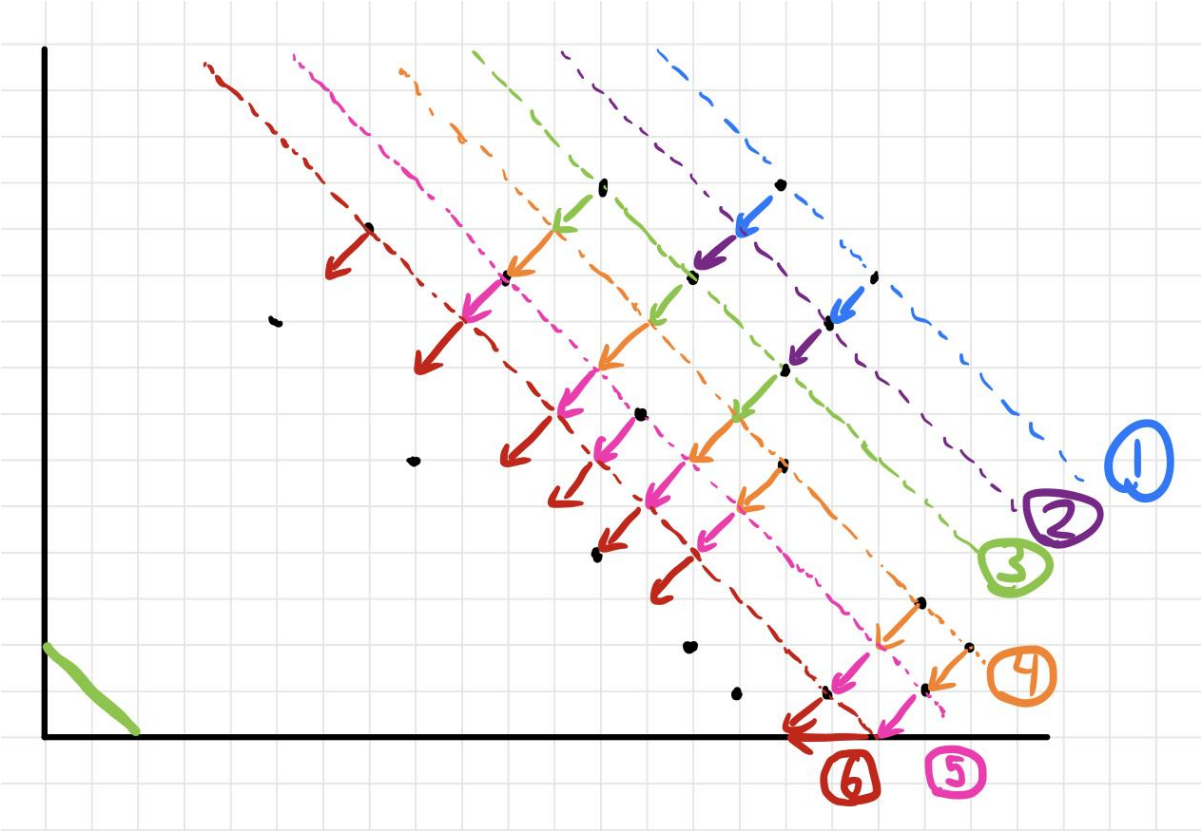
We describe and implement three reduction policies

p-chunk policy

- Let $k = \min\{|u| - |w| : u, w \in U\}$
- while the game is not yet won
  - Find all u such that $|u| = \max\{|u| : u \in U\}$
  - For each such u
    - do a reduction chunk of size k in any direction (pick arbitrary priority order)
  - combine like terms

This policy is named for the fact that in our applications (zeta functions of hypersurfaces in characteristic p), k is always p

A picture of p-chunk reduction



pros and cons of p-chunk reduction

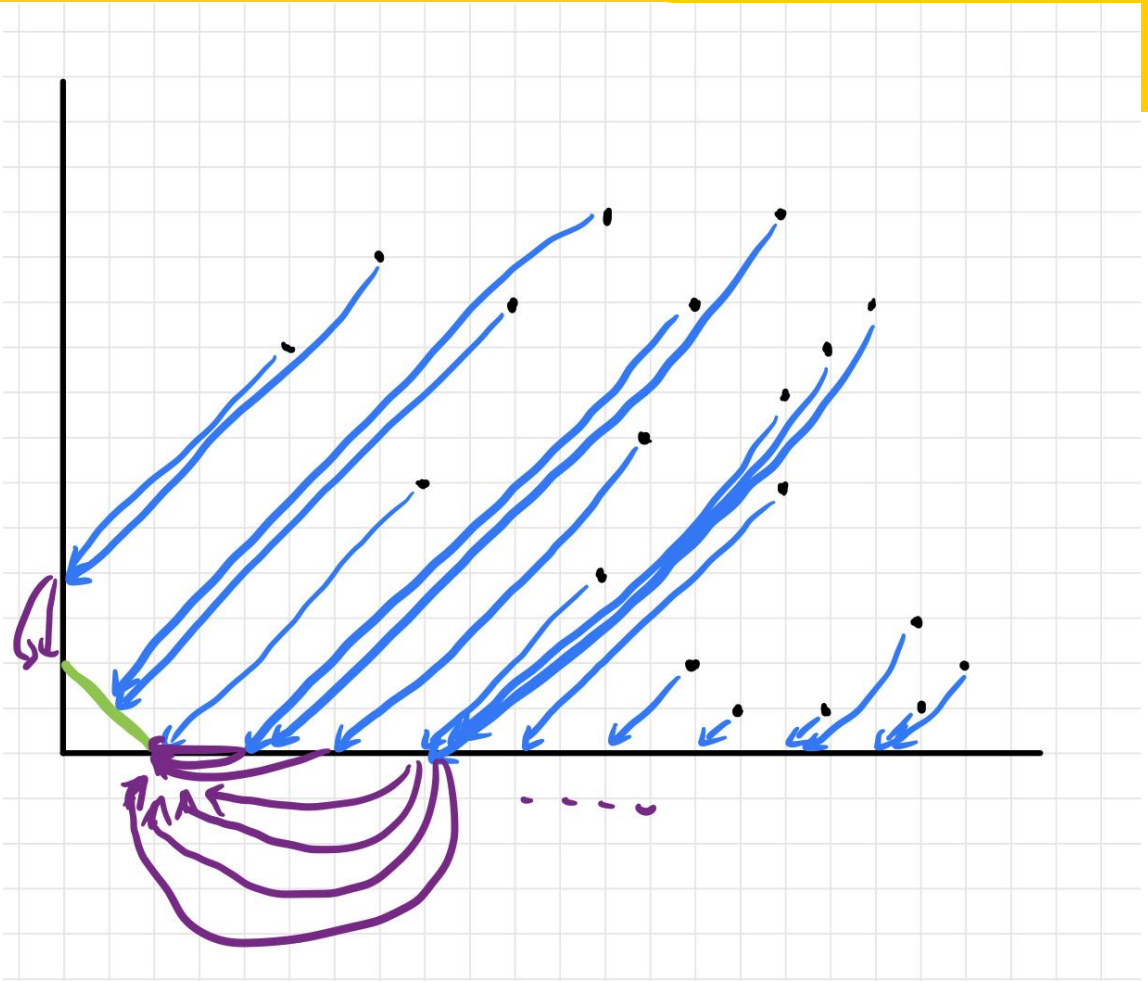
- pro: long chunks when p is large
- con: when p is small, we get lots of startup cost
- con: the combining of like terms is haphazard, in practice it might not be doing much

This policy is used in previous work of Costa-Harvey and Costa-Harvey-Kedlaya, though to them it is an implementation detail

depth first policy

- For each $u \in U$,
  - While N < |u|
    - Pick any direction v (pick arbitrary priority order)
    - Find the largest k such that V(u,k) is nonempty
    - Do a reduction chunk of size k in the direction v

A picture of depth-first reduction



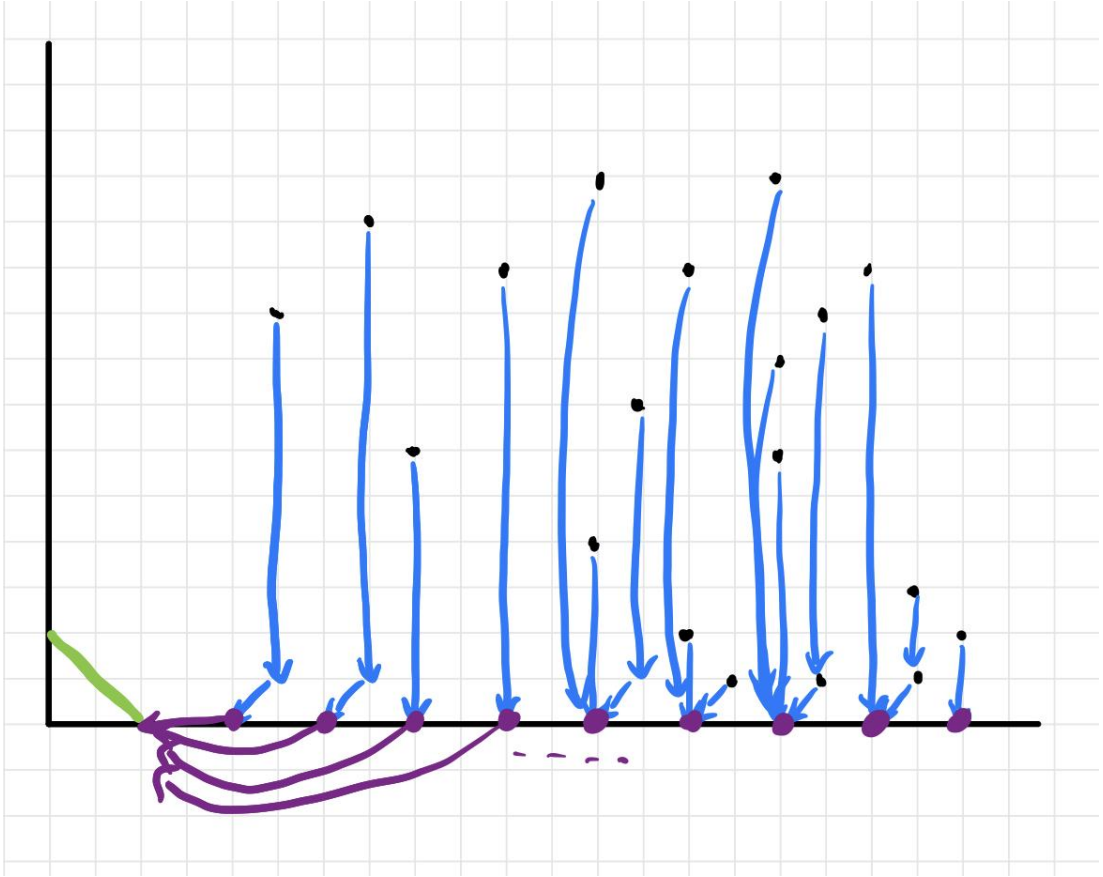
pros and cons of depth-first reduction

- pro: much longer chunks when p is small
- con: never takes advantage of combining like terms
- This is our state of the art for curves and surfaces ($p < 100$)

var-by-var policy

- for i in 1 to n
  - for $u \in U$
    - find the largest k such that $v = (0, \ldots, d, \ldots 0) \in V(u,k)$, where the nonzero component is the ith term (requires such a k to exist in $V(u,k)$)
    - Do a reduction chunk of size k in direction v
    - If necessary, do a reduction chunk in one of the following directions: $(0, \ldots, d\text{-}1, 1, \ldots, 0)$, $(0, \ldots, d\text{-}2, 2, \ldots, 0)$ … $(0, \ldots, 1, d\text{-}1, \ldots, 0)$, to ensure that the ith component of u is zero
  - combine like terms

A picture of var-by-var reduction



pros and cons of var-by-var reduction

- pro: lots of combining like terms when n is large, and U is very dense
- con: when d is large, there's a lot of short chunk overhead to adjust everything to exactly u = 0
- This is our state of the art for cubic threefolds and cubic fourfolds ($p < 15$)

future work

- policies that are tuned for
  - specific hardware configurations (GPU, single-core, multi-core)
  - specific classes of varieties (e.g. K3 surfaces) or base fields
- policy chosen based on analysis of this specific variety
- Generate policies using AI
  - automate tuning process
  - reinforcement learning / genetic algorithm (i.e. funsearch)
    - at the configuration/variety level
    - at the specific equation level
    - using an “abstract reduction problem simulation” with cost constants instead of doing actual matrix computations

UC San Diego

Thank you!

Hardware usage

- The Julia programming language provides very simple ways to take advantage of
  - **eliminating allocations in inner loops**
  - multi-core (via @spawn and @threads)
  - the GPU (via CUDA.jl and KernelAbstractions.jl)

- Julia's base single-core performance is comparable to C
  - C wins when Julia's overhead is a bottleneck
  - Julia wins when there is a fast library implementation that Julia can easily call
  - In either case, the difference probably isn't much more than 2x

Software design

- In our work, we designed modular components for
  - Linear algebra
  - Polynomial recurrences (especially memory management thereof)
  - Reduction policies

- In the future, we plan to use a similar structure for “geometric frontends” or classes of varieties, for example by adding hyperelliptic curves or degree 2 K3 surfaces