List Decoding of Algebraic Codes

Peter Beelen, Kristian Brander and Johan S.R. Nielsen

DTU Mathematics
Technical University of Denmark

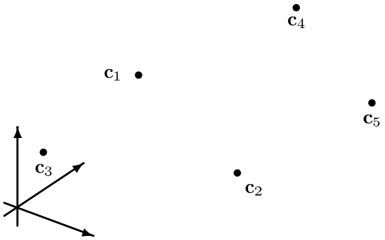
Contents

1. List decoding of error-correcting codes
2. Fast list decoding of Reed–Solomon codes
3. Fast list decoding of certain AG codes
4. Wu decoding of Reed–Solomon codes

Contents

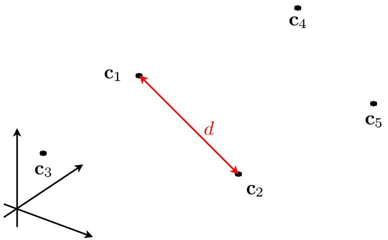
1. List decoding of error-correcting codes
2. Fast list decoding of Reed–Solomon codes
3. Fast list decoding of certain AG codes
4. Wu decoding of Reed–Solomon codes

Codewords and unique decoding



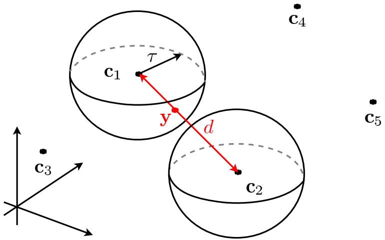
- Codewords: Vectors $\mathbf{c} \in \Sigma^n$. Code: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$.

Codewords and unique decoding



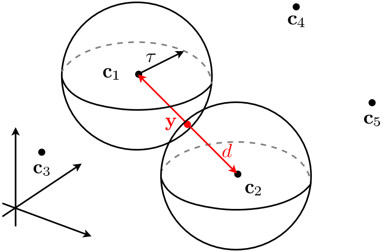
- Codewords: Vectors $\mathbf{c} \in \Sigma^n$. Code: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$.
- Minimum distance, d, is the minimal number of disagreeing positions between any two codewords.

Codewords and unique decoding



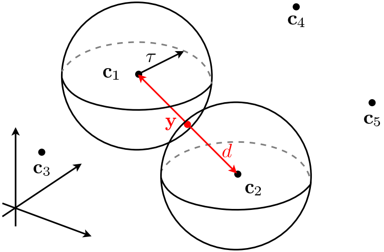
- Codewords: Vectors $\mathbf{c} \in \Sigma^n$. Code: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$.
- Minimum distance, d, is the minimal number of disagreeing positions between any two codewords.
- If the number of errors, τ, is less than $\frac{d}{2}$ then there is at most one codeword within distance τ from any received word $\mathbf{y}$.

List decoding



- If $\tau \geq \frac{d}{2}$ there might be a "small" list of codewords within distance τ from $\mathbf{y}$.
- The decoder thus get a list of candidate messages.

List decoding



- If $\tau \geq \frac{d}{2}$ there might be a "small" list of codewords within distance τ from $\mathbf{y}$.
- The decoder thus get a list of candidate messages.
- We require the lists to be polynomially bounded in the code length n.

Error-correcting codes and list decoding

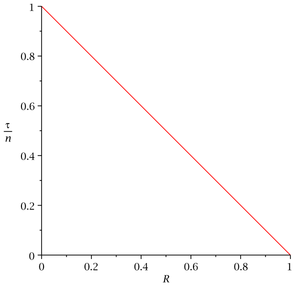
- The rate of an error-correcting code is rate $R = \frac{\log_{|\Sigma|}(|\mathcal{C}|)}{n}$.

Error-correcting codes and list decoding

- The rate of an error-correcting code is rate $R = \frac{\log_{|\Sigma|}(|\mathcal{C}|)}{n}$.
- The relative number of errors it can correct is denoted by $\frac{\tau}{n}$.

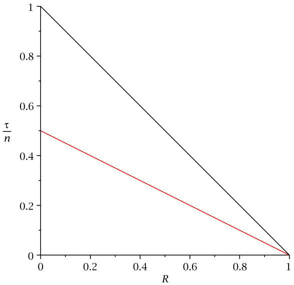
Error-correcting codes and list decoding

- The rate of an error-correcting code is rate $R = \frac{\log_{|\Sigma|}(|\mathcal{C}|)}{n}$.
- The relative number of errors it can correct is denoted by $\frac{\tau}{n}$.



Error-correcting codes and list decoding

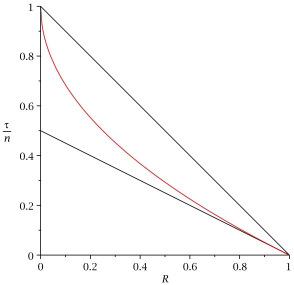
- The rate of an error-correcting code is rate $R = \frac{\log_{|\Sigma|}(|\mathcal{C}|)}{n}$.
- The relative number of errors it can correct is denoted by $\frac{\tau}{n}$.



- Unique decoding: $\tau/n < \frac{1}{2}(1-R)$.

Error-correcting codes and list decoding

- The rate of an error-correcting code is rate $R = \frac{\log_{|\Sigma|}(|\mathcal{C}|)}{n}$.
- The relative number of errors it can correct is denoted by $\frac{\tau}{n}$.



- Unique decoding: $\tau/n < \frac{1}{2}(1-R)$.
- Guruswami–Sudan algorithm: $\tau/n < 1-\sqrt{R}$.

- Furthermore: The code must be efficiently list decodable.

Contents

1 List decoding of error-correcting codes

2 Fast list decoding of Reed–Solomon codes

3 Fast list decoding of certain AG codes

4 Wu decoding of Reed–Solomon codes

Reed–Solomon codes

- A Reed–Solomon code of length n and rate $R = k/n$:

$$\mathcal{C} = \{(f(\alpha_1), \dots, f(\alpha_n)) \mid f(x) \in \mathbb{F}_q[x], \deg(f) < k\},$$

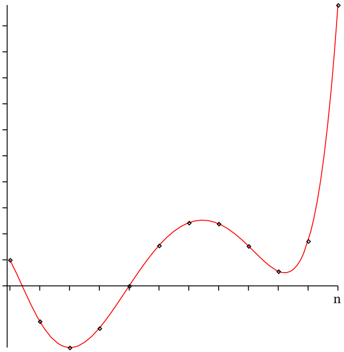
Alphabet is $\Sigma = \mathbb{F}_q$ and $\alpha_1, \dots, \alpha_n \in \mathbb{F}_q$ are distinct.

Reed–Solomon codes

- A Reed–Solomon code of length n and rate $R = k/n$:

$$\mathcal{C} = \{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f(x) \in \mathbb{F}_q[x], \deg(f) < k\},$$

Alphabet is $\Sigma = \mathbb{F}_q$ and $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ are distinct.

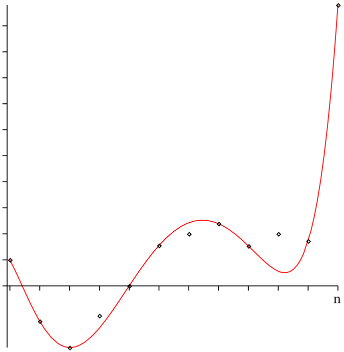


Reed–Solomon codes

- A Reed–Solomon code of length n and rate $R = k/n$:

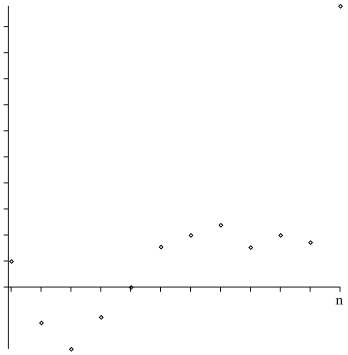
$$\mathcal{C} = \{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f(x) \in \mathbb{F}_q[x], \deg(f) < k\},$$

Alphabet is $\Sigma = \mathbb{F}_q$ and $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ are distinct.



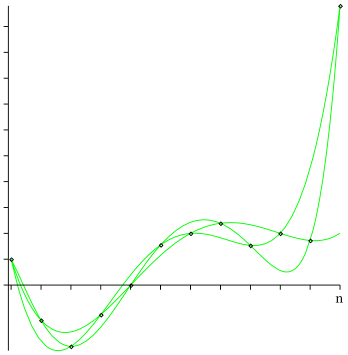
List decoding Reed–Solomon codes

- A list decoder must find $f(x) \in \mathbb{F}_q[x]$, with $\deg(f) < k$, that passes through $n - \tau$ of the received points.



List decoding Reed–Solomon codes

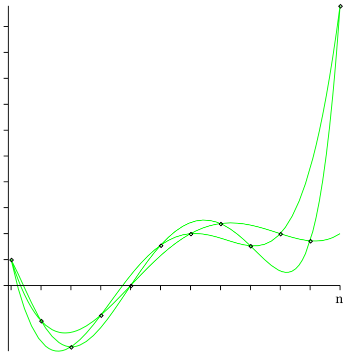
- A list decoder must find $f(x) \in \mathbb{F}_q[x]$, with $\deg(f) < k$, that passes through $n - \tau$ of the received points.



- Interpolate $Q(x, y)$ through received points, with multiplicity s.

List decoding Reed–Solomon codes

- A list decoder must find $f(x) \in \mathbb{F}_q[x]$, with $\deg(f) < k$, that passes through $n - \tau$ of the received points.

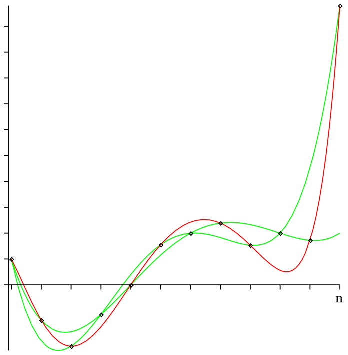


- Interpolate $Q(x, y)$ through received points, with multiplicity s.
- ... of least weighted degree.

$$\deg_w(x^i y^j) = i + (k-1)j$$

List decoding Reed–Solomon codes

- A list decoder must find $f(x) \in \mathbb{F}_q[x]$, with $\deg(f) < k$, that passes through $n - \tau$ of the received points.



- Interpolate $Q(x, y)$ through received points, with multiplicity s.
- ... of least weighted degree.

$$\deg_w(x^i y^j) = i + (k-1)j$$

- If $\tau/n < 1 - \sqrt{R}$ then

$$Q(x, f(x)) = 0$$

Translation of the interpolation problem

- List decoding depends on a fast interpolation algorithm.

Translation of the interpolation problem

- List decoding depends on a fast interpolation algorithm.
- The $\mathbb{F}_q[x]$–module of interpolation polynomials with $\deg_y(Q) \le \ell$, is spanned by

$$\left\{E^s, E^{s-1}(y-R), \ldots, (y-R)^s, (y-R)^{s+1}, \ldots, (y-R)^\ell\right\},$$

where $E(x) = \prod_{i=1}^n (x - \alpha_i)$ and $R(\alpha_i) = y_i$ for $1 \le i \le n$.

Translation of the interpolation problem

- List decoding depends on a fast interpolation algorithm.
- The $\mathbb{F}_q[x]$–module of interpolation polynomials with $\deg_y(Q) \leq \ell$, is spanned by

$$\left\{E^s, E^{s-1}(y-R), \ldots, (y-R)^s, (y-R)^{s+1}, \ldots, (y-R)^{\ell}\right\},$$

  where $E(x) = \prod_{i=1}^{n}(x-\alpha_i)$ and $R(\alpha_i) = y_i$ for $1 \leq i \leq n$.
- Introduce matrix $\ell+1 \times \ell+1$ matrix $\mathbf{A}$,

$$[\mathbf{A}]_{ij} = \text{Coefficient to } y^i \text{ in } j\text{-th basis function}$$

Translation of the interpolation problem

- List decoding depends on a fast interpolation algorithm.
- The $\mathbb{F}_q[x]$–module of interpolation polynomials with $\deg_y(Q) \leq \ell$, is spanned by

$$\left\{E^s, E^{s-1}(y-R), \ldots, (y-R)^s, (y-R)^{s+1}, \ldots, (y-R)^{\ell}\right\},$$

where $E(x) = \prod_{i=1}^{n}(x-\alpha_i)$ and $R(\alpha_i) = y_i$ for $1 \leq i \leq n$.

- Introduce matrix $\ell+1 \times \ell+1$ matrix $\mathbf{A}$,

$$[\mathbf{A}]_{ij} = \text{Coefficient to } y^i \text{ in } j\text{-th basis function}$$

- Then,

$$Q(x,y) = \sum_{i=0}^{\ell} q_i(x) y^i \in \mathbb{F}_q[x,y],$$

is an interpolation polynomial if and only if $\mathbf{q} = (q_0, \ldots, q_\ell)$ is in the $\mathbb{F}_q[x]$–column span of $\mathbf{A}$.

Interpolation

- For $s = 2$ and $\ell = 3$,

$$\mathbf{A} = \begin{bmatrix} E^2 & -ER & R^2 & -R^3 \\ 0 & E & -2R & 3R^2 \\ 0 & 0 & 1 & -3R \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Interpolation

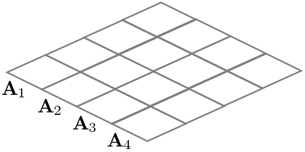
- For $s = 2$ and $\ell = 3$,

$$\mathbf{A} = \begin{bmatrix} E^2 & -ER & R^2 & -R^3 \\ 0 & E & -2R & 3R^2 \\ 0 & 0 & 1 & -3R \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

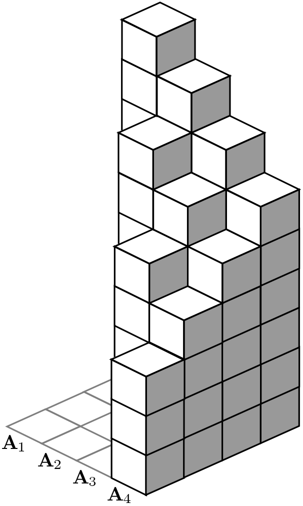
- The column span of **A** gives all interpolation polynomials. We look for short vectors, with respect to weighted degree.
- Gaussian elimination-style algorithm: Cancel highest terms.

Algorithm: Gaussian elimination

- Represent matrix as grid.



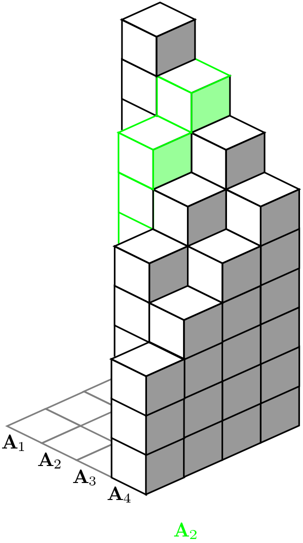
Algorithm: Gaussian elimination



- Represent matrix as grid.
- Represent (i, j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

Algorithm: Gaussian elimination

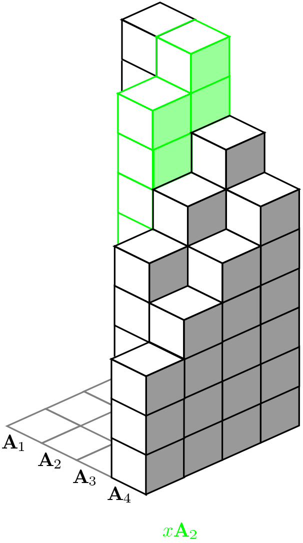


- Represent matrix as grid.
- Represent (i,j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.

Algorithm: Gaussian elimination

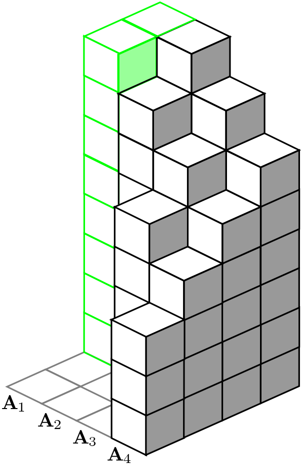


- Represent matrix as grid.
- Represent (i, j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.

Algorithm: Gaussian elimination



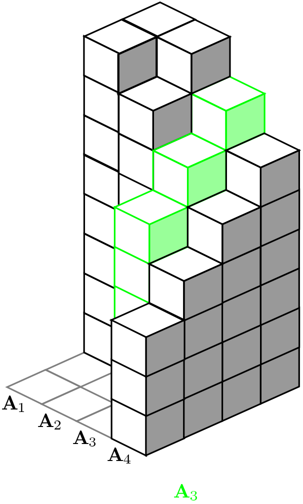
$\mathbf{A}_1 + \alpha x \mathbf{A}_2 \to \mathbf{A}_1$

- Represent matrix as grid.
- Represent (i, j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.

Algorithm: Gaussian elimination

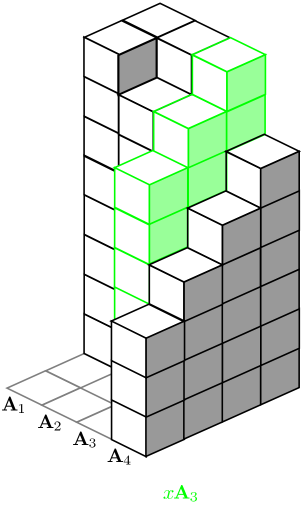


- Represent matrix as grid.
- Represent (i,j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.

Algorithm: Gaussian elimination

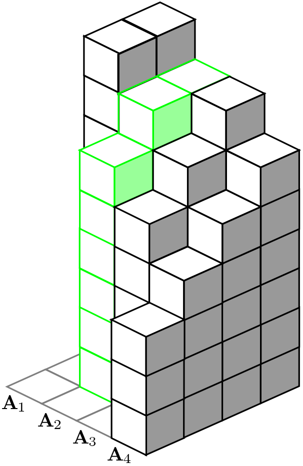


- Represent matrix as grid.
- Represent (i, j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.

Algorithm: Gaussian elimination



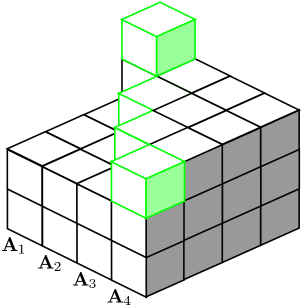
$\mathbf{A}_2 + \alpha x \mathbf{A}_3 \to \mathbf{A}_2$

- Represent matrix as grid.
- Represent (i, j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.

Algorithm: Gaussian elimination

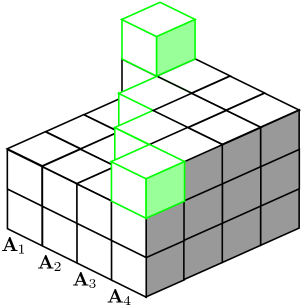


- Represent matrix as grid.
- Represent (i, j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.
- Continue the process, until leading coordinates occur in distinct rows.

Algorithm: Gaussian elimination



- Represent matrix as grid.
- Represent (i, j)-th entry by stack of cubes:

$$\deg_w(\mathbf{A}_{i,j}) = \deg(\mathbf{A}_{i,j}) + (k-1)j.$$

- Gaussian elimination.
- Continue the process, until leading coordinates occur in distinct rows.
- Leads to algorithm requiring $\mathcal{O}\left(\ell^5 n^2\right)$ $\mathbb{F}_q$-multiplications.

Algorithm: Divide and conquer

- Extend and generalize idea behind divide and conquer algorithm by Alekhnovich.
- Introduce matrix $\mathbf{U}(\mathbf{A}, t)$ representing the column operations made when "cutting down" the stack, i.e.
  - $\deg_w(\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t)) \leq \deg_w(\mathbf{A}) - t$ or
  - $\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t)$ has all leading coordinates in distinct rows,

where $\deg_w(\mathbf{A}) = \sum_i \deg_w(\mathbf{A}_i)$.

Algorithm: Divide and conquer

- Extend and generalize idea behind divide and conquer algorithm by Alekhnovich.
- Introduce matrix $\mathbf{U}(\mathbf{A}, t)$ representing the column operations made when "cutting down" the stack, i.e.
  - $\deg_w(\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t)) \leq \deg_w(\mathbf{A}) - t$ or
  - $\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t)$ has all leading coordinates in distinct rows,

  where $\deg_w(\mathbf{A}) = \sum_i \deg_w(\mathbf{A}_i)$.
- Observation:

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\mathbf{A}, \lceil t/2 \rceil) \cdot \mathbf{U}(\mathbf{A}', t - d),$$

where $\mathbf{A}' = \mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t/2)$ and $d = \deg_w \mathbf{A} - \deg_w \mathbf{A}'$.

Algorithm: Divide and conquer

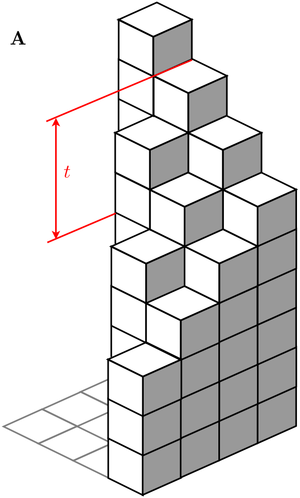
- Extend and generalize idea behind divide and conquer algorithm by Alekhnovich.
- Introduce matrix $\mathbf{U}(\mathbf{A}, t)$ representing the column operations made when "cutting down" the stack, i.e.
  - $\deg_w(\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t)) \leq \deg_w(\mathbf{A}) - t$ or
  - $\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t)$ has all leading coordinates in distinct rows,

  where $\deg_w(\mathbf{A}) = \sum_i \deg_w(\mathbf{A}_i)$.
- Observation:

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\mathbf{A}, \lceil t/2 \rceil) \cdot \mathbf{U}(\mathbf{A}', t - d),$$

  where $\mathbf{A}' = \mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t/2)$ and $d = \deg_w \mathbf{A} - \deg_w \mathbf{A}'$.
- Leads to divide and conquer algorithm. Handle base case $\mathbf{U}(\mathbf{A}, 1)$ by Gaussian elimination.

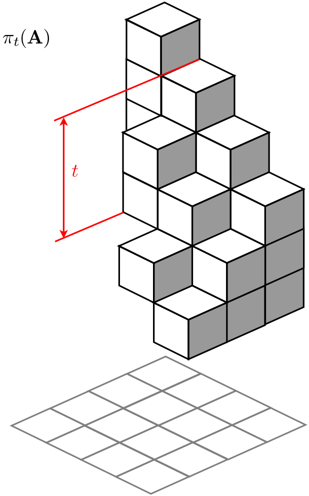
Algorithm: Divide and conquer



- Subproblems are easy:

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\pi_t(\mathbf{A}), t).$$

Algorithm: Divide and conquer

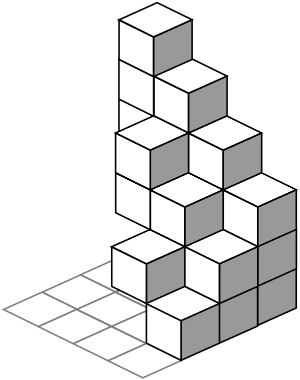


- Subproblems are easy:

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\pi_t(\mathbf{A}), t).$$

Algorithm: Divide and conquer

$\pi_t(\mathbf{A})$

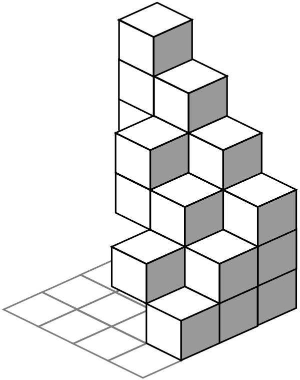


- Subproblems are easy:

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\pi_t(\mathbf{A}), t).$$

Algorithm: Divide and conquer

$\pi_t(\mathbf{A})$



- Subproblems are easy:

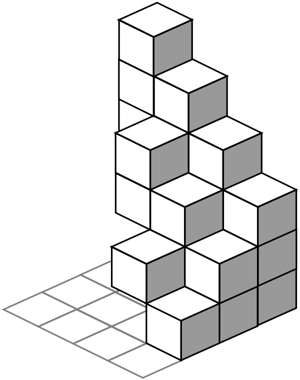
$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\pi_t(\mathbf{A}), t).$$

- Combining subproblems is easy:

  Entries in $\mathbf{U}(\mathbf{A}, t)$ have at most $2t$ coefficients.

Algorithm: Divide and conquer

$\pi_t(\mathbf{A})$



- Subproblems are easy:

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\pi_t(\mathbf{A}), t).$$

- Combining subproblems is easy:

  Entries in $\mathbf{U}(\mathbf{A}, t)$ have at most $2t$ coefficients.

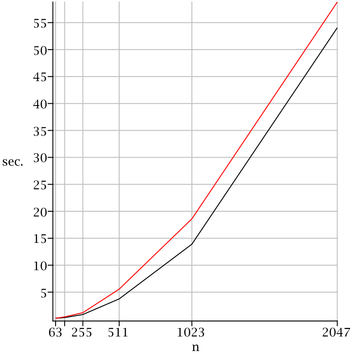
- Leads to algorithm requiring

$$\mathcal{O}\left(\ell^5 n \log^2(\ell n) \log\log(\ell n)\right)$$

$\mathbb{F}_q$-multiplications.

Comparison and conclusions

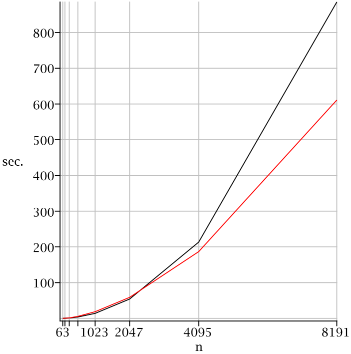
- The divide and conquer algorithm is asymptotically faster than Gaussian elimination.



Gaussian elimination
$\mathcal{O}\left(\ell^5 n^2\right)$
Divide and conquer
$\mathcal{O}\left(\ell^5 n \log^2(\ell n) \log\log(\ell n)\right)$

Comparison and conclusions

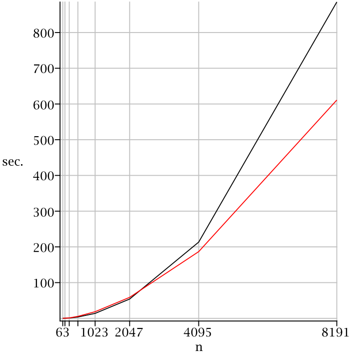
- The divide and conquer algorithm is asymptotically faster than Gaussian elimination.



Gaussian elimination
$\mathcal{O}\left(\ell^5 n^2\right)$
Divide and conquer
$\mathcal{O}\left(\ell^5 n \log^2(\ell n) \log\log(\ell n)\right)$

Comparison and conclusions

- The divide and conquer algorithm is asymptotically faster than Gaussian elimination.



Gaussian elimination
$\mathcal{O}\left(\ell^5 n^2\right)$
Divide and conquer
$\mathcal{O}\left(\ell^5 n \log^2(\ell n) \log\log(\ell n)\right)$

- The algorithm works in a more general setting: list decoding of certain algebraic geometry codes.

Contents

1 List decoding of error-correcting codes

2 Fast list decoding of Reed–Solomon codes

3 Fast list decoding of certain AG codes

4 Wu decoding of Reed–Solomon codes

AG codes

- $\mathfrak{C}$ a simple C_{ab} curve, i.e. a nonsingular affine curve given by a polynomial of the form $F(x_1, x_2) = 0$ such that

AG codes

- $\mathfrak{C}$ a simple C_{ab} curve, i.e. a nonsingular affine curve given by a polynomial of the form $F(x_1, x_2) = 0$ such that
  - The numbers $\gamma = \deg_{X_2} F$ and $\delta = \deg_{X_1} F$ are relatively prime.

AG codes

- $\mathfrak{C}$ a simple C_{ab} curve, i.e. a nonsingular affine curve given by a polynomial of the form $F(x_1, x_2) = 0$ such that
  - The numbers $\gamma = \deg_{X_2} F$ and $\delta = \deg_{X_1} F$ are relatively prime.
  - Any monomial $x_1^i x_2^j$ in the support of F satisfies $\gamma i + \delta j \leq \gamma\delta$.

AG codes

- $\mathfrak{C}$ a simple C_{ab} curve, i.e. a nonsingular affine curve given by a polynomial of the form $F(x_1, x_2) = 0$ such that
  - The numbers $\gamma = \deg_{X_2} F$ and $\delta = \deg_{X_1} F$ are relatively prime.
  - Any monomial $x_1^i x_2^j$ in the support of F satisfies $\gamma i + \delta j \leq \gamma\delta$.
- A simple C_{ab}-curve has a unique point at infinity denoted by P_∞.
- $v_{P_\infty}(x_1^i x_2^j) = -i\gamma - j\delta$.

AG codes

- $\mathfrak{C}$ a simple C_{ab} curve, i.e. a nonsingular affine curve given by a polynomial of the form $F(x_1, x_2) = 0$ such that
  - The numbers $\gamma = \deg_{X_2} F$ and $\delta = \deg_{X_1} F$ are relatively prime.
  - Any monomial $x_1^i x_2^j$ in the support of F satisfies $\gamma i + \delta j \leq \gamma\delta$.
- A simple C_{ab}-curve has a unique point at infinity denoted by P_∞.
- $v_{P_\infty}(x_1^i x_2^j) = -i\gamma - j\delta$.
- An AG code from a simple C_{ab}-curve of length n:

$$\mathcal{C} = \{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f(x) \in L(\mu P_\infty), v_{P_\infty}(f) + \mu \geq 0\},$$

Alphabet is $\Sigma = \mathbb{F}_q$ and $\alpha_1, \ldots, \alpha_n \in \mathfrak{C}(\mathbb{F}_q)$ are distinct affine points.

List decoding AG codes

- A list decoder must find $f(x) \in \mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$, with $v_{P_\infty}(f) + \mu \geq 0$, that passes through $n - \tau$ of the received points.

List decoding AG codes

- A list decoder must find $f(x) \in \mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$, with $v_{P_\infty}(f) + \mu \geq 0$, that passes through $n - \tau$ of the received points.

- Interpolate $Q(x_1, x_2, y)$ through received points, with multiplicity s.

List decoding AG codes

- A list decoder must find $f(x) \in \mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$, with $v_{P_\infty}(f) + \mu \geq 0$, that passes through $n - \tau$ of the received points.

- Interpolate $Q(x_1, x_2, y)$ through received points, with multiplicity s.
- ... of least weighted degree.

$$\deg_w(x_1^{i_1} x_2^{i_2} y^j) = i_1\gamma + i_2\delta + (k-1)j$$

List decoding AG codes

- A list decoder must find $f(x) \in \mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$, with $v_{P_\infty}(f) + \mu \geq 0$, that passes through $n - \tau$ of the received points.

- Interpolate $Q(x_1, x_2, y)$ through received points, with multiplicity s.
- ... of least weighted degree.

$$\deg_w(x_1^{i_1} x_2^{i_2} y^j) = i_1\gamma + i_2\delta + (k-1)j$$

- If $\tau/n < 1 - \sqrt{R}$ then

$$Q(x_1, x_2, f(x_1, x_2)) = 0$$

Translation of the interpolation problem

- The $\mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$–module of interpolation polynomials with $\deg_y(Q) \leq \ell$, is spanned by

$$\left\{E^s, E^{s-1}(y-R), \ldots, (y-R)^s, (y-R)^{s+1}, \ldots, (y-R)^{\ell}\right\}.$$

Translation of the interpolation problem

- The $\mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$–module of interpolation polynomials with $\deg_y(Q) \leq \ell$, is spanned by

$$\left\{E^s, E^{s-1}(y-R), \ldots, (y-R)^s, (y-R)^{s+1}, \ldots, (y-R)^\ell\right\}.$$

- E satisfies

$$(E) = \sum_{i=1}^{n} \alpha_i - nP_\infty$$

and $R(\alpha_i) = y_i$ for $1 \leq i \leq n$.

Translation of the interpolation problem

- The $\mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$–module of interpolation polynomials with $\deg_y(Q) \leq \ell$, is spanned by

$$\left\{E^s, E^{s-1}(y - R), \dots, (y - R)^s, (y - R)^{s+1}, \dots, (y - R)^\ell\right\}.$$

- E satisfies

$$(E) = \sum_{i=1}^{n} \alpha_i - nP_\infty$$

  and $R(\alpha_i) = y_i$ for $1 \leq i \leq n$.

- Find a generating set of the module viewed as $\mathbb{F}_q[x_1]$ module. One finds a generating set of cardinality $\gamma(\ell + 1)$.

Translation of the interpolation problem

- The $\mathbb{F}_q[x_1, x_2]/(F(x_1, x_2))$–module of interpolation polynomials with $\deg_y(Q) \leq \ell$, is spanned by

$$\left\{E^s, E^{s-1}(y-R), \ldots, (y-R)^s, (y-R)^{s+1}, \ldots, (y-R)^{\ell}\right\}.$$

- E satisfies

$$(E) = \sum_{i=1}^{n} \alpha_i - nP_\infty$$

and $R(\alpha_i) = y_i$ for $1 \leq i \leq n$.

- Find a generating set of the module viewed as $\mathbb{F}_q[x_1]$ module. One finds a generating set of cardinality $\gamma(\ell+1)$.
- Introduce matrix $\gamma(\ell+1) \times \gamma(\ell+1)$ matrix $\mathbf{A}$,

$$[\mathbf{A}]_{(ij),(i'j')} = \text{Coefficient to } x_2^i y^j \text{ in } (i', j')\text{-th basis function}$$

Algorithm: Divide and conquer

- Extend and generalize idea behind divide and conquer algorithm by Alekhnovich further.

Algorithm: Divide and conquer

- Extend and generalize idea behind divide and conquer algorithm by Alekhnovich further.
- Again leads to divide and conquer algorithm.

Algorithm: Divide and conquer

- Extend and generalize idea behind divide and conquer algorithm by Alekhnovich further.
- Again leads to divide and conquer algorithm.
- Leads to algorithm requiring

$$\mathcal{O}\left(\ell^5\gamma^3(n+\gamma\delta)\log^2(\ell(n+\gamma\delta))\log\log(\ell(n+\gamma\delta))\right)$$

$\mathbb{F}_q$-multiplications.

Algorithm: Divide and conquer

- Extend and generalize idea behind divide and conquer algorithm by Alekhnovich further.
- Again leads to divide and conquer algorithm.
- Leads to algorithm requiring
$$\mathcal{O}\left(\ell^5\gamma^3(n+\gamma\delta)\log^2(\ell(n+\gamma\delta))\log\log(\ell(n+\gamma\delta))\right)$$
$\mathbb{F}_q$-multiplications.
- For the well-known Hermitian curve one can list-decode one-point AG codes in
$$\mathcal{O}\left(\ell^5 n^2\log^2(\ell n)\log\log(\ell n)\right)$$
$\mathbb{F}_{q^2}$-multiplications. Note that in this case $\gamma = q$, $\delta = q+1$ and $n = q^3$.

Contents

1. List decoding of error-correcting codes
2. Fast list decoding of Reed–Solomon codes
3. Fast list decoding of certain AG codes
4. Wu decoding of Reed–Solomon codes

The key equation for RS codes.

- Sudan's algorithm for $\ell = 1$: find

$$Q(x, y) = q_1(x)y + q_0(x)$$

such that $Q(\alpha_i, y_i) = 0$.

The key equation for RS codes.

- Sudan's algorithm for $\ell = 1$: find

$$Q(x, y) = q_1(x)y + q_0(x)$$

  such that $Q(\alpha_i, y_i) = 0$.

- Leads to a (Gao) key equation

$$q_1(x)R(x) \equiv -q_0(x) \mod E(x),$$

The key equation for RS codes.

- Sudan's algorithm for $\ell = 1$: find

$$Q(x, y) = q_1(x)y + q_0(x)$$

  such that $Q(\alpha_i, y_i) = 0$.

- Leads to a (Gao) key equation

$$q_1(x)R(x) \equiv -q_0(x) \mod E(x),$$

- Which implies the standard key equation

$$\Lambda(X)S(X) \equiv \Omega(x) \mod x^{n-k}.$$

The key equation for RS codes.

- Sudan's algorithm for $\ell = 1$: find

$$Q(x, y) = q_1(x)y + q_0(x)$$

such that $Q(\alpha_i, y_i) = 0$.

- Leads to a (Gao) key equation

$$q_1(x)R(x) \equiv -q_0(x) \mod E(x),$$

- Which implies the standard key equation

$$\Lambda(X)S(X) \equiv \Omega(x) \mod x^{n-k}.$$

- Solving the key equation: use EEA on $S(x)$ and x^{n-k}. Finds $\Lambda(x)$ and $\Omega(x)$ if $2\tau < n - k + 1$.

The Wu list decoder.

- The Wu list decoder focuses on finding all relevant pairs $(\Lambda(x), \Omega(x))$ if $2\tau \geq n - k + 1$.

The Wu list decoder.

- The Wu list decoder focuses on finding all relevant pairs $(\Lambda(x), \Omega(x))$ if $2\tau \geq n-k+1$.
- Idea: working in the $\mathbb{F}_q[x]$-module generated by $y - S(x)$ and x^{n-k} we have

$$\Omega(x) - \Lambda(x)y = f_1(x)h_1(x,y) + f_2(x)h_2(x,y).$$

The Wu list decoder.

- The Wu list decoder focuses on finding all relevant pairs $(\Lambda(x), \Omega(x))$ if $2\tau \geq n - k + 1$.
- Idea: working in the $\mathbb{F}_q[x]$-module generated by $y - S(x)$ and x^{n-k} we have

$$\Omega(x) - \Lambda(x)y = f_1(x)h_1(x, y) + f_2(x)h_2(x, y).$$

- $h_1(x, y)$ and $h_2(x, y)$ are the output of (essentially) EEA on $y - S(x)$ and x^{n-k}.

The Wu list decoder.

- The Wu list decoder focuses on finding all relevant pairs $(\Lambda(x), \Omega(x))$ if $2\tau \geq n - k + 1$.
- Idea: working in the $\mathbb{F}_q[x]$-module generated by $y - S(x)$ and x^{n-k} we have

$$\Omega(x) - \Lambda(x)y = f_1(x)h_1(x, y) + f_2(x)h_2(x, y).$$

- $h_1(x, y)$ and $h_2(x, y)$ are the output of (essentially) EEA on $y - S(x)$ and x^{n-k}.
- $f_1(x)$ and $f_2(x)$ are unknown polynomials, but upper bounds on their degrees are known.

The Wu list decoder.

As before: $\Omega(x) - \Lambda(x)y = f_1(x)h_1(x,y) + f_2(x)h_2(x,y)$.

The Wu list decoder.

As before: $\Omega(x) - \Lambda(x)y = f_1(x)h_1(x,y) + f_2(x)h_2(x,y)$.

- For error positions $x = \alpha$ we can determine the ratio between $f_1(\alpha)$ and $f_2(\alpha)$.

The Wu list decoder.

As before: $\Omega(x) - \Lambda(x)y = f_1(x)h_1(x,y) + f_2(x)h_2(x,y)$.

- For error positions $x = \alpha$ we can determine the ratio between $f_1(\alpha)$ and $f_2(\alpha)$.
- $f_1(x)$ and $f_2(x)$ can be determined solving a rational interpolation problem (if τ is not too big).

The Wu list decoder.

As before: $\Omega(x) - \Lambda(x)y = f_1(x)h_1(x,y) + f_2(x)h_2(x,y)$.

- For error positions $x = \alpha$ we can determine the ratio between $f_1(\alpha)$ and $f_2(\alpha)$.
- $f_1(x)$ and $f_2(x)$ can be determined solving a rational interpolation problem (if τ is not too big).
- Wu's list decoder can correct (generalized) Reed–Solomon codes up to $\tau < n - \sqrt{n(n-d)}$ errors.

The Wu list decoder.

As before: $\Omega(x) - \Lambda(x)y = f_1(x)h_1(x,y) + f_2(x)h_2(x,y)$.

- For error positions $x = \alpha$ we can determine the ratio between $f_1(\alpha)$ and $f_2(\alpha)$.
- $f_1(x)$ and $f_2(x)$ can be determined solving a rational interpolation problem (if τ is not too big).
- Wu's list decoder can correct (generalized) Reed–Solomon codes up to $\tau < n - \sqrt{n(n-d)}$ errors.
- Work in progress: apply Wu's list decoder to other classes of codes.

Thank you for your attention!