Interpolation and List Decoding of Algebraic Codes

Peter Beelen and Kristian Brander

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 Conclusions
5 Future work
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 Conclusions
5 Future work
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, $\tau$, is less than $\frac{d}{2}$ then there is at most one codeword within distance $\tau$ from any received word $\mathbf{y}$. 
List decoding

- If $\tau \geq \frac{d}{2}$ there might be a “small” list of codewords within distance $\tau$ from $y$.
- 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 $\tau$ from $y$.
- The decoder thus get a list of candidate messages.
- We require the lists to be \textit{polynomially bounded} in the code length $n$. 
The rate of an error-correcting code is $rate \ R = \frac{\log |\Sigma|(|C|)}{n}$.
Error-correcting codes and list decoding

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

- The relative number of errors it can correct is denoted by $\frac{\tau}{n}$.

- Capacity:
  \[
  \frac{\tau}{n} < 1 - R.
  \]
The rate of an error-correcting code is \( \text{rate } R = \frac{\log |\Sigma|(|C|)}{n} \).

The relative number of errors it can correct is denoted by \( \frac{\tau}{n} \).

- **Capacity:**
  \( \frac{\tau}{n} < 1 - R \).

- **Unique decoding:**
  \( \frac{\tau}{n} < \frac{1}{2}(1 - R) \).
The rate of an error-correcting code is rate $R = \frac{\log |\Sigma|(|C|)}{n}$.

The relative number of errors it can correct is denoted by $\frac{\tau}{n}$.

Capacity:
$\frac{\tau}{n} < 1 - R$.

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

Guruswami–Sudan algorithm:
$\frac{\tau}{n} < 1 - \sqrt{R}$. 
The rate of an error-correcting code is \( \text{rate } R = \frac{\log|\Sigma|(|C|)}{n} \).

The relative number of errors it can correct is denoted by \( \frac{\tau}{n} \).

Capacity:
\( \frac{\tau}{n} < 1 - R \).

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

Guruswami–Sudan algorithm:
\( \frac{\tau}{n} < 1 - \sqrt{R} \).

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 Conclusions
5 Future work
Reed–Solomon codes

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

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

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

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

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.
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 \( s \).
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 \( \deg_y(Q) \leq \ell \), 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 \( 1 \leq i \leq n \).
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 $A$,

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

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

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

- For $s = 2$ and $\ell = 3$, 
  \[
  A = \begin{bmatrix}
  E^2 & -ER & R^2 & -R^3 \\
  0 & E & -2R & 3R^2 \\
  0 & 0 & 1 & -3R \\
  0 & 0 & 0 & 1
  \end{bmatrix}.
  \]

- 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:

\[
\deg_w(A_{i,j}) = \\
\deg(A_{i,j}) + (k - 1)j.
\]
Algorithm: Gaussian elimination

- Represent matrix as grid.
- Represent \((i,j)\)-th entry by stack of cubes:
  \[
  \deg_w(A_{i,j}) = \deg(A_{i,j}) + (k - 1)j.
  \]
- Gaussian elimination.
Algorithm: Gaussian elimination

- Represent matrix as grid.
- Represent \((i, j)\)-th entry by stack of cubes:
  \[
  \deg_w(A_{i,j}) = \deg(A_{i,j}) + (k - 1)j.
  \]
- Gaussian elimination.
Algorithm: Gaussian elimination

- Represent matrix as grid.
- Represent \((i, j)\)-th entry by stack of cubes:
  \[
  \deg_w(A_{i,j}) = \deg(A_{i,j}) + (k - 1)j.
  \]
- Gaussian elimination.
Algorithm: Gaussian elimination

- Represent matrix as grid.
- Represent \((i,j)\)-th entry by stack of cubes:
  
  \[
  \deg_w(A_{i,j}) = \deg(A_{i,j}) + (k - 1)j.
  \]

- Gaussian elimination.
Algorithm: Gaussian elimination

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

  \[
  \deg_w(A_{i,j}) = \deg(A_{i,j}) + (k - 1)j.
  \]

- Gaussian elimination.
Algorithm: Gaussian elimination

- Represent matrix as grid.
- Represent \((i, j)\)-th entry by stack of cubes:
  \[
  \text{deg}_w(A_{i,j}) = \text{deg}(A_{i,j}) + (k - 1)j.
  \]
- Gaussian elimination.
Algorithm: Gaussian elimination

- Represent matrix as grid.
- Represent $(i,j)$-th entry by stack of cubes:
  \[
  \deg_w(A_{i,j}) = \deg(A_{i,j}) + (k - 1)j.
  \]
- 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:
  \[
  \deg_w(A_{i,j}) = \deg(A_{i,j}) + (k - 1)j.
  \]
- Gaussian elimination.
- Continue the process, until leading coordinates occur in distinct rows.
- Leads to algorithm requiring \(\mathcal{O}(\ell^5 n^2)\) \(\mathbb{F}_q\)-multiplications.
Algorithm: Divide and conquer

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

where $\text{deg}_w(A) = \sum_i \text{deg}_w(A_i)$. 

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

Introduce matrix $U(A, t)$ representing the column operations made when “cutting down” the stack, i.e.

- $\deg_w(A \cdot U(A, t)) \leq \deg_w(A) - t$ or
- $A \cdot U(A, t)$ has all leading coordinates in distinct rows,

where $\deg_w(A) = \sum_i \deg_w(A_i)$.

Observation:

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

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

Algorithm: Divide and conquer

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

where $\deg_w(A) = \sum_i \deg_w(A_i)$.

- Observation:
  \[
  U(A, t) = U(A, \lceil t/2 \rceil) \cdot U(A', t - d),
  \]
  where $A' = U(A, t/2)$ and $d = \deg_w A - \deg_w A'$.

- Leads to divide and conquer algorithm. Handle base case $U(A, 1)$ by Gaussian elimination.
Algorithm: Divide and conquer

Subproblems are easy:

\[ U(A, t) = U(\pi_t(A), t). \]
Algorithm: Divide and conquer

Subproblems are easy:

\[ U(A, t) = U(\pi_t(A), t). \]
Algorithm: Divide and conquer

\[ \pi_t(A) \]

- Subproblems are easy:
  \[ U(A, t) = U(\pi_t(A), t). \]
Algorithm: Divide and conquer

\[ \pi_t(A) \]

- Subproblems are easy:
  \[ U(A, t) = U(\pi_t(A), t). \]

- Combining subproblems is easy:
  Entries in \( U(A, t) \) have at most \( 2t \) coefficients.
Algorithm: Divide and conquer

Subproblems are easy:

$$U(A, t) = U(\pi_t(A), t).$$

Combining subproblems is easy:

Entries in $U(A, t)$ have at most $2t$ coefficients.

Leads to algorithm requiring

$$O(\ell^5 n \log^2(\ell n) \log \log(\ell n))$$

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

\[ \mathcal{O} \left( \ell^5 n^2 \right) \]

Gaussian elimination

\[ \mathcal{O} \left( \ell^5 n \log^2(\ell n) \log \log(\ell n) \right) \]
Comparison and conclusions

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

Gaussian elimination
\[ O(\ell^5 n^2) \]

Divide and conquer
\[ O(\ell^5 n \log^2(\ell n) \log \log(\ell n)) \]
Comparison and conclusions

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

\[ \mathcal{O}(\ell^5 n^2) \]

\[ \mathcal{O}(\ell^5 n \log^2(\ell n) \log \log(\ell n)) \]

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

- \( \mathcal{C} \) 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

- \( C \) 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

- 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

- \( C \) 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 \( P_\infty \).
- \( \nu_{P_\infty}(x_1^i x_2^j) = -i\gamma - j\delta \).
AG codes

- $\mathcal{C}$ 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 $P_\infty$.
- $\nu_{P_\infty}(x_1^i x_2^j) = -i \gamma - j \delta$.
- An AG code from a simple $C_{ab}$-curve of length $n$:

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

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

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

Interpolate \( Q(x_1, x_2, y) \) through received points, with multiplicity \( s \).
A list decoder must find \( f(x_1, x_2) \in \mathbb{F}_q[x_1, x_2]/(F(x_1, x_2)) \), with \( \nu_{P_\infty}(f) + \mu \geq 0 \), 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.

\[
\deg_w(x_1^{i_1} x_2^{i_2} y^j) = i_1 \gamma + i_2 \delta + \mu j
\]
List decoding AG codes

- A list decoder must find \( f(x_1, x_2) \in \mathbb{F}_q[x_1, x_2]/(F(x_1, x_2)) \), with \( \nu_{P_\infty}(f) + \mu \geq 0 \), 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.

\[
\deg_w(x_1^{i_1} x_2^{i_2} y^j) = i_1 \gamma + i_2 \delta + \mu j
\]

- 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 $\text{deg}_y(Q) \leq \ell$, 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

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

- $E$ 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 $\gamma(\ell + 1)$. 
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

  \[ \{ E^s, E^{s-1}(y - R), \ldots, (y - R)^s, (y - R)^{s+1}, \ldots, (y - R)\ell \} \].

- $E$ 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 $A$,

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

\[ O \left( \ell^5 \gamma^3 (n + \gamma \delta) \log^2(\ell(n + \gamma \delta)) \log \log(\ell(n + \gamma \delta)) \right) \]

\( \mathbb{F}_q \)-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

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

$\mathbb{F}_q$-multiplications.

- For the well-known Hermitian curve one can list-decode one-point AG codes in

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

$\mathbb{F}_{q^2}$-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 Conclusions
5 Future work
List decoding may correct **twice as many errors** as unique decoding.
Conclusions

- List decoding may correct twice as many errors as unique decoding.
- Guruswami–Sudan algorithm
  \[ \frac{\tau}{n} < 1 - \sqrt{R} \]
Conclusions

- List decoding may correct twice as many errors as unique decoding.
- Guruswami–Sudan algorithm
  - $\tau/n < 1 - \sqrt{R}$
  - Reed–Solomon codes:
    \[ O(\ell^5 n \log^2(\ell n) \log \log(\ell n)) \].
Conclusions

- List decoding may correct twice as many errors as unique decoding.
- Guruswami–Sudan algorithm
  - $\tau/n < 1 - \sqrt{R}$
  - Reed–Solomon codes:
    $$\mathcal{O}(\ell^5 n \log^2(\ell n) \log \log(\ell n)).$$
  - Hermitian codes:
    $$\mathcal{O}(\ell^5 n^2 \log^2(\ell n) \log \log(\ell n)).$$
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. Conclusions
5. Future work
Future work

- Extend the decoder to a more general class of AG codes.
Future work

- Extend the decoder to a more general class of AG codes.
- Improve list-decoding complexity further.
Future work

- Extend the decoder to a more general class of AG codes.
- Improve list-decoding complexity further.
- Get closer to capacity $1 - R$. 
Thank you for your attention ...
Thank you for your attention ...

... and in fact your presence in spite of ash clouds and the like ...
Thank you for your attention ...

... and in fact your presence in spite of ash clouds and the like ...