Advanced code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

Advanced
code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest
nonzero vector in L?

$L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$

Advanced

code-based cryptography

Daniel J. Bernstein

University of Illinois at Chicago &

Technische Universiteit Eindhoven

Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$$
$$= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z}$$

Advanced
code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$$
$$= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z}$$
$$= (-1, 7)\mathbf{Z} + (3, 3)\mathbf{Z}$$

Advanced code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$ $= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z} \\ &= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z} \\ &= (-1, 7)\mathbf{Z} + (3, 3)\mathbf{Z} \\ &= (-4, 4)\mathbf{Z} + (3, 3)\mathbf{Z}. \end{aligned}$$

Advanced code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$
\begin{aligned}
L &= (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (3, 3)\mathbf{Z} \\
&= (-4, 4)\mathbf{Z} + (3, 3)\mathbf{Z}.
\end{aligned}
$$

$(-4, 4), (3, 3)$ are orthogonal.
Shortest vectors in L are $(0, 0)$, $(3, 3)$, $(-3, -3)$.

ed
sed cryptography

. Bernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest
nonzero vector in L?

$$L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$$
$$= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z}$$
$$= (-1, 7)\mathbf{Z} + (3, 3)\mathbf{Z}$$
$$= (-4, 4)\mathbf{Z} + (3, 3)\mathbf{Z}.$$

$(-4, 4), (3, 3)$ are orthogonal.
Shortest vectors in L are
$(0, 0)$, $(3, 3)$, $(-3, -3)$.

graphy

n

is at Chicago &

siteit Eindhoven

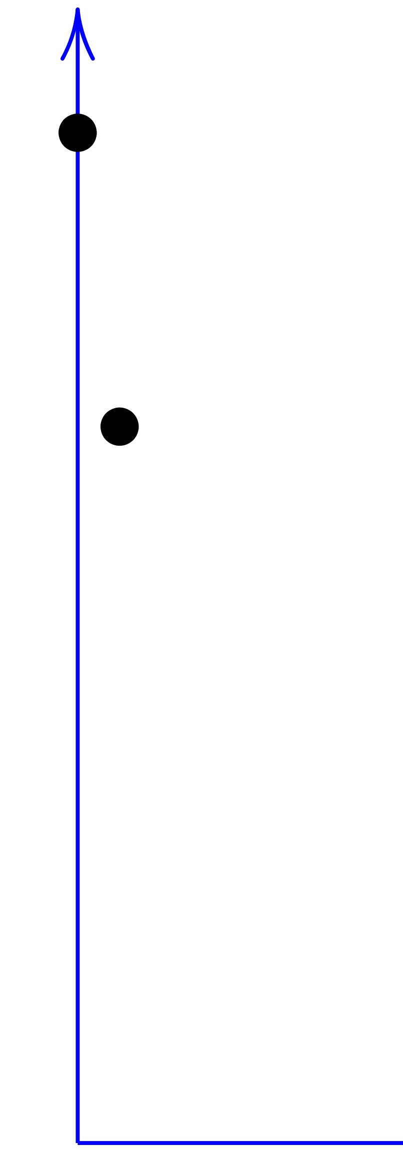
Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$
\begin{aligned}
L &= (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (3, 3)\mathbf{Z} \\
&= (-4, 4)\mathbf{Z} + (3, 3)\mathbf{Z}.
\end{aligned}
$$

$(-4, 4), (3, 3)$ are orthogonal.
Shortest vectors in L are $(0, 0)$, $(3, 3)$, $(-3, -3)$.



ago &
hoven

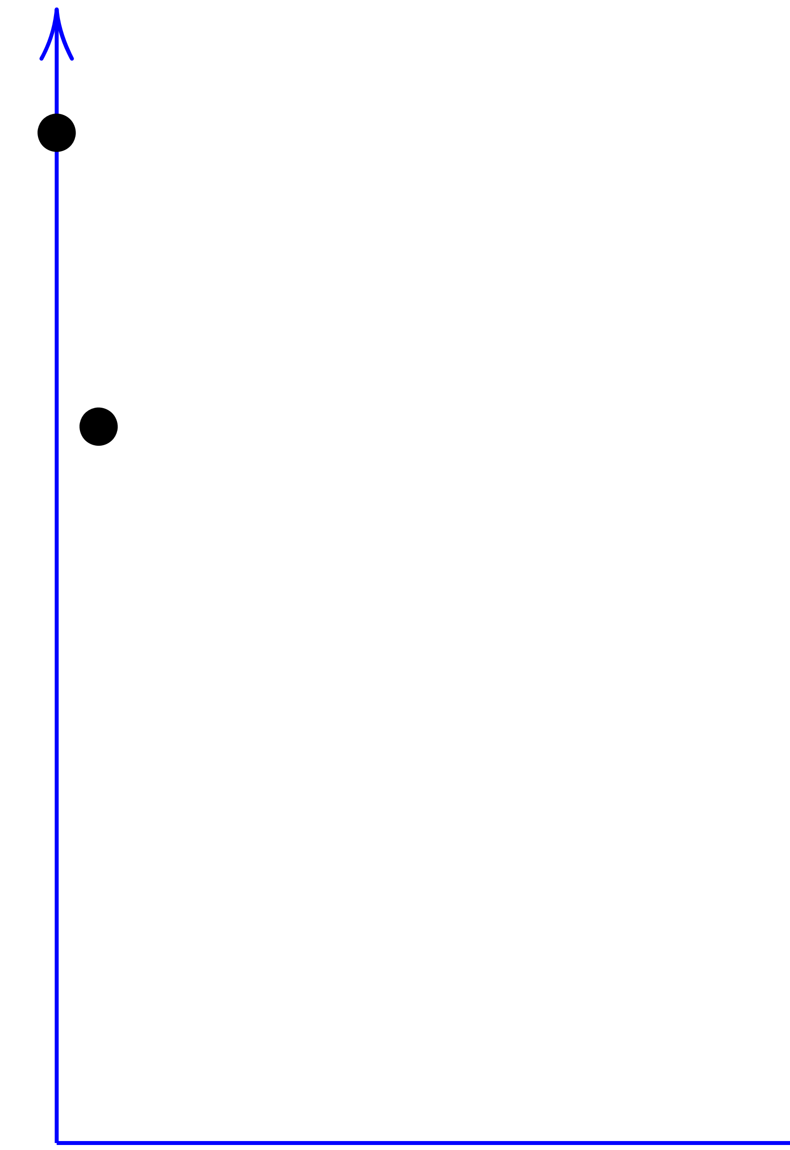
Lattice-basis reduction

Define $L = (0,24)\mathbf{Z} + (1,17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest
nonzero vector in L?

$$
\begin{aligned}
L &= (0,24)\mathbf{Z} + (1,17)\mathbf{Z} \\
&= (-1,7)\mathbf{Z} + (1,17)\mathbf{Z} \\
&= (-1,7)\mathbf{Z} + (3,3)\mathbf{Z} \\
&= (-4,4)\mathbf{Z} + (3,3)\mathbf{Z}.
\end{aligned}
$$

$(-4,4), (3,3)$ are orthogonal.
Shortest vectors in L are
$(0,0)$, $(3,3)$, $(-3,-3)$.



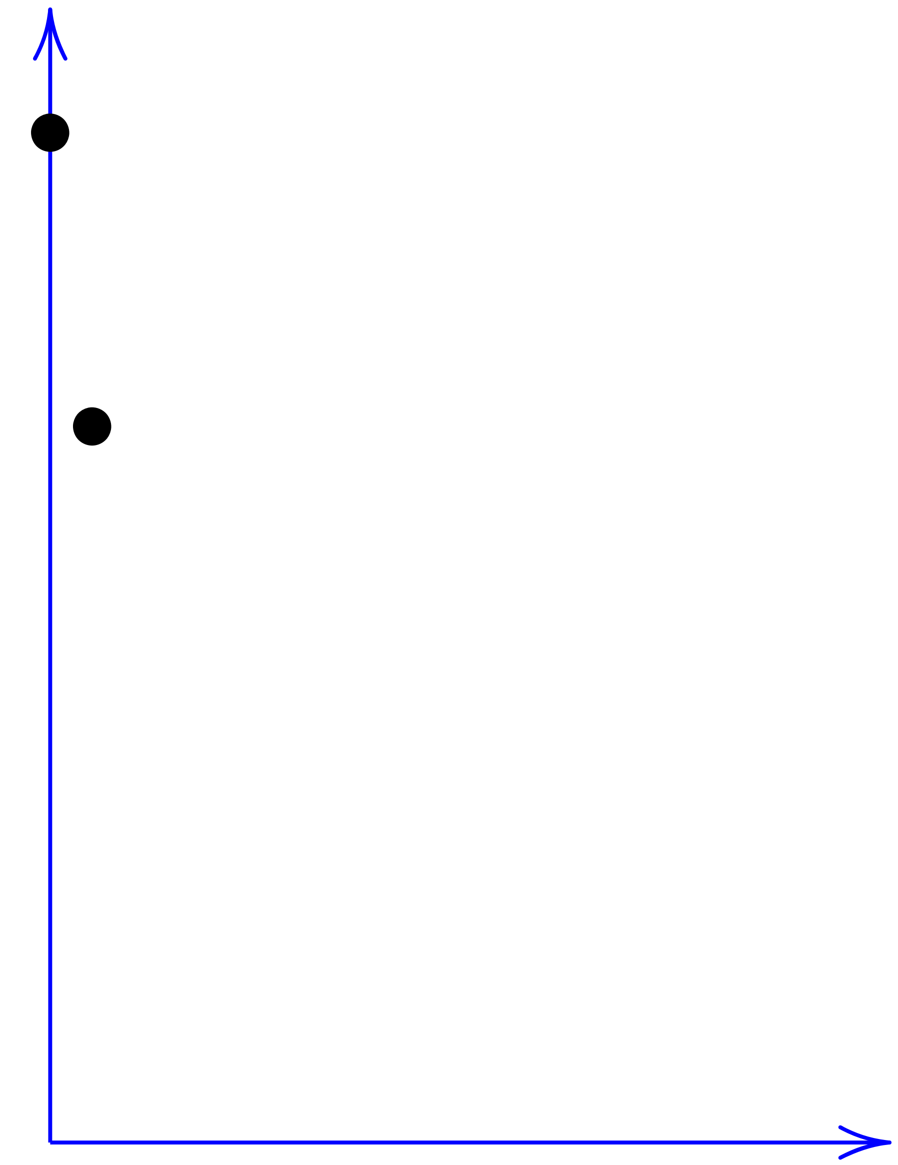
Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$
\begin{aligned}
L &= (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (3, 3)\mathbf{Z} \\
&= (-4, 4)\mathbf{Z} + (3, 3)\mathbf{Z}.
\end{aligned}
$$

$(-4, 4), (3, 3)$ are orthogonal.
Shortest vectors in L are $(0, 0)$, $(3, 3)$, $(-3, -3)$.



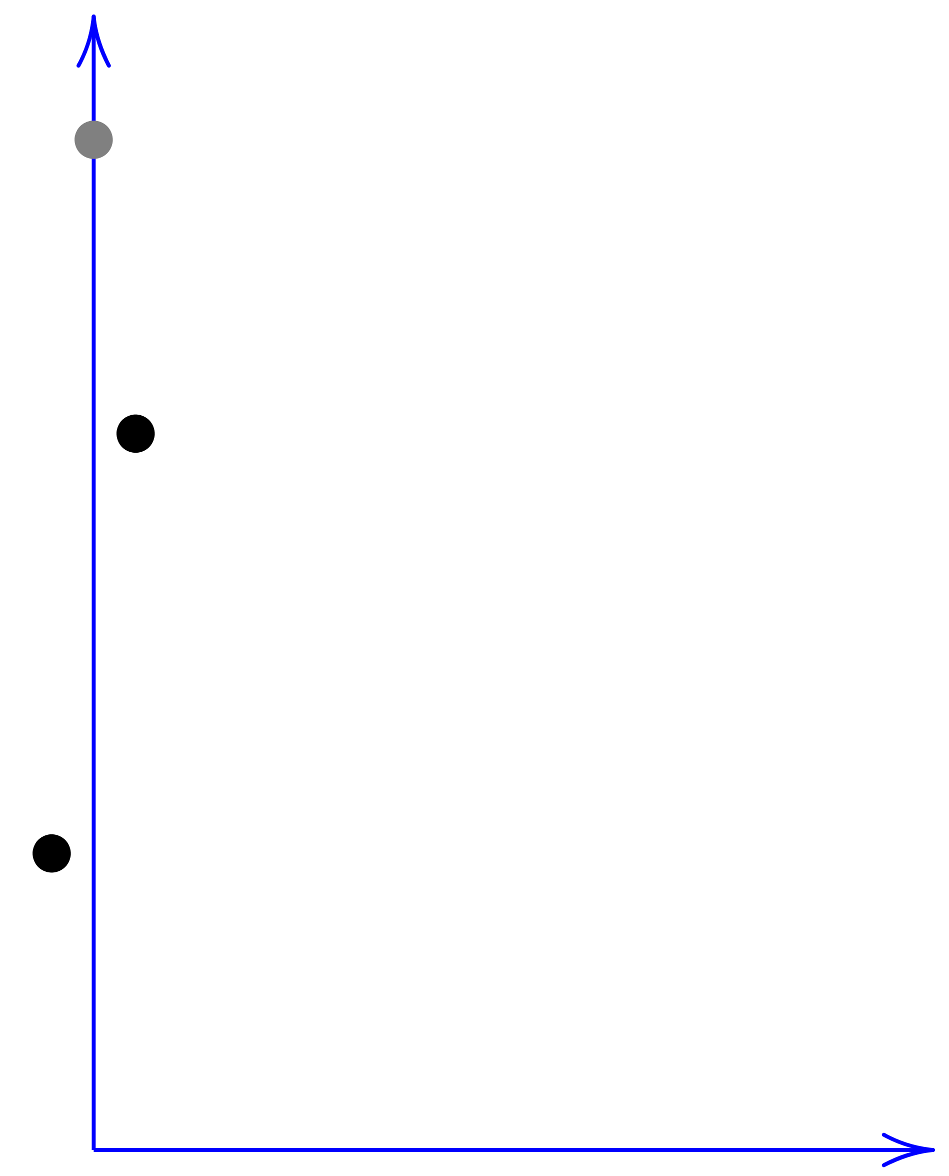
Lattice-basis reduction

Define $L = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$
\begin{aligned}
L &= (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 7)\mathbf{Z} + (3, 3)\mathbf{Z} \\
&= (-4, 4)\mathbf{Z} + (3, 3)\mathbf{Z}.
\end{aligned}
$$

$(-4, 4), (3, 3)$ are orthogonal.
Shortest vectors in L are $(0, 0)$, $(3, 3)$, $(-3, -3)$.



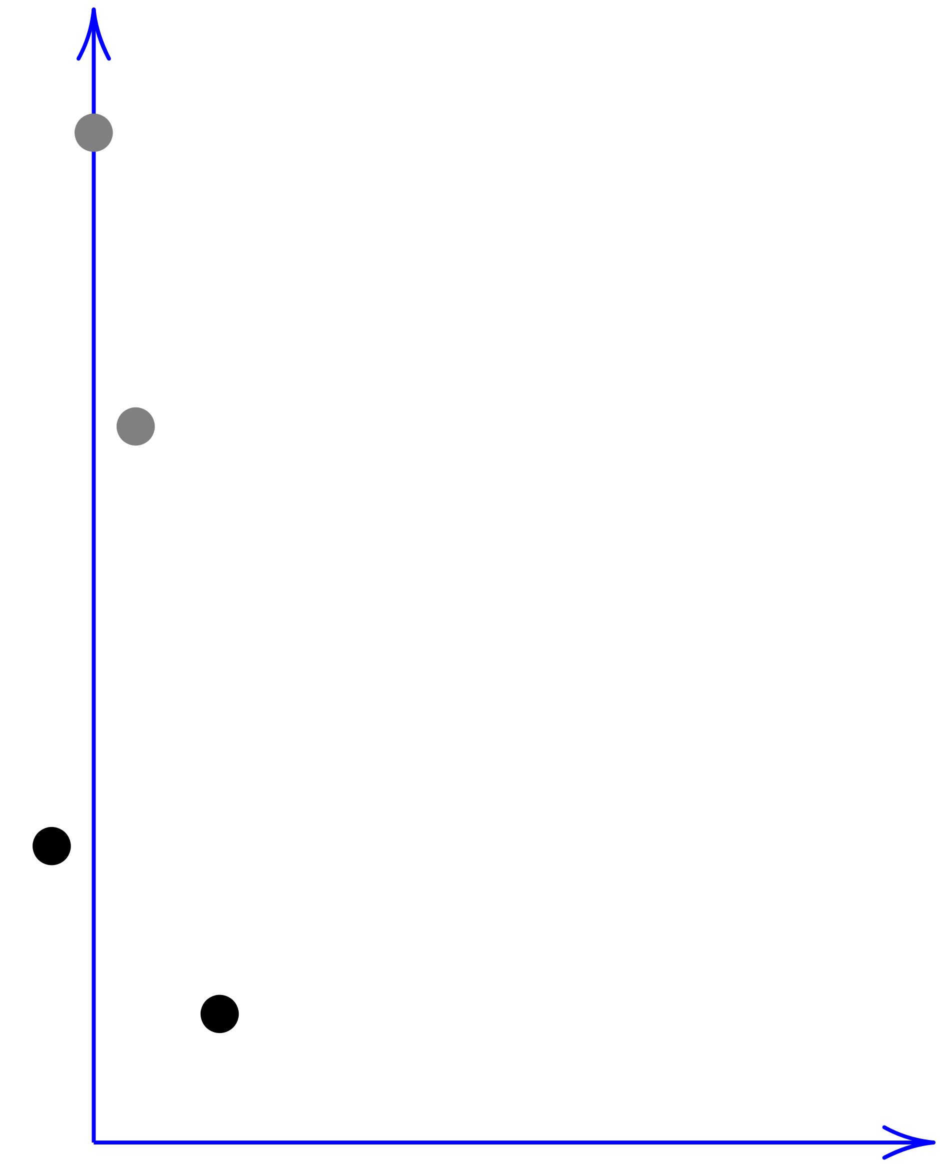
Lattice-basis reduction

Define $L = (0,24)\mathbf{Z} + (1,17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$L = (0,24)\mathbf{Z} + (1,17)\mathbf{Z}$$
$$= (-1,7)\mathbf{Z} + (1,17)\mathbf{Z}$$
$$= (-1,7)\mathbf{Z} + (3,3)\mathbf{Z}$$
$$= (-4,4)\mathbf{Z} + (3,3)\mathbf{Z}.$$

$(-4,4), (3,3)$ are orthogonal.
Shortest vectors in L are $(0,0)$, $(3,3)$, $(-3,-3)$.



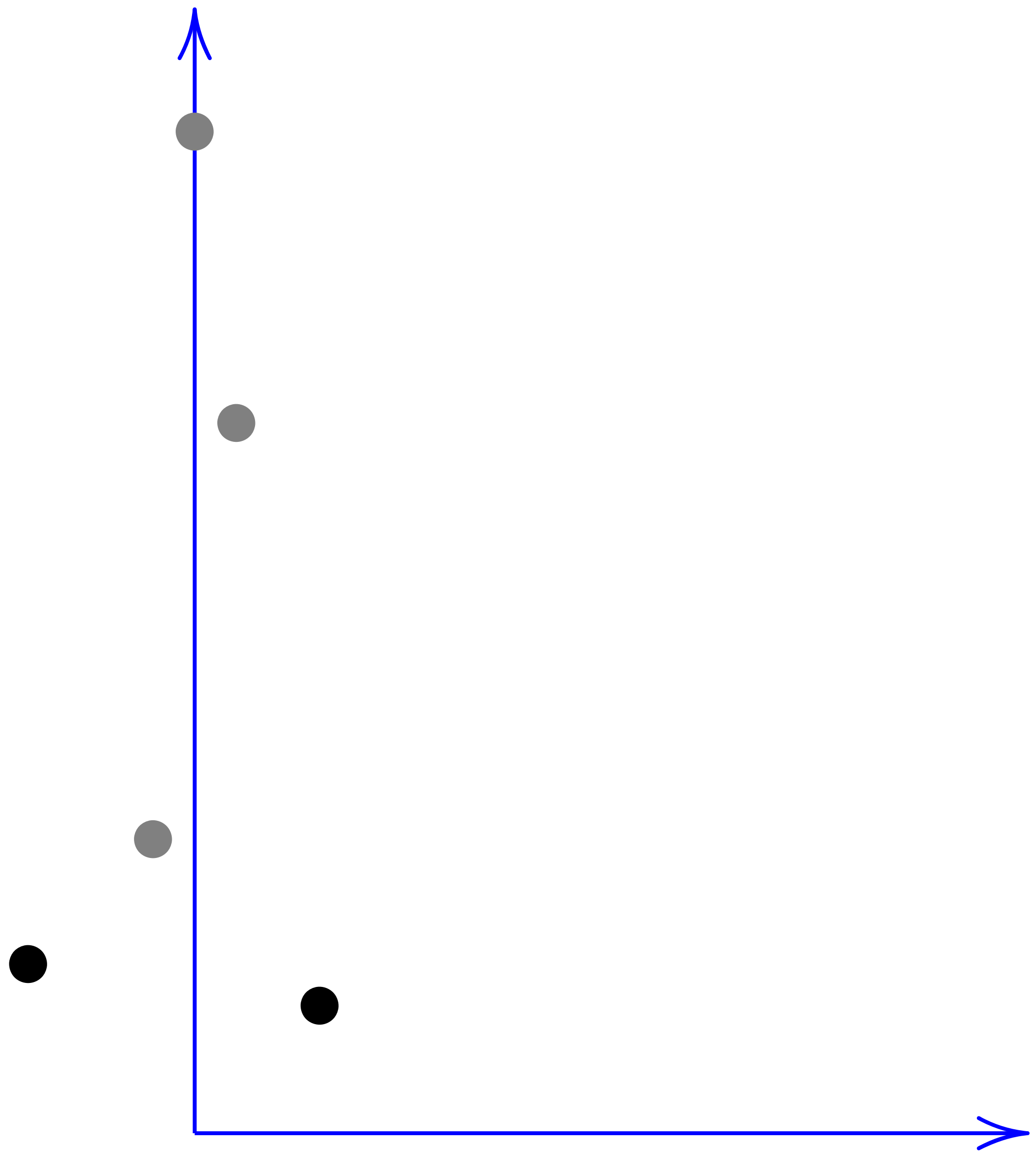
<u>Lattice-basis reduction</u>

Define $L = (0,24)\mathbf{Z} + (1,17)\mathbf{Z}$
$= \{(b, 24a + 17b) : a, b \in \mathbf{Z}\}$.

What is the shortest
nonzero vector in L?

$$\begin{aligned} L &= (0,24)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,7)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,7)\mathbf{Z} + (3,3)\mathbf{Z} \\ &= (-4,4)\mathbf{Z} + (3,3)\mathbf{Z}. \end{aligned}$$

$(-4,4), (3,3)$ are orthogonal.
Shortest vectors in L are
$(0,0)$, $(3,3)$, $(-3,-3)$.



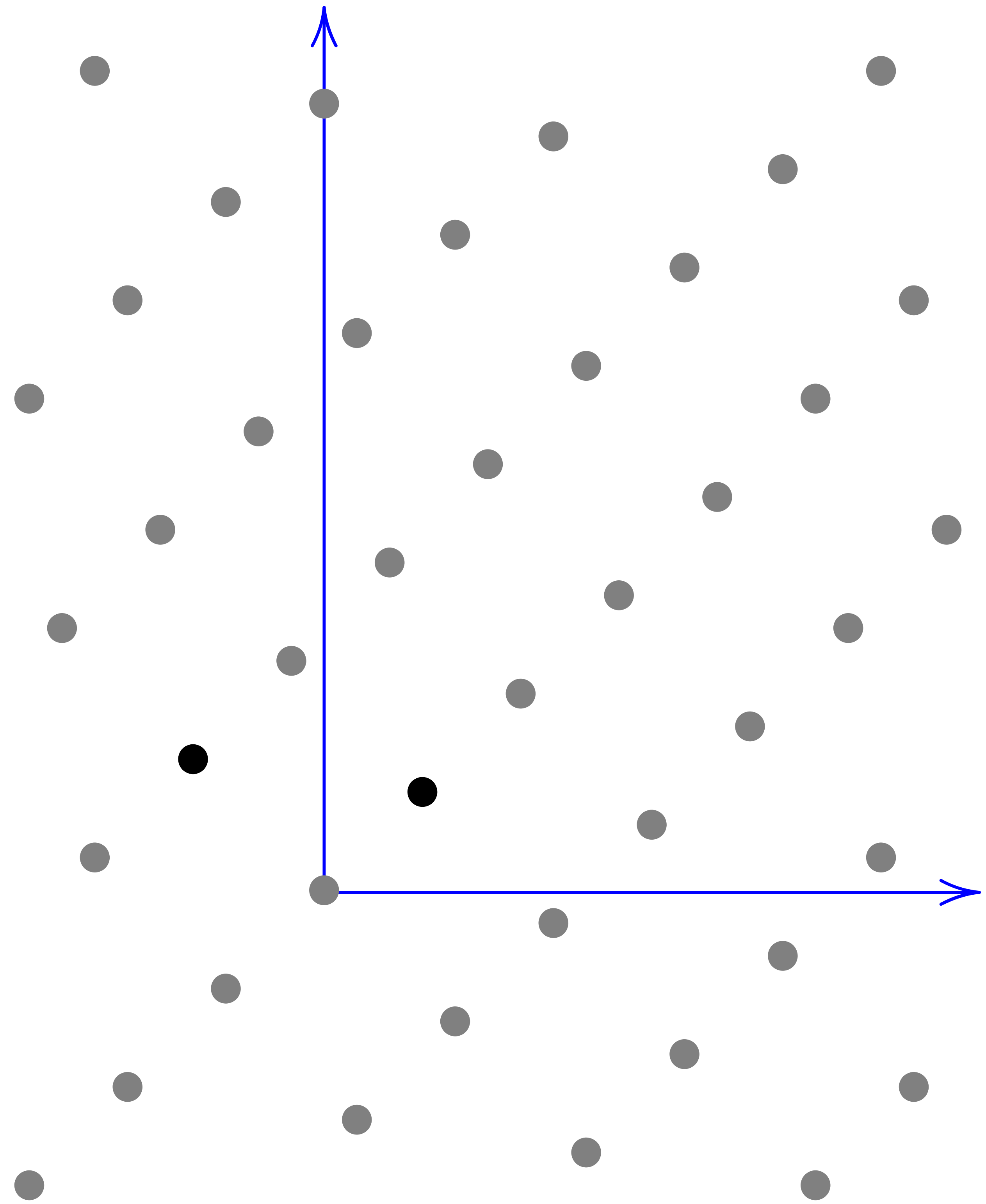
Lattice-basis reduction

Define $L = (0,24)\mathbf{Z} + (1,17)\mathbf{Z}$
$= \{(b, 24a+17b) : a, b \in \mathbf{Z}\}$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0,24)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,7)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,7)\mathbf{Z} + (3,3)\mathbf{Z} \\ &= (-4,4)\mathbf{Z} + (3,3)\mathbf{Z}. \end{aligned}$$

$(-4,4), (3,3)$ are orthogonal.
Shortest vectors in L are $(0,0)$, $(3,3)$, $(-3,-3)$.



asis reduction

$. = (0, 24)\mathbf{Z} + (1, 17)\mathbf{Z}$

$4a + 17b) : a, b \in \mathbf{Z}\}.$

the shortest

vector in L?

$24)\mathbf{Z} + (1, 17)\mathbf{Z}$

$., 7)\mathbf{Z} + (1, 17)\mathbf{Z}$

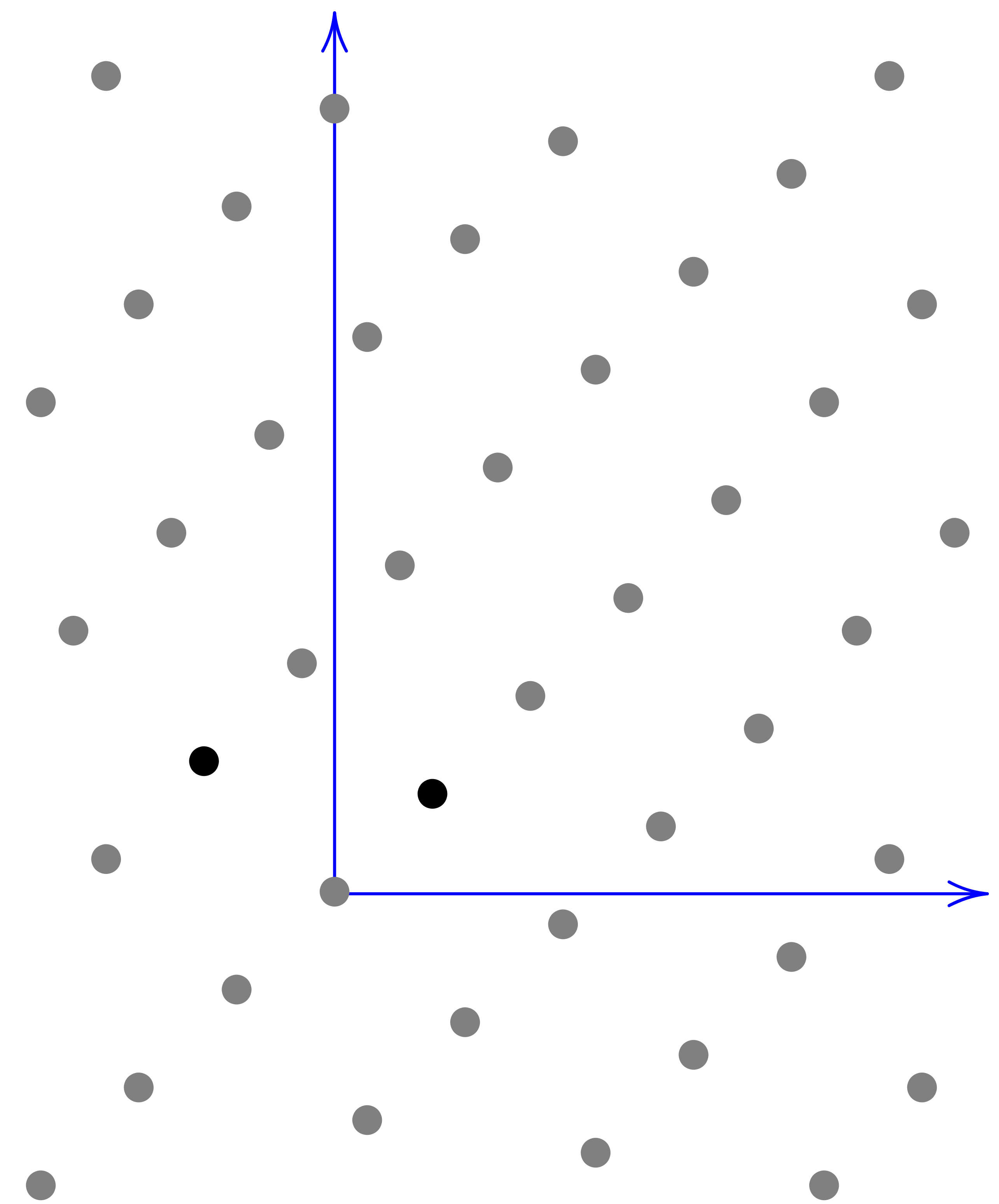
$., 7)\mathbf{Z} + (3, 3)\mathbf{Z}$

$4, 4)\mathbf{Z} + (3, 3)\mathbf{Z}.$

$(3, 3)$ are orthogonal.

vectors in L are

$3, 3), (-3, -3).$



Another

Define L

What is

nonzero

ction

$\mathbf{Z} + (1, 17)\mathbf{Z}$

$: a, b \in \mathbf{Z}\}$.

est

L?

$, 17)\mathbf{Z}$

$., 17)\mathbf{Z}$

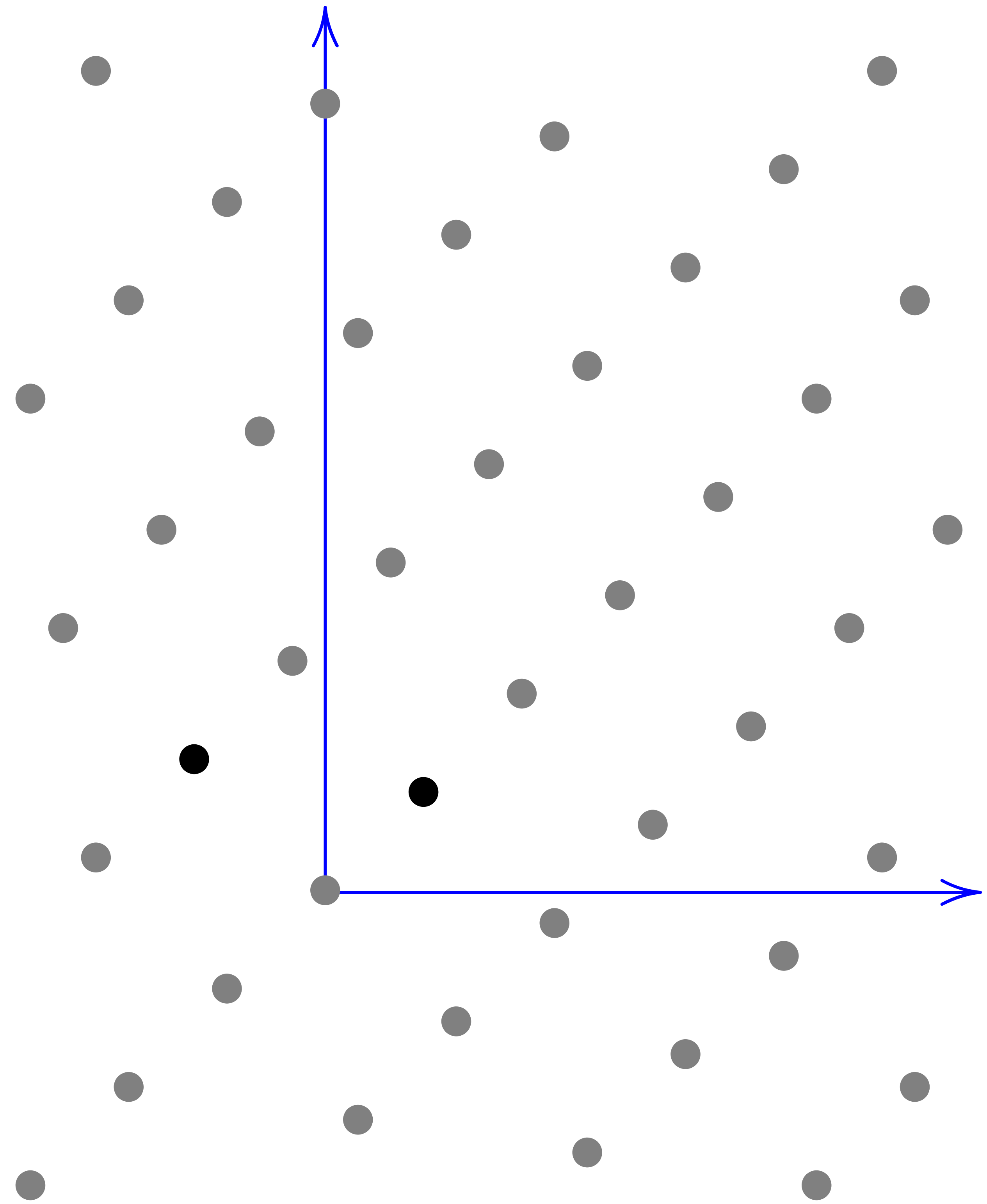
$3, 3)\mathbf{Z}$

$3, 3)\mathbf{Z}$.

orthogonal.

n L are

$-3)$.



Another example:

Define $L = (0, 25)$

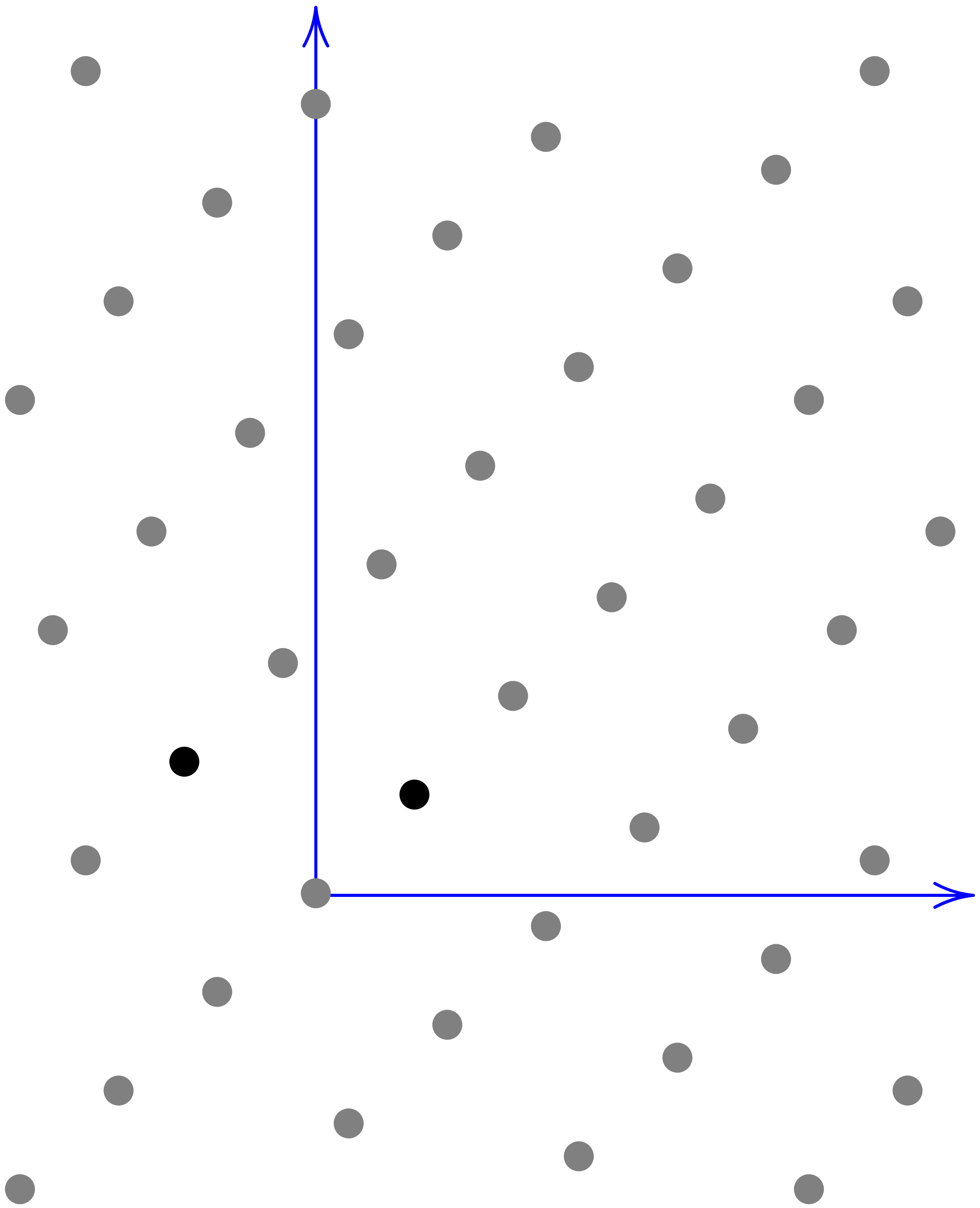
What is the shorte

nonzero vector in

7)**Z**

}.

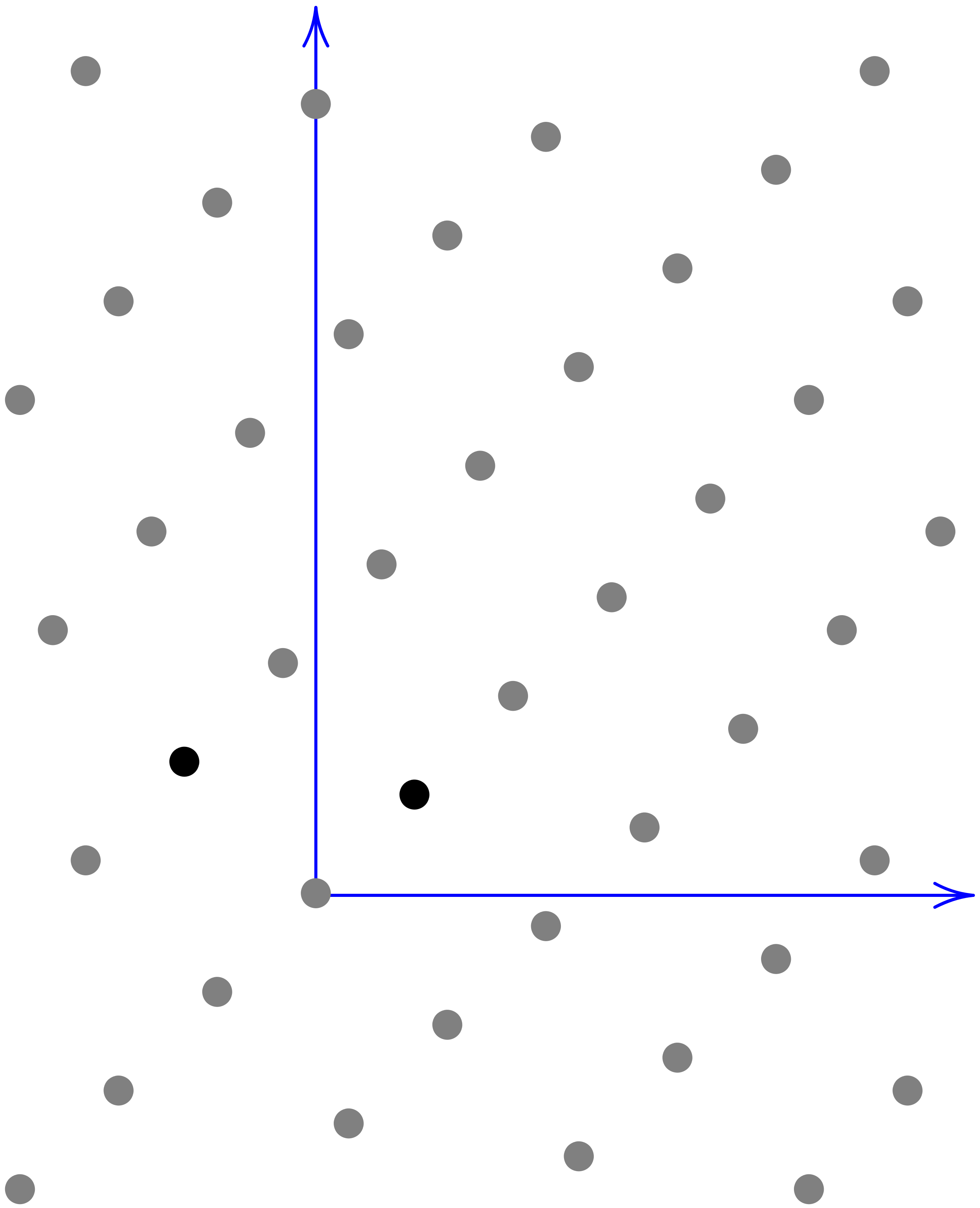
l.



Another example:

Define $L = (0, 25)\mathbf{Z} + (1, 17$

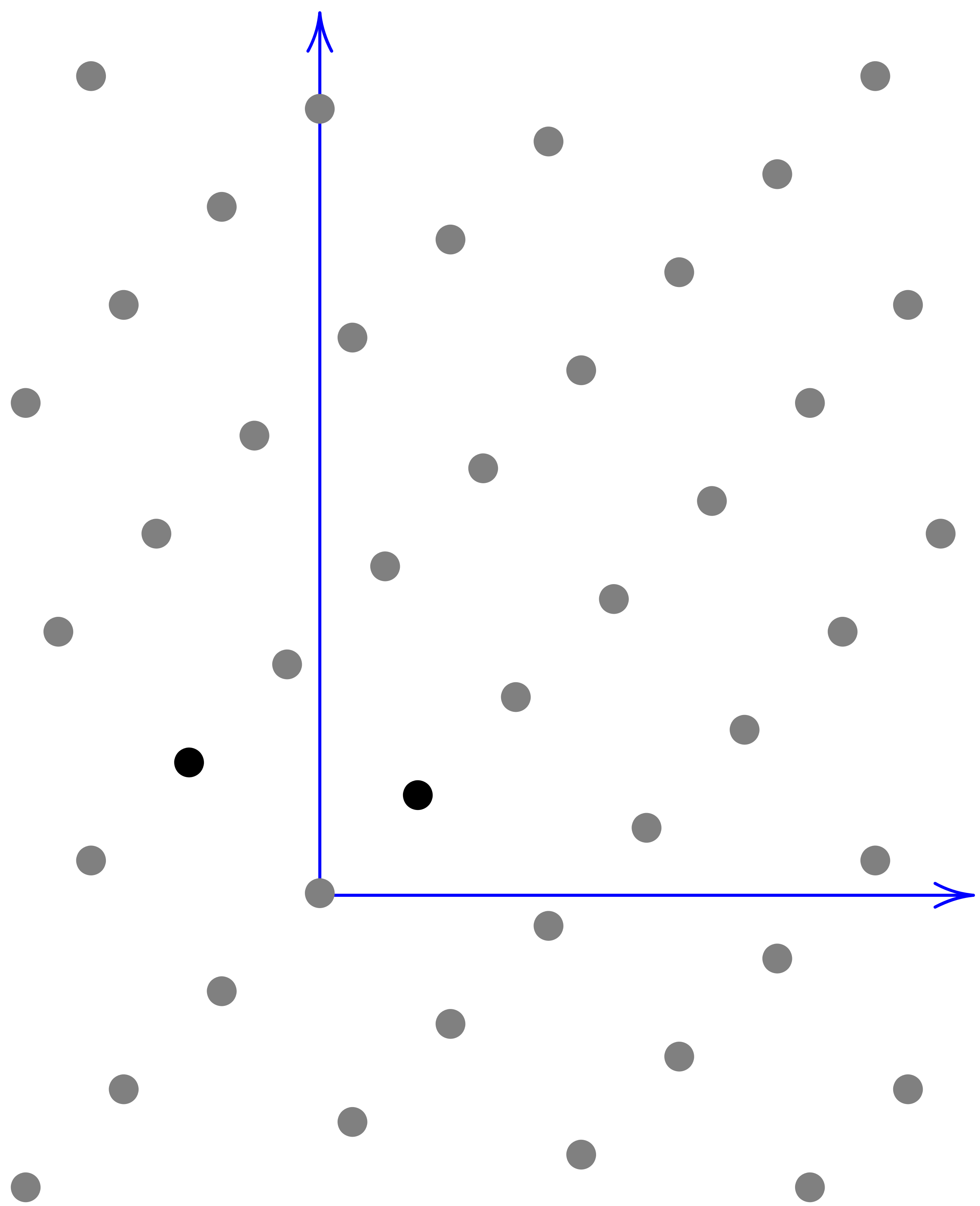
What is the shortest nonzero vector in L?



Another example:

Define $L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

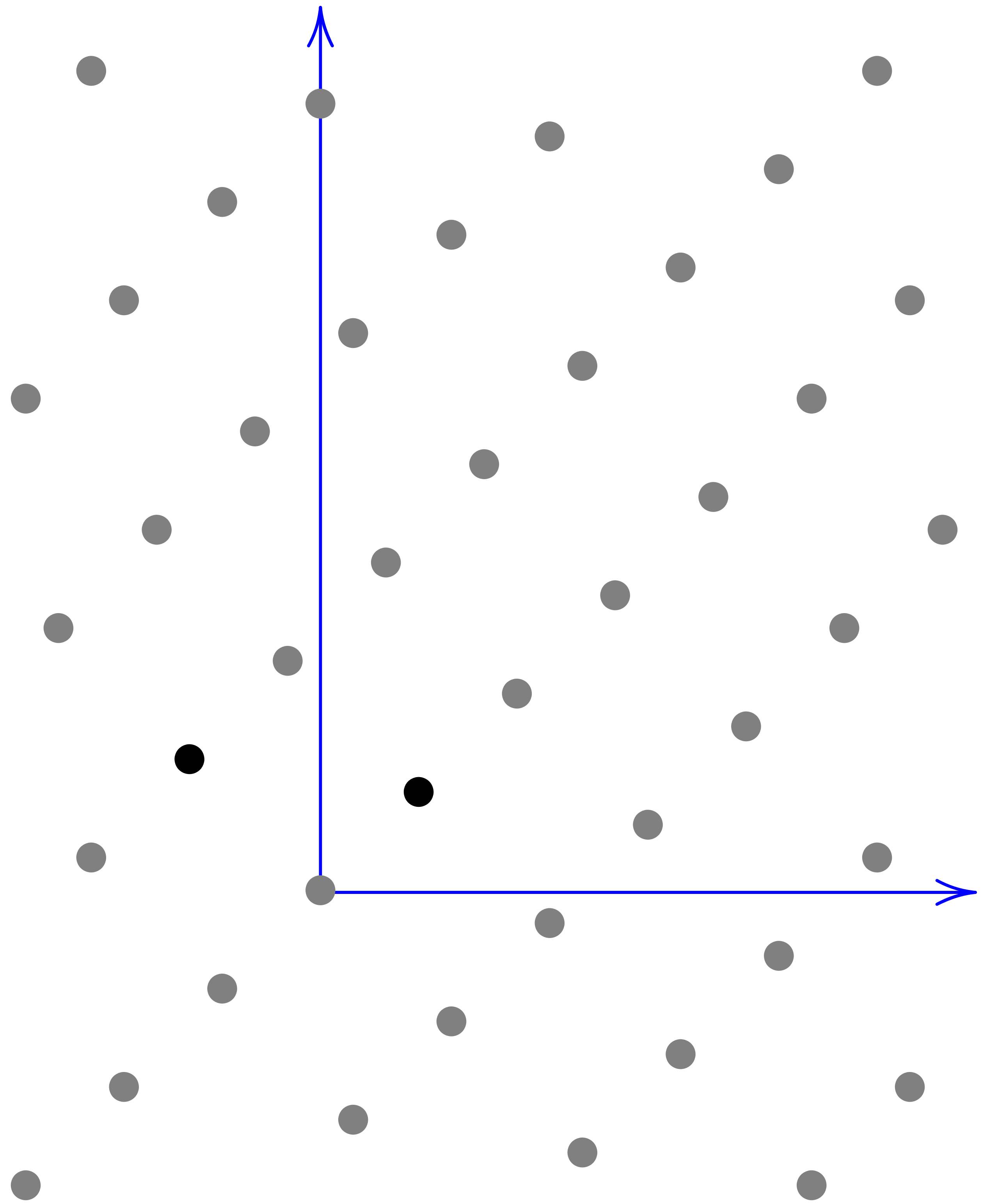


Another example:

Define $L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

$L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$

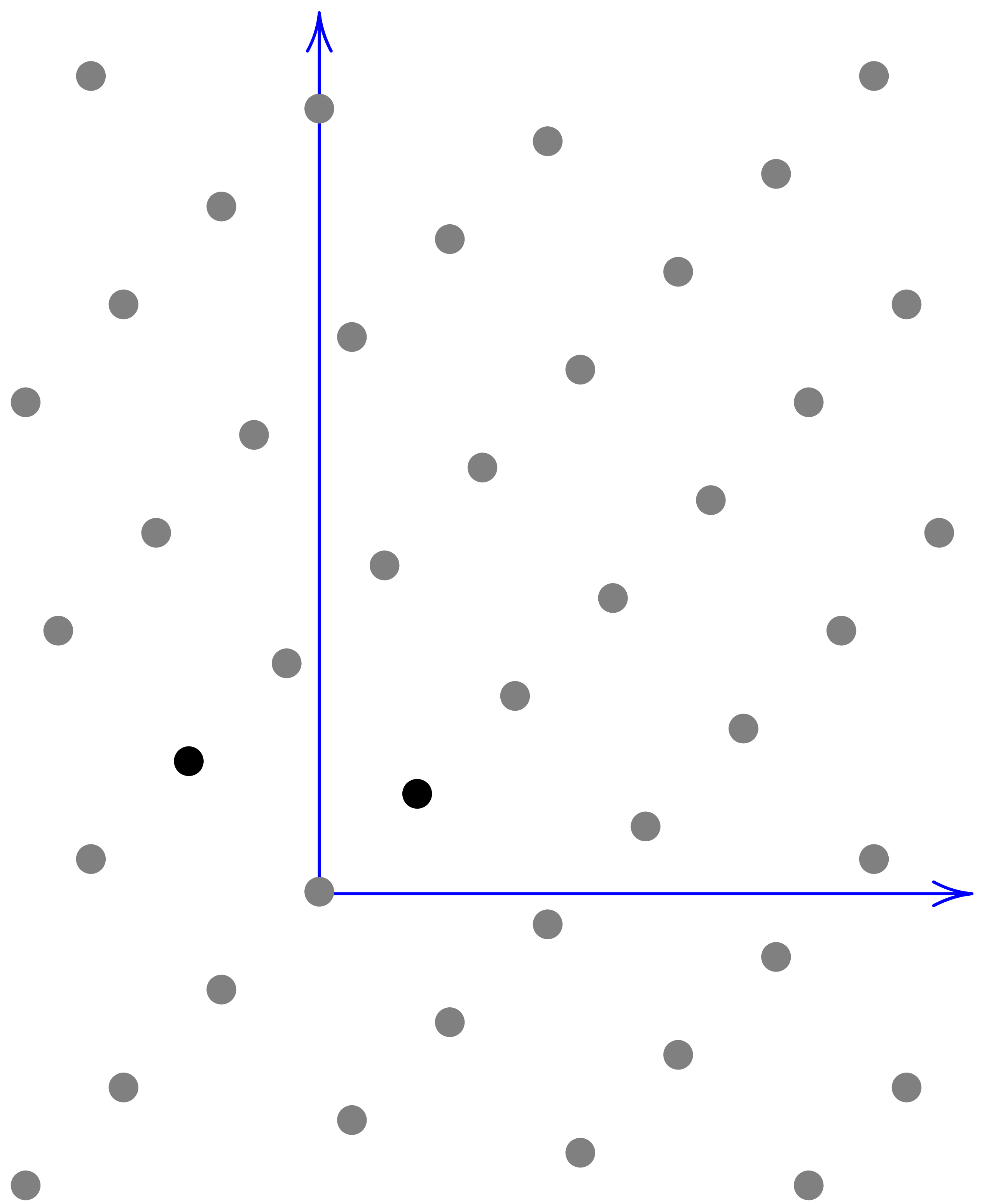


Another example:

Define $L = (0,25)\mathbf{Z} + (1,17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0,25)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,8)\mathbf{Z} + (1,17)\mathbf{Z} \end{aligned}$$

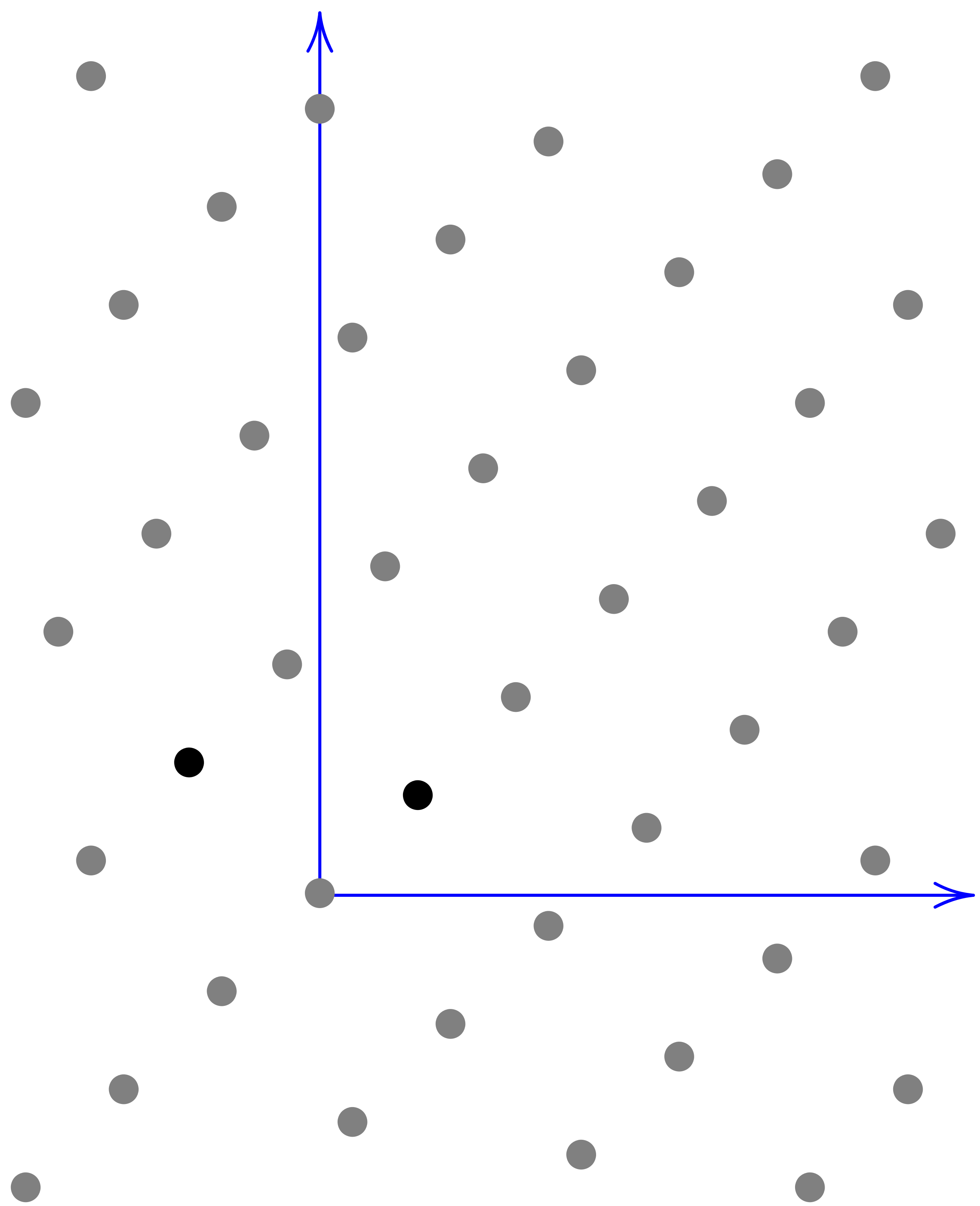


Another example:

Define $L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z} \\ &= (-1, 8)\mathbf{Z} + (1, 17)\mathbf{Z} \\ &= (-1, 8)\mathbf{Z} + (3, 1)\mathbf{Z}. \end{aligned}$$



Another example:

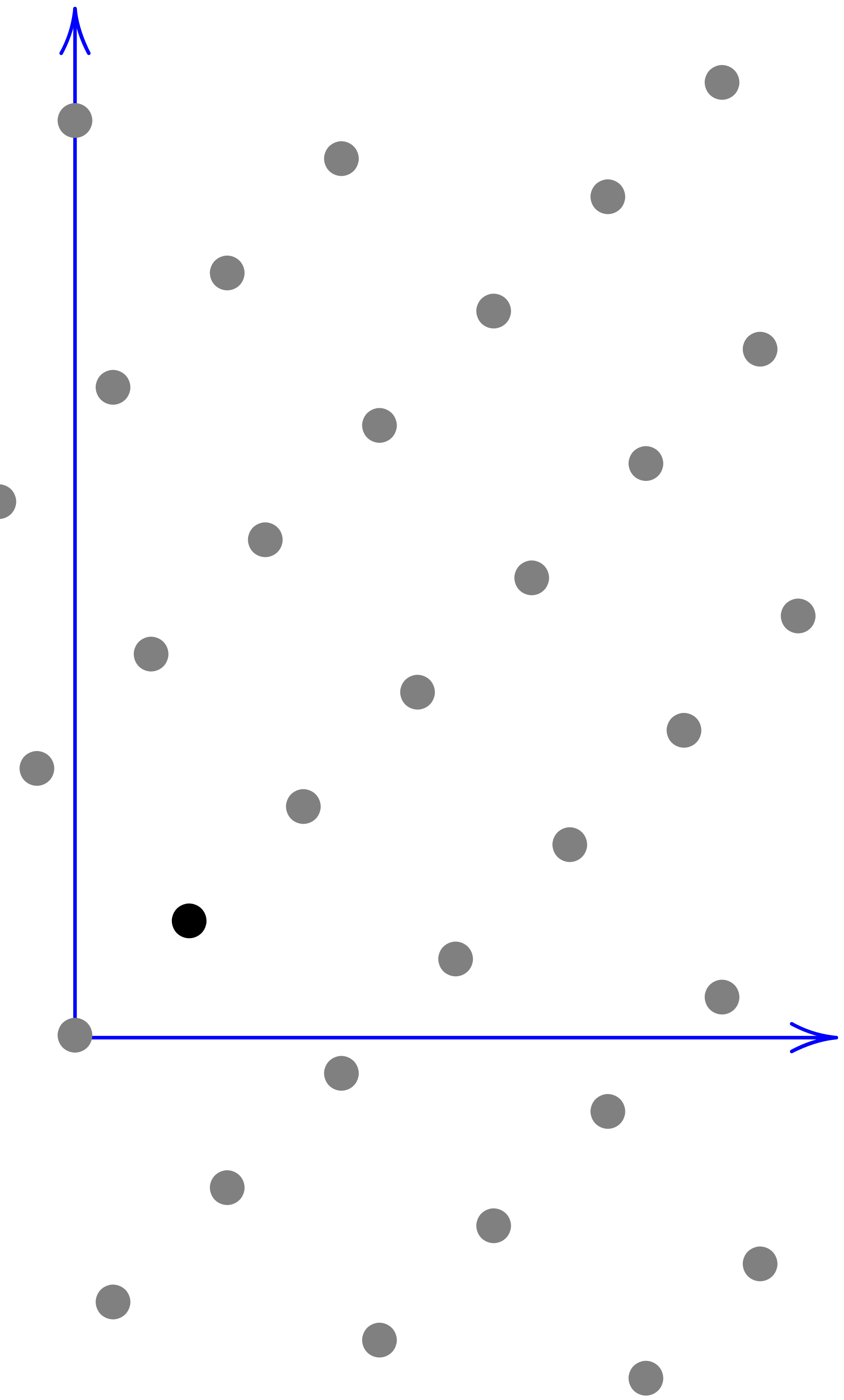
Define $L = (0,25)\mathbf{Z} + (1,17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0,25)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,8)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,8)\mathbf{Z} + (3,1)\mathbf{Z}. \end{aligned}$$

Nearly orthogonal.

Shortest vectors in L are $(0,0)$, $(3,1)$, $(-3,-1)$.



Another example:

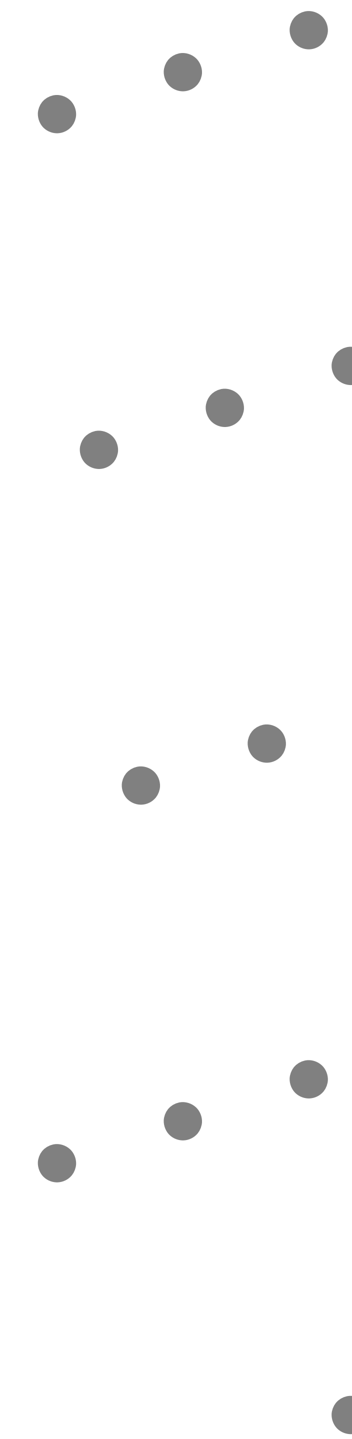
Define $L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$.

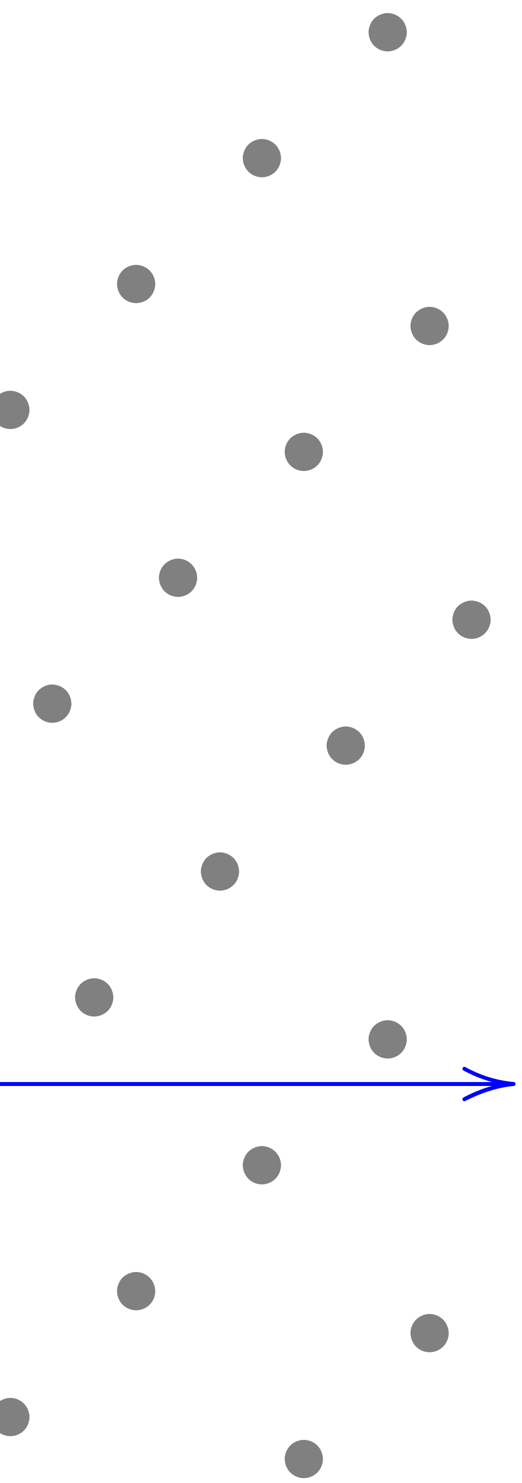
What is the shortest nonzero vector in L?

$$
\begin{aligned}
L &= (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 8)\mathbf{Z} + (1, 17)\mathbf{Z} \\
&= (-1, 8)\mathbf{Z} + (3, 1)\mathbf{Z}.
\end{aligned}
$$

Nearly orthogonal.

Shortest vectors in L are $(0, 0)$, $(3, 1)$, $(-3, -1)$.





Another example:

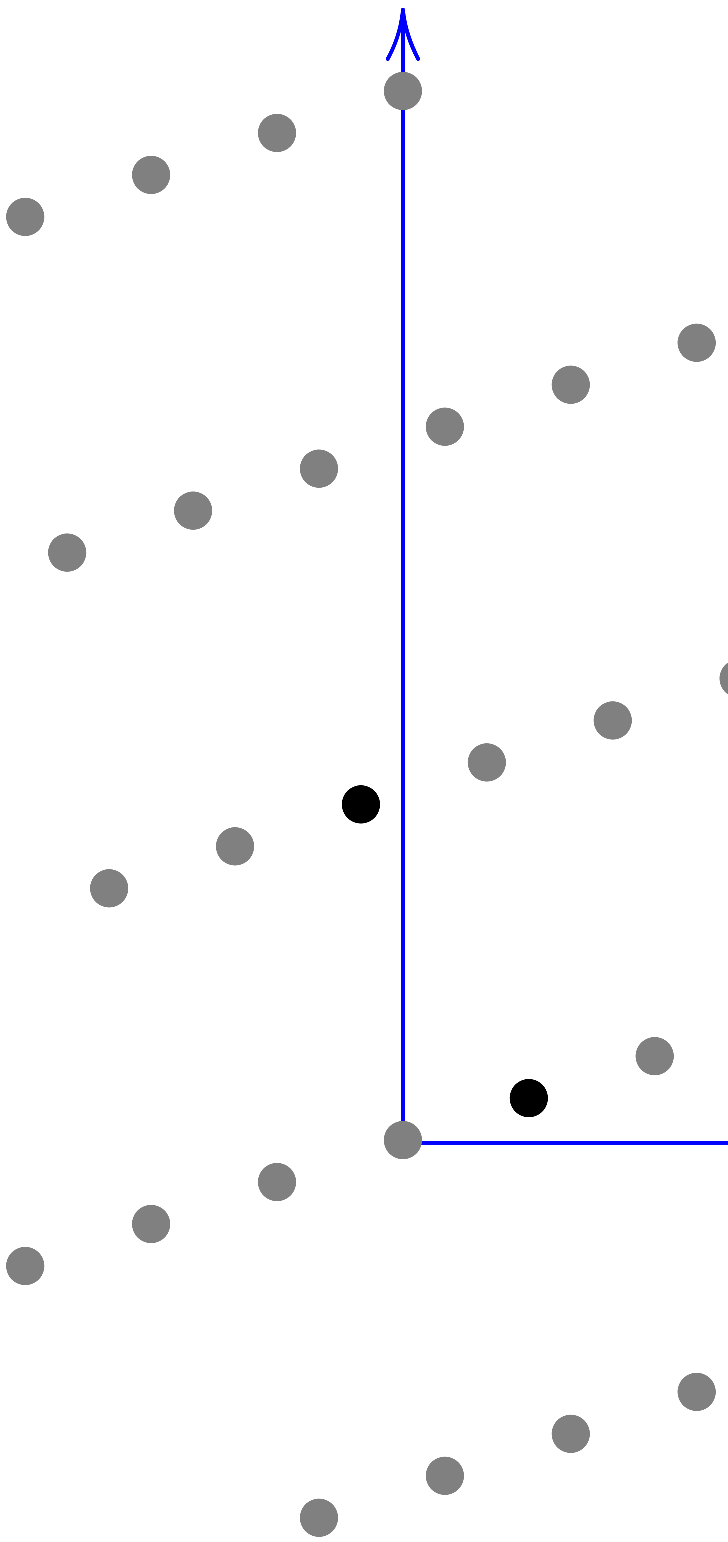
Define $L = (0,25)\mathbf{Z} + (1,17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0,25)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,8)\mathbf{Z} + (1,17)\mathbf{Z} \\ &= (-1,8)\mathbf{Z} + (3,1)\mathbf{Z}. \end{aligned}$$

Nearly orthogonal.

Shortest vectors in L are $(0,0)$, $(3,1)$, $(-3,-1)$.



Another example:

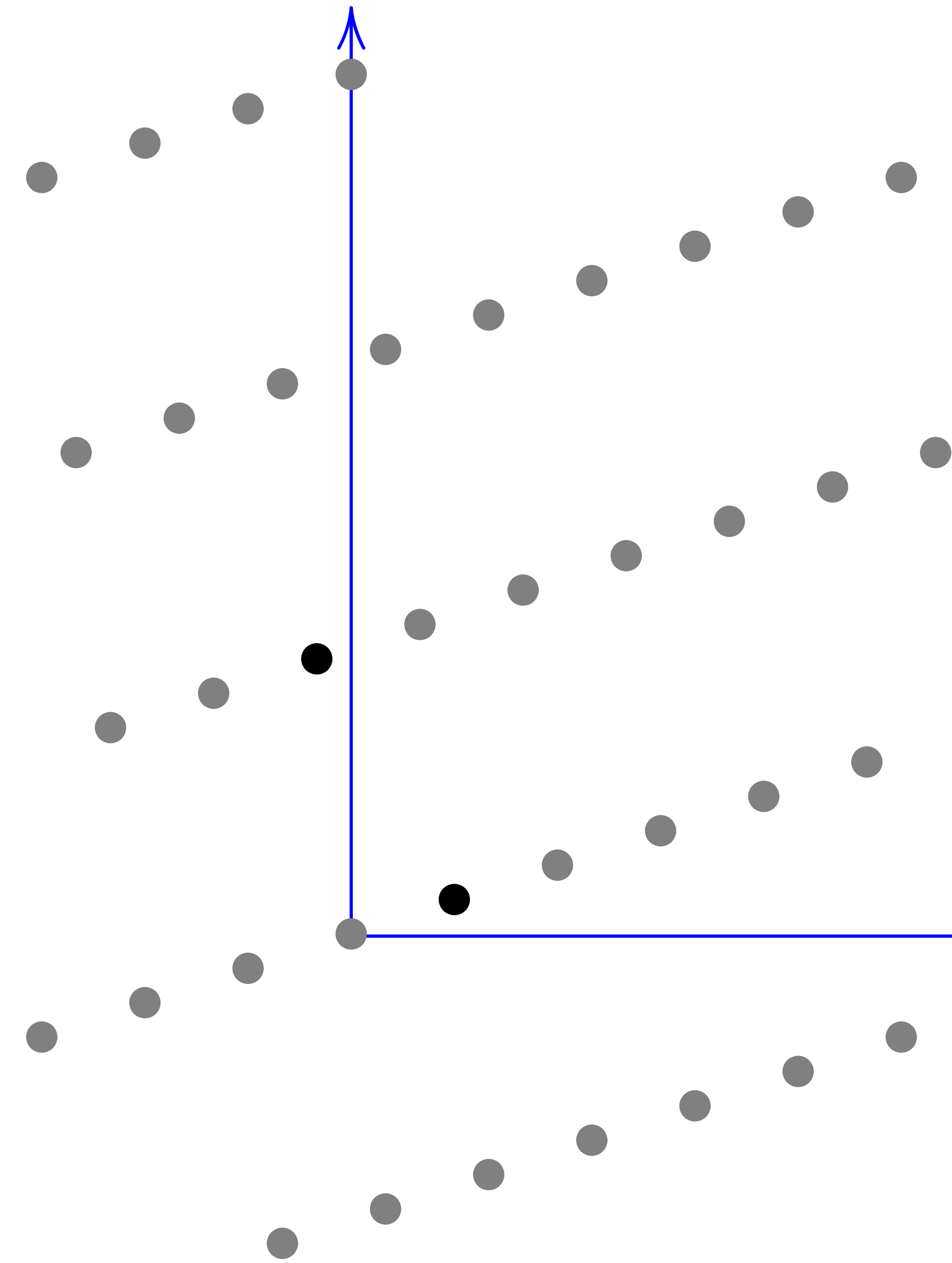
Define $L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

$$L = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}$$
$$= (-1, 8)\mathbf{Z} + (1, 17)\mathbf{Z}$$
$$= (-1, 8)\mathbf{Z} + (3, 1)\mathbf{Z}.$$

Nearly orthogonal.

Shortest vectors in L are $(0, 0)$, $(3, 1)$, $(-3, -1)$.



Another example:

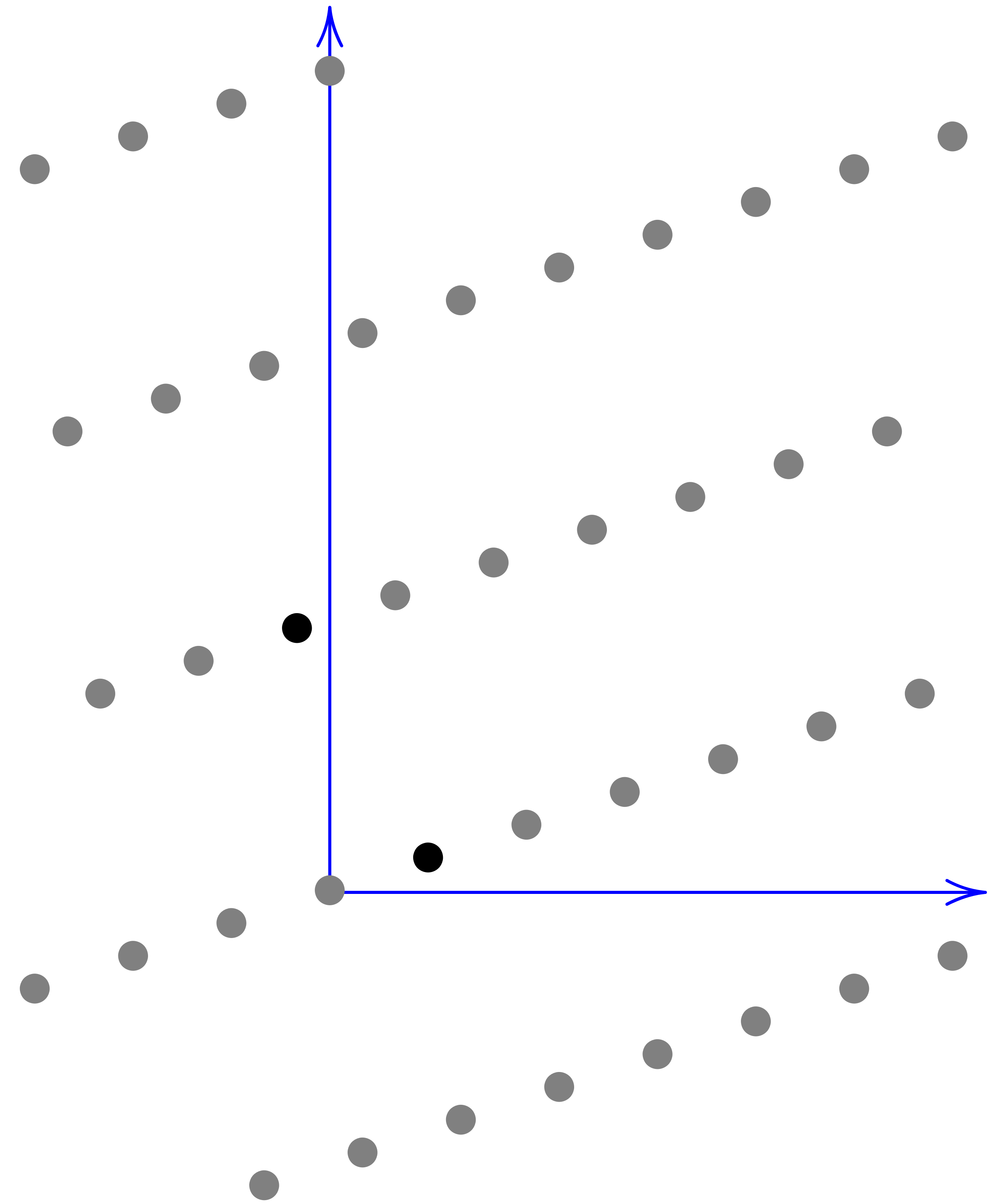
Define $L = (0,25)\mathbf{Z} + (1,17)\mathbf{Z}$.

What is the shortest nonzero vector in L?

$$L = (0,25)\mathbf{Z} + (1,17)\mathbf{Z}$$
$$= (-1,8)\mathbf{Z} + (1,17)\mathbf{Z}$$
$$= (-1,8)\mathbf{Z} + (3,1)\mathbf{Z}.$$

Nearly orthogonal.

Shortest vectors in L are $(0,0)$, $(3,1)$, $(-3,-1)$.



example:

$. = (0, 25)\mathbf{Z} + (1, 17)\mathbf{Z}.$

the shortest

vector in L?

$25)\mathbf{Z} + (1, 17)\mathbf{Z}$

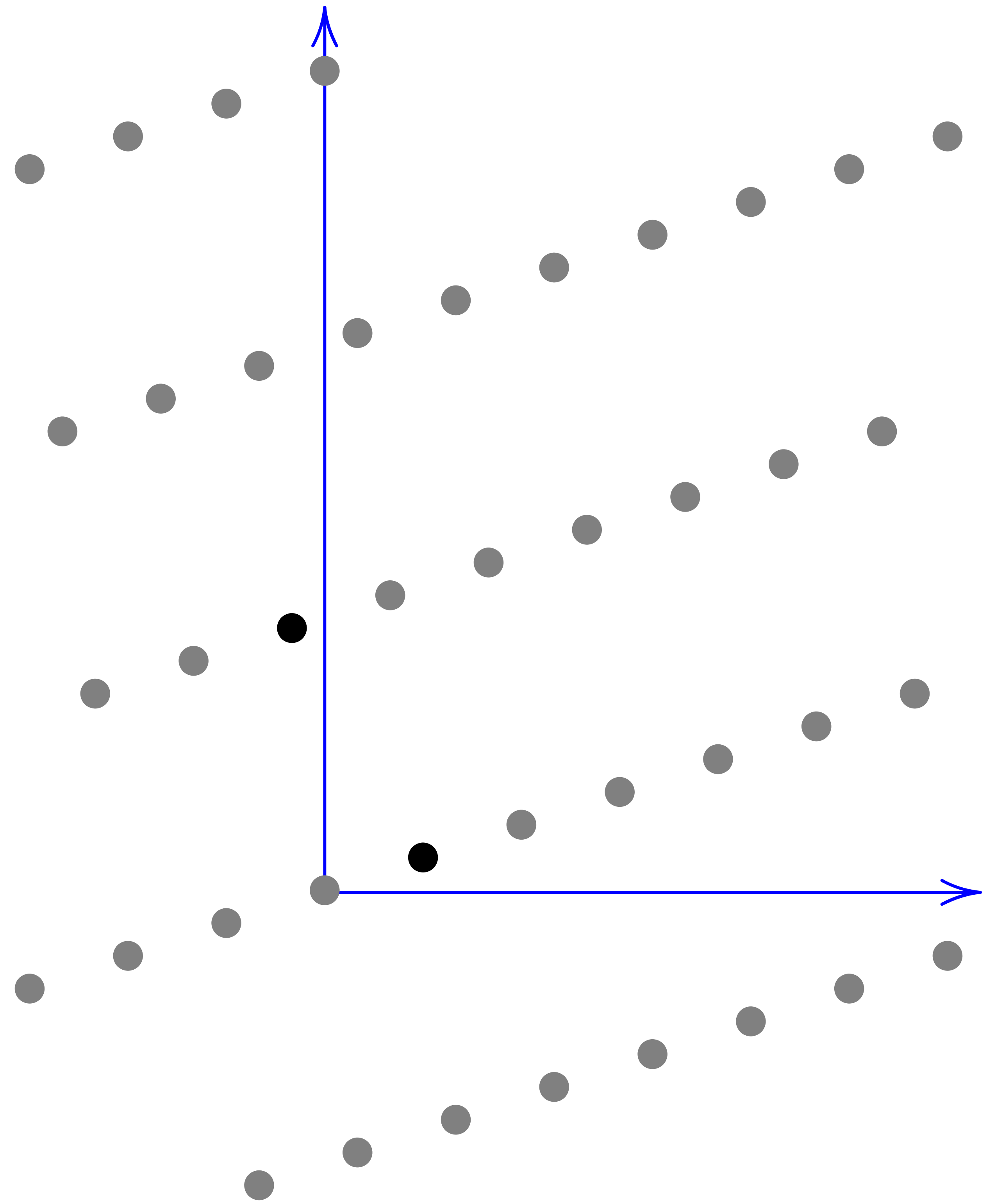
$., 8)\mathbf{Z} + (1, 17)\mathbf{Z}$

$., 8)\mathbf{Z} + (3, 1)\mathbf{Z}.$

rthogonal.

vectors in L are

$3, 1), (-3, -1).$



Polynom

Define F

$r_0 = (10$

$r_1 = (10$

$L = (0,$

What is

nonzero

$\mathbf{Z} + (1, 17)\mathbf{Z}$.

est

L?

$, 17)\mathbf{Z}$

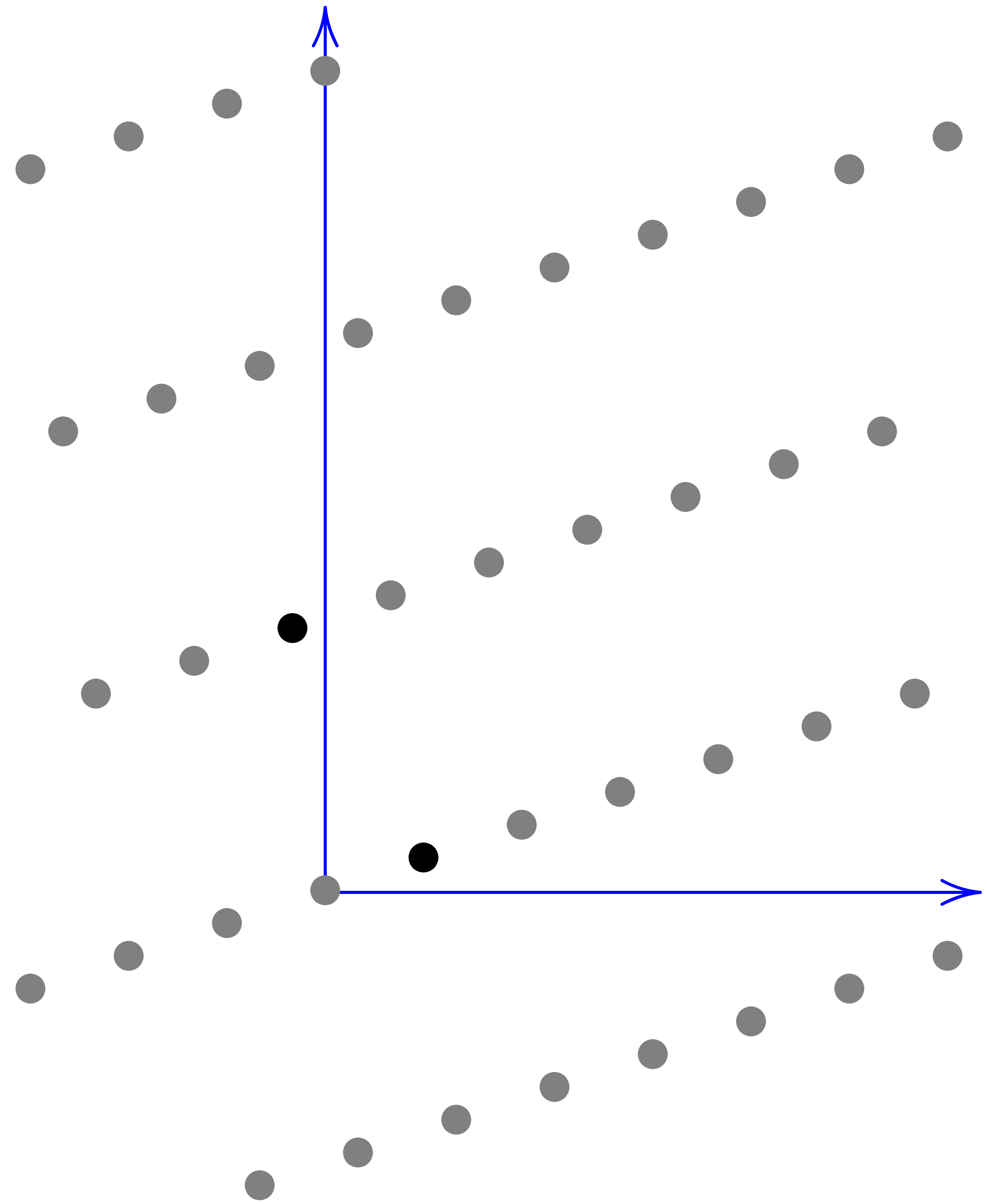
$., 17)\mathbf{Z}$

$3, 1)\mathbf{Z}$.

.

n L are

$-1)$.



Polynomial lattices

Define $P = \mathbf{F}_2[x]$,

$r_0 = (101000)_x =$

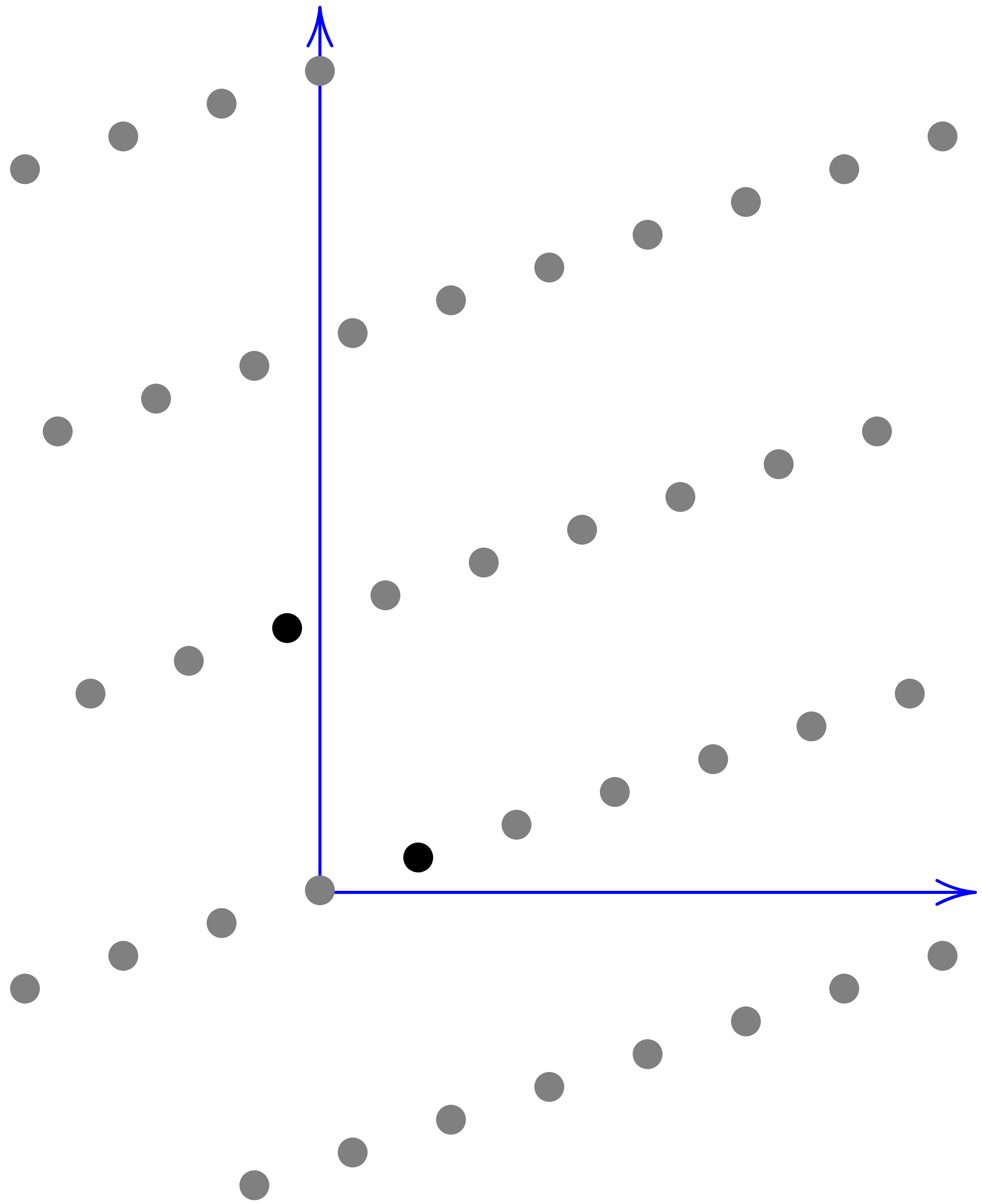
$r_1 = (10011)_x = x$

$L = (0, r_0)P + (1,$

What is the shorte

nonzero vector in

)**Z**.



Polynomial lattices

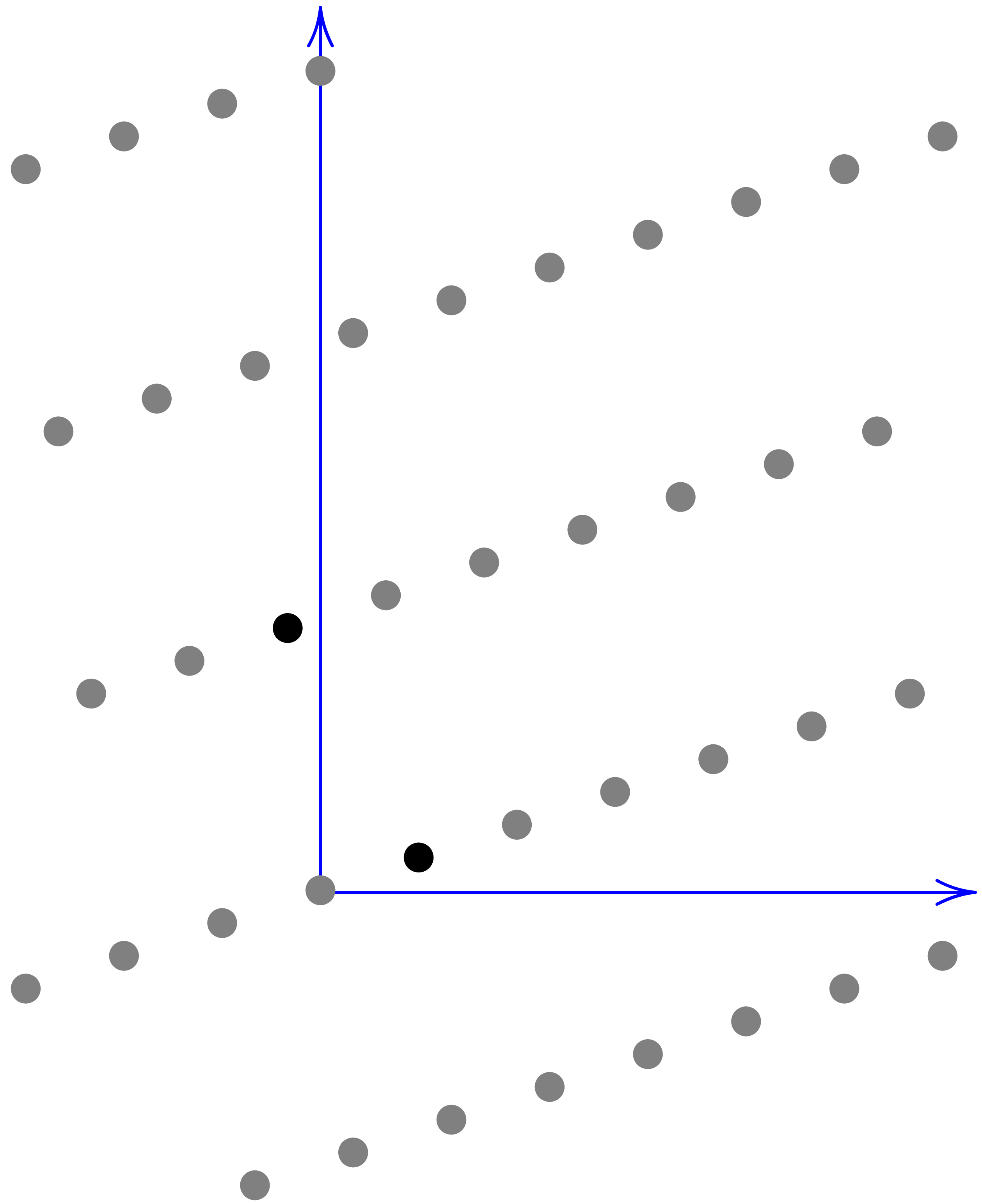
Define $P = \mathbf{F}_2[x]$,

$r_0 = (101000)_x = x^5 + x^3 \in$

$r_1 = (10011)_x = x^4 + x + 1$

$L = (0, r_0)P + (1, r_1)P$.

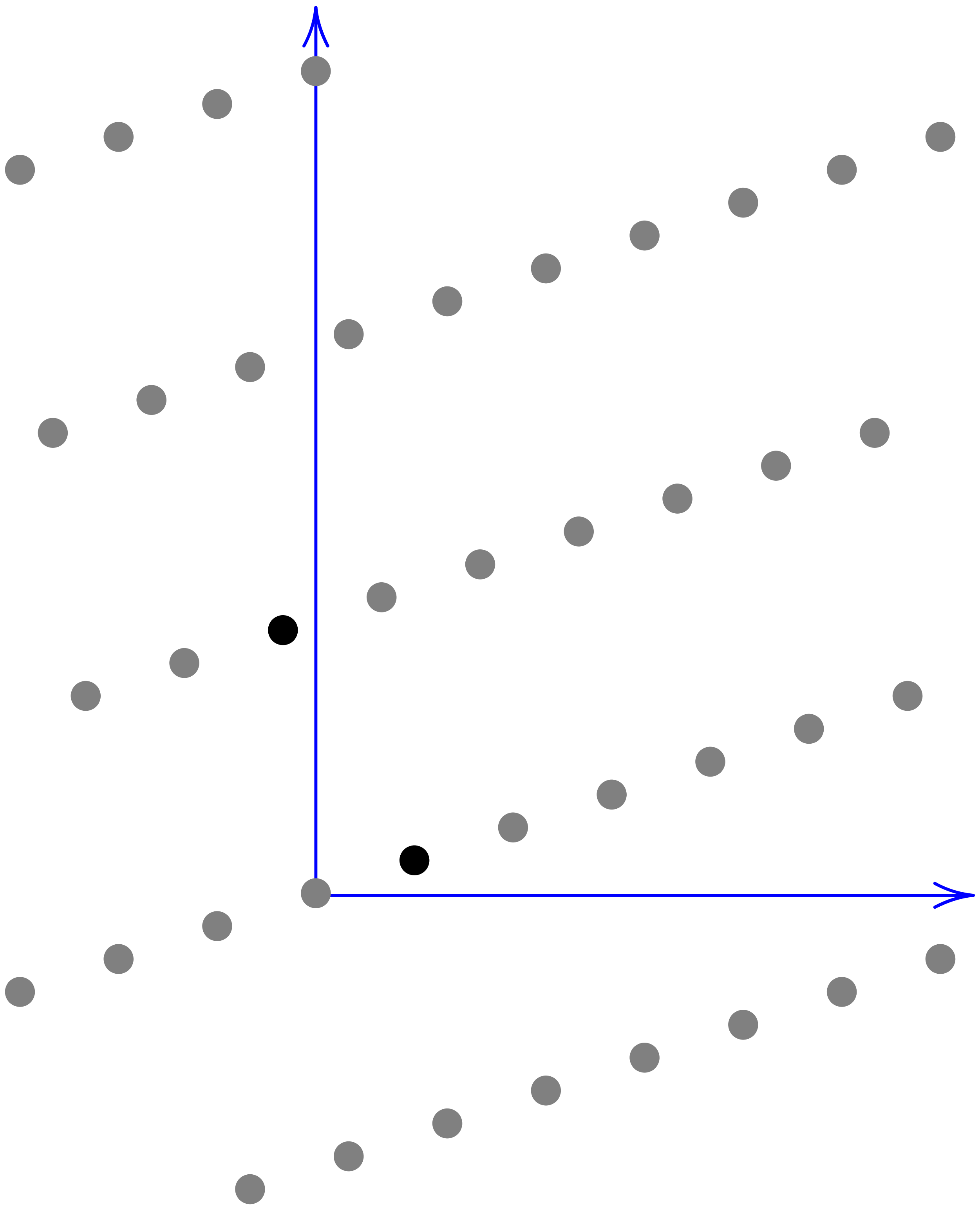
What is the shortest nonzero vector in L?



Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

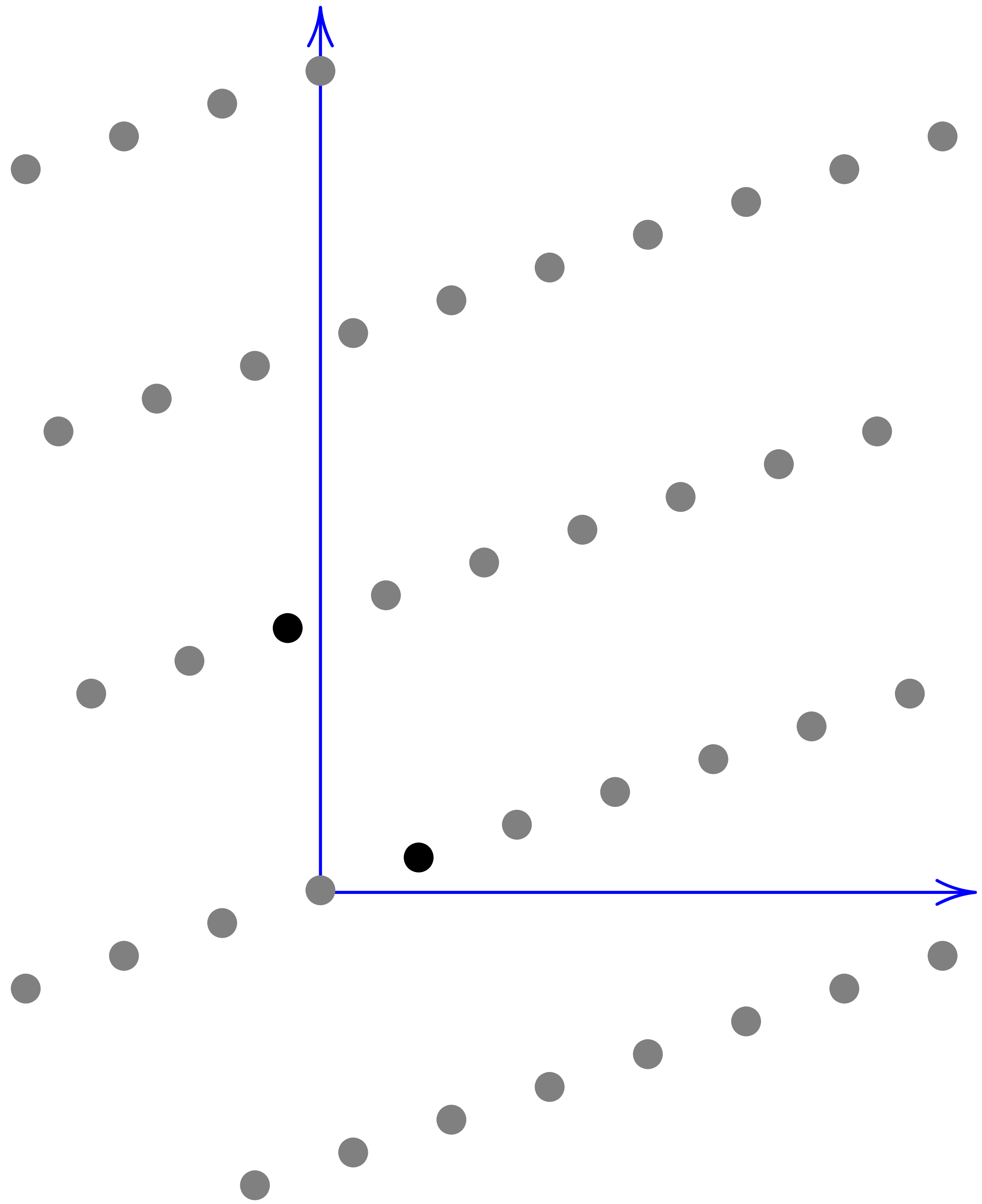


Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$L = (0, 101000)P + (1, 10011)P$

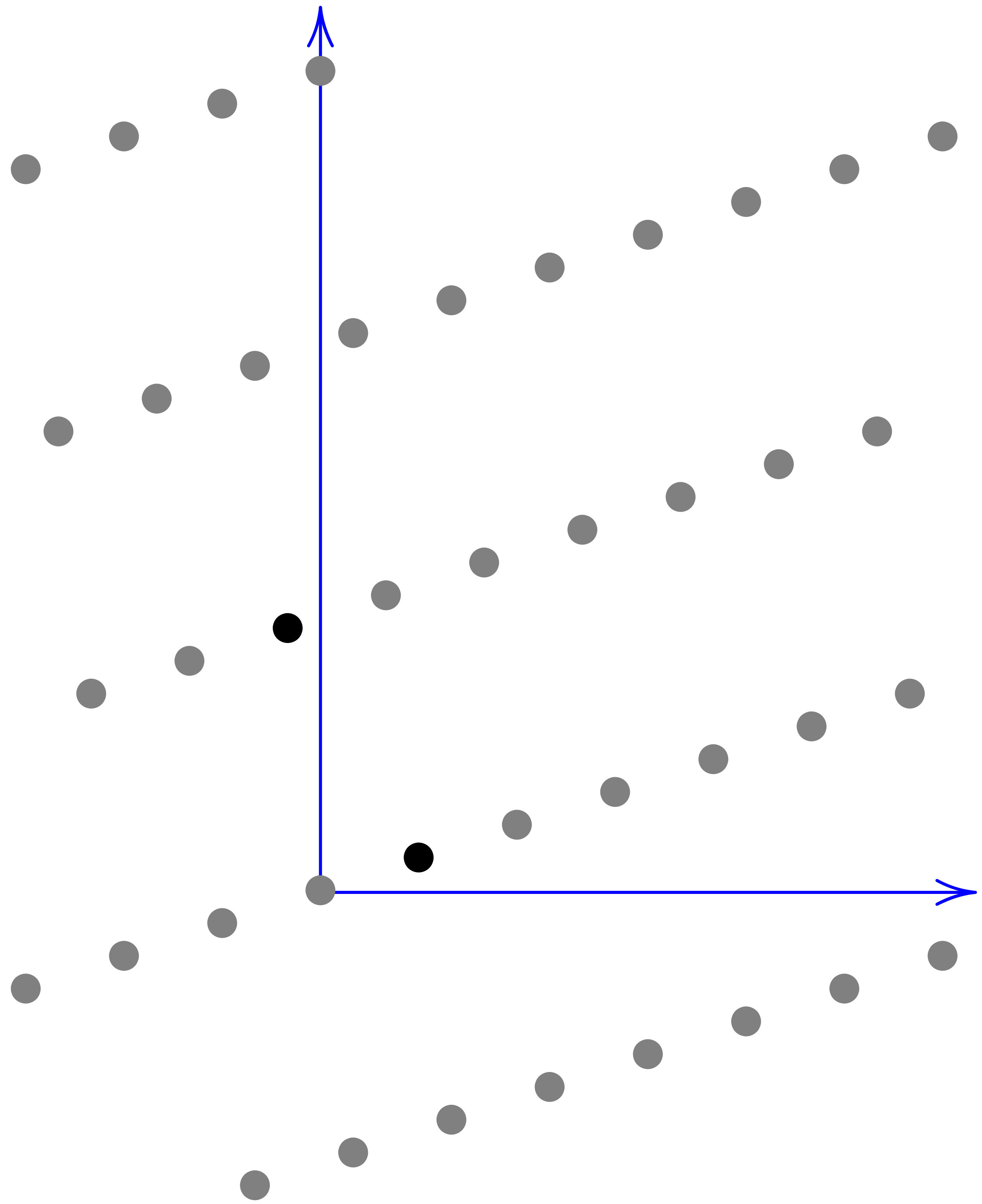


Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$L = (0, 101000)P + (1, 10011)P$$
$$= (10, 1110)P + (1, 10011)P$$

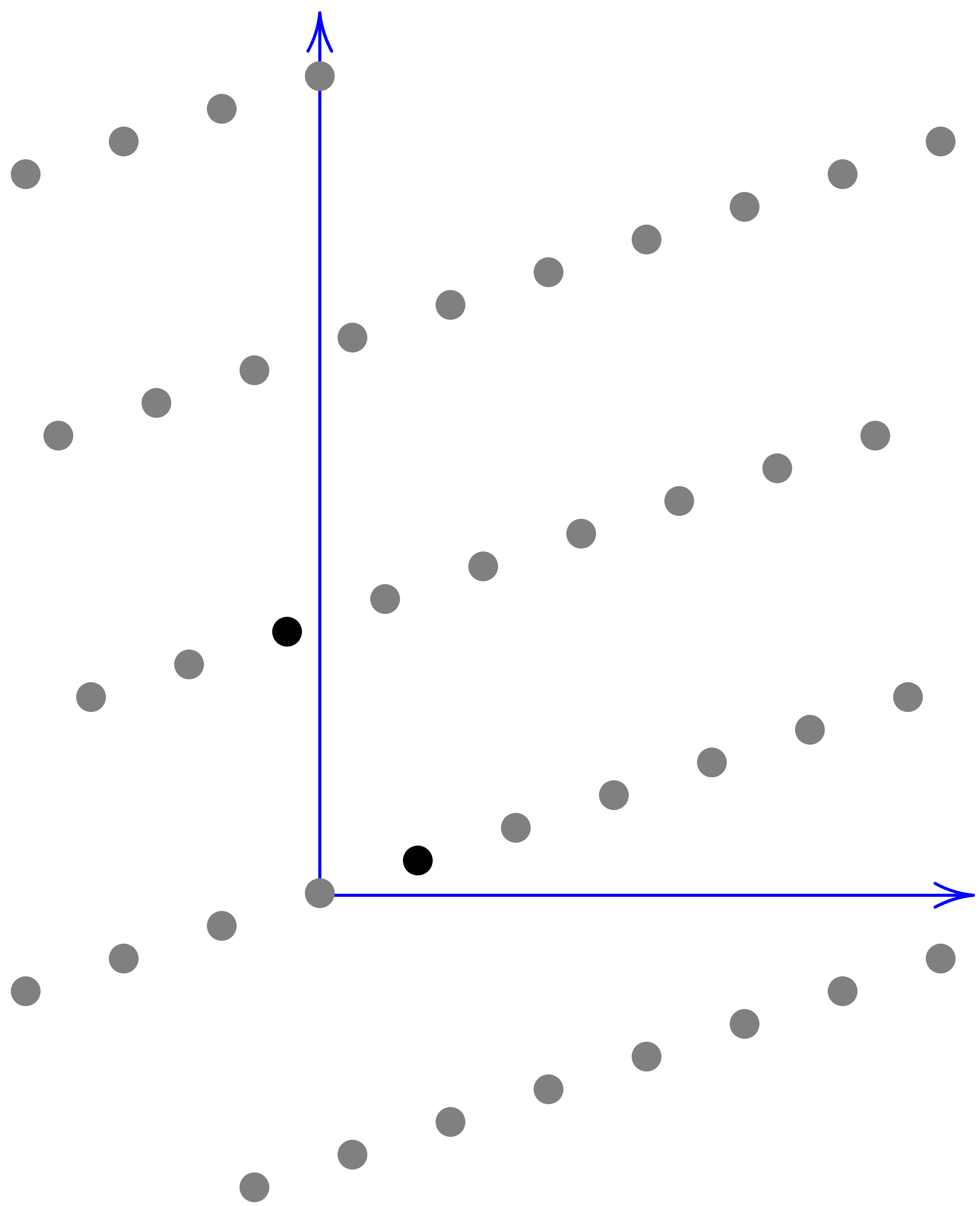


Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$L = (0, 101000)P + (1, 10011)P$$
$$= (10, 1110)P + (1, 10011)P$$
$$= (10, 1110)P + (111, 1)P.$$



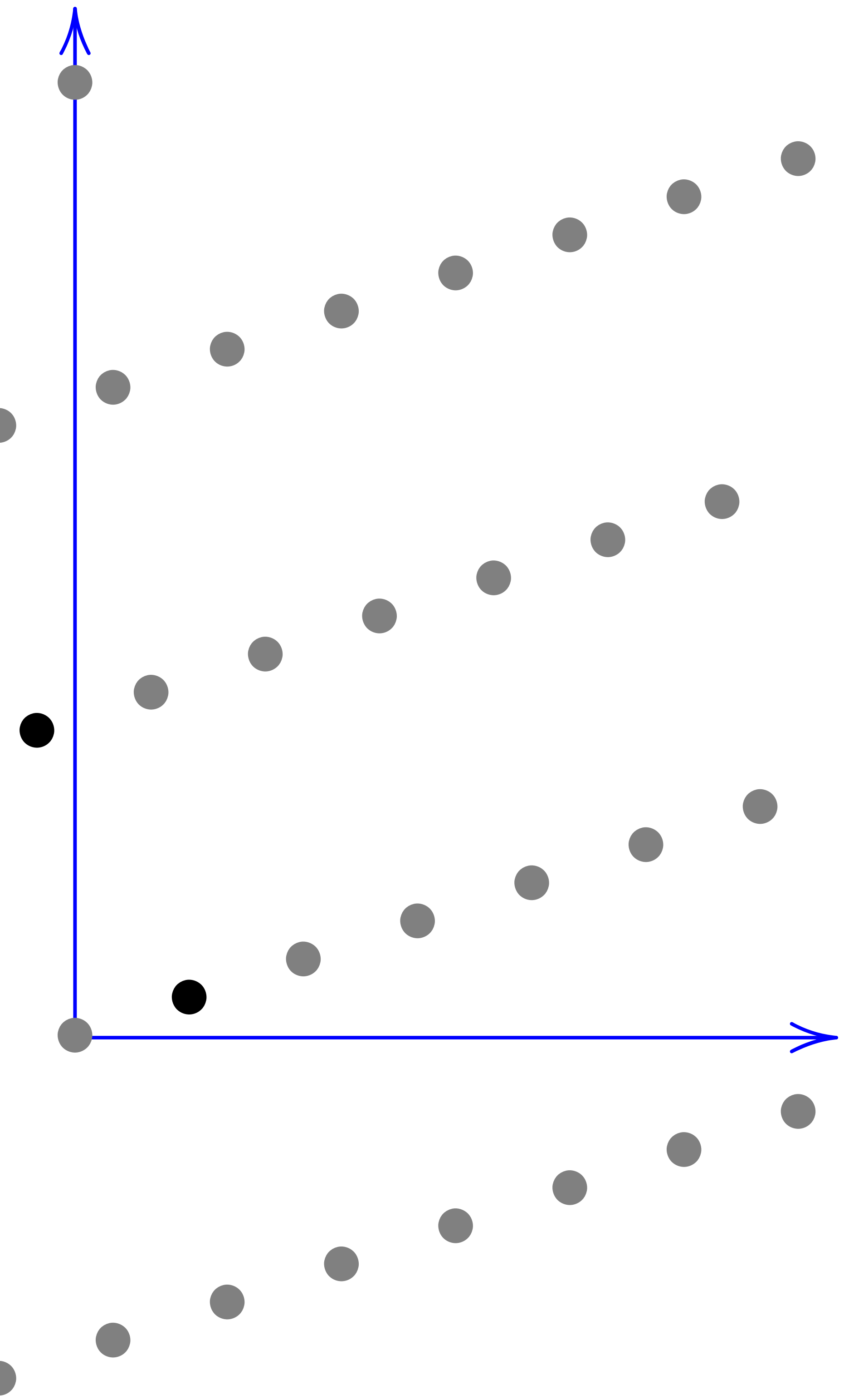
Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$
\begin{aligned}
L &= (0, 101000)P + (1, 10011)P \\
&= (10, 1110)P + (1, 10011)P \\
&= (10, 1110)P + (111, 1)P.
\end{aligned}
$$

$(111, 1)$: shortest nonzero vector.
$(10, 1110)$: shortest independent vector.



Polynomial lattices

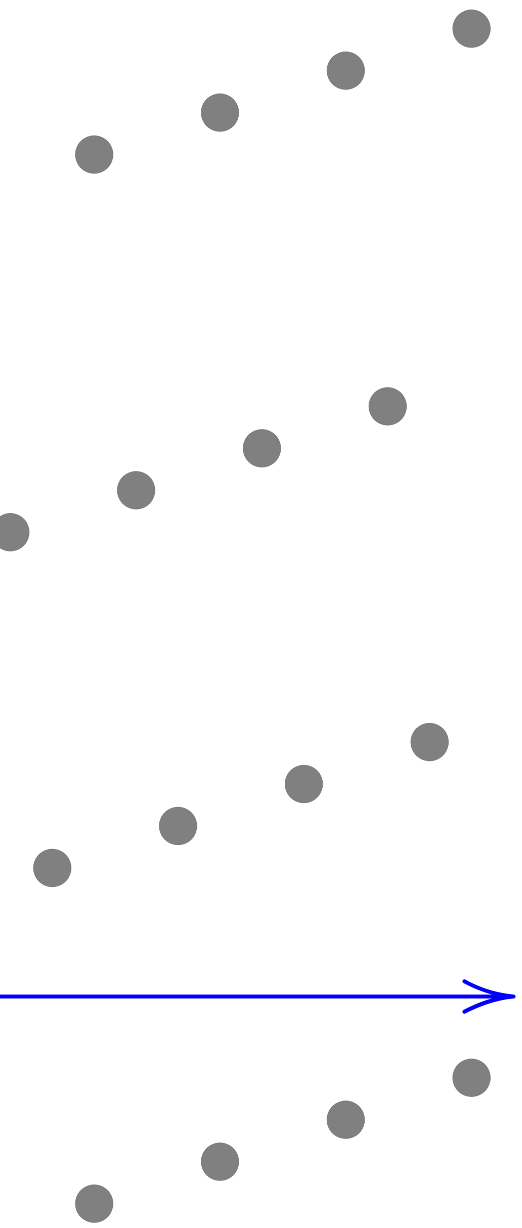
Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$L = (0, 101000)P + (1, 10011)P$$
$$= (10, 1110)P + (1, 10011)P$$
$$= (10, 1110)P + (111, 1)P.$$

$(111, 1)$: shortest nonzero vector.
$(10, 1110)$: shortest independent vector.

Degree c
is define



Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$
\begin{aligned}
L &= (0, 101000)P + (1, 10011)P \\
&= (10, 1110)P + (1, 10011)P \\
&= (10, 1110)P + (111, 1)P.
\end{aligned}
$$

$(111, 1)$: shortest nonzero vector.
$(10, 1110)$: shortest independent vector.

Degree of $(q, r) \in$
is defined as max{

Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0, 101000)P + (1, 10011)P \\ &= (10, 1110)P + (1, 10011)P \\ &= (10, 1110)P + (111, 1)P. \end{aligned}$$

$(111, 1)$: shortest nonzero vector.
$(10, 1110)$: shortest independent vector.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}$
is defined as $\max\{\deg q, \deg$

Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0, 101000)P + (1, 10011)P \\ &= (10, 1110)P + (1, 10011)P \\ &= (10, 1110)P + (111, 1)P. \end{aligned}$$

$(111, 1)$: shortest nonzero vector.
$(10, 1110)$: shortest independent vector.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$ is defined as $\max\{\deg q, \deg r\}$.

Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0, 101000)P + (1, 10011)P \\ &= (10, 1110)P + (1, 10011)P \\ &= (10, 1110)P + (111, 1)P. \end{aligned}$$

$(111, 1)$: shortest nonzero vector.
$(10, 1110)$: shortest independent vector.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$ is defined as $\max\{\deg q, \deg r\}$.

Can use other metrics, or equivalently rescale L.

e.g. Define $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$ as $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Polynomial lattices

Define $P = \mathbf{F}_2[x]$,
$r_0 = (101000)_x = x^5 + x^3 \in P$,
$r_1 = (10011)_x = x^4 + x + 1 \in P$,
$L = (0, r_0)P + (1, r_1)P$.

What is the shortest nonzero vector in L?

$$\begin{aligned} L &= (0, 101000)P + (1, 10011)P \\ &= (10, 1110)P + (1, 10011)P \\ &= (10, 1110)P + (111, 1)P. \end{aligned}$$

$(111, 1)$: shortest nonzero vector.
$(10, 1110)$: shortest independent vector.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$ is defined as $\max\{\deg q, \deg r\}$.

Can use other metrics, or equivalently rescale L.

e.g. Define $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$ as $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10011\sqrt{x})$, degree 4.5.
$(10, 1110\sqrt{x})$, degree 3.5.
$(111, 1\sqrt{x})$, degree 2.

nial lattices

$P = \mathbf{F}_2[x]$,
$01000)_x = x^5 + x^3 \in P$,
$0011)_x = x^4 + x + 1 \in P$,
$r_0)P + (1, r_1)P$.

the shortest
vector in L?

$101000)P + (1, 10011)P$
$, 1110)P + (1, 10011)P$
$, 1110)P + (111, 1)P$.

shortest nonzero vector.
$0)$: shortest
dent vector.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$
is defined as $\max\{\deg q, \deg r\}$.

Can use other metrics,
or equivalently rescale L.

e.g. Define $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$
as $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10011\sqrt{x})$, degree 4.5.
$(10, 1110\sqrt{x})$, degree 3.5.
$(111, 1\sqrt{x})$, degree 2.

Warning
shortest
after sh

s

$x^5 + x^3 \in P$,

$x^4 + x + 1 \in P$,

$r_1)P$.

est

L?

$+ (1, 10011)P$

$- (1, 10011)P$

$- (111, 1)P$.

nonzero vector.

st

r.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$
is defined as $\max\{\deg q, \deg r\}$.

Can use other metrics,
or equivalently rescale L.

e.g. Define $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$
as $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10011\sqrt{x})$, degree 4.5.
$(10, 1110\sqrt{x})$, degree 3.5.
$(111, 1\sqrt{x})$, degree 2.

Warning: Sometin
shortest independe
after shortest non

$\in P$,
$\in P$,

$1)P$
$)P$
P.

ector.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$
is defined as $\max\{\deg q, \deg r\}$.

Can use other metrics,
or equivalently rescale L.

e.g. Define $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$
as $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10011\sqrt{x})$, degree 4.5.
$(10, 1110\sqrt{x})$, degree 3.5.
$(111, 1\sqrt{x})$, degree 2.

Warning: Sometimes
shortest independent vector
after shortest nonzero vecto

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$ is defined as $\max\{\deg q, \deg r\}$.

Can use other metrics, or equivalently rescale L.

e.g. Define $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$ as $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10011\sqrt{x})$, degree 4.5.
$(10, 1110\sqrt{x})$, degree 3.5.
$(111, 1\sqrt{x})$, degree 2.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

Degree of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$ is defined as $\max\{\deg q, \deg r\}$.

Can use other metrics, or equivalently rescale L.

e.g. Define $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$ as $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10011\sqrt{x})$, degree 4.5.
$(10, 1110\sqrt{x})$, degree 3.5.
$(111, 1\sqrt{x})$, degree 2.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define
$r_0 = 101000$, $r_1 = 10111$,
$L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10111\sqrt{x})$, degree 4.5.
$(10, 110\sqrt{x})$, degree 2.5.
$(1101, 11\sqrt{x})$, degree 3.

of $(q, r) \in \mathbf{F}_2[x] \times \mathbf{F}_2[x]$
d as $\max\{\deg q, \deg r\}$.

other metrics,
alently rescale L.

ine $L \subseteq \mathbf{F}_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$
$\sqrt{x})P + (1, r_1\sqrt{x})P$.

ve generators for L:
$00\sqrt{x})$, degree 5.5.
$1\sqrt{x})$, degree 4.5.
$0\sqrt{x})$, degree 3.5.
$\sqrt{x})$, degree 2.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define
$r_0 = 101000$, $r_1 = 10111$,
$L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10111\sqrt{x})$, degree 4.5.
$(10, 110\sqrt{x})$, degree 2.5.
$(1101, 11\sqrt{x})$, degree 3.

For any
in $P =$

Euclid/S
Define r
$r_3 = r_1$

$\mathbf{F}_2[x] \times \mathbf{F}_2[x]$

$\{\deg q, \deg r\}$.

trics,

cale L.

$_2[\sqrt{x}] \times \mathbf{F}_2[\sqrt{x}]$

$1, r_1\sqrt{x})P$.

tors for L:

egree 5.5.

gree 4.5.

gree 3.5.

e 2.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define

$r_0 = 101000$, $r_1 = 10111$,

$L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:

$(0, 101000\sqrt{x})$, degree 5.5.

$(1, 10111\sqrt{x})$, degree 4.5.

$(10, 110\sqrt{x})$, degree 2.5.

$(1101, 11\sqrt{x})$, degree 3.

For any field k, an

in $P = k[x]$ with c

Euclid/Stevin com

Define $r_2 = r_0$ mod

$r_3 = r_1 \bmod r_2$, etc

$_2[x]$

$; r\}$.

$_2[\sqrt{x}]$

.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define

$r_0 = 101000$, $r_1 = 10111$,

$L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:

$(0, 101000\sqrt{x})$, degree 5.5.

$(1, 10111\sqrt{x})$, degree 4.5.

$(10, 110\sqrt{x})$, degree 2.5.

$(1101, 11\sqrt{x})$, degree 3.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 >$ de

Euclid/Stevin computation:

Define $r_2 = r_0 \bmod r_1$,

$r_3 = r_1 \bmod r_2$, etc.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define
$r_0 = 101000$, $r_1 = 10111$,
$L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10111\sqrt{x})$, degree 4.5.
$(10, 110\sqrt{x})$, degree 2.5.
$(1101, 11\sqrt{x})$, degree 3.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$,
$r_3 = r_1 \bmod r_2$, etc.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define
$r_0 = 101000$, $r_1 = 10111$,
$L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10111\sqrt{x})$, degree 4.5.
$(10, 110\sqrt{x})$, degree 2.5.
$(1101, 11\sqrt{x})$, degree 3.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$,
$r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$;
$q_{i+2} = q_i - \lfloor r_i / r_{i+1} \rfloor q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define $r_0 = 101000$, $r_1 = 10111$, $L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10111\sqrt{x})$, degree 4.5.
$(10, 110\sqrt{x})$, degree 2.5.
$(1101, 11\sqrt{x})$, degree 3.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$, $r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$;
$q_{i+2} = q_i - \lfloor r_i/r_{i+1} \rfloor q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P =$
$(q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Warning: Sometimes shortest independent vector is *after* shortest nonzero vector.

e.g. Define
$r_0 = 101000$, $r_1 = 10111$,
$L = (0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

Successive generators for L:
$(0, 101000\sqrt{x})$, degree 5.5.
$(1, 10111\sqrt{x})$, degree 4.5.
$(10, 110\sqrt{x})$, degree 2.5.
$(1101, 11\sqrt{x})$, degree 3.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$,
$r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$;
$q_{i+2} = q_i - \lfloor r_i/r_{i+1} \rfloor q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P =$
$(q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Can continue until $r_{i+1} = 0$.
$\gcd\{r_0, r_1\} = r_i / \operatorname{leadcoeff} r_i$.

: Sometimes
independent vector is
ortest nonzero vector.

ine
000, $r_1 = 10111$,
$r_0\sqrt{x})P + (1, r_1\sqrt{x})P$.

ve generators for L:
$00\sqrt{x})$, degree 5.5.
$1\sqrt{x})$, degree 4.5.
$\sqrt{x})$, degree 2.5.
$1\sqrt{x})$, degree 3.

For any field k, any r_0, r_1
in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$,
$r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$;
$q_{i+2} = q_i - \lfloor r_i/r_{i+1} \rfloor q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P =$
$(q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Can continue until $r_{i+1} = 0$.
$\gcd\{r_0, r_1\} = r_i/\operatorname{leadcoeff} r_i$.

Reducin
is a "hal
stopping

nes
ent vector is
zero vector.

10111,
$(1, r_1\sqrt{x})P$.

tors for L:
egree 5.5.
gree 4.5.
ee 2.5.
gree 3.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation: Define $r_2 = r_0 \bmod r_1$, $r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$; $q_{i+2} = q_i - \lfloor r_i / r_{i+1} \rfloor q_{i+1}$. Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P = (q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Can continue until $r_{i+1} = 0$. $\gcd\{r_0, r_1\} = r_i / \operatorname{leadcoeff} r_i$.

Reducing lattice b
is a "half gcd" co
stopping halfway t

is

r.

P.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation: Define $r_2 = r_0 \bmod r_1$, $r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$; $q_{i+2} = q_i - \lfloor r_i / r_{i+1} \rfloor q_{i+1}$. Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have

$$(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P =$$
$$(q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P.$$

Can continue until $r_{i+1} = 0$. $\gcd\{r_0, r_1\} = r_i / \operatorname{leadcoeff} r_i$.

Reducing lattice basis for L is a "half gcd" computation stopping halfway to the gcd.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$,
$r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$;
$q_{i+2} = q_i - \lfloor r_i / r_{i+1} \rfloor q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P =$
$(q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Can continue until $r_{i+1} = 0$.
$\gcd\{r_0, r_1\} = r_i / \operatorname{leadcoeff} r_i$.

Reducing lattice basis for L is a "half gcd" computation, stopping halfway to the gcd.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$,
$r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$;
$q_{i+2} = q_i - \lfloor r_i / r_{i+1} \rfloor q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P =$
$(q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Can continue until $r_{i+1} = 0$.
$\gcd\{r_0, r_1\} = r_i / \operatorname{leadcoeff} r_i$.

Reducing lattice basis for L is a "half gcd" computation, stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation:
Define $r_2 = r_0 \bmod r_1$,
$r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$;
$q_{i+2} = q_i - \lfloor r_i / r_{i+1} \rfloor\, q_{i+1}$.
Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have
$(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P =$
$(q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Can continue until $r_{i+1} = 0$.
$\gcd\{r_0, r_1\} = r_i / \operatorname{leadcoeff} r_i$.

Reducing lattice basis for L is a "half gcd" computation, stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with
$\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so
$\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

For any field k, any r_0, r_1 in $P = k[x]$ with $\deg r_0 > \deg r_1$:

Euclid/Stevin computation: Define $r_2 = r_0 \bmod r_1$, $r_3 = r_1 \bmod r_2$, etc.

Extended: $q_0 = 0$; $q_1 = 1$; $q_{i+2} = q_i - \lfloor r_i / r_{i+1} \rfloor q_{i+1}$. Then $q_i r_1 \equiv r_i \pmod{r_0}$.

Lattice view: Have $(0, r_0\sqrt{x})P + (1, r_1\sqrt{x})P = (q_i, r_i\sqrt{x})P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

Can continue until $r_{i+1} = 0$. $\gcd\{r_0, r_1\} = r_i / \operatorname{leadcoeff} r_i$.

Reducing lattice basis for L is a "half gcd" computation, stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases; $\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with $\deg r_j\sqrt{x} \le (\deg r_0)/2$. Then $\deg q_j \le (\deg r_0)/2$ so $\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$. Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree $\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$ for some $\epsilon \in \{-1, 1\}$. Shortest independent vector.

field k, any r_0, r_1
$k[x]$ with $\deg r_0 > \deg r_1$:

Stevin computation:
$_2 = r_0 \bmod r_1$,
mod r_2, etc.

d: $q_0 = 0$; $q_1 = 1$;
$q_i - \lfloor r_i/r_{i+1} \rfloor q_{i+1}$.
$r_1 \equiv r_i \pmod{r_0}$.

view: Have
$)P + (1, r_1\sqrt{x})P =$
$)P + (q_{i+1}, r_{i+1}\sqrt{x})P$.

tinue until $r_{i+1} = 0$.
$_1\} = r_i / \operatorname{leadcoeff} r_i$.

Reducing lattice basis for L
is a "half gcd" computation,
stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with
$\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so
$\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree
$\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$
for some $\epsilon \in \{-1, 1\}$.
Shortest independent vector.

Proof of
Take any

ny r_0, r_1
deg $r_0 >$ deg r_1:

nputation:
d r_1,
c.

; $q_1 = 1$;
$_{+1}\rfloor\, q_{i+1}$.
mod r_0).

e
$_1\sqrt{x})P =$
$_{-1}, r_{i+1}\sqrt{x})P$.

l $r_{i+1} = 0$.
eadcoeff r_i.

Reducing lattice basis for L is a "half gcd" computation, stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases; $\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with $\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so $\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree $\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$ for some $\epsilon \in \{-1, 1\}$.
Shortest independent vector.

Proof of "shortest

Take any $(q, r\sqrt{x})$

eg r_1:

$)P$.

.

Reducing lattice basis for L
is a "half gcd" computation,
stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with
$\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so
$\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree
$\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$
for some $\epsilon \in \{-1, 1\}$.
Shortest independent vector.

Proof of "shortest":
Take any $(q, r\sqrt{x})$ in lattice

Reducing lattice basis for L
is a "half gcd" computation,
stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with
$\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so
$\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree
$\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$
for some $\epsilon \in \{-1, 1\}$.
Shortest independent vector.

Proof of "shortest":
Take any $(q, r\sqrt{x})$ in lattice.

Reducing lattice basis for L
is a "half gcd" computation,
stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with
$\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so
$\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree
$\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$
for some $\epsilon \in \{-1, 1\}$.
Shortest independent vector.

Proof of "shortest":
Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$
for some $u, v \in P$.

Reducing lattice basis for L
is a "half gcd" computation,
stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with
$\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so
$\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree
$\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$
for some $\epsilon \in \{-1, 1\}$.
Shortest independent vector.

Proof of "shortest":
Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$
for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(r q_j - q r_j)/r_0$
and $u = \pm(q r_{j+\epsilon} - r q_{j+\epsilon})/r_0$.

Reducing lattice basis for L
is a "half gcd" computation,
stopping halfway to the gcd.

$\deg r_i$ decreases; $\deg q_i$ increases;
$\deg q_{i+1} + \deg r_i = \deg r_0$.

Say j is minimal with
$\deg r_j\sqrt{x} \le (\deg r_0)/2$.
Then $\deg q_j \le (\deg r_0)/2$ so
$\deg(q_j, r_j\sqrt{x}) \le (\deg r_0)/2$.
Shortest nonzero vector.

$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$ has degree
$\deg r_0\sqrt{x} - \deg(q_j, r_j\sqrt{x})$
for some $\epsilon \in \{-1, 1\}$.
Shortest independent vector.

Proof of "shortest":
Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$
for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(rq_j - qr_j)/r_0$
and $u = \pm(qr_{j+\epsilon} - rq_{j+\epsilon})/r_0$.

If $\deg(q, r\sqrt{x}) < \deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
then $\deg v < 0$ so $v = 0$;
i.e., any vector in lattice
shorter than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
is a multiple of $(q_j, r_j\sqrt{x})$.

g lattice basis for L

f gcd” computation,

; halfway to the gcd.

ecreases; deg q_i increases;

$+ \deg r_i = \deg r_0$.

minimal with

$x \le (\deg r_0)/2$.

g $q_j \le (\deg r_0)/2$ so

$_j\sqrt{x}) \le (\deg r_0)/2$.

nonzero vector.

$_{+\epsilon}\sqrt{x})$ has degree

$x - \deg(q_j, r_j\sqrt{x})$

e $\epsilon \in \{-1, 1\}$.

independent vector.

Proof of “shortest”:

Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$

for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(r q_j - q r_j)/r_0$
and $u = \pm(q r_{j+\epsilon} - r q_{j+\epsilon})/r_0$.

If $\deg(q, r\sqrt{x}) < \deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
then $\deg v < 0$ so $v = 0$;
i.e., any vector in lattice
shorter than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
is a multiple of $(q_j, r_j\sqrt{x})$.

Classical

Fix integ

integer r

integer t

distinct

monic g

with $g(a$

asis for L

mputation,

to the gcd.

eg q_i increases;

$= \deg r_0$.

vith

$_0)/2$.

eg $r_0)/2$ so

deg $r_0)/2$.

vector.

s degree

$_i, r_j\sqrt{x})$

$1\}$.

ent vector.

Proof of "shortest":

Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$

for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(rq_j - qr_j)/r_0$
and $u = \pm(qr_{j+\epsilon} - rq_{j+\epsilon})/r_0$.

If $\deg(q, r\sqrt{x}) < \deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
then $\deg v < 0$ so $v = 0$;
i.e., any vector in lattice
shorter than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
is a multiple of $(q_j, r_j\sqrt{x})$.

Classical binary Go

Fix integer $n \geq 0$;
integer $m \geq 1$ with
integer $t \geq 0$;
distinct $a_1, \ldots, a_n$
monic $g \in \mathbf{F}_{2^m}[x]$
with $g(a_1) \cdots g(a$

,

r

eases;

.

Proof of "shortest":

Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$

for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(rq_j - qr_j)/r_0$
and $u = \pm(qr_{j+\epsilon} - rq_{j+\epsilon})/r_0$.

If $\deg(q, r\sqrt{x}) < \deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
then $\deg v < 0$ so $v = 0$;
i.e., any vector in lattice
shorter than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
is a multiple of $(q_j, r_j\sqrt{x})$.

Classical binary Goppa code

Fix integer $n \geq 0$;
integer $m \geq 1$ with $2^m \geq n$;
integer $t \geq 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree
with $g(a_1) \cdots g(a_n) \neq 0$.

Proof of “shortest”:

Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$

for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(r q_j - q r_j)/r_0$
and $u = \pm(q r_{j+\epsilon} - r q_{j+\epsilon})/r_0$.

If $\deg(q, r\sqrt{x}) < \deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
then $\deg v < 0$ so $v = 0$;
i.e., any vector in lattice
shorter than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
is a multiple of $(q_j, r_j\sqrt{x})$.

Classical binary Goppa codes

Fix integer $n \ge 0$;
integer $m \ge 1$ with $2^m \ge n$;
integer $t \ge 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \ne 0$.

Proof of "shortest":

Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$

for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(r q_j - q r_j)/r_0$
and $u = \pm(q r_{j+\epsilon} - r q_{j+\epsilon})/r_0$.

If $\deg(q, r\sqrt{x}) < \deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
then $\deg v < 0$ so $v = 0$;
i.e., any vector in lattice
shorter than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
is a multiple of $(q_j, r_j\sqrt{x})$.

Classical binary Goppa codes

Fix integer $n \geq 0$;
integer $m \geq 1$ with $2^m \geq n$;
integer $t \geq 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \neq 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Proof of "shortest":

Take any $(q, r\sqrt{x})$ in lattice.

$$(q, r\sqrt{x}) = u(q_j, r_j\sqrt{x}) + v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$$

for some $u, v \in P$.

$q_j r_{j+\epsilon} - q_{j+\epsilon} r_j = \pm r_0$
so $v = \pm(rq_j - qr_j)/r_0$
and $u = \pm(qr_{j+\epsilon} - rq_{j+\epsilon})/r_0$.

If $\deg(q, r\sqrt{x}) < \deg(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
then $\deg v < 0$ so $v = 0$;
i.e., any vector in lattice
shorter than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$
is a multiple of $(q_j, r_j\sqrt{x})$.

Classical binary Goppa codes

Fix integer $n \geq 0$;
integer $m \geq 1$ with $2^m \geq n$;
integer $t \geq 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \neq 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \geq 2^{n-mt}$.

“shortest”:

y $(q, r\sqrt{x})$ in lattice.

$) = u(q_j, r_j\sqrt{x})$

$+ v(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$

e $u, v \in P$.

$q_{j+\epsilon} r_j = \pm r_0$

$\pm(rq_j - qr_j)/r_0$

$\pm(qr_{j+\epsilon} - rq_{j+\epsilon})/r_0$.

$, r\sqrt{x})$

deg$(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$

g $v < 0$ so $v = 0$;

vector in lattice

than $(q_{j+\epsilon}, r_{j+\epsilon}\sqrt{x})$

tiple of $(q_j, r_j\sqrt{x})$.

Classical binary Goppa codes

Fix integer $n \geq 0$;
integer $m \geq 1$ with $2^m \geq n$;
integer $t \geq 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \neq 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \geq 2^{n-mt}$.

Goal: Fi

$v = c +$

”:

) in lattice.

$_j\sqrt{x})$

$_{+\epsilon}, r_{j+\epsilon}\sqrt{x})$

.

$\pm r_0$

$r_j)/r_0$

$- r q_{j+\epsilon})/r_0$.

$r_{j+\epsilon}\sqrt{x})$

$v = 0$;

lattice

$, r_{j+\epsilon}\sqrt{x})$

$_j, r_j\sqrt{x})$.

Classical binary Goppa codes

Fix integer $n \ge 0$;
integer $m \ge 1$ with $2^m \ge n$;
integer $t \ge 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \ne 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \ge 2^{n-mt}$.

Goal: Find $c \in \Gamma$ g

$v = c + e$, assumi

.

x)

0.

Classical binary Goppa codes

Fix integer $n \geq 0$;
integer $m \geq 1$ with $2^m \geq n$;
integer $t \geq 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \neq 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \geq 2^{n-mt}$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \leq t$

Classical binary Goppa codes

Fix integer $n \geq 0$;
integer $m \geq 1$ with $2^m \geq n$;
integer $t \geq 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \neq 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \geq 2^{n-mt}$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \leq t/2$.

Classical binary Goppa codes

Fix integer $n \geq 0$;
integer $m \geq 1$ with $2^m \geq n$;
integer $t \geq 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \neq 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \geq 2^{n-mt}$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \leq t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$
to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero
$(q_j, r_j\sqrt{x})$ in the lattice $L =$
$(0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Classical binary Goppa codes

Fix integer $n \ge 0$;
integer $m \ge 1$ with $2^m \ge n$;
integer $t \ge 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \ne 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \ge 2^{n-mt}$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \le t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$
to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero
$(q_j, r_j\sqrt{x})$ in the lattice $L =$
$(0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \ne 0}(x - a_i)$ and
$E = \sum_i F e_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

Classical binary Goppa codes

Fix integer $n \ge 0$;
integer $m \ge 1$ with $2^m \ge n$;
integer $t \ge 0$;
distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;
monic $g \in \mathbf{F}_{2^m}[x]$ of degree t
with $g(a_1) \cdots g(a_n) \ne 0$.

Note that $x - a_i$
has a reciprocal in $\mathbf{F}_{2^m}[x]/g$.

Define linear subspace $\Gamma \subseteq \mathbf{F}_2^n$
as set of $(c_1, \ldots, c_n)$ with
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Then $\#\Gamma \ge 2^{n-mt}$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \le t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$
to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero
$(q_j, r_j\sqrt{x})$ in the lattice $L =$
$(0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \ne 0}(x - a_i)$ and
$E = \sum_i F e_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

$e_i = 0$ if $F(a_i) \ne 0$.
$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$.

l binary Goppa codes

ger $n \geq 0$;

$m \geq 1$ with $2^m \geq n$;

$t \geq 0$;

$a_1, \ldots, a_n \in \mathbf{F}_{2^m}$;

$\in \mathbf{F}_{2^m}[x]$ of degree t

$a_1) \cdots g(a_n) \neq 0$.

at $x - a_i$

ciprocal in $\mathbf{F}_{2^m}[x]/g$.

near subspace $\Gamma \subseteq \mathbf{F}_2^n$

f $(c_1, \ldots, c_n)$ with

$x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

$\Gamma \geq 2^{n-mt}$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \leq t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$
to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero
$(q_j, r_j\sqrt{x})$ in the lattice $L =$
$(0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \neq 0}(x - a_i)$ and
$E = \sum_i F e_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

$e_i = 0$ if $F(a_i) \neq 0$.
$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$.

This dec

"corrects

Why doe

$\sum_i e_i/(x$
$\sum_i c_i/(x$
so $s = E$
so $(F, E$

oppa codes

h $2^m \ge n$;

$\in \mathbf{F}_{2^m}$;

of degree t

$_n) \ne 0$.

$\mathbf{F}_{2^m}[x]/g$.

pace $\Gamma \subseteq \mathbf{F}_2^n$

c_n) with

0 in $\mathbf{F}_{2^m}[x]/g$.

^{t}.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \le t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$
to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero
$(q_j, r_j\sqrt{x})$ in the lattice $L =$
$(0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \ne 0}(x - a_i)$ and
$E = \sum_i F e_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

$e_i = 0$ if $F(a_i) \ne 0$.
$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$.

This decoder
"corrects $\lfloor t/2 \rfloor$ er

Why does this wor

$\sum_i e_i/(x - a_i) =$
$\sum_i c_i/(x - a_i) = ($
so $s = E/F$ in $\mathbf{F}_{2^r}$
so $(F, E\sqrt{x}) \in L$.

s

t

r.

$\mathbf{F}_2^n$

$x]/g$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \le t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$
to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero
$(q_j, r_j\sqrt{x})$ in the lattice $L =$
$(0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \ne 0}(x - a_i)$ and
$E = \sum_i Fe_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

$e_i = 0$ if $F(a_i) \ne 0$.
$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$.

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ"

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

Goal: Find $c \in \Gamma$ given
$v = c + e$, assuming $|e| \le t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$
to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero
$(q_j, r_j\sqrt{x})$ in the lattice $L =$
$(0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \ne 0}(x - a_i)$ and
$E = \sum_i F e_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

$e_i = 0$ if $F(a_i) \ne 0$.
$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$.

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

Goal: Find $c \in \Gamma$ given $v = c + e$, assuming $|e| \le t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$ to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero $(q_j, r_j\sqrt{x})$ in the lattice $L = (0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \ne 0}(x - a_i)$ and
$E = \sum_i F e_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

$e_i = 0$ if $F(a_i) \ne 0$.
$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$.

This decoder "corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \le |e| \le t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Goal: Find $c \in \Gamma$ given $v = c + e$, assuming $|e| \le t/2$.

Lift $\sum_i v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$ to $s \in \mathbf{F}_{2^m}[x]$ with $\deg s < t$.
Find shortest nonzero $(q_j, r_j\sqrt{x})$ in the lattice $L = (0, g\sqrt{x})\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

Define $E, F \in \mathbf{F}_{2^m}[x]$ by
$F = \prod_{i:e_i \ne 0}(x - a_i)$ and
$E = \sum_i F e_i/(x - a_i)$.
Fact: $E/F = r_j/q_j$ so
F is monic denominator of r_j/q_j.

$e_i = 0$ if $F(a_i) \ne 0$.
$e_i = E(a_i)/F'(a_i)$ if $F(a_i) = 0$.

This decoder "corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \le |e| \le t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":
$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,
so $E/F = r_j/q_j$. Done!

nd $c \in \Gamma$ given

e, assuming $|e| \le t/2$.

$v_i/(x - a_i)$ from $\mathbf{F}_{2^m}[x]/g$

$_{2^m}[x]$ with $\deg s < t$.

rtest nonzero

x) in the lattice $L =$

)$\mathbf{F}_{2^m}[x] + (1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

$, F \in \mathbf{F}_{2^m}[x]$ by

$_{e_i \ne 0}(x - a_i)$ and

$F e_i/(x - a_i)$.

$/F = r_j/q_j$ so

nic denominator of r_j/q_j.

$F(a_i) \ne 0$.

$a_i)/F'(a_i)$ if $F(a_i) = 0$.

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \le |e| \le t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":
$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,
so $E/F = r_j/q_j$. Done!

The squ

$\Gamma(g)$ co

$\sum_i c_i/(x$

$\sum_i c_i/(x$

given

ng $|e| \le t/2$.

) from $\mathbf{F}_{2^m}[x]/g$

n $\deg s < t$.

zero

attice $L =$

$(1, s\sqrt{x})\mathbf{F}_{2^m}[x]$.

$[x]$ by

$a_i)$ and

$a_i)$.

$_j$ so

inator of r_j/q_j.

0.

if $F(a_i) = 0$.

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \le |e| \le t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":
$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,
so $E/F = r_j/q_j$. Done!

<u>The squarefree cas</u>

$\Gamma(g)$ contains $\Gamma(g$

$\sum_i c_i/(x - a_i) = ($

$\sum_i c_i/(x - a_i) = ($

$/2.$

$m[x]/g$

$t.$

$F_{2^m}[x].$

$_j/q_j.$

$= 0.$

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \le |e| \le t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":
$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,
so $E/F = r_j/q_j$. Done!

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x$
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x$

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \le |e| \le t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":
$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,
so $E/F = r_j/q_j$. Done!

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \leq |e| \leq t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":
$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,
so $E/F = r_j/q_j$. Done!

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:
$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

This decoder

"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$

so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$

so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:

$\deg(F, E\sqrt{x}) \le |e| \le t/2$

$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":

$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,

so $E/F = r_j/q_j$. Done!

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:

$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2

corrects t errors for $\Gamma(g^2)$,

hence corrects t errors for $\Gamma(g)$.

This decoder
"corrects $\lfloor t/2 \rfloor$ errors for Γ".

Why does this work?

$\sum_i e_i/(x - a_i) = E/F$ and
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$
so $s = E/F$ in $\mathbf{F}_{2^m}[x]/g$
so $(F, E\sqrt{x}) \in L$.

$(F, E\sqrt{x})$ is a short vector:
$\deg(F, E\sqrt{x}) \le |e| \le t/2$
$< t + 1/2 - \deg(q_j, r_j\sqrt{x})$.

Recall proof of "shortest":
$(F, E\sqrt{x}) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,
so $E/F = r_j/q_j$. Done!

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:
$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2
corrects t errors for $\Gamma(g^2)$,
hence corrects t errors for $\Gamma(g)$.

(Not covered in this talk:
correcting $\approx t + t^2/n$ errors.
See, e.g., "jet list decoding".)

coder

s $\lfloor t/2 \rfloor$ errors for Γ".

es this work?

$x - a_i) = E/F$ and

$x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$

E/F in $\mathbf{F}_{2^m}[x]/g$

$\sqrt{x}) \in L$.

) is a short vector:

$E\sqrt{x}) \le |e| \le t/2$

$/2 - \deg(q_j, r_j\sqrt{x})$.

roof of "shortest":

$) \in (q_j, r_j\sqrt{x})\mathbf{F}_{2^m}[x]$,

$= r_j/q_j$. Done!

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:

$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2

corrects t errors for $\Gamma(g^2)$,

hence corrects t errors for $\Gamma(g)$.

(Not covered in this talk:

correcting $\approx t + t^2/n$ errors.

See, e.g., "jet list decoding".)

Proof: A

$\sum_i c_i/(x$

rors for Γ”.

rk?

E/F and

0 in $\mathbf{F}_{2^m}[x]/g$

$m[x]/g$

rt vector:

$| \le t/2$

$q_j, r_j\sqrt{x})$.

hortest”:

$\sqrt{x})\mathbf{F}_{2^m}[x]$,

Done!

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:

$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2

corrects t errors for $\Gamma(g^2)$,

hence corrects t errors for $\Gamma(g)$.

(Not covered in this talk:

correcting $\approx t + t^2/n$ errors.

See, e.g., “jet list decoding”.)

Proof: Assume

$\sum_i c_i/(x - a_i) =$

.

$x]/g$

$x]$,

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:
$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2
corrects t errors for $\Gamma(g^2)$,
hence corrects t errors for $\Gamma(g)$.

(Not covered in this talk:
correcting $\approx t + t^2/n$ errors.
See, e.g., "jet list decoding".)

Proof: Assume
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x$

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:
$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2
corrects t errors for $\Gamma(g^2)$,
hence corrects t errors for $\Gamma(g)$.

(Not covered in this talk:
correcting $\approx t + t^2/n$ errors.
See, e.g., "jet list decoding".)

Proof: Assume
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:
$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2
corrects t errors for $\Gamma(g^2)$,
hence corrects t errors for $\Gamma(g)$.

(Not covered in this talk:
correcting $\approx t + t^2/n$ errors.
See, e.g., "jet list decoding".)

Proof: Assume
$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.
Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$
so $F'/F = \sum c_i/(x - a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

The squarefree case

$\Gamma(g)$ contains $\Gamma(g^2)$:
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

Amazing fact:
$\Gamma(g) = \Gamma(g^2)$ if g is squarefree.

Previous decoder for g^2
corrects t errors for $\Gamma(g^2)$,
hence corrects t errors for $\Gamma(g)$.

(Not covered in this talk:
correcting $\approx t + t^2/n$ errors.
See, e.g., "jet list decoding".)

Proof: Assume
$\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x-a_i)$.
Then $F'/F = \sum_{i:c_i\neq 0} 1/(x-a_i)$
so $F'/F = \sum c_i/(x-a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:
if $F = \sum_j F_j x^j$ then

$$F' = \sum_j jF_j x^{j-1} = \sum_{j\in 1+2\mathbf{Z}} jF_j x^{j-1} = \left(\sum_{j\in 1+2\mathbf{Z}} \sqrt{jF_j}\, x^{(j-1)/2}\right)^2.$$

arefree case

ntains $\Gamma(g^2)$:

$x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$ if

$x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g^2$.

g fact:

$\Gamma(g^2)$ if g is squarefree.

s decoder for g^2

t errors for $\Gamma(g^2)$,

orrects t errors for $\Gamma(g)$.

vered in this talk:

ng $\approx t + t^2/n$ errors.

., “jet list decoding”.)

Proof: Assume

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.

Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$

so $F'/F = \sum c_i/(x - a_i)$

so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$

so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:

if $F = \sum_j F_j x^j$ then

$$F' = \sum_j jF_j x^{j-1}$$
$$= \sum_{j \in 1+2\mathbf{Z}} jF_j x^{j-1}$$
$$= (\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j} x^{(j-1)/2})^2.$$

The Mc

Standard

$t \geq 2$; m

1978 Mc

$n = 102$

This is t

$\approx 2^{60}$ pre

se

^{2}):

0 in $\mathbf{F}_{2^m}[x]/g$ if

0 in $\mathbf{F}_{2^m}[x]/g^2$.

is squarefree.

for g^2

or $\Gamma(g^2)$,

rrors for $\Gamma(g)$.

nis talk:

$^2/n$ errors.

decoding".)

Proof: Assume

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.
Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$
so $F'/F = \sum c_i/(x - a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:
if $F = \sum_j F_j x^j$ then

$$F' = \sum_j jF_j x^{j-1} = \sum_{j \in 1+2\mathbf{Z}} jF_j x^{j-1} = (\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j} x^{(j-1)/2})^2.$$

The McEliece cryp

Standardize intege

$t \geq 2$; $m \geq 1$ with

1978 McEliece exa

$n = 1024$, $m = 10$

This is too small:

$\approx 2^{60}$ pre-quantum

$x]/g$ if

$x]/g^2$.

ree.

(g).

.

.)

Proof: Assume

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.
Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$
so $F'/F = \sum c_i/(x - a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:
if $F = \sum_j F_j x^j$ then

$$F' = \sum_j jF_j x^{j-1} = \sum_{j \in 1+2\mathbf{Z}} jF_j x^{j-1} = \left(\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j} x^{(j-1)/2}\right)^2.$$

The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

Proof: Assume

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.
Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$
so $F'/F = \sum c_i/(x - a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:
if $F = \sum_j F_j x^j$ then

$$F' = \sum_j jF_j x^{j-1} = \sum_{j \in 1+2\mathbf{Z}} jF_j x^{j-1} = \left(\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j} x^{(j-1)/2}\right)^2.$$

The McEliece cryptosystem

Standardize integers $n \geq 0$;
$t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

Proof: Assume

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.
Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$
so $F'/F = \sum c_i/(x - a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:
if $F = \sum_j F_j x^j$ then

$$F' = \sum_j jF_j x^{j-1} = \sum_{j \in 1+2\mathbf{Z}} jF_j x^{j-1} = (\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j} x^{(j-1)/2})^2.$$

The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$:
$\approx 2^{87}$ pre-quantum security.

Proof: Assume

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.
Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$
so $F'/F = \sum c_i/(x - a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:
if $F = \sum_j F_j x^j$ then
$F' = \sum_j jF_j x^{j-1}$
$= \sum_{j \in 1+2\mathbf{Z}} jF_j x^{j-1}$
$= (\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j} x^{(j-1)/2})^2$.

The McEliece cryptosystem

Standardize integers $n \geq 0$;
$t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$:
$\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$:
$\approx 2^{146}$ pre-quantum security.

Proof: Assume

$\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

Write $F = \prod_{i:c_i \neq 0}(x - a_i)$.
Then $F'/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$
so $F'/F = \sum c_i/(x - a_i)$
so $F'/F = 0$ in $\mathbf{F}_{2^m}[x]/g$
so g divides F' in $\mathbf{F}_{2^m}[x]$.

F' is a square:
if $F = \sum_j F_j x^j$ then

$$F' = \sum_j jF_j x^{j-1} = \sum_{j \in 1+2\mathbf{Z}} jF_j x^{j-1} = \left(\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j} x^{(j-1)/2}\right)^2.$$

The McEliece cryptosystem

Standardize integers $n \geq 0$;
$t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$:
$\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$:
$\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$:
$\approx 2^{263}$ pre-quantum security.

Assume

$x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$.

$= \prod_{i:c_i \neq 0}(x - a_i)$.

$/F = \sum_{i:c_i \neq 0} 1/(x - a_i)$

$= \sum c_i/(x - a_i)$

$= 0$ in $\mathbf{F}_{2^m}[x]/g$

ides F' in $\mathbf{F}_{2^m}[x]$.

quare:

$\sum_j F_j x^j$ then

$_j\, jF_j x^{j-1}$

$_{j \in 1+2\mathbf{Z}}\, jF_j x^{j-1}$

$\sum_{j \in 1+2\mathbf{Z}} \sqrt{jF_j}\, x^{(j-1)/2})^2$.

The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$:
$\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$:
$\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$:
$\approx 2^{263}$ pre-quantum security.

Alice's s

$g \in \mathbf{F}_{2^m}$

distinct

0 in $\mathbf{F}_{2^m}[x]/g$.

$_0(x - a_i)$.

$_{c_i \neq 0} 1/(x - a_i)$

$x - a_i)$

$_{2^m}[x]/g$

$\mathbf{F}_{2^m}[x]$.

en

x^{j-1}

$\overline{jF_j} x^{(j-1)/2})^2$.

The McEliece cryptosystem

Standardize integers $n \geq 0$;
$t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example:
$n = 1024$, $m = 10$, $t = 50$.
This is too small:
$\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$:
$\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$:
$\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$:
$\approx 2^{263}$ pre-quantum security.

Alice's secrets: mo
$g \in \mathbf{F}_{2^m}[x]$ with d
distinct $a_1, \ldots, a_n$

$x]/g$.

$- a_i)$

$/2)^2$.

The McEliece cryptosystem

Standardize integers $n \ge 0$; $t \ge 2$; $m \ge 1$ with $2^m \ge n$.

1978 McEliece example: $n = 1024$, $m = 10$, $t = 50$. This is too small: $\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$: $\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$: $\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$: $\approx 2^{263}$ pre-quantum security.

Alice's secrets: monic irredu $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

<u>The McEliece cryptosystem</u>

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example: $n = 1024$, $m = 10$, $t = 50$. This is too small: $\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$: $\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$: $\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$: $\approx 2^{263}$ pre-quantum security.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example: $n = 1024$, $m = 10$, $t = 50$. This is too small: $\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$: $\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$: $\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$: $\approx 2^{263}$ pre-quantum security.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example: $n = 1024$, $m = 10$, $t = 50$.
This is too small: $\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$: $\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$: $\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$: $\approx 2^{263}$ pre-quantum security.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example: $n = 1024$, $m = 10$, $t = 50$. This is too small: $\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$: $\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$: $\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$: $\approx 2^{263}$ pre-quantum security.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

The McEliece cryptosystem

Standardize integers $n \geq 0$; $t \geq 2$; $m \geq 1$ with $2^m \geq n$.

1978 McEliece example: $n = 1024$, $m = 10$, $t = 50$. This is too small: $\approx 2^{60}$ pre-quantum security.

$n = 2048$, $m = 11$, $t = 32$: $\approx 2^{87}$ pre-quantum security.

$n = 3408$, $m = 12$, $t = 67$: $\approx 2^{146}$ pre-quantum security.

$n = 6960$, $m = 13$, $t = 119$: $\approx 2^{263}$ pre-quantum security.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

Eliece cryptosystem

dize integers $n \geq 0$;
$n \geq 1$ with $2^m \geq n$.

cEliece example:
4, $m = 10$, $t = 50$.
too small:
e-quantum security.

8, $m = 11$, $t = 32$:
e-quantum security.

8, $m = 12$, $t = 67$:
re-quantum security.

0, $m = 13$, $t = 119$:
re-quantum security.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

1978 Mc

Bob cho
and ranc
with $|e|$

otosystem

ers $n \geq 0$;
$2^m \geq n$.

ample:
, $t = 50$.

n security.

, $t = 32$:
n security.

, $t = 67$:
m security.

, $t = 119$:
m security.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

1978 McEliece +

Bob chooses rand
and random $e \in \mathbf{F}$
with $|e| = t$; sends

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

1978 McEliece + randomiza

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify Γ by an $(n - mt) \times n$ generator matrix G.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify Γ by an $(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

Alice's secrets: monic irreducible $g \in \mathbf{F}_{2^m}[x]$ with $\deg g = t$; distinct $a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

Note that $g(a_1) \cdots g(a_n) \neq 0$. Define Γ as before.

Alice's public key: $mt \times n$ matrix K over $\mathbf{F}_2$ such that $\Gamma = \operatorname{Ker} K$.

Bob chooses random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends Ke.

Alice receives Ke, finds $v \in \mathbf{F}_2^n$ with $Kv = Ke$, decodes v to find $v - e$.

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify Γ by an $(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G whenever $mt < n - mt$. Compress K to $mt(n - mt)$ bits by requiring systematic form.

ecrets: monic irreducible
$[x]$ with $\deg g = t$;
$a_1, \ldots, a_n \in \mathbf{F}_{2^m}$.

at $g(a_1) \cdots g(a_n) \neq 0$.
Γ as before.

public key:
matrix K over $\mathbf{F}_2$
nt $\Gamma = \operatorname{Ker} K$.

ooses random $e \in \mathbf{F}_2^n$
$= t$; sends Ke.

ceives Ke,
$\in \mathbf{F}_2^n$ with $Kv = Ke$,
v to find $v - e$.

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$
and random $e \in \mathbf{F}_2^n$
with $|e| = t$; sends $c + e$.

Publicly specify Γ by an
$(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G
whenever $mt < n - mt$.
Compress K to $mt(n - mt)$ bits
by requiring systematic form.

Does str
help atta
e.g., con

onic irreducible

eg $g = t$;

$\in \mathbf{F}_{2^m}$.

$\cdot\, g(a_n) \neq 0$.

.

over $\mathbf{F}_2$

K.

om $e \in \mathbf{F}_2^n$

s Ke.

$Kv = Ke$,

$v - e$.

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$
and random $e \in \mathbf{F}_2^n$
with $|e| = t$; sends $c + e$.

Publicly specify Γ by an
$(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G
whenever $mt < n - mt$.
Compress K to $mt(n - mt)$ bits
by requiring systematic form.

Does structure of
help attacker decr
e.g., compute g, a

ucible

0.

!

,

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify Γ by an $(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G whenever $mt < n - mt$. Compress K to $mt(n - mt)$ bits by requiring systematic form.

Does structure of Γ help attacker decrypt—e.g., compute $g, a_1, \ldots, a_n$?

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$
and random $e \in \mathbf{F}_2^n$
with $|e| = t$; sends $c + e$.

Publicly specify Γ by an
$(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G
whenever $mt < n - mt$.
Compress K to $mt(n - mt)$ bits
by requiring systematic form.

Does structure of Γ
help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify Γ by an $(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G whenever $mt < n - mt$.
Compress K to $mt(n - mt)$ bits by requiring systematic form.

Does structure of Γ help attacker decrypt— e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks" are much slower than information-set decoding.
(Less conservative variants of McEliece encourage research.)

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$
and random $e \in \mathbf{F}_2^n$
with $|e| = t$; sends $c + e$.

Publicly specify Γ by an
$(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G
whenever $mt < n - mt$.
Compress K to $mt(n - mt)$ bits
by requiring systematic form.

Does structure of Γ
help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks"
are much slower than
information-set decoding.
(Less conservative variants of
McEliece encourage research.)

Does K leak more than Γ?

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$ and random $e \in \mathbf{F}_2^n$ with $|e| = t$; sends $c + e$.

Publicly specify Γ by an $(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G whenever $mt < n - mt$. Compress K to $mt(n - mt)$ bits by requiring systematic form.

Does structure of Γ help attacker decrypt— e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks" are much slower than information-set decoding. (Less conservative variants of McEliece encourage research.)

Does K leak more than Γ?

No with 1978 McEliece: matrix is explicitly randomized.

1978 McEliece + randomization:

Bob chooses random $c \in \Gamma$
and random $e \in \mathbf{F}_2^n$
with $|e| = t$; sends $c + e$.

Publicly specify Γ by an
$(n - mt) \times n$ generator matrix G.

1986 Niederreiter improvements:

Send Ke instead of $c + e$.

K is smaller than G
whenever $mt < n - mt$.
Compress K to $mt(n - mt)$ bits
by requiring systematic form.

Does structure of Γ
help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks"
are much slower than
information-set decoding.
(Less conservative variants of
McEliece encourage research.)

Does K leak more than Γ?

No with 1978 McEliece:
matrix is explicitly randomized.

No with 1986 Niederreiter:
matrix has systematic form.

cEliece + randomization:

oses random $c \in \Gamma$
dom $e \in \mathbf{F}_2^n$
$= t$; sends $c + e$.

specify Γ by an
$) \times n$ generator matrix G.

ederreiter improvements:

e instead of $c + e$.

aller than G
er $mt < n - mt$.
ss K to $mt(n - mt)$ bits
ring systematic form.

Does structure of Γ
help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks"
are much slower than
information-set decoding.
(Less conservative variants of
McEliece encourage research.)

Does K leak more than Γ?

No with 1978 McEliece:
matrix is explicitly randomized.

No with 1986 Niederreiter:
matrix has systematic form.

Better th

Rest of t
with Cho
some de
McEliece

randomization:

om $c \in \Gamma$
$_2^n$
s $c + e$.

by an
erator matrix G.

improvements:

of $c + e$.

G

$- mt$.
$t(n - mt)$ bits
matic form.

Does structure of Γ
help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?

All known “structural attacks”
are much slower than
information-set decoding.
(Less conservative variants of
McEliece encourage research.)

Does K leak more than Γ?

No with 1978 McEliece:
matrix is explicitly randomized.

No with 1986 Niederreiter:
matrix has systematic form.

Better throughput

Rest of this talk (j
with Chou and Sc
some details of ho
McEliece run reall

tion:

rix G.

ents:

bits

n.

Does structure of Γ
help attacker decrypt—
e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks"
are much slower than
information-set decoding.
(Less conservative variants of
McEliece encourage research.)

Does K leak more than Γ?

No with 1978 McEliece:
matrix is explicitly randomized.

No with 1986 Niederreiter:
matrix has systematic form.

Better throughput than ECC

Rest of this talk (joint work
with Chou and Schwabe, 20
some details of how to make
McEliece run really fast.

Does structure of Γ help attacker decrypt— e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks" are much slower than information-set decoding. (Less conservative variants of McEliece encourage research.)

Does K leak more than Γ?

No with 1978 McEliece: matrix is explicitly randomized.

No with 1986 Niederreiter: matrix has systematic form.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Does structure of Γ help attacker decrypt— e.g., compute $g, a_1, \ldots, a_n$?

All known "structural attacks" are much slower than information-set decoding. (Less conservative variants of McEliece encourage research.)

Does K leak more than Γ?

No with 1978 McEliece: matrix is explicitly randomized.

No with 1986 Niederreiter: matrix has systematic form.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for $(n, t) = (2048, 32)$; $\approx 2^{87}$.

79715 Ivy Bridge cycles for $(n, t) = (3408, 67)$; $\approx 2^{146}$.

306102 Ivy Bridge cycles for $(n, t) = (6960, 119)$; $\approx 2^{263}$.

ructure of Γ
acker decrypt—
mpute $g, a_1, \ldots, a_n$?

wn "structural attacks"
h slower than
tion-set decoding.
nservative variants of
e encourage research.)

leak more than Γ?

1978 McEliece:
s explicitly randomized.

1986 Niederreiter:
nas systematic form.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for $(n,t) = (2048, 32)$; $\approx 2^{87}$.

79715 Ivy Bridge cycles for $(n,t) = (3408, 67)$; $\approx 2^{146}$.

306102 Ivy Bridge cycles for $(n,t) = (6960, 119)$; $\approx 2^{263}$.

The add

Fix $n =$

Big fina
is to fin
of $F =$

For each
compute
41 adds,

Γ

ypt—

$_1, \ldots, a_n$?

ural attacks”

han

coding.

variants of

ge research.)

than Γ?

Eliece:

randomized.

derreiter:

atic form.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for $(n, t) = (2048, 32)$; $\approx 2^{87}$.

79715 Ivy Bridge cycles for $(n, t) = (3408, 67)$; $\approx 2^{146}$.

306102 Ivy Bridge cycles for $(n, t) = (6960, 119)$; $\approx 2^{263}$.

The additive FFT

Fix $n = 4096 = 2^1$

Big final decoding

is to find all roots

of $F = F_{41}x^{41} + \cdot$

For each $\alpha \in \mathbf{F}_{2^{12}}$

compute $F(\alpha)$ by

41 adds, 41 mults.

ks”

of

n.)

ed.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for $(n, t) = (2048, 32)$; $\approx 2^{87}$.

79715 Ivy Bridge cycles for $(n, t) = (3408, 67)$; $\approx 2^{146}$.

306102 Ivy Bridge cycles for $(n, t) = (6960, 119)$; $\approx 2^{263}$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41.$

Big final decoding step is to find all roots in $\mathbf{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0x^0$

For each $\alpha \in \mathbf{F}_{2^{12}}$, compute $F(\alpha)$ by Horner's r

41 adds, 41 mults.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for $(n, t) = (2048, 32)$; $\approx 2^{87}$.

79715 Ivy Bridge cycles for $(n, t) = (3408, 67)$; $\approx 2^{146}$.

306102 Ivy Bridge cycles for $(n, t) = (6960, 119)$; $\approx 2^{263}$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step is to find all roots in $\mathbf{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$, compute $F(\alpha)$ by Horner's rule: 41 adds, 41 mults.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for $(n, t) = (2048, 32)$; $\approx 2^{87}$.

79715 Ivy Bridge cycles for $(n, t) = (3408, 67)$; $\approx 2^{146}$.

306102 Ivy Bridge cycles for $(n, t) = (6960, 119)$; $\approx 2^{263}$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step is to find all roots in $\mathbf{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$, compute $F(\alpha)$ by Horner's rule: 41 adds, 41 mults.

Or use "Chien search": compute $F_i\gamma^i$, $F_i\gamma^{2i}$, $F_i\gamma^{3i}$, etc. Cost per point: again 41 adds, 41 mults.

Better throughput than ECC

Rest of this talk (joint work with Chou and Schwabe, 2013): some details of how to make McEliece run really fast.

Our constant-time software for batches of 256 decodings:

26544 Ivy Bridge cycles for $(n, t) = (2048, 32)$; $\approx 2^{87}$.

79715 Ivy Bridge cycles for $(n, t) = (3408, 67)$; $\approx 2^{146}$.

306102 Ivy Bridge cycles for $(n, t) = (6960, 119)$; $\approx 2^{263}$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step is to find all roots in $\mathbf{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0 x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$, compute $F(\alpha)$ by Horner's rule: 41 adds, 41 mults.

Or use "Chien search": compute $F_i\gamma^i$, $F_i\gamma^{2i}$, $F_i\gamma^{3i}$, etc. Cost per point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

hroughput than ECC

this talk (joint work
ou and Schwabe, 2013):
tails of how to make
e run really fast.

stant-time software
hes of 256 decodings:

vy Bridge cycles for
$(2048, 32)$; $\approx 2^{87}$.

vy Bridge cycles for
$(3408, 67)$; $\approx 2^{146}$.

Ivy Bridge cycles for
$(6960, 119)$; $\approx 2^{263}$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step
is to find all roots in $\mathbf{F}_{2^{12}}$
of $F = F_{41}x^{41} + \cdots + F_0 x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,
compute $F(\alpha)$ by Horner's rule:
41 adds, 41 mults.

Or use "Chien search": compute
$F_i \gamma^i$, $F_i \gamma^{2i}$, $F_i \gamma^{3i}$, etc. Cost per
point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asympto
normally
so Horne
$\Theta(nt) =$

than ECC

joint work
hwabe, 2013):
w to make
y fast.

software
decodings:

cycles for
; $\approx 2^{87}$.

cycles for
; $\approx 2^{146}$.

e cycles for
); $\approx 2^{263}$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step
is to find all roots in $\mathbf{F}_{2^{12}}$
of $F = F_{41}x^{41} + \cdots + F_0 x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,
compute $F(\alpha)$ by Horner's rule:
41 adds, 41 mults.

Or use "Chien search": compute
$F_i\gamma^i$, $F_i\gamma^{2i}$, $F_i\gamma^{3i}$, etc. Cost per
point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics:
normally $t \in \Theta(n/$
so Horner's rule co
$\Theta(nt) = \Theta(n^2/\lg$

C

13):

e

s:

r

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step is to find all roots in $\mathbf{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$, compute $F(\alpha)$ by Horner's rule: 41 adds, 41 mults.

Or use "Chien search": compute $F_i\gamma^i$, $F_i\gamma^{2i}$, $F_i\gamma^{3i}$, etc. Cost per point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics:
normally $t \in \Theta(n/\lg n)$, so Horner's rule costs $\Theta(nt) = \Theta(n^2/\lg n)$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step is to find all roots in $\mathbf{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$, compute $F(\alpha)$ by Horner's rule: 41 adds, 41 mults.

Or use "Chien search": compute $F_i\gamma^i$, $F_i\gamma^{2i}$, $F_i\gamma^{3i}$, etc. Cost per point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics: normally $t \in \Theta(n/\lg n)$, so Horner's rule costs $\Theta(nt) = \Theta(n^2/\lg n)$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step is to find all roots in $\mathbf{F}_{2^{12}}$ of $F = F_{41}x^{41} + \cdots + F_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$, compute $F(\alpha)$ by Horner's rule: 41 adds, 41 mults.

Or use "Chien search": compute $F_i\gamma^i$, $F_i\gamma^{2i}$, $F_i\gamma^{3i}$, etc. Cost per point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics:
normally $t \in \Theta(n/\lg n)$, so Horner's rule costs $\Theta(nt) = \Theta(n^2/\lg n)$.

Wait a minute.
Didn't we learn in school that FFT evaluates an n-coeff polynomial at n points using $n^{1+o(1)}$ operations?
Isn't this better than $n^2/\lg n$?

itive FFT

4096 = 2^{12}, $t = 41$.

decoding step
d all roots in $\mathbf{F}_{2^{12}}$
$F_{41}x^{41} + \cdots + F_0 x^0$.

$\alpha \in \mathbf{F}_{2^{12}}$,
e $F(\alpha)$ by Horner's rule:
41 mults.

"Chien search": compute
γ^{2i}, $F_i\gamma^{3i}$, etc. Cost per
gain 41 adds, 41 mults.

t: **6.01** adds, **2.09** mults.

Asymptotics:
normally $t \in \Theta(n/\lg n)$,
so Horner's rule costs
$\Theta(nt) = \Theta(n^2/\lg n)$.

Wait a minute.
Didn't we learn in school
that FFT evaluates
an n-coeff polynomial
at n points
using $n^{1+o(1)}$ operations?
Isn't this better than $n^2/\lg n$?

Standard

Want to
$F = F_0$
at all th

Write F
Observe
$F(\alpha) =$
$F(-\alpha)$

F_0 has n
evaluate
by same
Similarly

^{2}, $t = 41$.

step

in $\mathbf{F}_{2^{12}}$

$\cdots + F_0 x^0$.

,

Horner's rule:

.

rch": compute

etc. Cost per

dds, 41 mults.

lds, **2.09** mults.

Asymptotics:
normally $t \in \Theta(n/\lg n)$,
so Horner's rule costs
$\Theta(nt) = \Theta(n^2/\lg n)$.

Wait a minute.
Didn't we learn in school
that FFT evaluates
an n-coeff polynomial
at n points
using $n^{1+o(1)}$ operations?
Isn't this better than $n^2/\lg n$?

Standard radix-2 F

Want to evaluate
$F = F_0 + F_1 x + \cdot$
at all the nth root

Write F as $F_0(x^2)$
Observe big overla
$F(\alpha) = F_0(\alpha^2) +$
$F(-\alpha) = F_0(\alpha^2)$ -

F_0 has $n/2$ coeffs;
evaluate at $(n/2)$r
by same idea recu
Similarly F_1.

.

rule:

npute

t per

ults.

mults.

Asymptotics:

normally $t \in \Theta(n/\lg n)$,
so Horner's rule costs
$\Theta(nt) = \Theta(n^2/\lg n)$.

Wait a minute.
Didn't we learn in school
that FFT evaluates
an n-coeff polynomial
at n points
using $n^{1+o(1)}$ operations?
Isn't this better than $n^2/\lg n$?

Standard radix-2 FFT:

Want to evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1}x$
at all the nth roots of 1.

Write F as $F_0(x^2) + xF_1(x^2$
Observe big overlap between
$F(\alpha) = F_0(\alpha^2) + \alpha F_1(\alpha^2)$,
$F(-\alpha) = F_0(\alpha^2) - \alpha F_1(\alpha^2)$

F_0 has $n/2$ coeffs;
evaluate at $(n/2)$nd roots of
by same idea recursively.
Similarly F_1.

Asymptotics:
normally $t \in \Theta(n/\lg n)$,
so Horner's rule costs
$\Theta(nt) = \Theta(n^2/\lg n)$.

Wait a minute.
Didn't we learn in school
that FFT evaluates
an n-coeff polynomial
at n points
using $n^{1+o(1)}$ operations?
Isn't this better than $n^2/\lg n$?

Standard radix-2 FFT:

Want to evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
at all the nth roots of 1.

Write F as $F_0(x^2) + xF_1(x^2)$.
Observe big overlap between
$F(\alpha) = F_0(\alpha^2) + \alpha F_1(\alpha^2)$,
$F(-\alpha) = F_0(\alpha^2) - \alpha F_1(\alpha^2)$.

F_0 has $n/2$ coeffs;
evaluate at $(n/2)$nd roots of 1
by same idea recursively.
Similarly F_1.

otics:

y $t \in \Theta(n/\lg n)$,

er's rule costs

$= \Theta(n^2/\lg n)$.

minute.

ve learn in school

T evaluates

eff polynomial

nts

$^{+o(1)}$ operations?

s better than $n^2/\lg n$?

Standard radix-2 FFT:

Want to evaluate

$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$

at all the nth roots of 1.

Write F as $F_0(x^2) + x F_1(x^2)$.

Observe big overlap between

$F(\alpha) = F_0(\alpha^2) + \alpha F_1(\alpha^2)$,

$F(-\alpha) = F_0(\alpha^2) - \alpha F_1(\alpha^2)$.

F_0 has $n/2$ coeffs;

evaluate at $(n/2)$nd roots of 1

by same idea recursively.

Similarly F_1.

Useless i

Standard

FFT con

1988 Wa

independ

"additive

Still quit

1996 von

some im

2010 Ga

much be

We use

plus som

$/ \lg n$),
osts
n).

school
es
mial

rations?
han $n^2/\lg n$?

Standard radix-2 FFT:

Want to evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
at all the nth roots of 1.

Write F as $F_0(x^2) + x F_1(x^2)$.
Observe big overlap between
$F(\alpha) = F_0(\alpha^2) + \alpha F_1(\alpha^2)$,
$F(-\alpha) = F_0(\alpha^2) - \alpha F_1(\alpha^2)$.

F_0 has $n/2$ coeffs;
evaluate at $(n/2)$nd roots of 1
by same idea recursively.
Similarly F_1.

Useless in char 2:
Standard workarou
FFT considered im

1988 Wang–Zhu,
independently 198
"additive FFT" in
Still quite expensiv

1996 von zur Gath
some improvemen

2010 Gao–Mateer:
much better addit

We use Gao–Mate
plus some new imp

n?

Standard radix-2 FFT:

Want to evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
at all the nth roots of 1.

Write F as $F_0(x^2) + xF_1(x^2)$.
Observe big overlap between
$F(\alpha) = F_0(\alpha^2) + \alpha F_1(\alpha^2)$,
$F(-\alpha) = F_0(\alpha^2) - \alpha F_1(\alpha^2)$.

F_0 has $n/2$ coeffs;
evaluate at $(n/2)$nd roots of 1
by same idea recursively.
Similarly F_1.

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are p
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
“additive FFT” in char 2.
Still quite expensive.

1996 von zur Gathen–Gerha
some improvements.

2010 Gao–Mateer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvement

Standard radix-2 FFT:

Want to evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
at all the nth roots of 1.

Write F as $F_0(x^2) + xF_1(x^2)$.
Observe big overlap between
$F(\alpha) = F_0(\alpha^2) + \alpha F_1(\alpha^2)$,
$F(-\alpha) = F_0(\alpha^2) - \alpha F_1(\alpha^2)$.

F_0 has $n/2$ coeffs;
evaluate at $(n/2)$nd roots of 1
by same idea recursively.
Similarly F_1.

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are painful.
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
"additive FFT" in char 2.
Still quite expensive.

1996 von zur Gathen–Gerhard:
some improvements.

2010 Gao–Mateer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvements.

d radix-2 FFT:

evaluate

$+ F_1 x + \cdots + F_{n-1} x^{n-1}$

e nth roots of 1.

as $F_0(x^2) + x F_1(x^2)$.

big overlap between

$F_0(\alpha^2) + \alpha F_1(\alpha^2)$,

$= F_0(\alpha^2) - \alpha F_1(\alpha^2)$.

$n/2$ coeffs;

at $(n/2)$nd roots of 1

idea recursively.

F_1.

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are painful.
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
"additive FFT" in char 2.
Still quite expensive.

1996 von zur Gathen–Gerhard:
some improvements.

2010 Gao–Mateer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvements.

Gao and

$F = F_0 +$

on a size

Main ide

$F_0(x^2 +$

Big over

$F_0(\alpha^2 +$

and $F(\alpha$

$F_0(\alpha^2 +$

"Twist"

Then $\{\alpha$

size-$(n/2$

Apply sa

FFT:

$\cdots + F_{n-1}x^{n-1}$

s of 1.

$+ xF_1(x^2)$.

ap between

$\alpha F_1(\alpha^2)$,

$- \alpha F_1(\alpha^2)$.

nd roots of 1

rsively.

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are painful.
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
"additive FFT" in char 2.
Still quite expensive.

1996 von zur Gathen–Gerhard:
some improvements.

2010 Gao–Mateeer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvements.

Gao and Mateer e

$F = F_0 + F_1x + \cdot$

on a size-n $\mathbf{F}_2$-line

Main idea: Write

$F_0(x^2 + x) + xF_1($

Big overlap betwe

$F_0(\alpha^2 + \alpha) + \alpha F_1$

and $F(\alpha + 1) =$

$F_0(\alpha^2 + \alpha) + (\alpha -$

"Twist" to ensure

Then $\{\alpha^2 + \alpha\}$ is

size-$(n/2)$ $\mathbf{F}_2$-linea

Apply same idea r

x^{n-1}

).

n

).

f 1

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are painful.
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
"additive FFT" in char 2.
Still quite expensive.

1996 von zur Gathen–Gerhard:
some improvements.

2010 Gao–Mateer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvements.

Gao and Mateer evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1}x$
on a size-n $\mathbf{F}_2$-linear space.

Main idea: Write F as
$F_0(x^2 + x) + xF_1(x^2 + x)$.

Big overlap between $F(\alpha) =$
$F_0(\alpha^2 + \alpha) + \alpha F_1(\alpha^2 + \alpha)$
and $F(\alpha + 1) =$
$F_0(\alpha^2 + \alpha) + (\alpha + 1)F_1(\alpha^2$

"Twist" to ensure $1 \in$ space
Then $\{\alpha^2 + \alpha\}$ is a
size-$(n/2)$ $\mathbf{F}_2$-linear space.
Apply same idea recursively.

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are painful.
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
"additive FFT" in char 2.
Still quite expensive.

1996 von zur Gathen–Gerhard:
some improvements.

2010 Gao–Mateer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvements.

Gao and Mateer evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
on a size-n $\mathbf{F}_2$-linear space.

Main idea: Write F as
$F_0(x^2 + x) + x F_1(x^2 + x)$.

Big overlap between $F(\alpha) =$
$F_0(\alpha^2 + \alpha) + \alpha F_1(\alpha^2 + \alpha)$
and $F(\alpha + 1) =$
$F_0(\alpha^2 + \alpha) + (\alpha + 1) F_1(\alpha^2 + \alpha)$.

"Twist" to ensure $1 \in$ space.
Then $\{\alpha^2 + \alpha\}$ is a
size-$(n/2)$ $\mathbf{F}_2$-linear space.
Apply same idea recursively.

in char 2: $\alpha = -\alpha$.
d workarounds are painful.
nsidered impractical.

ang–Zhu,
dently 1989 Cantor:
e FFT" in char 2.
ce expensive.

n zur Gathen–Gerhard:
provements.

o–Mateer:
etter additive FFT.

Gao–Mateer,
ne new improvements.

Gao and Mateer evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
on a size-n $\mathbf{F}_2$-linear space.

Main idea: Write F as
$F_0(x^2 + x) + x F_1(x^2 + x)$.

Big overlap between $F(\alpha) =$
$F_0(\alpha^2 + \alpha) + \alpha F_1(\alpha^2 + \alpha)$
and $F(\alpha + 1) =$
$F_0(\alpha^2 + \alpha) + (\alpha + 1) F_1(\alpha^2 + \alpha)$.

"Twist" to ensure $1 \in$ space.
Then $\{\alpha^2 + \alpha\}$ is a
size-$(n/2)$ $\mathbf{F}_2$-linear space.
Apply same idea recursively.

We gene
$F = F_0$
for any

$\Rightarrow$ sever
not all o
by simpl

For $t =$

For $t \in$
F_1 is a c
Instead
this cons
multiply
and com

$\alpha = -\alpha$.
unds are painful.
npractical.

9 Cantor:
char 2.
ve.

hen–Gerhard:
ts.

:
ive FFT.

eer,
provements.

Gao and Mateer evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
on a size-n $\mathbf{F}_2$-linear space.

Main idea: Write F as
$F_0(x^2 + x) + x F_1(x^2 + x)$.

Big overlap between $F(\alpha) =$
$F_0(\alpha^2 + \alpha) + \alpha F_1(\alpha^2 + \alpha)$
and $F(\alpha + 1) =$
$F_0(\alpha^2 + \alpha) + (\alpha + 1) F_1(\alpha^2 + \alpha)$.

"Twist" to ensure $1 \in$ space.
Then $\{\alpha^2 + \alpha\}$ is a
size-$(n/2)$ $\mathbf{F}_2$-linear space.
Apply same idea recursively.

We generalize to
$F = F_0 + F_1 x + \cdot$
for any $t < n$.

$\Rightarrow$ several optimiz
not all of which ar
by simply tracking

For $t = 0$: copy F

For $t \in \{1, 2\}$:
F_1 is a constant.
Instead of multiply
this constant by ea
multiply only by g
and compute subs

ainful.

rd:

ts.

Gao and Mateer evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
on a size-n $\mathbf{F}_2$-linear space.

Main idea: Write F as
$F_0(x^2 + x) + x F_1(x^2 + x)$.

Big overlap between $F(\alpha) =$
$F_0(\alpha^2 + \alpha) + \alpha F_1(\alpha^2 + \alpha)$
and $F(\alpha + 1) =$
$F_0(\alpha^2 + \alpha) + (\alpha + 1) F_1(\alpha^2 + \alpha)$.

"Twist" to ensure $1 \in$ space.
Then $\left\{\alpha^2 + \alpha\right\}$ is a
size-$(n/2)$ $\mathbf{F}_2$-linear space.
Apply same idea recursively.

We generalize to
$F = F_0 + F_1 x + \cdots + F_t x^t$
for any $t < n$.

$\Rightarrow$ several optimizations,
not all of which are automat
by simply tracking zeros.

For $t = 0$: copy F_0.

For $t \in \{1, 2\}$:
F_1 is a constant.
Instead of multiplying
this constant by each α,
multiply only by generators
and compute subset sums.

Gao and Mateer evaluate
$F = F_0 + F_1 x + \cdots + F_{n-1} x^{n-1}$
on a size-n $\mathbf{F}_2$-linear space.

Main idea: Write F as
$F_0(x^2 + x) + x F_1(x^2 + x)$.

Big overlap between $F(\alpha) =$
$F_0(\alpha^2 + \alpha) + \alpha F_1(\alpha^2 + \alpha)$
and $F(\alpha + 1) =$
$F_0(\alpha^2 + \alpha) + (\alpha + 1) F_1(\alpha^2 + \alpha)$.

"Twist" to ensure $1 \in$ space.
Then $\{\alpha^2 + \alpha\}$ is a
size-$(n/2)$ $\mathbf{F}_2$-linear space.
Apply same idea recursively.

We generalize to
$F = F_0 + F_1 x + \cdots + F_t x^t$
for any $t < n$.

$\Rightarrow$ several optimizations,
not all of which are automated
by simply tracking zeros.

For $t = 0$: copy F_0.

For $t \in \{1, 2\}$:
F_1 is a constant.
Instead of multiplying
this constant by each α,
multiply only by generators
and compute subset sums.

Mateer evaluate

$+ F_1 x + \cdots + F_{n-1}x^{n-1}$

e-n $\mathbf{F}_2$-linear space.

ea: Write F as

$x) + xF_1(x^2 + x)$.

rlap between $F(\alpha) =$

$\alpha) + \alpha F_1(\alpha^2 + \alpha)$

$\alpha + 1) =$

$\alpha) + (\alpha + 1)F_1(\alpha^2 + \alpha)$.

to ensure $1 \in$ space.

$\alpha^2 + \alpha\}$ is a

2) $\mathbf{F}_2$-linear space.

ame idea recursively.

We generalize to

$F = F_0 + F_1 x + \cdots + F_t x^t$

for any $t < n$.

$\Rightarrow$ several optimizations,

not all of which are automated

by simply tracking zeros.

For $t = 0$: copy F_0.

For $t \in \{1, 2\}$:

F_1 is a constant.

Instead of multiplying

this constant by each α,

multiply only by generators

and compute subset sums.

Syndrom

Initial de

$s_0 = r_1$ -

$s_1 = r_1\alpha$

$s_2 = r_1\alpha$

$\vdots$,

$s_t = r_1\alpha$

$r_1, r_2, \ldots$

scaled by

Typically

mapping

Not as s

still n^{2+}

valuate

$\cdots + F_{n-1}x^{n-1}$

ear space.

F as

$(x^2 + x)$.

en $F(\alpha) =$

$(\alpha^2 + \alpha)$

$+ 1)F_1(\alpha^2 + \alpha)$.

$1 \in$ space.

a

ar space.

ecursively.

We generalize to

$F = F_0 + F_1 x + \cdots + F_t x^t$

for any $t < n$.

$\Rightarrow$ several optimizations, not all of which are automated by simply tracking zeros.

For $t = 0$: copy F_0.

For $t \in \{1, 2\}$:

F_1 is a constant.

Instead of multiplying this constant by each α, multiply only by generators and compute subset sums.

Syndrome comput

Initial decoding st

$s_0 = r_1 + r_2 + \cdots$

$s_1 = r_1\alpha_1 + r_2\alpha_2$

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2$

$\vdots$,

$s_t = r_1\alpha_1^t + r_2\alpha_2^t$

$r_1, r_2, \ldots, r_n$ are re

scaled by Goppa c

Typically precomp

mapping bits to sy

Not as slow as Ch

still $n^{2+o(1)}$ and h

x^{n-1}

=

$+ \alpha)$.

e.

We generalize to
$F = F_0 + F_1 x + \cdots + F_t x^t$
for any $t < n$.

$\Rightarrow$ several optimizations,
not all of which are automated
by simply tracking zeros.

For $t = 0$: copy F_0.

For $t \in \{1, 2\}$:
F_1 is a constant.
Instead of multiplying
this constant by each α,
multiply only by generators
and compute subset sums.

Syndrome computation

Initial decoding step: compu

$s_0 = r_1 + r_2 + \cdots + r_n$,
$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n$
$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n$
$\vdots$,
$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n$

$r_1, r_2, \ldots, r_n$ are received bit
scaled by Goppa constants.
Typically precompute matrix
mapping bits to syndrome.
Not as slow as Chien search
still $n^{2+o(1)}$ and huge secret

We generalize to

$F = F_0 + F_1 x + \cdots + F_t x^t$

for any $t < n$.

$\Rightarrow$ several optimizations, not all of which are automated by simply tracking zeros.

For $t = 0$: copy F_0.

For $t \in \{1, 2\}$: F_1 is a constant. Instead of multiplying this constant by each α, multiply only by generators and compute subset sums.

Syndrome computation

Initial decoding step: compute

$s_0 = r_1 + r_2 + \cdots + r_n$,

$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n\alpha_n$,

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2$,

$\vdots$,

$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n\alpha_n^t$.

$r_1, r_2, \ldots, r_n$ are received bits scaled by Goppa constants.

Typically precompute matrix mapping bits to syndrome.

Not as slow as Chien search but still $n^{2+o(1)}$ and huge secret key.

eralize to

$+ F_1 x + \cdots + F_t x^t$

$t < n$.

al optimizations,

f which are automated

y tracking zeros.

0: copy F_0.

$\{1, 2\}$:

constant.

of multiplying

stant by each α,

only by generators

npute subset sums.

Syndrome computation

Initial decoding step: compute

$s_0 = r_1 + r_2 + \cdots + r_n$,

$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n\alpha_n$,

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2$,

$\vdots$,

$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n\alpha_n^t$.

$r_1, r_2, \ldots, r_n$ are received bits scaled by Goppa constants.

Typically precompute matrix mapping bits to syndrome.

Not as slow as Chien search but still $n^{2+o(1)}$ and huge secret key.

Compare

$F(\alpha_1) =$

$F(\alpha_2) =$

$\vdots$,

$F(\alpha_n) =$

$\cdots + F_t x^t$

ations,

re automated

zeros.

$_0$.

ying

ach α,

enerators

et sums.

Syndrome computation

Initial decoding step: compute

$s_0 = r_1 + r_2 + \cdots + r_n$,

$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n\alpha_n$,

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2$,

$\vdots$,

$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n\alpha_n^t$.

$r_1, r_2, \ldots, r_n$ are received bits scaled by Goppa constants.

Typically precompute matrix mapping bits to syndrome.

Not as slow as Chien search but still $n^{2+o(1)}$ and huge secret key.

Compare to multip

$F(\alpha_1) = F_0 + F_1\alpha$

$F(\alpha_2) = F_0 + F_1\alpha$

$\vdots$,

$F(\alpha_n) = F_0 + F_1\alpha$

ted

Syndrome computation

Initial decoding step: compute

$s_0 = r_1 + r_2 + \cdots + r_n,$

$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n\alpha_n,$

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2,$

$\vdots,$

$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n\alpha_n^t.$

$r_1, r_2, \ldots, r_n$ are received bits scaled by Goppa constants.

Typically precompute matrix mapping bits to syndrome.

Not as slow as Chien search but still $n^{2+o(1)}$ and huge secret key.

Compare to multipoint evalu

$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots +$

$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots +$

$\vdots,$

$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots +$

Syndrome computation

Initial decoding step: compute

$s_0 = r_1 + r_2 + \cdots + r_n,$

$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n\alpha_n,$

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2,$

$\vdots,$

$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n\alpha_n^t.$

$r_1, r_2, \ldots, r_n$ are received bits scaled by Goppa constants.

Typically precompute matrix mapping bits to syndrome.

Not as slow as Chien search but still $n^{2+o(1)}$ and huge secret key.

Compare to multipoint evaluation:

$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t,$

$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t,$

$\vdots,$

$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t.$

Syndrome computation

Initial decoding step: compute

$s_0 = r_1 + r_2 + \cdots + r_n$,

$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n\alpha_n$,

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2$,

$\vdots$,

$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n\alpha_n^t$.

$r_1, r_2, \ldots, r_n$ are received bits scaled by Goppa constants.

Typically precompute matrix mapping bits to syndrome.

Not as slow as Chien search but still $n^{2+o(1)}$ and huge secret key.

Compare to multipoint evaluation:

$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t$,

$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t$,

$\vdots$,

$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t$.

Matrix for syndrome computation is transpose of matrix for multipoint evaluation.

Syndrome computation

Initial decoding step: compute

$s_0 = r_1 + r_2 + \cdots + r_n,$

$s_1 = r_1\alpha_1 + r_2\alpha_2 + \cdots + r_n\alpha_n,$

$s_2 = r_1\alpha_1^2 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2,$

$\vdots,$

$s_t = r_1\alpha_1^t + r_2\alpha_2^t + \cdots + r_n\alpha_n^t.$

$r_1, r_2, \ldots, r_n$ are received bits scaled by Goppa constants.

Typically precompute matrix mapping bits to syndrome.

Not as slow as Chien search but still $n^{2+o(1)}$ and huge secret key.

Compare to multipoint evaluation:

$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t,$

$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t,$

$\vdots,$

$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t.$

Matrix for syndrome computation is transpose of matrix for multipoint evaluation.

Amazing consequence: syndrome computation is as few ops as multipoint evaluation. Eliminate precomputed matrix.

ne computation

ecoding step: compute

$+ r_2 + \cdots + r_n,$

$_1 + r_2\alpha_2 + \cdots + r_n\alpha_n,$

$^2_1 + r_2\alpha_2^2 + \cdots + r_n\alpha_n^2,$

$^t_1 + r_2\alpha_2^t + \cdots + r_n\alpha_n^t.$

$., r_n$ are received bits

y Goppa constants.

y precompute matrix

g bits to syndrome.

slow as Chien search but

$o(1)$ and huge secret key.

Compare to multipoint evaluation:

$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t,$
$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t,$
$\vdots,$
$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t.$

Matrix for syndrome computation is transpose of matrix for multipoint evaluation.

Amazing consequence: syndrome computation is as few ops as multipoint evaluation. Eliminate precomputed matrix.

Transpo

If a linea

compute

then rev

exchang

compute

1956 Bo

independ

for Bool

1973 Fi

preserve

preserve

number

ation

ep: compute

$+ r_n,$

$+ \cdots + r_n\alpha_n,$

$+ \cdots + r_n\alpha_n^2,$

$+ \cdots + r_n\alpha_n^t.$

eceived bits

onstants.

ute matrix

yndrome.

ien search but

uge secret key.

Compare to multipoint evaluation:

$$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t,$$
$$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t,$$
$$\vdots,$$
$$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t.$$

Matrix for syndrome computation is transpose of matrix for multipoint evaluation.

Amazing consequence: syndrome computation is as few ops as multipoint evaluation. Eliminate precomputed matrix.

Transposition prin

If a linear algorith

computes a matri

then reversing edg

exchanging inputs

computes the tran

1956 Bordewijk;

independently 195

for Boolean matri

1973 Fiduccia ana

preserves number

preserves number

number of nontriv

ute

α_n,
α_n^2,

α_n^t.

s

k

but
key.

Compare to multipoint evaluation:

$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t$,
$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t$,
$\vdots$,
$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t$.

Matrix for syndrome computation is transpose of matrix for multipoint evaluation.

Amazing consequence: syndrome computation is as few ops as multipoint evaluation. Eliminate precomputed matrix.

Transposition principle: If a linear algorithm computes a matrix M then reversing edges and exchanging inputs/outputs computes the transpose of

1956 Bordewijk; independently 1957 Lupano for Boolean matrices.

1973 Fiduccia analysis: preserves number of mults; preserves number of adds pl number of nontrivial output

Compare to multipoint evaluation:

$F(\alpha_1) = F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t,$
$F(\alpha_2) = F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t,$
$\vdots,$
$F(\alpha_n) = F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t.$

Matrix for syndrome computation is transpose of matrix for multipoint evaluation.

Amazing consequence: syndrome computation is as few ops as multipoint evaluation. Eliminate precomputed matrix.

Transposition principle: If a linear algorithm computes a matrix M then reversing edges and exchanging inputs/outputs computes the transpose of M.

1956 Bordewijk; independently 1957 Lupanov for Boolean matrices.

1973 Fiduccia analysis: preserves number of mults; preserves number of adds plus number of nontrivial outputs.

e to multipoint evaluation:

$$= F_0 + F_1\alpha_1 + \cdots + F_t\alpha_1^t,$$
$$= F_0 + F_1\alpha_2 + \cdots + F_t\alpha_2^t,$$

$$= F_0 + F_1\alpha_n + \cdots + F_t\alpha_n^t.$$

or syndrome computation
ose of
or multipoint evaluation.

g consequence:
e computation is as few
nultipoint evaluation.
e precomputed matrix.

Transposition principle:
If a linear algorithm
computes a matrix M
then reversing edges and
exchanging inputs/outputs
computes the transpose of M.

1956 Bordewijk;
independently 1957 Lupanov
for Boolean matrices.

1973 Fiduccia analysis:
preserves number of mults;
preserves number of adds plus
number of nontrivial outputs.

We built
producin
Too mar
gcc ran

point evaluation:

$\alpha_1 + \cdots + F_t\alpha_1^t,$

$\alpha_2 + \cdots + F_t\alpha_2^t,$

$\alpha_n + \cdots + F_t\alpha_n^t.$

me computation

oint evaluation.

ence:

ation is as few

evaluation.

puted matrix.

Transposition principle:
If a linear algorithm
computes a matrix M
then reversing edges and
exchanging inputs/outputs
computes the transpose of M.

1956 Bordewijk;
independently 1957 Lupanov
for Boolean matrices.

1973 Fiduccia analysis:
preserves number of mults;
preserves number of adds plus
number of nontrivial outputs.

We built transposi
producing C code.
Too many variable
gcc ran out of me

uation:

$F_t\alpha_1^t$,

$F_t\alpha_2^t$,

$F_t\alpha_n^t$.

tation

tion.

few

.

rix.

Transposition principle:
If a linear algorithm
computes a matrix M
then reversing edges and
exchanging inputs/outputs
computes the transpose of M.

1956 Bordewijk;
independently 1957 Lupanov
for Boolean matrices.

1973 Fiduccia analysis:
preserves number of mults;
preserves number of adds plus
number of nontrivial outputs.

We built transposing compil
producing C code.
Too many variables for $m =$
gcc ran out of memory.

Transposition principle:
If a linear algorithm computes a matrix M then reversing edges and exchanging inputs/outputs computes the transpose of M.

1956 Bordewijk; independently 1957 Lupanov for Boolean matrices.

1973 Fiduccia analysis: preserves number of mults; preserves number of adds plus number of nontrivial outputs.

We built transposing compiler producing C code.
Too many variables for $m = 13$; gcc ran out of memory.

Transposition principle:
If a linear algorithm computes a matrix M then reversing edges and exchanging inputs/outputs computes the transpose of M.

1956 Bordewijk; independently 1957 Lupanov for Boolean matrices.

1973 Fiduccia analysis: preserves number of mults; preserves number of adds plus number of nontrivial outputs.

We built transposing compiler producing C code.
Too many variables for $m = 13$; `gcc` ran out of memory.

Used `qhasm` register allocator to optimize the variables.
Worked, but not very quickly.

Transposition principle:
If a linear algorithm computes a matrix M then reversing edges and exchanging inputs/outputs computes the transpose of M.

1956 Bordewijk; independently 1957 Lupanov for Boolean matrices.

1973 Fiduccia analysis: preserves number of mults; preserves number of adds plus number of nontrivial outputs.

We built transposing compiler producing C code.
Too many variables for $m = 13$; `gcc` ran out of memory.

Used `qhasm` register allocator to optimize the variables.
Worked, but not very quickly.

Wrote faster register allocator.
Still excessive code size.

Transposition principle:
If a linear algorithm
computes a matrix M
then reversing edges and
exchanging inputs/outputs
computes the transpose of M.

1956 Bordewijk;
independently 1957 Lupanov
for Boolean matrices.

1973 Fiduccia analysis:
preserves number of mults;
preserves number of adds plus
number of nontrivial outputs.

We built transposing compiler
producing C code.
Too many variables for $m = 13$;
`gcc` ran out of memory.

Used `qhasm` register allocator
to optimize the variables.
Worked, but not very quickly.

Wrote faster register allocator.
Still excessive code size.

Built new interpreter,
allowing some code compression.
Still big; still some overhead.

sition principle:
ar algorithm
es a matrix M
ersing edges and
ing inputs/outputs
es the transpose of M.

rdewijk;
dently 1957 Lupanov
ean matrices.

duccia analysis:
s number of mults;
s number of adds plus
of nontrivial outputs.

We built transposing compiler
producing C code.
Too many variables for $m = 13$;
gcc ran out of memory.

Used qhasm register allocator
to optimize the variables.
Worked, but not very quickly.

Wrote faster register allocator.
Still excessive code size.

Built new interpreter,
allowing some code compression.
Still big; still some overhead.

Better s
stared a
wrote d
with san

Small c

Speedup
translate
to transp

Further
merged
scaling b

ciple:
m
k M
ges and
/outputs
spose of M.

7 Lupanov
ces.

lysis:
of mults;
of adds plus
ial outputs.

We built transposing compiler producing C code.
Too many variables for $m = 13$; gcc ran out of memory.

Used qhasm register allocator to optimize the variables.
Worked, but not very quickly.

Wrote faster register allocator.
Still excessive code size.

Built new interpreter, allowing some code compression.
Still big; still some overhead.

Better solution:
stared at additive
wrote down transp
with same loops e

Small code, no ov

Speedups of addit
translate easily
to transposed algo

Further savings:
merged first stage
scaling by Goppa

M.

/

us

s.

We built transposing compiler
producing C code.
Too many variables for $m = 13$;
gcc ran out of memory.

Used `qhasm` register allocator
to optimize the variables.
Worked, but not very quickly.

Wrote faster register allocator.
Still excessive code size.

Built new interpreter,
allowing some code compression.
Still big; still some overhead.

Better solution:
stared at additive FFT,
wrote down transposition
with same loops etc.

Small code, no overhead.

Speedups of additive FFT
translate easily
to transposed algorithm.

Further savings:
merged first stage with
scaling by Goppa constants.

We built transposing compiler producing C code.
Too many variables for $m = 13$; `gcc` ran out of memory.

Used `qhasm` register allocator to optimize the variables.
Worked, but not very quickly.

Wrote faster register allocator.
Still excessive code size.

Built new interpreter, allowing some code compression.
Still big; still some overhead.

Better solution:
stared at additive FFT, wrote down transposition with same loops etc.

Small code, no overhead.

Speedups of additive FFT translate easily to transposed algorithm.

Further savings:
merged first stage with scaling by Goppa constants.

t transposing compiler
ng C code.
ny variables for $m = 13$;
out of memory.

asm register allocator
nize the variables.
but not very quickly.

aster register allocator.
essive code size.

w interpreter,
some code compression.
still some overhead.

Better solution:
stared at additive FFT,
wrote down transposition
with same loops etc.

Small code, no overhead.

Speedups of additive FFT
translate easily
to transposed algorithm.

Further savings:
merged first stage with
scaling by Goppa constants.

Results

60493 Iv

8622 fo
20846 fo
7714 fo
14794 fo
8520 fo

Code wi
We're st

More inf
cr.yp.t

ng compiler

es for $m = 13$;
emory.

er allocator
riables.
very quickly.

ter allocator.
e size.

ter,
e compression.
e overhead.

Better solution:
stared at additive FFT,
wrote down transposition
with same loops etc.

Small code, no overhead.

Speedups of additive FFT
translate easily
to transposed algorithm.

Further savings:
merged first stage with
scaling by Goppa constants.

Results

60493 Ivy Bridge c

8622 for permuta
20846 for syndrom
7714 for BM.
14794 for roots.
8520 for permuta

Code will be publi
We're still speedin

More information:
cr.yp.to/papers

er

13;

or

y.

or.

ssion.

.

Better solution:
stared at additive FFT,
wrote down transposition
with same loops etc.

Small code, no overhead.

Speedups of additive FFT
translate easily
to transposed algorithm.

Further savings:
merged first stage with
scaling by Goppa constants.

Results

60493 Ivy Bridge cycles:

8622 for permutation.
20846 for syndrome.
7714 for BM.
14794 for roots.
8520 for permutation.

Code will be public domain.
We're still speeding it up.

More information:
cr.yp.to/papers.html#m

Better solution:
stared at additive FFT,
wrote down transposition
with same loops etc.

Small code, no overhead.

Speedups of additive FFT
translate easily
to transposed algorithm.

Further savings:
merged first stage with
scaling by Goppa constants.

Results

60493 Ivy Bridge cycles:

8622 for permutation.
20846 for syndrome.
7714 for BM.
14794 for roots.
8520 for permutation.

Code will be public domain.
We're still speeding it up.

More information:
cr.yp.to/papers.html#mcbits