Faster exponential time algorithms for the shortest vector problem

Panagiotis Voulgaris Daniele Micciancio

University of California, San Diego

January 19, 2010,
SODA

Applications of lattice algorithms

Useful in a number of fields:

- Combinatorial Problems:
  - Knapsack problems, Integer Programming, ...
- Algebraic Number Theory:
  - Factoring polynomials with rational coefficients, ...
- Cryptanalysis applications:
  - Ntru, Special cases of RSA, ...
- Cryptography based directly on Lattices:
  - LWE variants, Fully Homomorphic crypto, ...

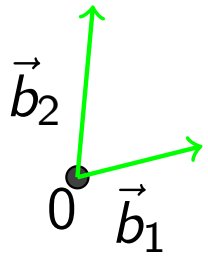
Shortest Vector Problem (SVP)

SVP is a foundational lattice problem:

- Exact SVP is known to be NP-complete
- In most applications approximations are enough
- However approx. algorithms utilize exact SVP for lower dimensions

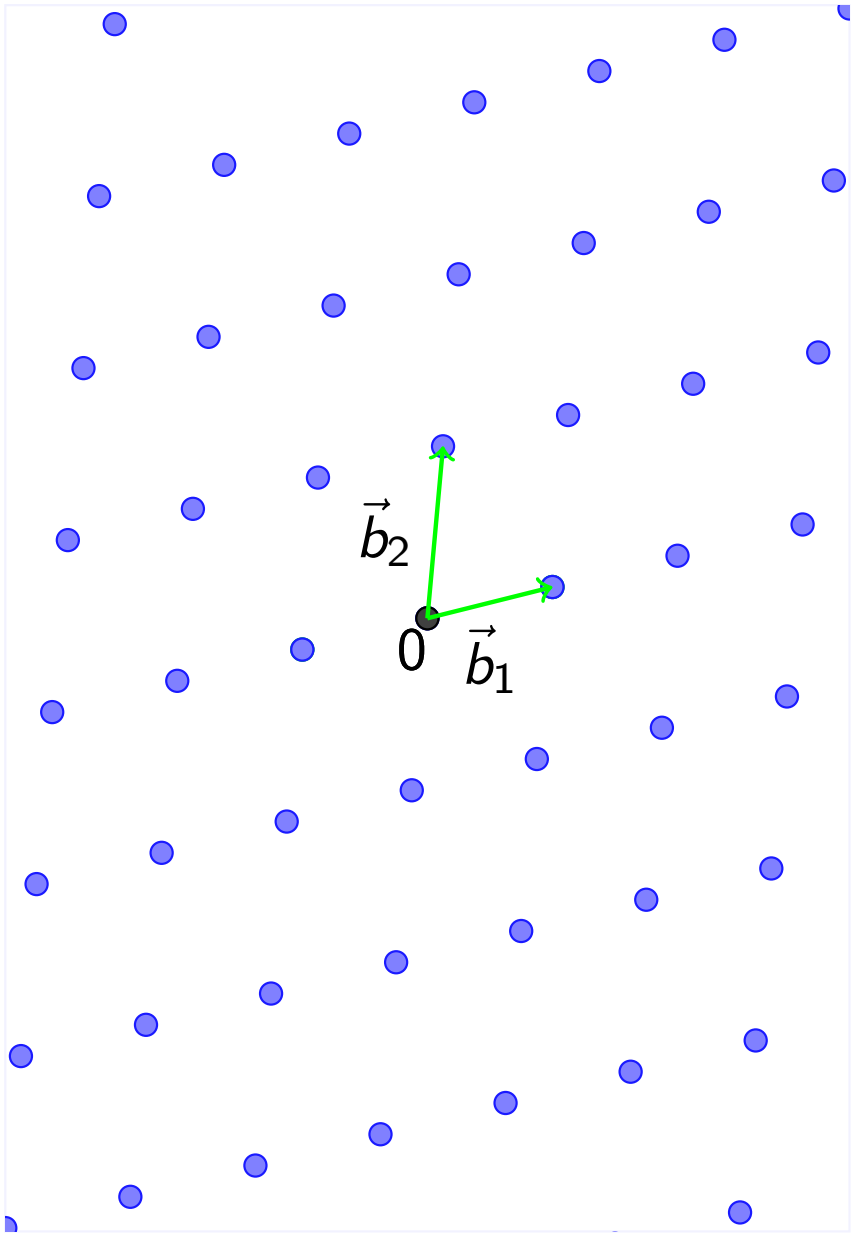
1. Background
   - Definitions
   - Existing Algorithms
2. Contribution
   - List Sieve
   - Theoretical Analysis
   - Implementation
3. Final Remarks
   - Summary

Shortest Vector Problem (SVP)



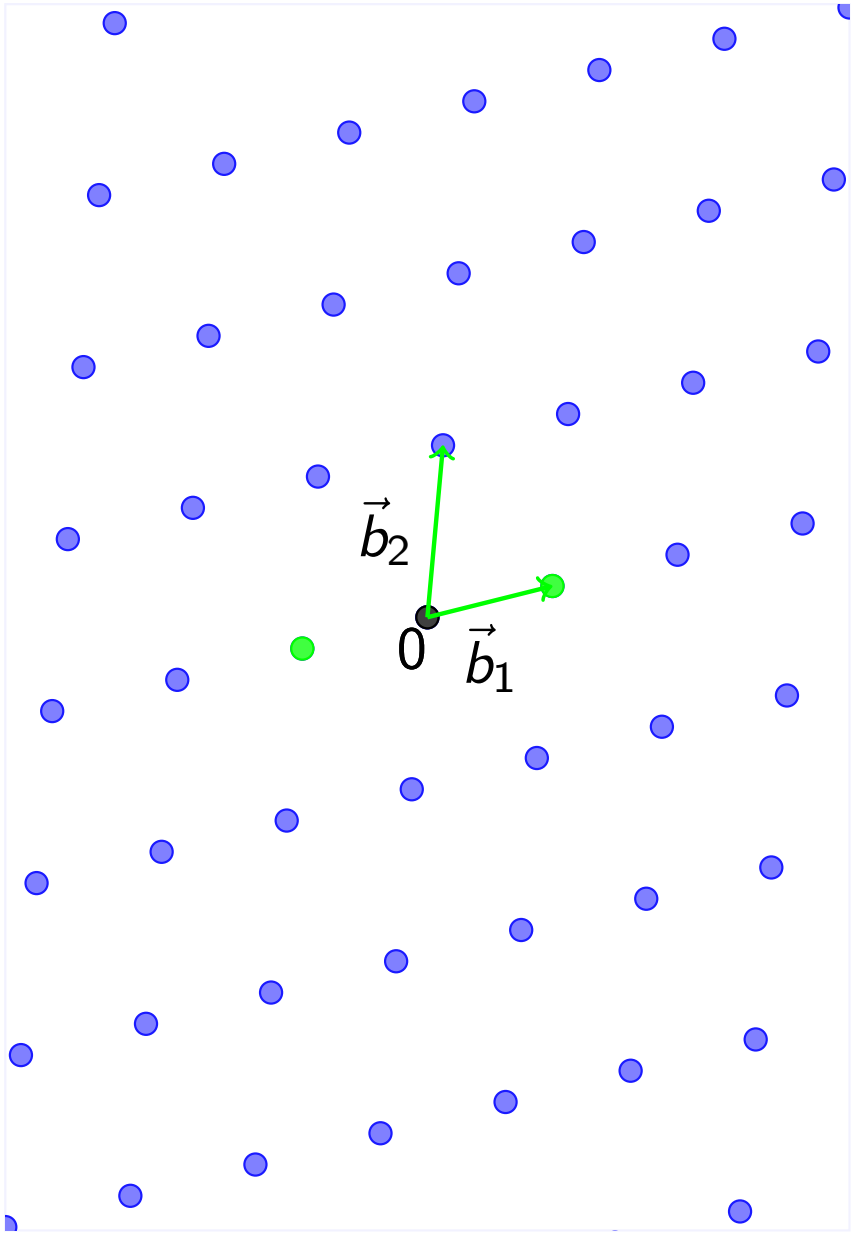
- Given a linearly indep. *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$

Shortest Vector Problem (SVP)



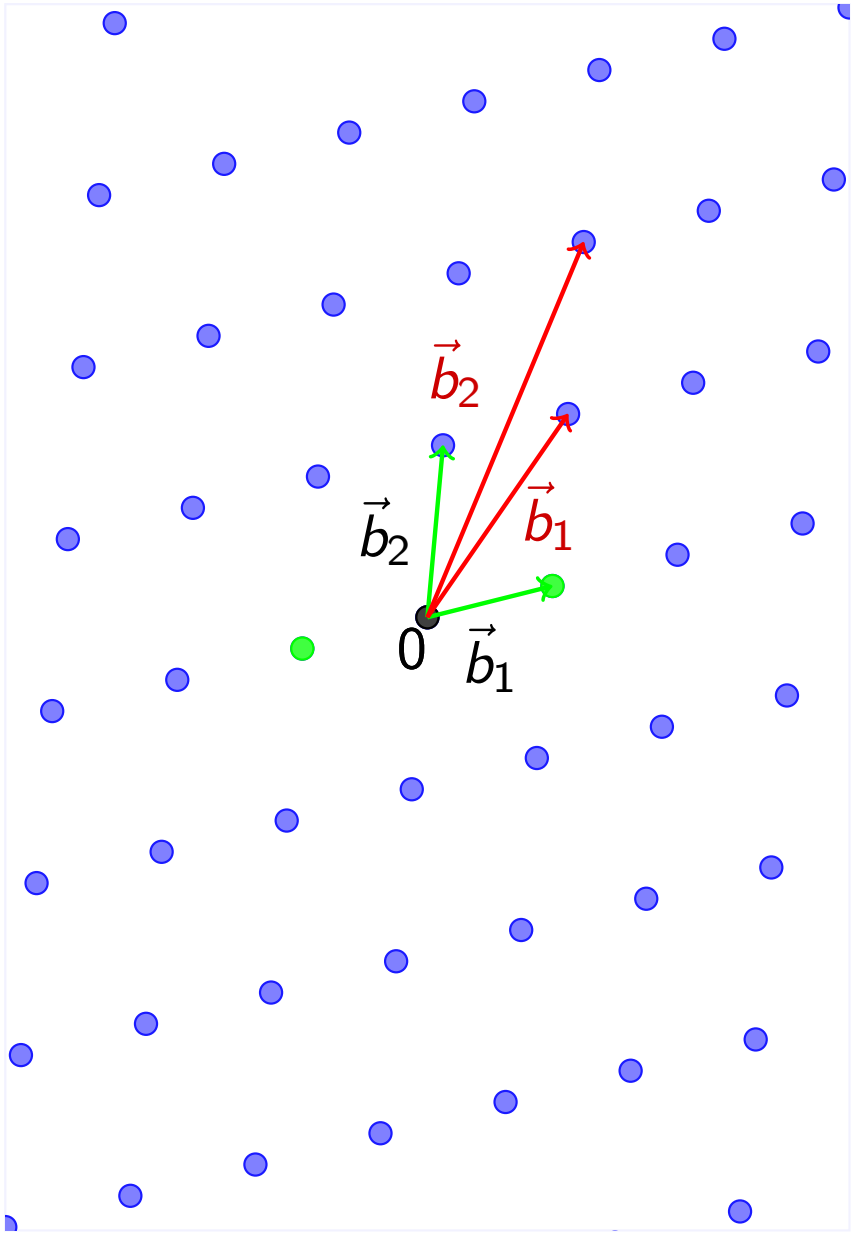
- Given a linearly indep. *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$
- *Lattice* is the closure of $\mathbf{B}$ under $(+, -)$: $\mathcal{L}(\mathbf{B}) = \{\sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$

Shortest Vector Problem (SVP)



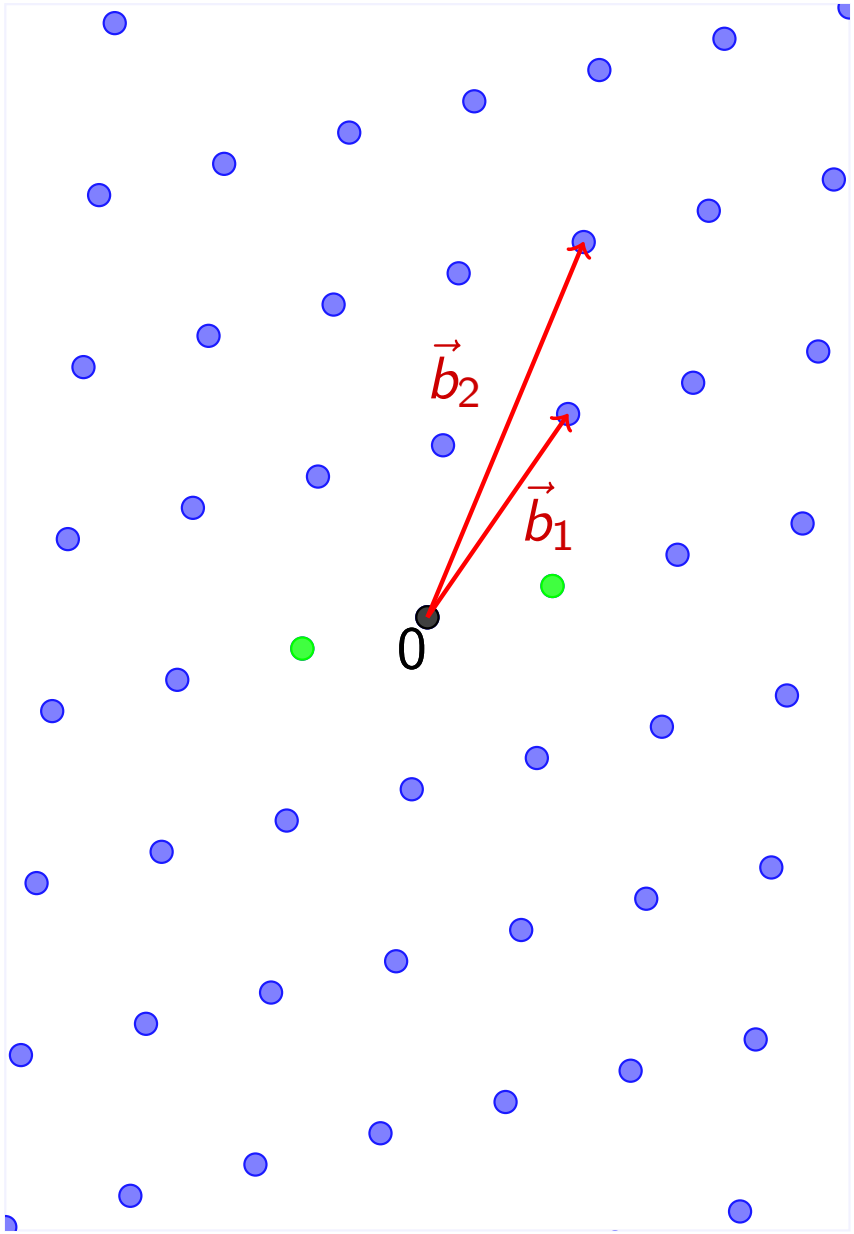
- Given a linearly indep. *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$
- *Lattice* is the closure of $\mathbf{B}$ under $(+, -)$: $\mathcal{L}(\mathbf{B}) = \{\sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$
- *Shortest lattice point*: $\vec{s} \in \mathcal{L}(\mathbf{B}) \setminus \vec{0}$ such that: $\forall \vec{p} \in \mathcal{L}(\mathbf{B}) \setminus \vec{0},\ \|\vec{s}\| \leq \|\vec{p}\|$

Shortest Vector Problem (SVP)



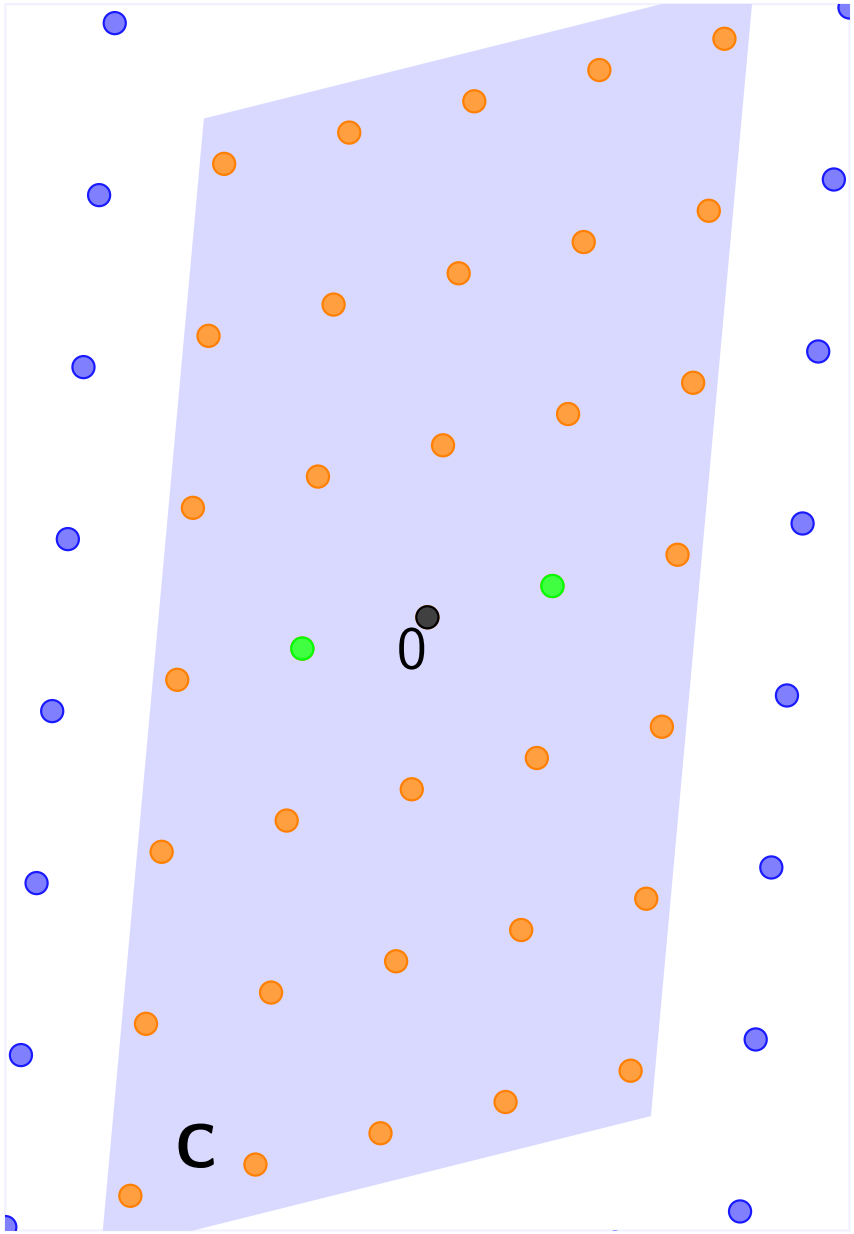
- Given a linearly indep. *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$
- *Lattice* is the closure of $\mathbf{B}$ under $(+, -)$: $\mathcal{L}(\mathbf{B}) = \{\sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$
- *Shortest lattice point*: $\vec{s} \in \mathcal{L}(\mathbf{B}) \setminus \vec{0}$ such that: $\forall \vec{p} \in \mathcal{L}(\mathbf{B}) \setminus \vec{0},\ \|\vec{s}\| \leq \|\vec{p}\|$
- Notice that the basis is not unique

Shortest Vector Problem (SVP)



- Given a linearly indep. *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$
- *Lattice* is the closure of $\mathbf{B}$ under $(+, -)$: $\mathcal{L}(\mathbf{B}) = \{\sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$
- *Shortest lattice point*: $\vec{s} \in \mathcal{L}(\mathbf{B}) \setminus \vec{0}$ such that: $\forall \vec{p} \in \mathcal{L}(\mathbf{B}) \setminus \vec{0},\ \|\vec{s}\| \leq \|\vec{p}\|$
- Notice that the basis is not unique
- *Shortest Vector Problem:* Given a basis $\mathbf{B}$, find a shortest lattice point $\vec{s}$

1st Approach: Enumeration

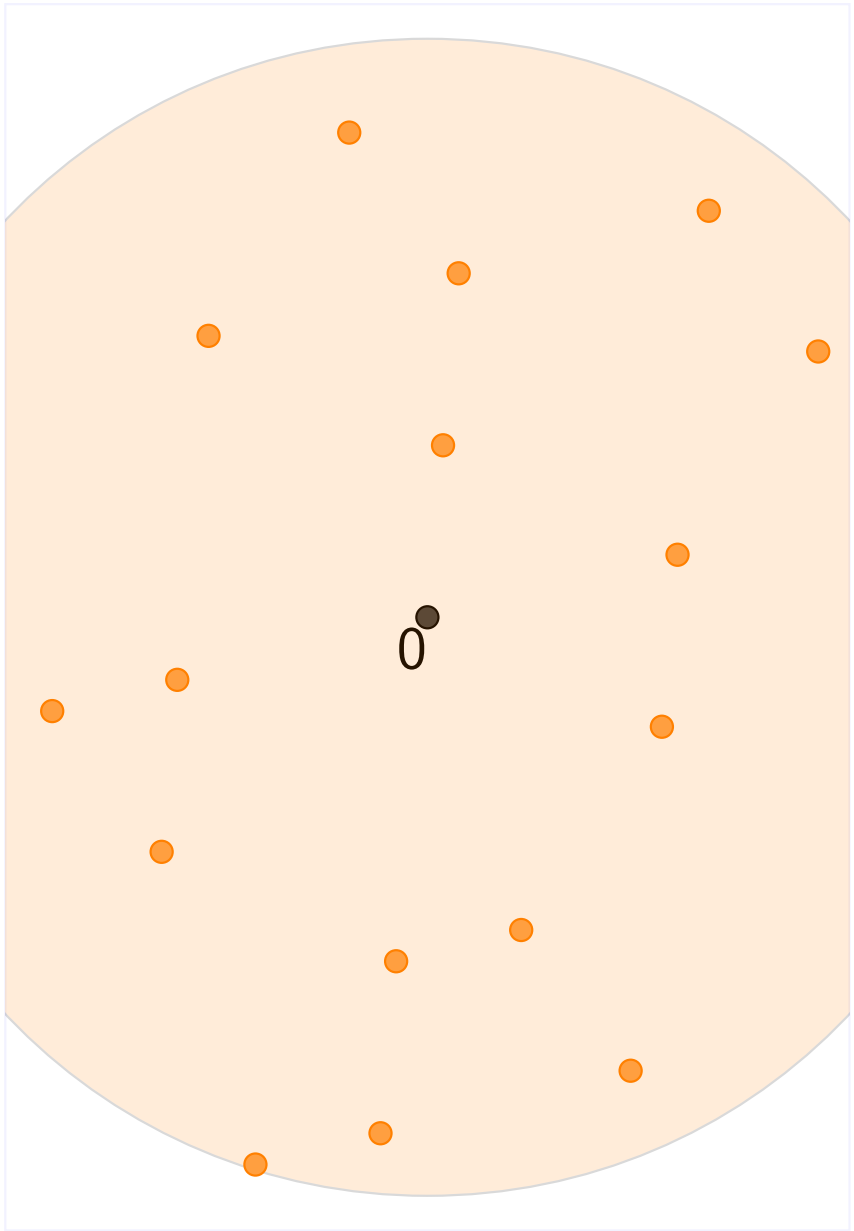


Main idea

Given a basis **B**,
determine a region **C**,
such that $\vec{s} \in \mathbf{C}$.
Enumerate all the points in **C**

- Advantages:
  - Minimal space
- Disadvantages:
  - #Points can be $2^{O(n \log n)}$

2nd Approach: Sieving



Main idea

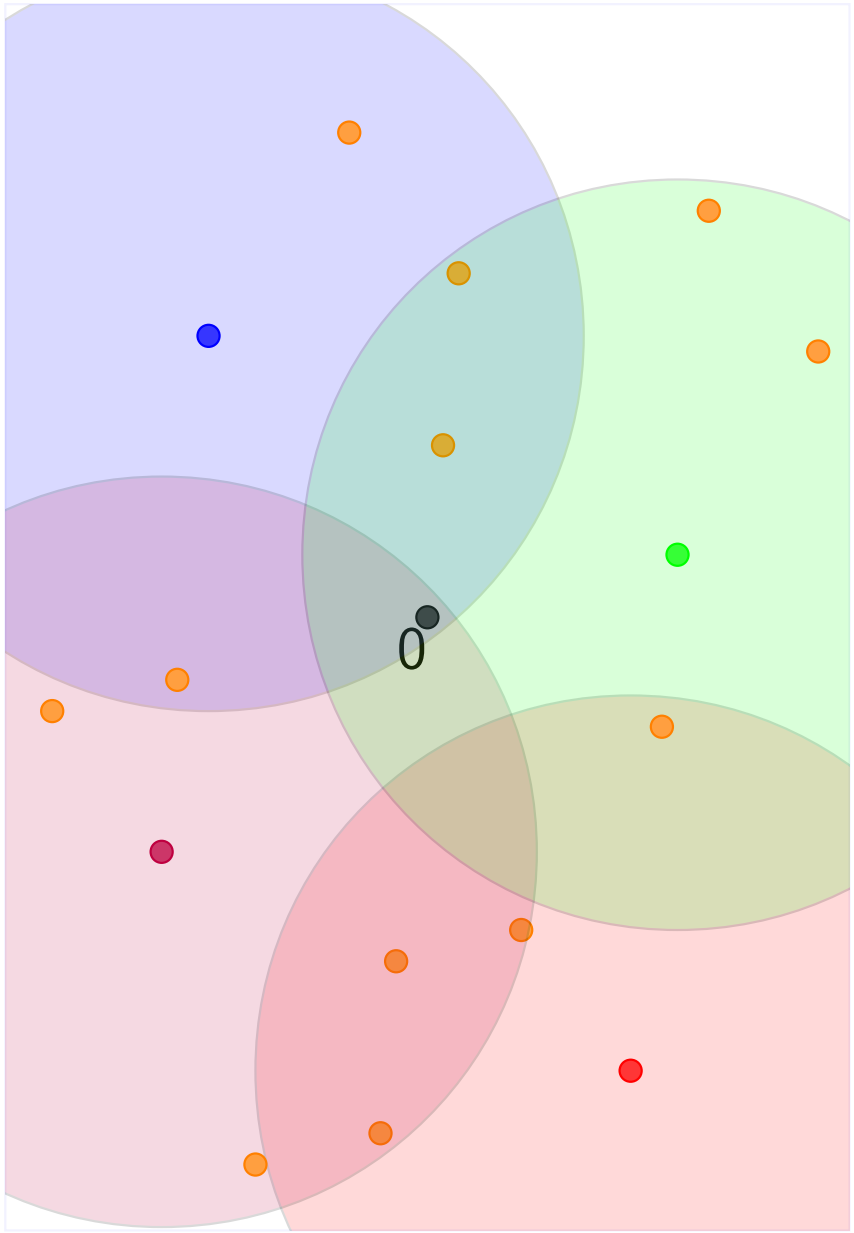
Sample 2^{cn} points, $\|\vec{p}\| \leq R_0$

Cover the samples with spheres of radius $R_1 < R_0$ centered at samples

Obtain shorter vectors by subtracting the centers

- Advantages:
  - #Points bounded by $2^{O(n)}$
- Disadvantages:
  - Space complexity of $2^{O(n)}$
  - Impractical?

2nd Approach: Sieving



Main idea

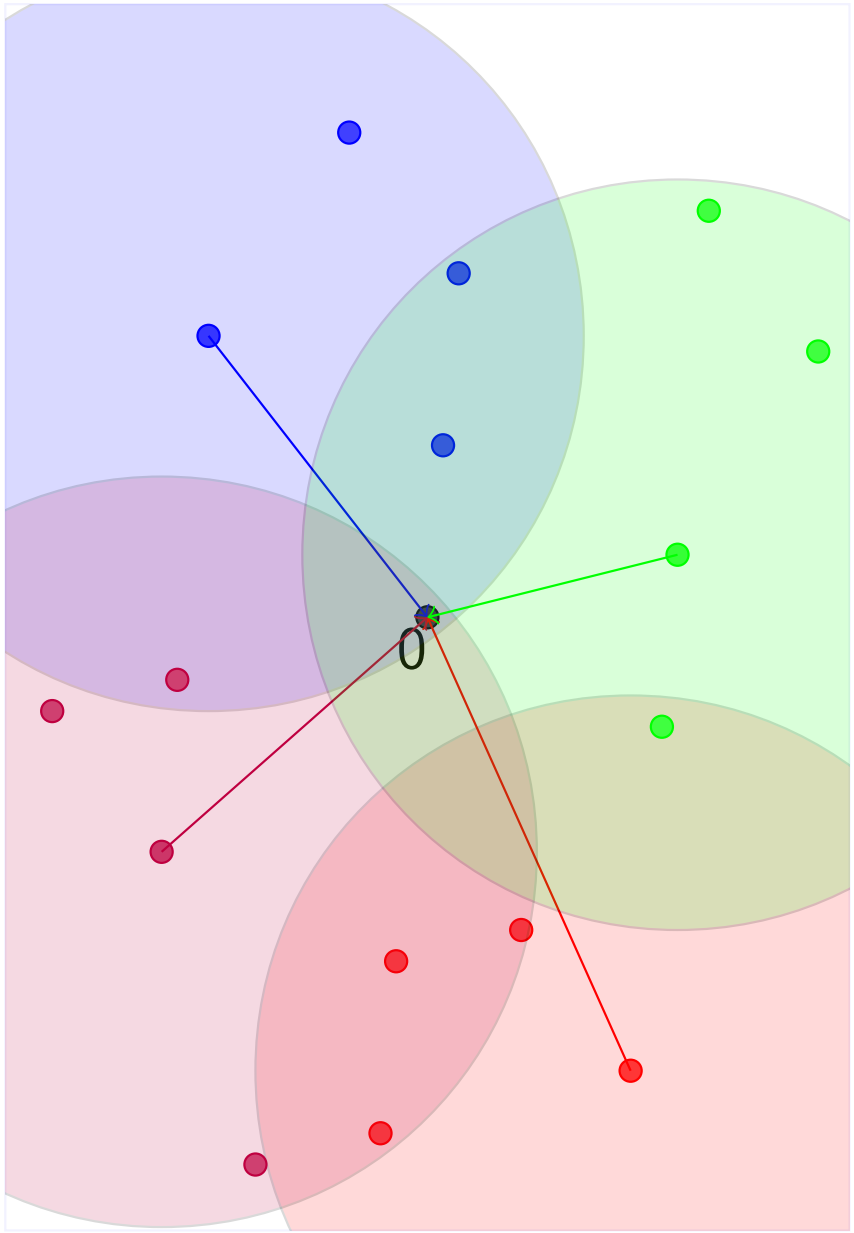
Sample 2^{cn} points, $\|\vec{p}\| \leq R_0$

Cover the samples with spheres of radius $R_1 < R_0$ centered at samples

Obtain shorter vectors by subtracting the centers

- Advantages:
  - #Points bounded by $2^{O(n)}$
- Disadvantages:
  - Space complexity of $2^{O(n)}$
  - Impractical?

2nd Approach: Sieving



Main idea

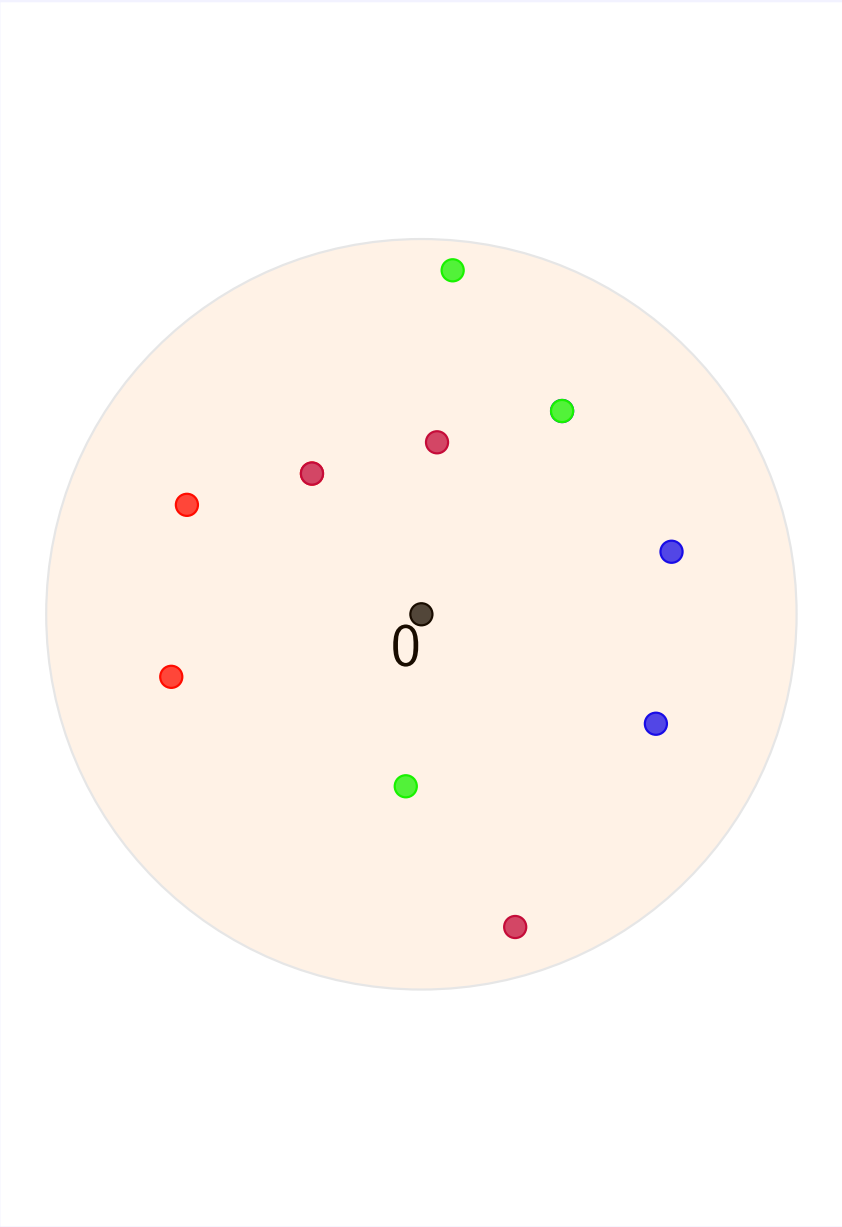
Sample 2^{cn} points, $\|\vec{p}\| \leq R_0$

Cover the samples with spheres of radius $R_1 < R_0$ centered at samples

Obtain shorter vectors by subtracting the centers

- Advantages:
  - #Points bounded by $2^{O(n)}$
- Disadvantages:
  - Space complexity of $2^{O(n)}$
  - Impractical?

2nd Approach: Sieving



Main idea

Sample 2^{cn} points, $\|\vec{p}\| \leq R_0$
Cover the samples with spheres of radius $R_1 < R_0$ centered at samples
Obtain shorter vectors by subtracting the centers

- Advantages:
  - #Points bounded by $2^{O(n)}$
- Disadvantages:
  - Space complexity of $2^{O(n)}$
  - Impractical?

Time-line: Sieving Algorithms

Year, Authors	Time	Space	Practice
2001, Ajtai, Kumar, Sivakumar	$2^{O(n)}$	$2^{O(n)}$	–
2004, Regev	2^{16n}	2^{8n}	–
2008, Nguyen, Vidick	$2^{5.9n}$	$2^{2.95n}$	Practical
2010, This work	$2^{3.2n}$	$2^{1.33n}$	$> 10^2$ speed-up

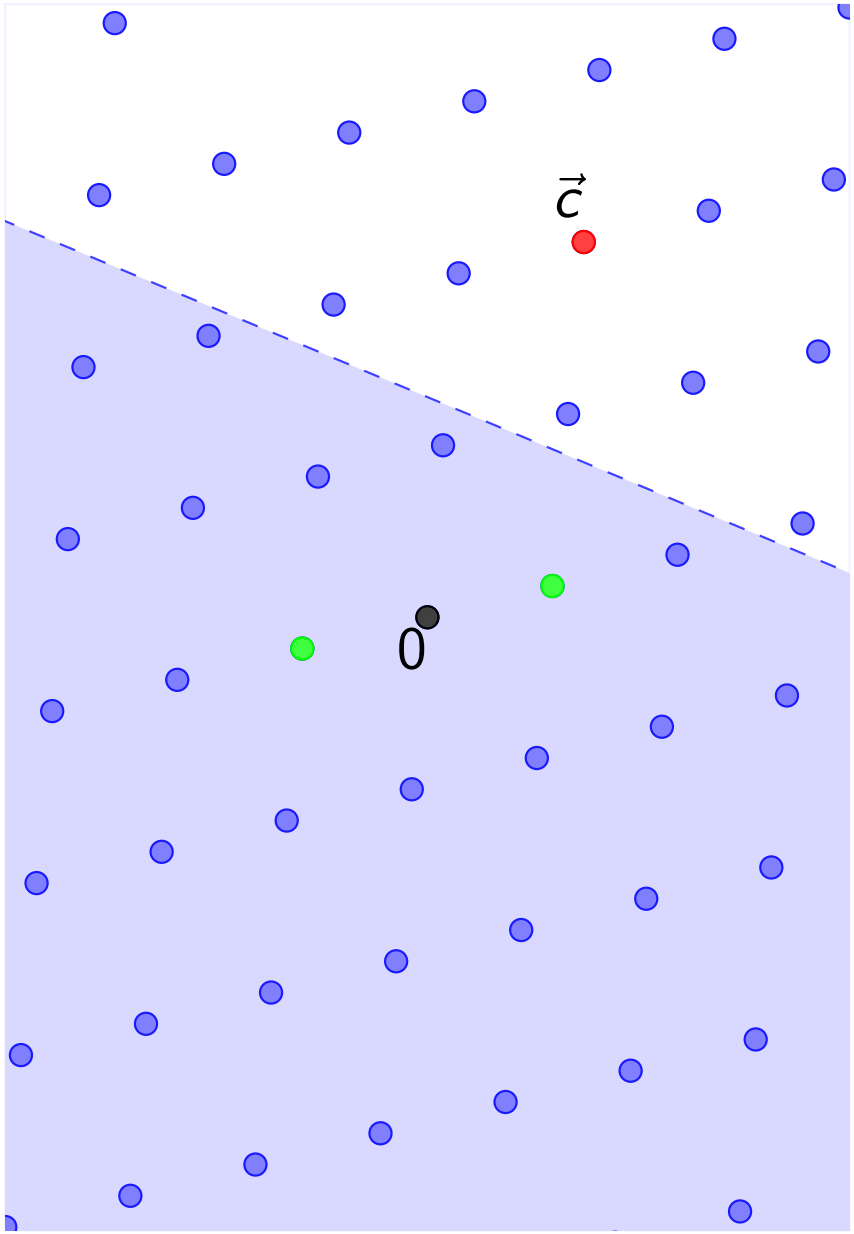
Table: Time-line of Sieving Algorithms

Time-line: Sieving Algorithms

Year, Authors	Time	Space	Practice
2001, Ajtai, Kumar, Sivakumar	$2^{O(n)}$	$2^{O(n)}$	–
2004, Regev	2^{16n}	2^{8n}	–
2008, Nguyen, Vidick	$2^{5.9n}$	$2^{2.95n}$	Practical
2010, This work	$2^{3.2n}$	$2^{1.33n}$	$> 10^2$ speed-up
2010, Pujol, Stelhé	$2^{2.46n}$	$2^{1.233n}$	–

Table: Time-line of Sieving Algorithms

Points and halfspaces



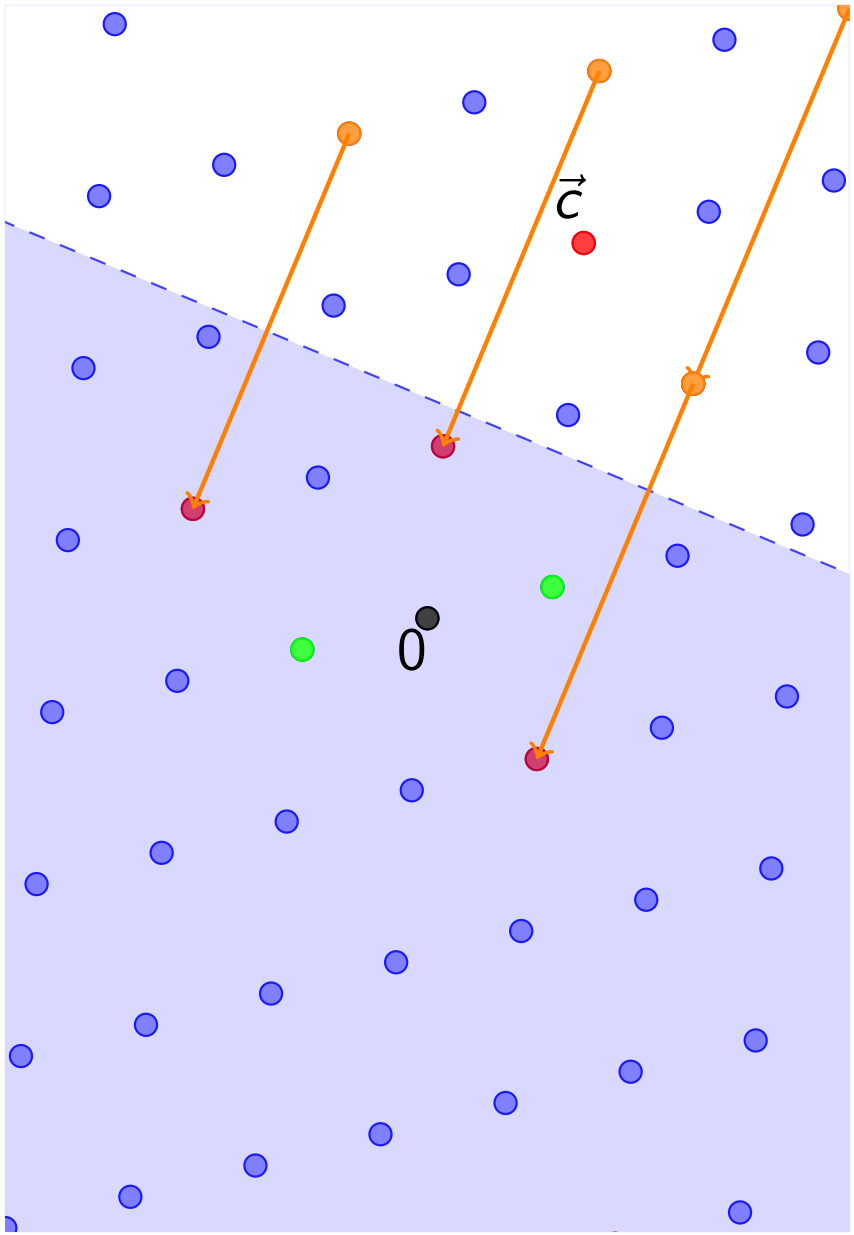
Algorithm: Reduce($\vec{p}, \vec{c}$)

`while` $\|\vec{p} - \vec{c}\| < \|\vec{p}\|$

$\vec{p} \leftarrow \vec{p} - \vec{c}$

- $\vec{c}$ defines two half-spaces:
- $\vec{c}$ halfspace: $\|\vec{p} - \vec{c}\| < \|\vec{p}\|$
- $\vec{0}$ halfspace: $\|\vec{p} - \vec{c}\| \geq \|\vec{p}\|$

Points and halfspaces

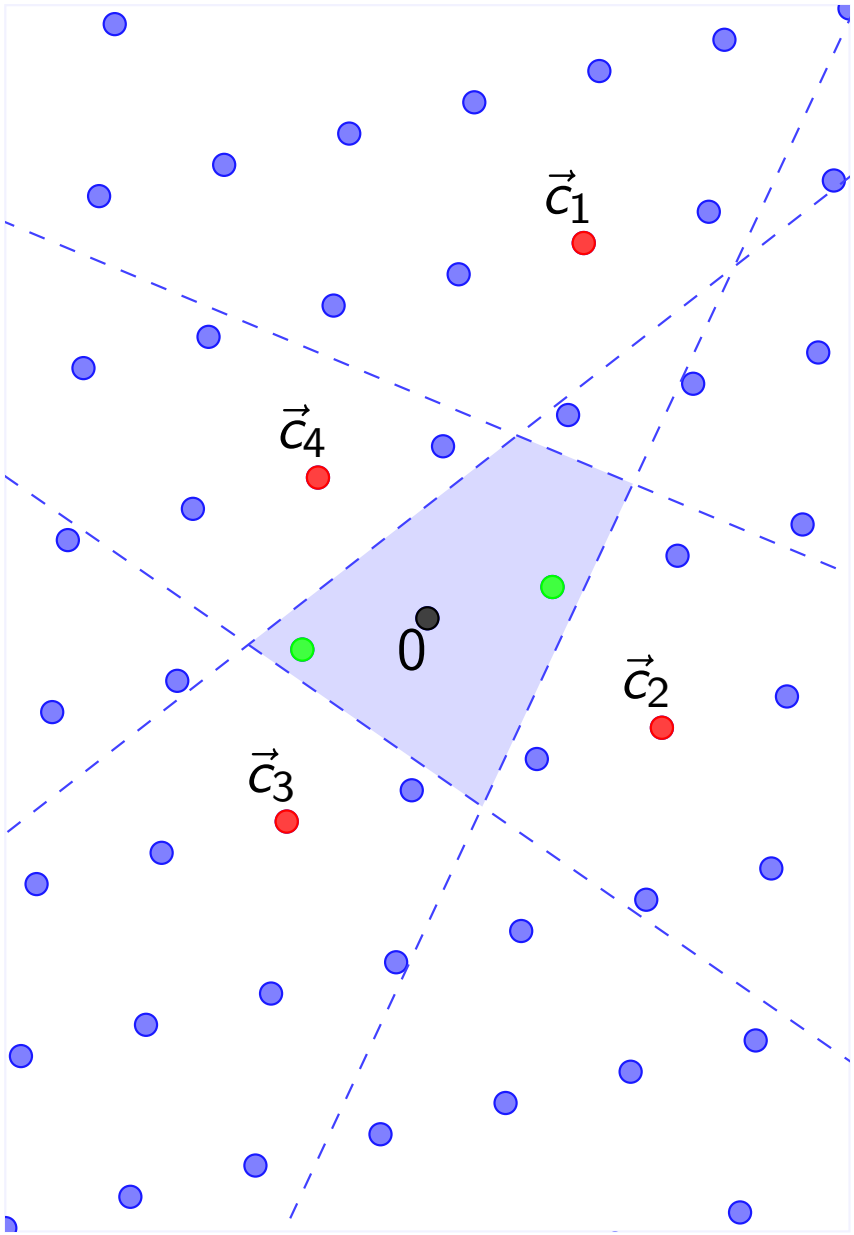


Algorithm: Reduce($\vec{p}, \vec{c}$)

```
while ||p − c|| < ||p||
    p ← p − c
```

- $\vec{c}$ defines two half-spaces:
- $\vec{c}$ halfspace: $\|\vec{p} - \vec{c}\| < \|\vec{p}\|$
- $\vec{0}$ halfspace: $\|\vec{p} - \vec{c}\| \geq \|\vec{p}\|$
- Subtracting $\vec{c}$, brings any point in the $\vec{0}$ halfspace

Reduce with a list of points

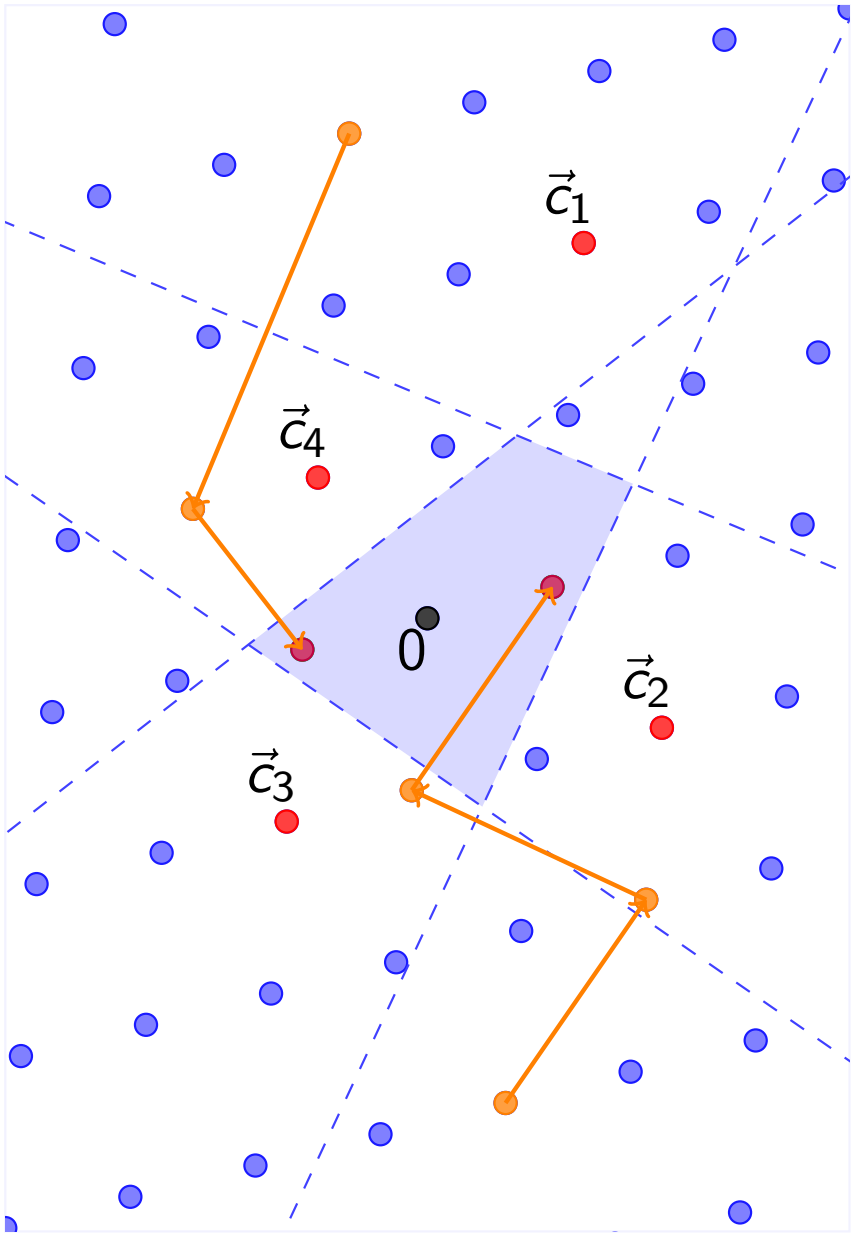


Algorithm: Reduce($\vec{p}, C$)

```
while ∃c_i ∈ C, such that:
        ‖p − c_i‖ < ‖p‖
    p ← p − c_i
```

- Consider a set of points C
- Notice the intersection of the $\vec{0}$ halfspaces

Reduce with a list of points

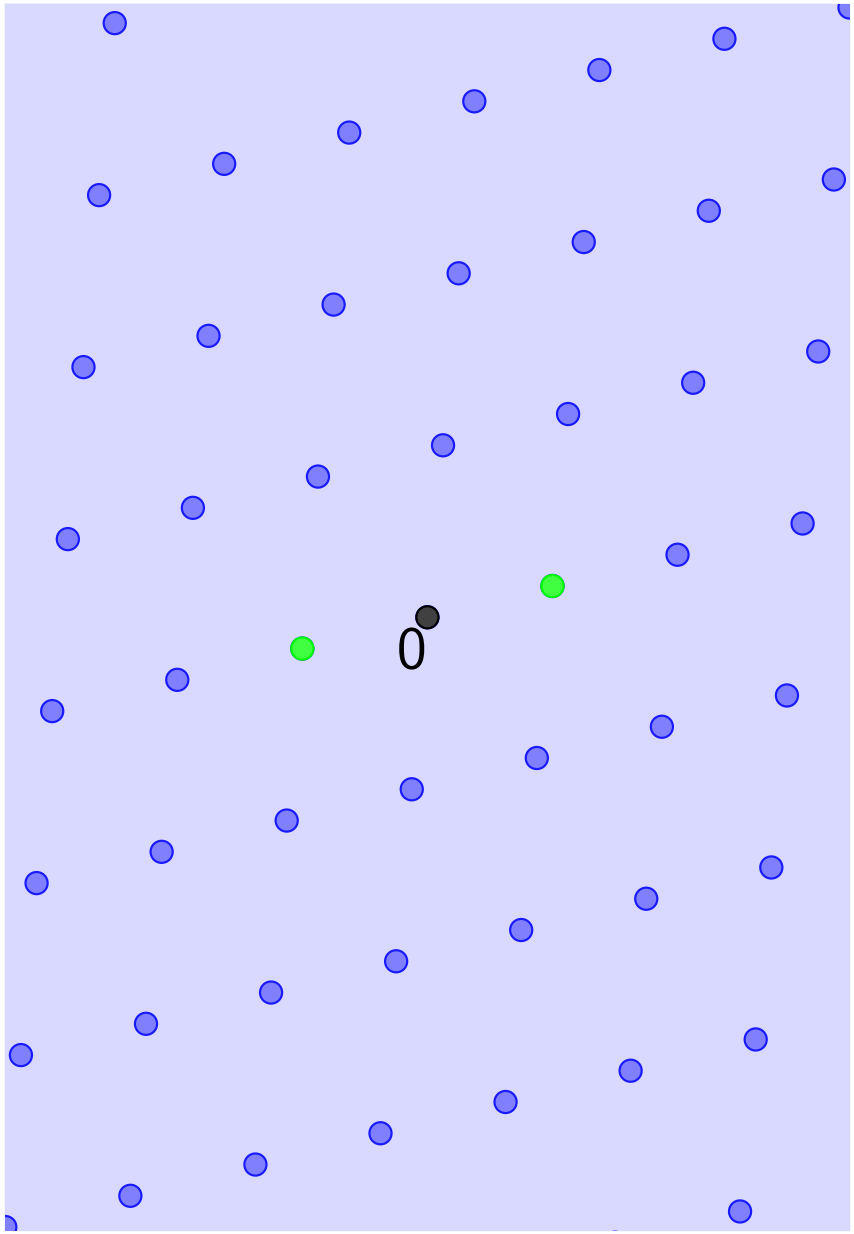


Algorithm: Reduce($\vec{p}, C$)

```
while ∃c_i ∈ C, such that:
        ‖p − c_i‖ < ‖p‖
    p ← p − c_i
```

- Consider a set of points C
- Notice the intersection of the $\vec{0}$ halfspaces
- When `Reduce` terminates, $\vec{p}$ is in the intersection of the $\vec{0}$ halfspaces.

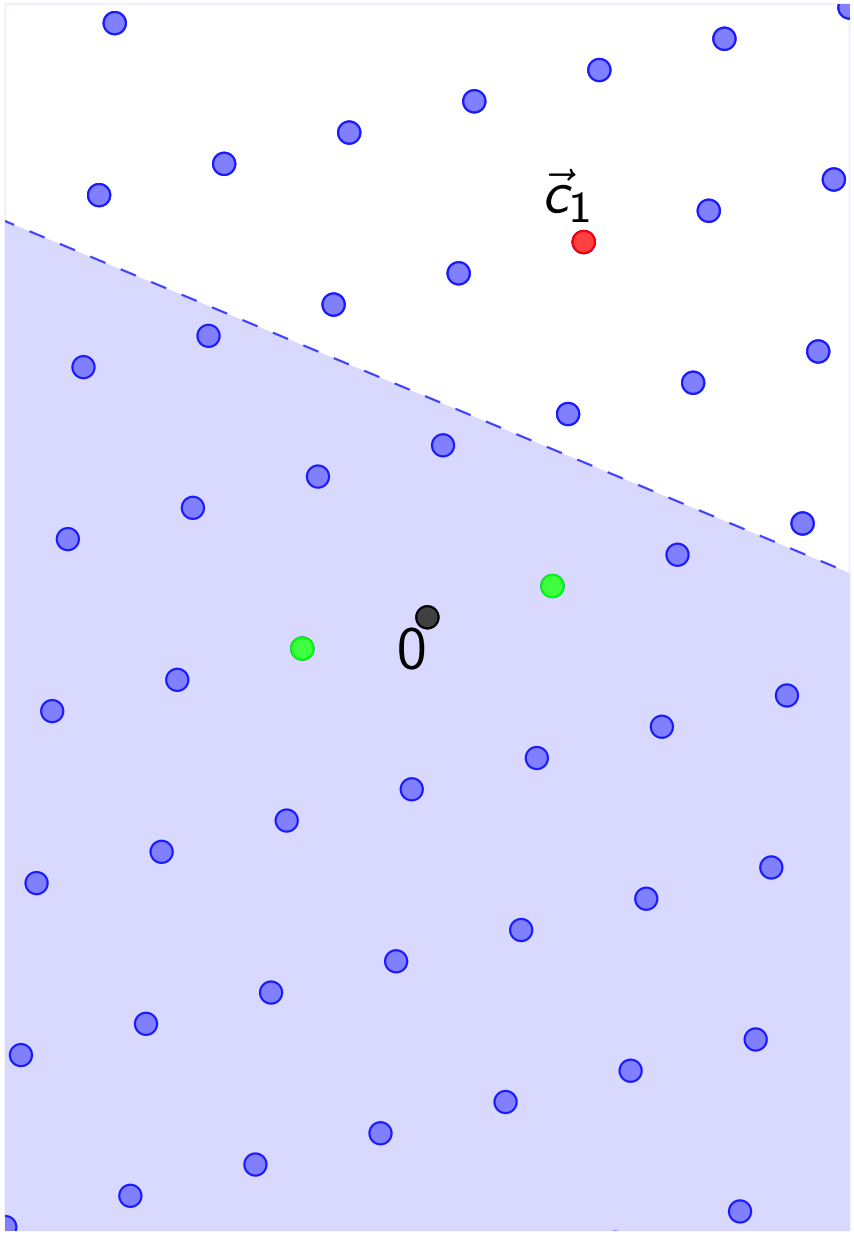
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
}

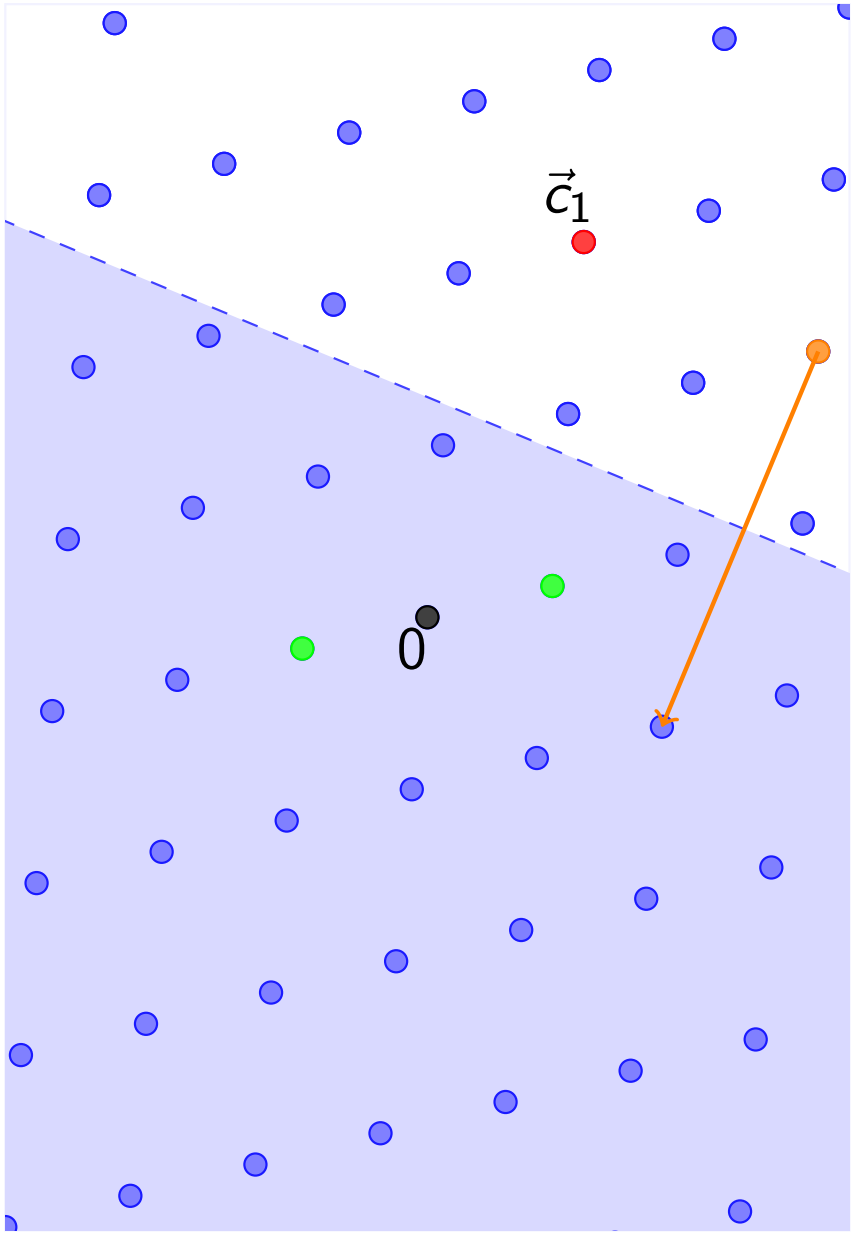
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample(`$\mathbf{B}$`)`
  $\vec{p}' \leftarrow$ `Reduce(`$\vec{p}, C$`)`
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`

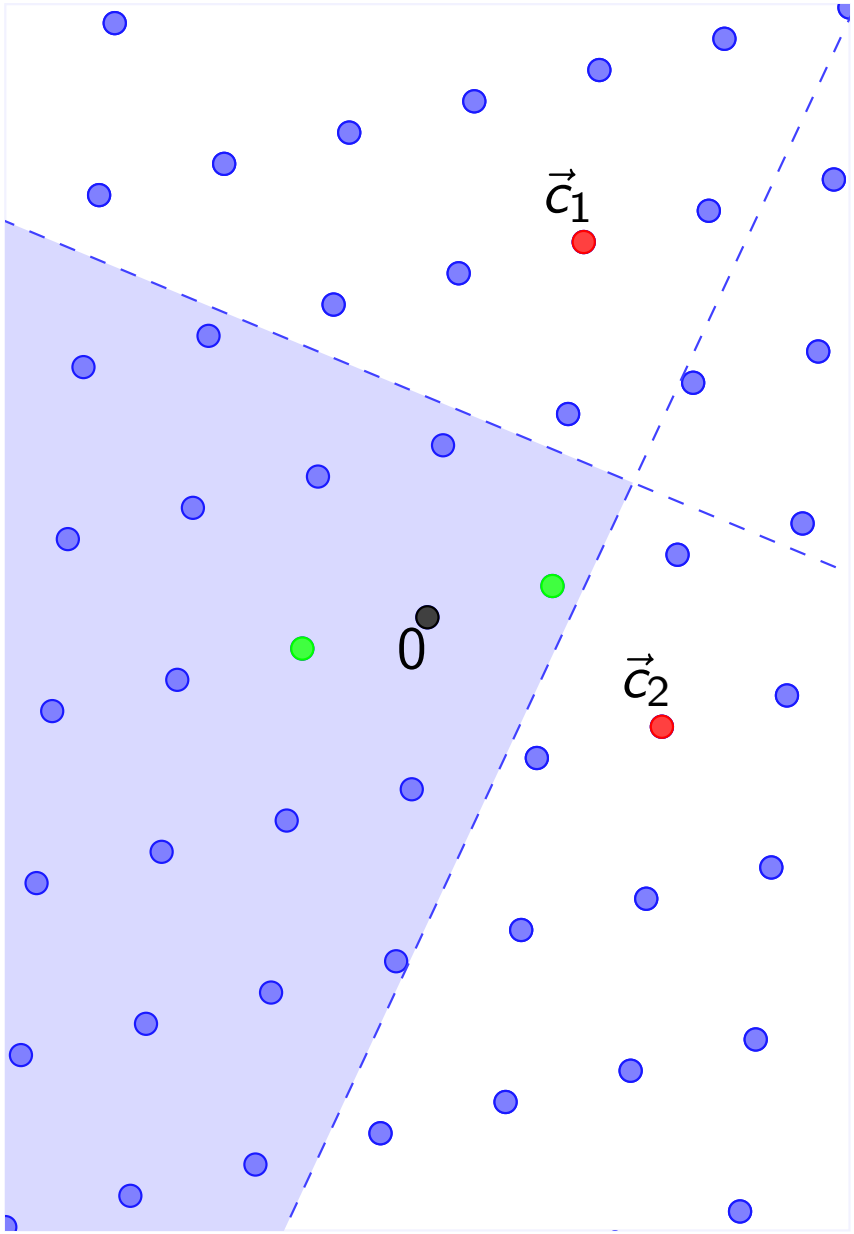
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow \texttt{Sample}(\mathbf{B})$
  $\vec{p}' \leftarrow \texttt{Reduce}(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`

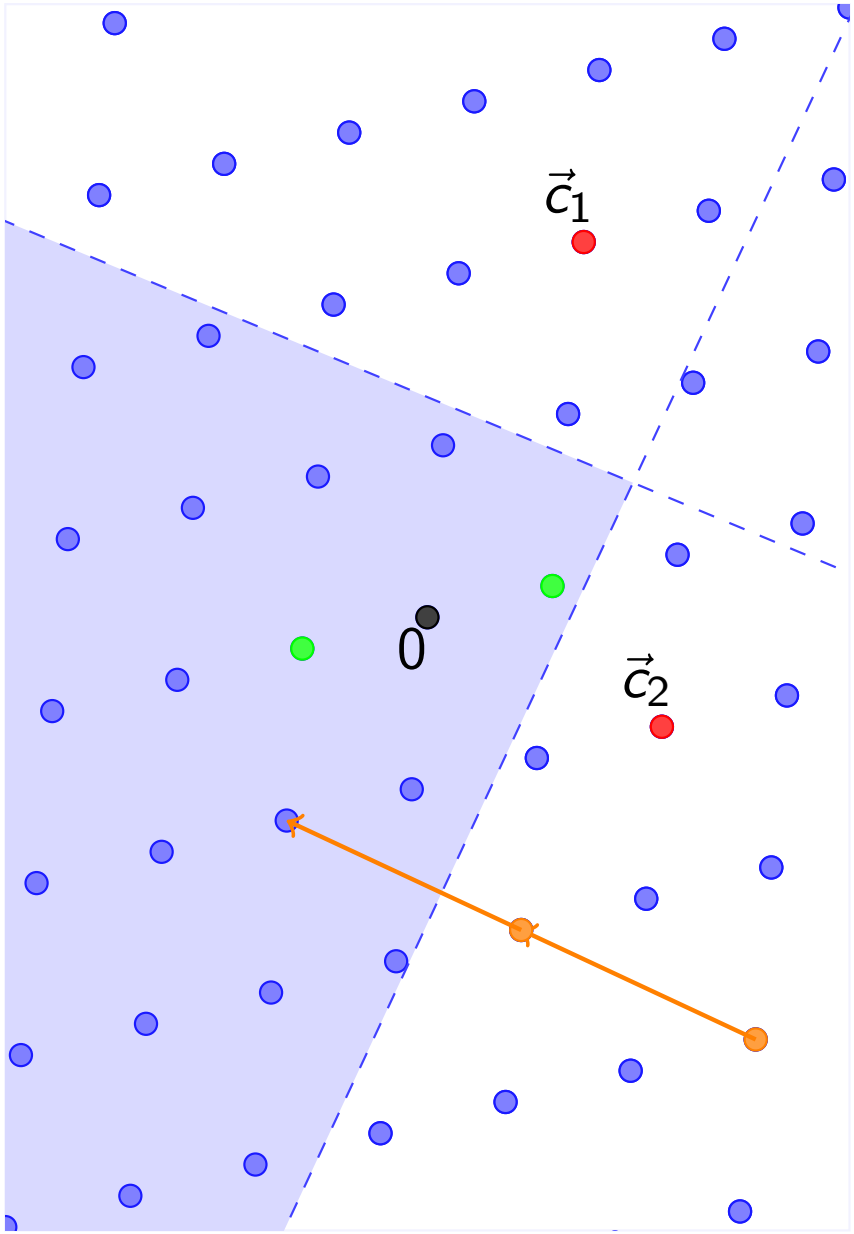
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
}

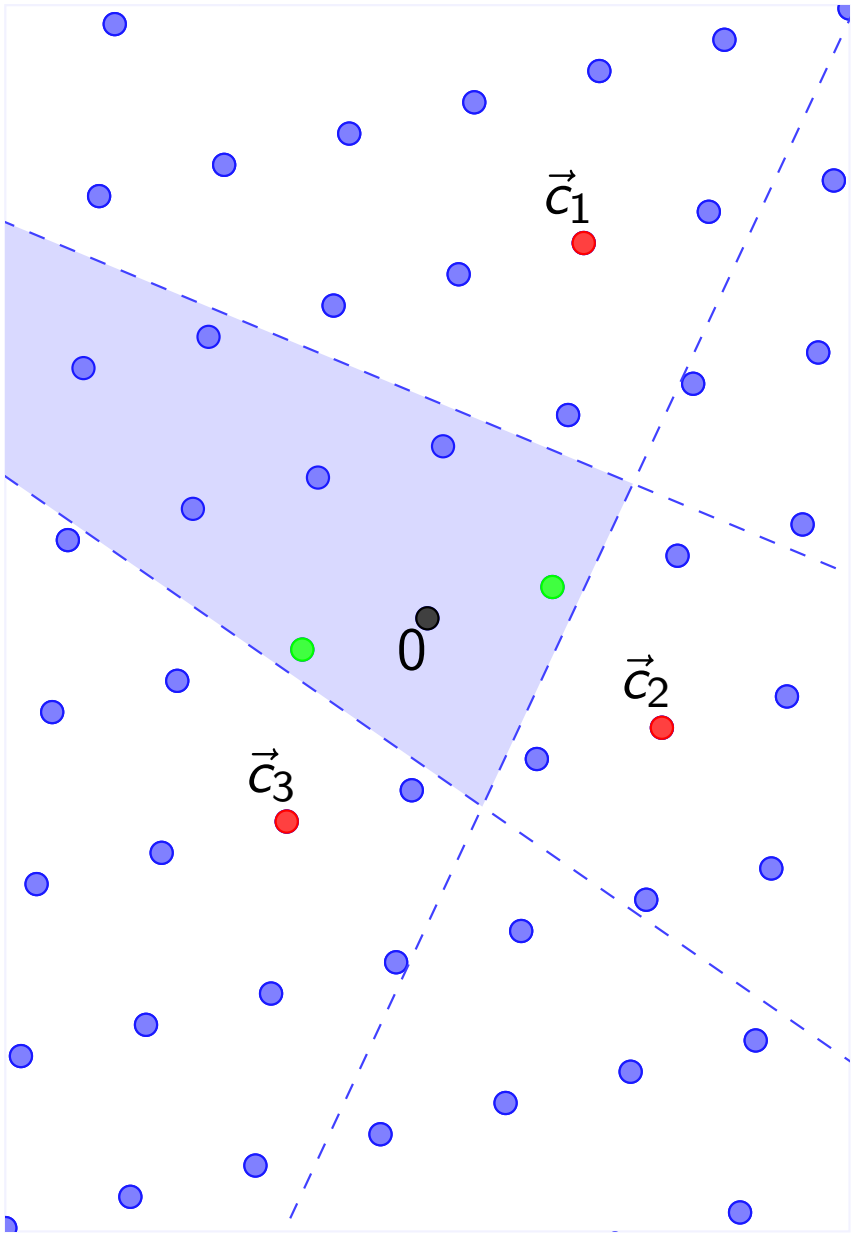
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`

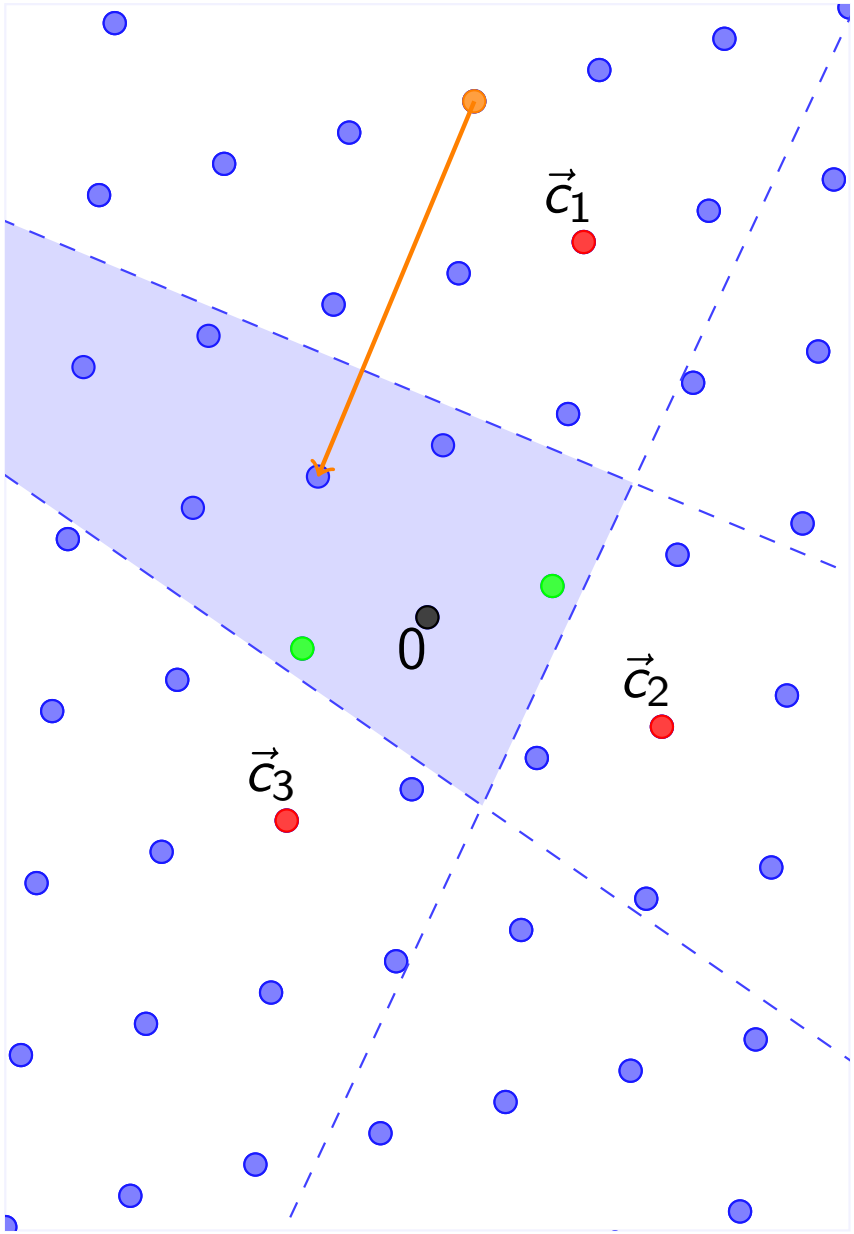
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
}

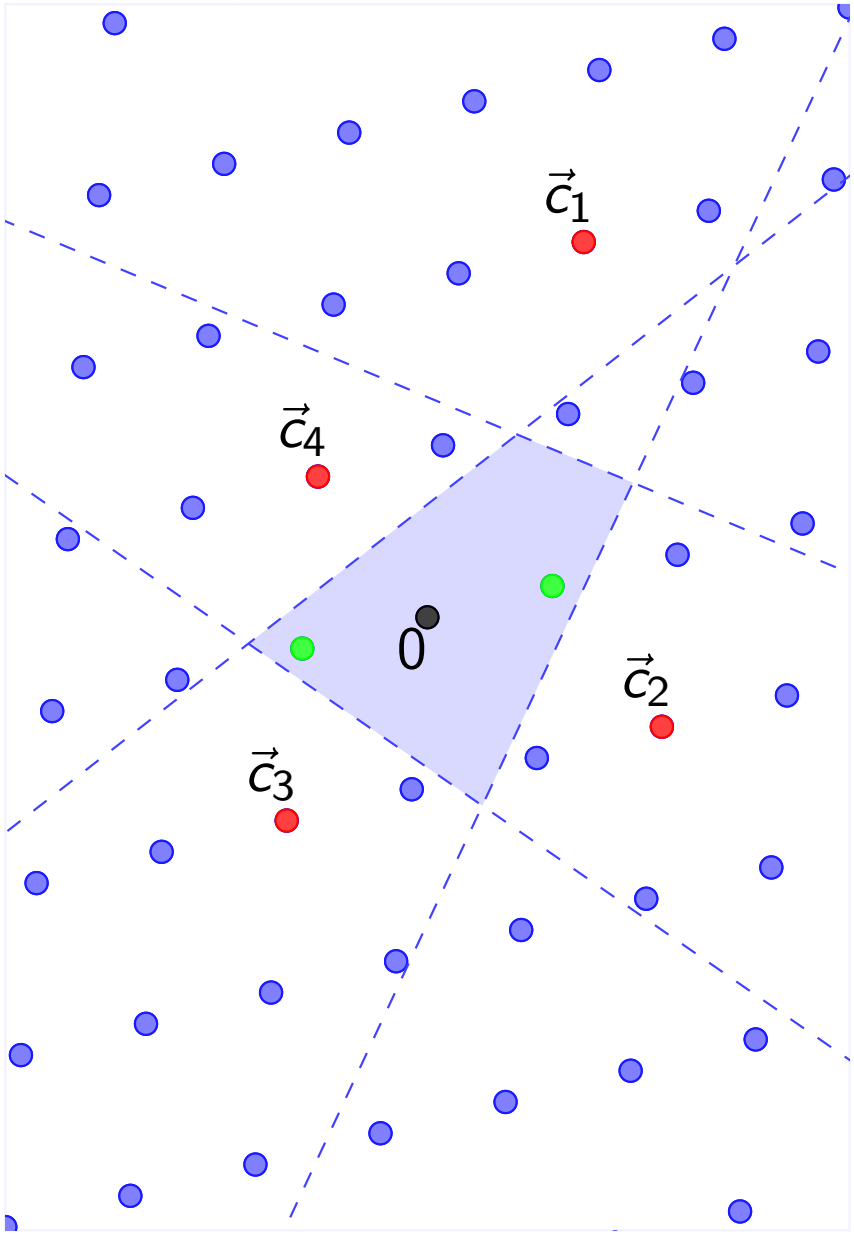
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
}

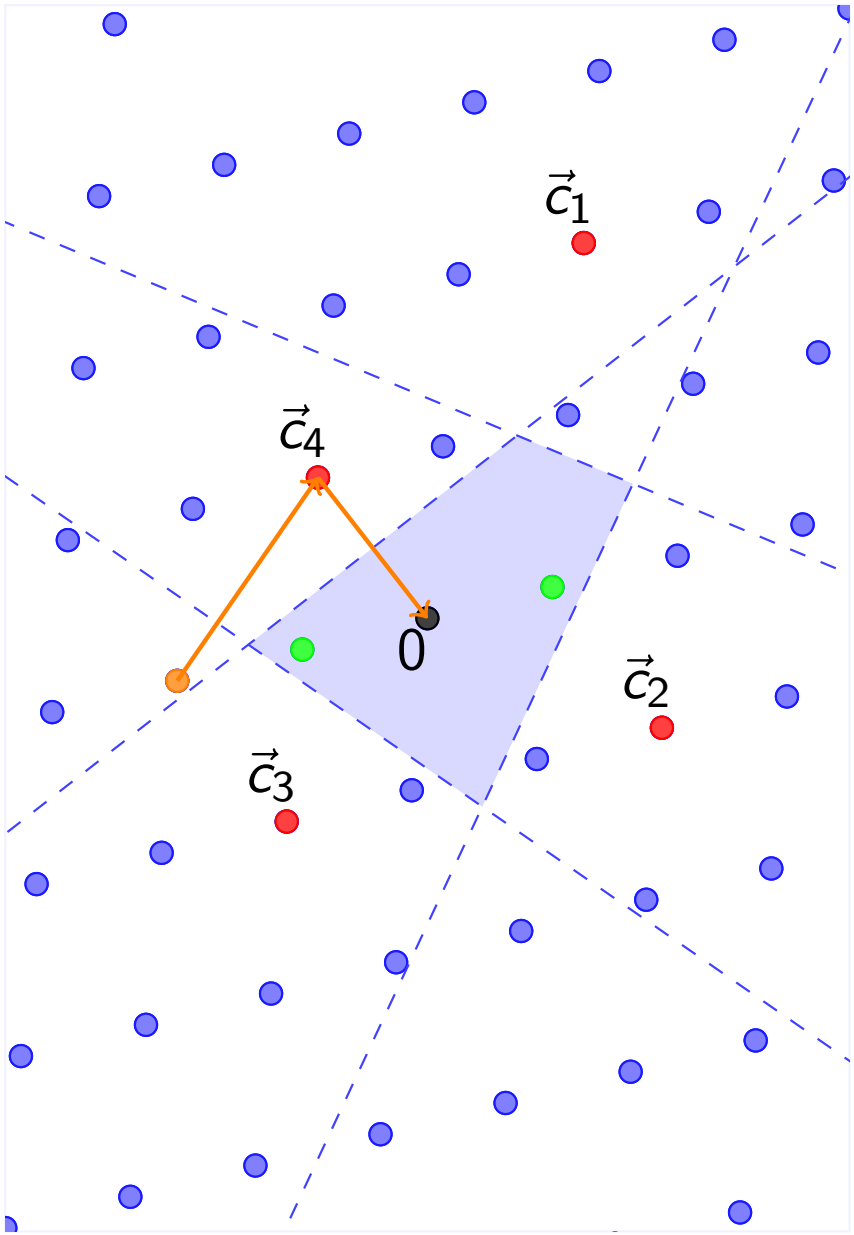
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
}

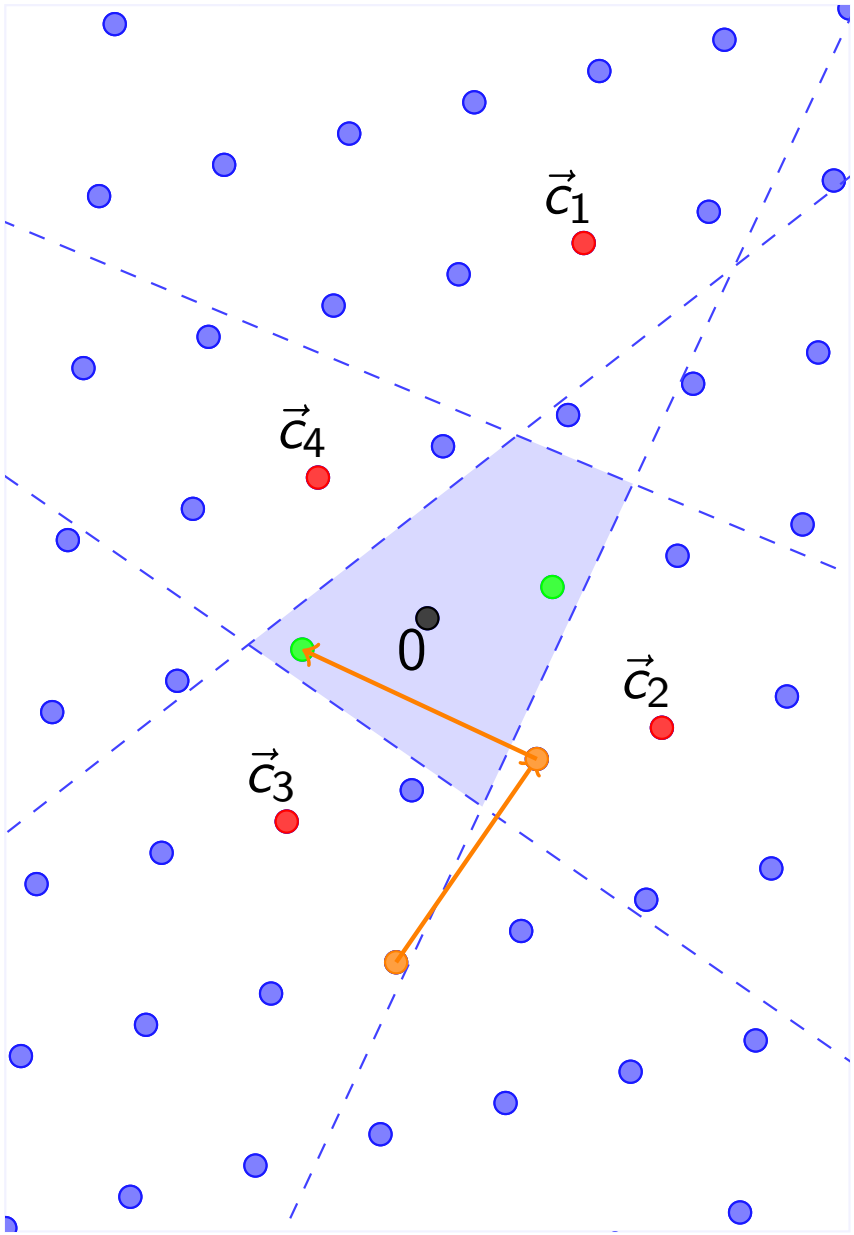
List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
}

List Sieve - Example



Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

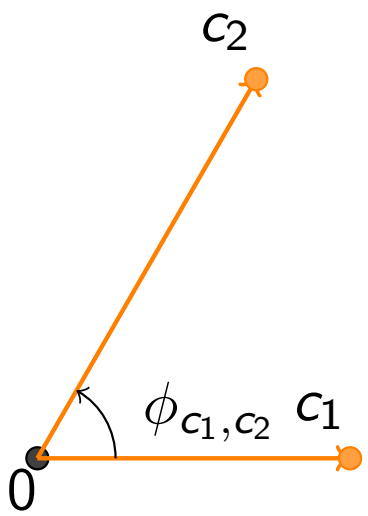
$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`

Analysis of List Sieve

The analysis has two parts:

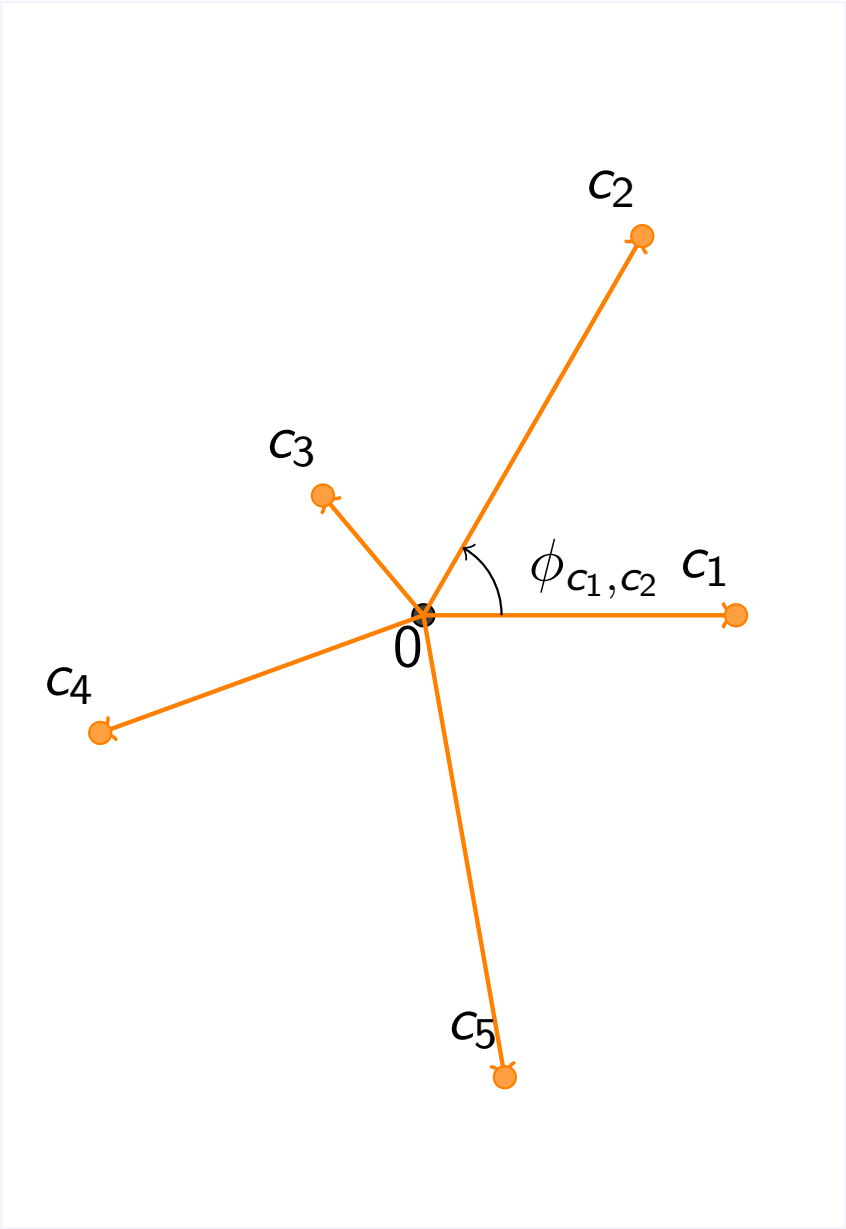
- Space Complexity
  Bound #Points in C
- Time Complexity
  Bound the probability of getting $\vec{0}$ (collision)

Lower bounds on angles $\Rightarrow$ upper bound on points.



Let ϕ_{c_1,c_2} angle between c_1, c_2

Lower bounds on angles $\Rightarrow$ upper bound on points.



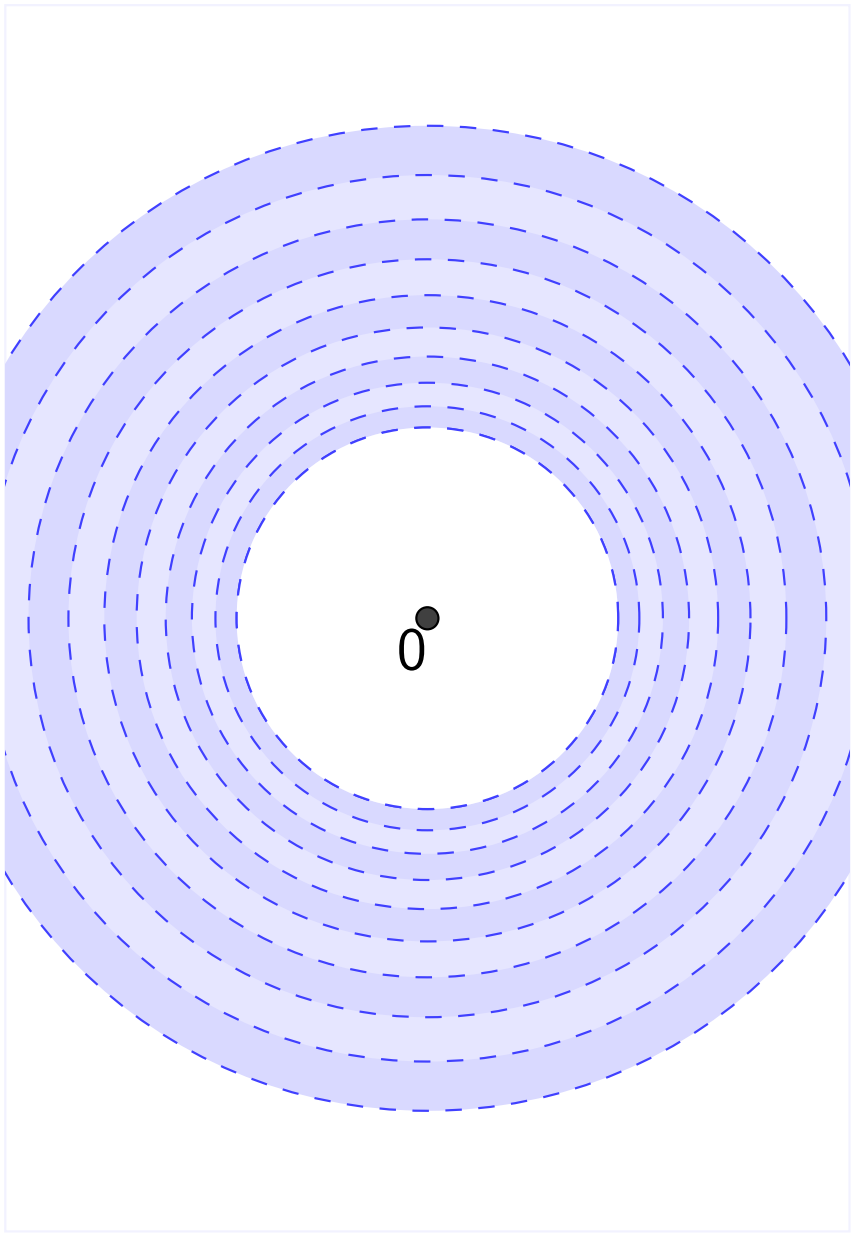
Let ϕ_{c_1,c_2} angle between c_1, c_2

Theorem:
Kabatiansky, Levenshtein 1978

Let set S such that
$\forall c_i, c_j \in S : \phi_{c_i,c_j} > \phi_0$ then:
$|S| \leq 2^{k(\phi_0)n+o(n)}$

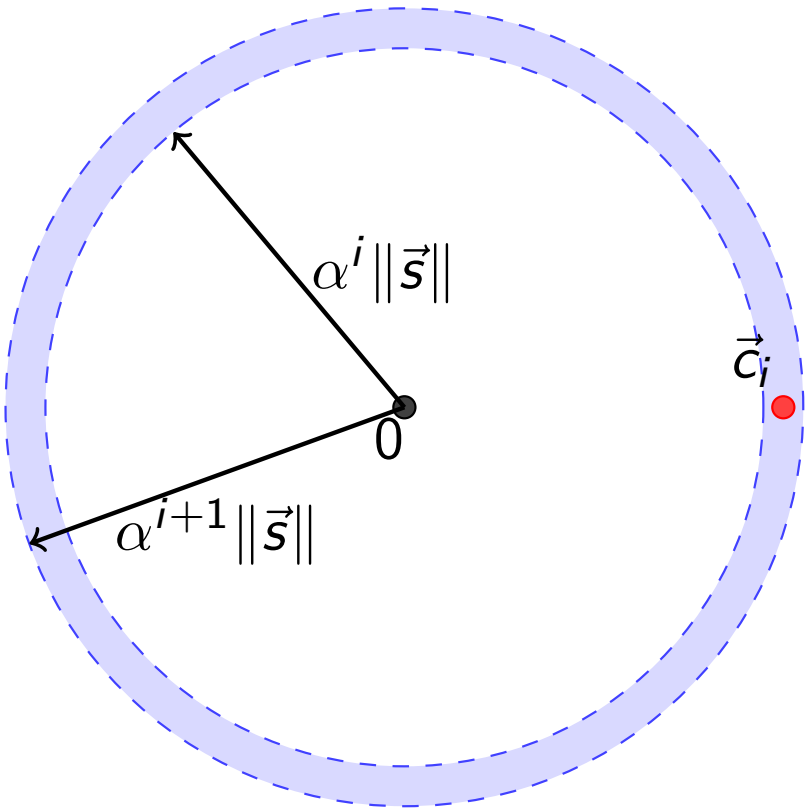
Divide C in subsets with lower bounded angles.

Bounding $|C|$: Spherical Shells



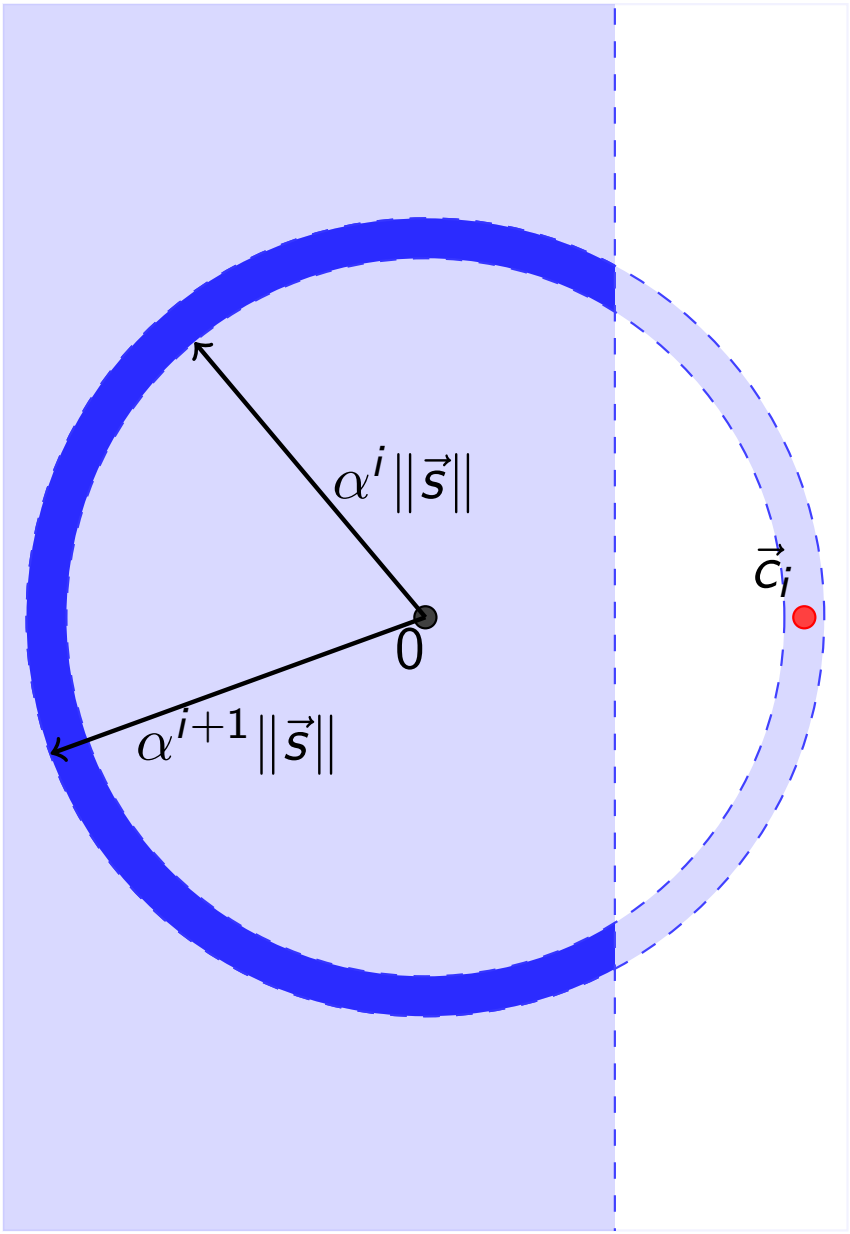
- Divide space to thin shells: $S_i = \text{Shell}(\alpha^i \|\vec{s}\|, \alpha^{i+1} \|\vec{s}\|)$, $1 < \alpha < 1.1$
- C is covered by poly(n) such shells
- If $\forall i$ we lower bound the angles of $S_i \cap C$ then: $|S_i \cap C| \leq 2^{kn}$ and $|C| \leq poly(n)2^{kn}$

Bounding the angles of points in C



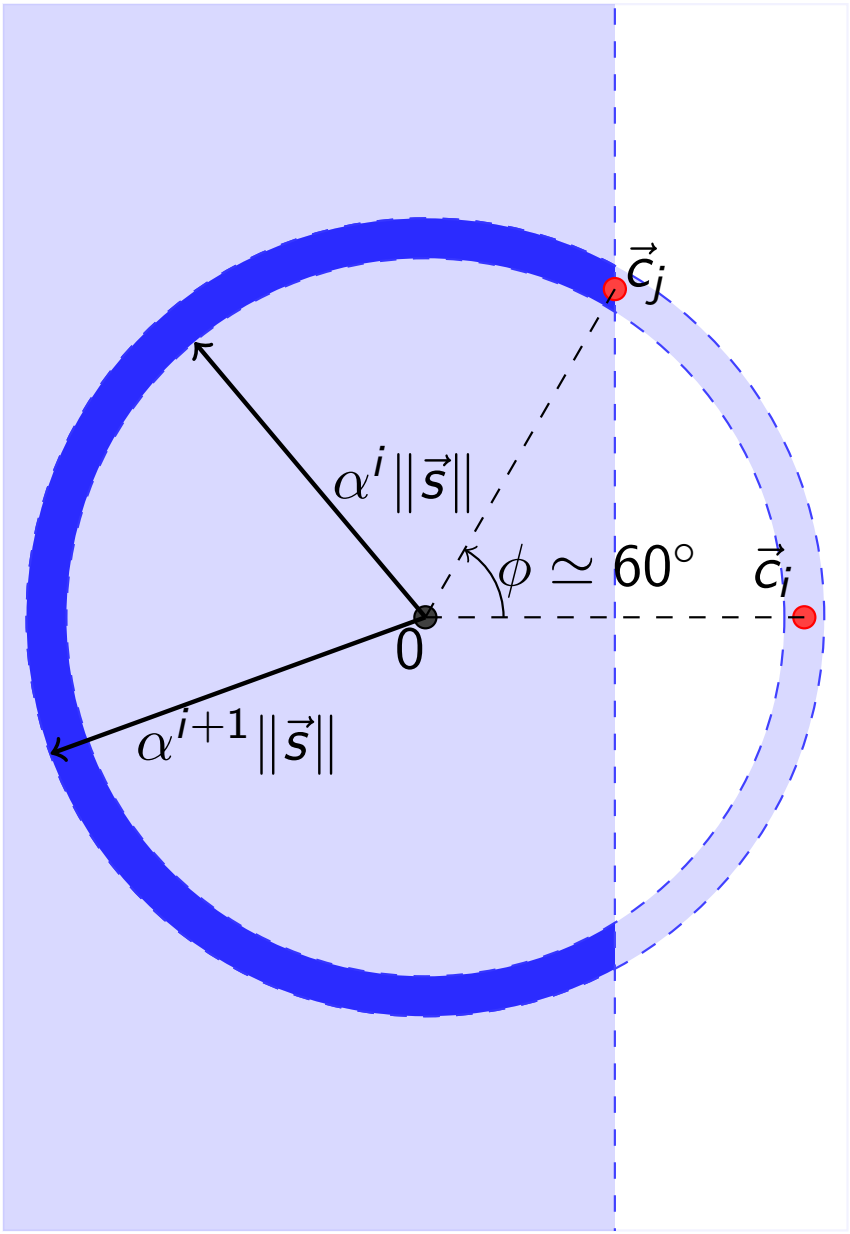
- Consider one shell S_i
- c_i is a point in $C \cap S_i$

Bounding the angles of points in C



- Consider one shell S_i
- c_i is a point in $C \cap S_i$
- A new point should be in $\vec{0}$-halfspace

Bounding the angles of points in C



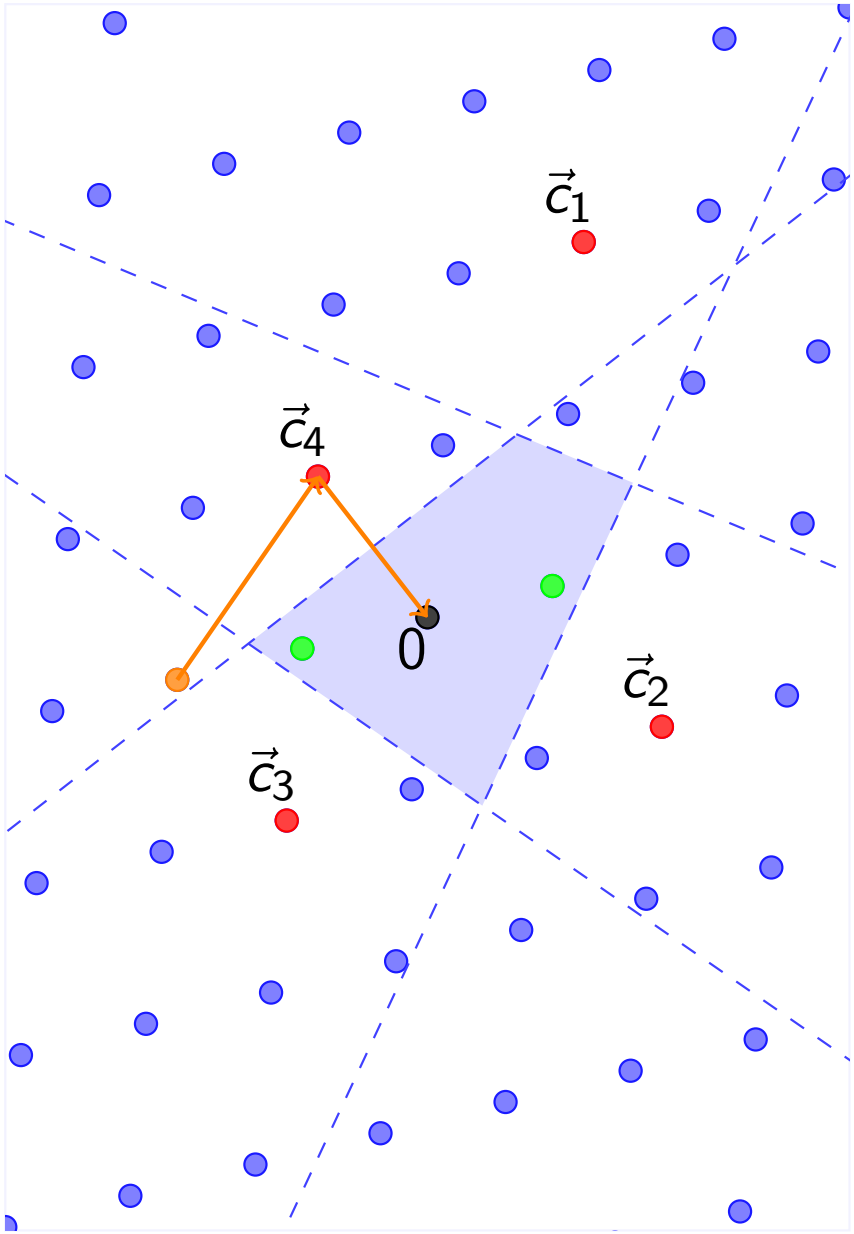
- Consider one shell S_i
- c_i is a point in $C \cap S_i$
- A new point should be in $\vec{0}$-halfspace
- Therefore $\phi_{\vec{c}_i,\vec{c}_j}$ is lower bounded

Analysis of List Sieve

The analysis has two parts:

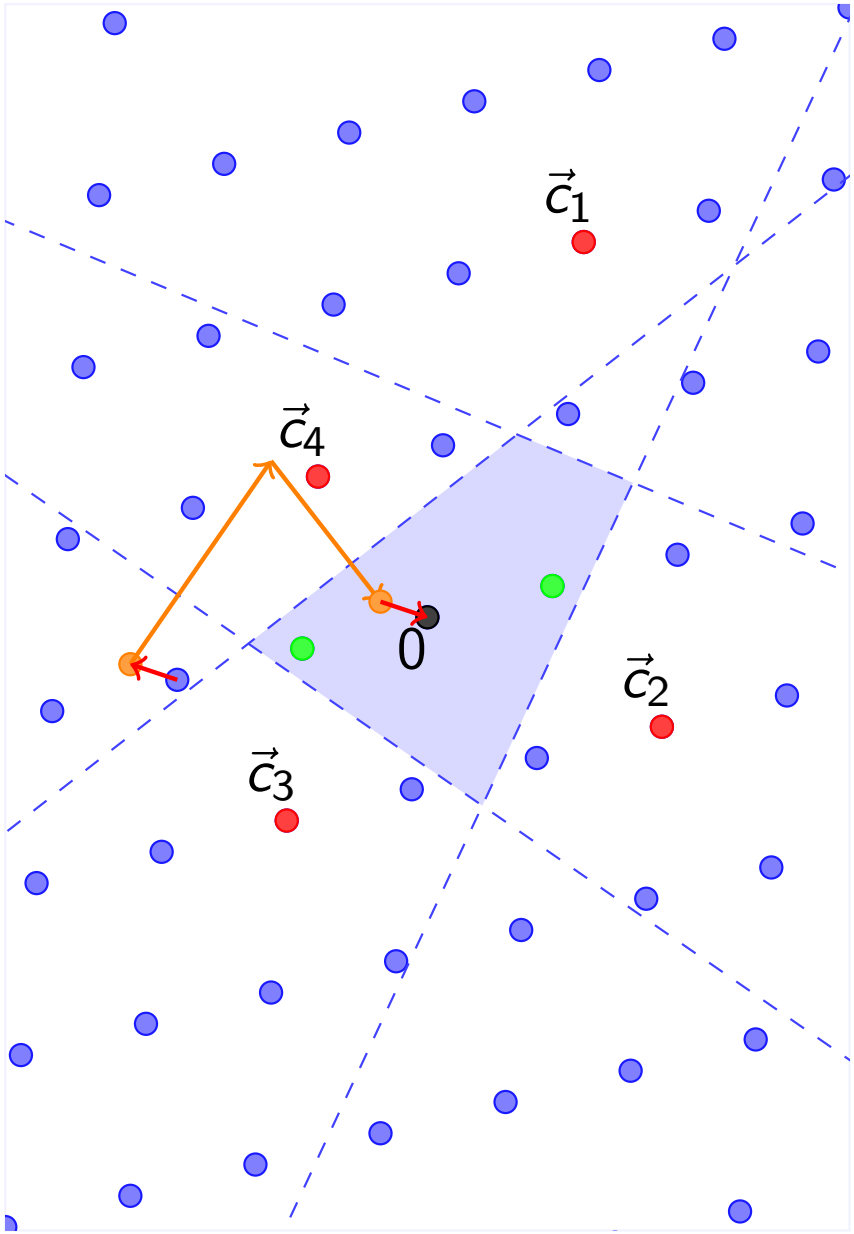
- Space Complexity
  Bound #Points in C
- Time Complexity
  Bound the probability of getting $\vec{0}$ (collision)

Perturbation Technique, AKS



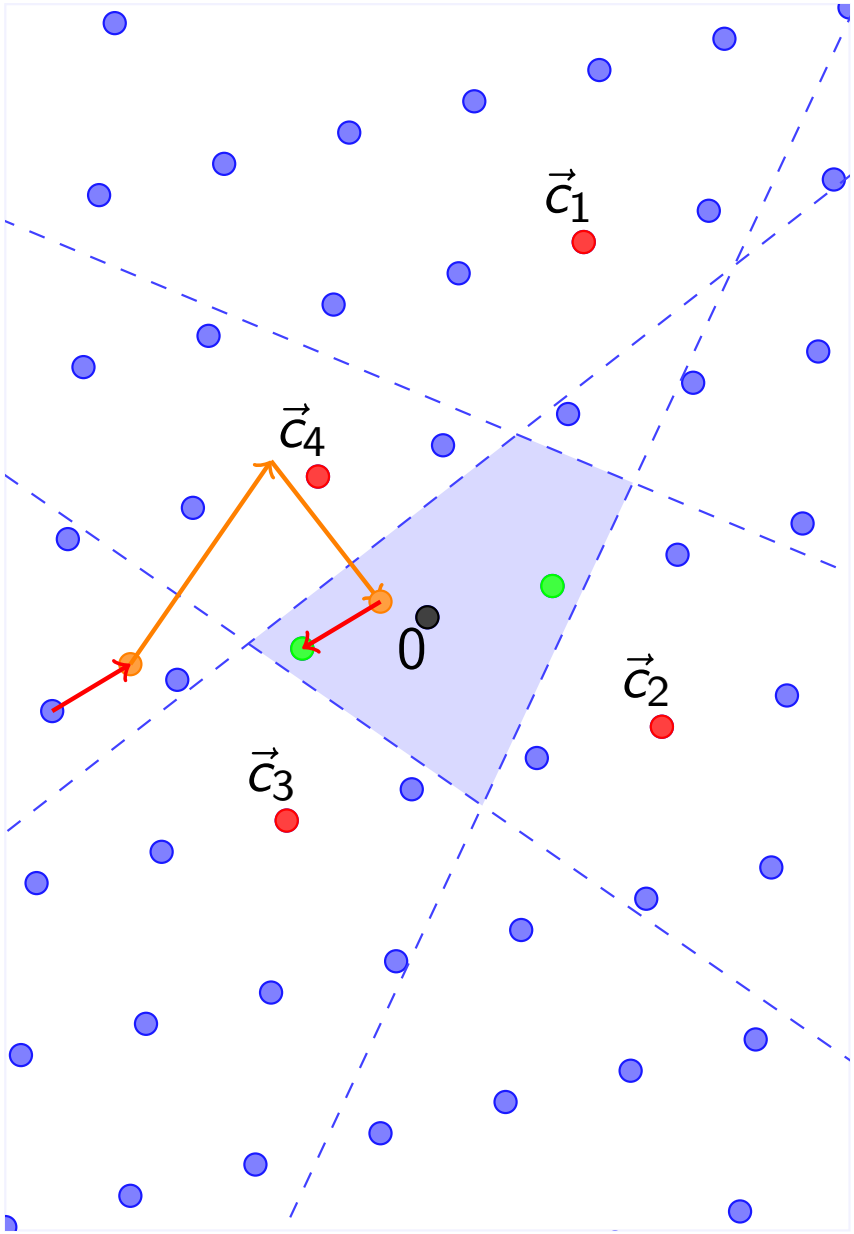
- Instead of sampling a lattice point $\vec{p}$

Perturbation Technique, AKS



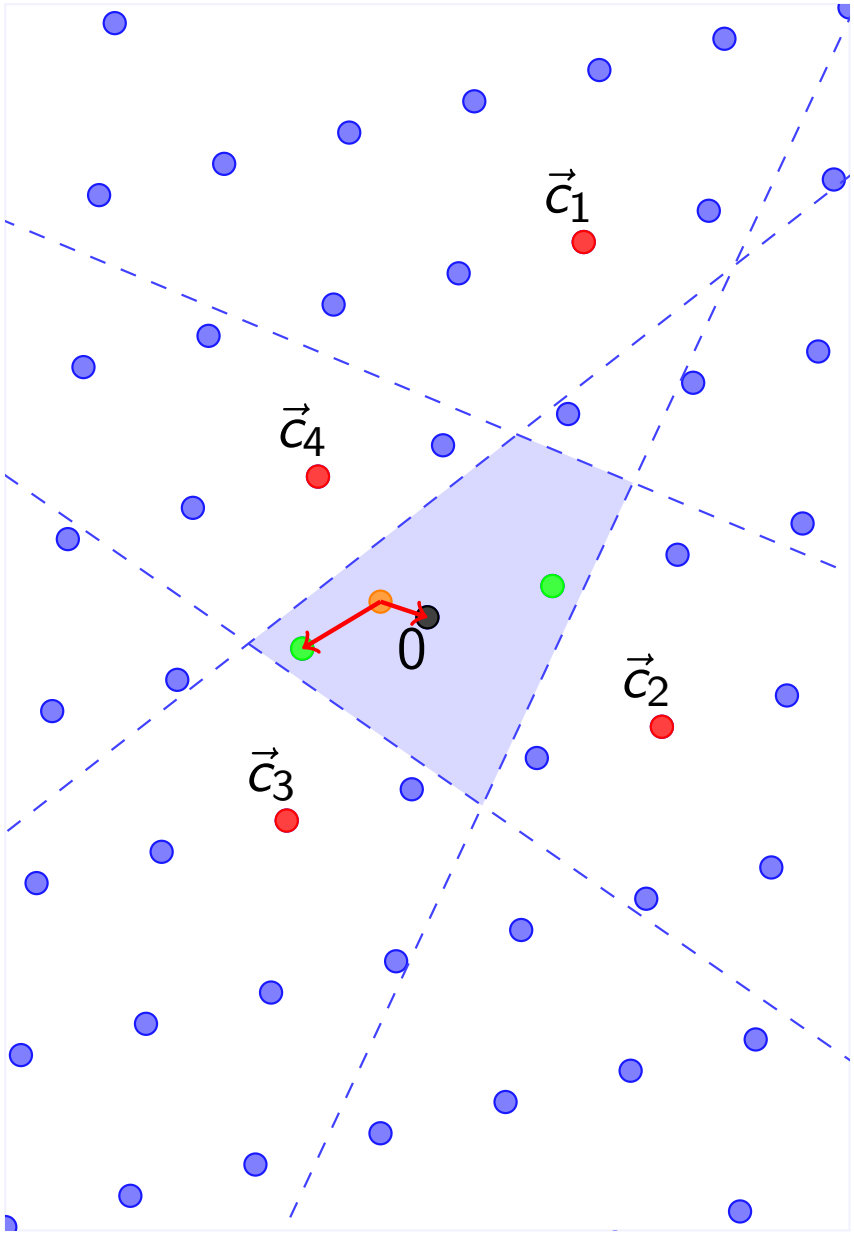
- Instead of sampling a lattice point $\vec{p}$
- Sample $(\vec{p}, \vec{\epsilon})$, so that $\vec{p} - \vec{\epsilon} \in \mathcal{L}$
- Reduce$(\vec{p}, C)$ and consider $\vec{p}' - \vec{\epsilon}$

Perturbation Technique, AKS



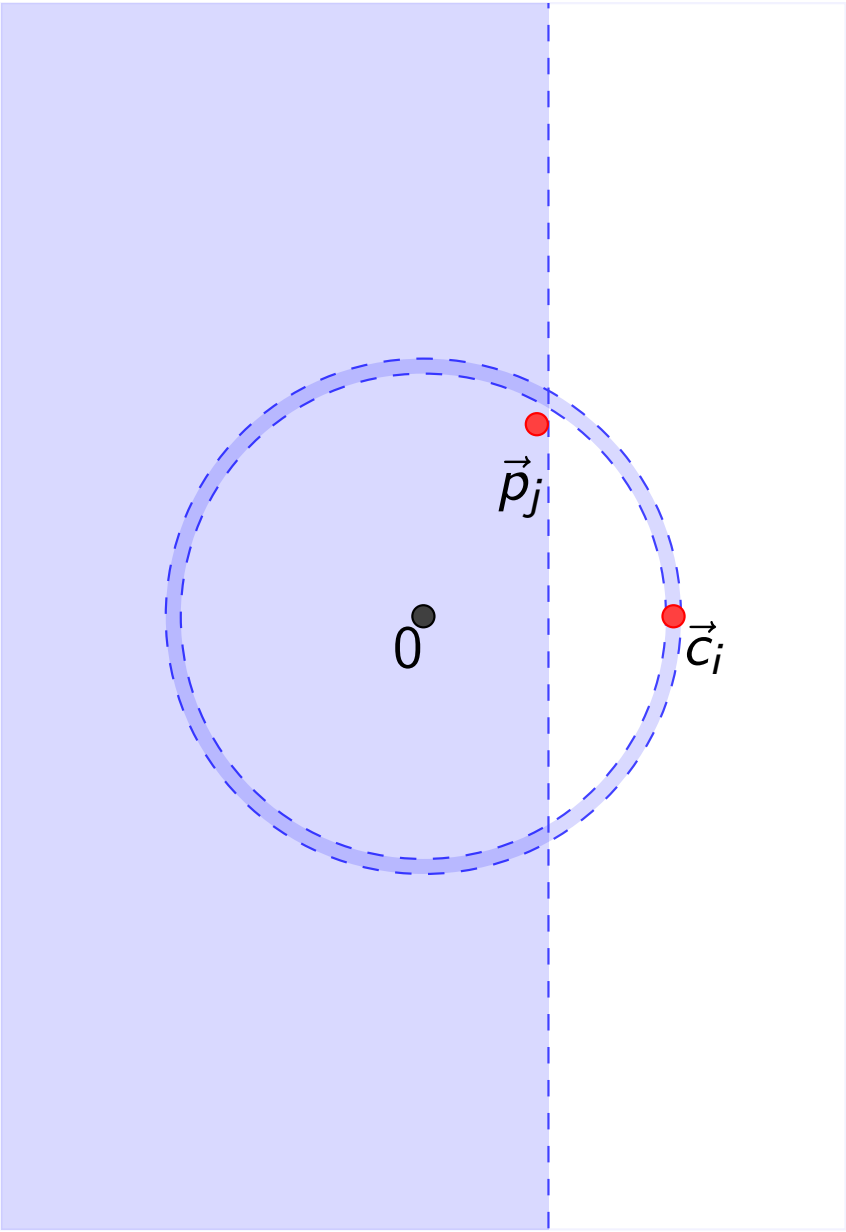
- Instead of sampling a lattice point $\vec{p}$
- Sample $(\vec{p}, \vec{\epsilon})$, so that $\vec{p} - \vec{\epsilon} \in \mathcal{L}$
- Reduce$(\vec{p}, C)$ and consider $\vec{p}' - \vec{\epsilon}$
- $\vec{p}$ can correspond to two lattice points
- Reduce is oblivious of $\vec{\epsilon}$,

Perturbation Technique, AKS



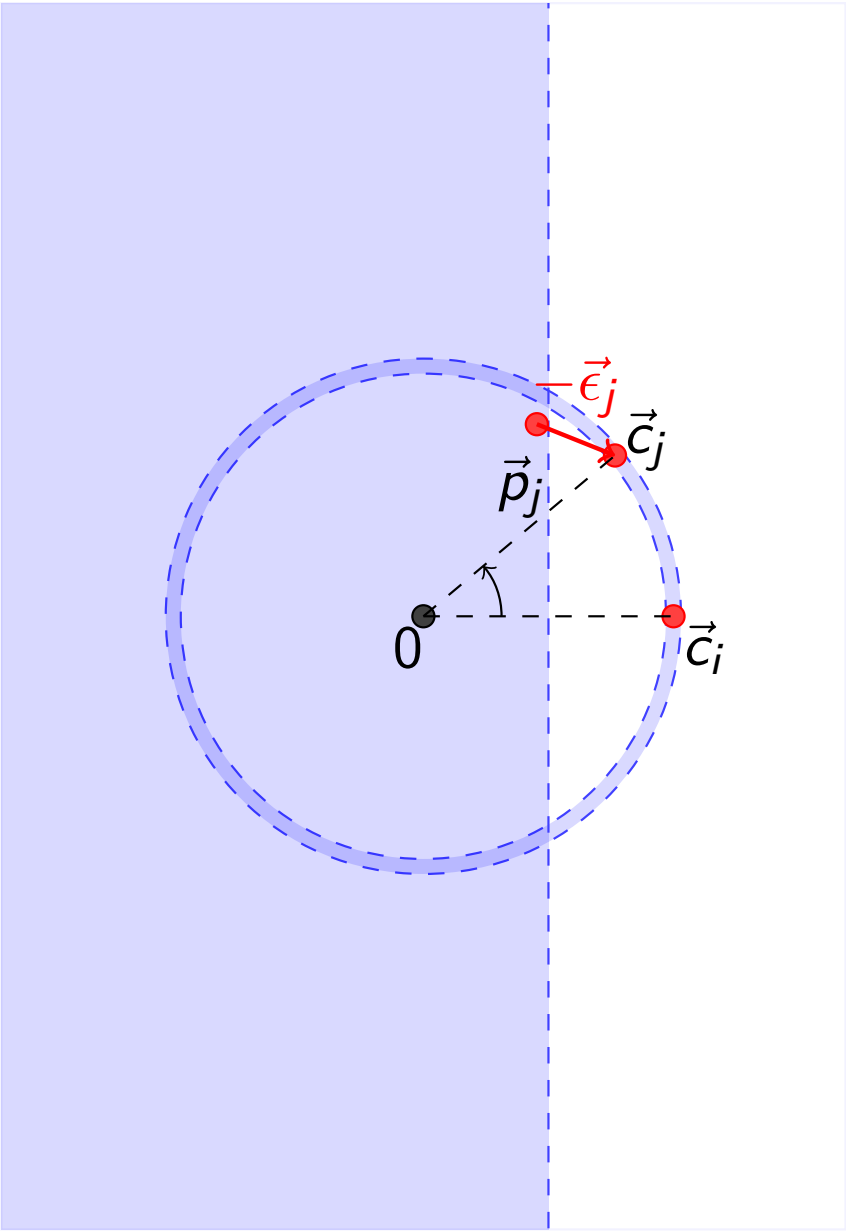
- Instead of sampling a lattice point $\vec{p}$
- Sample $(\vec{p}, \vec{\epsilon})$, so that $\vec{p} - \vec{\epsilon} \in \mathcal{L}$
- $\texttt{Reduce}(\vec{p}, C)$ and consider $\vec{p}' - \vec{\epsilon}$
- $\vec{p}$ can correspond to two lattice points
- $\texttt{Reduce}$ is oblivious of $\vec{\epsilon}$,
- Lots of collisions $\Rightarrow$ lots of points near $\vec{0}$ (and near $\vec{s}$)
- $\Rightarrow$ non negligible probability of finding $\vec{s}$

Disadvantages of Perturbations



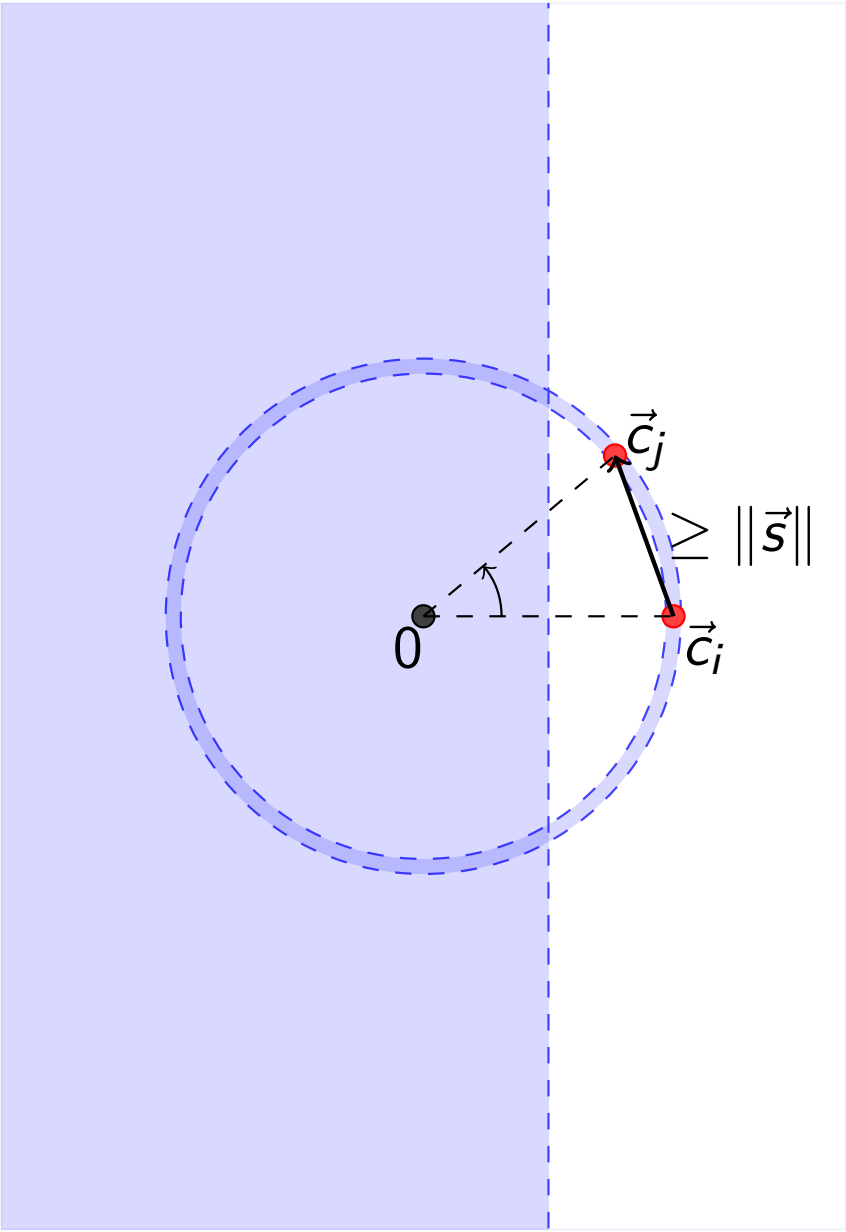
- After `Reduce` $\vec{p}_j$ is further from $\vec{c}_j$

Disadvantages of Perturbations



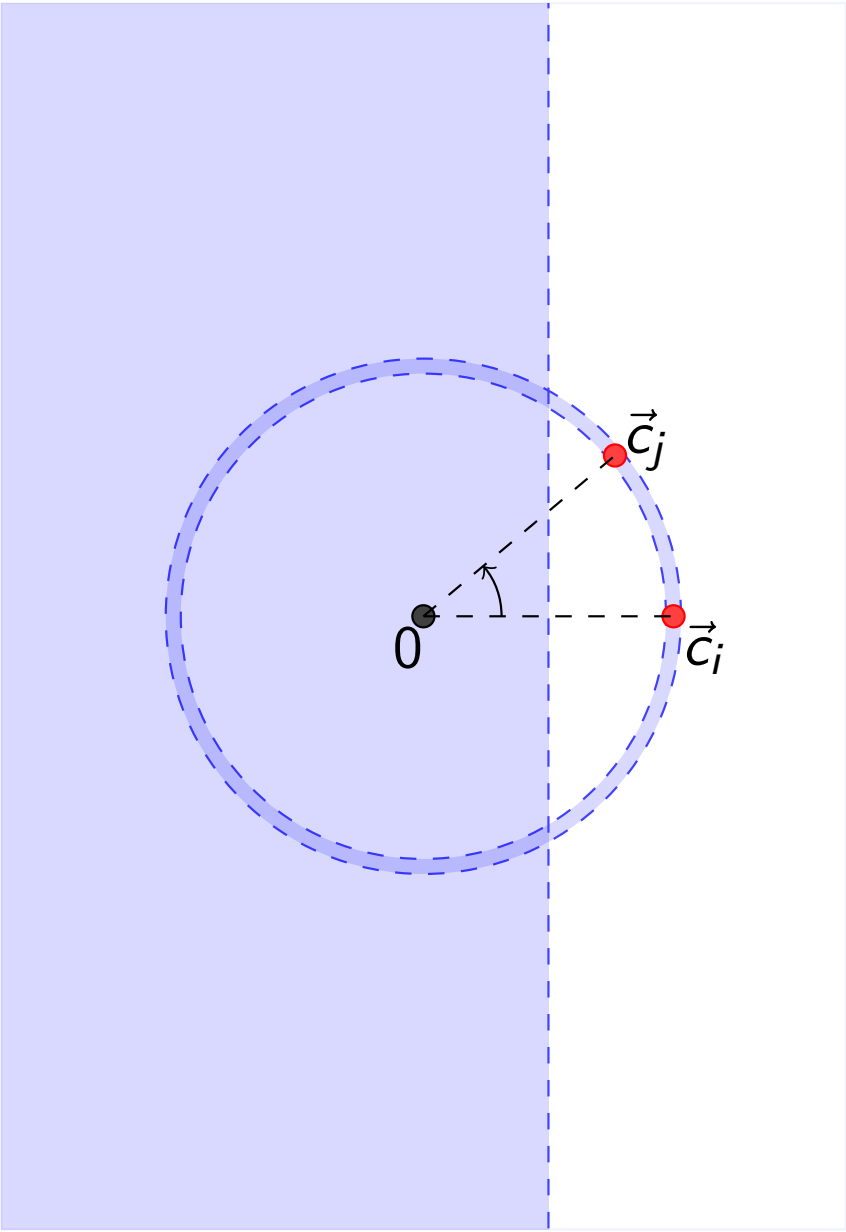
- After `Reduce` $\vec{p}_j$ is further from $\vec{c}_j$
- But the perturbation decreases the minimum angles

Disadvantages of Perturbations



- After `Reduce` $\vec{p}_j$ is further from $\vec{c}_j$
- But the perturbation decreases the minimum angles
- This is especially bad for shells near $\vec{0}$

Disadvantages of Perturbations



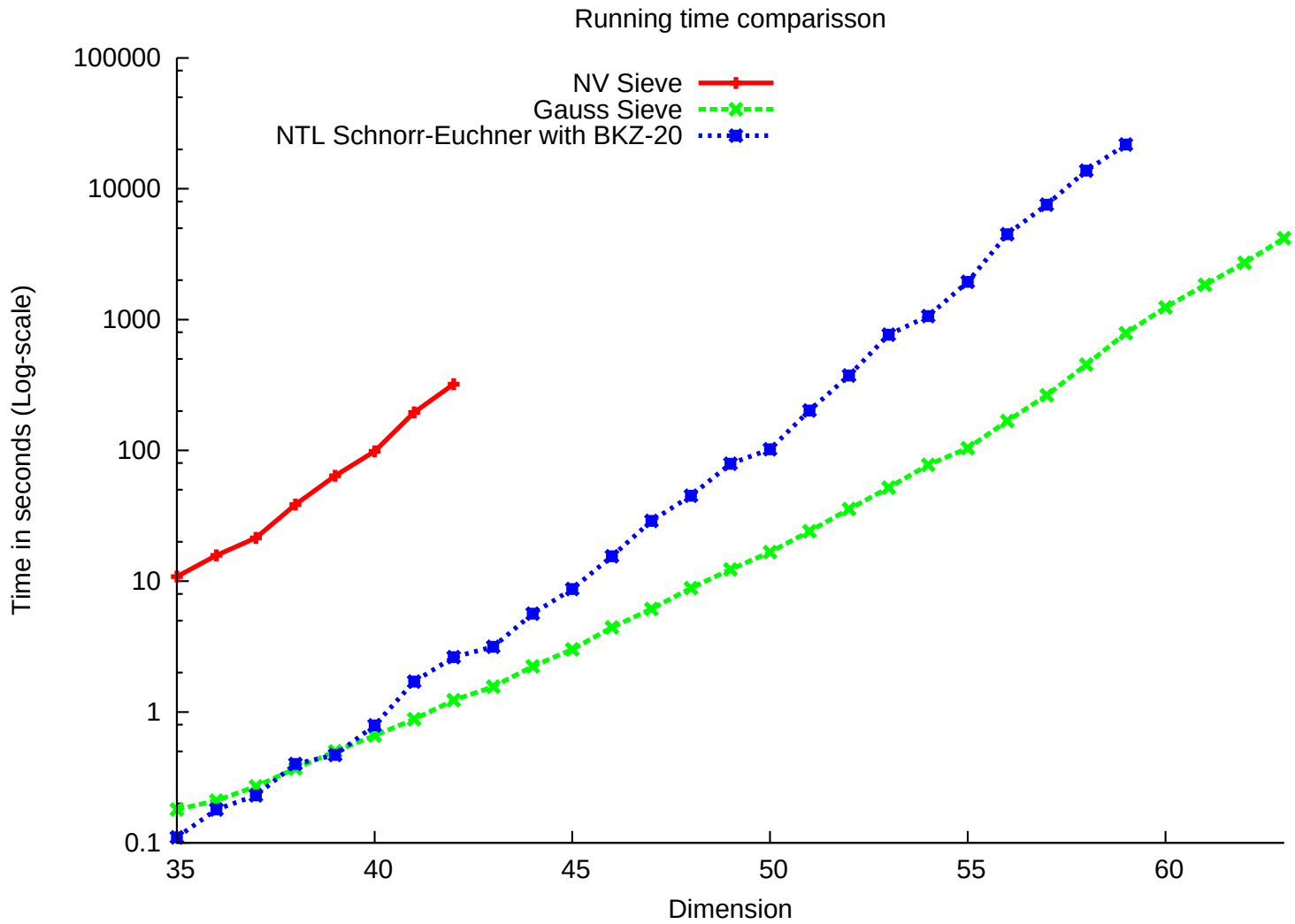
- After Reduce $\vec{p}_j$ is further from $\vec{c}_j$
- But the perturbation decreases the minimum angles
- This is especially bad for shells near $\vec{0}$
- Perturbations greatly increase space bounds: $2^{0.41n+o(n)}$ VS $2^{1.33n+o(n)}$

Practical variant Gauss Sieve

Practical implementation – Gauss Sieve:

- No perturbations (Proposed in [NV 2008])
- The list C is fully reduced:
  $\forall \vec{c}_i, \vec{c}_j \in C \ \|\vec{c}_i - \vec{c}_j\| \geq \|\vec{c}_i\|$
  Therefore $\phi_{\vec{c}_i,\vec{c}_j} \geq 60°$!

Running time comparison



Gauss Sieve

- $\simeq 10^2$ to 10^3 faster, $\simeq 70\times$ less points
- $2^{0.21n+o(n)}$ space bound
- Faster than NTL for dimensions > 40
- Bottleneck is time, not space

Implementation available at `http://cse.ucsd.edu/~pvoulgar/`

Summary

We improve the work of [AKS 2001] and [NV 2008] with:

- List Sieving:
  - Lower space bounds in theory
  - Faster implementations in practice
  - Better algorithmic intuition
- Connection with spherical codes:
  - Use of powerful theorems for analysis [KL 1978]
- Faster heuristic:
  - Much faster, less space than previous implementation

Open Problems

Open Problems:

- SVP in 2^{cn} time with poly(n) space
- Other lattice problems in 2^{cn} time/space (CVP, SIVP)
- Deterministic variant

Specific to our work:

- Bound time complexity without perturbations

Thank you for attending!