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, ...

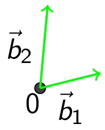
Signifficance of Shortest Vector Problem

Foundational problem for lattices:

- Exact SVP is known to be NP-complete.
- Although in most applications we need approximations.
- Approximation algorithms utilize SVP-oracles.

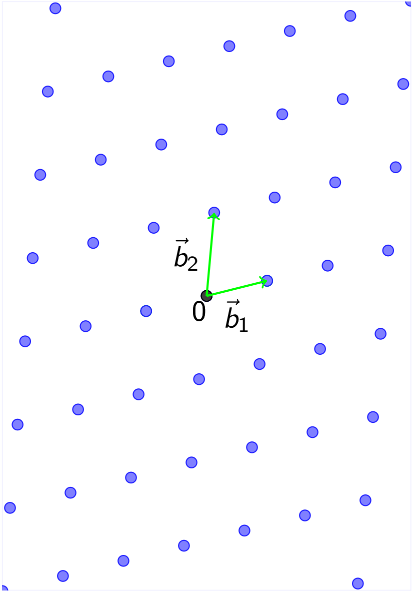
Two techniques for exact-SVP:

- Enumeration. Time: $2^{O(nlogn)}$
- Sieving. Time: $2^{O(n)}$, Space: $2^{O(n)}$



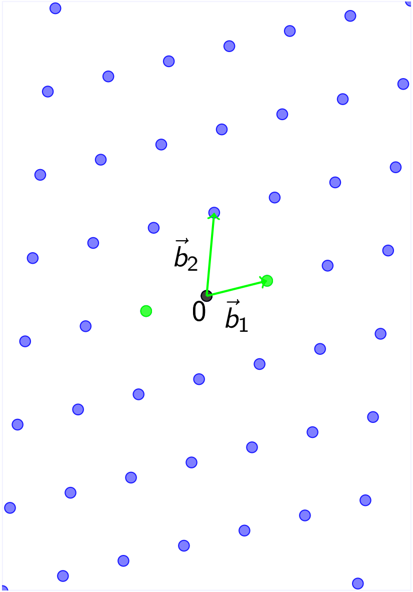
- Given a *basis*:
  $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$
  of m linearly independent vectors in $\mathbb{R}^n$.

Lattices - Shortest Vector Problem (SVP)

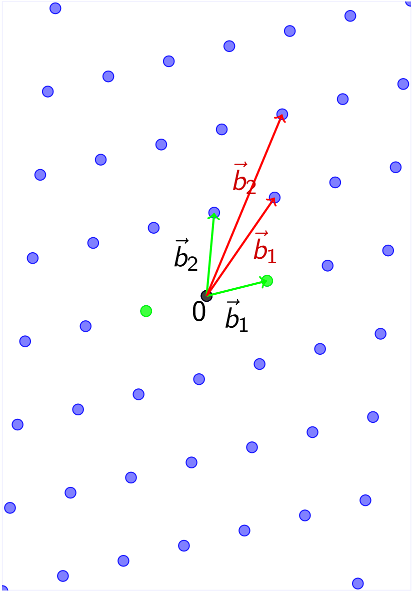


- Given a *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$ of m linearly independent vectors in $\mathbb{R}^n$.
- *Lattice* is: $\mathcal{L}(\mathbf{B}) = \{\vec{p} = \sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$.

Lattices - Shortest Vector Problem (SVP)

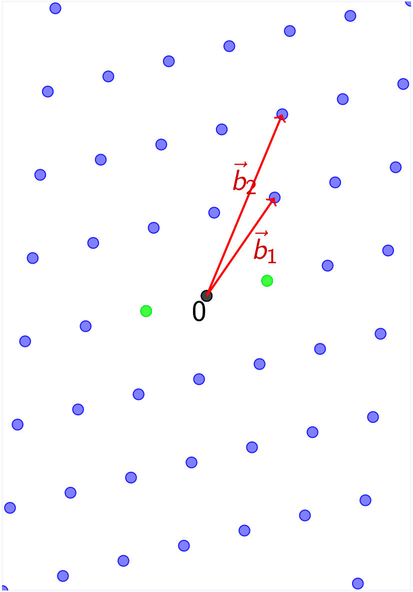


- Given a *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$ of m linearly independent vectors in $\mathbb{R}^n$.
- *Lattice* is: $\mathcal{L}(\mathbf{B}) = \{\vec{p} = \sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$.
- *Shortest lattice point*: $\vec{s} \in \mathcal{L}(\mathbf{B}) \setminus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



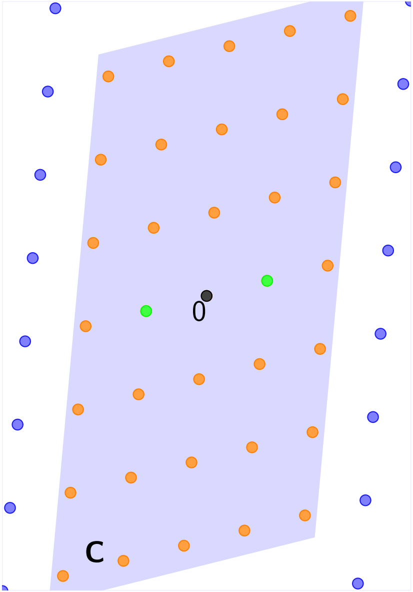
- Given a *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$ of m linearly independent vectors in $\mathbb{R}^n$.
- *Lattice* is: $\mathcal{L}(\mathbf{B}) = \{\vec{p} = \sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$.
- *Shortest lattice point*: $\vec{s} \in \mathcal{L}(\mathbf{B}) \setminus \vec{0}$ 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.

Lattices - Shortest Vector Problem (SVP)



- Given a *basis*: $\mathbf{B} = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m\}$ of m linearly independent vectors in $\mathbb{R}^n$.
- *Lattice* is: $\mathcal{L}(\mathbf{B}) = \{\vec{p} = \sum a_i \cdot \vec{b}_i, a_i \in \mathbb{Z}\}$.
- *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 **B**, find a shortest lattice point.

Solving SVP: Enumeration

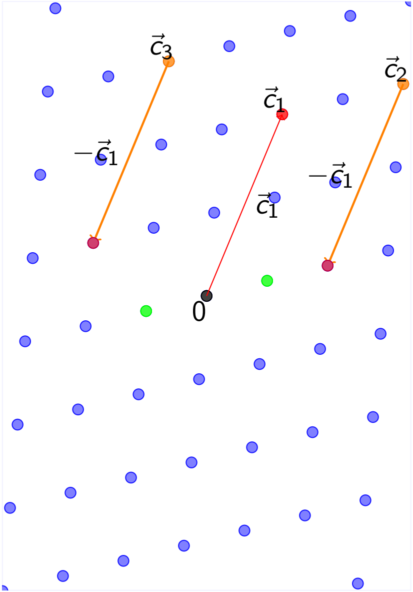


Main idea

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

- Advantages:
  - Space requirements.
  - Fast heuristics.
- Disadvantages:
  - #Points can be $2^{O(n \log n)}$

Solving SVP: Sieving – Prelude



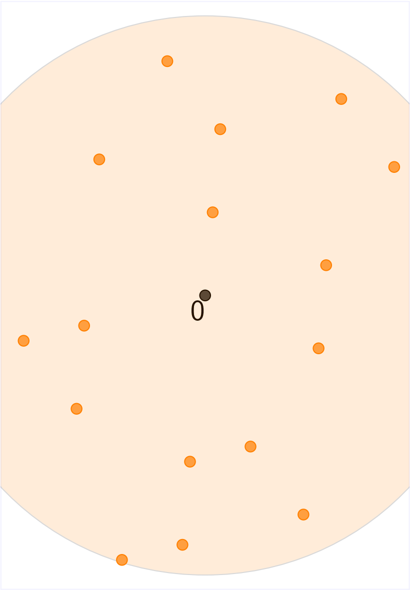
Notice that:

If $\vec{c}_1, \vec{c}_2 \in \mathcal{L} \Rightarrow \vec{c}_2 - \vec{c}_1 \in \mathcal{L}$.

Main idea

Subtract nearby points to get shorter vectors.

Solving SVP: Sieving

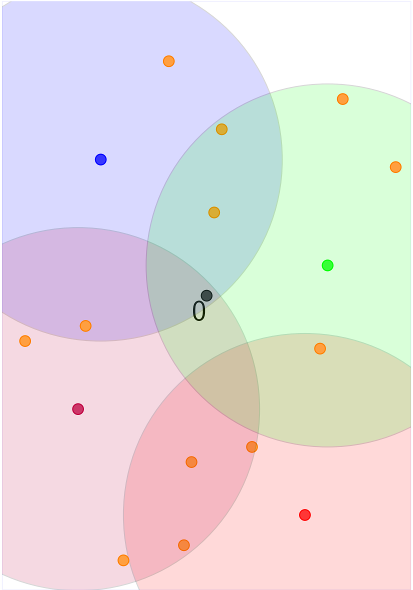


Main idea

Sample 2^{cn} points.
Use "centers" to decrease the norms of near-by points.

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

Solving SVP: Sieving

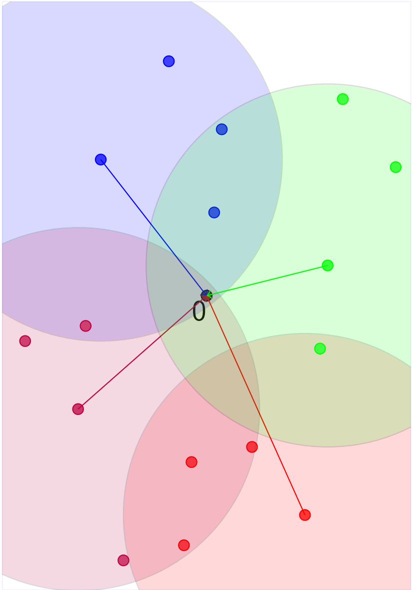


Main idea

Sample 2^{cn} points.
Use "centers" to decrease the norms of near-by points.

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

Solving SVP: Sieving

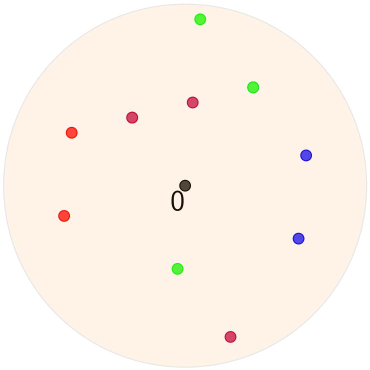


Main idea

Sample 2^{cn} points.
Use "centers" to decrease the norms of near-by points.

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

Solving SVP: Sieving



Main idea

Sample 2^{cn} points.
Use "centers" to decrease the norms of near-by points.

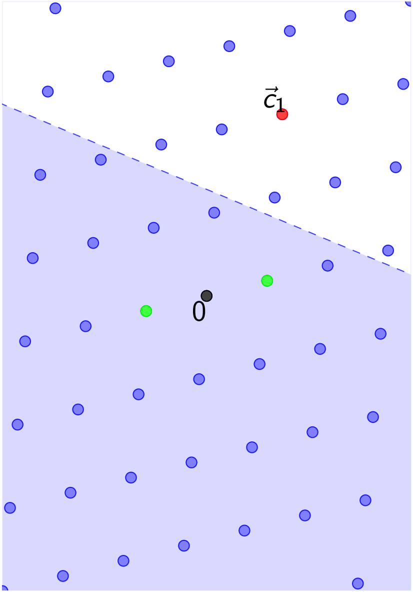
- Advantages:
  - #Points bounded by $2^{O(n)}$
- Disadvantages:
  - Space complexity of $2^{O(n)}$
  - Impractical ???

Time-line: Sieving Algorithms

Year, Authors	Time	Space	Practice
2001, AKS	$2^{O(n)}$	$2^{O(n)}$	–
2004, R	2^{16n}	2^{8n}	–
2008, NV	$2^{5.9n}$	$2^{2.95n}$	Practical
2010, MV	$2^{3.2n}$	$2^{1.33n}$	$> 10^2$ speed-up
2010, PS	$2^{2.465n}$	$2^{1.233n}$	–

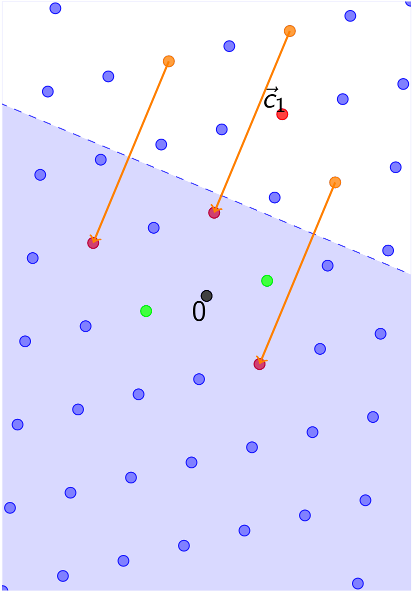
Table: Time-line of Sieving Algorithms

Optimal use of centers



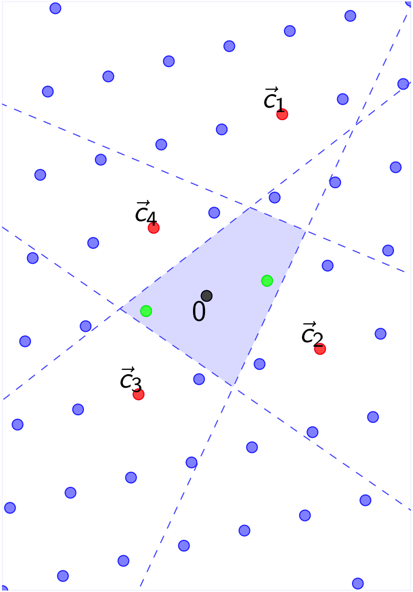
- Every point $\vec{c}_i$, defines two half-spaces.

Optimal use of centers



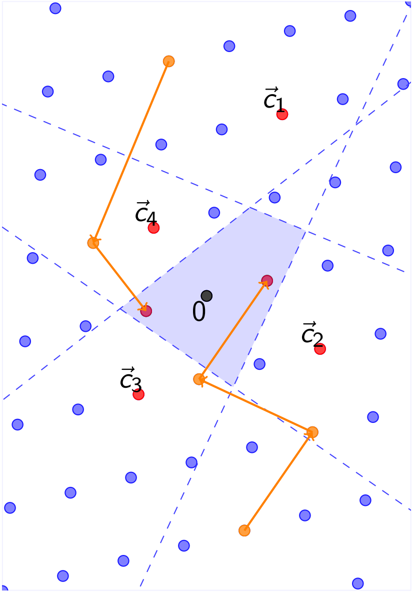
- Every point $\vec{c}_i$, defines two half-spaces.
- Subtracting $\vec{c}_i$, brings any point in the $\vec{0}$ halfspace.

Optimal use of centers



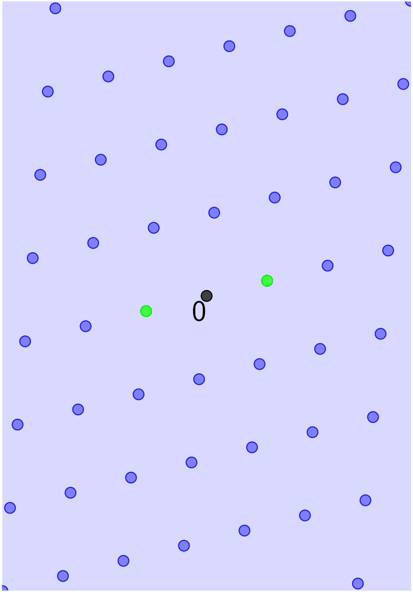
- Every point $\vec{c}_i$, defines two half-spaces.
- Subtracting $\vec{c}_i$, brings any point in the $\vec{0}$ halfspace.
- Given a set C of $\vec{c}_i$, consider the intersection of the $\vec{0}$ halfspaces.

Optimal use of centers



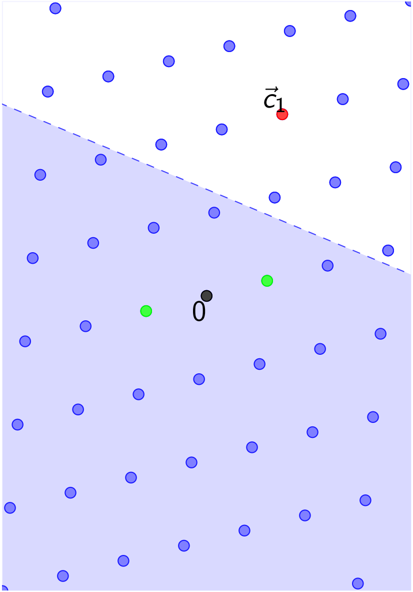
- Every point $\vec{c}_i$, defines two half-spaces.
- Subtracting $\vec{c}_i$, brings any point in the $\vec{0}$ halfspace.
- Given a set C of $\vec{c}_i$, consider the intersection of the $\vec{0}$ halfspaces.
- Subtracting $\vec{c}_i$, brings any point to this intersection.
- We call this procedure $\texttt{Reduce}(\vec{p}, C)$.

List Sieve - Example



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

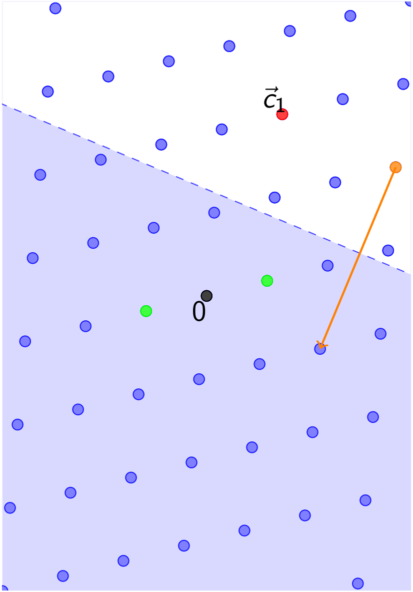
$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `Return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`



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` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`

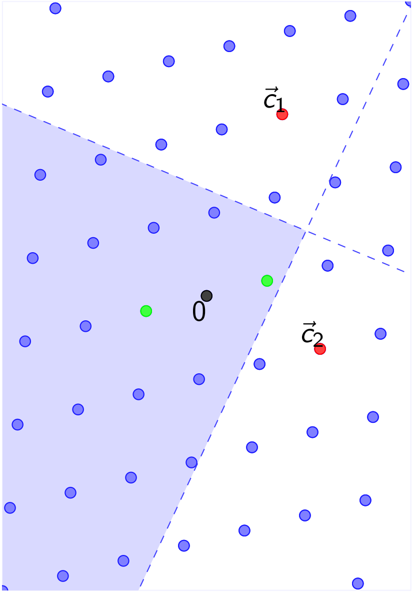
List Sieve - Example



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` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`

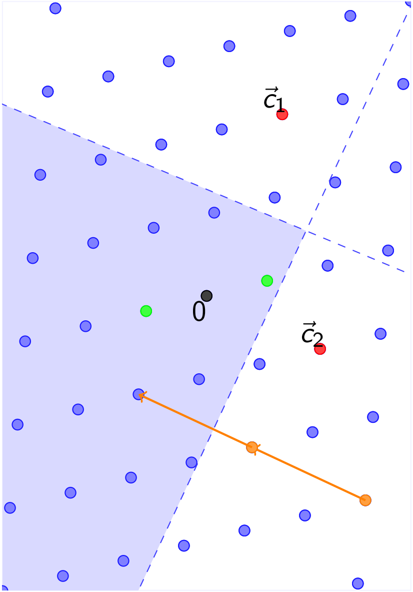
List Sieve - Example



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

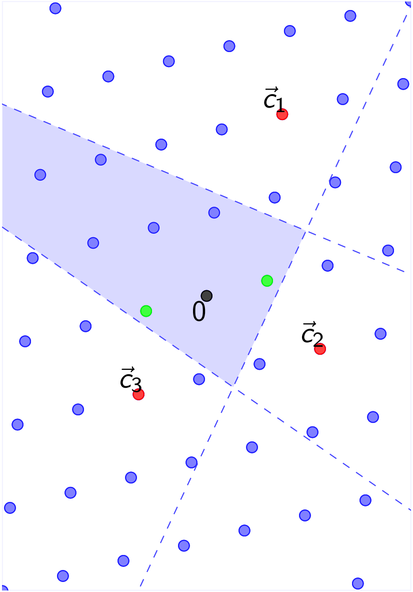
$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `Return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`

List Sieve - Example



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

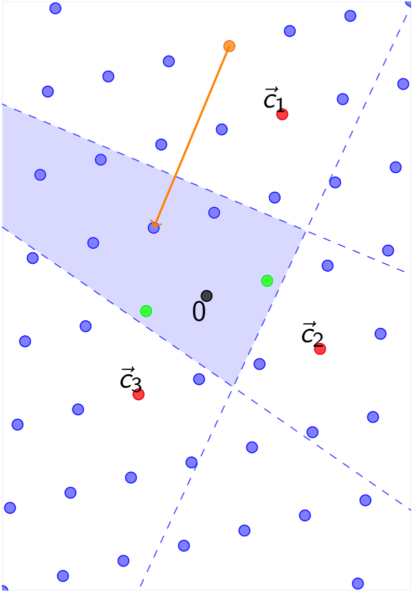
$C \leftarrow \{\}$
`while (true) {`
  $\vec{p} \leftarrow$ `Sample`$(\mathbf{B})$
  $\vec{p}' \leftarrow$ `Reduce`$(\vec{p}, C)$
  `if` $(\vec{p}' = \vec{0})$
    `continue`
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$
    `Return` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
`}`



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` $\vec{p}'$
  $C \leftarrow C \cup \{\vec{p}'\}$
$\}$

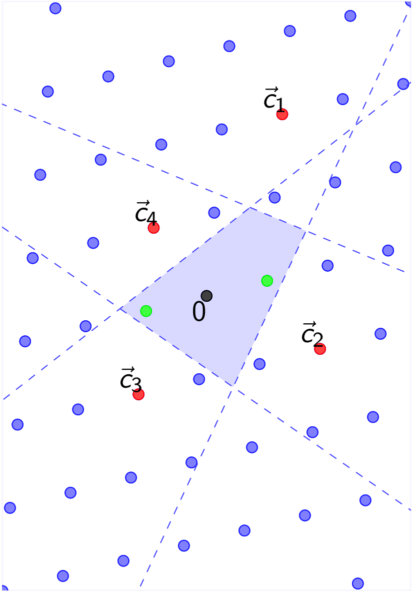
List Sieve - Example



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`

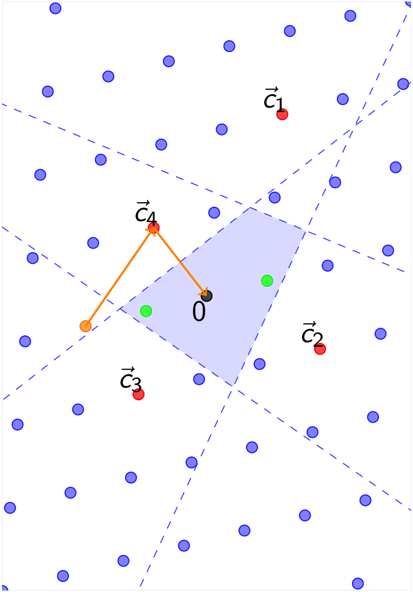
  `if` $(\|\vec{p}'\| \leq \|\vec{s}\|)$

    `Return` $\vec{p}'$

  $C \leftarrow C \cup \{\vec{p}'\}$

`}`

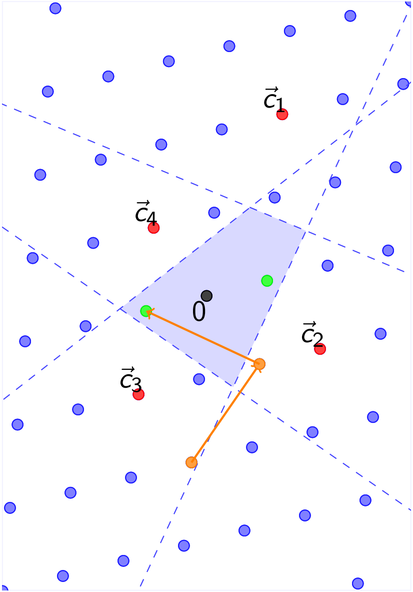
List Sieve - Example



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

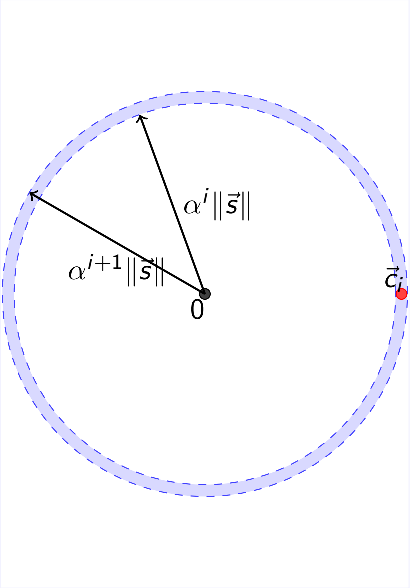


Algorithm: ListSieve($\mathbf{B}$, $\|\vec{s}\|$)

```
C ← {}
while (true) {
   p ← Sample(B)
   p' ← Reduce(p, C)
   if (p' = 0)
      continue
   if (||p'|| ≤ ||s||)
      Return p'
   C ← C ∪ {p'}
}
```

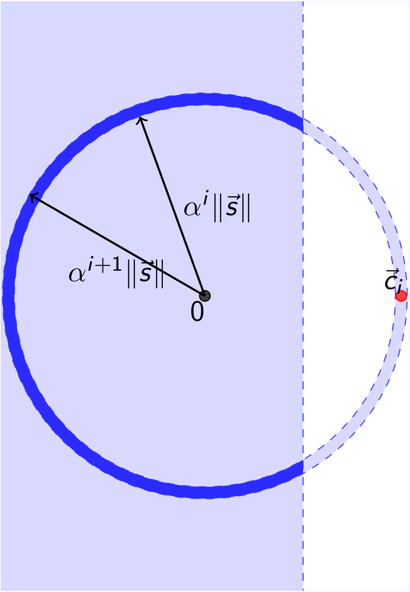
Sieving: Main questions.

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



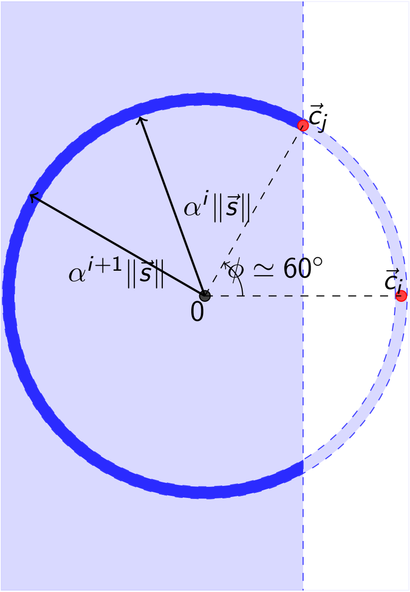
- Consider the points on a thin spherical shell, $S_i = \text{Shell}(\alpha^i \|\vec{s}\|, \alpha^{i+1} \|\vec{s}\|)$, $\alpha > 1$.

Bounding the angles of points in C



- Consider the points on a thin spherical shell, $S_i = \text{Shell}(\alpha^i\|\vec{s}\|, \alpha^{i+1}\|\vec{s}\|)$, $\alpha > 1$.
- Notice that $\vec{c}_j$ should be reduced with $\vec{c}_i$.

Bounding the angles of points in C



- Consider the points on a thin spherical shell, $S_i = \text{Shell}(\alpha^i\|\vec{s}\|, \alpha^{i+1}\|\vec{s}\|)$, $\alpha > 1$.
- Notice that $\vec{c}_j$ should be reduced with $\vec{c}_i$.
- Therefore the $\phi_{\vec{c}_i,\vec{c}_j}$ angle is lower bounded.
- Can we bound #points, with the lower bound of their angles?

Spherical Codes

Theorem (Kabatiansky, Levenshtein KL 1978)

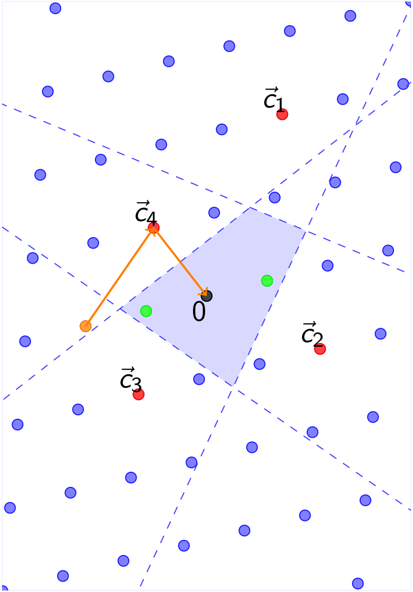
If $\forall \vec{c}_i, \vec{c}_j \in S$, $\phi_{\vec{c}_i,\vec{c}_j} \geq \phi_0$ *then:*

$$|S| \leq 2^{kn+o(n)}, \;\; k = -0.5 log(1 - cos(\phi_0)) - 0.099$$

- $|C \cap S_i|$ is bounded, for every S_i.
- Polynomially many S_i to cover C.
- We can bound $|C| = \sum |C \cap S_i|$ by $poly(n) \cdot 2^{O(n)}$.

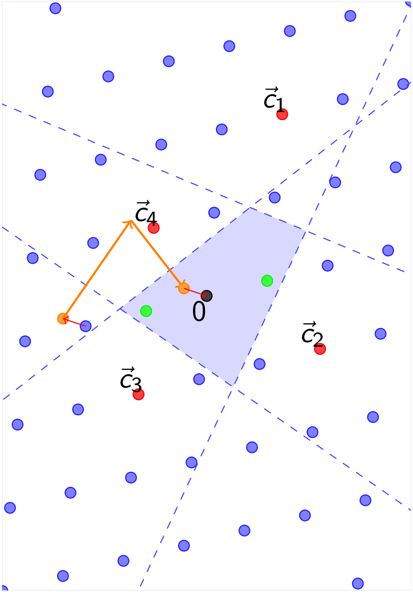
Connection between a sieving algorithms and spherical codes!

Bounding Collisions, Technique of AKS



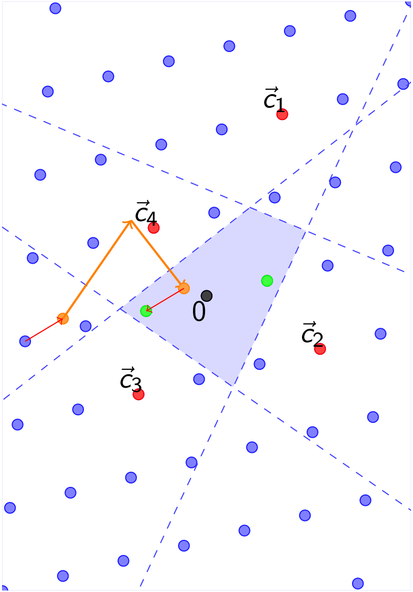
- Instead of sampling a lattice point $\vec{p}$

Bounding Collisions, Technique of AKS



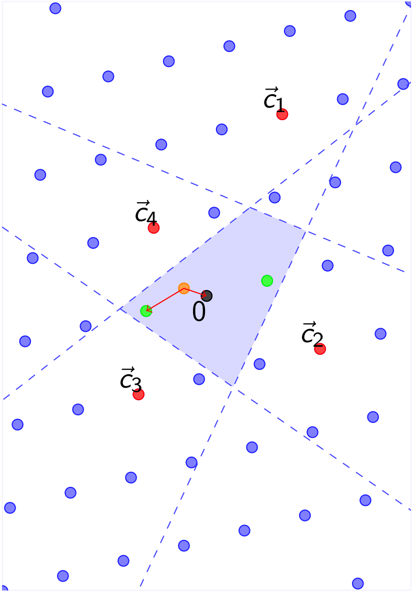
- Instead of sampling a lattice point $\vec{p}$
- Sample $(\vec{p}, \vec{\epsilon})$, so that $\vec{p} - \vec{\epsilon} \in \mathcal{L}$, with short $\|\vec{s}\| > \|\vec{\epsilon}\| > 0.5\|\vec{s}\|$.
- $\mathtt{Reduce}(\vec{p},\ C)$ and consider $\vec{p}' - \vec{\epsilon}$.

Bounding Collisions, Technique of AKS



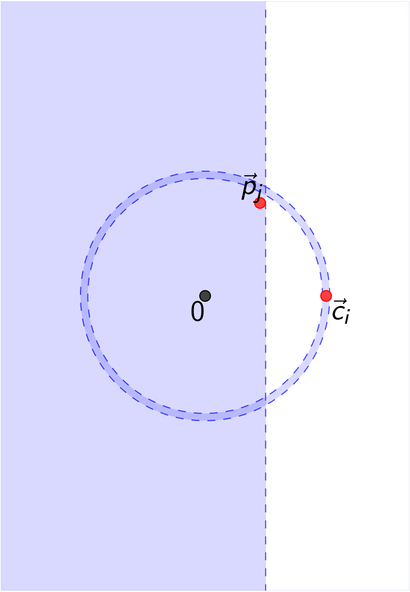
- Instead of sampling a lattice point $\vec{p}$
- Sample $(\vec{p}, \vec{\epsilon})$, so that $\vec{p} - \vec{\epsilon} \in \mathcal{L}$, with short $\|\vec{s}\| > \|\vec{\epsilon}\| > 0.5\|\vec{s}\|$.
- Reduce$(\vec{p},\ C)$ and consider $\vec{p}' - \vec{\epsilon}$.
- The trick is that $\vec{p}$ might correspond to two lattice points.
- Reduce is oblivious of $\vec{\epsilon}$,

Bounding Collisions, Technique of AKS



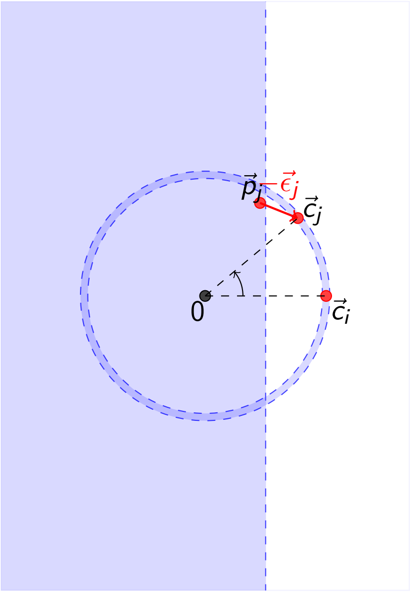
- Instead of sampling a lattice point $\vec{p}$
- Sample $(\vec{p}, \vec{\epsilon})$, so that $\vec{p} - \vec{\epsilon} \in \mathcal{L}$, with short $\|\vec{s}\| > \|\vec{\epsilon}\| > 0.5\|\vec{s}\|$.
- $\texttt{Reduce}(\vec{p},\ C)$ and consider $\vec{p}' - \vec{\epsilon}$.
- The trick is that $\vec{p}$ might correspond to two lattice points.
- $\texttt{Reduce}$ is oblivious of $\vec{\epsilon}$,
- so high probability of colissions $\Rightarrow$ high probability of finding $\vec{s}$.

Disadvantages of Perturbations



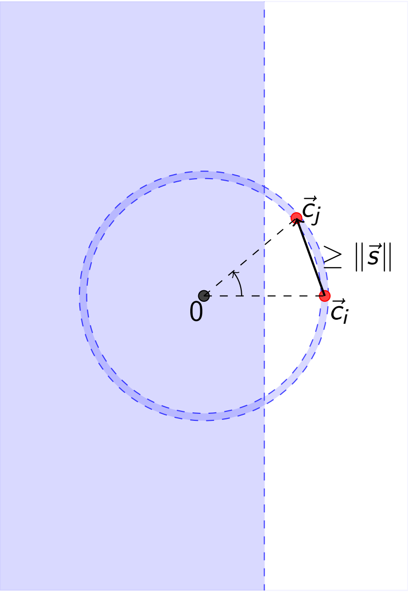
- Perturbations decrease the minimum angles.

Disadvantages of Perturbations



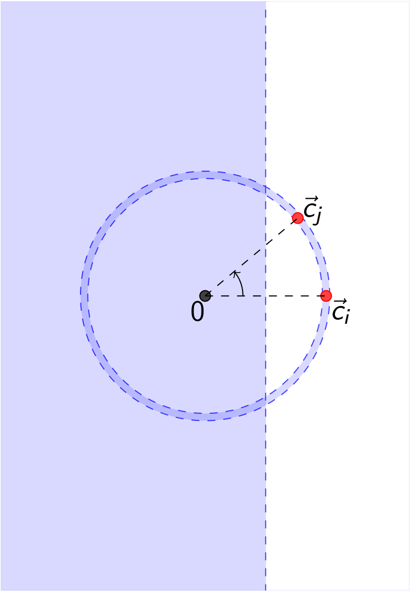
- Perturbations decrease the minimum angles.
- After `Reduce` we subtract $\vec{\epsilon}$, and the resulting point might be closer.

Disadvantages of Perturbations



- Perturbations decrease the minimum angles.
- After `Reduce` we subtract $\vec{\epsilon}$, and the resulting point might be closer.
- This affects the shells with small radius,
- so we use a different technique: $\|c_i - c_j\| \geq \|\vec{s}\|$.

Disadvantages of Perturbations



- Perturbations decrease the minimum angles.
- After `Reduce` we subtract $\vec{\epsilon}$, and the resulting point might be closer.
- This affects the shells with small radius,
- so we use a different technique: $\|c_i - c_j\| \geq \|\vec{s}\|$.
- 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]).
- List sieving.
- 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 $\phi_{\vec{c}_i, \vec{c}_j} \geq 60°$!

- Connection between kissing number and sieving.
- $\simeq 10^2$ to 10^3 faster, $\simeq 70\times$ less points.
- Proved space bounds of $2^{0.41n+o(n)}$, in practice $2^{0.21n+o(n)}$.
- Faster than NTL for dimensions > 40.
- Bottleneck is time, not space.

The code is at `cseweb.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.
  - 10^2 to 10^3 faster than previous implementation.

Open Problems

Open Problems:

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

Speciffic to our work:

- Bound time complexity without perturbations.
- Block reduction with higher block sizes?