Sieve algorithms for the Shortest Vector Problem

Xavier Pujol, Damien Stehlé

ENSL, LIP, CNRS, INRIA, Université de Lyon, UCBL

February 2\textsuperscript{nd}, 2010
Introduction

AKS

List-Sieve

Birthday paradox

Conclusion
Shortest Vector Problem

- Any lattice \( L \) contains non-zero vectors of minimal norm.
- Finding such vectors is NP-hard.
- Applications:
  - Integer Linear Programming (Lenstra 83).
  - Strong lattice reduction for cryptanalysis.
Any lattice $L$ contains non-zero vectors of minimal norm.
Finding such vectors is NP-hard.

Applications:
• Integer Linear Programming (Lenstra 83).
• Strong lattice reduction for cryptanalysis.
Shortest Vector Problem

- Any lattice $L$ contains non-zero vectors of minimal norm.
- Finding such vectors is NP-hard.
- Applications:
  - Integer Linear Programming (Lenstra 83).
  - Strong lattice reduction for cryptanalysis.
Shortest Vector Problem

- Any lattice $L$ contains non-zero vectors of minimal norm.
- Finding such vectors is NP-hard.
- Applications:
  - Integer Linear Programming (Lenstra 83).
  - Strong lattice reduction for cryptanalysis.
Solving SVP

Enumeration-based deterministic algorithms:

- With preprocessing: Kannan (1983). Cost: $2^{O(n \log n)}$.

Both algorithms use polynomial space.

Probabilistic sieve algorithms:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List-Sieve</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Birthday paradox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conclusion</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solving SVP

Enumeration-based deterministic algorithms:

- With preprocessing: Kannan (1983). Cost: $2^{O(n \log n)}$.

Both algorithms use polynomial space.

**Probabilistic sieve algorithms:**

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solving SVP

Enumeration-based deterministic algorithms:
- With preprocessing: Kannan (1983). Cost: $2^{O(n \log n)}$.

Both algorithms use polynomial space.

Probabilistic sieve algorithms:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKS</td>
<td>$2^{3.4n}$</td>
<td>$2^{2.5n}$</td>
</tr>
<tr>
<td>List-Sieve</td>
<td>$2^{4n}$</td>
<td>$2^{3n}$</td>
</tr>
<tr>
<td>List-Sieve with birthday paradox</td>
<td>$2^{4n}$</td>
<td>$2^{3.5n}$</td>
</tr>
</tbody>
</table>
Solving SVP

Enumeration-based deterministic algorithms:

- With preprocessing: Kannan (1983). Cost: $2^{O(n \log n)}$.

Both algorithms use polynomial space.

Probabilistic sieve algorithms:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKS</td>
<td>$2^{3.4n}$</td>
<td>$2^{2.8n}$</td>
</tr>
<tr>
<td>List-Sieve</td>
<td>$2^{2.1n}$</td>
<td>$2^{1.5n}$</td>
</tr>
<tr>
<td>List-Sieve with birthday paradox</td>
<td>$2^{n}$</td>
<td>$n$</td>
</tr>
</tbody>
</table>
Solving SVP

Enumeration-based deterministic algorithms:

- With preprocessing: Kannan (1983). Cost: $2^{O(n \log n)}$.

Both algorithms use polynomial space.

Probabilistic sieve algorithms:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKS</td>
<td>$2^{3.4n}$</td>
<td>$2^{2.0n}$</td>
</tr>
<tr>
<td>List-Sieve</td>
<td>$2^{3.2n}$</td>
<td>$2^{1.4n}$</td>
</tr>
<tr>
<td>List-Sieve with birthday paradox</td>
<td>$2^{2.5n}$</td>
<td>$2^{1.3n}$</td>
</tr>
</tbody>
</table>
Solving SVP

Enumeration-based deterministic algorithms:
- With preprocessing: Kannan (1983). Cost: $2^{O(n \log n)}$.
Both algorithms use polynomial space.

Probabilistic sieve algorithms:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKS</td>
<td>$2^{3.4n}$</td>
<td>$2^{2.0n}$</td>
</tr>
<tr>
<td>List-Sieve</td>
<td>$2^{3.2n}$</td>
<td>$2^{1.4n}$</td>
</tr>
<tr>
<td>List-Sieve with birthday paradox</td>
<td>$2^{2.5n}$</td>
<td>$2^{1.3n}$</td>
</tr>
</tbody>
</table>
Solving SVP

Enumeration-based deterministic algorithms:
- With preprocessing: Kannan (1983). Cost: $2^O(n \log n)$.
Both algorithms use polynomial space.

Probabilistic sieve algorithms:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKS</td>
<td>$2^{3.4n}$</td>
<td>$2^{2.0n}$</td>
</tr>
<tr>
<td>List-Sieve</td>
<td>$2^{3.2n}$</td>
<td>$2^{1.4n}$</td>
</tr>
<tr>
<td>List-Sieve with birthday paradox</td>
<td>$2^{2.5n}$</td>
<td>$2^{1.3n}$</td>
</tr>
</tbody>
</table>
Introduction

AKS

List-Sieve

Birthday paradox

Conclusion
History of AKS

- First version by Ajtai, Kumar and Sivakumar (2001).
- Refined analysis, implementation by Nguyen and Vidick (2008).
- Improved analysis with sphere-packing arguments by Micciancio and Voulgaris (2010).
History of AKS

- First version by Ajtai, Kumar and Sivakumar (2001).
  - Refined analysis, implementation by Nguyen and Vidick (2008).
  - Improved analysis with sphere-packing arguments by Micciancio and Voulgaris (2010).
History of AKS

• First version by Ajtai, Kumar and Sivakumar (2001).
• Simplified presentation by Regev (2004).
• Refined analysis, implementation by Nguyen and Vidick (2008).
• Improved analysis with sphere-packing arguments by Micciancio and Voulgaris (2010).
History of AKS

- First version by Ajtai, Kumar and Sivakumar (2001).
- Refined analysis, implementation by Nguyen and Vidick (2008).
- Improved analysis with sphere-packing arguments by Micciancio and Voulgaris (2010).
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- **Step 1**: sample random lattice vectors.
- **Step 2**: repeat the sieve until vectors are short enough.
- **Step 3**: return the closest pair of vectors.
AKS algorithm for SVP

• Step 1: sample random lattice vectors.
• Step 2: repeat the sieve until vectors are short enough.
• Step 3: return the closest pair of vectors.
AKS algorithm for SVP

• Step 1: sample random lattice vectors.
• Step 2: repeat the sieve until vectors are short enough.
• Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

• Step 1: sample random lattice vectors.
• Step 2: repeat the sieve until vectors are short enough.
• Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
AKS algorithm for SVP

- Step 1: sample random lattice vectors.
- Step 2: repeat the sieve until vectors are short enough.
- Step 3: return the closest pair of vectors.
Perturbations

Problem: the final set $S$ of vectors may be $\{0\}$.

- Solution: apply a small random perturbation to each sampled vector.
- Some information is hidden to the sieve algorithm: several lattice vectors might correspond to a given perturbed vector.
- $\Pr[\|u - v\| = \lambda(L)]$ for some $u, v \in S$.
- $> 2^{-O(n)}\Pr[u = v]$ for some $u, v \in S$. 
Perturbations

Problem: the final set $S$ of vectors may be $\{0\}$.

- Solution: apply a small random perturbation to each sampled vector.
- Some information is hidden to the sieve algorithm: several lattice vectors might correspond to a given perturbed vector.
- $\Pr[\|u - v\| = \lambda(L)]$ for some $u, v \in S$
- $> 2^{-O(n)}\Pr[u = v]$ for some $u, v \in S$. 

Perturbations

Problem: the final set $S$ of vectors may be $\{0\}$.

- **Solution**: apply a small random perturbation to each sampled vector.

- Some information is hidden to the sieve algorithm: several lattice vectors might correspond to a given perturbed vector.

- $\Pr[\|u - v\| = \lambda(L)]$ for some $u, v \in S$

- $> 2^{-O(n)} \Pr[u = v]$ for some $u, v \in S$. 
Perturbations

Problem: the final set $S$ of vectors may be $\{0\}$.

- Solution: apply a small random perturbation to each sampled vector.
- Some information is hidden to the sieve algorithm: several lattice vectors might correspond to a given perturbed vector.
- $\Pr[\|u - v\| = \lambda(L)]$ for some $u, v \in S$
  $> 2^{-O(n)}\Pr[u = v]$ for some $u, v \in S$. 

X. Pujol, D. Stehlé  
Sieve algorithms for the Shortest Vector Problem
Complexity of AKS

How many vectors are lost during the sieve?

\[
\text{fewer than } \frac{(R+R/4)^n}{(R/4)^n} = 2^{O(n)}
\]

at each step.

\rightarrow \text{ polynomial number of steps.}

\[2^{O(n)} \text{ vectors are enough.}\]

\text{Time complexity quadratic in space complexity.}

With a finer analysis: \(2^{3.4n}\)
Complexity of AKS

How many vectors are lost during the sieve?

→ fewer than \( \frac{(R+R/4)^n}{(R/4)^n} = 2^{O(n)} \)
at each step.

→ polynomial number of steps.

2\(^{O(n)}\) vectors are enough.

Time complexity quadratic in space complexity.

With a finer analysis: 2\(^{3.4n}\)
Complexity of AKS

How many vectors are lost during the sieve?

\[ \text{fewer than } \frac{(R+R/4)^n}{(R/4)^n} = 2^{O(n)} \text{ at each step.} \]

\[ \text{polynomial number of steps. } 2^{O(n)} \text{ vectors are enough.} \]

Time complexity quadratic in space complexity.
With a finer analysis: \( 2^{3.4n} \)
Complexity of AKS

How many vectors are lost during the sieve?

→ fewer than \( \frac{(R+R/4)^n}{(R/4)^n} = 2^{O(n)} \) at each step.

→ polynomial number of steps.

\( 2^{O(n)} \) vectors are enough.

Time complexity quadratic in space complexity.

With a finer analysis: \( 2^{3.4n} \)
How many vectors are lost during the sieve?

- fewer than \( \frac{(R+R/4)^n}{(R/4)^n} = 2^{O(n)} \) at each step.
- polynomial number of steps.
- \( 2^{O(n)} \) vectors are enough.

Time complexity quadratic in space complexity.

With a finer analysis: \( 2^{3.4n} \)
How many vectors are lost during the sieve?
→ fewer than \( \frac{(R+R/4)^n}{(R/4)^n} = 2^{O(n)} \) at each step.
→ polynomial number of steps. \( 2^{O(n)} \) vectors are enough.
Time complexity quadratic in space complexity.

With a finer analysis: \( 2^{3.4n} \)
How many vectors are lost during the sieve?
→ fewer than \( \frac{(R+R/4)^n}{(R/4)^n} = 2^{O(n)} \) at each step.
→ polynomial number of steps.
2\(^O(n)\) vectors are enough.
Time complexity quadratic in space complexity.
With a finer analysis: \( 2^{3.4n} \)
Introduction

AKS

List-Sieve

Birthday paradox

Conclusion
List-Sieve

- Algorithm introduced by Micciancio and Voulgaris (2010).
- Idea: create a set of short vectors by subtractions, as in AKS.
- Vectors are sampled one by one.
- All previous vectors are used to reduce a new vector.
List-Sieve

- Algorithm introduced by Micciancio and Voulgaris (2010).
- Idea: create a set of short vectors by subtractions, as in AKS.
  - Vectors are sampled one by one.
  - All previous vectors are used to reduce a new vector.
List-Sieve

- Algorithm introduced by Micciancio and Voulgaris (2010).
- Idea: create a set of short vectors by subtractions, as in AKS.
- Vectors are sampled one by one.
- All previous vectors are used to reduce a new vector.
List-Sieve

- Algorithm introduced by Micciancio and Voulgaris (2010).
- Idea: create a set of short vectors by subtractions, as in AKS.
- Vectors are sampled one by one.
- All previous vectors are used to reduce a new vector.
List-Sieve: example
List-Sieve: example
List-Sieve: example
List-Sieve: example
List-Sieve: example
List-Sieve: example
List-Sieve: example
List-Sieve: example
List-Sieve: example
Complexity of List-Sieve

- Lower bound for the angle between two vectors
  - Without perturbations: $2^{0.4n}$ vectors in the worst case.
  - With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
  - Without perturbations: $2^{0.4n}$ vectors in the worst case.
  - With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
  - Without perturbations: $2^{0.4n}$ vectors in the worst case.
  - With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
The complexity of List-Sieve involves:

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Complexity of List-Sieve

- Lower bound for the angle between two vectors
- Without perturbations: $2^{0.4n}$ vectors in the worst case.
- With perturbations: $2^{1.4n}$ vectors (more vectors around 0).
Introduction

AKS

List-Sieve

Birthday paradox

Conclusion
Birthday paradox

- Among 23 people, two of them have the same birthday with probability $> \frac{1}{2}$.
- If items are sampled from a set $S$ and i.i.d., a collision occurs with high probability after $O(\sqrt{|S|})$ steps.
- The uniform law is the worst case.
Birthday paradox

- Among 23 people, two of them have the same birthday with probability \( > \frac{1}{2} \).
- If items are sampled from a set \( S \) and i.i.d., a collision occurs with high probability after \( O(\sqrt{|S|}) \) steps.
- The uniform law is the worst case.
Birthday paradox

- Among 23 people, two of them have the same birthday with probability $> \frac{1}{2}$.
- If items are sampled from a set $S$ and i.i.d., a collision occurs with high probability after $O(\sqrt{|S|})$ steps.
- The uniform law is the worst case.
Applying the birthday paradox to AKS

- There must be enough vectors to ensure that the probability of collision is high at the end of the sieve.

- First solution: pigeonhole principle \( \rightarrow N = 2^{O(d)} \) vectors.

- All vectors in the final set are independent.

- Birthday paradox \( \rightarrow \sqrt{N} \) vectors suffice.

- Time complexity: \( 2^{2.7n} \) instead of \( 2^{3.4n} \).
Applying the birthday paradox to AKS

- There must be enough vectors to ensure that the probability of collision is high at the end of the sieve.

- First solution: pigeonhole principle \( \rightarrow N = 2^{O(d)} \) vectors.
  - All vectors in the final set are independent.
  - Birthday paradox \( \rightarrow \sqrt{N} \) vectors suffice.
  - Time complexity: \( 2^{2.7n} \) instead of \( 2^{3.4n} \).
Applying the birthday paradox to AKS

• There must be enough vectors to ensure that the probability of collision is high at the end of the sieve.

• First solution: pigeonhole principle $\rightarrow N = 2^{O(d)}$ vectors.
• All vectors in the final set are independent.
  • Birthday paradox $\rightarrow \sqrt{N}$ vectors suffice.
  • Time complexity: $2^{2.7n}$ instead of $2^{3.4n}$. 
Applying the birthday paradox to AKS

- There must be enough vectors to ensure that the probability of collision is high at the end of the sieve.

- First solution: pigeonhole principle $\rightarrow N = 2^{O(d)}$ vectors.
- All vectors in the final set are independent.
- Birthday paradox $\rightarrow \sqrt{N}$ vectors suffice.
- Time complexity: $2^{2.7n}$ instead of $2^{3.4n}$. 
Applying the birthday paradox to AKS

- There must be enough vectors to ensure that the probability of collision is high at the end of the sieve.

- First solution: pigeonhole principle $\rightarrow N = 2^{O(d)}$ vectors.
- All vectors in the final set are independent.
- Birthday paradox $\rightarrow \sqrt{N}$ vectors suffice.
- Time complexity: $2^{2.7n}$ instead of $2^{3.4n}$. 

\[0\]
Birthday paradox for List-Sieve

- Non-independent vectors.
- Solution:
  - Apply ListSieve, discarding all points that fall outside of the corona.
  - Sample small independent points by reducing random points w.r. to the first list.
Birthday paradox for List-Sieve

- Non-independent vectors.
- Solution:
  - Apply ListSieve, discarding all points that fall outside of the corona.
  - Sample small independent points by reducing random points w.r. to the first list.
Birthday paradox for List-Sieve

- Non-independent vectors.
- Solution:
  - Apply ListSieve, discarding all points that fall outside of the corona.
  - Sample small independent points by reducing random points w.r. to the first list.
Birthday paradox for List-Sieve

- Non-independent vectors.
- Solution:
  - Apply ListSieve, discarding all points that fall outside of the corona.
  - Sample small independent points by reducing random points w.r. to the first list.
Introduction

AKS

List-Sieve

Birthday paradox

Conclusion
The modifications of List-Sieve to apply the birthday paradox seem to be artefacts.

In practice, perturbations do not seem to be necessary either.

It is claimed in [MiVo10] that a heuristic version of List-Sieve outperforms enumeration-based algorithms.
• The modifications of List-Sieve to apply the birthday paradox seem to be artefacts.
• In practice, perturbations do not seem to be necessary either.
• It is claimed in [MiVo10] that a heuristic version of List-Sieve outperforms enumeration-based algorithms.
Conclusion

- The modifications of List-Sieve to apply the birthday paradox seem to be artefacts.
- In practice, perturbations do not seem to be necessary either.
- It is claimed in [MiVo10] that a heuristic version of List-Sieve outperforms enumeration-based algorithms.