Quantum speed-ups for sieving algorithms for the shortest vector problem

Elena Kirshanova

based on joint work with Erik Mårtensson, Eamonn W. Postlethwaite, Subhayan Roy Moulik

presented at AsiaCrypt’19

TQC 2020, Riga, Latvia
June 11, 2020
A lattice is a set \( \mathcal{L} = \{ \sum_{i=n}^{i=n} x_i b_i : x_i \in \mathbb{Z} \} \) for some linearly independent \( b_i \in \mathbb{R}^n \)

\( \{b_i\}_i \) — a basis of \( \mathcal{L} \)
Definitions

Minimum
\[ \lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L} \setminus 0} \| \mathbf{v} \| \]

Determinant
\[ \det(\mathcal{L}) = |\det(\mathbf{b}_i)_i| \]

Minkowski bound
\[ \lambda_1(\mathcal{L}) \leq \sqrt{n} \cdot \det(\mathcal{L})^{\frac{1}{n}} \]

A lattice is a set \( \mathcal{L} = \{ \sum_{i \leq n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \} \) for some linearly independent \( \mathbf{b}_i \in \mathbb{R}^n \)

\{\mathbf{b}_i\}_i \text{— a basis of } \mathcal{L}
The **Shortest Vector Problem (SVP)** asks to find \( \mathbf{v}_{\text{shortest}} \in \mathcal{L} \):

\[
\| \mathbf{v}_{\text{shortest}} \| = \lambda_1(\mathcal{L})
\]
The Shortest Vector Problem (SVP) asks to find $v_{\text{shortest}} \in \mathcal{L}$:

$$\|v_{\text{shortest}}\| = \lambda_1(\mathcal{L})$$

Often we are satisfied with an approximation ($\gamma$-SVP) to $v_{\text{shortest}}$:

$$\|v_{\text{short}}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$$
Why is SVP interesting?

\[ \|v_{\text{short}}\| \leq \gamma \cdot \lambda_1(\mathcal{L}) \]

Hardness of (approx)-SVP underlies all lattice-based cryptographic constructions.

- For \( \gamma = 2^{\log^{1-\epsilon} n} \) SVP is NP-hard
- Crypto is based on \( \gamma = \text{poly}(n) \)
- We assume SVP is infeasible for \( n > 350 \)
- What we can achieve now is \( n = 170 \) using lots of RAM and GPUs, see TU Darmstadt’s SVP challenge\(^1\)
- There is an open-source library G6K\(^2\) for solving SVP

---

\(^1\)https://www.latticechallenge.org/svp-challenge/
\(^2\)https://github.com/fplll/g6k
Asymptotics (+o() everywhere) of $\gamma$-SVP, $\gamma < \text{poly}(n)$, $n := \text{dim } \mathcal{L}$

<table>
<thead>
<tr>
<th>Classical</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Time  = $\frac{1}{2} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
<tr>
<td>Log Time  = $\frac{1}{4} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
<tr>
<td>Log Time  = $\frac{1}{2} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
<tr>
<td>Log Time  = $\frac{1}{4} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
<tr>
<td>Log Time  = $\frac{3}{0.5} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
<tr>
<td>Log Time  = $\frac{3}{0.5} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
<tr>
<td>Log Time  = $\frac{3}{0.5} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
<tr>
<td>Log Time  = $\frac{3}{0.5} e^n \log n$</td>
<td>Mem = $\text{poly}(n)$</td>
</tr>
</tbody>
</table>

Time/Memory trade-offs exist
Asymptotics (+$o()$ everywhere) of $\gamma$-SVP, $\gamma < \text{poly}(n)$, $n := \dim \mathcal{L}$

<table>
<thead>
<tr>
<th>Classical</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enumeration</strong></td>
<td><strong>Enumeration</strong></td>
</tr>
<tr>
<td>[ \log \text{Time} = \frac{1}{2e} n \log n ]</td>
<td>[ \log \text{Time} = \frac{1}{4e} n \log n ]</td>
</tr>
<tr>
<td>[ \text{Mem} = \text{poly}(n) ] [HS07]</td>
<td>[ \text{Mem} = \text{poly}(n) ] [ANS18]</td>
</tr>
</tbody>
</table>
Asymptotics (+o() everywhere) of $\gamma$-SVP, $\gamma < \text{poly}(n)$, $n := \dim \mathcal{L}$

**Classical**

- Log Time = $\frac{1}{2e} n \log n$
- Mem = poly$(n)$ [HS07]

**Quantum**

- Log Time = $\frac{1}{4e} n \log n$
- Mem = poly$(n)$ [ANS18]

**Enumeration**

Based on discrete Gaussian samplers

- Log Time = $1n$
- Log Mem = $1n$ [ADRS15]

- Log Time = $1.2553n$
- Log Mem = $0.5n$ [LLK18]
Asymptotics (+o() everywhere) of $\gamma$-SVP, $\gamma < \text{poly}(n)$, $n := \text{dim } \mathcal{L}$

**Classical**

- **Enumeration**
  - $\log \text{Time} = \frac{1}{2e} n \log n$
  - $\log \text{Mem} = \text{poly}(n)$ [HS07]

- **Based on discrete Gaussian samplers**
  - $\log \text{Time} = 1n$
  - $\log \text{Mem} = 1n$ [ADRS15]

- **Sieving (provable)**
  - $\log \text{Time} = 2.465n$
  - $\log \text{Mem} = 1.325n$ [PS09]

**Quantum**

- **Enumeration**
  - $\log \text{Time} = \frac{1}{4e} n \log n$
  - $\log \text{Mem} = \text{poly}(n)$ [ANS18]

- **Based on discrete Gaussian samplers**
  - $\log \text{Time} = 1.2553n$
  - $\log \text{Mem} = 0.5n$ [LLK18]

- **Sieving (provable)**
  - $\log \text{Time} = 1.799n$
  - $\log \text{Mem} = 1.286n$ [LMP15]
Asymptotics (+\(o()\) everywhere) of \(\gamma\)-SVP, \(\gamma < \text{poly}(n)\), \(n := \text{dim} \mathcal{L}\)

### Classical Enumeration

<table>
<thead>
<tr>
<th>Log Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log Time = \frac{1}{2e} n \log n)</td>
<td>(\text{Mem} = \text{poly}(n)) [HS07]</td>
</tr>
</tbody>
</table>

### Quantum Enumeration

<table>
<thead>
<tr>
<th>Log Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log Time = \frac{1}{4e} n \log n)</td>
<td>(\text{Mem} = \text{poly}(n)) [ANS18]</td>
</tr>
</tbody>
</table>

Based on discrete Gaussian samplers

<table>
<thead>
<tr>
<th>Log Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log Time = 1n)</td>
<td>(\log Mem = 1n) [ADRS15]</td>
</tr>
<tr>
<td>(\log Time = 1.2553n)</td>
<td>(\log Mem = 0.5n) [LLK18]</td>
</tr>
</tbody>
</table>

### Sieving (provable)

<table>
<thead>
<tr>
<th>Log Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log Time = 2.465n)</td>
<td>(\log Mem = 1.325n) [PS09]</td>
</tr>
<tr>
<td>(\log Time = 1.799n)</td>
<td>(\log Mem = 1.286n) [LMP15]</td>
</tr>
</tbody>
</table>

### Sieving (heuristic) +LSH

<table>
<thead>
<tr>
<th>Log Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log Time = 0.292n)</td>
<td>(\log Mem = 0.208n) [BDGL16]</td>
</tr>
<tr>
<td>(\log Time = 0.265n)</td>
<td>(\log Mem = 0.265n) [Laa16]</td>
</tr>
</tbody>
</table>

### Sieving (heuristic)

<table>
<thead>
<tr>
<th>Log Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log Time = 0.396n)</td>
<td>(\log Mem = 0.189n) [HK17]</td>
</tr>
<tr>
<td>(\log Time = 0.299n)</td>
<td>(\log Mem = 0.139n) [KMPR19]</td>
</tr>
</tbody>
</table>

Time/Memory trade-offs exist

For quantum algorithms Memory means quantumly addressable classical RAM.
Sieving for SVP
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: saturate space with enough lattice vectors so that their sums give short(er) vectors.
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: **saturate** space with enough lattice vectors so that their sums give short(er) vectors.
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: saturate space with enough lattice vectors so that their sums give short(er) vectors.
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: saturate space with enough lattice vectors so that their sums give short(er) vectors

\[ L = \pm L \]

\[ \| \pm x_1 + x_2 \| \]

is small
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: saturate space with enough lattice vectors so that their sums give short(er) vectors

\[ L = L' \]

\[ \|x_1 \pm x_2\| \text{ is small} \]

\[ \text{poly}(n) \]
How large $|L|$ should be?

Assumption: vectors (normalized) in $L$ are uniform iid on $S^{n-1}$. 
How large $|L|$ should be?

Assumption: vectors (normalized) in $|L|$ are uniform iid on $S^{n-1}$. 
How large $|L|$ should be?

Assumption: vectors (normalized) in $|L|$ are uniform iid on $S^{n-1}$.

$$\frac{S_{\text{surface}}}{S_{\text{sphere}}} = \sin \left( \frac{\pi}{3} \right)^n = \left( \sqrt{\frac{3}{4}} \right)^n$$
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: saturate space with enough lattice vectors so that their sums give short(er) vectors

\[ L = L' \]

\[ \|x_1 \pm x_2\| \text{ is small} \]

\[ L' = L' \]

\[ \text{Mem} = \left( \sqrt{\frac{3}{4}} \right)^{-n} = 2^{0.2075n} \]

\[ \text{Time}_{\text{Class}} = |L|^2 = 2^{0.415n} \]

All \( o(n) \) terms are omitted
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: **saturate** space with enough lattice vectors so that their sums give short(er) vectors

\[
\begin{align*}
L & = L \\
x_1 \pm x_2 & = L' \\
| |x_1 \pm x_2| | & \text{is small} \\
& = \text{poly}(n) \\
& = |L'|^2 = 2^{0.415n}
\end{align*}
\]

\[
\begin{align*}
\text{Mem} & = \left(\sqrt{\frac{3}{4}}\right)^{-n} = 2^{0.2075n} \\
\text{Time}^{\text{Class}} & = |L|^2 = 2^{0.415n} \\
\text{Grover over} L : \\
\text{Time}^{\text{Quant}} & = 2^{0.311n}
\end{align*}
\]

All \( o(n) \) terms are omitted
Basic 2-Sieve (Nguyen-Vidick sieve)

Main idea in all sieving algorithms: **saturate** space with enough lattice vectors so that their sums give short(er) vectors

\[
\begin{align*}
L & = \mathcal{L} \\
L & = \mathcal{L}' \\
\|x_1 \pm x_2\| & = \mathcal{L}'
\end{align*}
\]

- **Mem** = \( \left( \sqrt{\frac{3}{4}} \right)^{-n} = 2^{0.2075n} \)
- **Time\text{\textsuperscript{Class}}** = \(|L|^2 = 2^{0.415n} \)
- **Time\text{\textsuperscript{Quant}}** = \(2^{0.311n} \)

- we need to find almost all \((1 - o(1))\) fraction of close pairs
- ‘close’ pairs are much closer than the two random ones

All \(o(n)\) terms are omitted
Faster sieving with locality-sensitive hashing

How to improve classical runtime to $T = 2^{0.292n + o(n)}$?

Use Near Neighbour search!
Locality-sensitive filtering [BGJ15, BDGL16]
Locality-sensitive filtering [BGJ15, BDGL16]
Locality-sensitive filtering [BGJ15, BDGL16]

'centers' $u_i$ define buckets

$x_i \in L$
Locality-sensitive filtering [BGJ15, BDGL16]

For all $u_i$:
   For all $x \in L$
      If $|\langle x, u_i \rangle|$ is large enough
         put $x$ into $\text{Bucket}(u_i)$
Locality-sensitive filtering [BGJ15, BDGL16]

For all $u_i$:
For all $x \in L$
  If $|\langle x, u_i \rangle|$ is large enough
    put $x$ into $\text{Bucket}(u_i)$

For all $u_i$:
For all $(x, x') \in \text{Bucket}(u_i)$
  Check if $||x \pm x'||$ is short
For all $u_i$:
For all $x \in L$
If $|\langle x, u_i \rangle|$ is large enough
put $x$ into $\text{Bucket}(u_i)$

For all $u_i$:
For all $(x, x') \in \text{Bucket}(u_i)$
Check if $||x \pm x'||$ is short

For $2^{(0.142+o(1))n}$ many $u_i$'s:
$T = 2^{(0.349+o(1))n}$

When $u_i$'s are of special form
$T = 2^{(0.292+o(1))n}$
Locality-sensitive filtering [BGJ15, BDGL16]

For all \( u_i \):
  
  For all \( x \in L \):
    
    If \( |\langle x, u_i \rangle| \) is large enough
    
    put \( x \) into \( \text{Bucket}(u_i) \)

For all \( u_i \):
  
  For all \( (x, x') \in \text{Bucket}(u_i) \):
    
    Check if \( ||x \pm x'|| \) is short

For \( 2^{(0.142+o(1))n} \) many \( u_i \)’s:

\[
T = 2^{(0.349+o(1))n}
\]

When \( u_i \)’s are of special form

\[
T = 2^{(0.292+o(1))n}
\]

Grover over \( u_i \)’s gives

\[
T = M = 2^{(0.265+o(1))n}
\]

Lower memory?
Locality-sensitive filtering [BGJ15, BDGL16]

For all $\mathbf{u}_i$:
  For all $\mathbf{x} \in L$
    If $|\langle \mathbf{x}, \mathbf{u}_i \rangle|$ is large enough
      put $\mathbf{x}$ into $\text{Bucket}(\mathbf{u}_i)$

For all $\mathbf{u}_i$:
  For all $(\mathbf{x}, \mathbf{x}') \in \text{Bucket}(\mathbf{u}_i)$
    Check if $||\mathbf{x} \pm \mathbf{x}'||$ is short

For $2^{(0.142 + o(1))n}$ many $\mathbf{u}_i$'s:
  $T = 2^{(0.349 + o(1))n}$

When $\mathbf{u}_i$'s are of special form
  $T = 2^{(0.292 + o(1))n}$

Grover over $\mathbf{u}_i$'s gives
  $T = M = 2^{(0.265 + o(1))n}$

Lower memory?
3-Sieve as 3-List problem

Search for triples instead of pairs

Ex.: $\|12^3 + 6 + 18\| \leq 1$

This reduces required memory from $2^{0.2075n}$ to $2^{0.1887n}$
Configuration of good triples, [HK17]

All good triples are concentrated in the shape of 3-simplex
3-Sieve via triangle finding
3-Sieve via triangle finding

**Connect two points** \((i, j) \iff |\langle i, j \rangle| \approx 1/3\)
3-Sieve via triangle finding

Connect two points \((i, j) \iff |\langle i, j \rangle| \approx 1/3\)
3-Sieve via triangle finding

Connect two points \((i, j)\) ⇔ \(|\langle i, j \rangle| \approx 1/3\)

Good triples \((i, j, k)\) ⇔ triangles
Apply quantum triangle \((k\text{-clique})\) finding

\[ G = \{V, E\}, \ V - \text{lattice vectors}, \ e(v_i, v_j) \in E \iff |\langle v_i, v_j \rangle| \approx 1/3 \]

Run triangle listing on \(G\) (it’s a sparse graph!)
Apply quantum triangle ($k$-clique) finding

$G = \{V, E\}, \ V - \text{lattice vectors}, \ e(v_i, v_j) \in E \Leftrightarrow |\langle v_i, v_j \rangle| \approx 1/3$

Run triangle listing on $G$ (it’s a sparse graph!)

Vast literature on quantum triangle finding but in the query model
Apply quantum triangle \((k\text{-clique})\) finding

\[ G = \{V, E\}, \ V - \text{lattice vectors}, \ e(v_i, v_j) \in E \iff |\langle v_i, v_j \rangle| \approx 1/3 \]

Run triangle listing on \(G\) (it’s a sparse graph!)

Vast literature on quantum triangle finding but in the \textit{query} model

Adapt the triangle \textit{finding} algorithm of [Buhrman–de Wolf-Dürr–Heiligman–Høyer–Magniez–Santha]:

\[
\text{Time (find } \triangle \text{)} = \sqrt{|E|} \implies \text{Time (list all } \triangle' \text{'}s) = |V| \sqrt{|E|}
\]

This gives

\[
\text{Time}^{\text{Quant}} = 2^{0.335n} \quad \text{cf.} \quad \text{Time}^{\text{Class}} = 2^{0.396n}
\]
Apply quantum triangle ($k$-clique) finding

\[ G = \{V, E\}, \ V - \text{lattice vectors}, \ e(v_i, v_j) \in E \Leftrightarrow |\langle v_i, v_j \rangle| \approx 1/3 \]

Run triangle listing on $G$ (it’s a sparse graph!)

Vast literature on quantum triangle finding but in the query model

Adapt the triangle finding algorithm of [Buhrman–de Wolf–Dürr–Heiligman–Høyer–Magniez–Santha]:

\[
\text{Time (find } \Delta) = \sqrt{|E|} \implies \text{Time (list all } \Delta' \text{'s)} = |V| \sqrt{|E|}
\]

This gives

\[
\text{Time}^{\text{Quant}} = 2^{0.335n} \quad \text{cf.} \quad \text{Time}^{\text{Class}} = 2^{0.396n}
\]

The algorithm generalises to larger $k = \Theta(1)$ and time-optimal inner product leading to

\[
\text{Time}^{\text{Quant}} = 2^{0.299n+o(n)} \quad \text{Memory}^* = 2^{0.139n+o(n)}
\]

* quantumly addressable classical memory
More results and conclusions

- Overall now we have
  - best Time $\times$ Area
    \[ \text{Time}^{\text{Quant}} = 2^{0.299n + o(n)} \]
    \[ \text{Memory} = 2^{0.139n + o(n)} \]
    cf.
    \[ \text{Time}^{\text{Class}} = 2^{0.373n + o(n)} \]
    \[ \text{Memory} = 2^{0.186n + o(n)} \]
  - best Time achieved with $k = 2$
    \[ \text{Time}^{\text{Quant}} = 2^{0.265n + o(n)} \]
    \[ \text{Memory} = 2^{0.265n + o(n)} \]

Thank you!
More results and conclusions

- Overall now we have
  - best Time $\times$ Area
    - $\text{Time}^{\text{Quant}} = 2^{0.299n+o(n)}$
    - $\text{Memory} = 2^{0.139n+o(n)}$
  - best Time achieved with $k = 2$
    - $\text{Time}^{\text{Quant}} = 2^{0.265n+o(n)}$
    - $\text{Memory} = 2^{0.265n+o(n)}$

- Quantum memory?
  - There exists a quantum circuit that implements 2-Sieve of width $2^{0.2075n+o(n)}$ and depth $2^{0.1037n+o(n)}$.
More results and conclusions

- Overall now we have
  - best Time × Area
    \[ \text{Time}^{\text{Quant}} = 2^{0.299n+o(n)} \quad \text{Memory} = 2^{0.139n+o(n)} \]
    cf.
    \[ \text{Time}^{\text{Class}} = 2^{0.373n+o(n)} \quad \text{Memory} = 2^{0.186n+o(n)} \]
  - best Time achieved with \( k = 2 \)
    \[ \text{Time}^{\text{Quant}} = 2^{0.265n+o(n)} \quad \text{Memory} = 2^{0.265n+o(n)} \]

- Quantum memory?
  There exists a quantum circuit that implements 2-Sieve of width \( 2^{0.2075n+o(n)} \) and depth \( 2^{0.1037n+o(n)} \).

Thank you!


• [CCL17] Y. Chen, K. Chung, C. Lai. Space-efficient classical and quantum algorithms for the shortest vector problem


• [HS07] G. Hanrot, D. Stehlé. Improved Analysis of Kannan’s Shortest Lattice Vector Algorithm


• [Laa15] T. Laarhoven. Search problems in cryptography