Quantum NV Sieve on Grover for Solving Shortest Vector Problem

https://youtu.be/-VCd44r-tTk

한성대학교
HANSUNG UNIVERSITY

CryptoCraft LAB

Contribution

- Quantum NV Sieve 구현 후, Grover’s search를 적용
- Quantum NV Sieve의 자원 추정 (오라클, 그루버)
- 고차원에서 동작 가능한 Quantum NV Sieve를 위한 최적 구현
- 격자 기반 암호에 대한 암호 분석의 연구 범위 확장

배경지식

- KISTI 때 했던 것들이라 넘어가도록 하겠습니다.
  - Lattice, Basis, Approximate (LLL), Exact (Sieve) 알고리즘
  - Approximate 알고리즘 (고차원 격자) → Exact 알고리즘 (저차원 격자) ; 하이브리드

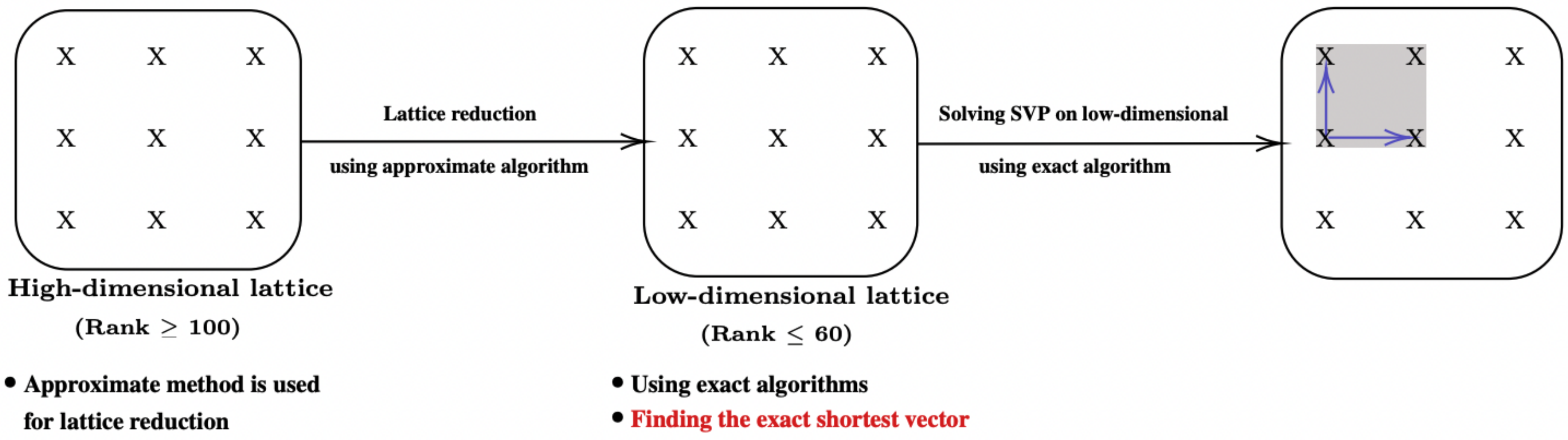

Fig. 2: Flow chart of approximate and exact algorithms for solving SVP.

NV Sieve

- NV Sieve의 목적
  - 짧은 벡터 손실 최소화 하며 범위 줄여나가면서 짧은 벡터 찾기
  - Sieve 알고리즘 세부 동작도 넘어가도록 하겠습니다. ($v - c$가 결국 구현해야하는 로직의 핵심)

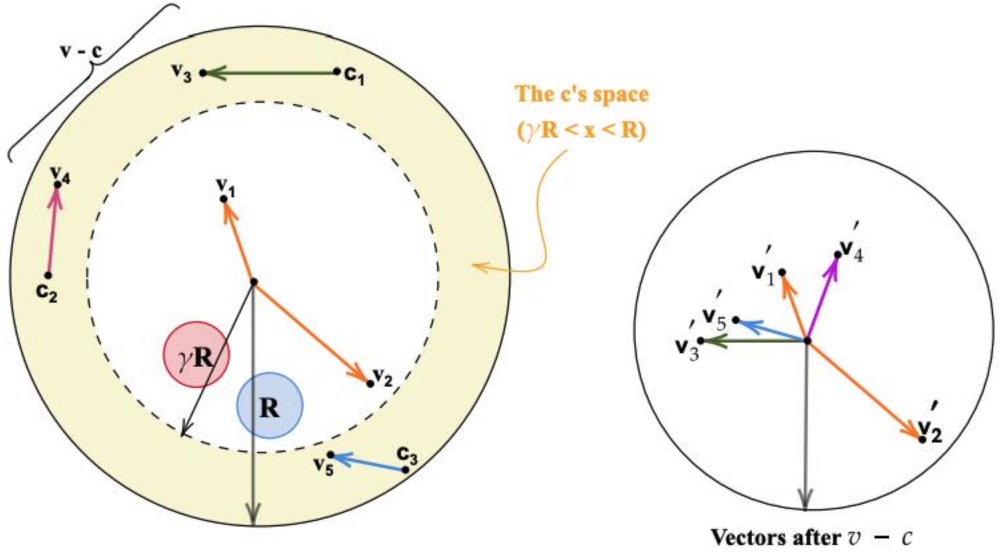

Fig. 5: The core logic in NV Sieve ($\exists c \in C||v-c|| \leq \gamma R$).

Quantum NV Sieve 전체 회로 (핵심 로직 위주)

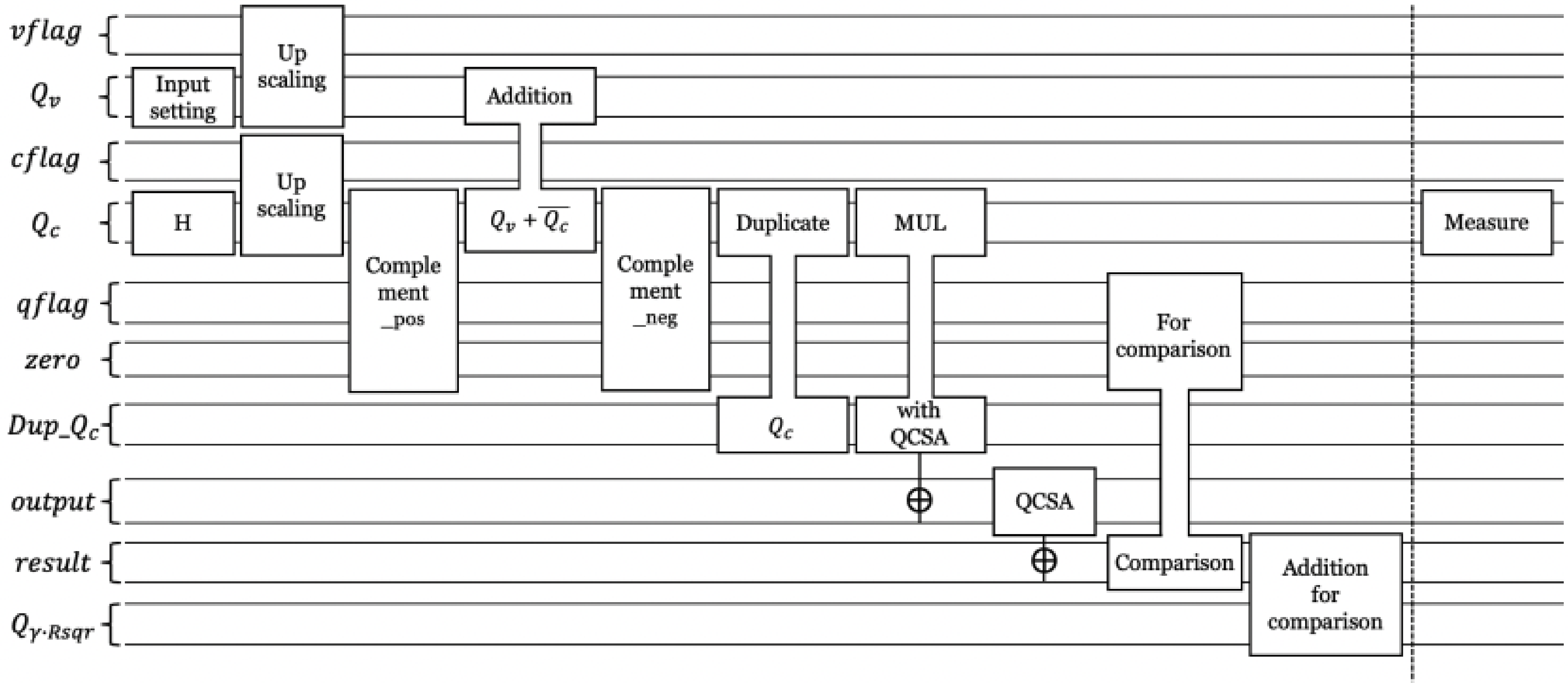

Fig. 6: Overall Quantum Circuit for Quantum NV Sieve.

Quantum NV Sieve 전체 로직

Algorithm 3: The quantum NV Sieve on the quantum circuit.

Input: Reduced lattice vector($\{v_0, ..., v_n\}$), dimension and rank of the lattice, A subset S in L and sieve factor γ ($\frac{2}{3} < \gamma < 1$)
Output: $\{c_0, ..., c_n\}$

Initiate quantum and classical registers ($carry, qflag, cflag, \gamma \cdot R_{sqr}$, $zero$, c_n.)
Let Q_v and Q_c be the qubits for lattice vectors v and c ▷ Q_c is the search target
Let $Q_{\gamma \cdot R_{sqr}}$ be the qubits for square of $\gamma \cdot R$

// **STEP 0: Extend** dim **and** $rank$ **for addressing the overflow**
$dim \leftarrow$ dimension+2, $rank \leftarrow$ rank
$sqr\_bitsize \leftarrow 2 \cdot dim$

// **STEP 1: Input setting** ($Q_v, Q_c, Q_{\gamma \cdot R_{sqr}}$)
$Q_{v_0}, \cdots, Q_{v_n} \leftarrow v_0, \cdots, v_n$ ▷ Using X gate
All(H)$|Q_c$
$Q_{\gamma \cdot R_{sqr}} \leftarrow \gamma \cdot R_{sqr}$ ▷ $0 \leq i < sqr\_bitsize$

// **STEP 2: Upscaling to address overflow**
for i in $rank$ **do**
    UPSCALING($Q_v[i], vflag[i]$)
end for

// **STEP 3: Two's complement for subtraction using adder**
for i in $rank$ **do**
    COMPLEMENT_pos($Q_c[i]$, $qflag[i], zero$) ▷ Outputs are $\overline{Q_c}$
end for

// **STEP 4:** $Q_v + \overline{Q_c}$ $(= Q_v - Q_c)$
for i in $rank$ **do**
    TAKAHASHI_ADDER($Q_v[i], Q_c[i]$) ▷ Store to $Q_c[i]$, [21]
end for

// **STEP 5: Two's complement for correct squaring**
for i in $rank$ **do**
    COMPLEMENT_neg($Q_c[i], qflag[rank + i], zero$)
end for

// **STEP 6: Duplicating qubit for squaring**
$Dup\_Q_c \leftarrow Q_c$

// **STEP 7: Squaring elements of vectors** (Q_c **and** Dup_Q_c)
for i in $rank$ **do**
    $output[i]$ = MUL($Q_c[i], Dup\_Q_c[i]$)
end for

// **STEP 8: Addition for squared results to obtain the size of the vector**
$result \leftarrow$ QCSA($output$)

// **STEP 9: Two's complement with** $sqr\_bitsize$
COMPLEMENT_compare($result[0 : sqr\_bitsize]$, $qflag[2 \cdot (rank - 2)]$, $zero$)

// **STEP 10: Size comparison between** $Q_{\gamma \cdot R_{sqr}}$ **and** $(||Q_v - Q_c||)^2$
TAKAHASHI_ADDER($Q_{\gamma \cdot R_{sqr}}$, $result[0 : sqr\_bitsize]$)

// **STEP 11: Measurement**
All(Measure)$|Q_c$

return $\{c_0, ..., c_n\}$

Up-scaling

- 오버플로우 방지를 위한 과정
- 상위 2비트를 추가하고, 데이터의 최상위 비트의 값을 최상위 2비트에 복사함으로써 업스케일링 (동일 값)

Algorithm 4: Quantum implementation for UPSCALING function

Input: ($Q_v[i]$, $vflag$) or ($Q_c[i]$, $cflag$), and dim
Output: $Q_v[i]$ or $Q_c[i]$

1: // Copy the MSB to the upper 2 qubits using CNOT gate
2: CNOT$|(Q_v[i][dim-3], vflag[i])$
3: CNOT$|(vflag[i], Q_v[i][dim-2])$
4: CNOT$|(vflag[i], Q_v[i][dim-1])$

5: CNOT$|(Q_c[i][dim-3], cflag[i])$
6: CNOT$|(cflag[i], Q_c[i][dim-2])$
7: CNOT$|(cflag[i], Q_c[i][dim-1])$

8: **return** $Q_v[i]$ or $Q_c[i]$

Complement

Algorithm 5: Quantum implementation for COMPLEMENT_pos function

Input: $Q_c[i]$, $qflag[i]$, $zero$
Output: Q_c or $\overline{Q_c}$

// Copy the MSB of Q_c to $qflag$ to check the sign bit
CNOT$(Q_c[dim-1], qflag)$

// Invert $qflag$ to take complement only when positive
X$|qflag$

// Invert Q_c
for i in dim **do**
    CNOT$|(qflag, Q_c[i])$
end for

// Create a new array of qubits and append $qflag$ to LSB
$NEW\_Q_c = []$
NEW_Q_c.append$(qflag)$

// Append 0 so that it has the same length as Q_c
for i in $dim-1$ **do**
    NEW_Q_c.append$(zero[i])$
end for

// Addition for $LSB+1$
TAKAHASHI_ADDER(NEW_Q_c, Q_c)

return Q_c or $\overline{Q_c}$

Positive / Negative

Algorithm 6: Quantum implementation for COMPLEMENT_neg function

Input: $Q_c[i]$, $qflag[i]$, $zero$
Output: Q_c or $\overline{Q_c}$

// Copy the MSB of Q_c to $qflag$ to check the sign bit
CNOT$(Q_c[dim-1], qflag)$

// Invert Q_c (no need X gate for $qflag$)
for i in dim **do**
    CNOT$|(qflag, Q_c[i])$
end for

// Create new array of qubits and append $qflag$ to LSB
$NEW\_Q_c = []$
NEW_Q_c.append$(qflag)$

// Append 0 so that it has the same length as Q_c
for i in $dim-1$ **do**
    NEW_Q_c.append$(zero[i])$
end for

// Addition for $LSB+1$
TAKAHASHI_ADDER(NEW_Q_c, Q_c)

return Q_c or $\overline{Q_c}$

Multiplication (Squaring)

- 현준님의 QCSA를 일부 사용
  - 전체 Depth를 매우 줄였음
  - 제 케이스 (높은 차원의 벡터)에서 약간의 오류가 발견되어 코드 수정
  - 해당 코드에서 맨 마지막에 큐비트 재사용을 위해 reverse 하는 부분은 사용하지 않음으로써 자원 절약 (본 작업에는 필요하지 않았음)
  - Takahashi 사용하여 ancilla 큐비트 사용 x → 덴스 최적화 위해서는 추후 다른 가산기 사용할 예정

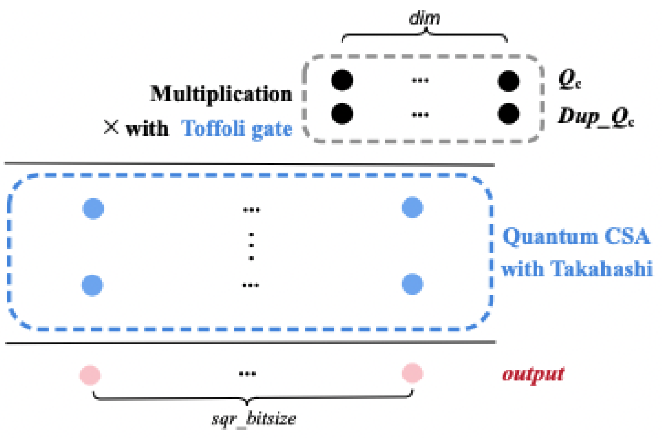

Fig. 7: Squaring using QCSA with Takahashi adder.

Algorithm 7: Quantum implementation for MUL function

Input: $Q_c[i]$, $Dup\_Q_c[i]$
Output: $output[0 : sqr\_bitsize]$

```
// Setting for multiplication
Let a be Q_c[i].
Letb be Dup_Q_c[i].
input = [[0 for i in do(sqr_bitsize)] for j in do (dim)]

// Multiply all elements of Q_c[i] and Dup_Q_c[i] using Toffoli gate
for i in dim do
    for j in dim do
        Toffoli|(a[i], b[j], input[i][i + j])
    end for
end for

// Addition of the results for each element at once
output = QCSA(input)

return output[0 : sqr_bitsize]
```

오라클 정확성 확인

- 테스트 벡터 넣어서 실제 계산과 맞는지 확인
- 연산 결과 정상 + 음수일 때 MSB=1, 양수일 때 0 나오면 정상

Table 1: Results from each step of quantum NV Sieve to check whether it has been implemented correctly (Rank and Dimension are 5; $rank = 5$, $dim = 7$, and $sqr\_bitsize = 14$).

STEP (Alg. 3)	Quantum variable	Values
STEP 1	Q_v	$\{1,3,1,1,5\}$
	Q_c	$\{8,1,4,4,6\}$
	$Q_{\gamma \cdot R_{sqr}}$	32
STEP 2	Q_v	$\{1,3,1,1,5\}$
	Q_c	$\{8,1,4,4,6\}$
	$Q_{\gamma \cdot R_{sqr}}$	32
STEP 3	$\overline{Q_c}$ (when positive)	$\{-8,-1,-4,-4,-6\}$
STEP 4	$Q_v + \overline{Q_c}$	$\{-7,2,-3,-3,-1\}$
STEP 5	$\overline{Q_v + \overline{Q_c}}$ (when negative)	$\{7,2,3,3,1\}$
STEP 6	Dup_Q_c	$\{7,2,3,3,1\}$
STEP 7	$\overline{Q_c} \cdot Dup\_Q_c$(Squaring for each element)	$\{49,4,9,9,1\}$
STEP 8	Sum_{Q_c}	72
STEP 9	$\overline{Sum_{Q_c}}$ (when positive)	-72
STEP 10	$(\gamma \cdot R)^2 + \overline{Sum_{Q_c}}$	-40
STEP 11	**Output**	$11111111011000_{(2)}$ (-40)
	MSB	1 (not short vector)

오라클 및 그루버 서치에 대한 비용 추정

Table 2: Resource Estimation of quantum NV Sieve oracle (R10D10 means the rank and the dimension of the lattice are 10).

Case	#CNOT	#1qCliff	#T	T-depth (Td)	Full depth (FD)	Qubit (M)	Td-M	FD-M
R10D10	$2^{16.1767}$	$2^{13.9067}$	$2^{15.7118}$	$2^{7.6037}$	$2^{11.5264}$	$2^{12.5454}$	$2^{20.1491}$	$2^{24.0718}$
R20D20	$2^{18.9097}$	$2^{16.6143}$	$2^{18.4212}$	$2^{8.4470}$	$2^{14.2190}$	$2^{15.2640}$	$2^{23.7110}$	$2^{29.4830}$
R30D30	$2^{20.5672}$	$2^{18.2624}$	$2^{20.0695}$	$2^{8.9454}$	$2^{15.2810}$	$2^{16.9202}$	$2^{25.8656}$	$2^{32.2012}$
R40D40	$2^{21.7859}$	$2^{19.4808}$	$2^{21.2880}$	$2^{9.3531}$	$2^{16.0566}$	$2^{18.1329}$	$2^{27.4860}$	$2^{34.1896}$
R50D50	$2^{22.6998}$	$2^{20.3885}$	$2^{22.1958}$	$2^{9.6165}$	$2^{16.6674}$	$2^{19.0527}$	$2^{28.6692}$	$2^{35.7201}$
R60D60	$2^{23.4836}$	$2^{21.1729}$	$2^{22.9802}$	$2^{9.8595}$	$2^{17.1715}$	$2^{19.8329}$	$2^{29.6924}$	$2^{37.0044}$
R70D70	$2^{24.1348}$	$2^{21.8228}$	$2^{23.6301}$	$2^{10.0660}$	$2^{17.6005}$	$2^{20.4848}$	$2^{30.5508}$	$2^{38.0853}$

Table 3: Quantum cost for Grover's search on quantum NV Sieve.

Case	#Total gates	T-depth (Td)	Full depth (FD)	Qubit (M)	Quantum cost	Td-M	FD-M
R10D10	$2^{18.1267}$	$2^{8.6073}$	$2^{12.5264}$	$2^{12.5456}$	$2^{30.6532} \cdot r$	$2^{21.1529}$	$2^{25.0720}$
R20D20	$2^{20.8481}$	$2^{9.4470}$	$2^{15.2190}$	$2^{15.2640}$	$2^{35.0664} \cdot r$	$2^{24.7110}$	$2^{32.4830}$
R30D30	$2^{22.5012}$	$2^{9.9454}$	$2^{16.2810}$	$2^{16.9202}$	$2^{37.7823} \cdot r$	$2^{26.8656}$	$2^{33.2012}$
R40D40	$2^{23.7199}$	$2^{10.3531}$	$2^{17.0566}$	$2^{18.1329}$	$2^{39.7765} \cdot r$	$2^{28.4860}$	$2^{35.1895}$
R50D50	$2^{24.6308}$	$2^{10.6165}$	$2^{17.6674}$	$2^{19.0527}$	$2^{41.2938} \cdot r$	$2^{29.6692}$	$2^{36.7201}$
R60D60	$2^{25.4150}$	$2^{10.8595}$	$2^{18.1715}$	$2^{19.8329}$	$2^{42.5865} \cdot r$	$2^{30.6924}$	$2^{38.0044}$
R70D70	$2^{26.0655}$	$2^{11.0660}$	$2^{18.6005}$	$2^{20.4848}$	$2^{43.6661} \cdot r$	$2^{31.5509}$	$2^{39.0853}$

양자 비용의 예상치.. 흐름..

- Dim 10 증가 시마다 로그함수처럼 증가
- Iteration=1 일 때의 값이긴 한데, KISTI때 봤던 결과를 적용하면 iteration이 그렇게 높지는 않을 것 (적어도 $\frac{\pi}{4}\sqrt{N/_M}$ 보단 작음)

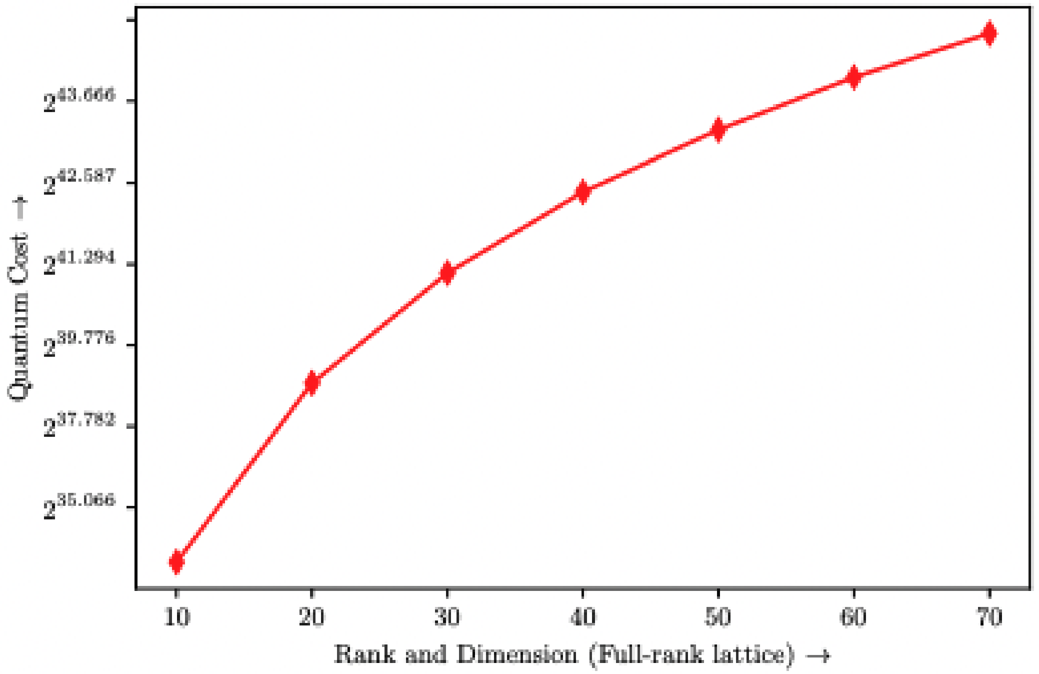

Fig. 8: Quantum cost for quantum NV Sieve on Grover's search ($r = 1$)

Further Discussions

- 다른 양자 공격의 양자 비용과의 비교
  - 70차원의 격자에 대해서도 64-bit 대칭키에 대한 그루버 서치 비용의 제곱근 정도

- 추가적인 최적화 포인트
  - QCSA에 사용한 adder가 큐비트를 줄이는 장점이 있음
  - 차라리 depth 최적화에 집중하도록 다른 adder를 사용하는 게 나을 듯

- NV Sieve의 복잡도와 관련된 사항
  - 다수의 솔루션 : 솔루션이 여러 개이므로 적합한 iteration을 찾는 것 중요, 그러나 앞서 언급했듯이 그렇게 크진 않을 듯
  - 검색 공간 (N) : 랭크와 디멘션에 의해 결정 ($N = 2^{rank \cdot dim}$) → 랭크와 디멘션이 커질 수록 해결 복잡도 높아짐
  - 입력 벡터 중 가장 긴 벡터인 R과 입력 벡터 v : 결과적으로 복잡도에 직접적 영향을 주는 건 R임
→ 범위를 줄이는 sieve factor γ도 R과 곱해지는 것이기 때문
→ 즉, 애초에 랜덤 샘플링에서 뽑히는 벡터 집합 v에서의 R이 영향을 줌

- Quantum Query Complexity
  - 오라클 적합성은 확인했으므로, 결과적으로 차원이 높아질수록 $O\sqrt{N}$을 따를 것 (KISTI 때 언급)

Conclusion

- 높은 차원에 대한 정확한 quantum NV Sieve 구현
  - 10~70차원까지 모두 구현하였으며, 정상동작 확인

- QCSA with Takahashi
  - Depth 감소 + 큐비트 절약
    → 해당 방식이 이전 구현보다 더 높은 차원의 격자에서도 더 효율적인 양자 회로를 구현할 수 있음을 의미
  - 이전 구현에 비해:
    - 랭크 및 디멘션이 70차원인 격자에 대한 T-depth : $2^{10.066}$
    - 랭크가 2, 디멘션이 4차원인 (이전 구현)의 T-depth: $2^{10.778}$

- NIST post-quantum security standard
  - Level 1 : 2^{157} : AES 128에 대한 Grover's search 복잡도
  - 랭크 및 디멘션이 70인 경우에도, 이보다 굉장히 낮은 비용

R70D70	$2^{26.0655}$	$2^{11.0660}$	$2^{18.6005}$	$2^{20.4848}$	$2^{43.6661}$ · r	$2^{31.5509}$	$2^{39.0853}$

  - 물론, 이는 전체 격자에 대한 복잡도는 아님 (암호에서는 보통 500 이상)
    →500차원에서 70까지 줄이는 알고리즘인 LLL에 대한 자원 추정도 필요
    → Quantum LLL도 이론?수학적으로는 나와있으나 구현을 한 것 같지는 않아서 구현할 예정)

세미나 늦지 않겠습니다.