KISTI 과제 관련
: Lattice, NV Sieve, Grover

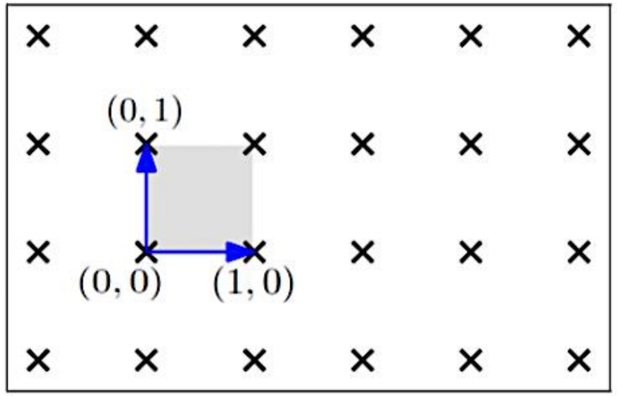
https://youtu.be/r97_lwUGXIc

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

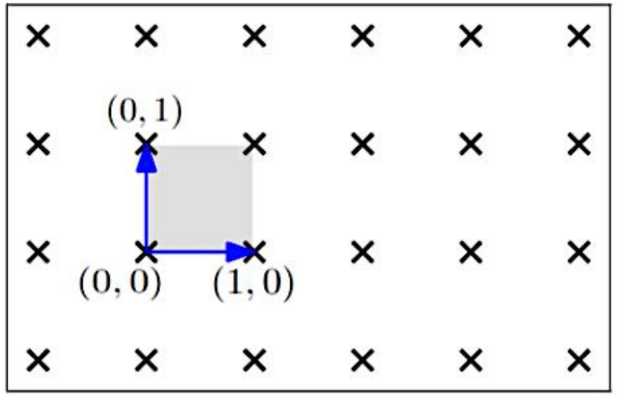
Lattice

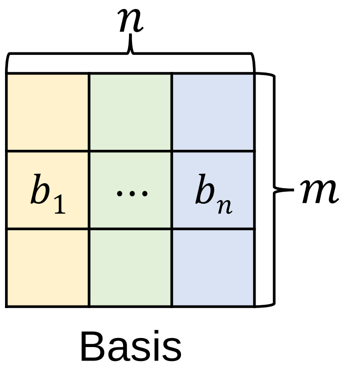
- $\boldsymbol{L(b_1, \cdots, b_n) = \{\sum_{i=1}^{n} x_i \cdot b_i, x_i \in \mathbb{Z}\}}$
  - x : 정수, $(b_1, \cdots, b_n)$: 기저 벡터 (Basis)
  - **Basis의 선형 결합**으로 이루어지는 격자 L의 포인트들
  - 격자는 점으로 이루어지므로 **가장 짧은 벡터 존재** (1개 이상)



Lattice basis

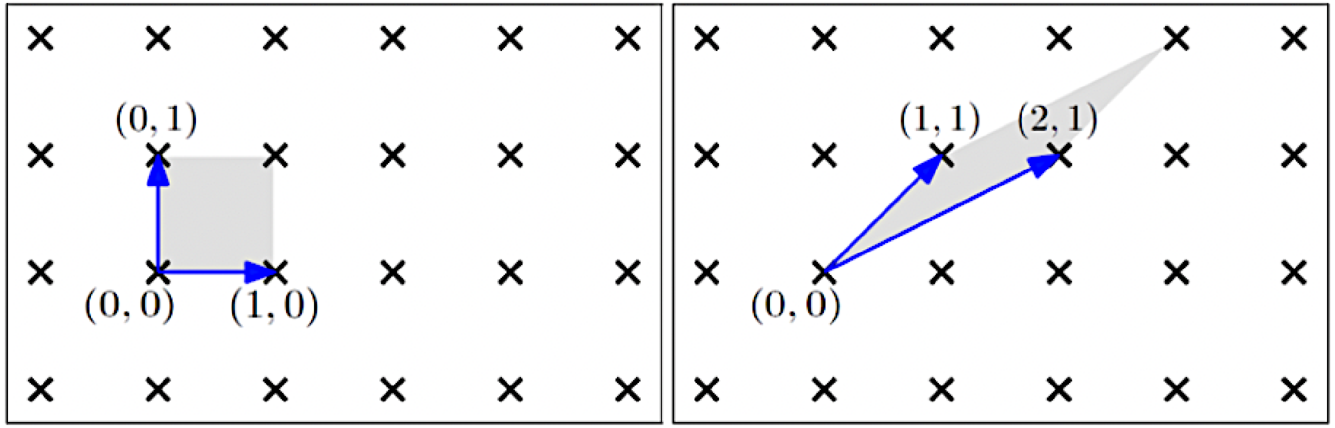
- **Basis (B)**
  - **격자의 모든 점을 만들 수 있는 벡터의 집합**
  - Basis를 구성하는 각 벡터 b_i는 길이가 m이며, 총 n개로 구성
  - **Rank (n)** : basis를 구성하는 벡터의 수
  - **Dimension (m)** : b_i의 길이
  - 일반적으로 **full-rank lattice** 사용 ($n = m$)





Lattice

- **Lattice의 basis는 유일한가?**
  - **유일하지 않다.**
  - 같은 격자지만 기저가 다름 (아래 그림의 파란색 벡터)
  - B_0 basis에 다른 벡터를 곱하여 B_1 basis를 만들 수 있다면, **두 basis는 같은 격자를 생성**



Lattice

- **두 basis가 같은 격자를 생성하는지 알 수 있는 방법**
  - 두 basis가 같은 격자를 생성한다면, 아래의 수식을 만족함을 의미
  - $\boldsymbol{B_0} = \boldsymbol{B_1} \Leftrightarrow B_0 = B_1 \cdot U, B_1 = B_0 \cdot V$

    $B_0 = B_0 \cdot V \cdot U$; 여기서, $V \cdot U$가 단위 행렬 (I)이 나와야 함

    → 따라서,V와 U는 **역행렬 관계**이므로 det(V) · det(U)= 1 을 만족해야 함*

    이때, V와 U가 정수이면서, 위의 수식을 만족하려면 행렬식이 ±1인 경우만 가능

    이를 만족하는 행렬은 많으나, $\boldsymbol{U} = \begin{pmatrix} \mathbf{1} & \boldsymbol{c} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}$ **면 만족** 가능, 이를 unimodular matrix (U)*라고 함

- 행렬식 조건을 만족하는 수많은 행렬이 있음
- **하나의 기저에 $\boldsymbol{U}$를 곱하면 $\boldsymbol{L(B) = L(B \cdot U)}$ 성립**
- **따라서, 격자의 기저는 유일하지 않으며, 서로 다른 기저로 동일한 격자 생성 가능**

*det(A): A에 대한 행렬식 (determinant)
* Unimodular matrix : 모든 요소가 정수이면서, 행렬식이 ±1을 만족

Lattice basis and Cryptography

- 하나의 격자를 생성하는 여러 기저가 있음을 확인하였음
- 더 나아가 기저에는 **Good basis, Bad basis**가 존재
  - Good basis (길이가 짧은 벡터로 구성됨, ex: HKZ*)
  - **Good basis → Bad basis :** easy ($B \cdot U^n$; Good basis에 U를 여러 번 곱하여 **bad basis 생성 가능**)
  - **Bad basis → Good basis :** hard (축소 필요)
  - 공개키 암호에서 **개인키로 공개키를 생성**하고, **공개키를 소인수 분해**하여 개인키를 얻는 것과 유사

- 암호에서는 공개키로 **Bad basis**를 사용해야 풀기 어려워짐
  - **공개키** : Bad basis
  - **개인키** : Good basis (Bad basis로 생성된 격자와 **동일한 격자를 생성**하는 basis)

*HKZ (reduced form) : 크기가 작은 벡터들로 이루어진 표현법, 좋은 기저에 해당

Shortest Vector Problem (SVP)

- 격자 상에서 영벡터가 아닌 **가장 짧은 벡터를 찾는 문제**
- Basis를 입력으로 받아, 가장 짧은 벡터를 출력
  - **해가 유일하지 않음** (1개 이상 (ex: $-\mathrm{x}, \mathrm{x} \in L$), 해가 240개인 L도 존재한다고 함)

- **SVP 해결이 어려워지는 경우**
  - **Bad basis가 입력**일 경우, 해결 어려움
    - **Good basis**가 입력될 경우, 해당 **basis 자체에 가장 짧은 벡터가 포함**될 가능성이 높음
  - **Rank (n, Basis의 벡터의 개수)가 커질수록** 어려움 (NP-hard)
    - 암호는 $\boldsymbol{n} \geq \mathbf{500}$인 경우를 사용
    - 뒤에 나올 AKS, Sieve 등은 **50~60을 타겟**으로 함

- **정보의 비대칭성***으로 인해 해결이 어려워지는 경우를 **암호에 활용** 가능

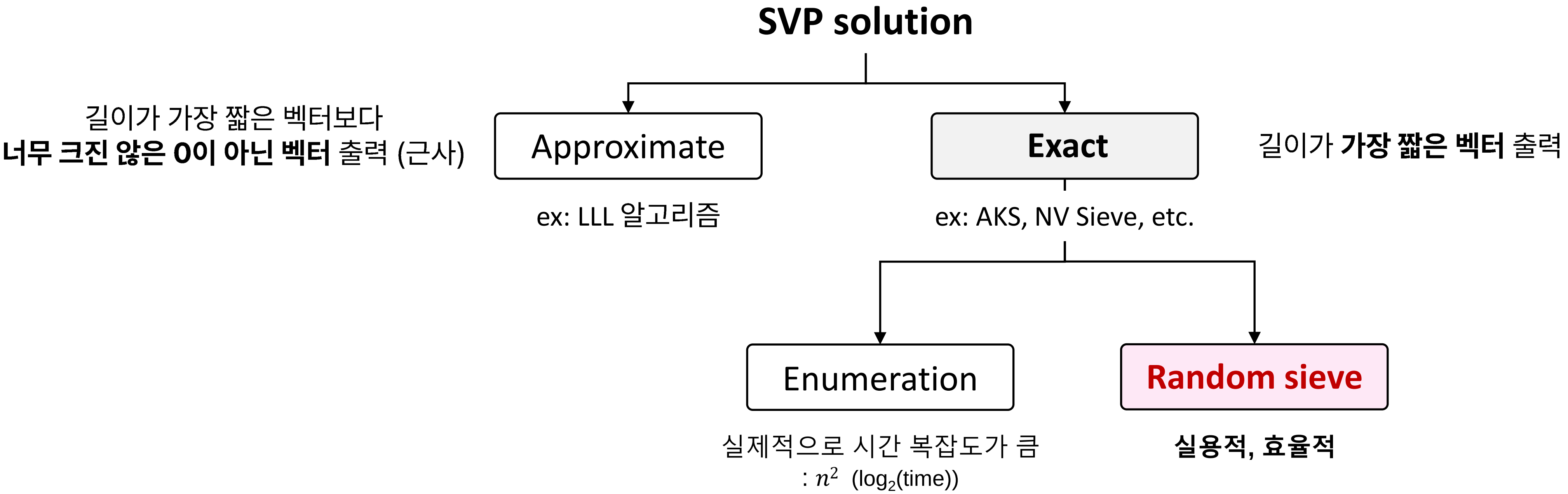
*정보의 비대칭성 : 한 방향으로의 연산은 쉽지만, 반대 방향은 어려움

SVP와 격자 기반 암호의 관계

- **SVP와 같은 격자 문제는 격자 기반 암호에 활용됨**
  - 격자 기반 암호화의 보안 강도는 **격자 문제의 어려움에 기반** (ex: RSA-소인수분해)
  - 정보의 비대칭성과 같은 **One-wayness를 활용**하여 격자 기반 암호 설계
  - Worst case에서 SVP를 풀기 어렵다는 것을 활용

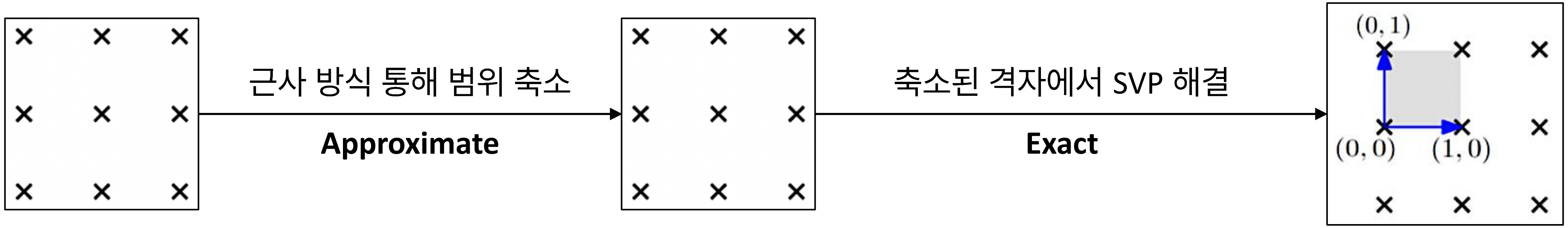
- **SVP 해결 → LWE와 같은 격자 기반 암호 체계 위협**
  - 양자 컴퓨터를 사용하여 소인수 분해 문제를 풀 수 있게 되면 RSA가 위협 받는 것과 유사
  - LWE 자체가 아닌 **SVP를 해결하기 위한 알고리즘을 공격 대상으로** 두어도 될 것으로 판단됨

SVP의 범주


SVP solution
길이가 가장 짧은 벡터보다
너무 크진 않은 0이 아닌 벡터 출력 (근사)
Approximate
ex: LLL 알고리즘
Exact
길이가 **가장 짧은 벡터** 출력
ex: AKS, NV Sieve, etc.
Enumeration
실제적으로 시간 복잡도가 큼
: n^2 ($\log_2$(time))
Random sieve
실용적, 효율적

SVP의 범주

고차원에서 Approximate 통해 범위 축소 후, 해당 벡터들을 입력으로 Exact 방식 수행하여 SVP 해결



고차원 격자
(Rank ≥ 100)

저차원 격자
(Rank ≤ 60)

- SVP를 위한 알고리즘 실행이 아닌 호출로 인한 **시간 오버헤드가 크게 발생**
- SV 찾기가 아닌 범위 축소를 위해 근사 방식 사용
  즉, **고차원 격자 상의 벡터를 걸러내는 역할**
  (고차원에서는 **근사 방식만 효율적**)

- Exact 알고리즘을 집중적으로 사용
- **정확하게 가장 짧은 벡터** 찾아냄

- 고차원에서는 **범위 축소**를 위한 **Approximate** 방식 사용
- **정확한 가장 짧은 벡터**를 찾는 방식은 **Exact** 알고리즘
- 따라서, 최상의 **실용적/이론적 SVP 솔루션은 저차원에서 정확하며 효율적**이어야 함
- 낮은 차원에서의 SVP를 정확하게 해결한 후, **해결 가능한 가장 높은 차원을 결정**하는 것이 중요

AKS Sieve

- 가장 유명한 **초기의 Exact 알고리즘**

- **한계점**
  - 많은 파라미터를 사용하지만, 최적 파라미터에 대한 언급 부재
  - **높은 시간 및 공간 복잡도**
  - **Impractical**
  - 발표 이후, 실질적 구현 및 분석 부재

[AKS] M.Ajtai,R.Kumar,andD.Sivakumar,"Asievealgorithmfortheshortest lattice vector problem," in Proc. 33rd Annu. ACM Symp. Theory Comput. (STOC), Heraklion, Greece, 2001, pp. 601–610.

NV Sieve

- Practical하지 않은 **AKS의 단점을 보완**하기 위한 Exact 알고리즘 등장
  - AKS에 비해 **낮은 시간 및 공간 복잡도**
  - **실용적**이며 **실제 구현 및 평가** 가능

- **이후, 많은 Sieve 알고리즘들 등장**
  - 더 낮은 시간 및 공간 복잡도를 갖는 방식들도 존재

Algorithm Name[References]	$\log_2(\text{time})$	$\log_2(\text{space})$
AKS Sieve [4]	$5.9n$	$2.95n$
NV Sieve [65]	$0.415n$	$0.2075n$
2-level NV Sieve [80]	$0.3836n$	$0.2557n$
3-level NV Sieve [82]	$0.3778n$	$0.2833n$
List Sieve [79]	$3.199n$	$2.1325n$
List Sieve-Birthday [67]	$2.465n$	$1.233n$
Gauss Sieve [79]	$0.415n$	$0.2075n$
NV+angular LSH [44]	$0.3366n$	$0.2075n$
Hash Sieve [44]	$0.3366n$	$0.3366n$
Sphere Sieve [47]	$0.298n$	$0.2075n$
NV+sub-quadratic NNS [13]	$0.3112n$	$0.2075n$
CP Sieve [14]	$0.298n$	$0.298n$
LD Sieve [11]	$0.292n$	$0.2075n$
Overlattice Sieve [12]	$0.3774n$	$0.2925n$

[NV Sieve] Nguyen, Phong Q., and Thomas Vidick. "Sieve algorithms for the shortest vector problem are practical." Journal of Mathematical Cryptology 2.2 (2008): 181-207.

NV Sieve 선택 이유

- AKS 보다 **실제적 / 효율적**이며, **Sieve의 가장 기초가 되는 알고리즘**
- 추가 구현이 들어가는 다른 sieve 알고리즘 구현 시, quantum 비용을 고려하여야 함
  - Classical에서 간단한 구현이 quantum에서는 큰 비용을 야기할 수 있음
- 또한, 더 나은 방식이라고 해서 quantum에서 무조건 효율적이지는 않음

  반대로, 기초적인 방식이라고 해서 무조건 효율적이지는 않음

Algorithm Name[References]	$\log_2(\text{time})$	$\log_2(\text{space})$
AKS Sieve [4]	$5.9n$	$2.95n$
NV Sieve [65]	$0.415n$	$0.2075n$
2-level NV Sieve [80]	$0.3836n$	$0.2557n$
3-level NV Sieve [82]	$0.3778n$	$0.2833n$
List Sieve [79]	$3.199n$	$2.1325n$
List Sieve-Birthday [67]	$2.465n$	$1.233n$
Gauss Sieve [79]	$0.415n$	$0.2075n$
NV+angular LSH [44]	$0.3366n$	$0.2075n$
Hash Sieve [44]	$0.3366n$	$0.3366n$
Sphere Sieve [47]	$0.298n$	$0.2075n$
NV+sub-quadratic NNS [13]	$0.3112n$	$0.2075n$
CP Sieve [14]	$0.298n$	$0.298n$
LD Sieve [11]	$0.292n$	$0.2075n$
Overlattice Sieve [12]	$0.3774n$	$0.2925n$

NV Sieve 동작 과정

Algorithm 4 Finding short lattice vectors based on sieving

Input: An LLL-reduced basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of a lattice L, a sieve factor γ such that $2/3 < \gamma < 1$, and a number N.

Output: A short non-zero vector of L.

$S \leftarrow \emptyset$
for $j = 1$ to N **do**
  $S \leftarrow S \cup \mathsf{sampling}(B)$ using algorithm $\mathcal{K}$ described in Section 4.2.1.
end for
Remove all zero vectors from S.
$S_0 \leftarrow S$
repeat
  $S_0 \leftarrow S$
  $S \leftarrow \mathsf{latticesieve}(S, \gamma)$ using Algorithm 5.
  Remove all zero vectors from S.
until $S = \emptyset$
Compute $\mathbf{v}_0 \in S_0$ such that $\|\mathbf{v}_0\| = \min\{\|\mathbf{v}\|, \mathbf{v} \in S_0\}$
return $\mathbf{v}_0$

- **목적** : 너무 많은 벡터를 손실하지 않으면서,
  가장 짧은 벡터 찾기 (영벡터 제외)
- **입력** : LLL 통해 리덕션 된 basis
- **출력** : 영벡터가 아닌 가장 짧은 벡터

- **Sieve factor γ**
  - $\frac{2}{3} < \gamma < 1$
    (1에 가까울수록 좋은 것으로 파악)
    정해지는 방법에 대해선 추후 공부할 예정

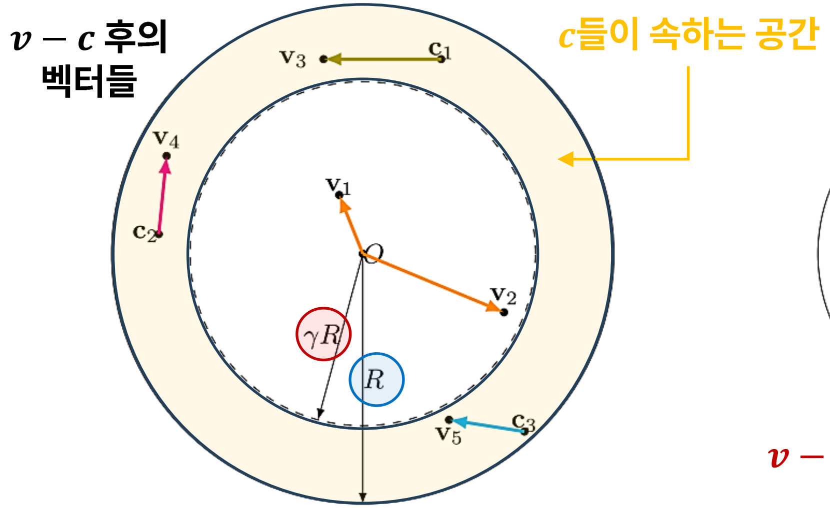
- **전체 구조**
  1. **LLL** 통해 리덕션 된 **basis**로부터 랜덤 샘플링 (κ)
  2. 샘플링 통해 S 구성 후, 영벡터 제거 → S_0
     - **S_0: 영벡터 제거된 + Sieve 적용 후 영벡터 제거된 출력 저장**
  3. S가 공집합이 될 때까지 **반복 적용**
  4. **S_0에 속하는 벡터 중 길이가 가장 짧은 벡터 반환**

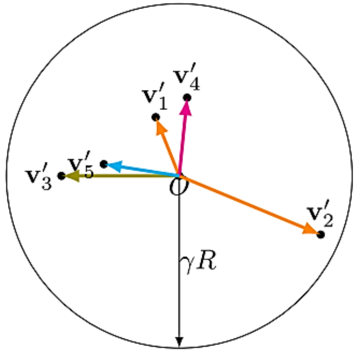
NV Sieve 동작 과정

Algorithm 5 The lattice sieve

Input: A subset $S \subseteq B_n(R)$ of vectors in a lattice L and a sieve factor $2/3 < \gamma < 1$.
Output: A subset $S' \subseteq B_n(\gamma R) \cap L$.

$R \leftarrow \max_{\mathbf{v} \in S} \|\mathbf{v}\|$
$C \leftarrow \emptyset$, $S' \leftarrow \emptyset$
for $\mathbf{v} \in S$ **do**
  if $\|\mathbf{v}\| \leq \gamma R$ **then**
    $S' \leftarrow S' \cup \{\mathbf{v}\}$
  else
    if $\exists \mathbf{c} \in C\ \|\mathbf{v} - \mathbf{c}\| \leq \gamma R$ **then**
      $S' \leftarrow S' \cup \{\mathbf{v} - \mathbf{c}\}$
    else
      $C \leftarrow C \cup \{\mathbf{v}\}$
    end if
  end if
end for
return S'





$v - c$ 들을 원점으로 이동해보면?
범위 안에 속함

- **목적** : 짧은 벡터에 대한 손실이 없게 하기 위해 c**를 랜덤으로 선택**하여 **범위를 줄여 나가며 γR 보다 짧은 벡터 얻기**
- **입력** : 최대 길이가 R인 격자 상의 벡터
- **출력** : γR보다 짧은 격자 상의 벡터
- **용어**
  - $B_n(R)$: 원점으로부터의 길이가 R보다 작은 격자 상의 벡터
  - S' : 범위 내의 벡터들을 저장
  - c : $\gamma R \leq x \leq R$에 속하는 충분한 수의 격자 상의 점
- **동작 과정**
  1. S 에 속한 벡터들 중 최대 길이 → R
     C, S' 초기화
  2. **γR보다 길이가 짧은 벡터들은 S'에 저장**
  3. **γR 보다 길이가 긴 벡터들은 c라는 포인트와 뺄셈** 한 후,
     그 길이가 **γR보다 짧으면 S'에 저장** / 길면 C에 저장
  4. S' 반환 (γR보다 길이가 짧은 벡터들)
- 해당 과정은 **알고리즘 4의 line 9에 해당되므로 반복**
  - 반복적으로 수행하여 충분히 짧은 벡터 집합들을 얻고,
    그 중에서 가장 짧은 벡터를 찾아냄

NV Sieve 중요 포인트

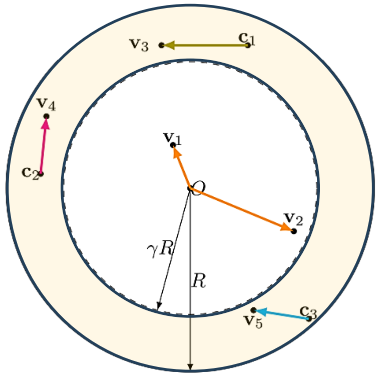
- **알고리즘 복잡도에 영향을 미치는 결정적 부분**
  - **c의 포인트 수를 측정하는 부분**
    - 충분히 많은 포인트들이 있고 이를 이용하여
      γR보다 짧은 길이의 벡터를 만들 수 있는 점을 찾아야 함
  - **처음 주어진 서브셋 S의 크기가 클수록 좋지 않음**
    - S의 rank가 커지면 c의 개수도 많아짐
      - **c도 격자 상의 벡터이고, S의 부분 집합이므로**
    - 해당 부분이 c의 포인트 수를 측정하는 것에 영향을 주므로
      전체적인 Sieve 알고리즘 퀄리티에 중요

- **실제 구현**
  - 실제 구현 시, rank를 약 50으로 설정
    ($30 \leq$ rank ≤ 48)
  - **$\gamma = 0.97$** 사용
  - **LLL 적용** 후 축소된 **8000개의 벡터를 입력**으로 사용
  - 48 rank까지 **700 MB RAM** 사용
    (많은 메모리 사용되지 않음)
  - C++의 long type vector 사용
    (64-bit OS에서 64-bit)

  - 구현 관련 부분은 추후 더 자세히 알아볼 예정

NV Sieve on Grover

- **적용 지점**
  - 우선, 고차원에서 벡터를 걸러내는 LLL은 classical로 구현
  - **Sieve (알고리즘 5)에서 c를 찾기 위한 과정에 Grover 적용**
  - **c는 격자 상의 포인트 (노랑)**, 이를 **중첩상태로 준비**하여 γR보다 짧은 길이의 벡터를 만들 수 있는 점을 찾아냄
    → 짧은 벡터를 찾기 위함



- **기대 효과**
  - 알고리즘의 시간 복잡도 감소 (공간 복잡도 감소 없음)*
    - **$\log_2$(time) : 0.415 → 0.312**
    - **$\log_2$(space) : 0.2075 → 0.2075**

- 그러나, **이론적 계산에 의한 결과**이며, 실험에 의한 분석 및 평가는 없음

*Laarhoven, Thijs, Michele Mosca, and Joop Van De Pol. "Finding shortest lattice vectors faster using quantum search." Designs, Codes and Cryptography 77 (2015): 375-400.

NV Sieve on Grover

- **실제 구현 시 예상되는 어려움들**
  - **Solution 개수에 대한 고려**
    - Solution 개수에 따라 Grover iteration이 달라지므로 중요한 부분
    - SVP에서는 1개 이상의 해가 존재하지만, 랜덤 격자를 사용하므로 그 개수는 구현해봐야 알 수 있을 것으로 보임
  - Grover search 대상들인 **격자 상의 벡터들을 중첩 상태로 어떻게 준비할지**
  - 올바른 양자 회로 구현물이 나올지
    - 알고리즘 5의 **벡터 간의 뺄셈 및 범위 비교 로직을 올바르게 수행 가능할지**

관련 Library

- fplll (https://github.com/fplll/experimental_sieve/blob/ktuplenew/fplll/sieve/sieve_gauss.cpp)
- 입력으로 사용 가능한 **벡터 제공**
- 원하는 **격자 생성** 가능 (NV Sieve는 Random lattice 사용; 모든 벡터가 균일하게 분포한다고 가정)
- Gaussian Sieve 샘플 코드 제공
    - NV Sieve가 더 간단하므로 **불필요한 부분 제거** 및 **필요 부분 추가** 가능할 것으로 보임

향후 계획

- 구체적인 파라미터의 값과 이유 파악
- Grover 적용한 연구 동향 좀 더 조사
- 30 페이지 채우기
- 이후, 구현 위한 코드 분석

감사합니다.