Enumeration Algorithms

1 Introduction

In the previous lecture, we defined computational problems for lattices, namely the algorithmic task of finding short non-zero vectors of a lattice (SVP), or a lattice point close to a given target (CVP, BDD). We then provided two algorithms (Simple-Rounding, Nearest-Plane) which, given a basis $\mathbf{B}$ of a lattice L, would tile the space according to the L-tilings $T = \mathcal{P}(\mathbf{B})$ and $T = \mathcal{P}(\mathbf{B}^\star)$ respectively. That is, given a target $\mathbf{t}$, these algorithm can find a lattice point $\mathbf{v} \in L$ such that the error $\mathbf{e} = \mathbf{t} - \mathbf{v}$ belongs to T. By studying the inner radius $\nu(\mathcal{P}(\mathbf{B}))$ and outer radius $\mu(\mathcal{P}(\mathbf{B}))$ we finally concluded with a length guarantees (as a function of $\mathbf{B}$) for which those algorithms solve approximate versions of BDD and CVP.

These algorithm were "fast"; they ran in time polynomial in the size of the input, but they only guaranteed to solve the closest vector problem up to a potentially large approximation factor. Today's lecture is about "slow" algorithms, whose running time can be as large as exponential in the input size, but that can solve SVP, CVP and BDD exactly. Namely, there will be two algorithms directly inspired from Simple-Rounding and Nearest-Plane, but instead of considering only one candidate solution, they will *enumerate* many potential solution and take the shortest/closest one.

The first one (SimpleEnum) is not particularly used in practice, nor very much studied in the literature because it is always slower than the former (FinckePohstEnum). But we find it to be a valuable pedagogical step.

2 Simple Enumeration

The simple enumeration algorithm generalizes simple-rounding, and when the parameter $\ell = 1$, the two algorithms are the same. This generalized algorithm again reduces the question to the lattice $\mathbb{Z}^n$, where finding a close point is as easy as rounding each coordinate. Yet, because this transformation from the lattice L to the lattice $\mathbb{Z}^n$ also distorts the space $\mathbb{R}^n$, we will now consider not only the closest integer (by rounding), we will take the set of all nearby integers.

Algorithm 1: SimpleEnum$(\mathbf{B}, \ell, \mathbf{t})$: Simple Enumeration Algorithm

Input : A basis $\mathbf{B} \in \mathbb{Q}^{n \times n}$ of a full rank lattice Λ, a target $\mathbf{t} \in \mathrm{Span}_{\mathbb{Q}}(L)$, and a side length $\ell \geq 1$.

Output: $\mathbf{c} \in L$ a closest lattice point to $\mathbf{t}$ that lies in $\mathbf{t} + \ell \cdot \mathcal{P}(\mathbf{B})$

$\mathbf{c} = (\infty, \infty, \ldots, \infty)$
$\mathbf{t}' \leftarrow \mathbf{B}^{-1} \cdot \mathbf{t}$
for $\mathbf{v}' \in \mathbb{Z}^n \cap (\mathbf{t}' + [-\ell/2, \ell/2)^n)$ **do**
  $\mathbf{v} = \mathbf{B} \cdot \mathbf{v}'$
  if $\|\mathbf{v} - \mathbf{t}\| < \|\mathbf{c} - \mathbf{t}\|$ **then**
    $\mathbf{c} \leftarrow \mathbf{v}$
  end
end
return $\mathbf{c}$

Note that enumerating integer points in an interval $t + [-\ell/2, \ell/2)$ simply consists of running a for loop from $\lceil t - \ell/2 \rceil$ to $\lceil t + \ell/2 \rceil - 1$. To do so in arbitrary dimension n, one could nest n such for loops, which is inconvenient to write down; technically this would require writing down a different algorithm for each dimension n. Recursive programming techniques allow to solve this issue. For convenience, we will simply write a for loop enumerating integer vectors in a hypercube $\mathbb{Z}^n \cap (\mathbf{t}' + [-\ell/2, \ell/2)^n)$.[1]

LEMMA 1 *Algorithm SimpleEnum is correct, that is, SimpleEnum*$(\mathbf{B}, \ell, \mathbf{t})$ *outputs a vector* $\mathbf{c} \in L$ *such that*

$$\mathbf{c} \in \underset{\mathbf{v} \in L \cap \mathbf{t} + \ell \cdot \mathcal{P}(\mathbf{B})}{\arg\min} \|\mathbf{v} - \mathbf{t}\|.$$

Note that because there may be more than one closest vector, we write $\mathbf{c} \in \arg\min$ and not $\mathbf{c} = \arg\min$.

PROOF: First, except before the first loop, $\mathbf{c}$ is always equal to $\mathbf{B}$ times an integer vector. Because $\ell \geq 1$, the loop is iterated at least once. Hence, the final value of $\mathbf{c}$ is in the lattice L. Next, we argue that the visited vectors $\mathbf{v}$ are exactly those belonging to $L \cap \mathbf{t} + \ell \cdot \mathcal{P}(\mathbf{B})$. This is merely a matter of rewriting

$$\mathbf{B} \cdot (\mathbb{Z}^n \cap (\mathbf{t}' + [-\ell/2, \ell/2)^n)) = (\mathbf{B} \cdot \mathbb{Z}^n) \cap \left(\mathbf{B} \cdot \mathbf{t}' + \ell \cdot \mathbf{B} \cdot [-1/2, 1/2)^n)\right) = L \cap (\mathbf{t} + \ell \cdot \mathcal{P}(\mathbf{B})).$$

□

COROLLARY 2 *Algorithm SimpleEnum*$(\mathbf{B}, \ell, \cdot)$ *solves* α*-BDD up to* $\alpha = \min\{1/2, \ell \cdot \nu(\mathcal{P}(\mathbf{B}))/\lambda_1(L)\}$.

PROOF: By the definition of α-BDD, we are looking for the unique lattice point $\mathbf{v}$ in the ball of radius $r = \alpha \cdot \lambda_1(L) \leq \ell \cdot \nu(\mathcal{P}(\mathbf{B}))$ centered at $\mathbf{t}$. By definition of the inner radius $\nu(\mathcal{P}(\mathbf{B}))$, it holds that the ball of radius $\alpha\lambda_1(L)$ is contained in the parallelepiped $\ell \cdot \mathcal{P}(\mathbf{B})$. □

By a similar reasoning, we can also claim that for large enough ℓ, SimpleEnum solves Exact-CVP.

COROLLARY 3 *If* $\ell \cdot \nu(\mathcal{P}(\mathbf{B})) \geq \mu(L)$ *for a basis* $\mathbf{B}$ *of* L*, then Algorithm SimpleEnum*$(\mathbf{B}, \ell, \cdot)$ *solves Exact-CVP in* L.

2.1 SVP variant

We note that by setting $\mathbf{t} = 0$, we can enumerate lattice vectors close to the origin, and hence find a short vector rather than a close vector. This requires the mild modification of ignoring $\mathbf{v}' = 0$ in the main loop; we we call this variant $\mathsf{SimpleEnum}^{\mathsf{SVP}}(\mathbf{B}, \ell)$. Using once again the same reasoning as for Corollary 3, we can claim

COROLLARY 4 *If* $\ell \cdot \nu(\mathcal{P}(\mathbf{B})) \geq \lambda_1(L)$ *for a basis* $\mathbf{B}$ *of* L*, then Algorithm SimpleEnum*$^{SVP}(\mathbf{B}, \ell)$ *solves Exact-SVP in* L.

[1] This kind abuse of notation should not be used without justification, as one may accidentally hide complexity and difficulty by doing for loops over arbitrary sets.

2.2 Complexity

LEMMA 5 *The complexity of* $\mathsf{SimpleEnum}(\cdot, \ell, \cdot)$ *is of* $O(n^3 + n^2 \cdot \lceil \ell \rceil^n)$ *arithmetic operations. Its bit complexity is* $\lceil \ell \rceil^n$ *up to a polynomial factor in the size of the input.*

PROOF: The $O(n^3)$ term accounts for the cost of matrix inversion at the $\mathbf{t}' \leftarrow \mathbf{B}^{-1} \cdot \mathbf{t}$ step. The remaining term is the cost of the loop. Each iteration costs $O(n^2)$ arithmetic operations, where the most costly step is the matrix-vector product $\mathbf{v} = \mathbf{B} \cdot \mathbf{v}'$. At last, we note that the loop iterates over at most $\lceil \ell \rceil^n$ elements. □

One can further remark that this asymptotic upper bound (mind the big Oh!) is actually rather tight, as we can also lower-bound the number of loops by $\lfloor \ell \rfloor^n$. For integers ℓ the number of loops is therefore always the same independently of $\mathbf{t}$, but for non-integral ℓ it does vary a bit.

COROLLARY 6 *Up to polynomial factor in the size of the input:*

- $\mathsf{SimpleEnum}(\mathbf{B}, \lambda_1(L), \cdot)$ *solves Exact-CVP (and therefore BDD) in time* $\lceil \mu(L)/\nu(\mathcal{P}(\mathbf{B})) \rceil^n$,
- $\mathsf{SimpleEnum}^{SVP}(\mathbf{B}, \mu(L))$ *solves Exact-SVP in time* $\lceil \lambda_1(L)/\nu(\mathcal{P}(\mathbf{B})) \rceil^n$.

Not that running the above requires knowing $\lambda_1(L)$ and $\mu(L)$ in advance. These can be known or approximately known in certain scenario, but what can we do if they are not? A crude upper bound can be given in terms of the input basis: $\lambda_1(L)$ is upper bounded by the length of the smallest vector of the input basis, while $\mu(L)$ is upper bounded by $\frac{1}{2}\sum \|\mathbf{b}_i\|$. But for a basis with arbitrarily long basis vectors, this can give an arbitrarily loose upper-bound on the actual quantity.

In the case of SVP, one can make do without knowledge of $\lambda_1(L)$ without losing much efficiency: one can start trying with $\ell = 1$, and increase ℓ by a factor $(1 + 1/n)$ until a vector of length less than $\ell \cdot \nu(L)$ is found.

3 Fincke-Pohst Enumeration

We now discuss an enumeration algorithm that generalizes the principle of the Nearest-Plane algorithm. It will proceed recursively by projection onto lower-dimensional lattices. It was invented by Fincke and Pohst, and is sometime referred simply as *lattice enumeration*, or qualified as a *Branch and Bound* type of algorithm.

Our presentation might be quite different from the literature in several ways. It is usually described in a much more explicit way using coordinates, being closer to actually implemented machine code. A more fundamental difference is that the version given here is a "breadth-first" approach, while the version used in practice is "depth-first". This difference will be the object of a detailed exercise, but we can already say that the latter is preferable is it consumes much less memory while having the same running time. We nevertheless find the former conceptually simpler.

In more detail, the algorithm enumerates lattice points in a euclidean ball, and is therefore readily fit to solve BDD/CVP/SVP problems in the ℓ_2 norm. But it can also be tweaked to work for arbitrary norm, by over-approximating the associated arbitrary ball by a euclidean ball.

The principle used in each recursive call is that for some projection map $\pi_{\mathbf{v}}^{\perp}$ orthogonal to a vector $\mathbf{v}$, the projection of lattice points whose distance to a target $\mathbf{t}$ is less than r is also at a distance less than r from the projection of $\mathbf{t}$. In equations:

$$\pi_{\mathbf{v}}^{\perp}(L \cap (\mathbf{t} + r\mathfrak{B}_2^n)) \subseteq \pi_{\mathbf{v}}^{\perp}(L) \cap (r\mathfrak{B}_2^{n-1} + \pi_{\mathbf{v}}^{\perp}(\mathbf{t})), \tag{1}$$

where $\mathfrak{B}_2^k$ is the unit euclidean ball in k dimensions (implicitly we are taking the ball to be in k dimensions on a k-dimensional subspace in $\mathbb{R}^n$). We can arbitrarily choose a primitive vector in L and will project the lattice orthogonally from this to get a lattice in $n-1$ dimensions. In Algorithm 2 below, we use a basis vector, but conceptually the algorithm is the same regardless of what primitive vectors are chosen at each step.

For a given vector $\mathbf{v}$ with orthogonal projection map $\pi_{\mathbf{v}}^{\perp}(\cdot)$, the algorithm will enumerate over the lattice $\pi_{\mathbf{v}}^{\perp}(L)$ to find the set $S' := \left\{\mathbf{s}' \in \pi_{\mathbf{v}}^{\perp}(L) : \|\pi_{\mathbf{v}}^{\perp}(\mathbf{t}) - \mathbf{s}'\|_2 \leq r\right\}$, i.e. the points whose distance is at most r from the target $\pi_{\mathbf{v}}^{\perp}(\mathbf{t})$. Then for every $\mathbf{s}' \in S'$, we find all lattice elements in the pre-image $(\pi_{\mathbf{v}}^{\perp})^{-1}(\mathbf{s}')$ whose distance to $\mathbf{t}$ is not bigger than r. Since $\mathbf{v}$ is orthogonal to each element of S', the distance criterion can be checked easily by only taking lattice vectors in $(\pi_{\mathbf{v}}^{\perp})^{-1}(\mathbf{s}')$ whose distance to $\mathbf{s}'$ is not greater than $\sqrt{r^2 - \|\mathbf{s}' - \pi_{\mathbf{b}_1}^{\perp}(\mathbf{t})\|_2^2}$. Finding all such lifts in $(\pi_{\mathbf{v}}^{\perp})^{-1}(\mathbf{s}')$, however, requires us to first calculate one lift of $\mathbf{s}'$ (denoted $\mathbf{x}$ in Algorithm 2) and then add integer multiples of $\mathbf{v}$ to it.

Algorithm 2: FinckePohstEnum$(\mathbf{B}, \mathbf{t}, r)$: Fincke-Pohst Enumeration Algorithm

Input : A basis $\mathbf{B} = (\mathbf{b}_1, \dots \mathbf{b}_n) \in \mathbb{Q}^{d \times n}$ of a lattice Λ, a target $\mathbf{t} \in \mathrm{Span}_{\mathbb{Q}}(L)$, and a radius $r \geq 0$.

Output: A list containing all vectors $\mathbf{x} \in L$ satisfying $\|\mathbf{x} - \mathbf{t}\|_2 \leq r$.

if $n = 1$ **then**
  return $\left\{z \cdot \mathbf{b}_1 \mid z \in \mathbb{Z}, \left\| \left(\frac{\langle \mathbf{0} - \mathbf{t}, \mathbf{b}_1 \rangle}{\|\mathbf{b}_1\|^2} + z \right) \mathbf{b}_1 \right\|_2 \leq r \right\}$
end
$S \leftarrow \{\}$
$\mathbf{B}' \leftarrow \left(\pi_{\mathbf{b}_1}^{\perp}(\mathbf{b}_2), \dots, \pi_{\mathbf{b}_1}^{\perp}(\mathbf{b}_n) \right)$
$S' \leftarrow$ FinckePohstEnum$(\mathbf{B}', \pi_{\mathbf{b}_1}^{\perp}(\mathbf{t}), r)$
for $\mathbf{s}' \in S'$ **do**
  $\rho \leftarrow \sqrt{r^2 - \|\mathbf{s}' - \pi_{\mathbf{b}_1}^{\perp}(\mathbf{t})\|_2^2}$
  $\mathbf{y} \leftarrow \mathbf{B}'^{-1} \cdot \mathbf{s}' \quad \in \mathbb{Z}^{n-1}$ // *the coordinates of* $\mathbf{s}'$ *in terms of* $\mathbf{B}'$
  $\mathbf{x} \leftarrow \mathbf{B} \cdot (0, \mathbf{y})^{\top} \quad \in L$ // *an arbitrary lift of* $\mathbf{s}'$ *to* L
  $Z \leftarrow \left\{ z \in \mathbb{Z} \mid \left\| \left(\frac{\langle \mathbf{x} - \mathbf{t}, \mathbf{b}_1 \rangle}{\|\mathbf{b}_1\|^2} + z \right) \mathbf{b}_1 \right\|_2 \leq \rho \right\}$
  $S \leftarrow S \cup \{\mathbf{x} + z\mathbf{b}_1 : z \in \mathbb{Z}\}$
end
return S

Lemma 7 *Algorithm 2 is correct. That is, the algorithm outputs the set* $S = L \cap (\mathbf{t} + r \cdot \mathfrak{B}_2)$.

Proof: The proof essentially follow the explanation given above. The case $n = 1$ is correct by construction. We then proceed by induction, assuming the algorithm is correct for dimension

$n-1$. One can see that $\mathbf{x} \in L$ is indeed a lift of $\mathbf{s}'$, that is $\pi_{\mathbf{b}_1}^{\perp}(\mathbf{x}) = \mathbf{s}'$. Then that $\mathbf{x} + Z \cdot \mathbf{b}_1$ is indeed the set of pre-images of $\mathbf{s}'$ whose norm is less than r, by invoking invoking pythagoras for the orthogonal vectors $\mathbf{b}_1 \perp \mathbf{s}'$. □

Note that Fincke-Pohst does not itself solve CVP or SVP but instead finds a large set of short or close vectors, and it only remains to pick the closest among them as done for CVP in Algorithm 3. The SVP variant is equally obvious.

Algorithm 3: FinckePohstCVP$(\mathbf{B}, \mathbf{t}, r)$: Fincke-Pohst CVP Algorithm

Input : A basis $\mathbf{B} = (\mathbf{b}_1, \dots \mathbf{b}_n) \in \mathbb{Q}^{n \times n}$ of a full rank lattice Λ, a target $\mathbf{t} \in \mathrm{Span}_{\mathbb{Q}}(L)$, and a radius $r \geq \mathrm{dist}(\mathbf{t}, L)$.

Output: A closest vector in L to the target $\mathbf{t}$.

$S' \leftarrow \mathsf{FinckePohstEnum}(\mathbf{B}, \mathbf{t}, r)$

return $\arg\min_{\mathbf{v} \in S'} \|\mathbf{v} - \mathbf{t}\|_2$

3.1 Complexity

To study the complexity of this algorithm, it is useful to think of the set of visited vectors in the various projected lattices organized in a *leveled tree*. Recall from the Gram-Schmidt Orthogonalization section of last lecture that we denote $\pi_i = \pi_{\mathbf{b}_1, \dots \mathbf{b}_{i-1}}^{\perp}$.

At the bottom of the tree is the trivial 0-dimensional lattice, with a single node. At level i the nodes are the elements of the set $S_i = \pi_{n-i}(L) \cap r\mathfrak{B}_2$. From a vector $\mathbf{v} \in S_i$ and $\mathbf{v}' \in S_{i+1}$ we put a directed edge from $\mathbf{v}$ to $\mathbf{v}'$ when $\pi_{n-i}(\mathbf{v}') = \mathbf{v}$. Note that by Equation (1), every node has one parent, i.e. any node at level $i+1$ is connected to one (and exactly one) node at level i, and is therefore connected to the root. Note however nodes may have zero, one, or several children.

The algorithm can be understood as a *breadth-first* exploration of this tree. One first constructs the set S_i recursively, and from there, one constructs S_{i+1} by constructing the children of each $\mathbf{v} \in S_i$; this is the set $\mathbf{x} + Z\mathbf{b}_1$ constructed in the loop of Algorithm 2. Constructing each child is algorithmically easy, and has complexity polynomial in the size of the input. The main factor of the complexity of the algorithm is therefore simply the size of the entire tree $\sum_{i=1}^{n} |S_i|$.

Note that we can upper bound the size of $|S_i|$ by using the generalization of Proposition 13 of Lecture 1, restated below for convenience.

Lemma 8 *Let L be a lattice of rank n. Then, for any radius $r \geq 0$ and any shift $\mathbf{t} \in \mathrm{Span}_{\mathbb{R}}(L)$, it holds that* $\frac{|r\mathfrak{B}^n \cap (L+\mathbf{t})|}{\mathrm{vol}(\mathfrak{B}^n)} \leq \frac{(r+2\mu(L))^n}{\det(L)}$.

That is, we have

$$|S_i| \leq \frac{(r + 2\mu(\pi_{n-i}(L)))^i}{\det(\pi_{n-i}(L))) \cdot \mathrm{vol}(\mathfrak{B}^i)}.$$

Note that the covering radius $\mu(L)$ of a lattice L can only decrease under projection, hence we can upper bound the numerator by $(r + 2\mu(L))^n$. Note further that the Gram-Schmidt orthogonalization of the basis $\pi_i(\mathbf{b}_i), \dots, \pi_i(\mathbf{b}_n)$ is simply $\mathbf{b}_i^\star, \dots, \mathbf{b}_n^\star$, hence $\det(\pi_{n-i}(L)) = \prod_{j=n-i}^{n} \|\mathbf{b}_j^\star\|$. At last, we recall that $\mathrm{vol}(\mathfrak{B}_2^i) \sim \frac{1}{\sqrt{\pi i}} \left(\frac{2\pi e}{i}\right)^{i/2}$ from Stirling's approximation. We conclude with the following.

THEOREM 9 *The number of nodes visited by* FinckePohstEnum$(\mathbf{B}, r, \cdot)$ *is at most*

$$\sum_{i=1}^{n} \frac{(r+2\mu(L))^i}{\prod_{j=n-i}^{n} \|\mathbf{b}_j^\star\|_2} \cdot \left(\frac{i}{2\pi e}\right)^{i/2} \cdot \sqrt{\pi \cdot i} \cdot (1+o(1)).$$

In particular, it is at most

$$O(n^{3/2}) \cdot \left(\frac{\sqrt{n} \cdot (r+2\mu(L))}{\sqrt{2\pi e} \cdot \min \|\mathbf{b}_i^\star\|_2}\right)^n.$$

3.2 Improvements

As mentioned at the beginning of the section, the above is not the standard form of Fincke-Pohst. We discuss below various enhancements of this algorithm. Those are clever tricks that matter a lot for practice but are hardly visible on the asymptotics.

1. The very first improvement is to switch from a breadth-first version of the algorithm to a depth-first version so as to avoid storing the large intermediate sets S_i, which results in a memory consumption essentially as large as the running time. See Exercise 3 on sheet 3. In fact, this technique drastically reduces memory consumption to barely more than the size of the input.

2. This variation leads to also unlock another opportunistic speed-up for solving SVP and CVP when the exact length is not known in advanced but only upper-bounded. Indeed, each time a solution below a certain radius r is found, one can dynamically decrease the enumeration radius to the length or distance of that vector.

3. With the trick above, we now would like to see the enumeration radius decrease as early as possible, and we can try to optimize our luck by wisely choosing in which order we visit the children of a given node.

4. On a lower implementation level, one can save a linear $\Theta(n)$ factor by carefully writing and maintaining the visited vector written in basis $\mathbf{B}$ and $\mathbf{B}^\star$ rather than writing them in the canonical basis of $\mathbb{R}^n$.

5. The best implementations also de-recursify the algorithm because such recursive function calls are in practice quite costly.

4 Toward Basis Reduction

In all the algorithms we have seen, the geometric shape of the input basis mattered a lot, either to improve the approximation factor of fast algorithms (SimpleRounding, NearestPlane), or to accelerate slow algorithm solving exact SVP and CVP (SimpleEnum, FinckePohstEnum). Focusing on the NearestPlane and FinckePohstEnum algorithms, we noted that

- to improve the approximation factor for approx-CVP via NearestPlane we want to minimize $\mu^{(2)}(\mathcal{P}(\mathbf{B}^\star)) = \frac{1}{2}\sqrt{\sum \|\mathbf{b}_i^\star\|_2^2} \leq \frac{\sqrt{n}}{2} \cdot \max \|\mathbf{b}_i^\star\|_2$

- to improve the approximation factor for BDD via NearestPlane we want to maximize $\nu^{(2)}(\mathcal{P}(\mathbf{B}^\star)) = \frac{1}{2}\min \|\mathbf{b}_i^\star\|_2$
- to speed up SVP/CVP/BDD via FinckePohstEnum we also want to maximize $\nu^{(2)}(\mathcal{P}(\mathbf{B}^\star)) = \frac{1}{2}\min \|\mathbf{b}_i^\star\|_2$

We also remark that solving SVP is equivalent to finding a basis $\mathbf{B}$ that minimizes $\mathbf{b}_1$ which is also equal to $\mathbf{b}_1^\star$. These objectives may sound opposite: for some tasks, we want the Gram-Schmidt norms to be small, sometimes to be large. But this is forgetting the invariant $\prod \|\mathbf{b}_i^\star\| = \det(L)$: making some $\mathbf{b}_i^\star$ small make the other ones larger. Because of this invariant, all these objective are actually aligned: what we want is for the Gram-Schmidt norm to be all *balanced*, all as close as possible to the root determinant $\det(L)^{1/n}$. This would optimize the min, the max and also the Euclidean $\sqrt{\sum \|\mathbf{b}_i^\star\|_2^2}$, according to the inequality of arithmetic and geometric means: $(\sum x_i)/n \geq (\prod x_i)^{1/n}$.

The case of SimpleRounding and SimpleEnum and the geometric control of $\mathcal{P}(\mathbf{B})$ are also related though a bit more complex, and will be discussed as well in further lectures.