In this lecture we present another application of the LLL algorithm, namely, integer programming in fixed dimension.

1 Integer Programming Overview

The Integer Programming problem (IP) is that of deciding whether there exists an integer solution to a given set of m rational inequalities on n variables. Equivalently, given a matrix $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, decide if there is a $z \in \mathbb{Z}^n$ such that $Az \leq b$. Yet another equivalent formulation is: given a matrix $A \in \mathbb{Q}^{m \times n}$ decide whether the set

$$\mathbb{Z}^n \cap \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

is non-empty. The integer programming problem is quite powerful, and many combinatorial problems can be formulated as instances of IP. In fact, it is 'too powerful' since it is **NP**-complete, as the following claim shows.

REMARK 1 Without the requirement on an integer solution (i.e., if we allow $z \in \mathbb{R}^n$), the problem is known as Linear Programming, and has a polynomial time solution (such as the ellipsoid method).

REMARK 2 There are many equivalent formulations of integer programming. We could, for instance, allow equalities in addition to inequalities. Moreover, we could ask to find an integer solution and not just decide if one exists.

CLAIM 1 *The integer programming problem is* **NP***-complete.*

PROOF: IP is in **NP** because the integer solution can be used as a witness and can be verified in polynomial time.[1] We now prove that IP is **NP**-hard by reduction from SAT. A SAT instance is described by a set of Boolean variables and clauses. We reduce it to an Integer Programming instance with the same number of variables. In addition, for each variable v_i we have the constraints

$$0 \leq v_i \leq 1.$$

For each clause we have a constraint that correspond to it; for example, for the clause $v_1 \vee \overline{v_3} \vee v_7$ in the SAT instance, we have the constraint

$$v_1 + (1 - v_3) + v_7 \geq 1.$$

Clearly, this reduction can be done in polynomial time. Moreover, it is easy to verify that if the given SAT instance has a satisfying assignment then the corresponding IP instance has an integer solution and vice versa. □

Although it is **NP**-complete, one might hope to obtain efficient algorithms for the case where the dimension (i.e., the number of variables) is fixed. For $n = 1$, it is easy to come up with an efficient solution. However, even for $n = 2$, this is no longer obvious. In the next section, we describe the celebrated algorithm by Lenstra that solves IP in polynomial time for any fixed dimension n.

[1] In fact, one also has to show that the size of the witness is polynomial in the input size. We omit the details.

2 Lenstra's Algorithm

In this section, we present a solution to a generalization of the integer programming problem. Informally, it can be stated as follows:

$$\text{Given a convex body } \mathcal{K} \subseteq \mathbb{R}^n, \text{ find a point in } \mathcal{K} \cap \mathbb{Z}^n \text{ or decide that this set is empty.} \quad (1)$$

Recall that a set $\mathcal{K}$ is convex if for all $x, y \in \mathcal{K}$ and any $\alpha \in [0,1]$, $\alpha x + (1-\alpha)y \in \mathcal{K}$. By a *convex body* we mean a convex bounded set with non-empty interior. We can assume without loss of generality that a set specified by a list of linear inequalities is a convex body[2]; hence, the above problem is indeed a generalization of the integer programming problem.

In order to define (1) formally, we need to specify how $\mathcal{K}$ is given. We could ask for a description of $\mathcal{K}$ as the set of solutions to a list of linear inequalities, or as the convex hull of a set of points. However, in order to achieve the greatest generality, we choose to use an oracle description of $\mathcal{K}$. That is, our algorithm is given access to an oracle that can answer queries like "is $x \in \mathcal{K}$?". However, for this oracle to be useful, it needs to be defined carefully. We postpone the discussion of this somewhat technical issue to the next section.

The following theorem, known as John's theorem, says that any convex body can be 'approximated' by an ellipsoid.

THEOREM 2
For any convex body $\mathcal{K} \subseteq \mathbb{R}^n$ there exists a pair (E', E) of concentric ellipsoids such that $E' \subseteq \mathcal{K} \subseteq E$ and E is obtained by expanding E' by a factor of n. Such a pair is known as a Löwner-John pair *for $\mathcal{K}$.*

The proof defines E' as the ellipsoid of maximum volume contained in $\mathcal{K}$. It is then shown that the ellipsoid obtained by expanding E' by a factor of n contains $\mathcal{K}$.

For this theorem to be useful for us, we need to have an efficient way to come up with such a pair. Luckily, there exists an efficient algorithm that finds a *weak* Löwner-John pair, where such a pair is defined as above with the factor n replaced by $(n+1)\sqrt{n}$.

We can now describe Lenstra'a algorithm. The first step in the algorithm is to compute a weak Löwner-John pair for $\mathcal{K}$. Using this pair, we can compute a linear transformation T that transforms the two concentric ellipsoids into two concentric balls. So now we have that $B(p,r) \subseteq T\mathcal{K} \subseteq B(p,R)$ for some point p and radii r, R with $\frac{R}{r} = (n+1)\sqrt{n}$. In other words, $T\mathcal{K}$ is 'ball-like'.

Our goal can be equivalently stated as: decide if $T\mathbb{Z}^n \cap T\mathcal{K}$ is empty. Notice that the set $T\mathbb{Z}^n$ is exactly the lattice whose basis is given by the columns of T, i.e., $\mathcal{L}(T)$. We apply the LLL algorithm to T and obtain some basis $b_1, \ldots, b_n$ for $\mathcal{L}(T)$. Equipped with this LLL reduced basis, we now show how to determine whether $\mathcal{L}(T) \cap T\mathcal{K}$ is empty.

Let k be such that $\|b_k\|$ is maximal. According to Homework 2, Question 2, for any $x \in \mathbb{R}^n$ we can find $y \in \mathcal{L}(T)$ such that

$$\|x - y\| \leq \frac{1}{2}(\|b_1\|^2 + \ldots + \|b_n\|^2)^{1/2} < n \cdot \|b_k\|.$$

By applying this procedure to p, we obtain a point $y \in \mathcal{L}(T)$ such that $\|y - p\| \leq n \cdot \|b_k\|$. If this y happens to be inside $T\mathcal{K}$ then we found a point in the intersection and we are done. Otherwise, y is also not in $B(p,r)$ and hence $\|y - p\| > r$. Therefore, $\|b_k\| > r/n$. This means that $\|b_k\|$ is not much smaller than r. In the following, we proceed by partitioning the lattice according to b_k and solving each subproblem recursively.

Let $\mathcal{L}'$ be the lattice spanned by $\{b_1, \ldots, b_{k-1}, b_{k+1}, \ldots, b_n\}$. Then $\mathcal{L}(T)$ can be written as $\bigcup_{i \in \mathbb{Z}} (\mathcal{L}' + i \cdot b_k)$. Homework 2, Question 4(g) says that in an LLL reduced basis, each vector rises a bit above the hyperplane

[2]This requires some more effort to make completely rigorous.

spanned by the other $n-1$ vectors. More precisely, it says that the distance between two successive hyperplanes is

$$\mathrm{dist}(b_k, \mathrm{span}(\mathcal{L}')) \geq 2^{-n(n-1)/4} \cdot \|b_k\| \geq 2^{-n(n-1)/4} \cdot r/n.$$

Hence, the number of hyperplanes $\mathrm{span}(\mathcal{L}') + i \cdot b_k$ that intersect $B(p, R)$ is at most

$$n 2^{n(n-1)/4} \cdot \frac{R}{r} < n^3 2^{n(n-1)/4}.$$

This is a constant that depends only on the dimension n. Hence, we can recurse on each of the resulting $n-1$ dimensional problems. More precisely, for each i in the appropriate range we consider the problem given by the intersection of the hyperplane $\mathrm{span}(\mathcal{L}') + i \cdot b_k$ with the body $T\mathcal{K}$ and the sublattice of $\mathcal{L}(T)$ that is contained in that hyperplane. After a linear transformation, this becomes an $n-1$ dimensional instance of our problem so we can recurse.

The depth of the recursion tree is n, so the total running time is $(n^3 2^{n(n-1)/4})^n$ times some polynomial in the input size. For any constant n this is polynomial in the input size, as required.

REMARK 3 There are improvements of this algorithm that achieve better running times (but with similar asymptotic running time).

3 How to describe convex bodies

It turns out that in order to perform interesting tasks on convex bodies (such as computing a weak Löwner-John pair, approximating volume, etc.) one needs an oracle that does more than simply answer if a point x is in the body. An oracle that is sufficient for most interesting tasks is what is known as a *well-guaranteed weak-separation oracle*, defined next.

By well-guaranteed, we mean that in addition to the oracle, our algorithm is given as input two numbers $r, R \in \mathbb{Q}$ such that $\mathcal{K}$ is contained in $B(0, R)$ and contains some ball of radius r. Intuitively, the purpose of this is to give the algorithm some sense of the scale of the body with which it is dealing. Moreover, since the numbers r and R are part of the input, the running time of our algorithm is allowed to be polynomial in the bit-size of their representation.

A *separation oracle* answers queries of them form "is $y \in \mathcal{K}$" for any $y \in \mathbb{Q}^n$. In case the answer is NO, the oracle returns a hyperplane that separates y from the body (i.e., y is on one side of the hyperplane and $\mathcal{K}$ is on the other side). This requirement is often too strong and is difficult to implement due to precision issue. Therefore, we settle for a slightly weaker oracle that is called a *weak separation oracle*. Such an oracle is given as input a query point $y \in \mathbb{Q}^n$ and a precision parameter $\varepsilon > 0$. Its output is the same as that of a separation oracle except that it can answer for any point y' that is within distance ε of y. Hence, when the query point y is very close to the boundary of $\mathcal{K}$, both YES and NO answers are valid (and a NO answer should be accompanied by a hyperplane that almost separates y from the body).

To summarize, a more formal way to define the problem we solved in the previous section is:

Given a convex body $\mathcal{K}$ by a well-guaranteed weak-separation oracle, find a point in $\mathcal{K} \cap \mathbb{Z}^n$ or say that it is empty in polynomial time for fixed n

References

[1] Karen Aardal. Lattice basis reduction and integer programming. Technical Report UU-CS-1999-37, Universiteit Utrecht, 1999.

[2] Lovász, L. *An Algorithmic Theory of Number, Graphs and Convexity*. SIAM Publications, 1986.