 A. Hulpke

The LLL algorithm and Applications

A *lattice* is a $\mathbb{Z}$-module (that is the $\mathbb{Z}$-span of a finite set of vectors) contained in $\mathbb{C}^n$. We also assume that an inner (hermitian) product $(\cdot,\cdot)$ has been defined on $\mathbb{C}^n$ and will consider norms with respect to this inner product. Often, when given a lattice, the problem is to find short vectors in this lattice. In general finding the guaranteed shortest vectors is hard, but surprisingly it is possible to find a good approximation in polynomial time.

The LLL Algorithm (Lenstra, Lenstra and Lovász) often solves this problem by computing, in polynomial time, a basis of short (but not guaranteed shortest) vectors.

LLL Algorithm

For a basis $\{\mathbf{b}_i\}$ of $\mathbb{R}^n$ let $\{\mathbf{b}^*{}_i\}$ be the corresponding Gram-Schmidt orthogonal basis (GSO), i.e.

$$\mathbf{b}^*{}_i = \mathbf{b}_i - \sum_{1\le j<i} \mu_{i,j}\mathbf{b}^*{}_j \qquad \text{where} \qquad \mu_{i,j} = \frac{\left(\mathbf{b}_i, \mathbf{b}^*{}_j\right)}{\left(\mathbf{b}^*{}_j, \mathbf{b}^*{}_j\right)}.$$

Lemma: Let $L \subseteq \mathbb{R}^n$ with basis $\{\mathbf{b}_i\}$. Then for any $f \in L \setminus \{0\}$ we have that

$$\|f\| \ge \min\{\|\mathbf{b}^*{}_1\|, \|\mathbf{b}^*{}_2\|, \ldots, \|\mathbf{b}^*{}_m\|\}.$$

Definition: We say that $\{\mathbf{b}_i\}$ is *reduced*, if $\|\mathbf{b}^*{}_i\|^2 \le 2\|\mathbf{b}^*{}_{i+1}\|^2$ for $1 \le i < n$.

Theorem: Let $\{\mathbf{b}_i\}$ be a reduced basis of the lattice L and $f \in L \setminus \{0\}$. Then $\|\mathbf{b}^*{}_1\| \le 2^{(n-1)/2}\|\mathbf{f}\|$

Algorithm: For a given basis, the following procedure computes a reduced basis:

while $i \le n$ **do**
  for $j \in \{i-1, i-2, \ldots, 1$ **do**
    $\mathbf{b}_i := \mathbf{b}_i - \lceil \mu_{i,j} \rfloor \cdot \mathbf{b}^*{}_j$. Update the GSO. {Replacement Step}
    if $i > 1$ and $\|\mathbf{b}^*{}_{i-1}\|^2 > \|\mathbf{b}^*{}_i\|^2$ **then**
      Exchange $\mathbf{b}_{i-1}$ and $\mathbf{b}_i$. Update the GSO. {Swap Step}
      $i := i - 1$;
    else
      $i := i + 1$;
    fi;
  od;
od;

The "full" LLL algorithm includes some improvements that produce better reduction (in part guideable with a parameter $1/4 < a < 1$), update the Gram-Schmidt parameters implicitly, and never need to write out the $\mathbf{b}^*{}_i$ explicitly. It takes $O(n^4 \log A)$, with $A = \max_i \|\mathbf{b}_i\}$, operations on scalars.

The LLL algorithm produes in polynomial time a vector whose length is bounded by a factor from the shortest possible.

Applications

A typical example of the powers of the LLL algorithm is the "subset sum" problem. We are given a set of positive integers, and want to express a particular number as combination (with coefficients 0 or 1) of these numbers if possible.

In general this problem is hard (NP-complete) and variations had been proposed as basis for public-key cryptosystems.

We want to rewrite the problem as a lattice problem: If n numbers $a_1, \ldots, a_n \in \mathbb{N}$ are given, form n vectors in $\mathbb{Q}^{n+1}$: $(1, 0, \ldots, 0, -a_1)$, $(0, 1, 0, \ldots, 0, -a_2)$, $(0, \ldots, 1, 0, -a_{n-1})$ and $(0, \ldots, 0, 1, -a_n)$. Also take the vector $(0, \ldots, 0, 0, s)$, where s is the number you want to express.

Assuming the a_i and s are large, a short element in the lattice spanned by these vectors must have a 0 in the last component. This can only be achieved by summing up some of the a_i to s. The entries of 1 in the corresponding vectors indicate which numbers are added.

For example, suppose we want to express $s = 1215$ as a combination of $\{366, 385, 392, 401, 422, 437\}$. We first set up the vectors:

```
gap> nums:=[366,385,392,401,422,437];;
gap> mat:=IdentityMat(Length(nums)+1);;
gap> for i in [1..6] do mat[i][7]:=-nums[i];od;
gap> mat[7][7]:=1215;;
gap> Display(mat);
[ [     1,     0,     0,     0,     0,     0,  -366 ],
  [     0,     1,     0,     0,     0,     0,  -385 ],
  [     0,     0,     1,     0,     0,     0,  -392 ],
  [     0,     0,     0,     1,     0,     0,  -401 ],
  [     0,     0,     0,     0,     1,     0,  -422 ],
  [     0,     0,     0,     0,     0,     1,  -437 ],
  [     0,     0,     0,     0,     0,     0,  1215 ] ]
```

Next we call the LLL for the standard inner product. We only care about the "basis" component:

```
gap> LLLReducedBasis(mat);
rec(basis:=[ [ 0, 0, 1, 1, 1, 0, 0 ], [ 0, 1, 1, 0, 0, 1, 1 ],
             [ 1, 0, 1, -1, 1, 1, -1 ], ...
```

The first vector (always the shortest) has a 0 in the last component and entries in position 3,4 and 5. Indeed the three numbers at these positions add up to 1215.

Note that the second vector has a 1 in the last component, i.e. we combine to "almost" 1215. If we want to eliminate such flukes (which easily could become short) we can either change the inner product to weight the last component more, or simply scale all the numbers up.

```
gap> for i in [1..7] do mat[i][7]:=mat[i][7]*1000;od;Display(mat);
[ [ 1, 0, 0, 0, 0, 0, -366000 ], ...
gap> b:=LLLReducedBasis(mat);
rec(basis:=[ [ 0, 0, 1, 1, 1, 0, 0 ], [ 2, -1, -1, 1, 1, 1, 0 ],
       [ 1, -2, 2, -2, 1, 0, 0 ], [ 1, 1, 1, -2, 0, 2, 0 ],
```

In general, however, as the LLL algorithm is not guaranteed to find **the** shortest vectors (this could be done by an exhaustive search of combinations), it is possible that it produces combinations of s with coefficients different from 1.

Nevertheless, this "often good" performance makes the subset sum scheme infeasible for cryptographic purposes.

Subset-Anagrams

For another example of a combination-type problem, suppose we have a (long) list of words. We want to find combinations of different words, that use the same letters[1]. Again one can solve this by exhaustive search.

For another solution, form a $n \times 26$ matrix M, each row corresponding to one word. The entries in the row count how often the letter occurs in this word.

```
gap> Read("deptnames.g");
gap> deptnames{[1..3]};
[ "jeff achter", "henry adams", "adam afandi" ]
gap> List(CHARS_LALPHA,i->Number(deptnames[1],j->j=i));
[ 1,0,1,0,2,2,0,1,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0 ]
gap> nam:=List(deptnames,n->List(CHARS_LALPHA,i->Number(n,j->j=i)));;
```

Two different combinations of words then correspond to a (row) vector $\mathbf{x} \in \{-1, 0, 1\}^n$ such that $\mathbf{x}M = \mathbf{0}$. (The 1's are one word, the -1's another.)

We thus are searching for $-1/0/1$-solutions to a system of equations.

Many combinatorial problems can be phrased this way!

To find such solutions, we want to find the $\mathbb{Z}$-nullspace of M, i.e. the vectors $\mathbf{v} \in \mathbb{Z}^n$ such that $\mathbf{v}M = \mathbf{0}$. These vectors form a lattice. Chances are good, that a short vector in this lattice will be a $-1/0/1$ solution.

[1]This was for example a weekly puzzle on NPR's "Weekend edition" on May 5, 2002: *Find two countries, such that the letters in their names can be rearranged to spell two other countries; for example: MALI + QATAR = IRAQ + MALTA*

To find the $\mathbb{Z}$-nullspace, ordinary Gauß-elimination will not work, as it produces rational solution. Multiplying out all denominators might create only a multiple of the lattice.

Instead we will use the Smith normal form: Write $PMQ = D$ with $P \in \mathbb{Z}^{n\times n}$, $Q \in \mathbb{Z}^{26\times 26}$ both invertible and $D \in \mathbb{Z}^{n\times 26}$ diagonal. Then find a $\mathbb{Z}$-basis for the $\mathbb{Z}$-nullspace of D. (This is very easy because of the diagonal form.) If $\mathbf{x}D = \mathbf{0}$ then $\mathbf{x}PMQ = \mathbf{0}$, thus $\mathbf{x}P \in N(M)$.

As P is invertible, it is easily seen that we will get a $\mathbb{Z}$-basis for $N(M)$ from a $\mathbb{Z}$-basis for $N(D)$.

To calculate this we use the online help, which tells us that the output to `SmithNormalFormIntegerMatTransf` contains a component `normal` which will be D and `rowtrans` which will be P. We also note that `NullspaceMat` finds a $\mathbb{Z}$-basis in this particular case, as D is diagonal.

```
gap> snf:=SmithNormalFormIntegerMatTransforms(nam);
gap> mat:=NullspaceMat(snf.normal);
gap> Set(Flat(mat));
[ 0, 1 ]
gap> mat:=mat*snf.rowtrans;;
gap> Length(Set(Flat(mat)));
2083
```

Now we perform LLL reduction on this basis. We are only intrested in those vectors in the solution, whose entries are $-1, 0, 1$. We sort the solution by to the number of entries.

```
gap> red:=LLLReducedBasis(mat,1);; # 1 should be 1-epsilon
gap> sol:=Filtered(red.basis,i->IsSubset([-1,0,1],Set(i)));;
gap> Sort(sol,function(a,b) return a*a<b*b;end);
```

Let us finally verify the first solution:

```
gap> sol[1];
[ 0, -1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 1, -1, -1, 0, 0, 1,
[...]
gap> Filtered([1..106],i->sol[1][i]=-1);
[ 2, 13, 20, 21, 32, 35, 46, 47 ]
gap> deptnames{last};
[ "henry adams", "vance blankers", "michael capps", "renzo cavalieri",
  "bryan elder", "tegan emerson", "jianli gu", "derek handwerk" ]
gap> Collected(Flat(last));
[ [ ' ', 8 ], [ 'a', 12 ], [ 'b', 2 ], [ 'c', 4 ], [ 'd', 4 ], [ 'e', 14 ],
[...]
gap> Filtered([1..106],i->sol[1][i]=1);
[ 5, 8, 9, 19, 24, 36, 55, 86 ]
[ "javier alvarez", "dan bates", "ryan becker", "karleigh cameron",
  "edwin chong", "melissa erdmann", "paul kennedy", "rachel pries" ]
gap> Collected(Flat(last));
[ [ ' ', 8 ], [ 'a', 12 ], [ 'b', 2 ], [ 'c', 4 ], [ 'd', 4 ], [ 'e', 14 ],
[...]
```