LLL

Trying to prove it on stainless

DUPARC Max & SERANDOUR Erwan

CS-550

December 21, 2022

Outline

LLL Algorithm

Formally verifying the LLL

Yet to prove

Conclusion

LLL Algorithm

Gram Schmidt

Algorithm 1 Gram-Schmidt

Require: $v_0, \cdots, v_{n-1}$

Ensure: $span\{v_0, \cdots, v_{n-1}\} = span\{g_0, \cdots, g_{n-1}\}$ and for $i \neq j$, $\langle g_i, g_j \rangle = 0$

```
g_0 := v_0
for i = 1 to n do
    g_i := b_i
    for j = 0 to i − 1 do
        μ_{i,j} := ⟨v_i, g_j⟩/⟨g_j, g_j⟩
        g_i := g_i − μ_{i,j} g_j
    end for
end for
return g_0, ··· , g_n.
```

α-reduced basis

Definition (α-reduced basis)

Given $\mathbf{B} = \{f_1, \cdots, f_n\}$ such that $f_i \in \mathbb{Z}^m$ and Gram Schmidt basis $\mathbf{G} = \{g_1, \cdots, g_n\}$. We say that it is α-reduced if:

- (size reduced) $1 \leq j < i \leq n$: $|\mu_{i,j}| \leq 1/2$, with $\mu_{i,j}$ defined earlier
- (Lovàsz condition) For $k = 1, ..., n-1 : ||g_{k-1}||^2 \leq \alpha ||g_k||^2$ where $\alpha \geq 1$.

LLL

Algorithm 2 The LLL Basis reduction algorithm

Require: A list of linearly independent vectors $f_0,\ldots,f_{n-1} \in \mathbb{Z}^m$ and $\alpha > \frac{4}{3}$

Ensure: a basis for the same lattice as $f_0,\ldots,f_{n-1}$ that is reduced w.r.t α

$i := 0$
$g_0,\cdots,g_{n-1} :=$ Gram-Schmidt$(f_0,\ldots,f_{n-1})$
while $i < n$ **do**
    for $j = i-1$ **downto** 0 **do**
        $\mu_{i,j} := \langle f_i, g_j\rangle / \langle g_j, g_j\rangle$
        $f_i := f_i - \lceil \mu_{i,j} \rfloor f_j$
    end for
    if $i = 0$ **or** $||g_{i-1}||^2 \leq \alpha ||g_i||^2$ **then**
        $i := i+1$
    else
        $h := g_{i-1}$
        $g_{i-1} := g_i + \mu_{i,i-1} g_{i-1}$
        $\mu := \langle h, g_{i-1}\rangle / \langle g_{i-1}, g_{i-1}\rangle$
        $g_i := h - \mu g_{i-1}$
        $(i, f_{i-1}, f_i) := (i-1, f_i, f_{i-1})$
    end if
end while
return $f_0,\ldots,f_{n-1}$.

Example

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1048576 & 3294200 & 10349040 & 1724840 \end{bmatrix} \xrightarrow{LLL} \begin{bmatrix} 0 & -59 & -43 & -97 \\ 0 & -54 & 65 & 78 \\ -1 & 23 & -16 & -15 \\ 6 & 1 & -2 & 0 \\ 0 & -24 & -88 & 128 \end{bmatrix}$$

Formally verifying the LLL

The properties, we want to verify

- Lattice preservation
- Lovàsz condition : For $k = 1, ..., n-1 : ||g_{k-1}||^2 \leq \alpha ||g_k||^2$ where $\alpha \geq 1$. This means that vectors in the basis are not much shorter than the previous ones.
- Nearly orthogonal : $1 \leq j < i \leq n : |\mu_{i,j}| \leq 1/2$. This means that f_i is component-wise the closest vector in the lattice to g_i, meaning that it is nearly orthogonal.

But we will have to assume some linear algebra properties, otherwise it will be too complicated.

Lattice preservation

```scala
case class Lattice(rows : BigInt, cols : BigInt, vectors : List[IntegralVector], equivalentLattice : List[IntegralVector]){
    require(rows>0 && cols>0)
    require(vectors.size==rows)
    require(vectors.forall(v => v.size==cols))
    require(lin_indpt(rows, cols, vectors.map(v => v.toVector())))

    def sum_rows(bjIdx : BigInt, alpha : BigInt, biIdx : BigInt):Lattice = {…
    }.ensuring(res => res.rows==this.rows && res.cols==this.cols)

    def swap(x:BigInt, y:BigInt):Lattice = {…
    }.ensuring(res => res.rows==this.rows && res.cols==this.cols)

}
```

Lovàsz condition

```
def lovaz_condition(vectors : List[Vector], until : BigInt, alpha : Rational, size : BigInt):Boolean={
  decreases(max(until, 0))

  if(!(until<=vectors.size && until>=1) || !(vectors.forall(v => v.size==size))){
    return false
  }

  if(until==1){
    true
  }else{
    val g_k = vectors.head
    val g_k_1 = vectors.tail.head
    g_k.norm() <= alpha*g_k_1.norm() &&  lovaz_condition(vectors.tail, until-1, alpha, size)
  }
}
```

Nearly orthogonal

```scala
def nearly_orthogonal(lattice : List[IntegralVector], gramBasis : List[Vector], i : BigInt, j : BigInt, size : BigInt):Boolean = {
  decreases(max(0, i*i-j))

  if(i>lattice.size || i<1 || j<1 || j>i || (i==lattice.size && i!=j)){
    return false
  }

  if(gramBasis.size!=lattice.size || size<1 || !(lattice.forall(v => v.size==size)) || !(gramBasis.forall(v => v.size==size)) || !(gramBasis.forall(v => v.nonZero())))
    return false
  }

  if(i==1 && j==1){
    true
  }else if(j<i){
    val gramVect = get(j, gramBasis)
    conserveNorm(j, gramBasis, size)
    conserveSize(j, gramBasis, size)
    conserveSizeIntegral(i, lattice, size)

    val mu = lattice(i).dotProduct(gramVect)/gramVect.norm()
    abs(mu)<=Rational(1,2) && nearly_orthogonal(lattice, gramBasis, i, j+1, size)
  }else{
    assert(i==j && i>1)
    val gramVect = get(0, gramBasis)
    conserveNorm(0, gramBasis, size)
    conserveSize(0, gramBasis, size)
    conserveSizeIntegral(i-1, lattice, size)
    val mu = lattice(i-1).dotProduct(gramVect)/gramVect.norm()
    abs(mu)<=Rational(1,2) && nearly_orthogonal(lattice, gramBasis, i-1, 1, size)
  }
}
```

The main Loop

```scala
def reduced(lattice : Lattice, alpha : Rational, index : BigInt): Boolean =
  (index==0 ||
    (lovaz_condition(lattice.gramSchmidt_basis(), index, alpha, lattice.cols)
    && nearly_orthogonal(lattice.vectors, lattice.gramSchmidt_basis(), index, lattice.cols)))

def LLL(lattice : Lattice, alpha : Rational): Lattice = {
  require(alpha > Rational(4,3))

  def LLLloop(index : BigInt, lattice : Lattice): Lattice = {
    require(index>=0 && index<=lattice.rows)
    require(reduced(lattice, alpha, index))
    if(index==lattice.rows){
      lattice
    }else if(index==0){
      LLLloop(index+1, lattice)
    }else{
      val aproxOrtho = LLLAprox(index, index, lattice, lattice.gramSchmidt_basis())
      if(lovaz_condition(aproxOrtho.gramSchmidt_basis(), index+1, alpha,  aproxOrtho.cols)){
        LLLloop(index+1, aproxOrtho)
      }else{
        val swapped = swap(index, lattice)
        assert(reduced(swapped, alpha, index-1))
        LLLloop(index-1, swapped)
      }
    }
  }.ensuring(res => res.sameLattice(lattice) &&  reduced(res, alpha, res.rows))
  LLLloop(0, lattice)
}.ensuring(res => res.sameLattice(lattice) &&  reduced(res, alpha, res.rows))
```

Proving swap branch respects invariant

```scala
def swap(index : BigInt, lattice : Lattice): Lattice = {
  require(index>=1 && index<lattice.rows)
  require(reduced(lattice, alpha, index))
  val swapped =  lattice.swap(index-1,index)

  if(index==1){
    swapped
  }else{
    lovaz_condition_take(lattice.gramSchmidt_basis(), index, index-1, alpha, lattice.cols)
    lattice.swap_conserve_gramschmit(index-1, index)
    lovaz_condition_append(swapped.gramSchmidt_basis(), index-1, alpha, swapped.cols)


    nearly_orthogonal_subset(lattice.vectors, lattice.gramSchmidt_basis(), lattice.cols, index, index, index-1, index-1)
    nearly_orthogonal_take(lattice.vectors, lattice.gramSchmidt_basis(), lattice.cols, index, index-1, index-1)
    lattice.swap_conserve_lattice(index-1, index)
    nearly_orthogonal_append(swapped.vectors, swapped.gramSchmidt_basis(), swapped.cols, index-1, index-1, index-1)

    swapped
  }

}.ensuring(res => res.sameLattice(lattice) && reduced(res, alpha, index-1) )

//axioms
def swap_conserve_gramschmit(x:BigInt, y:BigInt): Unit = {
  require(x>=0 && x<this.rows)
  require(y>=0 && y<this.rows)
  require(x<y)

}.ensuring(_ => take_vector(x, this.gramSchmidt_basis(), cols)==take_vector(x, this.swap(x,y).gramSchmidt_basis(), cols))



def swap_conserve_lattice(x:BigInt, y:BigInt): Unit = {
  require(x>=0 && x<this.rows)
  require(y>=0 && y<this.rows)
  require(x<y)
}.ensuring(_ => take_integral(x, this.vectors, cols)==take_integral(x, this.swap(x,y).vectors, cols))
```

Proving LLLAprox preserve invariant

```
def LLLAprox(i : BigInt, j: BigInt, lattice : Lattice, gramBasis : List[Vector]):Lattice = {
  decreases(j)
  require(i>=1 && i<lattice.rows && j<=i && j>=0)
  require(lattice.rows==gramBasis.size && gramBasis.forall(v => v.size==lattice.cols) && gramBasis.forall(v => v.nonZero()))
  require(lovaz_condition(lattice.gramSchmidt_basis(), i, alpha, lattice.cols))
  require(nearly_orthogonal(lattice.vectors, gramBasis, i_(i, j), j_(i, j), lattice.cols))


  if(j==0){
    lattice
  }else{
    val g = get(j-1, gramBasis)
    conserveSize(j-1, gramBasis, lattice.cols)
    conserveNorm(j-1, gramBasis, lattice.cols)
    val mu = lattice(i).dotProduct(g)/g.norm()
    lattice.sum_conserve_gramschmit_basis(j-1, -round(mu), i)
    val updated_lattice = lattice.sum_rows(j-1, -round(mu), i)
    assert(abs(upadted_mu)<=Rational(1,2))
    //axiom :: sum_preserve_orthogonality under j-1<j_(i,j)
    assert(nearly_orthogonal(updated_lattice.vectors, gramBasis, i_(i, j), j_(i, j), lattice.cols))
    LLLAprox(i, j-1, updated_lattice, gramBasis)
  }
}.ensuring(res => res.rows==lattice.rows && res.cols==lattice.cols && res.sameLattice(lattice) && lovaz_condition(res.gramSchmidt_basis(), i, alpha, res.cols)
&& nearly_orthogonal(res.vectors, gramBasis, i+1, i+1, res.cols))

def i_(i : BigInt, j: BigInt): BigInt = if(j==0){i+1}else{i}
def j_(i : BigInt, j: BigInt): BigInt = if(j==0){i+1}else{j}

def sum_conserve_gramschmit_basis(bjIdx : BigInt, alpha : BigInt, biIdx : BigInt):Unit={
  require(bjIdx<biIdx)
}.ensuring(_ => this.sum_rows(bjIdx, alpha, biIdx).gramSchmidt_basis()==this.gramSchmidt_basis())
```

Yet to prove

Measure[1]

First, given B a basis of vectors in $\mathbb{Z}^m$ and induced Gram-Schmidt basis G, we have that

$$\delta(G) = \delta(B) = \det(B^\top B) = \prod_{i \in n} \|g_i\|^2 \in \mathbb{N}$$

$$\Delta\big(G\big) = \Delta\big(B\big) = \prod_{i=0}^{n-1} \delta(B_i) = \prod_{i=0}^{n-1} \prod_{j=0}^{i-1} \|g_j\|^2 = \prod_{i=0}^{n-1} \|g_i\|^{2(n-i)} \in \mathbb{N}$$

Measure[2]

In our project, the *LLL* is handled by the function

LLLloop(index : BigInt, lattice : Lattice)
gramBasis $\leftarrow GramSchmidt(lattice)$

The measure that decrease, proving that the LLL finishes is given by

decreases(Δ(gramBasis), n-index)

Measure[3]

Why does this decreases ?

- If we do not switch, then gramBasis remains unchanged and Index is increased by 1.
- If we switch, then the gramBasis is updated as follows. Given g_i, g_{i+1}, we have that:

$$g_i^* = g_{i+1} + \mu_{i+1,i} g_i$$

$$g_{i+1}^* = g_i - \langle g_i, g_i^* \rangle / \langle g_i^*, g_i^* \rangle g_i^*$$

$$\text{gramBasis}^* \leftarrow \text{gramBasis}[: i-1] ++ [g_i^*, g_{i+1}^*] ++ \text{gramBasis}[i+2 :]$$

Measure[4]

As δ(gramBasis*)=δ(gramBasis), We have that

$$\|g_i^*\|^2\|g_{i+1}^*\|^2 = \|g_i\|^2\|g_{i+1}\|^2$$

Furthermore,

$$\begin{aligned}
&\|g_i\|^2 > \alpha\|g_{i+1}\|^2 \\
\iff &\alpha^{-1}\|g_i\|^2 > \|g_{i+1}\|^2 \\
\iff &\alpha^{-1}\|g_i\|^2 + \mu_{i+1,i}^2\|g_i\|^2 > \|g_{i+1}\|^2 + \mu_{i+1,i}^2\|g_i\|^2 \\
\iff &(\alpha^{-1} + \frac{1}{4})\|g_i\|^2 > \|g_{i+1} + \mu_{i+1,i}g_i\|^2 \\
\iff &\|g_i\|^2 > \|g_i^*\|^2
\end{aligned}$$

$$\|g_i^*\|^4\|g_{i+1}^*\|^2 < (\alpha^{-1} + \frac{1}{4})\|g_i\|^4\|g_{i+1}\|^2$$

$$\Delta(\text{gramBasis}^*) < (\alpha^{-1} + \frac{1}{4})\Delta(\text{gramBasis})$$

Conclusion

We have been a bit too zealous with the whole LLL:

- Based on many linear algebra properties, so we could not do it from the ground up.
- We have significantly improved our stainless understanding.

Thank you for your attention !

* for references, please refer to our report