Babai's Nearest Plane Algorithm

Eric Ren, Sage Binder, and Katherine Kosaian

March 17, 2025

Abstract

γ-CVP is the problem of finding a vector in L that is within γ times the closest possible to t, where L is a lattice and t is a target vector. If the basis for L is LLL-reduced, Babai's Closest Hyperplane algorithm solves γ-CVP for $\gamma = 2^{n/2}$, where n is the dimension of the lattice L, in time polynomial in n. This session formalizes said algorithm, using the AFP formalization of LLL [2, 1] and adapting a proof of correctness from the lecture notes of Stephens-Davidowitz [4].

Contents

1 Introduction

The (exact) *closest vector problem* (CVP) is the problem of finding the closest vector within a lattice L to a target vector t. This is equivalent to finding the shortest vector in the *lattice coset* $L - t := \{l - t : l \in L\}$. There is a corresponding family of weaker problems, γ-CVP (where γ is some real parameter), where one needs only find a vector in $L - t$ whose length is at most γ times the shortest possible. Through a reduction to the *shortest* *vector problem* [4], solutions to these problems may be used to factor rational polynomials. This problem is therefore of cryptographic interest.

Although exact CVP (or 1-CVP) is NP-Complete [3], Babai's Nearest Plane Algorithm solves $2^{n/2}$-CVP, where n is the dimension of L, in polynomial time, provided that L is presented using an LLL-reduced basis with parameter $\alpha = 4/3$. The proof in this document is mostly a straightforward algebraicization of the proof in Stephens-Davidowitz' lecture notes. It makes use of the coordinate systems defined by the original basis (denoted β) and the Gram-Schmidt orthogonalization of that basis (denoted $\tilde{\beta}$). Let $[u]_\beta$ denote the representation of a vector u under β, with coordinates $[u]_\beta^j$; $j = 1, ..., n$ (likewise for $\tilde{\beta}$). Also, let s_i denote the output of the algorithm after step i and let d be the shortest lattice coset vector, as witnessed by the vector v. The proof works by analysing the coordinates of $[s_n]_{\tilde{\beta}}$, showing that all are at most $1/2$ and that some later coordinates are exactly those of $[v]_{\tilde{\beta}}$.

The algorithm modifies coordinate $n-i$ in both bases for the last time in step i (formalized in lemma `coord_invariance`), during which both coordinates are decreased below $1/2$ (formalized in lemma `small_coord`). Combined, these facts imply that the output s_n has $\left|[s_n]_{\tilde{\beta}}^j\right| \leq 1/2$ for all indices j.

Since $\tilde{\beta}$ is orthogonal, we have

$$\|s_n\|^2 = \sum_{i=1}^{n} \left([s_n]_{\tilde{\beta}}^i \|\tilde{\beta}_i\|\right)^2, \tag{1}$$

so the preceding coordinate bounds $\|s_n\|^2$ by $\frac{1}{4}\sum_{i=1}^{n} \|\tilde{\beta}_i\|^2$. If the $\tilde{\beta}_i$ are all short compared to d, this bound suffices. In fact, if there is any short vector $\tilde{\beta}_I$ in $\tilde{\beta}$ then because β is LLL-reduced, any vector preceding $\tilde{\beta}_I$ in $\tilde{\beta}$ will not be much longer. This bounds the first I terms in Equation 1. By selecting I maximal, we may assume that $\tilde{\beta}$ ends in a series of $n - I$ long vectors. In this case it can be shown $[v]_{\tilde{\beta}}^j$ and $[s_n]_{\tilde{\beta}}^j$ differ by an integral amount for $j = I+1, ..., n$. Therefore, if $[v]_{\tilde{\beta}}^j$ and $[s_n]_{\tilde{\beta}}^j$ differ at all, they differ by at least 1, which would mean $\left|[v]_{\tilde{\beta}}^j\right| \geq 1/2$, since $\left|[s_n]_{\tilde{\beta}}^j\right| \leq 1/2$. This would force v to be longer than d, a contradiction. So $[v]_{\tilde{\beta}}^j = [s_n]_{\tilde{\beta}}^j$ for $j = I + 1, ...n$, which gives a tighter bound on the last $n - I$ terms in equation 1.

Precisely, let I denote $\max\{i : \|\tilde{\beta}_i\| \leq 2d\}$, meaning for all indices $j > I$, $\|\tilde{\beta}_j\| > 2d$. Now, for all $j > I$, $d^2 = \|v\| \geq ([v]_{\tilde{\beta}}^j)^2 \|\tilde{\beta}_j\|^2 > ([v]_{\tilde{\beta}}^j)^2 \cdot 4d^2$, meaning $1/4 > (\tilde{\beta}^j)^2$, or $1/2 > \left|[v]_{\tilde{\beta}}^j\right|$. Since $\left|[s_j]_{\tilde{\beta}}^j\right| \leq 1/2$ from the previous section, $\left|[v]_{\tilde{\beta}}^j - [s_j]_{\tilde{\beta}}^j\right| < 1$. Using properties of the change-of-basis between $\beta, \tilde{\beta}$ formalized in the LLL AFP session, we show that $[v]_{\tilde{\beta}}^j - [s_j]_{\tilde{\beta}}^j =$

$[v]^j_\beta - [s_j]^j_\beta = [v - s_j]^j_\beta$, so that $\left|[v - s_j]^j_\beta\right| < 1$ But since $v - s_j$ lies in the lattice, $[v - s_j]^j_\beta$ is integral, so $\left|[v - s_j]^j_\beta\right| = 0$, meaning $[v]^j_{\tilde\beta} = [s_j]^j_{\tilde\beta}$. Lemma `coord_invariance` gives that $[v]^j_{\tilde\beta} = [s_j]^j_{\tilde\beta} = [s_n]^j_{\tilde\beta}$. This is formalized by lemma `correct_coord`.

Now $\|s_n\|^2 = \sum_{i=1}^{n}([s_n]^i_{\tilde\beta}\|\tilde\beta_i\|)^2$, since $\tilde\beta$ is orthogonal. Splitting the sum around I equates this to $\sum_{i=1}^{I}([s_n]^i_{\tilde\beta})^2 + \sum_{i=I+1}^{n}([s_n]^i_{\tilde\beta})^2$. Lemma `small_coord` bounds the terms in the first sum by $\|\tilde\beta_i\|^2/4$, while lemma $\texttt{correct}_\texttt{c}\texttt{oord}$ bounds the terms in the second sum by d^2, giving $\|s_n\|^2 \leq (n-I)d^2 + \sum_{i=1}^{I}\|\tilde\beta_i\|^2/4$. If β is LLL-reduced with parameter α, $\|\tilde\beta_i\|^2 \leq \alpha^I\|\tilde\beta_I\|^2$ for all $i \leq I$, which, by the definition of I, is at most $4d^2$. So $\|s_n\|^2 \leq ((n-I) + I\alpha^I)d^2 \leq n\alpha^n d^2$. The standard choice of $\alpha = 4/3$ gives $\|s_n\|^2 \leq 2^n d^2$. All of this is formalized in the final section, which culminates in the main theorem.

To avoid having to prove that a shortest vector exists, we use the definition $\inf\{\|u-t\| : u \in L\}$ for d instead of $\min\{\|u-t\| : u \in L\}$ and rephrase the arguments above to allow $\|v\|$ to exceed d by a small constant factor ϵ. This workaround and its details are contained in the section on the closest distance and negligibly change the rest of the proof.

theory *Babai-Algorithm*

imports *LLL-Basis-Reduction.LLL*
HOL.Archimedean-Field
HOL−Analysis.Inner-Product

begin
fun *calculate-c*:: *rat vec* ⇒ *rat vec list* ⇒ *nat* => *int* **where**
calculate-c s L1 n = *round* ((*s* • (*L1*!((*dim-vec s*) − *n*))) / (*sq-norm-vec* (*L1*!((*dim-vec s*) − *n*))))

fun *update-s*:: *rat vec* ⇒ *rat vec list* ⇒ *rat vec list* ⇒ *nat* ⇒ *rat vec* **where**
update-s sn M Mt n = ((*rat-of-int* (*calculate-c sn Mt n*)) $\cdot_v$ *M*!((*dim-vec sn*)−*n*))

fun *Babai-Help*:: *rat vec* ⇒ *rat vec list* ⇒ *rat vec list* ⇒ *nat* ⇒ *rat vec* **where**
Babai-Help s M Mt 0 = *s* |
Babai-Help s M Mt (*Suc n*) = (*let B*= (*Babai-Help s M Mt n*) *in B*− (*update-s B M Mt* (*Suc n*)))

definition *Babai*:: *rat vec* ⇒ *rat vec list* ⇒ *rat vec* **where**
Babai s M = *Babai-Help s M* (*gram-schmidt* (*dim-vec s*) *M*) (*dim-vec s*)

end
theory *Babai*
  imports *Babai-Algorithm*

begin

This theory contains the proof of correctness of the algorithm. The main theorem is "theorem Babai-Correct", under the locale "Babai-with-assms". To use the theorem, one needs to show that lattice, the vectors in the lattice basis, and the target vector all have the same dimension, that the lattice basis vectors are linearly independent and form an invertible matrix, and that the lattice basis is LLL-weakly-reduced.

2 Locale setup for Babai

locale *Babai* =
  fixes *M* :: *int vec list*
  fixes *t* :: *rat vec*
  assumes *length-M*: *length M* = *dim-vec t*
begin

abbreviation *n* **where** *n* ≡ *length M*
definition α **where** (α::*rat*) = *4/3*
sublocale *LLL n n M* α⟨*proof*⟩

abbreviation *coset*::*rat vec set* **where** *coset*≡{(*map-vec rat-of-int x*)−*t*|*x*. *x*∈*L*}
abbreviation *Mt* **where** *Mt* ≡ *gram-schmidt n* (*RAT M*)

definition *s* :: *nat* ⇒ *rat vec* **where**
  s i = *Babai-Help* (*uminus t*) (*RAT M*) *Mt i*

definition *closest-distance-sq*:: *real* **where**
  closest-distance-sq = *Inf* {*real-of-rat* (*sq-norm x*::*rat*) |*x*. *x* ∈ *coset*}
end

Locale setup with additional assumptions required for main theorem

locale *Babai-with-assms* = *Babai*+
  fixes *mat-M mat-M-inv*:: *rat mat*
  assumes *basis*: *lin-indep M*
  defines *mat-M* ≡ *mat-of-cols n* (*RAT M*)
  defines *mat-M-inv* ≡

(*if* (*invertible-mat mat-M*) *then SOME B*. (*inverts-mat B mat-M*) ∧ (*inverts-mat mat-M B*) *else* (0_m *n n*))
assumes *inv*:*invertible-mat mat-M*
assumes *reduced*:*weakly-reduced M n*
assumes *non-trivial*:*0*<*n*
begin

lemma *dim-vecs-in-M*:
shows ∀ *v* ∈ *set M*. *dim-vec v* = *length M*
⟨*proof*⟩

lemma *inv1*:*mat-M* ∗ *mat-M-inv* = 1_m *n*
⟨*proof*⟩

lemma *inv2*:*mat-M-inv* ∗ *mat-M* = 1_m *n*
⟨*proof*⟩

sublocale *rats*: *vec-module TYPE*(*rat*) *n*⟨*proof*⟩

lemma *M-dim*: *dim-row mat-M* = *n dim-col mat-M* = *n*
⟨*proof*⟩

lemma *M-inv-dim*: *dim-row mat-M-inv* = *n dim-col mat-M-inv* = *n*
⟨*proof*⟩

lemma *Babai-to-Help*:
shows *s n* = *Babai-Algorithm.Babai* (*uminus t*) (*RAT M*)
⟨*proof*⟩

3 Coordinates

This section sets up the use of the lattice basis and its GS orthogonalization as coordinate systems and some properties of that coordinate system. The important lemma here is coord-invariance, which shows that after step i of the algorithm, all coordinates (in both systems) after n-i are invariant.

definition *lattice-coord* :: *rat vec* ⇒ *rat vec*
where *lattice-coord a* = *mat-M-inv* $*_v$ *a*

lemma *dim-preserve-lattice-coord*:
fixes *v*::*rat vec*
assumes *dim-vec v*=*n*

shows *dim-vec* (*lattice-coord* *v*) = *n* ⟨*proof*⟩
lemma *vec-to-col*:
assumes $i < n$
shows (*RAT M*)!*i* = *col mat-M i*
⟨*proof*⟩

lemma *unit*:
assumes $i < n$
shows *lattice-coord* ((*RAT M*)!*i*) = *unit-vec n i*
⟨*proof*⟩

lemma *linear*:
fixes *i*::*nat*
fixes *v1*::*rat vec*
and *v2*:: *rat vec*
and *q*:: *rat*
assumes *dim-vec v1* = *n*
assumes *dim-2*:*dim-vec v2* = *n*
assumes $0 \leq i$
assumes *dim-i*:$i<n$
shows (*lattice-coord* (*v1*+($q \cdot_v$ *v2*)))\$*i* = (*lattice-coord v1*)\$*i* + *q*∗((*lattice-coord v2*)\$*i*)
⟨*proof*⟩

lemma *sub-s*:
fixes *i*::*nat*
assumes $0 \leq i$
assumes $i<n$
shows *s* (*Suc i*) = (*s i*) −
((*rat-of-int* (*calculate-c* (*s i*) *Mt* (*Suc i*))) $\cdot_v$ (*RAT M*)!((*dim-vec* (*s i*)) −(*Suc i*)))
⟨*proof*⟩

lemma *M-locale-1*:
shows *gram-schmidt-fs-Rn n* (*RAT M*)
⟨*proof*⟩

lemma *M-locale-2*:
shows *gram-schmidt-fs-lin-indpt n* (*RAT M*)
⟨*proof*⟩

lemma *more-dim*: *length* (*RAT M*) = *n*
⟨*proof*⟩

lemma *Mt-gso-connect*:
fixes *j*::*nat*
assumes $j<n$
shows *Mt*!*j* = *gs.gso j*

⟨*proof*⟩

lemma *access-index-M-dim*:
  assumes $0 \leq i$
  assumes $i < n$
  shows *dim-vec* (*map of-int-hom.vec-hom M* ! *i*) = *n*
  ⟨*proof*⟩

lemma *s-dim*:
  fixes *i*::*nat*
  assumes $i \leq n$
  shows *dim-vec* (*s i*) = *n* ∧ (*s i*) ∈*carrier-vec n*
  ⟨*proof*⟩

lemma *dim-vecs-in-Mt*:
  fixes *i*::*nat*
  assumes $i<n$
  shows *dim-vec* (*Mt*!*i*) = *n*
  ⟨*proof*⟩
lemma *upper-tri*:
  fixes *i*::*nat*
    and *j*::*nat*
  assumes $j>i$
  assumes $j<n$
  shows ((*RAT M*)!*i*)· (*Mt*!*j*) =*0*
⟨*proof*⟩
lemma *one-diag*:
  fixes *i*::*nat*
  assumes $0\leq i$
  assumes $i<n$
  shows ((*RAT M*)!*i*)· (*Mt*!*i*)=*sq-norm* (*Mt*!*i*)
⟨*proof*⟩

lemma *coord-invariance*:
  fixes *j*::*nat*
  fixes *k*::*nat*
  fixes *i*::*nat*
  assumes $k\leq j$
  assumes $j+i\leq n$
  assumes $k>0$
  shows (*lattice-coord* (*s* (*j*+*i*)))\$(*n*−*k*) = (*lattice-coord* (*s j*))\$(*n*−*k*)
    ∧ (*s* (*j*+*i*)) · *Mt*!(*n*−*k*)=(*s j*) · *Mt*!(*n*−*k*)
  ⟨*proof*⟩

lemma *small-orth-coord*:
  fixes *i*::*nat*
  assumes $1\leq i$
  assumes $i\leq n$

shows $abs\ (\ (s\ i) \cdot Mt!(n-i)) \leq\ (sq\text{-}norm\ (Mt!(n-i)))*(1/2)$
⟨*proof*⟩
lemma *lattice-carrier*: $L \subseteq$ *carrier-vec n*
⟨*proof*⟩

4 Lattice Lemmas

lemma *lattice-sum-close*:
fixes u::*int vec* **and** v::*int vec*
assumes $u \in L$ $v \in L$
shows $u+v \in L$
⟨*proof*⟩

lemma *lattice-smult-close*:
fixes u::*int vec* **and** q::*int*
assumes $u \in L$
shows $q \cdot_v u \in L$

⟨*proof*⟩

lemma *smult-vec-zero*:
fixes v ::'a::*ring vec*
shows $0 \cdot_v v = 0_v$ (*dim-vec v*)
⟨*proof*⟩

lemma *coset-s*:
fixes i::*nat*
assumes $i \leq n$
shows $s\ i\ \in$ *coset*
⟨*proof*⟩

lemma *subtract-coset-into-lattice*:
fixes v::*rat vec*
fixes w::*rat vec*
assumes $v \in$ *coset*
assumes $w \in$ *coset*
shows $(v-w) \in$ *of-int-hom.vec-hom*' L
⟨*proof*⟩
lemma *t-in-coset*:
shows *uminus t* $\in$ *coset*
⟨*proof*⟩

5 Lemmas on closest distance

lemma *closest-distance-sq-pos*: *closest-distance-sq*≥ 0
⟨*proof*⟩

definition *witness*:: *rat vec*⇒*rat* ⇒ *bool*
where *witness v eps-closest* = (*sq-norm v* ≤ *eps-closest* ∧ *v*∈*coset*∧*dim-vec v* = *n*)

definition *epsilon*::*real* **where** *epsilon* = *11/10*

definition *close-condition*::*rat* ⇒ *bool*
where *close-condition eps-closest* ≡
(*if closest-distance-sq* = *0 then 0*≤ *real-of-rat eps-closest*
else real-of-rat (*eps-closest*)>*closest-distance-sq*)
∧ (*real-of-rat* (*eps-closest*)≤*epsilon*∗*closest-distance-sq*)

lemma *close-rat*:
obtains *eps-closest*::*rat*
where *close-condition eps-closest*
⟨*proof*⟩

definition *eps-closest*::*rat*
where *eps-closest* = (*if* ∃ *r*. *close-condition r then SOME r*. *close-condition r else 0*)

lemma *eps-closest-lemma*: *close-condition eps-closest*
⟨*proof*⟩

lemma *rational-tri-ineq*:
fixes *v*::*rat vec*
fixes *w*::*rat vec*
assumes *dim-vec v* = *dim-vec w*
shows (*sq-norm* (*v*+*w*))≤ *4*∗(*Max* {(*sq-norm v*), (*sq-norm w*)})
⟨*proof*⟩

lemma *witness-exists*:
shows ∃ *v*. *witness v eps-closest*
⟨*proof*⟩

6 More linear algebra lemmas

lemma *carrier-Ms*:
shows *mat-M* ∈*carrier-mat n n mat-M-inv* ∈*carrier-mat n n*
⟨*proof*⟩

lemma *carrier-L*:
fixes *v*::*rat vec*
assumes *dim-vec v* = *n*
shows *lattice-coord v*∈*carrier-vec n*
⟨*proof*⟩

lemma *sumlist-index-commute*:
fixes *Lst*::*rat vec list*

fixes *i*::*nat*
assumes *set Lst*⊆*carrier-vec n*
assumes *i*<*n*
shows (*gs.sumlist Lst*)$*i* = *sum-list* (*map* (λ*j*. (*Lst*!*j*)$*i*) [*0*..<(*length Lst*)])
⟨*proof*⟩

lemma *mat-mul-to-sum-list*:
fixes *A*::*rat mat*
fixes *v*::*rat vec*
assumes *dim-vec v* = *dim-col A*
assumes *dim-row A* = *n*
shows $A *_v v$ = *gs.sumlist* (*map* (λ*j*. *v*$*j* $\cdot_v$ (*col A j*)) [*0* ..< *dim-col A*])
⟨*proof*⟩

lemma *recover-from-lattice-coord*:
fixes *v*::*rat vec*
assumes *dim-vec v* = *n*
shows *v* = *gs.sumlist* (*map* (λ*i*. (*lattice-coord v*)$*i* $\cdot_v$ (*RAT M*)!*i*) [*0* ..< *n*])
⟨*proof*⟩

lemma *sumlist-linear-coord*:
fixes *Lst*::*int vec list*
assumes ⋀*i*. *i*<*length Lst* ⟹*dim-vec* (*Lst*!*i*) = *n*
shows *lattice-coord* (*map-vec rat-of-int* (*sumlist Lst*)) = *gs.sumlist* (*map lattice-coord* (*RAT Lst*))
⟨*proof*⟩

lemma *integral-sum*:
fixes *l*::*nat*
assumes ⋀*j1*. *j1* < *l* ⟹
map f [*0*..<*l*] ! *j1* ∈ ℤ
shows *sum-list*
(*map f* [*0*..<*l*]) ∈ ℤ
⟨*proof*⟩

lemma *int-coord*:
fixes *i*::*nat*
assumes *0*≤*i*
assumes *i*<*n*
fixes *v*::*int vec*
assumes *v*∈*L*
assumes *dim-vec v* = *n*
shows (*lattice-coord* (*map-vec rat-of-int v*))$*i*∈ℤ
⟨*proof*⟩

lemma *int-coord-for-rat*:

fixes *i*::*nat*
assumes *0*≤*i*
assumes *i*<*n*
fixes *v*::*rat vec*
assumes *v*∈*of-int-hom.vec-hom*' *L*
assumes *dim-vec v* = *n*
shows (*lattice-coord v*)$*i*∈ℤ
⟨*proof*⟩

7 Coord-Invariance

This section shows that the algorithm output matches true closest (or near-closest) vector in some trailing coordinates.

definition *I* **where**
I = (*if* ({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*) ≠ {}
then Max ({*i*∈{*0*..<*n*}. ((*sq-norm* (*Mt*!*i*)::*rat*))≤*4*∗*eps-closest*}::*nat set*) *else* −*1*)

lemma *I-geq*:
shows *I*≥−*1*
⟨*proof*⟩
lemma *I-leq*:
shows *I*<*n*
⟨*proof*⟩

lemma *index-geq-I-big*:
fixes *i*::*nat*
assumes *i*>*I*
assumes *i*<*n*
shows ((*sq-norm* (*Mt*!*i*)::*rat*))>*4*∗*eps-closest*
⟨*proof*⟩

lemma *scalar-prod-gs-from-lattice-coord*:
fixes *i*::*nat*
fixes *v*::*rat vec*
assumes *dim-vec v* = *n*
assumes *i*<*n*
shows *v*·*Mt*!*i*=*sum-list* (*map* (λ*k*. (*lattice-coord v*)$*k* ∗ (((*RAT M*)!*k*)·*Mt*!*i*)) [*i*..<*n*])
⟨*proof*⟩

lemma *correct-coord-help*:
fixes *i*::*nat*
assumes *i*<(*int n*)−*I*
assumes *witness v* (*eps-closest*)
assumes *0*<*i*
shows (*lattice-coord* (*s i*))$(*n*−*i*)=(*lattice-coord v*)$(*n*−*i*)

$\wedge$ ((*s i*) $\cdot$ *Mt*!(*n*−*i*) = *v* $\cdot$ *Mt*!(*n*−*i*))
⟨*proof*⟩

lemma *correct-coord*:
fixes *v*::*rat vec*
fixes *k*::*nat*
assumes *witness v eps-closest*
assumes *I*<*k*
assumes *k*<*n*
shows (*s n*) $\cdot$ *Mt*!(*k*) = *v* $\cdot$ *Mt*!(*k*)
⟨*proof*⟩

8 Main Theorem

This section culminates in the main theorem.

lemma *sq-norm-from-Mt*:
fixes *v*::*rat vec*
assumes *v-carr*:*v*∈*carrier-vec n*
shows *sq-norm v* = *sum-list* (*map* (λ*i*. (*v*•*Mt*!*i*)^*2*/(*sq-norm* (*Mt*!*i*))) [*0*..<*n*])
⟨*proof*⟩

lemma *bound-help*:
fixes *N*::*nat*
shows *real-of-rat* ((*rat-of-int N*)∗α^*N*) ∗ *epsilon*≤*2*^*N*
⟨*proof*⟩

lemma *present-bound-nicely*:
fixes *N*::*nat*
shows *real-of-rat* ((*rat-of-int N*)∗α^*N*∗ *eps-closest*)≤*2*^*N*∗*closest-distance-sq*
⟨*proof*⟩

lemma *basis-decay*:
fixes *i*::*nat*
fixes *j*::*nat*
assumes *i*<*n*
assumes *i*+*j*<*n*
shows *sq-norm* (*Mt*!*i*)≤ α^*j*∗*sq-norm*(*Mt*!(*i*+*j*))
⟨*proof*⟩

lemma *basis-decay-cor*:
fixes *i*::*nat*
fixes *j*::*nat*
assumes *i*<*n*
assumes *j*<*n*
assumes *i*≤*j*
shows *sq-norm* (*Mt*!*i*)≤ α^*n*∗*sq-norm*(*Mt*!*j*)
⟨*proof*⟩

theorem *Babai-Correct*:
shows *real-of-rat* ((*sq-norm* (*s n*))::*rat*) $\leq$ *2^n* $*$ *closest-distance-sq* $\wedge$ *s n* $\in$ *coset*
⟨*proof*⟩

end
end

References

[1] R. Bottesch, J. Divasón, and R. Thiemann. Two algorithms based on modular arithmetic: lattice basis reduction and Hermite normal form computation. *Archive of Formal Proofs*, March 2021. https://isa-afp.org/entries/Modular_arithmetic_LLL_and_HNF_algorithms.html, Formal proof development.

[2] J. Divasón, S. J. C. Joosten, R. Thiemann, and A. Yamada. A Verified Factorization Algorithm for Integer Polynomials with Polynomial Complexity. *Archive of Formal Proofs*, February 2018. https://isa-afp.org/entries/LLL_Factorization.html, Formal proof development.

[3] K. Kreuzer. Hardness of Lattice Problems. *Archive of Formal Proofs*, February 2023. https://isa-afp.org/entries/CVP_Hardness.html, Formal proof development.

[4] N. Stephens Davidowitz. Lecture 5: CVP and Babais Algorithm, August 2016.