K. Karamazen
De kleinste basis van een rooster en een generalisatie van Gauss reductie
1996.

De kleinste basis van een rooster en een generalisatie van Gauss reductie

door Krešimir Karamazen

29 april 1996

Introductie

In September 1995 ben ik begonnen met mijn afstudeerwerk (van de 6 sp omvang) voor mijn studie wiskunde. Ik was erg blij, dat ik roosterreductie mocht bestuderen omdat mij dit erg aanspreekt. De feedback van J. Top heeft geleid tot vele verbeteringen. Verder wil ik ook Bart Lamers bedanken voor zijn adviezen en opmerkingen. De laatste paragraaf (17) van deze tekst maakt geen onderdeel uit van mijn afstudeerwerk.

Inleiding

In deze tekst wordt een generalisatie van het Gauss algoritme voor rooster reductie vergeleken met het ultieme gereedschap op dat gebied: het LLL algoritme. (Gauss reductie noem ik verder gewoon "Reductie"). Het inzicht wordt gebruikt om een soort kleinste basis (kortst mogelijke basis) te geven in drie en vier dimensies. Verder wordt aangetoond dat de kleinste basis in hogere dimensies i.h.a. niet bestaat. Een algoritme dat alle rooster vectoren van lengte kleiner dan een gegeven waarde levert wordt uitgelegd.

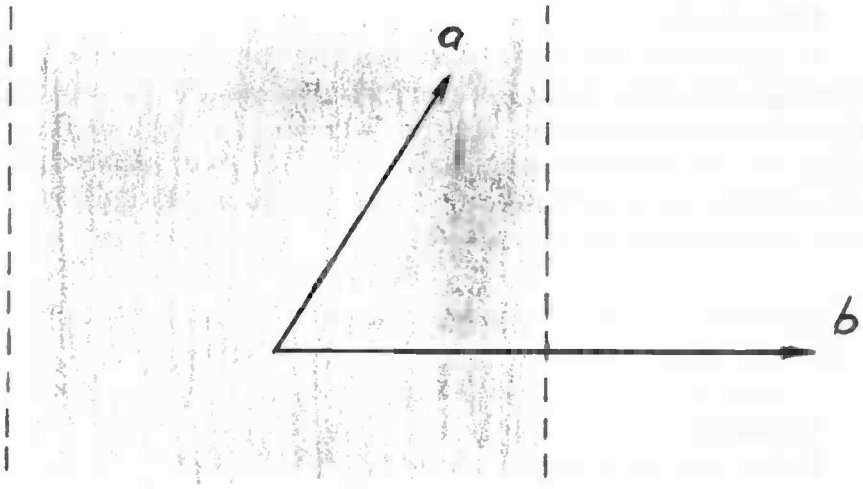
Aan de hand van de gemaakte experimenten kunnen we concluderen dat het geïmplementeerde "Reductie" algoritme een gedrukte concurrent voor het LLL algoritme is in de hieronder onderscheidene gevallen.

Het LLL algoritme levert meer informatie. Ten eerste berekent het LLL algoritme een orthogonalisatie die nodig is voor het berekenen van de kleinste vectoren in een rooster en zorgt ook voor de gunstige theoretische schattingen. Verder kan het LLL algoritme de overbodige lineair afhankelijke vectoren detecteren en uitschakelen. Dat is wat Reductie niet kan.

Beide algoritmen leveren een basis als uitvoer. Het is de bedoeling dat de vectoren in de uitvoer basis korter zijn dan in de invoer basis. Als de orthogonalisatie overbodig is, kan Reductie gebruikt worden. Het vergelijking criterium dat hier gehanteerd is: kortere vectoren in kortere tijd.

1 Algoritme "Reductie"

Het rooster reductie probleem lijkt op het orthogonalisatie probleem. Neem namelijk onafhankelijke vectoren $a, b, c, \ldots, z$ die een rooster definiëren. Probeer nu de nieuwe basis vectoren $a_1, b_1, \ldots, z_1$ in dat rooster te vinden die zo mogelijk meer onderling orthogonaal zijn. Dan hopen we dat $a_1, b_1, \ldots, z_1$ een soort minimale basis is. De onderliggende gedachte van het Reductie algoritme is, dat wordt geprobeerd elk tweetal vectoren in de basis zo orthogonaal mogelijk t.o.v. elkaar te maken.



$Z(a, b)$ geldt d.e.s.d.a. a zich in het gemarkeerde spoor bevindt.

Gegeven is een basis voor een rooster. Kies vectoren $a \neq b$ uit die basis. Als de projectie $\frac{a \cdot b}{b \cdot b} b$ van a op b in lengte groter is dan de helft van b dan is er een geheel getal $\#$ zo dat $a := a - \#b$ een projectie op b heeft, die in lengte kleiner of gelijk aan de helft van b is. Het getal $\#$ berekenen we als volgt:

$$\# := [a \cdot b / b \cdot b + \tfrac{1}{2}]$$

waarin $a \cdot b$ het inprodukt van a en b is en $[iets]$ het grootste gehele getal is dat niet groter is dan *iets*. Bovendien wordt a door deze toekenning korter en het verkregen stelsel blijft een basis. Deze operatie doen we voor elk paar verschillende vectoren uit $\{a, b, c, \ldots, z\}$ zolang het kan. Omdat vectoren in een rooster niet oneindig vaak korter kunnen worden, is het een eindig proces. Uiteindelijk geldt voor elk paar (h, p) uit $\{a, b, \ldots, z\}^2$ met $h \neq p$: de projectie van h op p is in lengte kleiner of gelijk aan de helft van p.

Het algoritme Reductie is een generalisatie van de Gauss reductie, zie [Cohen, pagina 23].

2 LLL algoritme

Dit is het LLL algoritme: op een gegeven moment zijn $a, b, \ldots, y$ al LLL-gereduceerd. We definiëren $A, B, \ldots, Y$, de Gram-Schmidt orthogonalisatie van $a, b, \ldots, y$:

$$A = a$$
$$B = b - \frac{b \cdot A}{A \cdot A} A$$
$$\ldots$$
$$Y = y - \frac{y \cdot A}{A \cdot A} A - \ldots - \frac{y \cdot X}{X \cdot X} X$$

(alleen A zit in het rooster i.h.a.) Hiervoor geldt dat $|B| > \frac{1}{2}|A|, |C| > \frac{1}{2}|B|, \ldots, |Y| > \frac{1}{2}|X|$.
($|B| > \frac{1}{2}|A|$ is de Siegel conditie op A en B. Deze kan vervangen worden door de sterkere Lovász conditie:

$$|B|^2 \geq (\tfrac{3}{4} - \mu^2)|A|^2$$

waarbij $\mu = b \cdot A / A \cdot A$. In de implementatie gebruik ik de Lovász conditie i.p.v. de Siegel conditie.)

Bovendien geldt dat de lengte van de projectie van m op $A, B, \ldots, L$ kleiner is dan de helft van de lengte van $A, B, \ldots, L$ respectievelijk.

Nu komt z aan de orde. De eis is dat de lengte van de projectie van z op $A, B, \ldots, Y$ kleiner is dan de helft van de lengte van $A, B, \ldots, Y$ respectievelijk. Dat is als volgt gedaan: bereken eerst het getal $\# := [z \cdot Y / Y \cdot Y + \frac{1}{2}]$. Daarna wordt de nieuwe z berekend volgens $z := z - \#y$. Dankzij het feit dat $y \cdot Y = Y \cdot Y$, voldoet de nieuwe z aan de eis ten opzichte van Y. Zo gaan we door met $X, W, V, \ldots, A$. De berekening van nieuwe goede z ten opzichte van X verandert de projectie van z op Y niet. Aan het eind van het proces voldoet z aan de eis.

Hierop volgt nog een eventuele verwisseling van z en y mocht het zijn dat $|Z| < \frac{1}{2}|Y|$. Het algoritme eindigt omdat produkten van bepaalde volumes minstens 3/4 kleiner worden bij elke verwisseling.

3 LLL en Reductie

Waarom zou het LLL algoritme beter zijn dan het Reductie algoritme? Er zijn twee heuristische redenen die beide volgen uit het feit dat in LLL gereduceerd wordt t.o.v. de Gram-Schmidt orthogonalisatie. Voor het gemak kunnen we met $Z(a, b)$ de volgende zin noteren: "de projectie van a op b is kleiner dan de helft van b". Dan werkt LLL naar $Z(b, A), Z(c, A), \ldots, Z(z, A), Z(c, B), Z(d, B), \ldots, \ldots, Z(z, Y)$.
Nu de twee redenen waarom dit sterk is:
1) $B, \ldots, Y$ zijn korter dan $b, \ldots, y$ respectievelijk dus daar op is de reductie sterker.
2) $A, B, \ldots, Y$ spannen de ruimte beter op dan $a, b, \ldots, y$ (omdat de eersten orthogonaal zijn en de tweeden kunnen i.h.a. nooit echt orthogonaal worden.)

Voor de uitvoer van het LLL algoritme gelden gunstige schattingen zoals die uit Voorbeeld 2 hieronder.

4 Voorbeelden

4.1 Voorbeeld 1

Hier geef ik een voorbeeld waarin het LLL algoritme beter werkt dan Reductie. Neem de volgende basis:

$a = (\frac{1}{2}, \frac{\sqrt{3}}{2}, 0)$
$b = (\frac{1}{2}, -\frac{\sqrt{3}}{2}, 0)$
$c = (1, 0, \epsilon)$

We hebben $|a| = |b| = 1$, $|c|^2 = 1 + \epsilon^2$. Nu berekenen we de lengtes van de projecties:

$|a \cdot b|/|a| = |a \cdot c|/|a| = \frac{1}{2}|a| \leq \frac{1}{2}|a|$
$|a \cdot b|/|b| = |b \cdot c|/|b| = \frac{1}{2}|b| \leq \frac{1}{2}|b|$
$|a \cdot c|/|c| = |b \cdot c|/|c| = 1/(2 + 2\epsilon^2)|c| \leq \frac{1}{2}|c|$

Het is hier te zien dat de eindconditie van Reductie bereikt is. Daarentegen lukt het aan het LLL algoritme hiervan wat meer te maken. Vector c is daar door $c - a - b$ vervangen en zo wordt $c = (0, 0, \epsilon)$ loodrecht op a en op b.

4.2 Voorbeeld 2

Gegeven een basis $a, b, \ldots, z$. Noteer met L de Gram matrix van die basis. In de eerste rij van de Gram matrix staan inprodukten van a met $a, b, \ldots, z$. De tweede rij bevat inprodukten van b enzovoort. De determinant van de Gram matrix is gelijk voor alle bases in het rooster.

Zij P het produkt van de lengtes van alle vectoren in de basis:

$$P = |a||b| \ldots |z|$$

Zij n het aantal elementen in de basis (dimensie van het rooster). Voor een basis die de uitkomst is van het LLL algoritme geldt:

$$P^2 \leq 2^{((n(n-1)/2)} detL$$

zoals in [Cohen, pagina 84] gegeven.
Voor Reductie bestaat er geen schatting van deze soort. Dat kunnen we zien aan de hand van een aanpassing van het al eerder gegeven voorbeeld.
Neem namelijk

$a = (\frac{1}{2}, \frac{\sqrt{3}}{2}, 0)$
$b = (\frac{1}{2}, -\frac{\sqrt{3}}{2}, 0)$
$c = (\cos\epsilon, 0, \sin\epsilon)$

met ϵ een klein reëel getal. We hebben $|a| = |b| = |c| = 1$.
Nu berekenen we de lengtes van de projecties:
(geschaalde projecties; $|a \cdot b|/|a|^2 \leq \frac{1}{2}$ is hetzelfde als $|a \cdot b|/|a| \leq \frac{1}{2}|a|$)

$|a \cdot b|/|a|^2 = |a \cdot b|/|b|^2 = \frac{1}{2} \leq \frac{1}{2}$
$|a \cdot c|/|a|^2 = |b \cdot c|/|b|^2 = |a \cdot c|/|c|^2 = |b \cdot c|/|c|^2 = |\cos\epsilon|/2 \leq \frac{1}{2}$

Zo is hier aan de eisen van Reductie voldaan. P^2 is steeds 1 en $\det L$ is gelijk aan $(\sqrt{3}\sin\epsilon)/2$. Er zijn dus Reductie gereduceerde bases zo dat

$$P^2 > K det L$$

voor elke vantevoren gekozen constante K.

4.3 Voorbeeld 3

Hier geef ik een voorbeeld waarin Reductie een kortere basis levert dan het LLL algoritme. Neem de volgende basis:

$a = (4, 0, 0)$
$b = (2, 2, 0)$
$c = (2, 1, 1)$.

De Gram-Schmidt orthogonalisatie van a, b, c is A, B, C:

$A = a$
$B = (0, 2, 0)$
$C = (0, 0, 1)$.

Hiermee doet het LLL algoritme niets meer. Er geldt namelijk:

$\frac{b \cdot A}{A \cdot A} = \frac{1}{2}$
$\frac{c \cdot A}{A \cdot A} = \frac{1}{2}$
$\frac{c \cdot B}{B \cdot B} = \frac{1}{2}$
$|B| = \frac{1}{2}|A|$
$|C| = \frac{1}{2}|B|$.

Daarentegen is de eindtoestand van Reductie niet bereikt:

$\frac{a \cdot b}{b \cdot b} = 1$
$\frac{b \cdot a}{a \cdot a} = \frac{1}{2}$
$\frac{c \cdot a}{a \cdot a} = \frac{1}{2}$
$\frac{a \cdot c}{c \cdot c} = 4/3$
$\frac{b \cdot c}{c \cdot c} = 1$
$\frac{c \cdot b}{b \cdot b} = 3/4$.

Aan vier van de zes eisen is niet voldaan. Tenminste twee van a, b, c zullen in Reductie nog korter worden, bijvoorbeeld:

$a := a - c$
$b := b - c$.

Nu hebben we dus:

$a = (2, -1, -1)$
$b = (0, 1, -1)$
$c = (2, 1, 1)$
$\frac{a \cdot b}{b \cdot b} = 0$
$\frac{b \cdot a}{a \cdot a} = 0$
$\frac{c \cdot a}{a \cdot a} = 1/3$
$\frac{a \cdot c}{c \cdot c} = 1/3$
$\frac{b \cdot c}{c \cdot c} = 0$
$\frac{c \cdot b}{b \cdot b} = 0.$

4.4 Voorbeeld 4

Hier geef ik nog een voorbeeld waarin Reductie een kortere basis levert dan het LLL algoritme. Bovendien zien we hier hoe het LLL algoritme de output levert die langer is dan de input. Neem de volgende basis:

$v1 = (3465, -2198, -3218, 1946, -4455)$
$v2 = (6930, -6594, 1609, -1946, 3326)$
$v3 = (-3465, -1099, -3218, 7784, 417)$
$v4 = (-567, -2110, 4955, -1868, -7789)$

De Reductie eindtoestand is hier bereikt. Het LLL algoritme wijzigt de eerste drie vectoren $v1, v2, v3$ niet. Maar in plaats van $v4$ krijgen we

$v4' = (-7497, -1011, 4955, 3970, -2917)$

Deze $v4'$ is langer dan $v4$ en heeft met $v1, v2, v3$ kleinere hoeken dan de originele $v4$:
Lengte$(v4) = 9669$
Lengte$(v4') = 10298$
Hoek$(v4, v1) = 75.04°$
Hoek$(v4', v1) = 75.02°$
Hoek$(v4, v2) = 88°$
Hoek$(v4', v2) = 59°$
Hoek$(v4, v3) = 71°$
Hoek$(v4', v3) = 64°$

5 Methodologie van het vergelijken

De hierboven gegeven voorbeelden laten zien dat het niet uitgesloten is dat LLL en Reductie concurrerend zijn wat de kwaliteit (kortere vectoren) van de gekregen basis betreft. Een andere vraag is welke van de twee een betere tijd-complexiteit heeft. Om kwaliteit en tijd-complexiteit van de twee algoritmen

na te gaan, heb ik het Reductie algoritme geïmplementeerd. Het LLL algoritme is overgenomen uit [Cohen, pagina 86-87].

Hier moet ik waarschuwen dat het daar uitgelegde subalgoritme SWAP een fout bevat. Er ontbreekt namelijk de aanpassing van de orthogonalisatie na de verwisseling. (En die aanpassing was niet nodig geweest als de volledige orthogonalisatie al in het begin berekend was).

In de programma's worden de tellers ingebouwd met de taak om het aantal vermenigvuldigingen en het aantal optellingen bij te houden. De programma's zijn in de bijlage gegeven.

De kwaliteit van het gekregen resultaat is een moeilijk te definiëren begrip. De vraag is wanneer een basis beter is dan een andere basis. Hierop volgt een degressie naar de ordening van de bases. Ik geef de algoritmen die de beste resultaten voor twee, drie en vier vectoren leveren.

6 Definitie van de kleinste basis

We definiëren eerst wat we bedoelen met "de kleinste basis". De kleinste basis is een basis van het rooster die in zekere zin kleiner is dan elke andere basis, of meer algemeen, kleiner dan elk ander onafhankelijk stelsel in het rooster.

De transformatie matrices zijn het natuurlijke gereedschap om alle bases van het rooster te beschrijven. Als $a, b, \ldots, c$ een basis van het rooster is dan is elke andere basis van hetzelfde rooster gekregen door een matrix van gehele getallen met determinant gelijk aan ± 1. En als een transformatie wordt gegeven door een matrix van gehele getallen en $det \neq \pm 1, 0$, dan levert deze een lineair onafhankelijk stelsel rooster elementen dat geen basis is.

Kijk naar de diagonale matrices met ± 1 op diagonaal. Alleen in dit eenvoudige geval, waarin een vector een plus of minus teken krijgt, zijn er al 2^n mogelijkheden. Terwijl we in wezen elke keer dezelfde basis krijgen. Er zijn ook nog $n!$ permutatie matrices die dezelfde basis leveren. Vanwege deze, voor mij ondoorzienbare, en in verband met het begrip "kleinere basis" onnodige complexiteit, zoek ik de definitie van kleinste basis ergens anders, namelijk bij een partiële ordening van de bases.

Laat $a, b, \ldots, z$ een basis van n vectoren van een rooster L. Voor deze L definiëren we $\mathcal{A}$ door

$$\mathcal{A}(L) := \{\{a_1, b_1, \ldots, z_1\} | a_1, b_1, \ldots, z_1 \text{ is een onafhankelijk stelsel van } n \text{ vectoren in} L\}.$$

Zij $a_1, b_1, \ldots, z_1$ een element van $\mathcal{A}$. Zij $a_2, b_2, \ldots, z_2$ een ander element van $\mathcal{A}$. Stel verder dat $a_1, b_1, \ldots, z_1$ en $a_2, b_2, \ldots, z_2$ gesorteerd zijn naar lengte, d.w.z. $|a_1| \leq |b_1| \leq \ldots \leq |z_1|$ en $|a_2| \leq |b_2| \leq \ldots \leq |z_2|$. Dan is het eerste stelsel $a_1, b_1, \ldots, z_1$ per definitie kleiner of gelijk aan het tweede stelsel $a_2, b_2, \ldots, z_2$ als $|a_1| \leq |a_2|, |b_1| \leq |b_2|, \ldots, |z_1| \leq |z_2|$. Zo wordt een partiële ordening op de verzameling van alle onafhankelijke stelsels van n vectoren gedefinieerd. (En de laatste zin is niet precies genoeg. Er zijn namelijk veel verschillende stelsels die volgens de ordening gelijk zijn. Dat past niet in de definitie van een partieel geordende verzameling. De remedie is om te kijken naar equivalentie klassen

van gelijke stelsels. Om de uitleg te vereenvoudigen laten we deze verdeling in equivalentie klassen uit de tekst weg.)
Als we kijken naar de bases alleen, terwijl we de overige lineair onafhankelijke stelsels weglaten, dan krijgen we een partiële ordening op de verzameling van alle bases. (Met dezelfde opmerking als boven.)
Het is nu mogelijk om de kleinste basis te definiëren als het kleinste element in een partieel geordende verzameling.

Definitie 1 *Een minimale basis van een rooster L is een minimaal element in $\mathcal{A}(\mathcal{L})$ dat ook een basis is.*

Definitie 2 *(Alternatieve definitie)*
De quasi-minimale basis van een rooster L is een minimale element tussen alle bases van L.

Definitie 3 *De kleinste basis van een rooster L is het kleinste element in $\mathcal{A}(\mathcal{L})$ dat ook een basis is.*

Definitie 4 *(Alternatieve definitie)*
Een quasi-kleinste basis van een rooster L is het kleinste element tussen alle bases van L.

In het hieronder gegeven voorbeeld zullen we zien dat de definities zoals boven gegeven niet equivalent zijn met de bijbehorende alternatieve definities. Daar zullen we namelijk een basis laten zien die wel quasi-kleinste maar niet de kleinste is. (En dus ook quasi- minimaal maar niet minimaal.) Bovendien wordt aangetoond, dat de kleinste basis niet hoeft te bestaan in dimensies groter dan vier.

Daarna worden de algoritmen gegeven die de kleinste basis leveren in dimensies twee, drie en vier.

7 Een voldoende voorwaarde voor de kleinste basis

Neem een gesorteerde basis $(a, b, \ldots, y, z)$. Stel dat we weten dat $(a, b, \ldots, y)$ kleinste is in het deelrooster opgespannen door $a, b, \ldots, y$. We zoeken een conditie op z die het kleinste zijn van $(a, b, \ldots, y, z)$ verzekert. De volgende conditie blijkt voldoende: voor alle gehele getallen $A, B, \ldots, Z$ is $|Aa + Bb + \ldots + Zz|$ groter of gelijk aan $|z|$ mits Z niet nul.

Voor een uitdrukking $Aa + Bb + \ldots + Zz$ waarin $Z \neq 0$ zullen we zeggen: ze gebruikt z.

Stelling 1 *Zij $(a, b, \ldots, y, z)$ een basis van het rooster L met $|a| \leq |b| \leq \ldots \leq |y| \leq |z|$ zo dat het volgende geldt:*
1. $(a, b, \ldots, y)$ is de kleinste basis in het deelrooster L' opgespannen door $a, b, \ldots, y$.

2. alle uitdrukkingen $|Aa + Bb + \ldots + Yy + Zz|$ *die* z *gebruiken zijn groter of gelijk aan* $|z|$ *(met* $A, B, \ldots, Y, Z$ *geheel).*

Dan is $(a, b, \ldots, y, z)$ *de kleinste basis.*

Bewijs van de stelling:
Om dit te bewijzen, ordenen we alle elementen van $\mathcal{A}(\mathcal{L})$ naar de vector lengte, zoals al opgemerkt in verband met de partiële ordening.
Alle elementen van $\mathcal{A}(\mathcal{L})$ krijgen we door $(a, b, \ldots, z)$ te transformeren met behulp van een niet-singuliere matrix van gehele getallen. Zo vinden we een $(a_1, b_1, \ldots, z_1)$ uit $\mathcal{A}(\mathcal{L})$. Bijvoorbeeld m_1 is gekregen door een transformatie $m_1 := Aa + Bb + \ldots + Zz$ met $A, B, \ldots, Z$ geheel.
Nu beweer ik dat tenminste een van $a_1, \ldots, z_1$ de vector z gebruikt. Want als dat niet zo was, dan zaten alle vectoren $a_1, \ldots, z_1$ in het opspansel van $a, \ldots y$, en dat impliceert afhankelijkheid.
Maar als z gebruikt was dan is het resultaat in lengte groter of gelijk aan z. Dus, er is altijd tenminste één van $a_1, b_1, \ldots, y_1, z_1$ die in lengte groter of gelijk aan z is.
Neem aan dat $a_1, b_1, \ldots, l_1$ allemaal z niet gebruiken en m_1 is de eerste die z gebruikt. Dan is $|m_1|$ groter of gelijk aan $|z|$. Maar $(a_1, b_1, \ldots, y_1, z_1)$ is geordend. Daarom zijn alle $m_1, n_1, \ldots, y_1, z_1$ in lengte groter of gelijk aan z. Dat levert

$$|m_1| \geq |z| \geq |m|, |n_1| \geq |z| \geq |n|, \ldots, |y_1| \geq |z| \geq |y|, |z_1| \geq |z|$$

waaruit volgt:

$$|m_1| \geq |m|, |n_1| \geq |n|, \ldots, |y_1| \geq |y|, |z_1| \geq |z|.$$

Nu moeten we nog bewijzen dat $|a_1|, |b_1|, \ldots, |k_1|, |l_1|$ groter of gelijk aan $|a|, |b|, \ldots, |k|$, en $|l|$, respectievelijk zijn. Omdat $a_1, b_1, \ldots, k_1, l_1$ allemaal z niet gebruiken, zitten ze in het deelrooster L' opgespannen door $a, \ldots, y$. Daarom kunnen we $a_1, b_1, \ldots, k_1, l_1$ tot een geordend onafhankelijk stelsel in het deelrooster L' uitbreiden. Voor het gemak kiezen we de uitbreidende vectoren heel groot, zo dat ze achter in de rij komen te staan. Het zo gekregen stelsel moet groter of gelijk zijn aan $(a, b, \ldots, y)$ dankzij het kleinste zijn $(a, b, \ldots, y)$. Daarom is ook $(a_1, b_1, \ldots, l_1)$ groter of gelijk aan $(a, b, \ldots, l)$. □

8 Voorbeelden

Het is niet zo dat er altijd een kleinste basis bestaat.

Neem bijvoorbeeld een vijfdimensionaal rooster opgespannen door

$v = (1, 0, 0, 0, 0)$
$w = (0, 1, 0, 0, 0)$
$x = (0, 0, 1, 0, 0)$
$y = (0, 0, 0, 1, 0)$
$z = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$

Het is duidelijk (eventueel door de toepassing van de stelling boven) dat (v, w, x, y) de kleinste is. De stelling is niet van toepassing voor (v, w, x, y, z) omdat

$$z_1 := v + w + x + y - 2z = (0, 0, 0, 0, 1)$$

in lengte kleiner is dan z. Maar (z_1, v, w, x, y) is geen basis. Het is wel het kleinste element van $\mathcal{A}(\mathcal{L})$. Om dat te zien merk op dat (z_1, v, w, x, y) orthogonaal is of kijk naar

$$Vv + Ww + Xx + Yy + Zz = (V, W, X, Y, 0) + Z(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2})$$
met V, W, X, Y, Z geheel.

Er is niets korter te krijgen dan (z_1, v, w, x, y). Zo zien we dat er hier geen kleinste basis bestaat.

Als we de alternatieve definitie gebruiken dan valt (z_1, v, w, x, y) buiten beschouwing omdat het geen basis is. In dat geval bestaat er wel een quasi-kleinste oplossing: (z_1, v, w, x, z). Dat laatste zien we als volgt. Er zijn twee keer vijf vectoren

$$\pm z_1, \pm v, \pm w, \pm x, \pm y$$

van lengte 1 in het rooster. De volgende kleinste lengte is die van $|z| = \frac{\sqrt{5}}{2}$. De eerste vijf vectoren zijn geen basis. Maar vier van die vijf vormen met z een quasi-kleinste basis.

Dit voorbeeld is uit te breiden naar dimensie zes:

$$v = (1, 0, 0, 0, 0, 0)$$
$$w = (0, 1, 0, 0, 0, 0)$$
$$x = (0, 0, 1, 0, 0, 0)$$
$$y = (0, 0, 0, 1, 0, 0)$$
$$z = (\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 0)$$
$$a = (0, 0, 0, 0, 0, N)$$

waarbij N een groot positief getal is. De kleinste onafhankelijke stelsel hier is

$$(z_1, v, w, x, y, a)$$

met $z_1 = (0, 0, 0, 0, 1, 0)$. De quasi-kleinste basis is (z_1, v, w, x, z, a).

Het bewijs hiervan berust op het feit dat elke uitdrukking $Vv + Ww + Xx + yY + Zz + Aa$, met V, W, X, Y, Z, A geheel, die a gebruikt, in lengte groter of gelijk is aan N.

Voor hogere dimensies dan zes breiden we deze stap inductief uit.

9 Een zwakkere voldoende voorwaarde voor de kleinste basis in een rang 2, 3 en 4

In de vorige voldoende voorwaarde voor de kleinste basis hadden we een basis $(a, b, \ldots, z)$ van kardinaliteit n met $|a| \leq |b| \leq \ldots \leq |z|$ zo dat $(a, b, \ldots, y)$ een kleinste basis van een deelrooster was en zo dat $|z|$ de kleinste was tussen alle uitdrukkingen $|Aa + Bb + \ldots + Zz|$ die z gebruiken.

Een zwakkere voorwaarde eist dat de uitdrukkingen $|Aa + Bb + \ldots + z|$ (dus met $Z = 1$) altijd groter of gelijk zijn aan $|z|$. De bewering is dat dan in gevallen $n = 2, 3, 4$ ook alle andere uitdrukkingen $|Aa + Bb + \ldots + Zz|$ (met willekeurige $Z \neq 0$) groter of gelijk zijn aan $|z|$.

Stelling 2 *Laat L een rooster zijn met de eigenschap $2 \leq rang(L) \leq 4$. Stel dat $(a, b, \ldots, y, z)$ een basis van het rooster L is met $|a| \leq |b| \leq \ldots \leq |y| \leq |z|$ zo dat het volgende geldt:*
1. $(a, b, \ldots, y)$ is de kleinste basis in het deelrooster L' opgespannen door $a, b, \ldots, y$.
2. alle uitdrukkingen $|Aa + Bb + \ldots + Yy + z|$ zijn groter of gelijk aan $|z|$ (met $A, B, \ldots, Y$ geheel).

Dan geldt: Alle uitdrukkingen $|Aa + Bb + \ldots + Yy + Zz|$ die z gebruiken zijn groter of gelijk aan $|z|$ (met $A, B, \ldots, Y, Z$ geheel).

Gevolg 1 *Onder de veronderstellingen van Stelling 2 geldt dat $(a, b, \ldots, y, z)$ de kleinste basis is.*

Bewijs: volgt uit Stelling 1.

Bewijs van de stelling:
Voor het gemak nemen we $n = 4$ en een basis (w, x, y, z) met $|w| \leq |x| \leq |y| \leq |z|$ zo dat (w, x, y) de kleinste basis is en zo dat $|z|$ de kleinste was tussen alle uitdrukkingen $|Ww + Xx + Yy + z|$. We construeren een orthogonalisatie w_1, x_1, y_1, z_1

$$
\begin{aligned}
w_1 &= w \\
x_1 &= x - \tfrac{x \cdot w_1}{w_1 \cdot w_1} w_1 \\
y_1 &= y - \tfrac{y \cdot w_1}{w_1 \cdot w_1} w_1 - \tfrac{y \cdot x_1}{x_1 \cdot x_1} x_1 \\
z_1 &= z - \tfrac{z \cdot w_1}{w_1 \cdot w_1} w_1 - \tfrac{z \cdot x_1}{x_1 \cdot x_1} x_1 - \tfrac{z \cdot y_1}{y_1 \cdot y_1} y_1
\end{aligned}
$$

Het doel is nu om de lengte van de projectie van z op $\text{span}(w_1, x_1, y_1) = \text{span}(w, x, y)$ te schatten. Er bestaat een geheel getal Y_0 zo dat de lengte van de projectie van $z - Y_0 y$ op y_1 kleiner of gelijk aan $\frac{1}{2}|y_1|$ is. De lengte van de projectie van $z - Yy$ op z_1 is gelijk aan $|z_1|$ voor alle Y. Zo iets doen we ook met $z - Y_0 y$ t.o.v. x_1 en dan ook verder met w_1. Het resultaat is dat er gehele getallen Y_0, X_0, W_0 bestaan zo dat projecties van $z - Y_0 y - X_0 x - W_0 w$ op y_1, x_1, w_1 in lengte kleiner of gelijk zijn aan de helft van $|y_1|, |x_1|, |w_1|$ respectievelijk en projectie van $z - Y_0 y - X_0 x - W_0 w$ op z_1 is in lengte gelijk aan $|z_1|$. Deze nieuwe vector $z - Y_0 y - X_0 x - W_0 w$ is in lengte groter of gelijk aan

$|z|$. Een bovenschatting voor het kwadraat van de lengte van de projectie van $z - Y_0y - X_0x - W_0w$ op span(w, x, y) is

$\frac{1}{4}|w_1|^2 + \frac{1}{4}|x_1|^2 + \frac{1}{4}|y_1|^2$
$\leq \frac{1}{4}|w|^2 + \frac{1}{4}|x|^2 + \frac{1}{4}|y|^2$
$\leq \frac{3}{4}|z|^2$.

Het feit dat de projecties van $z - Y_0y - X_0x - W_0w$ en z op z_1 in lengte gelijk zijn samen met het feit dat $|z|$ kleiner of gelijk is dan $|z - Y_0y - X_0x - W_0w|$ levert dat de projectie van z op span(w, x, y) in lengte kleiner of gelijk is aan de projectie van $z - Y_0y - X_0x - W_0w$ daarop en dus kleiner of gelijk aan $\sqrt{3}/2|z|$. Dat is de bovenschatting van de lengte van de projectie van z op span(w, x, y). De lengte $|z_1|$ moet hierdoor groter of gelijk aan $\frac{1}{2}|z|$ zijn. Als we nu een uitdrukking $Ww + Xx + Yy + Zz$ hebben met $|Z| > 1$ dan is de projectie van $Ww + Xx + Yy + Zz$ op z_1 in lengte groter of gelijk aan $|z|$ en hetzelfde geldt dus voor $Ww + Xx + Yy + Zz$ zelf. Hiermee is bewezen dat als $|z|$ kleiner of gelijk is aan elke $|Ww + Xx + Yy + z|$ dan is $|z|$ ook kleiner of gelijk aan elke $|Ww + Xx + Yy + Zz|$ die z gebruikt.

De bewijzen voor twee en drie vectoren gaan volgens dezelfde lijnen, alleen is er wat minder te orthogonaliseren en de schattingen zijn beter. □

Gevolg 2 *Het algoritme Reductie levert in het geval van een rooster van rang 2 de kleinste basis.*

Bewijs: Twee vectoren y, z met z in lengte groter of gelijk aan y voldoen aan $Z(z, y)$ waaruit volgt dat $z - Yy$ groter of gelijk aan z is voor alle gehele Y. □

10 De kleinste basis in het rang 3 geval

Stelling 3 *De kleinste basis in het rang 3 geval bestaat en a, b, c met $|a| \leq |c|, |b| \leq |c|$ is de kleinste wanneer*

$Z(a, b), Z(a, c), Z(b, c), Z(b, a), Z(c, a), Z(c, b),$ (1)
$Z(c, a \pm b)$ (2)

Opmerking:
De eerste regel is een Reductie postconditie. De tweede regel verzekert het kleinste zijn van de basis. Door een rechtstreekse uitbreiding van Reductie krijgen we het algoritme met de twee regels als postconditie. We moeten nameljik controleren of $Z(c, a \pm b)$ geldt en als niet dan de toekenningen

$c := c - C(a \pm b)$

uitvoeren met een geschikte gehele $C = [\frac{c \cdot (a \pm b)}{|a \pm b|^2} + \frac{1}{2}]$. Verder worden de verwisselingen uitgevoerd om te zorgen dat c de langste van de drie vectoren blijft.
Bewijs:

Neem aan dat (1) en (2) gelden. We gaan eerst bewijzen dat de basis de kleinste is.

Volgens Gevolg 2 is (a, b) de kleinste. Volgens de Stelling 2 is het genoeg om te bewijzen dat $|c - Aa - Bb|$ niet kleiner is dan $|c|$ voor alle gehele A, B.
Als $A = 0$ of $B = 0$ dan zijn we klaar door veronderstellingen.
Hetzelfde geldt als A in $\{1, -1\}$ zit en B in $\{1, -1\}$ zit.
Dus, $A \neq 0$ en $B \neq 0$ en tenminste één van A,B zit niet in $\{-1, 1\}$.
Nu moet bewezen worden, dat

$$|c - Aa - Bb| \geq |c|$$

of equivalent

$$|A|^2|a|^2 + |B|^2|b|^2 \geq 2Aa \cdot c + 2Bb \cdot c - 2ABa \cdot b$$

waarbij $a \cdot b$ het inprodukt van a en b is.
We gaan bewijzen dat

$$|A|^2|a|^2 + |B|^2|b|^2 \geq 2|Aa \cdot c| + 2|Bb \cdot c| + 2|ABa \cdot b|$$

(en verder schrijven we A voor $|A|$ en B voor $|B|$).
Maar hiervoor is het voldoende dat de volgende drie ongelijkheden gelden:

$A|a|^2 \geq 2A|a \cdot c|$
$B|b|^2 \geq 2B|b \cdot c|$
$A(A-1)|a|^2 + B(B-1)|b|^2 \geq 2AB|a \cdot b|$

De eerste twee ongelijkheden zeggen dat voldaan is aan $Z(c, a)$ en $Z(c, b)$. Als $A > B$ dan volgt de derde ongelijkheid uit $Z(b, a)$. Want dan geldt namelijk $A(A-1) \geq AB$.
Hetzelfde voor $B > A$.
Als $A = B$ dan geldt $A(A-1) \geq A^2/2$ en $B(B-1) \geq B^2/2$ waaruit het bewijs volgt. De condities (1) en (2) zijn altijd bereikbaar door het algoritme. Dat bewijst het bestaan. □

11 De kleinste basis in het rang 4 geval

Het onderstaande algoritme levert de kleinste basis in een rooster van dimensie vier.

Algoritme:

Neem een basis w, x, y, z. Pas het algoritme voor drie vectoren toe op elk drietal. Er zijn vier drietallen en als één van die niet de kleinste is dan moet hij nog een keer de invoer van het algoritme worden. Uiteindelijk is elk drietal dan de kleinste basis van door hem opgespannen deelrooster, omdat de vectoren alleen korter worden in de loop van het algoritme voor drie vectoren. Er is een langste

vector z tussen de vier. Als $|z|$ kleiner of gelijk aan elke $|z-(Ww+Xx+Yy)|$ is, dan hebben we volgens Gevolg 1 de kleinste basis.
Dat gaan we nu onderzoeken.
Als deze $Ww+Xx+Yy$ in lengte groter is dan twee keer de projectie van z op span(w,x,y) dan is $z-(Ww+Xx+Yy)$ in lengte groter of gelijk aan z. Immers, de projectie van z op $(Ww+Xx+Yy)$ is in lengte kleiner dan de projectie van z op span(w,x,y), dus in lengte kleiner dan de helft van $(Ww+Xx+Yy)$. Dat betekent dat $Z(z,Ww+Xx+Yy)$ geldt en daarom is $|z-(Ww+Xx+Yy)|$ niet kleiner dan $|z|$.

Dat waren de gevallen waarin $|Ww+Xx+Yy|$ groter was dan een vaste constante. Er zijn eindig veel gevallen waarin $|Ww+Xx+Yy|$ kleiner is dan die constante. Een algoritme om al die gevallen te vinden is beneden uitgelegd. We controleren dus voor elke $z-(Ww+Xx+Yy)$ of deze in lengte kleiner is dan z. Als dat zo is, dan vervangen we z door $z-(Ww+Xx+Yy)$ waarmee z korter is geworden en het nieuwe stelsel nog steeds een basis is. Dan starten we het algoritme vanaf het begin. Als er geen $z-(Ww+Xx+Yy)$ is, die korter is dan z, dan hebben we de kleinste basis. Eind algoritme.

Opmerking: Het is niet nodig om te kijken naar de uitdrukkingen $z-(Ww+Xx+Yy)$ waarin een van W,X,Y nul is. Dat is al gedaan door het algoritme voor drie vectoren.

12 De kleinste vectoren in een rooster

Dit is het algoritme eerst gegeven door U.Fincke en M.Pohst in [Fincke-Pohst].

Zij α een gegeven positieve constante. Het doel is om alle vectoren in het rooster te vinden die korter zijn dan α. Het is verstandig om de gegeven basis eerst te reduceren door bijvoorbeeld het LLL algoritme toe te passen. Dat is handig want daar wordt ook de orthogonalisatie berekend. Zo hebben we een (niet noodzakelijk geordende) basis $a,b,\ldots,z$ en een bijbehorende orthogonalisatie $a_1,b_1,\ldots,z_1$ te beginnen met $a_1=a, b_1=b-\frac{b\cdot a_1}{a_1\cdot a_1}a_1$ enzovoort. Nu gaan we het doel van het algoritme even versterken. Zij β een vector niet noodzakelijk in het rooster. Dan berekent KLEINSTE$(\alpha,\beta,(a,b,\ldots,z))$ alle vectoren γ in het rooster zo dat $|\gamma+\beta|$ kleiner is dan α.

Alle oplossingen zijn van de vorm

$$Aa+Bb+\ldots+Zz$$

voor bepaalde gehele $A,B,\ldots,Z$. We kiezen eerst Z. Door de keuze van Z is de projectie van $Aa+Bb+\ldots+Zz+\beta$ op z_1 volledig bepaald. Het is gelijk aan de projectie van $Zz+\beta$ op z_1 want $a,b,\ldots,y$ zijn loodrecht op z_1. Deze projectie mag niet in lengte groter zijn dan α. Zo zien we dat er maar eindig veel keuzes voor Z bestaan. Als een Z gekozen is dan komt Y aan de orde. Omdat Z vast is, bepaalt Y de projectie van $Aa+Bb+\ldots+Zz+\beta$ op y_1. Het kwadraat van de lengte van deze projectie opgeteld bij het kwadraat van de lengte van de vorige projectie moet kleiner zijn dan α^2. Zo gaan we door.

We proberen gewoon alle mogelijkheden.

Stel dat KLEINSTE$(\alpha, \beta, (a, b, \ldots, y))$ werkt voor elke $\alpha, \beta, a, b, \ldots, y$. We gaan het recursief met z uitbreiden.
Dat gaat als volgt. Zij α, β gegeven getal en vector. De vectoren die we zoeken zijn allemaal van de vorm $Aa + Bb + \ldots + Zz$, met $A, B, \ldots, Z$ geheel. De eis is dat $Aa + Bb + \ldots + Zz + \beta$ kleiner is dan α. De projectie op z_1 van deze som is gelijk aan $Zz_1 + \mu z_1$ waarin μz_1 de projectie van β op z_1 is en μ is een reëel getal (de projectie van $Aa + Bb + \ldots + Yy$ op z_1 is gelijk aan nul). En nu moet $|Zz_1 + \mu z_1|$ kleiner zijn dan α. Dus moeten we een Z zoeken, zo dat

$$\alpha/|z_1| - \mu > Z > -\alpha/|z_1| - \mu$$

Alle andere keuzes maken de term $|Zz_1 + \mu z_1|$ groter of gelijk aan α. Nu gaan we voor alle Z die aan het bovenstaande voldoen het volgende uitvoeren. Bereken de projectie τ van $Zz + \beta$ op span$(A, \ldots, Y)$. Voor elke Z nemen we

$$\text{KLEINSTE}(\sqrt{\alpha^2 - (Z + \mu)^2 |z_1|^2}, \tau, (a, b, \ldots, y))$$

als $\alpha^2 - (Z + \mu)^2 |z_1|^2 > 0$, en $\{\}$ anders.
Dat is een verzameling. Elke vector uit deze verzameling toegevoegd aan $\beta + Zz$ levert een vector kleiner dan α. Dat zijn alle oplossingen van het probleem.

Als de basis van het algoritme kunnen we nemen
KLEINSTE$(\alpha, \beta, ()) = \{0\}$

13 Het opzet van het experiment

De rooster bases die bij het zoeken van de extreme voorbeelden i.v.m. het abc-vermoeden (zoals in [Riele-Montgomery]) voorkomen, zien er als volgt uit:

$([1000p_1], 0, \ldots, 0, [10^6 p_1])$
$(0, [1000p_2], 0, \ldots, 0, [10^6 p_2])$
$\ldots$
$(0, \ldots, 0, [1000p_n], [10^6 p_n])$

waarbij $p_1, \ldots, p_n$ de eerste n priem getallen zijn. Het aantal vectoren is n. Het aantal indices per vector is $p = n + 1$.

Dit is de eerste experimentele basis. Het tweede voorbeeld krijgen we op de volgende manier.
Laat het LLL algoritme op het eerste voorbeeld los. Laat daarna het Reductie algoritme op het resultaat los.
De gekregen basis is het tweede voorbeeld.
Er is nog een mogelijkheid. De laatste vector in de invoer basis wordt gegeven als eerste, de voorlaatste als tweede enz. Dat is het derde voorbeeld.
We gaan meerdere algoritmen op deze voorbeeld bases loslaten.

We gaan de resultaten op de volgende manier vergelijken.

Neem twee uitkomsten en streep alle vectoren die in beide uitkomsten voorkomen weg. Concreter, als $\pm v$ in beide uitkomst-bases zit dan verwijder v uit

beide bases.

Als dat gedaan is dan blijven $a_1, b_1, \ldots, z_1$ en $a_2, b_2, \ldots, z_2$ over. We sorteren, zo dat $|a_1| \leq |b_1| \leq \ldots \leq |z_1|$ en $|a_2| \leq |b_2| \leq \ldots \leq |z_2|$. Als $|a_1| < |a_2|$ dan scoort het eerste algoritme $1:0$. Als verder $|b_2| < |b_1|$ dan wordt het $1:1$. Zo gaan we door. Als $|b_1| = |b_2|$ dan blijft de score ongewijzigd. Uiteindelijk hebben we een score. Bijvoorbeeld $10:3$ zegt dat het eerste algoritme beter is op tien plaatsen. Het tweede algoritme is beter op drie plaatsen. Behalve deze score worden ook maximale en minimale vector lengtes onder `Min:` en `Max:` gedrukt.
In alle algoritmen worden de tellers `aplus` en `amult` ingebouwd met de betekenis: het aantal optellingen en vermenigvuldigingen gebruikt.

14 De implementaties van LLL en Reductie

Het LLL algoritme wordt geïmplementeerd zoals in [Cohen, pagina 86-87] uitgelegd is. Orthogonalisatie wordt daar niet aanvankelijk, maar tijdens het algoritme berekend. Dat spaart wat rekentijd, maar heeft ook een nadeel tot gevolg: bij de verwisseling van twee vectoren moeten ook de twee orthogonalisatie vectoren aangepast worden. Dat is wat Cohen in zijn boek vergeten is. Deze verwisselingsoperatie is helaas vrij duur. Bovendien worden door deze aanpak veel te grote afrondingsfouten gemaakt. Hier ga ik het als volgt doen: ik neem het algoritme over, maar met de orthogonalisatie vantevoren berekend. Tijdens de experimenten leverde deze implementatie van het LLL algoritme dezelfde resultaten als LatticeReduce uit @Mathematica.

Het Reductie algoritme werkt als volgt. Eerst worden alle inprodukten `inp(i,j)` berekend. De rij `goed(i)` heeft de volgende betekenis:

`goed(i)=True` $\Rightarrow$ $\forall j \neq i : Z(i,j)$

In het begin zijn alle `goed(i)=False`.
De vectoren worden na elkaar onderzocht. Eerste vector, tweede vector, ..., tot en met n-de vector, en dan weer van voren af aan.
`LaatsteNietGoed` is de verst weg liggende vector die nog tenminste een keer aan de orde moet komen. Initieel: `LaatsteNietGoed = n`.
Hier is uitgelegd hoe het met de i-de vector gaat: (de i-de vector noem ik soms gewoon i)
Voor alle $k \neq i$ kijken we of i goed t.o.v. k staat. Als i slecht tegen k staat (dus als $Z(i,k)$ niet geldt) dan wordt i gekort tegen k. Daarna worden alle inprodukten `inp(j,k),j=1, ...,n` aangepast. Deze k is mischien niet meer goed en daarom wordt `goed(k)=False`. Dat heeft als gevolg dat k in de volgende ronde nog een keer moet worden aangepast. Daarom wordt `LaatsteNietGoed=k`. Verschillende andere `goed(.)` zijn ook niet meer goed. Dat wordt aangepast.

Zo gaat Reductie door totdat alle `goed(.)=True` zijn.

15 Andere versies van Reductie

Hier worden de twee andere versies van Reductie uitgelegd: Redalg en ReductieS.
De eerste alternatieve versie, die ik Redalg noem, berust op de volgende uitbreiding.
We definiëren de matrix van projecties:(de naam `gproj` komt van "gehele projecties")
`gproj(i,j)` $= [\frac{i \cdot j}{j \cdot j}]$
waarin de i en de j aan de rechter kant voor vector i en vector j staan.
Verder definiëren we
`goed(i)` = $\sum_{i=1}^{n}$ `|gproj(j,i)|`
De taak van goed is om aan te geven welke vector aan de orde komt voor het korten en welke niet. Conceptueel is dat dezelfde taak als die van goed in Reductie. Verder zijn de twee rijen helemaal verschillend.
Merk op dat `goed(i)` minimaal 1 is.
In de loop van het algoritme wordt het altijd gekort tegen vector i met de grootste `goed(i)`.
De initialisatie van Redalg kost niet veel meer dan bij Reductie. Uitvoer vectoren zijn korter en de rekentijd is iets langer. Het verschil is een factor vier in optellingen. Het aantal vermenigvuldigingen van Redalg en Reductie zijn ongeveer gelijk.

De tweede alternatieve versie heb ik ReductieS genoemd. Deze is eigenlijk iets meer dan Reductie. Het verschil zit in het volgende.
Reductie werkt naar de kleinste basis voor elk paar uit de basis.
ReductieS werkt naar de kleinste basis voor elk drietal uit de basis.
De letter S in de naam komt van de in ReductieS ingebouwde invoeg sort. De werkwijze van ReductieS berust op stelling 3 in paragraaf 10.
De uitvoer vectoren van ReductieS zijn korter dan die van Reductie. De rekentijd is langer. Het nadeel is dat de rekentijd zonder initialisatie $O(n^3)$ is, omdat elk drietal bekeken moet worden.

16 De experimenten

De experimenten zijn gemaakt op enkele voorbeeld bases. In het vergelijken van de algoritmen gebruik ik steeds de volgende criteria: kortere uitvoer vectoren in kortere tijd. Het eerste criterium noem ik de kwaliteit en het tweede criterium de rekentijd.

Het is belangrijk om op te merken dat de rekentijd uitgedrukt is in het aantal $+, -, *, /$ operaties. Geen rekening is gehouden met de groei van de lengte van de getallen als functie van de invoer lengte.

Het LLL algoritme levert meer dan korte vectoren in de acceptabele tijd. Ten eerste berekent het LLL algoritme de orthogonalisatie. Deze orthogonalisatie is nodig voor het berekenen van de kleinste vectoren in een rooster en zorgt ook voor de gunstige theoretische schattingen. Verder kan het LLL algoritme

de overbodige lineair afhankelijke vectoren detecteren en uitschakelen (zoals uitgelegd in [Pohst]). Dat is wat Reductie niet kan.

Het is dus niet vreemd dat Reductie soms beter is wat de twee criteria betreft.

Het berekenen van de orthogonalisatie bij het LLL algoritme noem ik de initialisatie. De initialisatie bij Reductie bedraagt het berekenen van de inprodukten.

Het voorbeeld in de paragraaf 13 is een ijle basis van het soort dat vaak in verband met het zoeken van lineaire afhankelijkheid relaties voorkomt.

In de eerste instantie maak ik geen gebruik van het ijl zijn van de basis. De meeste tijd gaat naar de initialisatie. Zo eist LLL $2n^3$ initialisatie operaties terwijl Reductie aan n^3 genoeg heeft. De rest van de algoritmen eist (empirisch, $n < 240$) $O(n^2)$ operaties met constanten van de orde 10 tot 100.
Later werd een uitvoer van Reductie aan LLL geboden. Ik vind het interessant dat vectoren daardoor langer worden.
In het volgende experiment krijgen de algoritmen LLL en Redalg de omkering van de eerste experiment invoer. (Redalg is een implementatie van Reductie.) Met deze invoer heeft het LLL algoritme meer moeite.
Het geval $n = 750$ wordt als volgt gedaan. De invoer van het experiment is een ijle basis. Als we daarvan gebruik maken dan eist de initialisatie bij Reductie een $O(n^2)$ tijd met een constante van orde 10.

Het LLL algoritme moet hiervoor aangepast worden. De reden is dat de orthogonalisatie zoals in het LLL algoritme voorgeschreven is een $O(n^3)$ tijd vraagt (weliswaar met lagere constanten dan vroeger). Dat is trager dan bij Reductie. Een algoritme uit [Cohen, pagina 100] gebruikt een andere orthogonalisatie om dit probleem op te lossen. Een soortgelijk idee is hier gebruikt om het LLL algoritme aan te passen.

16.1 Het vergelijken van LLL en Reductie

De experimenten zijn gemaakt voor enkele waarden van n. Hier zijn de uitkomsten:

	Reductie						LLL						Reductie:LLL
	#operaties X1000		ratio				#operaties X1000		ratio				
n	mult	plus	mult	plus	min	max	mult	plus	mult	plus	min	max	
30	29	53	16.3	42.8	6040	11230	38	45	11.7	18.5	5511	6689	0:29
35	41	73	15.2	40.2	6040	9476	58	65	11.3	17.3	5511	6849	0:34
55	130	206	14.4	38.5	6040	11326	202	215	10.5	14.7	5511	8197	0:54
65	201	305	14.1	37.8	6040	11990	324	340	10.3	14.0	5511	8197	0:64
75	295	433	13.9	37.5	6040	10204	487	506	10.1	13.4	5511	8197	0:74
85	411	583	13.4	36.2	6040	11453	697	719	10.0	13.0	5511	8197	0:84
95	556	766	13.1	35.3	6040	11535	960	986	9.9	12.7	5511	8573	0:94
120	1056	1375			6040	12483	1889	1939			5511	8573	0:119
240	7539	8609			6040	14339	14449	14607			5511	9383	0:239

Legende:

n is het aantal vectoren in de basis.
#operaties X1000 is het aantal operaties in duizenden.
mult zijn de vermenigvuldigingen.
plus zijn de optellingen.
ratio is het aantal operaties zonder initialisatie gedeeld door n^2
Reductie:LLL is de score, uitgelegd in paragraaf 13.

Aan de hand van de resultaten van de vergelijking in dit geval kunnen we het volgende zeggen:
1. Kwaliteit: Het LLL algoritme levert kortere vectoren dan het Reductie algoritme.
2. Rekentijd: Het Reductie algoritme is twee keer sneller dan het LLL algoritme.

Het initialisatie deel van Reductie eist $\frac{1}{2}pn^2$ optellingen en $\frac{1}{2}pn^2$ vermenigvuldigingen. Het initialisatie deel van LLL eist pn^2 optellingen en pn^2 vermenigvuldigingen. (p is het aantal indices per vector; hier $p = n + 1$)

In geval van dit voorbeeld lijkt het dat voor $n < 240$ beide algoritmen kwadratisch zijn, als we de initialisatie niet meetellen. Bij grote n, bijvoorbeeld $n = 240$, zien we deze kwadratische term bijna niet meer. Zo wordt Reductie twee keer sneller.
Het lijkt dat de hypothetische constanten bij de kwadratische term dalen. Dat zien we onder *ratio*.
De vraag is of dit gedrag zich voortzet als n groeit.

Deze analyse is gemaakt voor een algemene basis. We zien dat we hier met een initialisatie probleem te maken hebben. Geen gebruik is gemaakt van het feit dat het voorbeeld een ijle basis is. Initialisatie kan echter veel sneller gaan bij Reductie in dit geval en kost $O(n^2)$. Het ijl zijn van de basis kan gebruikt worden om de initialisatie van het LLL algoritme te versnellen, maar dat blijft nog steeds een $O(n^3)$ tijd.

16.2 De samenstelling van Reductie en LLL

LLL wordt toegepast. Op het resultaat van LLL wordt Reductie losgelaten. Daarna werkt LLL er nog een keer op.
De resultaten:

	LLL		Reductie(LLL)		LLL:Reductie(LLL)	LLL(Reductie(LLL))		Reductie(LLL): LLL(Reductie(LLL))
n	min	max	min	max		min	max	
35	5511	6849	5511	6849	0:9	5511	7269	25:1
45	5511		5511	7283	0:10	5511	7785	33:1
55	5511	8197	5511	8197	0:13	5511	7950	40:2

De conclusies:
We hebben in vorige paragrafen gezien dat LLL kortere vectoren in dit soort voorbeelden levert. Nu zien we dat Reductie de resultaten van LLL verbetert. Maar dat is niet alles. Wat hier opvalt is dat LLL in het nadeel van het resultaat

werkt als het na Reductie losgelaten wordt. Het levert namelijk de uitvoer die langer is dan de invoer.
We hebben hier dus een ander soort voorbeelden gegenereerd, waarin Reductie een beter algoritme is dan LLL, wat de lengte van de uitvoer vectoren betreft. Het gebeurde vaak tijdens de experimenten dat meervoudige composities van LLL en Reductie een steeds beter resultaat leverden. Dit is een verschijnsel bekend bij de neurale netwerken: het algoritme gaat niet rechtstreeks naar de oplossing, maar keert steeds terug om een betere omweg te vinden.

16.3 De omkering van de basis: het vergelijken van LLL en Redalg

In deze paragraaf gaan we de omkering van de invoer aan de algoritmen serveren. De eerste invoer vector wordt nu de laatste, de tweede wordt de voorlaatste enz. Het algoritme Reductie levert in deze situatie een langer resultaat.
Een andere versie van Reductie, namelijk Redalg, maakt de volgorde van de invoer niet veel uit. Hier vergelijken we LLL en Redalg[1].

	LLL						Redalg						Redalg:LLL
	#operaties X1000		ratio				#operaties X1000		ratio				
n	mult	plus	mult	plus	min	max	mult	plus	mult	plus	min	max	
20	31	61	56.84	130.99	5970	8510	16	54	29.725	118.845	5511	7600	17:0
25	47	81	49.1	103.7	6858	9114	26	82	27.7	112.9	5838	7576	25:0
35	114	188	57.2	117.0	6301	10082	70	219	38.3	154.6	5676	8346	34:0
45	222	337	63.4	120.3	6301	9734	130	385	39.7	160.6	5838	8479	43:0
55	366	537	64.7	121.1	6	9228	219	634	42.9	174.9	6356	9800	0:55
65	570	806	68.5	124.3	6161	10038	339	953	45.9	186.1	5839	10365	63:1
75	818	1127	69.0	123.8	6147	9812	506	1404	50.5	205.0	5676	9575	74:0
85	1154	1547	73.3	127.6	6334	10206	680	1813	49.7	201.4	5676	9525	81:0
120	2997	3796	86.6	142.1	6161	10333	1557	3650	46.1	186.4	5985	10377	79:38
240	20249	23392	110.0	164.6	6147	11989	9724	18171	46.8	188.4	6124	10594	211:25
750	(50576)	(61943)	killed: k was gelijk aan:460				(26496)	(106890)	45.6	183.0			

De conclusies:
Het LLL algoritme heeft met deze omkering veel meer moeite.
1. Kwaliteit: Redalg levert de kortere vectoren dan LLL behalve in het verrassende geval van $n = 55$.
2. Rekentijd: Redalg lijkt twee keer sneller te zijn dan LLL in vermenigvuldigingen. Eigenlijk verwacht ik dat voor grote n Redalg gewoon twee keer sneller is, net als Reductie.

Maar dat hangt af van de volgende, de meest interessante open vraag van deze tekst:
hoe gedragen zich hypothetische constanten bij de kwadratische term in de complexiteit analyse? Het zou mooi zijn als de daling van `ratio` van Redalg bij $n = 240$ naar $n = 750$ zich voortzette. Een andere mogelijkheid is dat het

[1] Voor $n = 750$ is in plaats van het LLL algoritme, het algoritme uit [Cohen, pagina 100] gebruikt. Helaas is het rekenen van deze niet voltooid. De ratio van Redalg is het enige gegeven in de $n = 750$ rij dat precies overeenkomt met de overige rijen.

algoritme "ontploft" bij een grote n.

Neem hier verder dat $p = n$.
Het is duidelijk dat initialisatie van Redalg een $c_1 n^3 + O(n^2)$ tijd kost. Met bekende constanten: $c_1 = \frac{1}{2}$ voor vermenigvuldigingen en evenzo voor de optellingen.
Nu gaan we aannemen dat een vector in de basis niet vaker dan C keer gekort wordt. Dat levert maximaal Cn kortingen die kunnen voorkomen. Bij elke korting hoort een $O(n)$ onderhoud met lage constanten (van orde 10) die we bij C toevoegen $C := 10C$. We hebben dus in totaal Cn^2 operaties nodig.
De constante C uit dit voorbeeld gaat niet ver van 45 en 185 voor vermenigvuldigingen en optellingen, respectievelijk. Enige probleem is dat we theoretisch niets expliciet voor C hebben.

17 Redalg geïtereerd: het nieuwe algoritme voor het zoeken van de kleinste vectoren in een rooster

Opmerking: deze laatste paragraaf maakt geen onderdeel uit van mijn afstudeerwerk. Bovendien blijven alle auteurs rechten behouden.

Algoritme Redalg is niet in staat om te bepalen of een stelsel vectoren afhankelijk is.

Dat wordt gebruikt om Redalg iteratief te laten werken op de volgende manier.

Gegeven is een basis $a, b, \ldots, z$. Met deze basis als uitvoer levert Redalg een gereduceerde basis $a_1, b_1, \ldots, z_1$. We voegen $2a_1, 2b_1, \ldots, 2z_1$ er bij. De nieuwe invoer is $a_1, b_1, \ldots z_1, 2a_1, 2b_1, \ldots, 2z_1$. Zo gaan we door: uitvoer vectoren samen met alle tweevouden van de uitvoer vectoren wordt de nieuwe invoer.

Het experiment is uitgevoerd op de basis uit paragraaf 13. De eerste kolom bevat de lengtes van de vectoren uit LLL uitvoer basis. De tweede kolom zijn de lengtes van de eerste n kleinste vectoren in het rooster. De derde kolom bevat de kleinste n vectoren van de eerste iteratie. In de overige kolommen staan de lengtes van de kleinste vectoren uit de tweede, de derde en de vierde iteratie. De letters L en R in de tweede kolom geven aan of een vector gevonden was door het LLL algoritme en/of door Redalg respektievelijk. In de laatste regel staat het aantal vectoren in het stelsel na de iteratie. De exponentiële groei van het aantal vectoren gebeurt blijkbaar niet, omdat elke keer steeds meer vectoren nul worden. De initialisatie van elke iteratie kan snel gedaan worden aan de hand van de inprodukten uit de vorige iteraties.

Hier zijn de resultaten die wijzen aan dat dit een algoritme voor het zoeken van de kleinste vectoren in een rooster is.

	$n = 10$						
LLL	kleinste			Redalg iteratie			
5511.75	L	5511.75	R1	5511.75	5511.75	5511.75	5511.75
5676.84		5654.22	R1	5654.22	5654.22	5654.22	5654.22
5765.28	L	5676.84	R1	5676.84	5676.84	5676.84	5676.84
5859.61		5743.31	R1	5743.31	5743.31	5743.31	5743.31
5938.77	L	5765.28	R2	5839.45	5765.28	5765.28	5765.28
6025.99		5839.45	R1	5938.77	5839.45	5859.61	5839.45
6147.63	L	5859.61	R3	5970.82	5938.77	5938.77	5859.61
6161.69	L	5938.77	R1	6001.09	5970.82	5970.82	5938.77
6461.89		5970.82	R1	6025.99	6001.09	6001.09	5970.82
6512.49		5975.11		6060.93	6060.93	6097.12	6001.09
				14	16	16	17

	$n = 20$						
LLL	kleinste			Redalg iteratie			
5511.75	L	5511.75	R1	5511.75	5511.75	5511.75	5511.75
5676.84		5654.22	R1	5654.22	5654.22	5654.22	5654.22
5765.28	L	5676.84	R1	5676.84	5676.84	5676.84	5676.84
5844.99		5743.31	R1	5743.31	5743.31	5743.31	5743.31
5859.61	L	5765.28	R3	5839.45	5839.45	5765.28	5765.28
5915.39		5838.44	R3	5844.99	5844.99	5838.44	5838.44
5938.77		5839.45	R1	5938.77	5859.61	5839.45	5839.45
5939.21	L	5844.99	R1	5939.21	5915.39	5844.99	5844.99
6025.99	L	5859.61	R2	5970.82	5938.77	5859.61	5859.61
6147.63	L	5915.39	R2	6046.85	5939.21	5915.39	5915.39
6151.58	L	5938.77	R1	6089.68	5970.82	5938.77	5938.77
6161.69	L	5939.21	R1	6097.12	6014.64	5939.21	5939.21
6224.24		5970.74	R4	6151.58	6025.99	5970.82	5970.74
6299.01		5970.82	R1	6153.37	6033.35	5990.55	5970.82
6452.14		5975.11		6168.59	6040.87	6025.99	5990.55
6461.89		5985.47		6188.86	6046.85	6033.35	6025.99
6512.49		5990.55	R3	6209.66	6049.46	6040.87	6033.35
6520.62		6001.09		6212.02	6097.12	6046.85	6040.87
6548.46		6014.64	R2	6275.21	6151.58	6049.46	6046.85
6615.52	L	6025.99	R2	6305.2	6153.37	6076.9	6049.46
				30	40	51	52

	$n = 30$						
LLL	kleinste			Redalg iteratie			
5511.75	L	5511.75	R1	5511.75	5511.75	5511.75	5511.75
5676.84		5654.22	R1	5654.22	5654.22	5654.22	5654.22
5765.28	L	5676.84	R2	5939.21	5676.84	5676.84	5676.84
5844.99		5743.31	R2	6001.09	5743.31	5743.31	5743.31
5859.61	L	5765.28	R3	6025.99	5839.45	5765.28	5765.28
5915.39		5838.44	R3	6040.87	5844.99	5838.44	5838.44
5938.77		5839.45	R2	6089.68	5915.39	5839.45	5839.45
5939.21	L	5844.99	R2	6098.45	5938.77	5859.61	5844.99
6025.99	L	5859.61	R3	6124.91	5939.21	5915.39	5859.61
6147.63	L	5915.39	R2	6183.21	5970.74	5938.77	5915.39
6151.58	L	5938.77	R2	6188.86	5970.82	5939.21	5938.77
6161.69	L	5939.21	R1	6191.25	5985.47	5970.74	5939.21
6191.25		5970.74	R2	6209.71	5990.55	5985.47	5970.74
6224.24		5970.82	R2	6214.79	6001.09	5990.55	5970.82
6299.01		5975.11		6223.48	6025.99	6001.09	5985.47
6324.93		5985.47	R2	6263.75	6040.87	6025.99	5990.55
6384.65		5990.55	R2	6264.59	6046.85	6040.87	6001.09
6452.14		6001.09	R1	6273.98	6076.9	6046.85	6025.99
6461.89		6014.64		6275.21	6089.68	6049.46	6033.35
6466.71	L	6025.99	R1	6295.22	6097.12	6076.9	6046.85
6502.11		6033.35	R4	6305.2	6124.91	6089.68	6049.46
6512.49		6040.87	R1	6356.9	6151.58	6097.12	6076.9
6513.11		6046.85	R2	6415.43	6153.37	6101.95	6089.68
6520.62		6049.46	R3	6417.67	6183.21	6124.91	6097.12
6529.01		6059.63		6424.04	6186.68	6151.58	6101.95
6548.46		6060.93		6443.06	6188.86	6153.37	6124.91
6615.52		6076.9	R2	6458.36	6191.25	6162.59	6151.58
6651.51		6089.68	R1	6554.82	6209.66	6168.59	6153.37
6677.62		6097.12	R2	6600.71	6209.71	6183.21	6162.59
6689.08		6098.45	R1	6681.43	6212.02	6188.86	6168.59
				47	64	73	73

Appendix

A De programmatekst van het LLL algoritme

```
Clear[LLLalg]
LLLalg[b_]:=Block[{bb,n,k,kmax,bster,Bgroot,mu,l,
q,temp,i,j,m,Bg,t,p,aplus,amult,aplusinit,amultinit },

(* tellers*)
aplus = 0;
amult = 0;


(* variabelen *)
bb = b;
n = Length[bb];
p = Length[bb[[1]] ];
bster = Table[0,{n},{p} ];
Bgroot = Table[0,{n}];
mu = Table[0,{n},{n}];

(* initialisatie *)
bster[[1]] = bb[[1]];   aplus+=p;
Bgroot[[1]] = N[Dot[bb[[1]],bb[[1]]]];  aplus+=p-1;amult+=p;

For[k=2,k<n+1,k++,      (*aplus+=2*)
 bster[[k]] = bb[[k]];  aplus+=p;
 For[j=1,j<k,j++,       (*aplus+=2;*)
  mu[[k,j]]=N[Dot[bb[[k]],bster[[j]]]/Bgroot[[j]]]; amult+=p+1;aplus+=p-1;
  bster[[k]] = bster[[k]]-mu[[k,j]] bster[[j]]; aplus+=p;amult+=p;
 ];
 Bgroot[[k]] = Dot[bster[[k]],bster[[k]]]; aplus+=p-1;amult+=p;
];(*For*)
(*Print[bster];
Print[{mu,Bgroot}];
Return[{mu,Bgroot}];*)
Print[''LLL-init:$mult:'',amult,'' $plus:'',aplus];
amultinit = amult;
aplusinit = aplus;
(**)
(*algoritme*)
k = 2;                  aplus++;
(**)
While[k<n+1,    aplus++;
 NogMeer = True;        aplus++;
 While[NogMeer,         aplus++;
  For[l=k-1,l>0,l--,    aplus+=2;

   If[Abs[mu[[k,l]]]>0.5,       aplus++;
    q = Round[mu[[k,l]]];       aplus++;
    bb[[k]] = bb[[k]] -q bb[[l]]; aplus+= p; amult+=p;
    mu[[k,l]] = mu[[k,l]]-q;    aplus++;
    For[i=1,i<l,i++,            aplus+=2;
     mu[[k,i]] = mu[[k,i]]-q mu[[l,i]]; aplus++;amult++;
    ];
   ];
  ];(*For*)

  aplus++;amult+=3;
  If[Bgroot[[k]] < (0.75 - mu[[k,k-1]]^2) Bgroot[[k-1]],
   temp = bb[[k]];
       aplus+=3;(*verwisseling m.b.v. pointers!*)
   bb[[k]] = bb[[k-1]];
   bb[[k-1]] = temp;
```

```
      (*dit kan niet met pointers!*)
   For[j=1,j<k-1,j++,                    aplus+=2;
    temp = mu[[k,j]];                    aplus++;
    mu[[k,j]] = mu[[k-1,j]];             aplus++;
    mu[[k-1,j]] = temp;                  aplus++;
   ];
   m = mu[[k,k-1]];              aplus++;
   Bg = Bgroot[[k]] + m^2 Bgroot[[k-1]];        aplus++;amult+=2;
   mu[[k,k-1]] = m Bgroot[[k-1]]/Bg;            amult+=2;
   Bgroot[[k]] = Bgroot[[k]] Bgroot[[k-1]]/Bg;
   Bgroot[[k-1]] = Bg;                          aplus+=1;
   For[i=k+1,i<n+1,i++,                         aplus+=2;
    t = mu[[i,k]];                              aplus+=1;
    mu[[i,k]] = mu[[i,k-1]] -m t;               aplus++;amult++;
    mu[[i,k-1]] = t+mu[[k,k-1]] mu[[i,k]];      aplus++;amult++;
   ];(*For*)
   k = Max[2,k-1];aplus++
  ,(*Else*)
   NogMeer= False;      aplus++;
   k=k+1;       aplus++;
  ]; (*IF*)
 ]; (*While NogMeer*)
];(* Alle grootste While*)
Print[''LLL:#mult:'',amult,'' #plus:'',aplus];
Print[''ratio:#mult:'',N[(amult-amultinit)/n^2],
'' #plus:'',N[(aplus-aplusinit)/n^2]];
bb
](*Block*)
```

B De programmatekst van het Redalg algoritme

```
Clear[Redalg]
Redalg[l_]:= Block[{inp,n,v,goed,gproj,k,i,m,p,aplus,amult,max,
amultinit,aplusinit},
(*aplus en amult zijn het aantal optellingen en vermenigvuldigingen*)
(*p is het aantal indices per vector*)
aplus=0;amult=0;
p = Length[ l[[1]] ];
(*Initialisatie*)
v=l;
(*aantal vectoren*)
n = Length[l];                  aplus++;
(*gproj[[i,j]] is de afgeronde projectie van i op j*)
gproj = Table[0,{n},{n}];
(**)
(*goed[[i]] staat voor: sum(ABS[gproj[[j,i]]])*)
(* goed[[i]] is minimaal 1 *)
goed = Table[0,{n}];            aplus += n;
(**)
(*inproducten declaratie*)
inp = Table[0,{n},{n}];
(*inproducten initialisatie*)
(**)
For[i=1,i<n+1,i++,              aplus+=2;
 For[k=i,k<n+1,k++,             aplus+=2;
  inp[[k,i]] = N[Dot[ v[[i]], v[[k]] ]];        aplus+=p-1;amult+=p;
  inp[[i,k]] = inp[[k,i]];      aplus++;
 ](*For*)
];(*For*)
(*gproj en goed initialisatie*)
(**)
For[i=1,i<n+1,i++,              aplus+=2;
 For[k=1,k<n+1,k++,             aplus+=2;
  gproj[[k,i]]=Round[inp[[k,i]]/inp[[i,i]]];aplus++;amult++;
  goed[[i]]+=Abs[gproj[[k,i]]]; aplus+=2;
```

```
 ](*For*)
];(*For*)
Print[''Redalg-init:#mult:'',amult,'' #plus:'',aplus];
amultinit = amult;
aplusinit = aplus;
(*Algoritme*)
(**)
While[True,
 (*zoek de grootste goed[[i]]*)
 max = 1;
 For[i=1,i<n+1,i++,                aplus+=2;
  If[goed[[i]]>goed[[max]],max=i;aplus++];              aplus++;
 ];
 (**)
 If[goed[[max]]==1,Break[]]; aplus++;
 (**)
 For[k=1,k<n+1,k++,  aplus+=2;
  m = gproj[[k,max]];            aplus++;
  If[m != 0,                     aplus++;
   If[k != max,                 aplus++;
    v[[k]] = v[[k]] - m v[[max]]; (*v[[k]] wordt korter tegen v[[max]]*)
              aplus+=p;amult+=p;
    (*Inproducten worden aangepast.*)
    inp[[k,k]] = inp[[k,k]]-2 m inp[[k,max]] + m m inp[[max,max]];
                      aplus+=2;amult+=4;
    For[i=1,i<n+1,i++,                   aplus+=2;
     If[i!=k,                            aplus++;
      inp[[i,k]] = inp[[i,k]]- m inp[[i,max]];
      inp[[k,i]] = inp[[i,k]];
                aplus+=2;amult++;
      If[inp[[i,i]]>0,                                      aplus++;
       nprki =  Round[inp[[k,i]]/inp[[i,i]]];               aplus++;amult++;
      ,nprki = 0;       aplus++ ];
      If[inp[[k,k]]>0,                                      aplus++;
       nprik =  Round[inp[[k,i]]/inp[[k,k]]];               aplus++;amult++;
      ,nprik = 0;       aplus++ ];
      goed[[i]] = goed[[i]]-Abs[gproj[[k,i]]]+Abs[nprki];       aplus+=3;
      goed[[k]] = goed[[k]]-Abs[gproj[[i,k]]]+Abs[nprik];       aplus+=3;
      gproj[[k,i]] = nprki;                                     aplus++;
      gproj[[i,k]] = nprik;                                     aplus++;
     ];(*If*)
    ];(*For*)
   ];(*If[ k!=max,*)
  ];(*If[m!=0,*)
 ];(*For[k*)
];(*While*)
Print[''Redalg:#mult:'',amult,'' #plus:'',aplus];
Print[''ratio:#mult:'',N[(amult-amultinit)/n^2],
'' #plus:'',N[(aplus-aplusinit)/n^2]];
Complement[v,{Table[0,{p}]}]
](*Block*)
```

Referenties

[Cohen] H. Cohen,
A Course in computational Algebraic Number Theory,
Graduate texts in Math. 138, Springer-Verlag 1993.

[Conway-Sloane] J.H. Conway en N.J.A. Sloane,
Sphere Packings, Lattices and Groups,
Grundlehren der matematischen Wissenschaften 290,
Second Edition, Springer-Verlag 1993.

[Fincke-Pohst] U. Fincke en M. Pohst,
Improved methods for calculating vectors of short length in a lattice, including a complexity analysis,
Math. Comp. 44 (1985), 463-471.

[HJLS] J. Hastad, B. Just, J.C. Lagarias en C.P. Schnorr,
Polynomial time algorithms for finding integer relations among real numbers,
Siam J. Comput. 18 (1989), 859-881.

[Kallen] Wilberd van der Kallen,
Lenstra Lenstra Lovász implementations,
http://www.math.ruu.nl/people/vdkallen/kallen.html
Last updated: Feb 3 1996

[LLL] A.K. Lenstra, H.W. Lenstra en L. Lovász,
Factoring polynomials with rational coefficients,
Math. Ann. 261 (1982), 515-534.

[Pohst] M. Pohst,
A Modification of the LLL-algorithm,
J. Symb. Comp. 4 (1987), 123-128

[Riele-Montgomery] Herman te Riele en Peter Montgomery,
Numerical experiments on the abc-conjecture using the LLL-algorithm,
Stieltjes Institute Workshop on
Constructive Methods for Diophantine Equations and
Elliptic Curves, Rotterdam, 31 may 1994 (manuscript)