From QR Factorization to Wireless Communication

Contact Information
Ron Estrin
ICME PhD Candidate
Email: restrin@stanford.edu

Ron Estrin and Jack Poulson
Institute for Computational and Mathematical Engineering
Stanford University

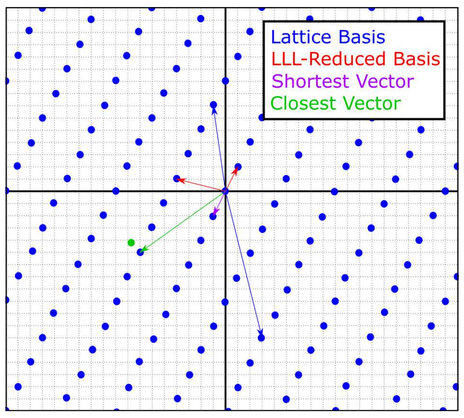
Contact Information
Jack Poulson
Google Research
Email: jpoulson@google.com

Abstract

Despite the numerous applications of lattice reduction (e.g., in MIMO systems, algebraic number theory and lattice-based cryptography) and the close connection between the Lenstra-Lenstra-Lovasz (LLL) algorithm and column-pivoted QR factorizations (through generalizing from permutations to unimodular transforms), the well-established techniques for exploiting level 3 BLAS within Householder QR have not yet been extended. We therefore propose a novel right-looking variant of a Householder-based LLL which both accumulates Householder transformations where possible and avoids redundant applications when the Lovasz condition requires column swaps.
We provide benchmarks of the new scheme, a tree-based, recursive extension, against the corresponding implementations from the popular NTL and FPLLL libraries.

Introduction

Lattice : Group closed under linear combinations with integer coefficients. E.g. $x = Bv$ with $B \in \mathbb{Z}^{m\times n}$ (blue points below).



A lattice is generated by some **basis** (blue vectors).
Shortest vector problem (SVP): Find the shortest vector in the lattice (purple vector).
Closest vector problem (CVP): Find the closest vector in lattice to given point (green vector is closest to green point).

Some Applications

MIMO Detection

MIMO (Multiple-input and multiple-output) methods use multiple transmitters and receive antennas for wireless communication. A signal $x \in (\mathbb{Z} + i\mathbb{Z})^n$ is transmitted by n transmitters and received by m receivers. Signal: $y = Hx + w$, $H \in \mathbb{C}^{m\times n}$ is the channel matrix, w is noise. To decode, solve CVP

$$\min_{x\in(\mathbb{Z}+i\mathbb{Z})^n} \|y - Hx\|_2.$$

Lattice-Based Cryptography

Lattice based cryptosystems developed based on the hardness of SVP (e.g. NTRU) and CVP (e.g. GGH). Bases are typically of dimension $n \in O(100)$, with column norms typically $\sim 10^{300}$, while the shortest vectors have norms ~ 1000.

The Lenstra-Lenstra-Lovász (LLL) Algorithm

Given a basis B for the lattice Λ, LLL produces a 'nice' basis $B' = BU$, where U is unimodular (see red vectors in left figure). The first column of B' is an approximation of the shortest vector. B' is a more suitable basis for computation for SVP and CVP, producing better approximate solutions than using B directly.
LLL also takes a parameter $\delta \in (\frac{1}{4}, 1)$ which controls how 'reduced' the basis will be, with δ larger typically implying a better reduced basis. LLL is defined for $\delta = 1$, but polynomial-time complexity is guaranteed only for $(\frac{1}{4}, 1)$.
The LLL algorithm can be thought of as forming a factorization that is a generalization of a Column-Pivoted QR.

Column-Pivoted QR

Input: Matrix A
Output: Factorization $AP = QR$ where:
- P is a permutation matrix
- Q is an orthogonal matrix
- R is upper triangular such that:

$$|r_{11}| \geq \cdots \geq |r_{nn}|$$

LLL

Input: Matrix B and $\delta \in (\frac{1}{4}, 1)$
Output: Factorization $BU = QR$ where:
- U is a unimodular matrix: $U \in \mathbb{Z}^{n\times n}$ and $\det U = \pm 1$
- Q is an orthogonal matrix
- R is upper triangular such that:
  - **(Size-Reduction Property)** $|r_{ji}/r_{jj}| \leq 1/2$ for $j < i$
  - **(Lovász Condition)** $\delta r_{k-1,k-1}^2 \leq (1 + r_{k,k-1}^2) r_{kk}^2$ for all k

Size reduction property ensures basis vectors are as orthogonal as possible. **Lovász condition** determines if swapping columns improves basis.

Outline of LLL Algorithm

while k < n **do**
  Update QR of $B_{:,1:k}$
  Size reduce b_k against $b_1, \ldots, b_{k-1}$
  Update R after size-reduction
  if Lovász condition holds for column k **then**
    $k \leftarrow k + 1$
  else
    Swap columns b_{k-1}, b_k
    $k \leftarrow k - 1$
  end if
end while

Challenges with LLL

- Entries are so large arbitrary precision datatypes are needed. These are slow so avoiding their use as much as possible while retaining accuracy desired.
- Want to keep basis in full precision, and QR factorization in lower precision while maintaining accuracy.
- Right-looking schemes suffer from catastrophic cancellation if not careful.
- Many swaps can lead to redundant computation, want cheap accurate updates.

Our Contributions

With LLL's close connection to CPQR, we take advantage of well known techniques for exploiting level 3 BLAS to implement an efficient LLL algorithm. In particular:

- We implement a few versions of the LLL algorithm (including deep insertion and deep reduction) and BKZ 2.0 (Blockwise Korkine-Zolotarev algorithm) into the open-source library **Elemental** [4].
- One implementation includes an efficient left-looking Householder based implementation, which updates the QR factorization column-wise using Householder transformations.
- Another implementation includes a right-looking Householder-Givens hybrid based implementation. In routines with few swaps we take advantage of level 3 BLAS panel Householder updates to the entire R factor. If swaps are encountered, the QR factorization is updated via a single Givens rotation. Blocking of Givens rotations reduces redundant computation. Using both interpolates between swap-light and swap-heavy regimes by taking advantage of efficient level 3 BLAS techniques and repeated cheap Givens updates.
- More tricks to maintain accuracy of QR and improve efficiency.
- We develop a tree-based recursive variant of LLL (à la Mergesort). It is particularly useful for quickly reducing entry sizes to allow for transition to cheaper datatypes (doubles vs. BigFloat) without reducing the entire basis using heavy-duty datatypes.

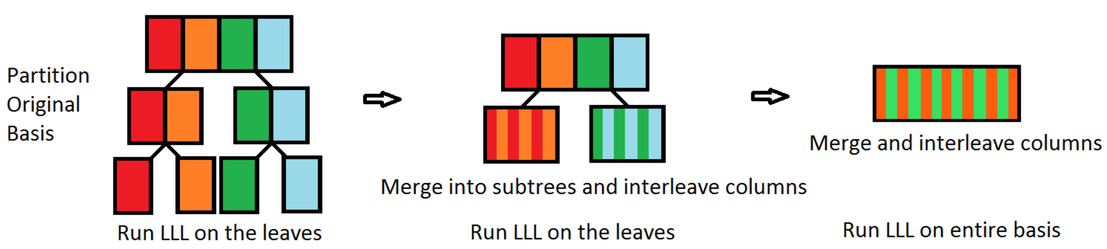
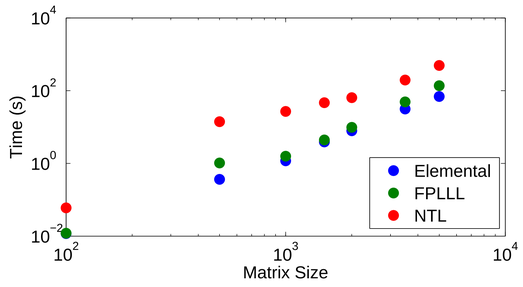


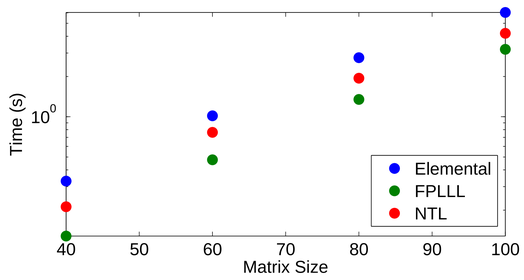
Figure: Schematic of the tree-based recursive LLL.

Numerical Experiments

We compare our right-looking hybrid implementation against the FPLLL [2] and the NTL [5] implementations of LLL in two regimes: Small (e.g. MIMO or nearly reduced bases) and large (e.g. cryptographic bases) entry bases.
The first figure compares implementations on knapsack type bases of size $(n + 1) \times n$ with 20-bit entries.



Our implementation is typically faster in this regime where relatively fewer swaps are required, and BLAS 3 techniques can be exploited.
The next figure compares implementations on the SVP Challenge [1] problem matrices. Column norms are $\|b_i\|_2 \in [10^{100}, 10^{300}]$.



Our implementation is slower, but performs many more swaps as well, producing a shorter first vector. FPLLL and NTL are particularly good at reducing large column norms quickly but crudely.

Future Work

- Develop heuristics for fast entry size reduction to switch to low-precision regime.
- More improvements required to accurately and quickly work with matrices with large entries.
- Already have working BKZ implementation for solving SVP problems. Further work explores way to improve the enumeration procedure for faster runtimes by exploiting y-sparsity [3].

References

[1] SVP Challenge.
http://www.latticechallenge.org/svp-challenge/.

[2] M. Albrecht, S. Bai, D. Cadé, X. Pujol, and D. Stehlé.
fplll-4.0, a floating-point LLL implementation.
Available at http://perso.ens-lyon.fr/damien.stehle.

[3] Dan Ding and Guizhen Zhu.
A random sampling algorithm for svp challenge based on y-sparse representations of short lattice vectors.
In *Proceedings of the 2014 Tenth International Conference on Computational Intelligence and Security*, CIS '14, pages 425–429, Washington, DC, USA, 2014. IEEE Computer Society.

[4] Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, and Nichols A. Romero.
Elemental: A new framework for distributed memory dense matrix computations.
ACM Trans. Math. Softw., 39(2):13:1–13:24, February 2013.

[5] Victor Shoup.
NTL: A Library for doing Number Theory.
http://www.shoup.net/ntl/.
Version: 9.6.2.