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-Lovász (LLL) algorithm and column-pivoted QR factorizations (through generalizing from permutations to unimodular forms), 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 Lovász 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).

**Shortest vector problem (SVP)**: Find the shortest vector in the lattice (purple vector).

**Closest vector problem (CVP)**: Find the closest vector to a 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 \( x \in \mathbb{C}^{m \times n} \) is transmitted by \( m \) transmitters and received by \( m \) receivers. Signal: \( y = Hx + w \), where \( H \in \mathbb{C}^{m \times n} \) is the channel matrix, \( w \) is noise. To decode, solve CVP:

\[
\min_{x \in \mathbb{C}^{m \times n}} \| y - Hx \|_2^2.
\]

**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 \( 10^{30} \), while the shortest vectors have norms \( 10^{1000} \).

### Numerical Experiments

We compare our right-looking hybrid implementation against the FPLL [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.

### 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 RI 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 Givens updates.
- More tricks to maintain accuracy of QR and improve efficiency.
- We develop a tree-based recursive variant of LLL (à la Mergesoit). It is particularly useful for quickly reducing entry sizes to allow for transition to cheaper data types (doubles vs. BigFloat) without reducing the entire basis using heavy-duty data types.

### 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 working BKZ implementation for solving SVP problems. Further work explores ways to improve the enumeration procedure for faster runtimes by exploiting y-sparcity [3].

### References

http://www.latticechallenge.org/svp-challenge/


A random sampling algorithm for svp challenge based on y-sparse representations of short lattice vectors.


NTL: A Library for doing Number Theory.
http://www.shoup.net/ntl/
Version 6.2.2.