JSIAM Letters Vol.9 (2017) pp.65–68 ©2017 Japan Society for Industrial and Applied Mathematics JSIAM Letters

Application of mixed integer quadratic program to shortest vector problems

Keiji Kimura[1], Hayato Waki[2] and Masaya Yasuda[2]

[1] Graduate School of Mathematics, Kyushu University, 744 Motooka, Nishi-ku Fukuoka 819-0395, Japan
[2] Institute of Mathematics for Industry, Kyushu University, 744 Motooka, Nishi-ku Fukuoka 819-0395, Japan

E-mail *k-kimura@math.kyushu-u.ac.jp*

Received March 29, 2017, Accepted June 8, 2017

Abstract

The security of lattice-based cryptography is mainly based on the fact that the shortest vector problem (SVP) is NP-hard. Our interest is to know how large-scale shortest vector problems can be solved by the state-of-the-art software for mixed-integer programs. For this, we provide a formulation for SVP via mixed integer quadratic program and show the numerical performance for TU Darmstadt's benchmark instances with the dimension up to 49.

Keywords shortest vector problem, mixed integer quadratic program, presolve

Research Activity Group Algorithmic Number Theory and Its Applications

1. Introduction

For $n \in \mathbb{N}$, let $b_1, \ldots, b_n$ be n linearly independent vectors over $\mathbb{R}$. The set of all integral combinations of the b_i is called a *lattice* of dimension n, denoted by

$$\mathcal{L}(B) = \left\{ Bx : x = (x_1, \ldots, x_n)^T \in \mathbb{Z}^n \right\},$$

where let $B = (b_1, \ldots, b_n)$ denote the $n \times n$ matrix. B is called a *basis* of the lattice. The shortest vector problem (SVP) is to find a nonzero shortest vector $0 \neq Bx \in \mathcal{L}(B)$. The hardness of the SVP assures the security of lattice-based cryptography. Every lattice has infinitely many bases. If B and C are two bases, there exists a unimodular matrix $U \in \mathrm{GL}_n(\mathbb{Z})$ with $B = CU$. Given an input basis, *lattice reduction* outputs a new basis with short and nearly orthogonal vectors. Such a reduced basis helps us to solve the SVP, and typical algorithms are LLL [1] and BKZ [2].

The contribution of this study is to provide a formulation for SVP via mixed integer quadratic program (MIQP). For this we use a presolve technique. We solve the MIQP problems generated from TU Darmstadt's benchmark instances in [3].

For a given positive integer n, we define $[n] := \{1, \ldots, n\}$. $\{\ell, u\}$ denotes the set of integers $\ell, \ell + 1, \ldots, u-1, u$ for given integers ℓ and u such that $\ell \leq u$.

2. Mixed integer program

We briefly introduce a mixed integer program (MIP) and an algorithm to solve MIPs, *i.e.*, a branch-and-bound (B&B) algorithm. The MIP is the problem of minimizing an objective function over a nonempty set with integrality restrictions. It is formulated by

$$\theta^* := \min_x \{ f(x) : x \in X, x_j \in \mathbb{Z} \ (j \in J) \}, \qquad (1)$$

with an objective function $f : X \to \mathbb{R}$, a subset X of $\mathbb{R}^n$ and a nonempty subset $J \subset [n]$. If $x \in \mathbb{R}^n$ satisfies $x \in X$ and $x_j \in \mathbb{Z}$ for all $j \in J$, then x is said to be feasible for (1) or a feasible solution of (1). The feasible solution x^* is an optimal solution of (1) if $f(x^*) \leq f(x)$ for all feasible solutions x of (1).

We define a relaxation problem of (1), which is used in the B&B algorithm for (1). This problem is obtained by removing the integrality restrictions $x_j \in \mathbb{Z} \ \ (j \in J)$.

$$\theta^*_R := \min_x \{ f(x) : x \in X \}. \qquad (2)$$

It is clear that the optimal value θ^*_R of (2) is a lower bound of the optimal value θ^* of (1), *i.e.*, $\theta^*_R \leq \theta^*$.

The B&B algorithm is a general and widely used algorithm to solve MIPs. It successively divides (1) by using the integrality on x_j $(j \in J)$ until all the generated problems are solved. The generated problems are called subproblems of (1). Then the B&B algorithm creates a search tree to enumerate the subproblems. The relaxation problems (2) are used to divide a given subproblem and to avoid the complete enumeration of all generated subproblems. See [4] for details.

For a positive semidefinite matrix Q and $q \in \mathbb{R}^n$, if we choose $f(x) = x^T Q x + 2q^T x$ and X is a nonempty polyhedron, then (1) is called *a mixed integer quadratic program (MIQP)*. The relaxation problem is called *a quadratic program*, which can be efficiently solved by a barrier method and an interior-point method.

3. MIQP formulation for SVPs

We provide an MIQP formulation for SVPs. For this we use a technique developed in [5]. This is one of the presolve techniques, and restricts the search space of the SVP. The presolve is a set of routines to remove unnecessary variables and constraints, and tighten the bounds

of decision variables of a given MIP. See [6] for details. Applying this technique to SVP, we can formulate an MIQP problem that has the same optimal solution to the one of the original SVP.

3.1 MIQP formulation

SVP is the problem to find the shortest vector in the lattice $\{Bx : x \in \mathbb{Z}^n\}$ except for the zero vector 0_n. This is formulated by

$$\min_x \left\{ \|Bx\|_2^2 : x = (x_1, \ldots, x_n)^T \in \mathbb{Z}^n, x \neq 0_n \right\}, \quad (3)$$

where $\|\cdot\|_2$ is the 2-norm. θ^* denotes the optimal value of (3). This is not the form of the MIQP problem because (3) has the the constraint $x \neq 0_n$.

We can however reformulate (3) into a form of MIQP if every decision variable in (3) has finite lower and upper bounds. To explain this, we consider the following optimization problem

$$\min_x \left\{ \|Bx\|_2^2 : \begin{array}{l} x \in \mathbb{Z}^n, x \neq 0_n, \\ \ell_i \leq x_i \leq u_i \ (i \in [n]) \end{array} \right\}, \quad (4)$$

where ℓ_i and u_i are finite values and $\ell_i \leq u_i$ for all $i = 1, \ldots, n$. If there exists $i \in [n]$ such that $\ell_i > 0$ or $u_i < 0$, then we can remove the constraint $x \neq 0_n$ from (4), and thus the problem has the form of MIQP. Otherwise, we add the binary auxiliary variables $y_{i,v}$ for $i \in [n]$ and $v \in \{\ell_i, u_i\}$ as follows.

$$\min_{x,y} \left\{ \|Bx\|_2^2 : \begin{array}{l} \displaystyle\sum_{i=1}^{n} y_{i,0} \leq n-1, \\ \displaystyle x_i = \sum_{v=\ell_i}^{u_i} v y_{i,v}, \sum_{v=\ell_i}^{u_i} y_{i,v} = 1, \\ y_{i,v} \in \{0,1\} \ (i \in [n], v \in \{\ell_i, u_i\}) \end{array} \right\} \quad (5)$$

We remark that all SVPs in a numerical experiment in Section 4 are converted into the form of (5).

As all the variables $y_{i,v}$ in (5) are binary, the constraint $\sum_{v=\ell_i}^{u_i} y_{i,v} = 1$ means the only one variable $y_{i,v}$ is 1 and the others are 0. Thus we obtain $x_i = v$ from $\sum_{v=\ell_i}^{u_i} v y_{i,v}$ for a v such that $y_{i,v} = 1$. The constraint $\sum_{i=1}^{n} y_{i,0} \leq n-1$ ensures that the number of zero in the decision variables x_1 to x_n is at most $n-1$, *i.e.*, $x \neq 0_n$. Therefore (5) is the form of MIQP.

3.2 Restriction of the search space of SVP

We introduce a technique to restrict the search space of SVP. This technique is to find finite lower and upper bounds ℓ_i and u_i of every decision variables in (3) so that an optimal solution of (3) is also optimal for (4).

The following lemma ensures that we can construct an optimization problem which has the same optimal solution as (3).

Lemma 1 *Let M be a positive number. We consider the following optimization problem*

$$\min_x \left\{ \|Bx\|_2^2 : \begin{array}{l} x = (x_1, \ldots, x_n)^T \in \mathbb{Z}^n, \\ x \neq 0_n, \|Bx\|_2 \leq M \end{array} \right\}. \quad (6)$$

If (6) *has a feasible solution, then the optimal value of* (6) *is the same as that of* (3).

For a given $M > 0$, we define the sets X_M and F_M by $X_M = \{x \in \mathbb{R}^n : x \neq 0, \|Bx\|_2 \leq M\}$ and $F_M = X_M \cap \mathbb{Z}^n$. It should be noted that F_M is the set of all feasible solutions of (6). Hence for any feasible solution $x \in F_M$, we have $\min_x \{x_j : x \in F_M\} \leq x_j \leq \max_x \{x_j : x \in F_M\}$ for all $j \in [n]$. Removing $x \neq 0_n$ and the integrality restrictions from optimization problems in the left-hand and right-hand sides, we obtain

$$\min_x \{x_j : x \in F\} \leq x_j \leq \max_x \{x_j : x \in F\}, \quad (7)$$

where $F = \{x \in \mathbb{R}^n : \|Bx\|_2 \leq M\}$.

By using (7), we propose an iterative approach in Algorithm 1 for obtaining tighter lower and upper bounds in (4). For $\ell, u \in \mathbb{R}^n$ so that $\ell \leq u$ and that may possibly take the values $\ell_i = -\infty$ and $u_i = +\infty$, we define the set $F(\ell, u)$ by $F(\ell, u) = \{x \in \mathbb{R}^n : x \in F, \ell \leq x \leq u\}$. Replacing F in (7) by $F(\ell, u)$, we can expect that tighter lower and upper bounds are computed in Algorithm 1.

Algorithm 1: Algorithm to obtain ℓ_i and u_i in (4)

Input: $B = (b_1, \ldots, b_n) \in \mathbb{Z}^{n \times n}$
Output: Lower and upper bounds ℓ_i, u_i in (4)
$\ell_i \leftarrow -\infty$, $u_i \leftarrow +\infty$ $(i = 1, \ldots, n)$;
$M \longleftarrow \min\{\|b_i\|_2 : i = 1, \ldots, n\}$;
do
  for $i \leftarrow 1$ **to** n **do**
    $v \leftarrow \min_x \{x_i : x \in F(\ell, u)\}$;
    if $\lceil v \rceil > \ell_i$ **then**
      $\ell_i \leftarrow \lceil v \rceil$ and update $F(\ell, u)$;
    end
    $v \leftarrow \max_x \{x_i : x \in F(\ell, u)\}$;
    if $\lfloor v \rfloor < u_i$ **then**
      $u_i \leftarrow \lfloor v \rfloor$ and update $F(\ell, u)$;
    end
  end
while $F(\ell, u)$ *is updated*;

Optimization problems in both sides in (7) and Algorithm 1 are called *second-order cone programs (SOCPs)*, and can be efficiently solved by primal-dual interior-point methods (PDIPMs). In fact, PDIPMs can compute an approximate solution to any given precision in polynomially many iterations. See [7] for details.

Finally, we remark that we may be able to solve SOCPs in Algorithm 1 without applying any SOCP solvers because they have simple forms. In fact, we can provide optimal values and solutions of some SOCP problems with a closed-form expression. For this, we focus on the first do-while loop in Algorithm 1. At the first, we solve the following SOCPs.

$$\min_x \{x_1 : \|Bx\|_2 \leq M\} \quad (8)$$

and $\max_x \{x_1 : \|Bx\|_2 \leq M, \ell_1 \leq x_1\}$. Their optimal values and solutions are provided with closed-form expressions. In fact, the optimal values are $-M\|d_1\|_2$ and $M\|d_1\|_2$, and optimal solutions are $-(M/\|d_1\|_2)B^{-1}d_1$ and $(M/\|d_1\|_2)B^{-1}d_1$, respectively. Here d_i $(i \in [n])$ is the ith column vector of B^{-T}. Hence $\ell_1 = \lceil -M\|d_1\|_2 \rceil$ and $u_1 = \lfloor M\|d_1\|_2 \rfloor$.

Algorithm 2: Algorithm for numerical experiments

Input: Dimension n and seed σ
Output: Optimal solutions x^* and value θ^* of (3)
1: $B_o \longleftarrow$ Generator in [3] from n and σ;
2: $B \longleftarrow$ Apply LLL reduction to B_o in Section 4.2 and 4.3, and BKZ reduction to B_o in Section 4.4;
3: Generate bounds ℓ_i and u_i $(i = 1, \ldots, n)$ by Algorithm 1 from B;
4: $(x^*, \theta^*) \longleftarrow$ Solve (3) via (5);

Next, we solve the following SOCP in Algorithm 1.

$$\min_x \{x_2 : \|Bx\|_2 \le M, \ell_1 \le x_1 \le u_1\}. \tag{9}$$

We assume that

$$\ell_1 \le -\left(\frac{M}{\|d_2\|_2}\right) d_1^T d_2 \le u_1 \tag{10}$$

holds. Then $x^* = -(M/\|d_2\|_2)B^{-1}d_2$ is optimal to (9). In fact, this solution x^* is optimal to $\min_x \{x_2 : \|Bx\|_2 \le M\}$. This is proved in a similar manner to the case of (8), and thus it follows from (10) that x^* is optimal to (9). On the other hand, (10) may fail. In fact it follows from Cauchy-Schwarz inequality that the inequality $\left|-(M/\|d_2\|_2)d_1^T d_2\right| \le M\|d_1\|_2$ holds, while the inequality $\left|-(M/\|d_2\|_2)d_1^T d_2\right| \le \lfloor M\|d_1\|_2 \rfloor$ may not hold because the value at the left-hand side may not be integer. If this inequality holds, then $x^* = -(M/\|d_2\|_2)B^{-1}d_2$ is an optimal solution of (9).

The following lemma is an extension of this discussion, and may improve the performance of Algorithm 1.

Lemma 2 *Let $i \in \{2, n\}$. In addition, $B^{-T} := (d_1, \ldots, d_n)$. We focus on the minimization of the ith iteration of the first do-while loop in Algorithm 1. Then we have $\ell_j = -\infty$ and $u_j = +\infty$ for all $j \in \{i, n\}$. If we have $\ell_j \le -Md_j^T d_i/\|d_i\|_2 \le u_j$ for all $j \in [i-1]$, then $x^* = -(M/\|d_i\|_2)B^{-1}d_i$ is optimal for the minimization and the optimal value is $-M\|d_i\|_2$. Similarly, if we have $\ell_i \le M\|d_i\|_2$ and $\ell_j \le Md_j^T d_i/\|d_i\|_2 \le u_j$ for all $j \in [i-1]$, then $x^* = (M/\|d_i\|_2)B^{-1}d_i$ is optimal for the maximization of the ith iteration of the first do-while loop in Algorithm 1, and the optimal value is $M\|d_i\|_2$.*

4. Numerical experiments

4.1 The setting for the numerical experiments

We report numerical results for some SVPs (5) obtained by the generator that is available at [3]. For the numerical experiments, we used a computer with 32 threads of Intel® Xeon® CPU E5-2687W with 3.1GHz and 128GB RAM, and applied Algorithm 2. In Algorithm 2, we applied the lattice reductions implemented in `fplll` [8] to the generated matrix to improve computational efficiency. We used CPLEX 12.6.3 [9] to solve (5). The parameters in CPLEX are default except for

- `preprocessing presolve = no`,
- `mip tolerances mipgap = 1e-10`,

The first parameter indicates whether we execute some of the presolve techniques implemented in CPLEX or not. To obtain lower and upper bounds of the variables of SVP, we chose the minimum value over $\|b_1\|_2, \ldots, \|b_n\|_2$ as M in Lemma 1.

Table 1. Numerical results for some benchmark problems in [3].

$(n$, seed)	Time	$\|Bx^*\|_2$	Nodes	α
(40, 0)	3036.76	1702.46	5.3×10^7	1.03
(40, 76)	55.48	1434.38	1.6×10^6	0.87
(41, 31)	753.02	1561.65	1.8×10^7	0.93
(41, 135)	282.30	1480.57	6.6×10^6	0.89
(42, 47)	4168.99	1495.81	5.0×10^7	0.89
(43, 2)	6086.09	1545.39	8.0×10^7	0.90
(44, 8)	1006.02	1573.49	2.3×10^7	0.91
(45, 79)	6717.56	1547.23	7.2×10^7	0.88
(46, 16)	14246.33	1565.88	1.4×10^8	0.89
(47, 95)	>86400	1678.65	4.7×10^8	0.94
(48, 7)	>86400	1873.50	4.0×10^8	1.04
(49, 7)	>86400	1927.24	4.1×10^8	1.07
(49, 126)	>86400	1659.49	5.2×10^8	0.91

4.2 Numerical results for (5)

Table 1 displays the results for some TU Darmstadt's benchmark problems in [3]. The first column shows the dimension n and seed that we used in the generator. The second column indicates the CPU time to solve SVP with the generated matrix in seconds. ">86400" means that the algorithm cannot solve the problem within 86400 seconds = 1 day. The third column stands for the optimal value or the computed upper bound of the optimal value. The forth column indicates the number of the generated subproblems in the B&B algorithm. The last column is *the approximation factor* α defined by

$$\alpha := \frac{\|Bx^*\|_2}{\Gamma(n/2+1)^{1/n}|\det(B)|^{1/n}/\sqrt{\pi}},$$

where x^* is the computed solution by CPLEX, and $\Gamma(n/2+1)$ stands for the value of the gamma function at $(n/2+1)$. In particular, the approximation factor is used as the measure of the quality of the computed solution. In [3], it is required to find a feasible solution whose approximation factor is less than 1.05. From Table 1, we see that SVPs with up to 46 are solved within 1 day. However, in each dimension of $n = 40$ and 41, there is a big gap on time required to solve the SVP with different seeds.

4.3 Behavior of the B&B algorithm for SVPs

The B&B algorithm for the MIQP formulation (5) finds an optimal solution soon, while it spends much computational time to prove optimality. The left figure of Fig. 1 displays the transition of upper and lower bounds by the B&B algorithm for SVP with $(n, \text{seed}) = (44, 8)$. We observe that (i) the B&B algorithm finds an optimal solution much early, (ii) the lower bounds get increased slowly, that is, it consumes much computational time to prove optimality. Therefore it is important to develop techniques so that the lower bounds get increased more quickly. It might be good to combine with conventional techniques in lattices.

The right figure of Fig. 1 displays a relationship between the approximation factors and CPU times for 200 randomly generated SVPs with $n = 40$. We see that the

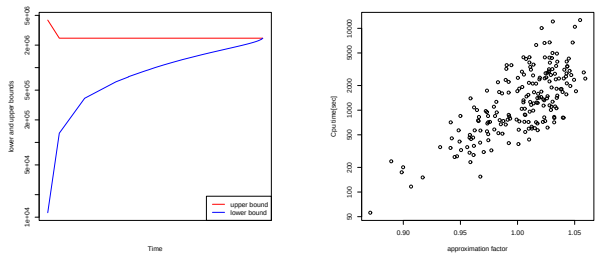

Fig. 1. The log-scale plot of upper and lower bounds in the search tree for SVP with $(n, \text{seed}) = (44, 8)$ (left), and the log-scale plot of CPU time and α for 200 SVPs with $n = 40$ (right).

Table 2. Numerical results for some benchmark problems in [3] reduced by BKZ20.

$(n$, seed)	Time	$\|Bx^*\|_2$	Nodes	α
(40, 0)	408.08	1702.46	1.0×10^7	1.03
(40, 76)	15.67	1434.38	4.5×10^5	0.87
(41, 31)	90.71	1561.65	2.5×10^6	0.93
(41, 135)	58.94	1480.57	1.5×10^6	0.89
(42, 47)	58.92	1495.81	1.7×10^6	0.89
(43, 2)	157.63	1545.39	4.0×10^6	0.90
(44, 8)	313.18	1573.49	7.6×10^6	0.91
(45, 79)	288.11	1547.23	7.2×10^6	0.88
(46, 16)	757.33	1565.88	1.7×10^7	0.89
(47, 95)	5497.33	1678.65	7.4×10^7	0.94
(48, 7)	13827.97	1703.24	1.5×10^8	1.04
(49, 7)	>86400	1826.20	5.3×10^8	1.01
(49, 126)	13319.51	1659.49	1.4×10^8	0.91

computational time and the approximation factor are related to positive correlation, *i.e.*, the B&B algorithm can solve quickly for SVPs whose approximation factor is small. In fact, the optimal values of such SVPs are also small and as a result, the CPU time for proving optimality becomes shorter.

4.4 LLL vs BKZ with block size 20

BKZ is a block-wise generalization of LLL, and it uses a block size parameter β. Larger β outputs a better basis, but it requires more running time. In practice, $\beta = 20$ achieves the best time/quality compromise. We applied BKZ reduction with block size 20 instead of LLL reduction to the generated matrix B. The setting of the numerical experiment is the same as in subsection 4.1.

Table 2 displays the numerical results of CPLEX for SVPs to which the BKZ reduction is applied. We observe from Tables 1 and 2 that by applying the BKZ reduction, we can solve more 3 to 20 times faster than SVPs to which the LLL reduction is applied. One of the reasons is because the search space of BKZ reduced basis is narrower than that of LLL reduced basis. In fact, we can prove under mild assumptions that upper and lower bounds obtained by Algorithm 1 for BKZ reduced basis are smaller than those for LLL reduced basis. We give a more precise statement as follows.

Lemma 3 *For given bases $B = (b_1, \ldots, b_n)$ and $\tilde{B} = (\tilde{b}_1, \ldots, \tilde{b}_n)$, we assume that (i) SVP for B has the same optimal value as SVP for $\tilde{B}$, (ii) Algorithm 1 for B returns $\ell_i = -M\|b_i\|_2$ and $u_i = M\|b_i\|_2$ for $i \in [n]$, and that (iii) $\|\tilde{b}_i\|_2 \leq \|b_i\|_2$ for $i \in [n]$. Then the bounds $\tilde{\ell}_i$ and $\tilde{u}_i$ by Algorithm 1 for $\tilde{B}$ satisfy $\ell_i \leq \tilde{\ell}_i \leq \tilde{u}_i \leq u_i$ for all $i \in [n]$.*

We remark that the BKZ reduced basis $\tilde{B}$ satisfies (iii) in Lemma 3 in comparison to the LLL reduced basis B because the BKZ reduction is a generalization of the LLL reduction (See [10, Section 2.3]). Hence the search space of SVP for $\tilde{B}$ is narrower than that of SVP for B if (ii) in Lemma 3 holds.

5. Conclusion

We provided an MIQP formulation of SVPs and solved some benchmark problems in [3] with the dimension up to $n = 49$. In addition, we find a feasible solution whose approximation factor α is relatively small for dimension $n \geq 50$. We displayed the numerical results by the-state-of-the-art commercial solver CPLEX for the benchmark problems. The BKZ reduction is more effective for the B&B algorithm than the LLL reduction although the former is more expensive than the latter.

To estimate the security level of the cryptography based on SVP, we need to solve SVPs with larger dimension n. It is important to develop cut separation techniques for SVP. See *e.g.*, [4, Chapter 8] for cut separation. This technique will find better lower bounds of optimal values in the B&B tree, and enable to solve such larger SVPs in reasonable time. The development of such a technique is important future work to improve the performance of the B&B algorithm for SVPs.

Acknowledgments

This work was supported by JST CREST Grant Number JPMJCR14D6, Japan.

References

[1] A. K. Lenstra, H. W. Lenstra Jr and L. Lovász, Factoring polynomials with rational coefficients, Math. Ann., **261** (1982), 515–534.
[2] C. P. Schnorr and M. Euchner, Lattice basis reduction: improved practical algorithms and solving subset sum problems, Math. Program., **66** (1994), 181–199.
[3] SVP CHALLENGE, `https://www.latticechallenge.org/svp-challenge/`.
[4] T. Achterberg: Constraint Integer Programming, Ph.D. Thesis, Technische Universität Berlin, 2007.
[5] K. Kimura and H. Waki, A Mixed Integer Quadratic Formulation for the Shortest Vector Problem, to appear in: Mathematical Modelling for Next-Generation Cryptography, T. Takagi et al. eds., Mathematics for Industry, Springer, 2017.
[6] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg and D. Weninger, Multi-Row Presolve Reductions in Mixed Integer Programming, in: Proc. of the Twenty-Sixth RAMP Symposium, pp. 181–196, 2014.
[7] F. Alizadeh and D. Goldfarb, Second-order cone programming, Math. Program., **95** (2003), 3–51.
[8] The FPLLL development team, fplll, a lattice reduction library, `https://github.com/fplll/fplll`, 2016.
[9] IBM ILOG CPLEX Optimizer 12.6.3, IBM ILOG 2015.
[10] N. Gama and P. Q. Nguyen, Predicting Lattice Reduction, in: Advances in Cryptology - EUROCRYPT 2008, N. Smart eds., LNCS, vol. 4965, pp. 31–51, Springer, 2008.