Computer Security Symposium 2018
22 - 25 October 2018

A Continuous Genetic Algorithm for Optimizing Pruning Function for Lattice Enumeration

Trong-Thuc Trang[1] Yoshinori Aono[2] Tsuyoshi Takagi[3]

Abstract: In 2010, Gama et al. proposed a sophisticated version of lattice enumeration called enumeration with extreme pruning. It has been used in several practical implementations of the well-known BKZ algorithms which are used in many records in the Darmstadt challenges. On the other side of success, some crucial open problems about the pruning functions have still remained: how to efficiently generate a near-optimal function satisfying requirements for the input of enumeration with extreme pruning or how to estimate how far a function from the optimal is. In the latest practical implementations, there are two optimization methods have been used without theoretical analysis for their convergence: Nelder-Mead and cross-entropy methods. In this paper, we propose a new approach for optimizing pruning function based on the continuous genetic algorithm. Though our work to make practically faster method is still in progress, to our best knowledge, it can be the first candidate method whose convergence is theoretically guaranteed by the theory of Harris chain.

Keywords: enumeration, extreme pruning, optimizing pruning function, continuous genetic algorithm.

1. Introduction

Lattice-based cryptography has been rising as a promising candidate for post-quantum cryptography instead of classical ones such as RSA and elliptic curve cryptography due to Shor's algorithm [27] used on quantum computer for solving integer factorization and discrete logarithm problems. Technically, most of lattice-based cryptosystems are based on Learning with Errors (LWE) and Short Integer Solutions (SIS) problems whose average cases are at least as hard as the worst cases of Shortest Vector Problem (SVP) and Closest Vector Problem (CVP) [1], [23]. Although both SVP and CVP are proved to be NP-hard under some circumstances [1], [2], [7], [18], there are a lot of efforts and improvements to break their hardness and lattice enumeration is one of the representative instances.

In the simplest form of lattice enumeration, it is just an exhaustive search whose researches date back to the beginning of 80s [14], [20], [22]. After that, it was still investigated and improved with pruned enumeration of Schnorr, Euchner, and Hörner [25], [26]. However, the picture of solving hard lattice problems with reasonable time complexity was actually changed when Gama et al. gave a technique [16], which is called extreme pruning, to speed up pruned enumeration. In this technique, a pruning function consisting of n radii $R_1^2 \leq R_2^2 \leq \cdots \leq R_n^2 = R^2$ are used for the input instead of only radius R^2 as in the original enumeration (n is the lattice dimension). On the other side of great successful on performance improvements, they have left important open problems that are not solved: a concrete mathematical form or family of pruning functions that are optimal in terms of reducing the complexity of pruned enumeration, or a measure which reveals how a given pruning function is near the optimal.

With regard to the optimal pruning functions, there are three methods are being used to heuristically generate them in practice: Nelder-Mead and gradient descent methods [15], and cross-entropy method [5], [9]. However, no one has proved its convergence to the optimal. It is therefore worth asking if there is any optimization method making its input converge to the optimal in theory.

So far, a lot of numerical optimization methods have been proposed. For improvement purpose, we investigated the convergence of discrete genetic algorithm [13], [24] and found out with a special setting it can converge to the desired optimal. Because of that motivation, we therefore propose a new method for generating pruning function by applying continuous genetic algorithm with elitist model [8]. The detailed pseudocode of our proposed method can be found in Algorithm 3 which uses Algorithm 1 and Algorithm 2 as its subroutines. Besides that, we also give some heuristics for modelling our method into a Harris chain, i.e. adapting properties of our algorithm to the properties of Harris chain (Section 2), in an effort to prove its convergence to the optimal pruning function. With regard to practical aspect, we implemented our own continuous genetic algorithm based method in basic C++ programming language. Experiments are done to see how good or bad our method is at the mo-

[1] Graduate School of Mathematics, Kyushu University.
[2] National Institute of Information and Communication Technology, Japan.
[3] The University of Tokyo.

ment, compared to cross-entropy method implemented in [6].

2. Preliminaries

In this section, we give some background needed for our work, including probability theory (Section 2.1) which is mainly about Harris chain, lattice theory (Section 2.2), enumeration algorithm (Sec 2.3) and extreme pruning technique (Section 2.4).

2.1 Probability Theory

Let the triple $(\Omega, \mathcal{F}, \mathbb{P})$ denote the probability space, where Ω is the sample space or the set of all possible outcomes, $\mathcal{F}$ is σ-algebra over Ω or set of all events (each event is equivalent to a subset of Ω), and $\mathbb{P} : \mathcal{F} \to [0, 1]$ is a probability measure.

Definition 1 (Markov chain, [12])**.** The collection of random variables $\{X_i\}_{i\in\mathbb{N}}$ defined on probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with respect to the filtration $\mathcal{F}_n$ ($\mathcal{F}_n = \sigma(X_0, \ldots, X_N)$, i.e. σ-algebra generated by $X_0, \ldots, X_n$) is called a Markov chain with transition probability p if

$$\mathbb{P}[X_{n+1} \in A \mid \mathcal{F}_n] = p(X_n, A)$$

where $n \in \mathbb{N}$ and $A \in \mathcal{F}$.

Typically, when we mention Markov chain, we assume we are dealing with countable state space Ω. Our problem, however, has an uncountable state space. We therefore need to consider a generalization of Markov chain, called Harris chain.

Definition 2 (Harris chain, [12])**.** A Markov chain is called Harris chain if there exists $A, B \in \mathcal{F}$, a function $q(x, y) \geq \epsilon > 0$ where $(x, y) \in A \times B$, and a probability measure ρ concentrated on B, i.e. $\rho(B) = 1$ satisfying

(i) $\mathbb{P}[\tau_A < \infty \mid X_0 \in \Omega] > 0$ where $\tau_A = \inf\{n \geq 0 : X_n \in A\}$.

(ii) $p(x, C) \geq \int_C q(x, y)\rho(dy) \geq \epsilon\rho(C)$ where $x \in A$ and $C \subseteq B$.

Definition 3 ([12])**.** A Harris chain is *recurrent* if

$$\mathbb{P}[\tau_\alpha < \infty \mid X_0 = \alpha] = 1$$

where $\tau_\alpha = \inf\{n \geq 1 : X_n = \alpha\}$.

Definition 4 ([12])**.** A recurrent Harris chain is *aperiodic* if

$$\gcd\{n \geq 1 : \mathbb{P}[X_n = \alpha \mid X_0 = \alpha] > 0\} = 1.$$

Definition 5. A measure $\pi : \mathcal{F} \to \mathbb{R} \cup \{-\infty, +\infty\}$ is a *stationary measure* if $\forall\{y\} \in \mathcal{F}$, simply write y,

$$\sum_{\{x\}\in\mathcal{F}} \pi(x)\, p(x, y) = \pi(y).$$

If stationary measure π is also a probability measure then we call it *stationary distribution*.

What we really consider on Harris chain is its convergence, which is stated in the following theorem.

Theorem 1 ([12])**.** *If $\{X_i\}_{i\in\mathbb{N}}$ is a aperiodic recurrent Harris chain with stationary distribution π and $\mathbb{P}[\tau_\alpha < \infty \mid X_0 = x] = 1$ where $\tau_\alpha = \inf\{n \geq 1 : X_n = \alpha\}$, then*

$$\lim_{n\to\infty} dist(p^n(x, \cdot) - \pi(\cdot)) = 0,$$

where $p^n(x, \cdot) = \mathbb{P}[X_n = \cdot \mid X_0 = x]$ and $dist(\cdot)$ is the total variation distance between two measures.

Finally, there is a useful observation named *68–95–99.7 rule*. Let X be a random variable drawn from normal distribution $\mathcal{N}(\mu, \sigma^2)$ with mean μ and standard deviation σ. That observation gives us a rule about the data

$$\mathbb{P}[|X - \mu| \leq \sigma] \approx 0.6827,$$
$$\mathbb{P}[|X - \mu| \leq 2\sigma] \approx 0.9545,$$
$$\mathbb{P}[|X - \mu| \leq 3\sigma] \approx 0.9973.$$

2.2 Lattice

We use bold letters written in lower case as row vectors and bold letters written in upper case as matrices throughout this paper. The inner product of two real vectors and the Euclidean norm of a real vector is denoted by $\langle\cdot,\cdot\rangle$ and $||\cdot||$, respectively. Notation $span(\cdot)$ stands for the real vector space spanned by a set of real vectors. The k-dimensional ball centered at $\mathbf{0}$ with radius $R > 0$ is $B_k(R)$.

Lattice of rank (or dimension) n is the set $L = \{\sum_{i=1}^n x_i\mathbf{b}_i : x_i \in \mathbb{Z}\}$ of all the linear integer combinations of n linearly independent m-dimensional real vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ ($n \leq m$). This set of vectors is called *lattice basis* and when $n \geq 2$ there are infinitely many equivalent bases up to element of $GL(n, \mathbb{Z})$, i.e. $\exists\mathbf{U} \in GL(n, \mathbb{Z}), \mathbf{UB} = \mathbf{B}'$ where $\mathbf{B}, \mathbf{B}'$ are matrices representing lattice bases by the row vectors. If $n = m$, the lattice is called *full rank*. Also, we denote $\lambda_1(L) = \min\{||\mathbf{x}|| : \mathbf{x} \in L \setminus \{\mathbf{0}\}$ and $\det(L)$ the first minimum and the determinant of lattice L, respectively.

Let

$$\pi_i : span(L) \to span(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$$

be the orthogonal projection map from vector space spanned by L to vector space that is the orthogonal complement of vector space spanned by the the first $i - 1$ basis vectors. The *Gram-Schmidt orthogonalization* $\mathbf{b}_i^*$ of a basis vector $\mathbf{b}_i$ is exactly $\pi_i(\mathbf{b}_i)$ and it can be attained by taking $\mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$ for $1 \leq i \leq n$ and $\mu_{i,j} = \langle\mathbf{b}_i, \mathbf{b}_j^*\rangle/||\mathbf{b}_j^*||^2$. Notice that one way to compute $\det(L)$ is taking $\prod_{i=1}^n ||\mathbf{b}_i^*||$.

Gaussian Heuristic gives us a way to estimate the number of lattice points in a measurable subset of $\mathbb{R}^m$. Particularly, if S is a "nice" set and $S \cap L \neq \emptyset$, then $|S \cap L| \approx vol(S)/\det(L)$ holds. As a simple application of this heuristic, we consider $B_n(R)$, then the radius satisfying $vol(B_n(R)) = \det(L)$ gives us an approximation of the first minimum of lattice which is denoted by $GH(L)$.

The lattice problem that is mostly considered is *Shortest Vector Problem* (SVP). In this problem, one is given a basis of lattice L and asked to find a lattice vector whose length achieves $\lambda_1(L)$. There are several algorithms for solving SVP approximately or exactly such as lattice reduction algorithms (LLL [21], BKZ [6], [10], [25]) and enumeration [4], [14], [16], [20], [22], [25], [26].

2.3 Lattice Enumeration

The original *enumeration* [14], [20], [22] aims to list all the lattice vectors inside an n-dimensional ball of radius R. Mathematically, given a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of lattice L and a bound $R > 0$, enumeration algorithm will output $\{\mathbf{x} \in L : ||\mathbf{x}|| \leq R\}$. Intuitively, in order to do that an enumeration tree of depth n will be built. The root is nil and the leaves are all lattice vector of length at most R. At each depth k (for $1 \leq k \leq n$), the nodes are the vectors of projected lattice $\pi_{n+1-k}(L)$ of rank k whose lengths are also at most R. It is well-known that the total number of nodes in the enumeration tree $T \approx \sum_{k=1}^{n} T_k$ where $T_k = vol(B_k(R))/\det(\pi_{n+1-k}(L))$ and if $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is LLL-reduced, then $T \lesssim 2^{O(n^2)}$.

It is not hard to see that the main reason of the existence of variants of enumeration is to reduce its exponential time complexity. Indeed, the *pruned enumeration* of Schnorr-Euchner [25] and later investigated in Schnorr-Hörner [26] is such a variant like that. The idea of this technique is really simple: we enumerate a part of the nodes on the enumeration tree and ignore the rest. More precisely, in pruned enumeration, we use bound R_k instead of bound R for each depth k of enumeration tree, where $0 \leq R_1 \leq \cdots \leq R_n = R$ are n real numbers and the vector $(R_1, \ldots, R_n)$ is the so-called *pruning function*. Unfortunately, the proposed pruning function in [25] has no analysis and limited practical experiments, and even in the latter research [26] still has some fundamental flaws shown in [16]. However, despite these drawbacks, pruned enumeration still gave the core idea for the work [16] of Gama-Nguyen-Regev.

2.4 Extreme pruning

Extreme pruning [16] is the name of the technique improving the running time of pruned enumeration of Schnorr, Euchner and Hörner [25], [26]. The key point of extreme pruning and also the contribution of Gama-Nguyen-Regev is that if a pruning function which gives a low probability of successfully outputting the desired lattice vector, let us denote it p, is used, the running time of pruned enumeration will be surprisingly sped up by a factor much more than $1/p$ times. As a consequence, even when we repeat pruned enumeration with extreme pruning for $\lfloor 1/p \rfloor$ equivalent bases of the same lattice, the total running time is still significantly less than the one obtained from pruned enumeration without extreme pruning for only one lattice basis. Along with providing a speed-up technique, the authors also gave their own analysis on the running time and success probability of pruned enumeration of Schnorr, Euchner, and Hörner which is adaptable for any pruning function and related to the optimization problem we are considering. In summary, under some assumptions mentioned in [16], the running time T_{pruned} and success probability p_{succ} of pruned enumeration are approximated as follows

$$T_{(R_1,\ldots,R_n)} = \sum_{k=1}^{n} \frac{vol(C_k(R_1, \ldots, R_k))}{\prod_{i=n+1-k}^{n} ||\mathbf{b}_i^*||}, \qquad (1)$$

$$p_{(R_1,\ldots,R_n)} = \mathbb{P}_{\mathbf{u} \sim S^{n-1}(R_n)} \left[\forall 1 \leq \ell \leq n, \sum_{i=1}^{\ell} u_i^2 \leq \frac{R_\ell^2}{R_n^2} \right] \qquad (2)$$

where $C_k(R_1, \ldots, R_k)$ is k-dimensional cylinder intersection defined as $\{\mathbf{u} \in \mathbb{R}^k : \forall 1 \leq \ell \leq k, \sum_{i=1}^{\ell} u_i^2 \leq R_\ell^2\}$ and $S^{n-1}(R_n)$ is n-dimensional sphere of radius R_n. Furthermore, there are some ways to upper bound and lower bound $T_{(R_1,\ldots,R_n)}$ and $p_{(R_1,\ldots,R_n)}$, which somehow may give more information than (1) and (2). The first idea is described in [16] and then an explanatory technical paper is given by [3]. They are implemented in state-of-the-art BKZ libraries [6], [15]. These both implementations also give us several ways to generate a pruning function which optimizes the running time bounds and satisfies some desired constraints, which are essentially needed for lattice enumeration with extreme pruning. Among those ways, Nelder-Mead and cross-entropy methods can be seen as the best method of fpLLL [15] and progressive BKZ [6], respectively. When it comes to comparison, it is hard to know which one is better because they are dealing with two different kinds of optimization problem. However, one can find some points of view of Aono *et al.* in [5] about the drawbacks of Nelder-Mead and gradient descent (also used in fpLLL) when we want to apply them to the optimization problem that cross-entropy method is being exploited.

3. Continuous Genetic Algorithm with Elitist Model

The idea of genetic algorithms (GA) is introduced by DeJong [11] (1980), Goldberg [17] (1989), and Holland [19] (1992), which is actually inspired by three natural processes: recombination (cross-over), mutation, and natural selection. In practice, GA is widely used in the field of solving search, optimization problems, and machine learning. Typically, in GA, we need to determine two basic things: population and finess function. *Population* is a set of possible solutions which "evolve" over generations toward better solutions. Each solution in population, which is called *individual*, is represented by its genetically encoding (e.g. binary, string, or real numbers). One important point of genetic algorithm is the *fitness function* which reveals how "fit" an individual in a population is in terms of some certain conditions. If one individual is less "fit" than the others, it tends to be eliminated from the population. In order to evolve, the recombination, mutation, and selection operators are carried out on population over generations. The specific ways that these operators are performed or combined may differ from algorithm to algorithm and from one encoding to another. Based upon the way of encoding the representation of solution, we have two main kinds of genetic algorithms: discrete and continuous. The outline of continuous GA, we will use this term if genes were encoded by real-valued vectors, is the same as the discrete version of GA (see [8] for more details). Also, by elitist model, we mean nothing, but the algorithm will save the elite individual from the previous generation to

the current generation.

3.1 Recombination

This operator simulates the action of mating between two individuals in a population of the same species. Notice that, in the continuous cases, recombination is not just simply doing swapping as in discrete cases (e.g. see **Fig. 1**). Roughly speaking, in continuous GA, we will add a small value for each position (behind crossing point) of one parent and subtract that small value for each corresponding position of the other. The small values we need are extracted from both parents. This means the parents are exchanging their characteristics which may contain some good factors for evolutionary process.

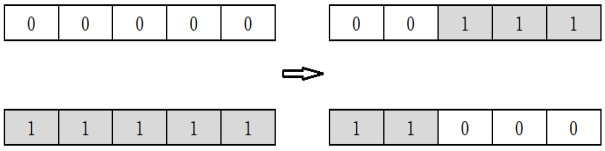

Fig. 1 Example of recombination of discrete GA with binary encoding.

3.2 Mutation

This operator is used to avoid degeneration of population which corresponds to getting trapped in local optimal when dealing with optimization problem. This changes the characteristics of target individual slightly or totally. We may not know if this change is good, but we can be sure, with an appropriate choice of parameters, mutation will help us get out of being stuck in the area of local optimal. Mutation of continuous GA is a bit more complicated than the one of discrete GA.

3.3 Selection and Evaluation

During the execution of GA, we always need to determine which pairs of individuals will take part in the recombination process or which individual will alive for the next generation, which are the purposes of selection operator. However, selection should not work randomly, but should depend on results of evaluation process. In general, to get better solutions, one can follow this strategy: individuals adapting to the fitness function more than the others have higher probability to be chosen for recombination or to be alive. It is therefore important to choose the fitness function for a specific problem carefully.

4. Proposed Method using Continuous GA with Elitist Model

The problem that we consider throughout this section is totally the same with the ones mentioned in [3] and [5], Section 4.1, (i). Specifically, given basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of n-dimensional lattice L, a radius $R = \alpha GH(L) > 0$ (α is a practical parameter), and a target probability $p_0 > 0$, we try to generate a *monotonic vector* $(f(1), \ldots, f(n)) \in [0,1]^n$ (i.e. $f(1) \leq \cdots \leq f(n)$) which can produce a pruning function $(R_1, \ldots, R_n)$, minimizing (1) subject to (2) at least p_0, by multiplying a scalar R, i.e. we are simply trying to solve

$$\arg\min \left\{ T_{(Rf(1),\ldots,Rf(n))} : p_{(Rf(1),\ldots,Rf(n))} \geq p_0 \right\}. \tag{3}$$

We introduce several concepts from the continuous GA and explain how do we adapt them for our purpose. We denote the j-th individual in population at i-th generation by $\mathbf{f}_j^{(i)}$, which is a monotonic vector in $[0,1]^n$. Therefore, the population at i-th generation is denoted by the set $\mathcal{P}^{(i)} = \left\{ \mathbf{f}_1^{(i)}, \ldots, \mathbf{f}_{N(i)}^{(i)} \right\}$ where $N(i)$ is the number of individuals of $\mathcal{P}^{(i)}$. Here $\mathcal{P}^{(0)}$ is called the initial population. In our algorithm, the fitness value of $\mathbf{f}_j^{(i)}$ is defined by

$$\frac{\sum_{k=1}^{N(i)} T_{(Rf_k^{(i)}(1),\ldots,Rf_k^{(i)}(n))}}{T_{(Rf_j^{(i)}(1),\ldots,Rf_j^{(i)}(n))}}.$$

An outline of our algorithm using continuous GA with elitist model [8] is shown in Algorithm 3 with subroutines Algorithm 1 and Algorithm 2.

4.1 Recombination and Mutation

In this section, we will show how recombination and mutation operators work explicitly in our method. A recombination step at the i-th generation takes two chosen parents $\mathbf{f}_j^{(i)}$ and $\mathbf{f}_k^{(i)}$ as input. We randomly choose a crossing point $\ell \in \{1, \ldots, n\}$ so that all the characteristics from this crossing point ℓ of both parents will be slightly exchanged to each other. More precisely, for each position h from crossing point, we compute the difference between the characteristics of $\mathbf{f}_j^{(i)}$ and $\mathbf{f}_k^{(i)}$ at position h and divide the difference by a random $M \in \{1, \ldots, 1000\}$, which is set as $\triangle$. Then, we subtract $\triangle$ from $f_j^{(i)}(h)$ add the $\triangle$ to $f_k^{(i)}(h)$. Intuitively, $\mathbf{f}_j^{(i)}$ and $\mathbf{f}_k^{(i)}$ are exchanging a small amount to each other just like recombination of gene in real life. The detailed description is shown in Algorithm 1.

Algorithm 1 Recombination operator

Input: Two individuals $\mathbf{f}_j^{(i)}$ and $\mathbf{f}_k^{(i)}$.
Output: Two new individuals $\mathbf{f}_{j'}^{(i)}$ and $\mathbf{f}_{k'}^{(i)}$.
1: Choose a crossing point $\ell \in \{1, \ldots, n\}$ and a parameter $M \in \{1, \ldots, 1000\}$ uniformly at random.
2: $\mathbf{f}_{j'}^{(i)} = \mathbf{f}_j^{(i)}$ and $\mathbf{f}_{k'}^{(i)} = \mathbf{f}_k^{(i)}$. ▷ *// taking copies*
3: **for** $h = \ell, \ldots, n$ **do**
4: $\quad \triangle = (f_j^{(i)}(h) - f_k^{(i)}(h))/M$
5: $\quad f_{j'}^{(i)}(h) = f_{j'}^{(i)}(h) - \triangle$
6: $\quad f_{k'}^{(i)}(h) = f_{k'}^{(i)}(h) + \triangle$
7: **end for**
8: Adjust $\mathbf{f}_{j'}^{(i)}$ and $\mathbf{f}_{k'}^{(i)}$ as two monotonic vectors in $[0,1]^n$.
9: **return** $\mathbf{f}_{j'}^{(i)}$ and $\mathbf{f}_{k'}^{(i)}$.

In mutation process, one characteristic of an individual will be changed in the way as follows. The amount for changing is defined by a ratio whose numerator is given by a

Algorithm 2 Mutation operator

Input: An individual $\mathbf{f}_j^{(i)}$.

Output: A new individual $\mathbf{f}_{j'}^{(i)}$.

1: Choose a sign parameter $h \in \{0,1\}$, a mutation point $\ell \in \{1,\ldots,n\}$, and a parameter $M \in \{1,\ldots,10\}$ uniformly at random.
2: $\mathbf{f}_{j'}^{(i)} = \mathbf{f}_j^{(i)}$. ▷ *// taking copy*
3: $\triangle = (-1)^h 0.9999^i/M$
4: $f_{j'}^{(i)}(\ell) = f_{j'}^{(i)}(\ell) + \triangle$
5: Adjust $\mathbf{f}_{j'}^{(i)}$ as a monotonic vectors in $[0,1]^n$.
6: **return** $\mathbf{f}_{j'}^{(i)}$.

slightly decreasing function $k_{mut}(i) = 0.9999^i$ and denominator is random integer $M \in \{1,\ldots,10\}$. Then we randomly add or subtract this amount of changing to a random characteristic of chosen individual. The purpose of using $k_{mut}(i)$ is that we heuristically want to "jump out" from the trap of local optimal, but not so far from the global optimal. Parameter M is needed just only for controlling how large the amount of changing. Algorithm 2 defines our mutation operator.

4.2 Selection and Evaluation

By selection, we want to mention both tasks of selection arising in Algorithm 3: parent selection (step 9) and selection for the next generation (step 28–30). Clearly, we can see in Algorithm 3, we are using selection methods that are proportionate to some criteria. For choosing parents, we want the children after recombination to be "fit" as much as possible, so we use fitness proportionate selection. In contrast, when we want to reduce redundant individuals to determine the population of the next generation, cost proportionate selection is chosen to use. In practice, roulette wheel selection technique is being used in our implementation.

4.3 Main Algorithm

In Algorithm 3, along with main operators of continuous GA, we also exploit the strategy of [8] including two phases: diversification and intensification. The algorithm starts with the diversification phase where GA operators try to identify "a promising area" [8]. Potentially, this area contains the global optimal and will show up after a certain number of generations (e.g. g_{local} in Algorithm 3). Once such an area is found, algorithm changes to the *intensification* phase. In our case, we reduce the number of individuals of population and mutation probabilty over generations, and again continue using GA operators in order to localize the position of the elite individual in this "promising" area until algorithm reaches the target generation g_{max}. Most importantly, after selection process is done, we always need to compare the elite individuals (smallest-cost individuals) of previous generation and current generation (step 32–36). If the previous elite individual is better than current one in terms of comparing cost, then we replace the highest-cost individual in current generation by previous elite individual.

Algorithm 3 Continuous GA Based Algorithm

Input: Basis $\mathbf{b}_1,\ldots,\mathbf{b}_n$ of lattice L, radius $R = \alpha GH(L) > 0$, target probability $p_0 > 0$, $N(0)$, recombination probability p_{rec}, mutation probability p_{mut}, generation thresholds g_{local} and g_{max}, localization parameter $\triangle$, and distribution parameter $\beta < 1$.

Output: A pruning function $(Rf(1),\ldots,Rf(n))$ satisfying (3).

1: $i = 0$ ▷ *// #generation*
2: Draw $N(i)$ individuals for $\mathcal{P}^{(i)}$ from Cartesian product of n Gaussian distributions $\mathcal{N}(1/n,(\beta/n)^2) \times \cdots \times \mathcal{N}(1,\beta^2)$ satisfying (2) at least p_0.
3: $\mathbf{f}$ = elite individual, whose cost (1) is smallest, from $\mathbf{P}^{(i)}$
4: **while** $i < g_{max}$ **do**
5: *// Evaluation*
6: Compute fitness values of $N(i)$ individuals of $\mathcal{P}^{(i)}$.
7: *// Recombination*
8: **for** $k = 1,\ldots,\lfloor p_{rec}N(i)/2 \rfloor$ **do**
9: Choose 2 distinct individuals from $\mathcal{P}^{(i)}$ based on fitness proportionate selection.
10: Run Alg. 1 with parents and get 2 new individuals.
11: Save to $\mathcal{P}^{(i)}$ any new individual that does not make (2) be smaller p_0.
12: **end for**
13: *// Mutation*
14: **for** each individuals in current $\mathcal{P}^{(i)}$ **do**
15: Choose $\epsilon \in [0,1]$ uniformly at random.
16: **if** $\epsilon \le p_{mut}$ **then**
17: Run Alg. 2 with current individual and get new individual.
18: Save to $\mathcal{P}^{(i)}$ if new individual does not make (2) be smaller p_0.
19: **end if**
20: **end for**
21: $i = i + 1$ ▷ *// move to next generation*
22: *// Intensification*
23: **if** $i \ge g_{local}$ **then**
24: $N(i) = N(i-1) - \triangle$
25: $p_{mut} = p_{mut}/\exp(1)$
26: **end if**
27: *// Selection*
28: **for** $k = 1,\ldots,|\mathcal{P}^{(i-1)}| - N(i)$ **do**
29: Discard individual from $\mathcal{P}^{(i-1)}$ based on cost (1) proportionate selection.
30: **end for**
31: *// Keeping elite individual*
32: **if** $\mathbf{f}$ has smaller cost (1) than elite individual of $\mathcal{P}^{(i)}$ **then**
33: Replace the individual whose cost (1) is maximum in $\mathcal{P}^{(i)}$ by $\mathbf{f}$.
34: **else**
35: $\mathbf{f}$ = elite individual of $\mathcal{P}^{(i)}$.
36: **end if**
37: **end while**
38: **return f**

5. Towards the Proof of Convergence

In this section, we give what we think about the spirit of continuous GA in Algorithm 3 based on a stochastic process in probability theory: Harris chain. We show that our method is able to be viewed as a Harris chain (Section 5.1) and has recurrence and aperiodicity properties (Section 5.2).

To our best knowledge, we cannot find any previous work on proving the convergence to the optimal of continuous GA like that of discrete GA. By using Markov chain on countable state space, discrete GA with elitist model corresponding to a specific optimization problem is proved to always converge to the desired optimal of that problem as in [13], [24]. It is therefore natural to think of a "continuous" version of such proofs and Harris chain is the most suitable answer.

5.1 Modelling Our Method as Harris Chain

Heuristic 1. *With the choice of $\beta = 0.3$, the elite individual of population $\mathcal{P}^{(0)}$ of the continuous GA with elitist model in Algorithm 3 will turn into an individual belonging to $\prod_{h=1}^{n}[0.1\mu_h, \min\{1.9\mu_h, 1\}]$ (where $\mu_h = h/n$ for $1 \leq h \leq n$) with a nonzero probability right in the first generation.*

By using $\beta = 0.3$ and the 68-95-99.7 rule, the probability that values drawn from $\mathcal{N}(\mu_h, (0.3\mu_h)^2)$ lie inside the interval $[0.1\mu_h, 1.9\mu_h]$ is 0.997. However, as h grows, $1.9\mu_h$ will exceed upper bound 1 and indeed it happens when $\mu_h > 10/19$ (approximately, $\mu_h > 1/2$), which means $h > n/2$. Notice that each individual of $\mathcal{P}^{(0)}$ is drawn from $\prod_{h=1}^{n}\mathcal{N}(\mu_h, (0.3\mu_h)^2)$ and it has to be a monotonic vector in $[0,1]^n$. Therefore, an individual $\mathbf{f}_j^{(0)}$ must belong to the set $\prod_{h=1}^{n}[0.1\mu_h, \min\{1.9\mu_h, 1\}]$ which which can be lower bounded by $\prod_{h=1}^{\lfloor n/2\rfloor}[0.1\mu_h, 1.9\mu_h] \times \prod_{h=\lfloor n/2\rfloor+1}^{n}[0.1\mu_h, \mu_h]$. Hence probability

$$\begin{aligned}
&\mathbb{P}\left[\mathbf{f}_j^{(0)} \in \prod_{h=1}^{n}[0.1\mu_h, \min\{1.9\mu_h, 1\}]\right] \\
&\gtrsim \mathbb{P}\left[\mathbf{f}_j^{(0)} \in \prod_{h=1}^{\lfloor n/2\rfloor}[0.1\mu_h, 1.9\mu_h] \times \prod_{h=\lfloor n/2\rfloor+1}^{n}[0.1\mu_h, \mu_h]\right] \\
&\approx 0.997^{\lfloor n/2\rfloor} 0.499^{(n-\lfloor n/2\rfloor)} > 0.
\end{aligned}$$

After generating initial individuals, Algorithm 3 steps into recombination process which calls Algorithm 1. By also using 68-95-99.7 rule, we can upper bound and lower bound quantity $\triangle$ by the interval $[-1.8\mu_h/M, 1.8\mu_h/M]$ with probability 0.997. Thus, with probability 0.994, $f_{j'}^{(i)}(h)$ and $f_{k'}^{(i)}(h)$ fall into $[0.1\mu_h - 1.8\mu_h/M, 1.9\mu_h + 1.8\mu_h/M]$. After adjustment both $f_{j'}^{(i)}(h)$ and $f_{k'}^{(i)}(h)$ will be in $[\max\{0, 0.1\mu_h - 1.8\mu_h/M\}, \min\{1.9\mu_h + 1.8\mu_h/M, 1\}] \supset [0.1\mu_h, \min\{1.9\mu_h, 1\}]$. Hence there will be a nonzero probability of new individuals reproduced from recombination process that will turn into $\prod_{h=1}^{n}[0.1\mu_h, \min\{1.9\mu_h, 1\}]$. We will not consider the individuals generated from mutation process because they tend to "jump out" the desired area. Notice that, before selection process, no individual in the initial population $\mathcal{P}^{(0)}$ before recombination is eliminated. Also, it is disadvantageous that we do not know what kind of the distribution of the individuals with high values of cost (1) is. For an ideal circumstance, we assume the distribution is "uniform". By "uniform", we want to describe that the individuals with high values of cost (1) uniformly lie in three parts of $\mathcal{P}^{(0)}$ after mutation including the initial part before recombination, the part added after recombination, and the part added after mutation. Hence after selection and determining the next generation, there will be a nonzero probability of individuals belonging to $\prod_{h=1}^{n}[0.1\mu_h, \min\{1.9\mu_h, 1\}]$ still "alive" in $\mathcal{P}^{(1)}$ which completes the proof.

We believe it is possible to get the same result with other desired area and other distribution of high cost individuals as long as the number of these individuals lying inside the initial part of $\mathcal{P}^{(0)}$ before recombination is not too dominant when compared to other parts.

Heuristic 2. *The continuous GA with elitist model in Algorithm 3 is a Markov chain with transition probability p.*

It is not hard to see continuous GA is a Markov chain with transition probability p because with $\mathbf{f}^{(0)}, \ldots, \mathbf{f}^{(n)}$ are $n+1$ previous states outputted from continuous GA and $B \subset [0,1]^n$, $p(\mathbf{f}^{(n)}, B) = \mathbb{P}[\mathbf{f}^{(n+1)} \in B \mid \sigma(\mathbf{f}^{(0)}, \ldots, \mathbf{f}^{(n)})] = \mathbb{P}[\mathbf{f}^{(n+1)} \in B \mid \mathbf{f}^{(n)}]$ where $\sigma(\mathbf{f}^{(0)}, \ldots, \mathbf{f}^{(n)})$ is the σ-algebra generated by $\mathbf{f}^{(0)}, \ldots, \mathbf{f}^{(n)}$.

We are close to what we believe, that is continuous GA with elitist model in Algorithm 3 is actually a Harris chain. Unfortunately, we do not know if the transition probability function p is continuous or not. As far as we know, this question is too complicated. We therefore assume Markov chain in Heuristic 2 has continuous transition probability function p such that $p(\mathbf{g}, d\mathbf{h}) = p(\mathbf{g}, \mathbf{h})d\mathbf{h}$ (here $p(\mathbf{g}, d\mathbf{h})$ means probability that current state $\mathbf{g}$ will transform to a state in a very small partition $d\mathbf{h}$ of B, see Example 6.8.2 in [12]) to induce the following important heuristic.

Heuristic 3. *As a consequence of Heuristic 2 under assumption that transition probability p is continuous and $p(\mathbf{g}, d\mathbf{h}) = p(\mathbf{g}, \mathbf{h})d\mathbf{h}$, there exist two sets $A, B \subset [0,1]^n$, a function q on $A \times B$ with $q(\mathbf{g}, \mathbf{h}) \geq \epsilon > 0$ and a probability measure ρ with $\rho(B) = 1$ such that they satisfy axiom (ii) of Definition 2.*

We choose two vectors $\mathbf{g}_0$ and $\mathbf{h}_0$ from $\prod_{h=1}^{n}[0.1\mu_h, \min\{1.9\mu_h, 1\}]$ as described in Heuristic 1. Assume $\mathbf{g}_0$ is of 0th generation and $\mathbf{h}_0$ is of 1st generation, we have $p(\mathbf{f}^{(0)}, \mathbf{f}^{(1)}) = p(\mathbf{g}_0, \mathbf{h}_0) > 0$. The function q is chosen as transition probability p. In order to do that, we let A and B be two "small enough" neighborhoods that lead to $p(\mathbf{g}, \mathbf{h}) \geq \epsilon > 0$ where $(\mathbf{g}, \mathbf{h}) \in A \times B$. Also, we define $\rho(C) = |C|/|B|$ for every $C \subset B$. Hence

$$p(\mathbf{g}, C) = \int_C p(\mathbf{g}, \mathbf{h})d\mathbf{h} \geq \int_C p(\mathbf{g}, \mathbf{h})\rho(d\mathbf{h}) \geq \epsilon\rho(C)$$

where $\mathbf{g} \in A$.

5.2 Recurrence and Aperiodicity

Heuristic 1 and Heuristic 3 show that the continuous GA with elitist model in Algorithm 3 satisfies axioms (i) and (ii) of Definition 2 and hence can be modelled as a Harris chain with transition probability p. Besides, thanks to the elitist model, which is saving the elite individual (lowest cost vector) found from the current generation and substituting to the highest cost vector found in the next generation if needed, we can easily find a lot of values $\mathbf{g}$ such that

$\mathbb{P}[\tau_{\mathbf{g}} < \infty \mid \mathbf{g}] = 1$ where $\tau_{\mathbf{g}} = \inf\{i \geq 1 : \mathbf{f}^{(i)} = \mathbf{g}\}$ when we run Algorithm 3. Also, if $\mathbf{g}$ is saved, it will be saved right at the next generation. Hence $\gcd\{m' = m - i \geq 1 : \mathbb{P}[\mathbf{f}^{(m)} = \mathbf{g} \mid \mathbf{f}^{(i)} = \mathbf{g}]\} = 1$ where $m > i$. Therefore, continuous GA with elitist model in Algorithm 3 is a recurrent, aperiodic Harris chain as in Definition 3 and Definition 4.

6. Experimental Results

We do experiments for two purposes: to choose suitable parameters for our algorithm (Section 6.1) and to compare it with cross-entropy method implemented in [6] (Section 6.2). The target success probability p_0 is set up as 0.0003 throughout all the experiments of both sections.

6.1 Parameter Setting

For the first purpose, we realize there are four parameters that essentially affect to the running time of our algorithm, that are $g_{local}, g_{max}, p_{rec}$, and p_{mut}. In order to estimate the running time, we count the number of calls (# calls) for subroutine computing cost (1). We define the default parameters of our algorithm $g_{local} = 3n$, $g_{max} = 5n$, $p_{rec} = 0.85$, and $p_{mut} = 0.9$ so that when one parameter varies, the others will stay as default setting. Lattice dimension n here is fixed at 100.

Table 1 Experiment with $g_{local} = n, 2n, 3n$, and $4n$.

g_{local}	mean # calls	mean log(cost (1))
n	8089	47.35
$2n$	12049	46.99
$3n$	16022	46.97
$4n$	19919	46.97

Table 2 Experiment with $g_{max} = 4n, 5n, 6n$, and $7n$.

g_{max}	mean # calls	mean log(cost (1))
$4n$	15167	46.98
$5n$	16022	46.97
$6n$	16815	47.09
$7n$	17595	47.10

Table 3 Experiment with $p_{rec} = 0.75, 0.8, 0.85, 0.9$ and 0.95.

p_{rec}	mean # calls	mean log(cost (1))
0.75	15029	47.11
0.8	16027	46.96
0.85	16022	46.97
0.9	16540	46.94
0.95	17201	46.92

Table 4 Experiment with $p_{mut} = 0.75, 0.8, 0.85, 0.9$ and 0.95.

p_{mut}	mean # calls	mean log(cost (1))
0.75	14741	47.10
0.8	15155	47.07
0.85	15603	47.14
0.9	16022	46.97
0.95	16490	46.98

According to **Table 1**, **Table 2**, **Table 3**, and **Table 4**, we can see the most suitable setting for the parameters is the default setting. By the most suitable setting, we mean that with this setting of parameters, our algorithm will generate a pruning function with low cost in an acceptable number of calls.

6.2 Comparison with Cross-Entropy

Now we can make a comparison between cross-entropy method and our method based on the found suitable setting. In order to compare them, we focus on two points of view: the shape of the best pruning function found and the convergence speed. It is straightforward to see from **Fig. 2**, **Fig. 3**, **Fig. 4**, and **Fig. 5**, to achieve a pruning function that is near to the pruning function generated by cross-entropy method, our algorithm has to pay a larger cost and a longer runing time.

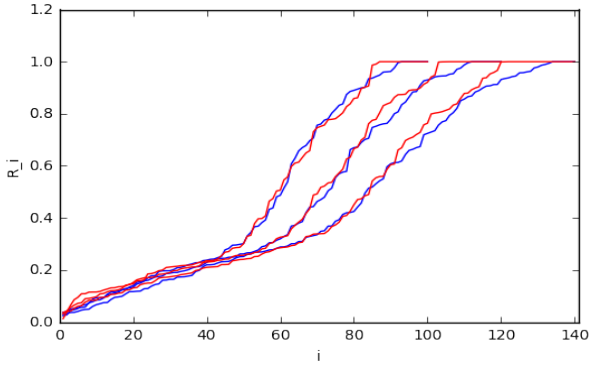

Fig. 2 Shapes of some best pruning functions generated from cross entropy-method (blue curve) and our method (red curve) with lattice dimension $n = 100, 120$, and 140.

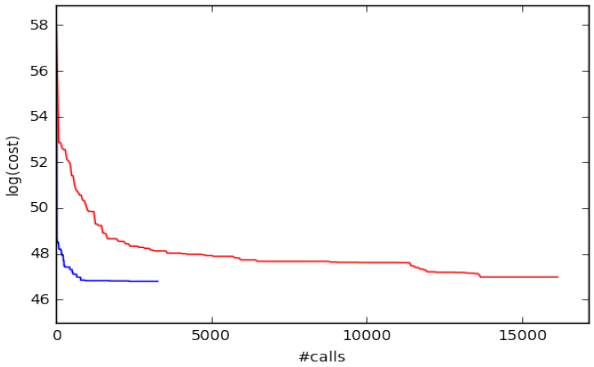

Fig. 3 Convergence speed of best pruning functions generated from cross-entropy method (blue curve) and our method (red curve) with lattice dimension 100.

7. Conclusion and Future Works

With regard to the theoretical aspect, the heuristics in Section 5 are what we first think about the adaption between our algorithm and Harris chain. At the moment, we have no clue about stationary distribution π to prove the convergence of our method modelled as Harris chain, so we leave it for our futher investigation. About practical aspect, from the previous section, one can see that our algorithm is outperformed by cross-entropy method in the perspective of convergence speed. It happens because we are still in progress of improving this algorithm to be more efficient.

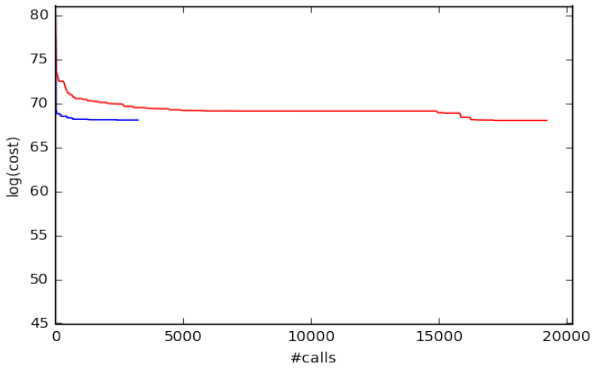

Fig. 4 Convergence speed of best pruning functions generated from cross-entropy method (blue curve) and our method (red curve) with lattice dimension 120.

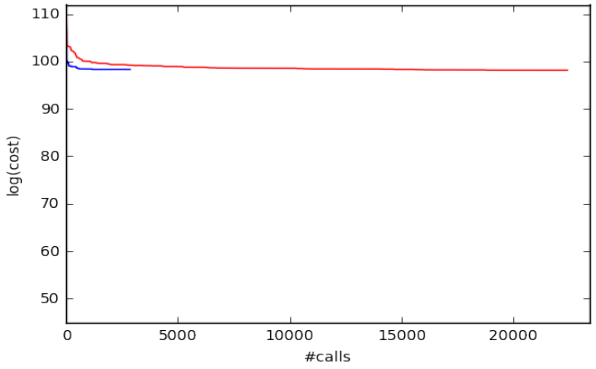

Fig. 5 Convergence speed of best pruning functions generated from cross-entropy method (blue curve) and our method (red curve) with lattice dimension 140.

For future works, we will investigate more about continuous GA and its connections with Harris chain so that we can have some explicit theorems and corresponding proofs for convergence analysis. We are continually improving the implementation of our method so that it will have some good aspects compared to cross-entropy method. In this paper, we are supposed to compare our method with the methods implemented in [15] including Nelder-Mead and gradient descent, and cross-entropy method. However, because of the difference between the target optimization problem of the methods of [15] and that of our continuous GA based method, we will try to figure out the circumstances such that the equivalence between two problems exists. We hope that our work will lead us to some interesting things around the optimal shapes of pruning functions in terms of reducing the cost (1) that are needed for our further researches.

References

[1] M. Ajtai, "Generating hard instances of lattice problems," Proc. 28th ACM STOC, pp. 99–108, 1996.
[2] M. Ajtai, "The shortest vector problem in L_2 is NP-hard for randomized reductions," Proc. 30th ACM STOC, pp. 10–19, 1998.
[3] Y. Aono, "A Faster Method for Computing Gama-Nguyen-Regev's Extreme Pruning Coefficients," arXiv: 1406.0342, 2014.
[4] Y. Aono and P.Q. Nguyen, "Random Sampling Revisited: Lattice enumeration with discrete pruning," EUROCRYPT 2017, LNCS, vol. 10211, pp. 65–102, 2017.
[5] Y. Aono, P.Q. Nguyen, T. Seito, and J. Shikata, "Lower bounds on lattice enumeration with extreme pruning," CRYPTO 2018. To appear.
[6] Y. Aono, Y. Wang, T. Hayashi, T. Takagi, "Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator," EUROCRYPT 2016, LNCS, vol. 9665, pp. 789–819, 2016.
[7] P.v.E. Boas, "Another NP-complete partition problem and the complexity of computing short vectors in a lattice," Technical Report 81-04, Mathematische Instituut, University of Amsterdam, 1981.
[8] R. Chelouah and P. Siarry, "A Continuous Genetic Algorithm Designed for the Global Optimization of Multimodal Functions," Journal of Heuristic, vol. 6(2), 191–213, 2000.
[9] Y. Chen, "Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe," doctoral dissertation, Univ. Paris 7, 2013.
[10] Y. Chen and P.Q. Nguyen, "BKZ 2.0: better lattice security estimates," ASIACRYPT 2011, LNCS, vol. 7073, pp 1–20, 2011.
[11] K.A. DeJong, "Adaptive System Design: A Genetic Approach," IEEE Transactions on SMC, pp 566–574.
[12] R. Durrett, "Probability: Theory and Examples," Cambridge University Press, 2010.
[13] A.E. Eiben, E.H.L. Aarts, and K.M. Van Hee, "Global Convergence of genetic algorithms: A Markov chain analysis," Parallel Problem Solving from Nature, pp. 4–12, 1991.
[14] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," Mathematics of Computation, vol. 44(170), pp. 463–471, 1985.
[15] The fpLLL development team. fpLLL, a lattice reduction library, 2016. available from ⟨https://github.com/fplll/fplll⟩.
[16] N. Gama, P. Q. Nguyen, and O. Regev, "Lattice enumeration using extreme pruning," EUROCRYPT 2010, LNCS, vol. 6110, pp. 257–278, 2010.
[17] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning," Addison-Wesley, 1989.
[18] O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert, "Approximating shortest lattice vectors is not harder than approximating closest lattice vectors," Information Processing Letters, vol. 71(2), pp. 55–61, 1999.
[19] J.H. Holland, "Adaption in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence," MIT Press, 1992.
[20] R. Kannan, "Improved algorithms for integer programming and related lattice problems," Proc. 15th ACM STOC, pp. 193–206, 1983.
[21] A.K. Lenstra, H.W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," Math. Ann., vol. 261, pp. 513–534, 1982.
[22] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," SIGSAM Bull., vol. 15(1), pp. 37–44, 1981.
[23] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," Proc. 37th ACM STOC, pp. 84–93, 2005.
[24] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithm," Trans. Neur. Netw., vol. 5(1), pp. 96–101, 1994.
[25] C.-P. Schnorr and M. Euchner, "Lattice basis reduction: improved practical algorithms and solving subset sum problems," Mathematical Programming, vol. 66, 181–199, 1994.
[26] C.-P. Schnorr and H.H. Hörner, "Attacking the Chor-Rivest cryptosystem by improved lattice reduction," EUROCRYPT 1995, LNCS, vol. 921, pp. 1–12, 1995.
[27] P.W. Shor, "Polynomial-Time for Prime Factorization and Discrete Logarithms on a Quantum Computer," SIAM Journal of Computing, vol. 26(5), 1484–1509, 1997.