Research Report

An optimization approach for the discrete logarithm problem

Youvin Jayasinghe

Keywords
Scientific computing, Computer security, Discrete mathematics

Abstract

The discrete logarithm problem has remained challenging to tackle, resulting in its wide use in cryptography. The only proven way to solve the problem in polynomial time is through Shor's algorithm, which runs on quantum computers, but present-day quantum computers are subjected to quantum errors when implementing Shor's algorithm. However, quantum annealers such as the D-Wave machine have come a long way. Further, another problem similar to the discrete logarithm problem, the prime factoring problem, has shown much progress on quantum annealers. In this context, it is encouraging to see the tractability of the discrete logarithm problem on quantum annealers. Further, the problem is scarcely attempted as an optimization.

In this work, we have represented a conversion of the discrete logarithm problem over the multiplicative group integer modulo and the elliptic curve discrete logarithm problem to an optimization problem, then to a binary quadratic form accepted by quantum annealers. Further, we tested our formulation for small scale problems successfully and discussed the complexities suggesting areas of improvement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data transfer is the cornerstone of modern-day telecommunications. To ensure the privacy of the transfers, the area of mathematics and computer science named cryptography is used. This process can be summarized into two parts: encryption or locking the data and the process of decryption or unlocking the data to access. The encryption and decryption method needs a key to be exchanged between the receiver and sender, and practically meeting up to share keys is almost impossible. That is where public-key cryptography systems came into existence. Here, no physical key exchange happens, and both the receiver and the sender come up with a shared key separately. The shared key is based on a mathematical problem that is easy to compute and difficult to reverse, called one-way functions. This is where the discrete logarithm problem(DLP) comes into use.

The discrete logarithm problem for a group G with elements α and $\beta = \alpha^n$ is finding n given α and β. Here α^n indicate conducting the group operator n times. Let us look into a key exchange protocol using a public key cryptography system based on the DLP to understand the cryptography application better. Whitfield Diffie and Martin Hellman introduced the Diffie-Hellman key exchange protocol [1]. Alice and Bob set up their private and public keys based on a group G with element α. First, Alice and Bob randomly choose their private keys $a, b \in \mathbb{N}$ respectively. Next, Alice sends α^a to Bob, and Bob sends α^b to Alice; these are the public keys. Finally, both calculate α^{ab} by applying their respective private keys. Note that for a hacker trying to infiltrate the information, the hacker should compute α^{ab}; that is, the hacker needs to find a given α^a or b given α^b, which is the discrete logarithm problem. Fortunately, the hard nature of the problem makes the data inaccessible to the hacker.

The most obvious method is to find the solution is brute force using different values until we find the solution. Still, it will be extremely difficult with groups with a very large number of elements. There are various algorithms developed to tackle the problem [2][3][4], but none can take the problem efficiently. Therefore we look into a more non-conventional approach. In this regard, we look into quantum computing, a recent advancement in computer science [5; 6; 7; 8; 9].

Considering the different frameworks in quntum computing, it is quantum annealing [10; 11] that is of our interest. These resources use an optimization process to find the global minimum. Whether the exact heuristics used in annealers are not clear, they have shown large improvements in the past decade[12] [13]and have solved mathematical computationally difficult problems [14][15]. The prime factoring problem(PFP) or factoring product of two large primes is also hard as the DLP. The two problems are closely related [16] and PFP has shown progress on quantum annealers [17][18][19]. Hence, we were motivated to see an implementation of the DLP on quantum annealers.

The quantum annealers accept problems in a specific format known as the quadratic unconstrained binary optimization problem(QUBO). The main objective is converting the DLP to a QUBO. We archive this feat in two phases. In the first phase, we transform the problems into pseudo-Boolean optimization problems(PBO). Then, in the next phase, we convert the problems into QUBO.

Finally, we convert a few small scale problems to QUBO problems. Then implement the problems in advantage performance update 4.1, an annealer provided by D-Wave and afterwards implement in hybrid solver for general BQM, which produce better results.

We discuss the preliminaries required for the formulation in chapter 1. In chapter 2, we present current related literature. Then we restate our problems as quadratic unconstrained binary optimization problems(QUBO) in chapter 3. In chapter 5 we present the implementation of the QUBO on annealing and experiment. Finally, in chapter 6, we briefly discuss the theoretical bounds of our formulation and provide our concluding remarks.

Chapter 2

Preliminaries

Summary

We focus on two desecrate logarithm problems(DLP). The first problem is the DLP over a multiplicative group of integer modulo which is; solve for x an integer, for a given prime p, generator α and $\beta \in \mathbb{Z}_p^*$, where $\alpha^x = \beta \pmod{p}$. Next we look in to Elliptic curve discrete logarithm problem. This problem is based on finite groups defined on the elliptic curve with an addition operator. The elliptic curve discrete logarithm problem is; for an elliptic curve $y^2 = x^3 + ax + b \pmod{p}$, solve for n given P and R, points on the elliptic curve such that $nP = R$. Note that R should be a point generated by P. The order of the group generated by P can be found from baby step giant step algorithm.

2.1 Groups and the the DLP over a multiplicative group of integer modulo

Consider A a nonempty set with a binary operator $*$ defined from $A \times A$ to A. For $\alpha, \beta \in A$, $\alpha * \beta$ denotes the results after applying the operator. The operator follows the associative law if $(p*q)*r = p*(q*r)$, for all $p, q, r \in A$. If for $i \in A$ follows $p*i = i*p = p$ for all p, then i is an identity element. Further, for $p \in A$ if there exist $p' \in A$ such that $p*p' = p'*p = i$ then p' is the inverse of p, given i exist. A set G with a binary operator, having an identity element, follows the associative law, and has an inverse element for all elements, then G is a group. In addition to the above properties, if the group follows commutative property $p*q = q*p$, it is called an abelian group. Further, for H subset of G and H follows the group properties, then H is a subgroup of G.

Consider a group G with identity element i and $p \in G$ with p' the inverse. Then for $n \in \mathbb{Z}$:

$$a^n = \begin{cases} \underbrace{p*p*p*\cdots*p}_{\text{n-times}} & n > 0 \\ i & n = 0 \\ \underbrace{p'*p'*p'*\cdots*p'}_{\text{(-n)-times}} & n < 0 \end{cases} \tag{2.1}$$

A group(G) is cyclic if there is a fixed α for all $\beta \in G$ such that $\alpha^n = \beta \; n \in \mathbb{Z}$. Such elements are called the generators of the group. Further for $\gamma \in G$ the subgroup generated by γ is defined as $\{\gamma^n | n \in \mathbb{Z}\}$. An element of this group is known as a element generated by γ. Multiplicative group of integer modulo p $(\mathbf{Z}_p^*)$ where p is a prime is a cyclic abedian group. This is the base of the DLP over integer modulo p.

2.1.1 The DLP

The discrete logarithm problem is defined as follows: Solve for x an integer, for a given prime p, Generator α and $\beta \in \mathbb{Z}_p^*$, where $\alpha^x = \beta \pmod{p}$. The calculation for given x to find β is straightforward, but no known classical algorithm can solve the DLP in polynomial time, and the problem is considered hard.

2.1.2 Diffie-Hellman key exchange application

Consider Diffie-Hellman key exchange based on the group integer modulo 17 with generator 3. As the private key, Alice chooses 15, and Bob chooses 13. Alice generates $3^{15} \pmod{17} = 6$ and shares 6 with Bob, and Bob generates $3^{13} \pmod{17} = 12$ and shares 12 with Alice. Finally, Alice computes $12^{15} \pmod{17} = 10$ using his private key while Bob computes $6^{13} \pmod{17} = 10$ using his private key. Both came up with the common key 10.

2.2 Elliptic curve over integer modulo p

We define an elliptic curve over integer modulo p ($\mathbf{F}_p$) as,

$$E(\mathbf{F}_p) : \{(x, y) \in \mathbf{F}_p^2 | y^2 \equiv x^3 + ax + b \pmod{p} \ , \ 4a^3 + 27b^2 \not\equiv 0 \pmod{0}\} \cup \mathcal{O} \tag{2.2}$$

Where p is a prime number greater than three and a , $b \in \mathbf{F}_p$. $\mathcal{O}$ denote the point at the ∞. $E(\mathbf{F}_p)$ is represented as $y^2 = x^3 + ax + b \pmod{p}$. Let $P(x_p, y_p)$ and $Q(x_q, y_q)$ be elements of $E(\mathbf{F}_p)$. Then the addition P+Q is defined as in the table 2.1.

Table 2.1: Point addition on elliptic curve $y^2 = x^3 + ax + b \pmod{(p \text{ prime})}$

Condition	Gradient(m)	Coordinate of P+Q
$x_p \neq x_q$	$\frac{y_p - y_q}{x_p - x_q} \pmod{p}$	$x = (m^2 - x_p - x_Q) \pmod{p}$ $y = (-m(x - x_p) - y_p) \pmod{p}$
$x_p = x_q$	$\frac{3x_p^2 + a}{2y_p} \pmod{p}$	$x = (m^2 - 2x_p) \pmod{p}$ $y = (-m(x - x_p) - y_p) \pmod{p}$

Another operator is the scalar multiplier. For a natural number n, the scalar multiplier nP is

$$nP = \underbrace{P + P + P + \cdots + P}_{\text{n-times}} \tag{2.3}$$

To calculate nP, we need to do the addition operator $n-1$ times. If n has k Boolean digits, this way of calculation is $O(2^k)$. However, an algorithm exists that can solve nP in polynomial time $O(k)$ called double and add.

Here n is converted to binary, and the algorithm is defined starting from the most significant bit. The most significant bit is always omitted. Then starting from P, for every bit with value 0, we double the previous result, and if the value 1, we double and add P to the previous result. Finally, we obtain nP in $O(\log n)$, which is better than $O(n)$.

Example 2.2.1. *Consider* $78P$*. Here* $n = 78$ *and the binary representation is* $(1001110)_2$*. Then by following the table 2.2 we can calculate the result for* $78P$*.*

Table 2.2: Double and add algorithm

Bit	Operation	Result
1	omitted	P
0	double	$2P$
0	double	$4P$
1	double and add	$9P$
1	double and add	$19P$
1	double and add	$39P$
0	double	$78P$

2.2.1 The ECDLP

For a given natural number n and a point P on the elliptic curve, we can calculate $nP = R$ in polynomial time using the double and add algorithm; here R is a point on the elliptic. However, there is no known polynomial-time algorithm to calculate n given P and R. This problem is known as the elliptic curve discrete logarithm problem(ECDLP), which is considered hard. Note that R should be a point generated by P.

2.2.2 Subgroups

Elliptic curves($E(\mathbf{F}_p)$) are groups under point addition, and cyclic subgroups are created using scalar multiplication on elliptic curves.

Example 2.2.2. *Consider the elliptic curve $y^2 = x^3 + x + 2 \pmod 5$ and the point $P = (1, 3)$.*
Then,

$$\begin{aligned} 1P &= (1,3) \\ 2P &= (4,0) \\ 3P &= (1,2) \\ 4P &= \mathcal{O} \\ 5P &= (1,3) \\ 6P &= (4,0) \\ 7P &= (1,2) \\ 8P &= \mathcal{O} \\ &\vdots \end{aligned}$$

Here we can see the cyclic subgroup with four distinct elements generated by point $P = (1, 3)$.

The number of points on the subgroup is called the order of the subgroup, and it is the smallest positive integer n such that $nP = \mathcal{O}$. Clearly, the order is four for the example 2.2.2. However, for large scale problems listing the elements to find the order will be challenging. Hence, the important question is, what is the order of the subgroup generated by a point P on an elliptic curve? To answer this question, we follow the baby step giant step algorithm 2.2.1.

Algorithm 2.2.1. *Given $P \in E(\mathbf{F}_p)$ to compute the order of the subgroup generated by P.*

1. *Choose an integer $m > P^{\frac{1}{4}}$.*

2. *Compute and save jP for all $j \in \{0, 1, 2, 3......, m\}$ (baby step).*

3. *Compute $(p + 1)P$ and save as Q.*

4. *Compute the points $Q + k(2m)P$ for $k \in \{-m, -m + 1, ..., m\}$ until $Q + k(2m)P = \pm jP$ for some j (giant step).*

5. *Let's say the above relation holds at $k = k_0$ and $j = j_0$. Then choose $M = p + 1 + 2mk_0 \pm j_0$, such that $MP = \mathcal{O}$.*

6. *Factor M. Say $q_1, q_2, ..., q_r$ be distinct prime factors of M. If $\frac{M}{q_i}P = \mathcal{O}$ replace M with $\frac{M}{q_i}$ and repeat the step until $\frac{M}{q_i}P \neq \mathcal{O}$, for all q_i. Then M is the order of the subgroup.*

2.2.3 Diffie-Hellman key exchange application

Consider the elliptic curve $y^2 = x^3 + x + 1 \pmod 5$ and point $P = (0, 1)$ as the generator. The order of the subgroup generated by P is known to be 9. Alice chooses 2 as his private key and computes $Q_a = 2p = (4, 2)$. Bob chooses 4 as his private key and computes $Q_b = 4p = (3, 4)$. Alice shares Q_a with Bob, and Bob computes $4Q_a = (0, 4)$ while Bob shares Q_b with Alice, and Alice computes $2Q_b = (0, 4)$. Both come up with the common key (0,4).

Chapter 3

State of the Art

Conversion of the DLP to a quadratic pseudo-Boolean function is challenging as the DLP, and the Quadratic pseudo Boolean function has no direct relationship. Moreover, even presenting as an optimization problem is challenging. Hence, only a few attempts are available as an optimization approach in the literature. Consider the DLP $\alpha^x = \beta \pmod{p}$; p is a prime, where α is the generator of $\mathbb{Z}_p^*$and $\beta \in \mathbb{Z}_p^*$. Laksari et al.[20] restated the problem as an optimization as in the following.

$$\mathbf{g}(x) = \alpha^x - \beta \pmod{p} \tag{3.1}$$

The optimization process was carried out with heuristics named particle swarm optimization. Later by Mistra et al.[21], a better performing heuristic of the firefly algorithm was used on an optimization representation of the DLP. The optimization formulation is the following.

$$\mathbf{g}(x) = (\beta - \alpha^x) \pmod{p} \tag{3.2}$$

These approaches show the potential of DLP over integer modulo (p prime) being tackled by an optimization. However, there are no optimization attempts on

ECDLP found in the literature, but heuristic attacks can be found applicable to both DLPs. Including Kijma et al. [22] collision random walk approach, Barbulescu et al.[23] and Joux et al. [24] approaches resulting quasi-polynomial heuristics. Furthermore, for ECDLP in 2012, Montgomery et al. [25] made a heuristic implementation of the DLP on PlayStation 3 game consoles resulting in approximately 30% speedups over more traditional methods. These heuristics are not as powerful as in quantum annealers.

As another attempt, we can find quantum computer implementations of the DLP in the literature. After the discovery of the Shor's algorithm [26], an algorithm that can handle the DLP over integer modulo on quantum computers physical applications of different scales were carried out [27][28][29]. Accordingly, it was shown that the DLP is solvable efficiently in polynomial time. Furthermore, this algorithm was extended to handle the ECDLP as well[30]. Whether the algorithm is ground braking, current-day quantum computers cannot handle the algorithm. However, quantum annealers have come a long way, and no implementation of DLP is found in the literature.

Chapter 4

Quadratic formulation

Summary

The QUBO formation for a given DLP is twofold. The initial phase is to convert the problem into pseudo-Boolean optimization problem(PBO). Next is quadratic reformulation. First, the PBO is converted to a polynomial through the min-term normal form. Afterwards, the formulation is converted to a quadratic unconstrained binary optimization problem(QUBO) using process called quadratization. This formation is accepted by the quantum annealers.

4.1 Pseudo-Boolean optimization formulation of DLP over a multiplicative modular group

Recall the DLP over a multiplicative modular group Z_p^* .

$$\alpha^x = \beta \pmod{p} \tag{4.1}$$

Solve for x, where p is a prime number. α is a generator of Z_p^* and $\beta \in Z_p^*$. It is not hard to see that we can restate the problem with k a whole number as $\alpha^x = kp+\beta$. We convert this to an optimization problem(minimization problem) as follows.

$$\mathbf{g}(x,k) = (\alpha^x - \beta - kp)^2 \tag{4.2}$$

Now $\mathbf{g}(x,k) \geq 0$ and at minimum $\mathbf{g}(x,k) = 0$. Let's say $\mathbf{g}(x_0,k_0) = 0$ then, $\alpha^{x_0} - \beta - k_0 p = 0$ and $\alpha^{x_0} = \beta \pmod{p}$. Hence we find the solution by minimizing 4.2.

Next, we convert the problem into pseudo-Boolean optimization(PBO) problem. For which we need to represent x and k as Boolean variables. Now $1 \leq x \leq p-1$ and $k \leq \frac{\alpha^{p-1}}{p}$. Then the respective Boolean representations are as follows.

$$x = x_1 + 2x_2 + \cdots + 2^n x_{n+1}, \tag{4.3}$$

$$k = k_1 + 2k_2 + \cdots + 2^m k_{m+1} \tag{4.4}$$

Here $n = \lfloor \log_2 \{p-1\} \rfloor$, $m = \lfloor \log_2 \{\frac{\alpha^{p-1}}{p}\} \rfloor$ and x_i,k_j are Boolean variables. Finally we represent function 4.2 in pseudo-Boolean form.

$$\mathbf{g}(x,k) = (\alpha^{x_1+2x_2+\cdots+2^n x_{n+1}} - \beta - (k_1 + 2k_2 + \cdots + 2^m k_{m+1})p)^2 \tag{4.5}$$

4.1.1 Example

Consider the problem $2^x = 2 \pmod{3}$. We restate the problem as $\mathbf{g}(x,k) = (2^x - 2 - 3k)^2$. Clearly as $x \leq 2$ and $k \leq \frac{2^2}{3}$ we require two binary variables for x and one for k. Say x_1, x_2 and k_1 respectively. Accordingly the corresponding PBO function is follows.

$$\mathbf{g}(\mathbf{x_1}, \mathbf{x_2}, \mathbf{k_1}) = (2^{2x_2+x_1} - 2 - 3k_1)^2 \tag{4.6}$$

4.2 Polynomial transformation

This section discusses how we convert the PBO to polynomial pseudo-Boolean optimization. The polynomial pseudo-Boolean function is multi-linear as for any binary variable x_i, $x_i^n = x_i$. Initially, we define the characteristic equation as the following.

$$\chi_a(\mathbf{x}) = \prod_{a_i=1} x_i \prod_{a_j=0} (1 - x_j), \tag{4.7}$$

Notice that $\chi_a(\mathbf{x}) = 1$ only when $x = \alpha$ and $\chi_a(\mathbf{x}) = 0$ otherwise. Then the polynomial transformation or the *min-term normal form* [31] is

$$\mathbf{f}(\mathbf{x}) = \sum f(a)\chi_a(\mathbf{x}). \tag{4.8}$$

4.2.1 Example

Consider the PBO obtained in the previous section (equation4.6). Here three Boolean variables are present. The following table 4.1 shows the characteristic equation and its coefficients.

Table 4.1: Min–terms for Equation 4.6

α	x_1	x_2	k_1	$\chi_\alpha(\mathbf{x_1}, \mathbf{x_2}, \mathbf{k_1})$	$\mathbf{g}(x_1, x_2, k_1)$
000	0	0	0	$(1-x_1)(1-x_2)(1-k_1)$	1
001	0	0	1	$(1-x_1)(1-x_2)(k_1)$	16
010	0	1	0	$(1-x_1)(x_2)(1-k_1)$	0
011	0	1	1	$(1-x_1)(x_2)(k_1)$	9
100	1	0	0	$(x_1)(1-x_2)(1-k_1)$	4
101	1	0	1	$(x_1)(1-x_2)(k_1)$	1
110	1	1	0	$(x_1)(x_2)(1-k_1)$	36
111	1	1	1	$(x_1)(x_2)(k_1)$	9

Then using equation 4.8, we obtain the following.

$$\begin{aligned}
\mathbf{g}(x_1, x_2, k_1) = & 1(1-x_1)(1-x_2)(1-k_1) \\
& + 16(1-x_1)(1-x_2)(k_1) \\
& + 0(1-x_1)(x_2)(1-k_1) \\
& + 9(1-x_1)(x_2)(k_1) \\
& + 4(x_1)(1-x_2)(1-k_1) \\
& + 1(x_1)(1-x_2)(k_1) \\
& + 36(x_1)(x_2)(1-k_1) \\
& + 9(x_1)(x_2)(k_1)
\end{aligned} \tag{4.9}$$

After simplification we get a polynomial pseudo-Boolean function.

$$\mathbf{g}(\mathbf{x}, \mathbf{k}) = 1 - x_1 + 3x_2 + 15k_1 + 33x_1x_2 - 18x_2k_1 - 6x_1k_1 - 18x_1x_2k_1. \tag{4.10}$$

4.3 Quadratic pseudo-Boolean formulation

Once we obtain a polynomial pseudo-Boolean function, we must reduce the order to two to obtain a quadratic Boolean unconstrained problem(QUBO). The degree reduction process is known as quadratization. There are several techniques developed to handle quadratization. We utilized three main quadratization techniques in QUBO formation: Rosenberg's quadratization, Freedman and Drineas quadratization and Ishikawa's quadratization.

4.3.1 Rosenberg's quadratization

Rosenberg's quadratization [32] technique is a general technique applicable to every pseudo-Boolean function, given that it is in multi-linear(polynomial) form.

The method is based on an iterative procedure. Initially, we choose two Boolean variables from a chosen higher-order monomial(term), say x_i and x_j. Then replace all occurrences of $x_i x_j$ with $y_{i,j}$ an auxiliary Boolean variable. To ensure $y_{i,j} = x_i x_j$ when minimized, we add a penalty term $M(x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij})$ where M is a sufficiently large positive penalty factor. Which is illustrated by the table 4.2.

Table 4.2: Verification of the quadratization

x_i	x_j	y_{ij}	$x_i x_j$	$x_i y_{ij}$	$x_j y_{ij}$	$x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij}$
0	0	0	0	0	0	0
0	0	1	0	0	0	3
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	1	1	1	0

Finally, we repeat the above steps till we obtain a quadratic formulation.

4.3.1.1 Example

Consider the polynomial pseudo-Boolean function 4.10 we obtained.

$$\mathbf{g}(\mathbf{x_1}, \mathbf{x_2}, \mathbf{k_1}) = 1 - x_1 + 3x_2 + 15k_1 + 33x_1x_2 - 18x_2k_1 - 6x_1k_1 - 18x_1x_2k_1. \quad (4.11)$$

There is only one monomial with an order higher than two. From which, we replace x_1x_2 with y.

$$\mathbf{g}(x_1, x_2, k_1, y) = 1 - x_1 + 3x_2 + 15k_1 + 33x_1x_2 - 6k_1x_1 - 18k_1x_2 - 18k_1y \quad (4.12)$$

Afterwards, we add in the penalty term with penalty factor M.

$$\mathbf{g}(x_1, x_2, k_1, y) = 1-x_1+3x_2+15k_1+33x_1x_2-6k_1x_1-18k_1x_2-18k_1y+M(x_1x_2-2x_1y-2x_2y+3y) \quad (4.13)$$

Let's set M=101.

$$\mathbf{g}(x_1, x_2, k_1, y) = 1 - x_1 + 3x_2 + 15k_1 + 303y + 134x_1x_2 - 6k_1x_1 - 18k_1x_2 - 202x_1y - 202x_2y - 18k_1y \quad (4.14)$$

4.3.1.2 Complexity analysis

Rosenberg's quadratization technique adds large positive coefficients to the pseudo-Boolean function and is considered to have a negative impact on the computational performance [33]. Further, when we apply the Rosenberg's method to a monomial, each iteration results in the degree to reduce by one, adding an auxiliary variable. Therefore we require $n - 2$ auxiliary variable for each monomial with degree n.

This has less meaning when in applications because the Rosenberg's method is not a term-wise quadratization technique. For a given pseudo-Boolean function, we require $O(2^n)$ auxiliary variables.

Proof. Consider a pseudo-Boolean function with n Boolean variables and having all possible combinations as monomials. Notice that when we take all possible combinations of two variables in Rosenberg's method, we reduce degree three and four monomials to two. Also, after we reduce the order of all monomials with initial degree $r \in 3, 4, ...$ to two, all degree $r+1$ monomials reduce to degree three and require only one auxiliary variable to reduce the degree to two. Therefore we require a maximum of $\binom{n}{r}$ additional auxiliary variables for all monomials of degree r. We follow these facts in table 4.3

Table 4.3: Order of the Rosenberg's method

Degree of the monomials	Max number of auxiliary variables
less than 4:	$\binom{n}{2}$
degree 5:	$\binom{n}{5}$
$\vdots$	$\vdots$
degree n-1:	$\binom{n}{n-1}$
order n:	$\binom{n}{n}$

Then the maximum number of auxiliary variables required is $\binom{n}{2} + \sum_{r=5}^{n} \binom{n}{r} \leq 2^n$, hence the Rosenberg's quadratization is $\mathbf{O}(2^n)$ $\square$

4.3.2 Freedman and Drineas quadratization

Freedman and Drineas quadratization[34] is a term-wise quadratization technique applicable to negative monomials with a degree greater than two. For a negative monomial $-\prod_{i=1}^{n} x_i$ following is the representation of the quadratization. Where x_i, $i \in \{1, 2....n\}$ are Boolean variables, and y is a auxiliary Boolean variable.

$$-\prod_{i=1}^{n} x_i = \min_{y \in \{0,1\}} (n-1)y - \sum_{i=1}^{n} x_i y \tag{4.15}$$

4.3.2.1 Example

Consider the polynomial pseudo-Boolean function 4.10 we obtained.

$$\mathbf{g}(\mathbf{x_1}, \mathbf{x_2}, \mathbf{k_1}) = 1 - x_1 + 3x_2 + 15k_1 + 33x_1x_2 - 18x_2k_1 - 6x_1k_1 - 18x_1x_2k_1. \quad (4.16)$$

There is only one monomial($-18x_1x_2k_1$) with a degree greater than two, which is negative. Therefore we apply the quadratization to this monomial only.

$$-18k_1x_1x_2 \to 18\{2y - (x_1y + x_2y + k_1y)\} \quad (4.17)$$

By substituting in 4.16, we obtain the QUBO as the following.

$$\mathbf{g}(x_1, x_2, k_1, y) = 1 - x_1 + 3x_2 + 15k_1 + 36y + 33x_1x_2 - 6k_1x_1 - 18k_1x_2 - 18x_1y - 18x_2y - 18k_1y \quad (4.18)$$

4.3.2.2 Complexity analysis

For a given monomial with higher order(Grater than 2) we only require one auxiliary variable. However, we require $\mathbf{O}(2^n)$ auxiliary variables for pseudo-Boolean function with n binary variables.

Proof. Consider a pseudo-Boolean function containing n variables having all possible combinations as monomials. For $r \in \mathbb{N}$ there are $\binom{n}{r}$ monomials with degree r. Therefore the number of monomials with a degree greater than two is $\sum_{r=3}^{n} \binom{n}{r}$. As for a monomial, we require one auxiliary variable, then the number of auxiliary variables required is $\sum_{r=3}^{n} \binom{n}{r}$ or $\mathbf{O}(2^n)$. □

4.3.3 Ishikawa's quadratization

Ishikawa's quadratization [35] is also a term-wise quadratization technique, applicable to positive monomials with a degree greater than two. For a positive monomial $\prod_{i=1}^{n} x_i$ following is the representation of the quadratization. Where x_i, $i \in \{1, 2....n\}$ are binary variables.

$$\prod_{i=1}^{n} x_i = \min_{y \in \{0,1\}^m} \sum_{i=1}^{m} y_i(c_{i,n}(-|x| + 2i) - 1) + \frac{|x|(|x| - 1)}{2} \tag{4.19}$$

where,

$$|x| = \sum_{i=1}^{n} x_i \text{ and } m = \lfloor \frac{n-1}{2} \rfloor$$

$$c_{i,n} = \begin{cases} 1, & \text{if } n \text{ is odd and } i = m \\ 2, & \text{otherwise} \end{cases}$$

4.3.3.1 Example

Consider the PBO

$$\mathbf{f}(x_1, x_2, x_3) = 9x_1x_2x_3 + 8x_1x_2 - 6x_2x_3 + x1 - 2x_2 + x_3 \tag{4.20}$$

There is only one monomial($9x_1x_2x_3$) with a degree greater than 2, which is positive. Therefore we apply the quadratization to this monomial only. Now,

$$\begin{aligned} |x| \quad &= x_1 + x_2 + x_3 \\ m &= \lfloor \frac{3-1}{2} \rfloor = 1 \\ c_{1,3} &= 1 \end{aligned} \tag{4.21}$$

Then by substituting in 4.19,

$$9x_1x_2x_3 \to 9\{y + x_1x_2 + x_1x_3 + x_2x_3 - x_1y - x_2y - x_3y\} \tag{4.22}$$

then the QUBO for 4.20 is the following.

$$\mathbf{f}(x_1, x_2, x_3, y) = x_1 - 2x_2 + x_3 + 9y + 17x_1x_2 + 9x_1x_3 + 3x_2x_3 - 9x_1y - 9x_2y - 9x_3y \tag{4.23}$$

4.3.4 Complexity analysis

For a given monomial of degree n grater than two we require $\lfloor \frac{n-1}{2} \rfloor$ auxiliary variables. Further, we require $\mathbf{O}(n2^n)$ auxiliary variables for pseudo-Boolean function with n binary variables.

Proof. Consider a pseudo-Boolean function containing n variables having all possible combinations as monomials. For $r \in \mathbb{N}$ there are $\binom{n}{r}$ monomials with degree r. Therefore the number of monomials with a degree greater than two is $\sum_{r=3}^{n} \binom{n}{r}$. Hence,

$$\begin{aligned}
\text{Number of auxiliary variables} &= \sum_{r=3}^{n} [\lfloor \frac{r-1}{2} \rfloor \binom{n}{r}] \\
&\leq \sum_{r=3}^{n} [\lfloor \frac{n-1}{2} \rfloor \binom{n}{r}] \\
&\leq n \sum_{r=3}^{n} \binom{n}{r} \\
&\leq n2^n
\end{aligned}$$

Therefore, Ishikawa's quadratization requires $\mathbf{O}(n2^n)$ auxiliary variables. □

4.4 QUBO formulation for DLP over elliptic curves.

Recall the elliptic curve discrete logarithm problem (ECDLP).

$$nP = R \tag{4.24}$$

Solve for n, where $E : y^2 = x^3 + ax + b \pmod{p}$, p a prime number grater than 3 and $P, R \in E$, $R \neq \mathcal{O}$(point at infinity) The approach we use here is similar to the methodology in section 4.1. First, we need to convert the problem into an optimization problem. We achieve this using norm properties. Consider,

$$\mathbf{g}(n) = \begin{cases} \|nP - R\|_E^2, & nP \neq \mathcal{O} \\ M, & nP = \mathcal{O} \end{cases} \tag{4.25}$$

M is a positive constant and $\|\|_E$ represents the Euclidean norm. From norm properties, we know that $\mathbf{g}(n) \geq 0$ and $\mathbf{g}(n) = 0$ if and only if $nP = R$. Therefore by minimizing 4.25, we obtain the solution. Next, we need to convert the problem into a PBO. For which we need a Boolean representation of n. To represent n as Boolean variables, we need to know the bounds of n or the subgroup order generated by P. We use the baby step giant step algorithm 2.2.1 to determine the order of the subgroup. Let us say the order is N. Then the Boolean representation is as follows.

$$n = 2^0 x_1 + 2^1 x_2 + \cdots + 2^m x_{m+1} \tag{4.26}$$

Where $m = \lfloor \log_2(N-1) \rfloor$ and x_i are Boolean variables for all $i \in \{1, 2, \cdots, m\}$.

With that the PBO is the following.

$$\mathbf{g}(x) = \begin{cases} \|\{2^0x_1 + 2^1x_2 + \cdots + 2^m x_{m+1}\}P - R\|_E^2, & n \neq 0 \\ M, & n = 0, N \end{cases}, \; x \in \{0,1\}^m \tag{4.27}$$

Finally we follow 4.1 and section to obtain a QUBO.

4.4.1 Example

Consider the problem $nP = R$ over an elliptic curve $E : x^2 = y^3 + x + 2 \pmod 5$, where $P = (3, 6)$ and $Q = (80, 87)$. The order of the subgroup generated by P is 5. The optimization problem is the following.

$$\mathbf{g}(n) = \begin{cases} \|nP - R\|_E^2, & nP \neq \mathcal{O} \\ 100, & nP = \mathcal{O} \end{cases} \tag{4.28}$$

Since $n < 5$ we require 3 Boolean variables to represent n. Accordingly we can represent n as the following.

$$n = n_1 + 2n_2 + 4n_3 \tag{4.29}$$

Here n_1, n_2, n_3 are Boolean variables. Further, the PBO is the following.

$$\mathbf{g}(n_1, n_2, n_3) = \begin{cases} \|(n_1 + 2n_2 + 4n_3)P - R\|_E^2, & n_1 + 2n_2 + 4n_3 \neq 0, 5 \\ 100, & n_1 + 2n_2 + 4n_3 = 0, 5 \end{cases} \tag{4.30}$$

The following table 4.4 shows the characteristic equation and its coefficients.

Table 4.4: Min–terms for Equation 4.30

α	n_1	n_2	n_3	$\chi_\alpha(\mathbf{n_1}, \mathbf{n_2}, \mathbf{n_3})$	$\mathbf{g}(n_1, n_2, n_1)$
000	0	0	0	$(1-n_1)(1-n_2)(1-n_3)$	100
001	0	0	1	$(1-n_1)(1-n_2)(n_3)$	5929
010	0	1	0	$(1-n_1)(n_2)(1-n_3)$	5929
011	0	1	1	$(1-n_1)(n_2)(n_3)$	12490
100	1	0	0	$(n_1)(1-n_2)(1-n_3)$	12490
101	1	0	1	$(n_1)(1-n_2)(n_3)$	100
110	1	1	0	$(n_1)(n_2)(1-n_3)$	0
111	1	1	1	$(n_1)(n_2)(n_3)$	5929

Then using equation 4.8, we obtain the following.

$$\begin{aligned}\mathbf{g}(n_1, n_2, n_3) =&100(1-n_1)(1-n_2)(1-n_3)\\ &+5929(1-n_1)(1-n_2)(n_3)\\ &+5929(1-n_1)(n_2)(1-n_3)\\ &+12490(1-n_1)(n_2)(n_3)\\ &+12490(n_1)(1-n_2)(1-n_3)\\ &+100(n_1)(1-n_2)(n_3)\\ &+0(n_1)(n_2)(1-n_3)\\ &+5929(n_1)(n_2)(n_3)\end{aligned} \tag{4.31}$$

After simplification we obtain the multi-linear PBO.

$$\begin{aligned}\mathbf{g}(n_1, n_2, n_3) =&100+12390n_1+12390n_2+5829n_3\\ &-18319n_1n_2-18219n_1n_3+732n_2n_3+17587n_1n_2n_3\end{aligned} \tag{4.32}$$

Finally, replacing n_1, n_2 with y, the auxiliary variable in Rosenberg's quadratization 4.3.1 to obtain a QUBO. Here we choose the penalty factor to be 17587.

$$\begin{aligned}\mathbf{g}(n_1, n_2, n_3, y) =& 100 + 12390n_1 + 5829n_2 + 5829n_3 + 34442y + 17587n_1n_2 \\ & - 18219n_1n_3 - 35174n_1y + 732n_2n_3 - 35174n_2y + 17587n_3y\end{aligned} \tag{4.33}$$

Chapter 5

Experimental Results

5.1 QUBO to anneling

Once the QUBO formulation is done, the problem is converted into a matrix and directly input into the quantum annealing . The matrix form is defined as the following.

$$\text{Minimize } \mathbf{x}^T Q \mathbf{x}$$
$$\text{subject to } \mathbf{x} \in S$$

Where S is a set of Boolean variables and Q represent the coefficient matrix derived from the QUBO. This matrix is known as *Q-matrix* [36]. The formulation is better understood with example 5.1.1.

Example 5.1.1. *Consider the QUBO 4.14,*

$$\mathbf{g}(x_1, x_2, k_1, y) = 1-x_1+3x_2+15k_1+303y+134x_1x_2-6k_1x_1-18k_1x_2-202x_1y-202x_2y-18k_1y \tag{5.1}$$

The solution is not effected by the constant therefore we omit it. The QUBO consists of a linear part $-x_1 + 3x_2 + 15k_1 + 303y$ and a quadratic part (monomials *of order 2)* $134x_1x_2 - 6k_1x_1 - 18k_1x_2 - 202x_1y - 202x_2y - 18k_1y$. *Notice that for any Boolean variable* $x = x^2$*, therefore the linear part can be restated as,*

$$-x_1^2 + 3x_2^2 + 15k_1^2 + 303y^2 \tag{5.2}$$

Then we can rewrite in the matrix form.

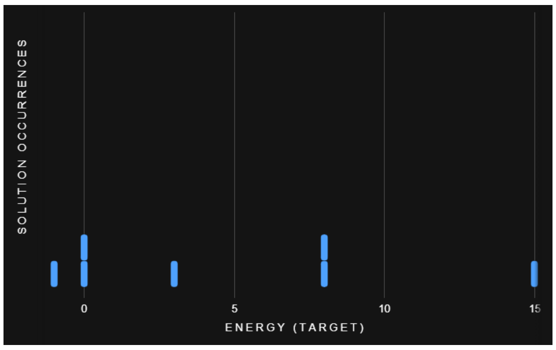
$$\mathbf{g}(x_1, x_2, k_1) = \begin{pmatrix} x_1 & x_2 & k_1 \end{pmatrix} \begin{bmatrix} -1 & 134 & -6 & -202 \\ 0 & 3 & -18 & -202 \\ 0 & 0 & 15 & -18 \\ 0 & 0 & 0 & 303 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ k_1 \end{bmatrix} \tag{5.3}$$

Once the Q-matrix is constructed, we need to embed the problem into the annealing hardware. Consider variables that take instances -1 and +1 called Ising or spin variables. A Boolean variable (x) can be converted to a Ising variable (s) through $s = 1 - 2x$. We obtain an Ising model by converting all Boolean variables in the QUBO to Ising variables. These models can define an instance in physical qubits in the annealing through a physical Hamiltonian. The physical Hamiltonian is defined as the sum of the initial Hamiltonian and the problem Hamiltonian. Initially, the annealing starts at the lowest energy level of the initial Hamiltonian. Finally, the state corresponding to the lowest energy state of the problem Hamiltonian is output as the solution.

5.2 Results

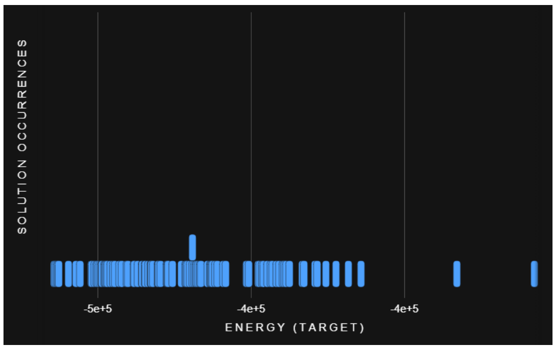
Initial tests were carried out with Advantage performance update 4.1, annealing provided by D-Wave. The parameters were set to take 100 reads(solution outputs) with $20\mu s$ for each read. Once the reads were completed, we received a sample set of solutions from which the sample with the lowest energy was picked. For the example QUBO 4.14 we obtained using Rosenberg's method a sample set of eight solutions. The corresponding energy of the problem Hamiltonian is shown in figure 5.1. The solution corresponding to the lowest energy occurred 15 times

Figure 5.1: Energy of the problem Hamiltonian and number of reads of the sample set ($2^x = 2 \pmod 3$)



out of 100, and the solution was accurate. Consider the problem $3^x = 3 \pmod 5$. Once the QUBO formation was done, we submitted the problem to the annealing. The energies of the problem Hamiltonian are shown in the figure 5.2. Here, the

Figure 5.2: Energy of the problem Hamiltonian and number of reads of the sample set ($3^x = 3 \pmod 5$)

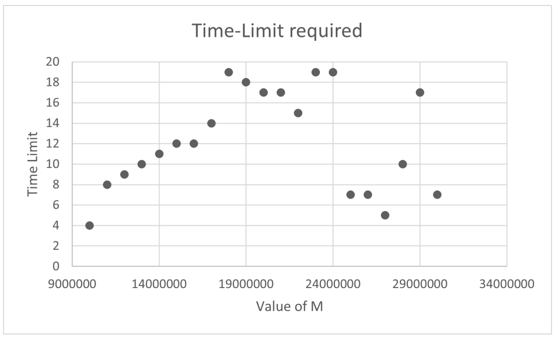


accurate solution did not correspond to the least energy, but the solution was among the sample set.

The hybrid solver for general BQM overcame this issue. D-wave provides this annealing. The hybrid name suggests using classical computing resources(CPUs) and Quantum resources(QPUs). As inputs, we only need to input a time limit other than the QUBO. The minimum time limit we can set is 3 seconds. As outputs, we receive a single solution.

While carrying out tests with this annealing, we observed that the coefficient in the QUBO affects the time limit. To further test, we used different penalty factors in Rosenberg's method 4.3.1 and checked the minimum time we can obtain the accurate solution consistently. Figure 5.3 shows how the time limits were set with the change of the penalty factor(M) for equation 4.13. As we can see,

Figure 5.3: Effect of penalty factor in Rosenberg's method on the time limit for equation 4.13



there was no evident pattern, and this randomness continued to other problems. However, setting the penalty factor minimum and keeping the number of variables low allowed us to obtain accurate solutions with a time limit set at 3 sec for most problems.

Chapter 6

Discussion

Because the Q-matrix can be given directly to annealing through PyQUBO[37], a library in Python programming language; we need not consider the complexities of embedding the QUBO on to the annealer. Annealers provide a promising solution to hard problems [17][38]. In fact, we also obtained solutions to small scale problems running on open-source annealers provided by D-Wave. However, there are difficulties in formulating the QUBO and is discussed in this section. Furthermore, a special class of set functions called submodular can be solved in polynomial time. Moreover, in this chapter, we check whether our formulation is submodular.

6.1 Check for submodularity

A set function is a mapping from a set to a real value. For a set function $\mathbf{f}$ defined on a set A, the discrete derivative for $S \subset A$ with respect to $e \in A\{e \notin S\}$ is the following

$$\mathbf{\Delta}_f(e|S) = \mathbf{f}(S \cup e) - \mathbf{f}(S) \tag{6.1}$$

The discrete derivative definition for submodularity is the following.

Definition 6.1.1. *A set function* $\mathbf{f}$ *on set* A *is submodular, if for all* $e \in A$

$$\mathbf{\Delta}_f(e|X) = \mathbf{f}(X \cup e) - \mathbf{f}(X) \tag{6.2}$$

is non-increasing in X

In recent years, efficient algorithms were developed to solve submodular functions[39][40]. Let us look into an example to check whether our formulation is submodular.

Example 6.1.1. *Consider $2^x = 3 \pmod 5$,using the methodology we discussed in section 4.1 let $\mathbf{f}$ be the PBO that represent $(2^x - 3 - k5)^2$ with x, k in Boolean form, $x_1 + 2x_2 + 4x_3$ and $k = k_1 + 2k_2$.*

$$\mathbf{f}(x_1, x_2, x_3, k_1, k_2) = (2^{x_1+2x_2+4x_3} - 3 - (k_1 + 2k_2)5)^2 \tag{6.3}$$

Let $\mathbf{g}(S)$, the set function representation of equation 6.3, where S is the set of Boolean variables corresponding to value 1. Now $2^3 - 3 - (1)5 = 0$, therefore $\mathbf{f}(1, 1, 0, 1, 0) = 0$ and $\mathbf{g}(\{x_1, x_2, k_1\}) = 0$.
Notice that as $\mathbf{g}(\{x_1, x_2, k_1\}) = 0$ and $\mathbf{g}(S) \geq 0$; $\mathbf{\Delta}_g(k_1|\{x_1, x_2\}) < 0$

$$\mathbf{\Delta}_g(k_1|\{x_1, x_2\}) = \mathbf{g}(\{x_1, x_2, k_1\}) - \mathbf{g}(\{x_1, x_2\}) < 0 \tag{6.4}$$

Further, $\mathbf{\Delta}_g(x_3|\{x_1, x_2, k_1\}) > 0$

$$\mathbf{\Delta}_g(x_3|\{x_1, x_2, k_1\}) = \mathbf{g}(\{x_1, x_2, k_1, x_3\}) - \mathbf{g}(\{x_1, x_2, k_1\}) > 0 \tag{6.5}$$

That is

$$\mathbf{\Delta}_g(k_1|\{x_1, x_2\}) < \mathbf{\Delta}_g(x_3|\{x_1, x_2, k_1\}) \tag{6.6}$$

This violate the condition in definition6.1.1, therefore the formulation is not submodular.

A similar set to $\{x_1, x_2, k_1\}$ were the solution to the PBO exist in every formulation. When we consider the discrete derivative through that set as in example 6.1.1, the condition for submodularity violates; therefore, the formulation will never be submodular.

6.2 Pre-computation challengers

Our formulation is twofold as there are two phases, DLP to PBO and then to QUBO. Perhaps the biggest challenge is to optimize the operations in these two phases. Many Boolean variables are required for DLP with a large order cyclic group. However, the increase in required variables is on a log scale. Furthermore, we add in an additional variable(k) in optimization formulation of the DLP over a multiplicative modular group 4.1, resulting in more Boolean variables.

Then in the QUBO formulation phase, we resort to min-term normal form, which we compute through classical computers. The complexity of the constructing min-term normal form is $\mathbf{O}(2^n)$, but our formulation is not the only formulation suffering in pre-computation. The minor embedding problem [41] is a popular example.

Afterwards, we used quadratization techniques to obtain a QUBO. The question to ask is; what is the best quadratization? For the quadratization techniques we discussed in chapter 3 Rosenberg's method and Freedman and Drineas method generate auxiliary variables in $\mathbf{O}(2^n)$ while Ishikawa's method generates $\mathbf{O}(n2^n)$. If all higher-order monomials(> 2) have negative coefficients, then Freedman and Drineas method is better as there is no penalty factor compared to Rosenberg's method. However, when it comes to pseudo-Boolean functions with higher degree monomials having both negative and positive coefficients, the best quadratization is difficult to determine and will depend on the PBO itself.

6.3 Conclusion

We successfully formulated QUBO that is accepted by the annealing. Further, we obtained accurate solutions for small scale problems. DLP based cryptosystems can be concerned about this, but our formulation cost is high. Nevertheless, the initial formulation of the prime factoring problem also suffered in pre-computation, which was later modified to produce promising experimental results. We present our work as an initial step since there has never been a QUBO formation for the DLP and only a few previous optimization approaches. Further, the most costly step is the second phase of our formulation, where we resort to polynomial transformation. It will be interesting to explore other methodologies to work around this stage.

Some results in this thesis were published in [42].

References

[1] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976. 1

[2] P. Wang and F. Zhang, "An efficient collision detection method for computing discrete logarithms with pollard's rho," *Journal of Applied Mathematics*, vol. 2012, 2012. 2

[3] E. Teske, "Speeding up pollard's rho method for computing discrete logarithms," in *International Algorithmic Number Theory Symposium*, pp. 541–554, Springer, 1998. 2

[4] S. D. Galbraith and P. Gaudry, "Recent progress on the elliptic curve discrete logarithm problem," *Designs, Codes and Cryptography*, vol. 78, no. 1. 2

[5] A. Mahasinghe and J. Wang, "Efficient quantum circuits for toeplitz and hankel matrices," *Journal of Physics A: Mathematical and Theoretical*, vol. 49, no. 27, p. 275301, 2016. 2

[6] "Phase-modified ctqw unable to distinguish strongly regular graphs efficiently," *Journal of Physics A: Mathematical and Theoretical*, vol. 48, no. 26, p. 265301, 2015. 2

[7] A. Mahasinghe, J. Wang, and J. Wijerathna, "Quantum walk-based search

and symmetries in graphs," *Journal of Physics A: Mathematical and Theoretical*, vol. 47, no. 50, p. 505301, 2014. 2

[8] A. Mahasinghe, "Vibration analysis of cyclic symmetrical systems by quantum algorithms," *Mathematical Problems in Engineering*, vol. 2019, 2019. 2

[9] A. Mahasinghe, S. Bandaranayake, and K. De Silva, "Solovay–kitaev approximations of special orthogonal matrices," *Advances in Mathematical Physics*, vol. 2020, pp. 1–7, 2020. 2

[10] A. Mahasinghe, R. Hua, M. J. Dinneen, and R. Goyal, "Solving the hamiltonian cycle problem using a quantum computer," in *Proceedings of the Australasian Computer Science Week Multiconference*, pp. 1–9, 2019. 2

[11] A. Mahasinghe, V. Fernando, and P. Samarawickrama, "Qubo formulations of three np problems," *Journal of Information and Optimization Sciences*, vol. 42, no. 7, pp. 1625–1648, 2021. 2

[12] D. Willsch, M. Willsch, C. D. G. Calaza, F. Jin, H. De Raedt, M. Svensson, and K. Michielsen, "Benchmarking advantage and d-wave 2000q quantum annealers with exact cover problems," *arXiv preprint arXiv:2105.02208*, 2021. 2

[13] Y. Chen, C. Quintana, D. Kafri, B. Chiaro, A. Dunsworth, B. Foxen, J. Wenner, J. Martinis, H. Neven, and A. Quantum, "Progress towards quantum annealer v2. 0 i: Hardware," in *APS March Meeting Abstracts*, vol. 2018, pp. C26–008, 2018. 2

[14] M. J. Dinneen, A. Mahasinghe, and K. Liu, "Finding the chromatic sums of graphs using a d-wave quantum computer," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4811–4828, 2019. 2

[15] C. Calude, M. Dinneen, and R. Hua, "Qubo formulations for the graph isomorphism problem and related problems," *Theoretical Computer Science*, vol. 701, pp. 54–69, 2017. doi : 10.1016/j.tcs.2017.04.016. 2

[16] E. Bach, *Discrete logarithms and factoring.* University of California at Berkeley, 1984. 2

[17] S. Jiang, K. A. Britt, A. J. McCaskey, T. S. Humble, and S. Kais, "Quantum annealing for prime factorization," *Scientific reports*, vol. 8, no. 1, pp. 1–9, 2018. 2, 30

[18] B. Wang, F. Hu, H. Yao, and C. Wang, "Prime factorization algorithm based on parameter optimization of ising model," *Scientific reports*, vol. 10, no. 1, pp. 1–10, 2020. 2

[19] A. Shi, H. Guan, and W. Zhang, "Efficient diabatic quantum algorithm in number factorization," *Physics Letters A*, vol. 384, no. 28, p. 126745, 2020. 2

[20] E. Laskari, G. Meletiou, and M. Vrahatis, "The discrete logarithm problem as an optimization task: A first study," in *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*, pp. 1–6, 2004. 10

[21] M. Mishra, U. Chaturvedi, K. K. Shukla, R. V. Yampolskiy, *et al.*, "A modified chaotic firefly algorithm for solving discrete logarithm problem and analysis," in *2015 IEEE Symposium Series on Computational Intelligence*, pp. 1885–1892, IEEE, 2015. 10

[22] S. Kijima and R. Montenegro, "Collision of random walks and a refined analysis of attacks on the discrete logarithm problem," in *IACR International Workshop on Public Key Cryptography*, pp. 127–149, Springer, 2015. 11

[23] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé, "A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic," *Advances in Cryptology – EUROCRYPT 2014*, pp. 1–16, 2014. 11

[24] A. Joux, "A new index calculus algorithm with complexity

$$l(1/4 + o(1))$$

in small characteristic," *Selected Areas in Cryptography – SAC 2013*, pp. 355–379, 2013. 11

[25] J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery, "Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction," *International Journal of Applied Cryptography*, vol. 2, no. 3, pp. 212–228, 2012. 11

[26] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999. 11

[27] B. P. Lanyon, T. J. Weinhold, N. K. Langford, M. Barbieri, D. F. James, A. Gilchrist, and A. G. White, "Experimental demonstration of a compiled version of shor's algorithm with quantum entanglement," *Physical Review Letters*, vol. 99, no. 25. 11

[28] J. J. Vartiainen, A. O. Niskanen, M. Nakahara, and M. M. Salomaa, "Implementing shor's algorithm on josephson charge qubits," *Physical Review A*, vol. 70, no. 1, p. 012319, 2004. 11

[29] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of shor's quantum factoring

algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 6866, pp. 883–887, 2001. 11

[30] J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," *arXiv preprint quant-ph/0301141*, 2003. 11

[31] E. Boros and P. Hammer, "Pseudo-boolean optimization," *Discrete applied mathematics*, vol. 123, no. 1-3, pp. 155–225, 2002. doi : `10.1016/S0166-218X(01)00341-9`. 14

[32] P. Hammer, I. Rosenberg, and S. Rudeanu, "On the determination of the minima of pseudo-boolean functions," *Studii si Cercetari matematice*, vol. 14, pp. 359–364, 1963. 15

[33] E. Rodriguez Heck, *Linear and quadratic reformulations of nonlinear optimization problems in binary variables.* PhD thesis, Université de Liège, Liège, Belgique, 2018. 17

[34] D. Freedman and P. Drineas, "Energy minimization via graph cuts: settling what is possible," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 939–946 vol. 2, 2005. 18

[35] H. Ishikawa, "Higher-order clique reduction in binary graph cut," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. 20

[36] F. Glover, G. Kochenberger, and Y. Du, "Quantum bridge analytics i: a tutorial on formulating and using qubo models," *4OR*, vol. 17, no. 4, pp. 335–371, 2019. 26

[37] M. Zaman, K. Tanahashi, and S. Tanaka, "Pyqubo: Python library for map-

ping combinatorial optimization problems to qubo form," *arXiv preprint arXiv:2103.01708*, 2021. 30

[38] Z. Bian, F. Chudak, R. Israel, B. Lackey, W. G. Macready, and A. Roy, "Discrete optimization using quantum annealing on sparse ising models," *Frontiers in Physics*, vol. 2, p. 56, 2014. 30

[39] D. Chakrabarty, P. Jain, and P. Kothari, "Provable submodular minimization using wolfe's algorithm," *Advances in Neural Information Processing Systems*, vol. 27, 2014. 31

[40] D. Dadush, L. A. Végh, and G. Zambelli, "Geometric rescaling algorithms for submodular function minimization," *Mathematics of Operations Research*, vol. 46, no. 3, pp. 1081–1108, 2021. 31

[41] W. Vinci, T. Albash, G. Paz-Silva, I. Hen, and D. A. Lidar, "Quantum annealing correction with minor embedding," *Physical Review A*, vol. 92, no. 4, p. 042310, 2015. 32

[42] A. Mahasinghe and Y. Jayasinghe, "An initial step toward a quantum annealing approach to the discrete logarithm problem," *Security and Privacy*, vol. 5, no. 4, p. e234, 2022. 33

[43] A. Commeine and I. Semaev, "An algorithm to solve the discrete logarithm problem with the number field sieve," in *International Workshop on Public Key Cryptography*, pp. 174–190, Springer, 2006.

Appendix A

Python code used in experimentation with hybrid solver

```python
import numpy as np
from pyqubo import Binary
from dwave.system import LeapHybridSampler
import sympy as sym
from sympy import *
a=int(input("Enter the Alpha Value "))
b=int(input("Enter Beta value "))
p=int(input("Enter Prime value "))
xmax=int(2**(np.floor(np.log2(p-1))+1))
xvar=int(np.floor(np.log2(p-1))+1)
kmax=int(2**(np.floor(np.log2(a**(p-1)/p))+1))
kvar=int(np.floor(np.log2(a**(p-1)/p))+1)
ListX=list(range(0,xmax))
ListK=list(range(0,kmax))
ListFx=[]
for x in ListX:
    for k in ListK:
        ListFx.append((a**x-b-k*p)**2)

stringX=""
ListVX=[]
for j in range(xvar):
    stringX=stringX+"x"+str(j+1)+" "
stringK=""
ListVK=[]
for j in range(kvar):
    stringK=stringK+"k"+str(j+1)+" "
ListVX=stringX.split(" ")
```

```
ListVK=stringK.split(" ")
ListVX.pop()
ListVK.pop()
x=[]
for i in range(xvar):
    x.append(sym.Symbol(ListVX[i]))
k=[]
for i in range(kvar):
    k.append(sym.Symbol(ListVK[i]))
mintermList=[]
for Xi in range(xmax):
    binaryx=[0]*xvar
    res = [int(i) for i in bin(Xi)[2:]]
    res.reverse()
    for i in range(len(res)):
        binaryx[i]=res[i]
    minx=""
    for i in range(xvar):
        if binaryx[i]==1:
            minx=minx+f"(x[{i}])*"
        else:
            minx=minx+f"(1-x[{i}])*"
    for Ki in range(kmax):
        binaryk=[0]*kvar
        res = [int(i) for i in bin(Ki)[2:]]
        res.reverse()
        for i in range(len(res)):
            binaryk[i]=res[i]
        mink=""
        for i in range(kvar):
            if i<kvar-1:
                if binaryk[i]==1:
                    mink=mink+f"(k[{i}])*"
                else:
                    mink=mink+f"(1-k[{i}])*"
            else:
                if binaryk[i]==1:
                    mink=mink+f"(k[{i}])"
                else:
                    mink=mink+f"(1-k[{i}])"
        mintermList.append(eval(minx+mink))

MinTermN=0
for i in range(len(ListFx)):
    MinTermN=MinTermN+expand(ListFx[i]*mintermList[i])
print(MinTermN)
pnty=""
Itt=MinTermN.copy()
NotQUBO=True
```

```python
count=1
y=[]
M=int(input("Enter a sutable M value"))
while NotQUBO==True:
    xi=input("enter first variable type")
    xiN=int(input("enter first variable number"))
    xj=input("enter second variable type")
    xjN=int(input("enter second variable number"))
    y.append(sym.Symbol(f"y{count}"))
    Itt=Itt.subs({eval(xi)[xiN-1]*eval(xj)[xjN-1]:y[count-1]})
    pnty=pnty+f" +expand(M*({xi}[{xiN-1}]*{xj}[{xjN-1}]-2*y[{count-1}]*{xi}[{xiN-1}]-2*y[{count-1}]*{xj}[{xjN-1}]+3*y[{count-1}]))"
    print("")
    print(pnty)
    print("")
    print(Itt)
    IsQUBO=input("Is it quadratized {yes/no}")
    count=count+1
    if IsQUBO=="yes":
        break
QUBO=Itt+eval(pnty)
print(QUBO)

x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21,y22,y23,y24,y25,y26 = Binary("x1"),Binary("x2"),Binary("x3"),Binary("x4"),Binary("x5"),Binary("x6"),Binary("x7"),Binary("x8"),Binary("x9"),Binary("x10"),Binary("k1"),Binary("k2"),Binary("k3"),Binary("k4"),Binary("k5"),Binary("k6"),Binary("k7"),Binary("k8"),Binary("k9"),Binary("k10"),Binary("y1"),Binary("y2"),Binary("y3"),Binary("y4"),Binary("y5"),Binary("y6"),Binary("y7"),Binary("y8"),Binary("y9"),Binary("y10"),Binary("y11"),Binary("y12"),Binary("y13"),Binary("y14"),Binary("y15"),Binary("y16"),Binary("y17"),Binary("y18"),Binary("y19"),Binary("y20"),Binary("y21"),Binary("y22"),Binary("y23"),Binary("y24"),Binary("y25"),Binary("y26")
G =eval(str(QUBO))
model=G.compile()
bqm = model.to_bqm(index_label=True)
sampler=LeapHybridSampler(token="Replace with API Token")
sampleset = sampler.sample(bqm,time_limit=3)
decoded_samples = model.decode_sampleset(sampleset)
best_sample = min(decoded_samples, key=lambda x: x.energy)
print(best_sample.sample)
```

Appendix B

Python code to use Advantage system 4.1

```python
#Replace embedding to annealer cord with this cord to use Advantage_system4.1
import dimod
import neal
from dwave.system.samplers import DWaveSampler
from dwave.system.composites import FixedEmbeddingComposite
from minorminer.busclique import find_clique_embedding
import dwave_networkx as dnx
from pyqubo import Binary, Constraint, Placeholder,Array, LogEncInteger

x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15,y16,y17,y18,y19,y20 = Binary("x1"),Binary("x2"),Binary("x3"),Binary("x4"),Binary("x5"),Binary("x6"),Binary("x7"),Binary("x8"),Binary("x9"),Binary("x10"),Binary("k1"),Binary("k2"),Binary("k3"),Binary("k4"),Binary("k5"),Binary("k6"),Binary("x7"),Binary("x8"),Binary("k9"),Binary("k10"),Binary("y1"),Binary("y2"),Binary("y3"),Binary("y4"),Binary("y5"),Binary("y6"),Binary("y7"),Binary("y8"),Binary("y9"),Binary("y10"),Binary("y11"),Binary("y12"),Binary("y13"),Binary("y14"),Binary("y15"),Binary("y16"),Binary("y17"),Binary("y18"),Binary("y19"),Binary("y20"),
G =eval(str(QUBO))
model=G.compile()
dw_sampler = DWaveSampler(endpoint="https://cloud.dwavesys.com/sapi",token="Replace with API Token",solver="Advantage_system4.1")
sampler_kwargs = {"num_reads": 100,"annealing_time": 20,"num_spin_reversal_transforms": 4,"auto_scale":True ,"chain_strength": 2.0,"chain_break_fraction": True }
graph_size=16
```

```
sampler_size=len(model.variables)
p16_working_graph = dnx.pegasus_graph(graph_size,node_list=
    dw_sampler.nodelist,edge_list=dw_sampler.edgelist)
embedding = find_clique_embedding(sampler_size,p16_working_graph)
sampler = FixedEmbeddingComposite(dw_sampler, embedding)
bqm = model.to_bqm(index_label=True)
sampleset = sampler.sample(bqm,**sampler_kwargs)
decoded_samples = model.decode_sampleset(sampleset)
best_sample = min(decoded_samples, key=lambda x: x.energy)
print(best_sample.sample)
```