Finding the Minimal Generating Set for a Finitely Generated Abelian Group on a Quantum Computer

August 6, 2004

Abstract

In this paper we use basic concepts of Quantum Computing to try to solve the problem of finding generators of a finitely generated abelian group. We present three algorithms based on random selection of group elements to find a minimal generating set for a finite abelian group. We also describe an algorithm for finding a minimal generating set for a torsion free abelian group.

Introduction

Quantum computers have been shown to be at least as fast and useful as classical ones. In this paper we attempt to explore the problem of finding a minimal generating set for a finitely generated abelian group using some of the advantages that quantum computers provide.

1 Introduction to Quantum Computing and Our Problem

1.1 Our problem

In this paper we outline several approaches to finding generators for finite and infinite abelian groups on a quantum computer. Given any finitely generated abelian group we attempt to find a set of elements that generates the whole group and then reduce it to a minimal generating set.

There are basic assumptions common to both the finite and infinite cases. In the finite case we assume we are given the group elements which we represent as individual quantum states whereas in the infinite case we are given a large and varied enough subset of the group elements to guarantee a high probability that a generating set of this subset will also generate the whole group. More specific details about the groups are also assumed and are explained in their respective sections.

1.2 Basics of Quantum Computing

Quantum computers are the new frontier in the computing industry because in theory they allow you to do everything that a classical computer can and often faster. There
have already been examples where quantum computers are able to perform computations and solve theoretical problems significantly faster than they could be done on classical computers.

Information in quantum computers is stored in strings of n-qubits, the quantum equivalent of n-bit strings of zeroes and ones in classical computation. Like classical bits, qubits can be in states |0⟩ or |1⟩, but unlike the classical case they can also be in a superposition of these two states, represented as

$$|x⟩ = a |0⟩ + b |1⟩,$$

where a and b are complex numbers. (This notation, called Dirac notation, is the standard used in quantum mechanics.) You can think of a qubit as a vector over \( \mathbb{C}^2 \), where

$$|0⟩ = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad |1⟩ = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

form an orthonormal basis for the vector space. Although this allows for many more qubit states than classical bit states, we cannot actually find out the values of a and b, so we can never know all of the information about a quantum state. Instead, measuring a qubit gives us a 0 with probability \(|a|^2\) and a 1 with probability \(|b|^2\), where \(|a|^2 + |b|^2 = 1\). A consequence of this is that you can get true randomness by creating an entangled qubit state and then measuring it. Whereas n-bit systems in classical computers are just simple strings, in a quantum computer an n-qubit is a vector in the space \( \mathbb{C}^{2^n} \).

To further introduce Dirac notation, inner products of the vectors \(|v⟩\) and \(|w⟩\) are written \( ⟨v|w⟩ \) and outer products \(|v⟩⟨w|\), where \((⟨v|w⟩)|x⟩ = ⟨w|x⟩|v⟩\).

The ability for qubits to be in any of an infinite number of superpositions of these pure states rather than only zero or one leads to the consequence that quantum computers can perform operations at least as quickly as classical computers by simply not introducing entangled states and often in fact much quicker by performing simultaneous operations on each part of an entangled state. Just as classical bits are manipulated through logic gates, quantum states are manipulated using unitary operators - matrices \( U \) for which \( U^\dagger U = I \), where \( U^\dagger \) is the adjoint of \( U \), defined as the unique matrix that satisfies \( ⟨v|U^\dagger U|w⟩ = ⟨U^\dagger v|w⟩ \). Equivalently, a matrix is unitary if

$$⟨Uv|Uw⟩ = ⟨v|w⟩$$

### 2 Finite Abelian Group

We present three algorithms to solve the finite case of our problem. We first state our problem for the finite case as well as all the assumptions we will be using. The problem is to:

Find a minimal generating set for a finite Abelian Group \( G \).

However in addition we also assume the following: We assume the existence of an operator
such that \( M|j\rangle|k\rangle = |j + k\rangle|k\rangle \). We not only know how to add elements in \( G \) through the operator \( M \) but we also know a decomposition of \( G \) into cyclic groups such as

\[
G \cong \mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \cdots \oplus \mathbb{Z}_{m_n}
\]

and possess a function, \( \phi \) which given an element of \( G \) can return to us the decomposition of that element into components according to the group structure. In other words \( \phi \) is the natural mapping corresponding to the isomorphism in equation (1). Notice that this implies that we also know the prime factorization of the order of \( G \).

Two of the following algorithms will first compute a minimal generating set for each \( p \)-subgroup of \( G \), which is information that is interesting in its own right, and then proceed to use these generating sets to easily construct a minimal generating set for \( G \) as follows.

**Proposition 2.1** Let \( G \) be a finite abelian group such that

\[
G \cong A_{p_1} \oplus A_{p_2} \oplus \cdots \oplus A_{p_k}
\]

where each \( A_{p_i} \) is a \( p \)-subgroup of \( G \) for distinct primes \( p_i \). Assume also that we are given a minimal generating set \( X_i \) for each \( A_{p_i} \). Then there exists a procedure to construct a minimal generating set for \( G \) requiring time linear in \( \max(|X_1|, |X_2|, \ldots, |X_k|) \).

**Proof** Let \( X_{ij} \) denote the \( j \)th element of \( X_i \). It is a standard textbook result that if given two elements \( a \) and \( b \) of an abelian group, such that \( (|a|, |b|) = 1 \) then \( \langle a \rangle \oplus \langle b \rangle \cong \langle a + b \rangle \).

Using induction we can then show that \( \{X_{i1}, X_{i2}, \ldots, X_{it}\} \) generates the same subgroup as does the single element \( \sum_{t=1}^{t} X_{i1} \) since the orders of the elements are powers of distinct primes. Similarly, we can add the second elements of each \( X_i \) (when they exist) together, and so on. In fact, it is clear that the number of generators required to generate \( G \) using this method is the same as for the \( p \)-subgroup of \( G \) that requires the greatest number of generators. This is the number that we would expect from group theory.

The first algorithm we present that relies on Simon’s subgroup problem and provides added information about the generators of each \( p \)-subgroup. The second algorithm is based entirely on random selection, only provides a minimal generating set, and is the fastest of the three. The third and last algorithm gives as much information as the first, but requires many applications of the addition operator \( M \). The first algorithm employs the quantum Fourier Transform via Simon’s Hidden Subgroup Problem, which can only be approximated with infinitesimal precision, whereas the second and third do not.

### 2.1 Using Simon’s Hidden Subgroup Problem

One strategy in finding the generators of a finite abelian group is to employ the algorithm used to solve Simon’s Hidden Subgroup problem. The extended algorithm to abelian groups for Simon’s Hidden Subgroup problem solves the problem:

*Given an abelian group \( G \) which is isomorphic to \( \mathbb{Z}_{m_1} \oplus \mathbb{Z}_{m_2} \oplus \cdots \oplus \mathbb{Z}_{m_n} \), and a function \( f : G \to G \) with \( H_0 \subseteq G \) such that \( f \) is constant on \( H_0 \) and distinct on its cosets, find the generators for \( H_0 \).*

This problem is solved in part using the following two classically achievable algorithms for abelian groups, which we will also employ:
Proposition 2.2 There exists a classically deterministic algorithm that, given a subset \( X \subseteq G \), returns a linearly independent subset of \( G \) that generates the subgroup \( \langle X \rangle \), provided that \( G \) is abelian and we are given its decomposition into cyclic groups. The algorithm runs in time polynomial in \( n \), where \( n \) is the number of cyclic groups in \( G \)’s decomposition, and linear in \( |X| \).

Proposition 2.3 There exists a classical deterministic algorithm that, given a linearly independent subset \( X \subseteq G \), returns a linearly independent subset of \( G \) that generates the orthogonal subgroup of \( \langle X \rangle \), once again provided that \( G \) is abelian and its decomposition into cyclic groups is given. The algorithm runs in time polynomial in \( n \), the number of cyclic groups in \( G \)’s decomposition.

The algorithm proceeds according to the following steps, which we more thoroughly explain afterwards.

Step 1 Construct a function which is constant on a given \( p \)-subgroup, \( A_p \), of \( G \) and distinct on its cosets.

Step 2 Apply the methods in Simon’s Algorithm to achieve a superimposed state of the elements in \( A_p^\perp \).

Step 3 Measure the superimposed state enough times to guarantee with probability infinitesimally close to 1 that we have a generating set for \( A_p^\perp \).

Step 4 Apply the algorithms described in Propositions 2.2 and 2.3 to achieve a generating set for the \( p \)-subgroup \( A_p \).

Step 5 Repeat the above process for all \( p \)-subgroups to achieve the generating sets for each.

Then use the procedure described in Proposition 2.1 to achieve a minimal generating set for \( G \).

Let \( A_p \) be a \( p \)-subgroup of \( G \). The construction of the function is simple using our given operator for addition within the group. Assuming the existence of an operator \( M \) such that \( M|j k\rangle = |j + k k\rangle \) we will apply \( M|A_p\rangle \) to the entangled state

\[
|\psi\rangle = \sum_{g \in G} |e\rangle|g\rangle
\]

to replace the first qubit by \(|A_p|\) times the second. Clearly the function \( f(x) = |A_p|x \) is constant on elements of \( A_p \) as it sends them to the identity. It is also distinct on all the cosets of \( A_p \) as it is simply a group homomorphism with \( A_p \) as its kernel.

Applying the procedures of Simon’s Algorithm to achieve a superimposed state of \( H_0^\perp \) is a technique one can look up in any paper on Simon’s Algorithm. For example see [1].

Once we have the state \(|H_0^\perp\rangle\) we may begin to make measurements to randomly pick elements of \( A_p^\perp \). Assume we have picked a set of elements, \( X \), such that \( \langle X \rangle \neq A_p^\perp \). Since \(|\langle X\rangle|/|A_p^\perp| \leq \frac{1}{2} \) if we pick another element we will increase \( \langle X \rangle \) with probability at least
Therefore we must pick an expected number of at most 2 elements in order to increase \( \langle X \rangle \) and therefore make at most \( \log |A_p^\perp| \) number of measurements. To check that we have in fact found a generating set for \( A_p^\perp \) we keep track of the number of elements in our generating set at every stage. After \( \log |A_p^\perp| \) number of picks we run the algorithm described by Proposition 2.2. Then we continue picking elements and adding them to our generating set, and at every step reducing the generating set using Proposition 2.2 as well as keeping track of the size of the generating set. We repeat this step until the size doesn’t change either 100 or \( 2 \log |A_p^\perp| \) times in a row, whichever is larger. For at most \( \log |A_p^\perp| \) number of the steps we might conceivably get the same sized set back which would still generate a larger subgroup at every step, but if after the other \( \log |A_p^\perp| \) steps we still retain the same sized generating set either it generates \( A_p^\perp \) or we must have picked an element in the generated subgroup \( \max(\log |A_p^\perp|, 50) \) number of times, which has a probability of at most \( \frac{1}{2}^{\max(\log |A_p^\perp|, 50)} \). So with running time polynomial in \( \log |A_p^\perp| \) we can find a minimal generating set for \( A_p^\perp \) with probability at least \( 1 - \frac{1}{2}^{50} \). We then simply apply the algorithm described in Proposition 2.3 to get a generating set for our \( p \)-subgroup \( A_p \).

As outlined in step 5 we repeat the procedure for every prime \( p \) dividing the order of our group \( G \) to get all the generators for every \( p \) group.

### 2.1.1 Running Time

During the course of the algorithm we must apply the \( M \) operator a total of \( \sum |A_p| \) number of times, for all primes \( p \) dividing \( |G| \). After picking an element we ran the algorithm described by Proposition 2.2, which runs in polynomial time with respect to the order of the inputed set, which will be of size \( O(\log |G|) \). Since we will picking elements on order of \( \log |G| \) number of times, the total number of steps remains polynomial in \( \log |G| \).

### 2.2 Unstructured Random Algorithm

This algorithm is based on selecting elements randomly from \( G \). Eventually the set \( X \) of elements that have been selected so far will be large enough to generate all of \( G \). Each iteration of the algorithm is as follows:

**Step 1** Begin with a uniform superposition of all elements in \( G \), select one randomly by measurement, and add it to the set \( X \). Alternatively, select \( n \) random elements at a time and add them all to \( X \). This could improve the running time of the algorithm, but by at most the constant factor \( n \), so we will just consider the case where \( n = 1 \).

**Step 2** Apply the algorithm described by Proposition 2.2 to \( X \). Use the resulting linearly independent set in place of \( X \) from now on.

**Step 3** Apply the algorithm described by Proposition 2.3 to \( X \). If the orthogonal subgroup of \( \langle X \rangle \) is just \( \{0\} \), then \( X \) is a linearly independent subset of \( G \) that generates \( G \), hence we are finished. Otherwise, go back to Step 1.
2.2.1 Running Time

At Step 1, we wish to select an element of $G$ that is not already in the subgroup $\langle X \rangle$. As long as $\langle X \rangle$ is a proper subgroup of $G$, the probability that this will happen is at least $1/2$ by Lagrange’s Theorem. If it does happen, $\langle X \rangle$ will be enlarged by a factor of at least 2. The number of times that $\langle X \rangle$ needs to double in size is $\log |G|$, hence the expected number of times that Step 1 needs to be repeated is $O(\log |G|)$. Since classical algorithms 1 and 2 are polynomial in time $\log |G|$, the overall algorithm is polynomial in $\log |G|$. Note the algorithm does not require any applications of the operator $M$.

2.3 Semi-Structured Random Algorithm

Unlike the algorithm based on the Hidden Subgroup problem, the algorithm above gives no information about the $p$-subgroup structure of $G$. If we wish to have this information, there is still an alternative which does not use Fourier transforms. For every prime $p$ that divides $|G|$, we will find a minimal generating set for the (unique) $p$-subgroup of $G$. Let the prime factorization of $|G|$ be

$$|G| = p_1^{\alpha_1}p_2^{\alpha_2} \cdots p_t^{\alpha_t}.$$  

Then, by the Fundamental Theorem of Finite Abelian Groups,

$$G \cong A_{p_1} \oplus A_{p_2} \oplus \cdots \oplus A_{p_t},$$

where $A_{p_i}$ is the $p$-subgroup corresponding to $p_i$.

The following steps will be repeated for each $i \in \{1, 2, \ldots, t\}$. For the sake of clarity, however, we shall drop the subscript $i$ in $p_i$ and $A_{p_i}$. As before, let $X$ be the set of elements that have been obtained so far.

Step 1:

Begin in the superposition state

$$|\psi\rangle = \sum_{g \in G} |e\rangle|g\rangle,$$

where $e$ is the group identity. Now, apply the operator $M^{|G|}$. The elements in the first register are now uniformly distributed among $P$, so we measure the first register to obtain a random element of $P$, and add it to our set $X$.

Step 2:

Apply the classical algorithm described by Proposition 2.2.

Step 3:

Again, we need to test whether or not $\langle X \rangle = P$. The algorithm described by Proposition 2.3 is again what we will use, but this time we wish to find the perpendicular subgroup to $\langle X \rangle$ with respect to $A_p$. In order to be able to use this algorithm with respect to some abelian group $H$, the following conditions must be met:

(i) we must have a breakdown of $H$ into a direct sum of cyclic groups, and

(ii) for any $h \in H$, we must be able to find the coordinates of $h$ according to this breakdown.
We have already stated at the beginning of the section that both \((i)\) and \((ii)\) hold for \(H = G\).

Take the breakdown of \(G\) as described by equation (1). For each \(j \in \{1, 2, \ldots, n\}\), factor \(m_j\) as \(m_j = b_j p^{\beta_j}\), where \(b_j\) is relatively prime to \(p\). We assume that we have the ability to factor, so that \(b_j\) and \(\beta_j\) are known. It is a textbook group theory result that

\[ A_p \cong \mathbb{Z}_{p^{\beta_1}} \oplus \cdots \oplus \mathbb{Z}_{p^{\beta_n}}, \]

satisfying \((i)\). Next, suppose that we are given \(q \in A_p\). We can compute \(\phi(q) = (q_1, q_2, \ldots, q_n)\). Each \(\mathbb{Z}_{m_j} \cong \mathbb{Z}_{b_j} \oplus \mathbb{Z}_{p^{\beta_j}}\), and a mapping between them is the natural isomorphism of taking \(q_j \mod b_j\) and \(p^{\beta_j}\). This will map the generator 1 to the generator \((1, 1)\). In our case however since \(q \in A_p\) we know that the first component is simply 0. This is the isomorphism which satisfies \((ii)\).

In summary, we can again use the algorithm described by Proposition 2.3 to test whether \(\langle X \rangle = A_p\). If so, we have a minimal generating set for \(A_p\), so we move to the next prime (that is, increment the counter \(i\) defined above). Otherwise, go back to step 1.

### 2.3.1 Running time

This algorithm is essentially the previous one performed on each \(P\)-subgroup of \(G\). The time required for running the algorithms described by Propositions 2.2 and 2.3 on \(A_{p_i}\) is polynomial in \(\log(|A_{p_i}|)\), and hence \(O(\log(|P_i|)^n)\) for some \(n \in \mathbb{N}\). Therefore the total time required is

\[ O\left(\sum_{i=1}^{t} (\log(|P_i|))^n\right), \]

whereas for the Random Algorithm above the time required was

\[ O(\log(|G|)^n) = O\left(\left(\sum_{i=1}^{t} \log(|P_i|)\right)^n\right). \]

As long as \(n > 1\) and \(t > 1\), this represents a savings in time, according to the inequality

\[ \sum_{i=1}^{t} a_i^n < \left(\sum_{i=1}^{t} a_i\right)^n \]

where the \(a_i\) are any positive numbers.

The other main difference in running time is that the operator \(M_{[G]} / [G]\) is required in step one. This operator needs to be applied at most as many times as there are measurements, which was calculated in the above paragraph, times \(|G|\) since depending on the prime the function we use is \(M_{[G]} / [A_p]\).

### 3 Torsion Free Infinite Abelian Group

Before presenting our algorithm for finding the generators of a Torsion Free Infinite Abelian Group we first prove some results in probability which are then useful in the discussion of the algorithm.
3.1 Preliminaries

Given an event $A$, the probability that that event occurs we write to be $P(A)$. Also we write $(a_1, a_2, \ldots, a_n)$ to denote the greatest common divisor of $a_1, a_2, \ldots, a_n$.

**Theorem 3.1** For $a, b \in \mathbb{Z}$ we have $P((a, b) = 1) = 6\pi^2 = \zeta(2)$. In general for $a_1, \ldots, a_n \in \mathbb{Z}$ we have $P((a_1, \ldots, a_n) = 1) = \zeta(n)^{-1}$.

**Proof** Consider the probability that the numbers $a_1, a_2, \ldots, a_n$ are divisible by a prime $p$. This is clearly just $\frac{1}{p^n}$ and so the probability that at least one of these numbers is not divisible by $p$ is simply $1 - \frac{1}{p^n}$. If $p$ and $q$ are different primes, not being divisible by $q$ and not being divisible by $p$ are independent events, and therefore not being divisible by both $p$ and $q$ is just the product of the individual probabilities. So then we may conclude that $P((a_1, \ldots, a_n) = 1) = \prod_{p \text{prime}} (1 - \frac{1}{p^n})$.

Dropping the subscript 'p prime' for convenience we then have $P((a_1, \ldots, a_n) = 1)^{-1} = \prod (1 - \frac{1}{p^n})^{-1} = \prod \frac{1}{1 - \frac{1}{p^n}} = \prod \frac{1}{1 - \frac{1}{p^n}} = \prod \sum_{i=0}^{\infty} (\frac{1}{p^n})^i = \sum_{k=1}^{\infty} \frac{1}{k\pi} = \zeta(n)$. So then $P((a_1, \ldots, a_n) = 1) = \zeta(n)^{-1}$.

We may think of picking numbers as picking $1 \times 1$ matrices and asking for the probability that their determinants are relatively prime. In the following theorems we will be discussing 'randomly' picked matrices from $M_n(\mathbb{Z})$. What we mean by this is that for any entry $a$ of this matrix, any modulus $m$, and any integer $k$ with $0 \leq k < m$, the probability that that $a$ is congruent to $k \bmod m$ is always $1/m$. We may then consider the probability of picking random matrices as the limit that this modulus goes to infinity. We also extend the above result to $n \times n$ matrices, but first we need a lemma.

**Lemma 3.2** Given $A \in M_n(\mathbb{Z})$ the probability that $|A| \neq 0 \bmod p$ for $p$ a prime is $\prod_{i=1}^{n} (1 - \frac{1}{p^n})$.

**Proof** We proceed by counting possibilities. For the first row of our matrix we have $p^n - 1$ possibilities where we exclude only the 0 vector. We will show that the number of possibilities for the $i^{th}$ row where $i \geq 1$ of our matrix given that we have already chosen all the prior rows is $p^n - p^i$.

Since we will be calculating the number of possibilities for the $i^{th}$ row given we have chosen all the rows from 0 to $i - 1$ we will denote $c_j$ to be the $j^{th}$ row where $0 \leq j < i$. Now it follows that the $i^{th}$ row can not be any of the vectors of the form $\sum_{k=0}^{i-1} a_k c_k$ where each of the $a_k \in \mathbb{Z}_p$. Since $|A| \neq 0 \bmod p$ we have that all the $c_k$ are linearly independent. Since $p$ is a prime all the linear combinations of $c_k$ are distinct and therefore there are exactly $p^i$ vectors in $\{\sum_{k=0}^{i-1} a_k c_k | a_k \in \mathbb{Z}_p\}$. It follows that we have $p^n - p^i$ choices for the $i^{th}$ row and therefore the total number of possible matrices such that $|A| \neq 0 \bmod p$ is $\prod_{i=0}^{n-1} (p^n - p^i)$.

Since the total number of choices we had from the beginning is simply $p^{n^2}$ the probability that $|A| \neq 0 \bmod p$ is

$$\prod_{i=0}^{n-1} (p^n - p^i) = \prod_{i=0}^{n-1} (p^{n-i} - 1) = \prod_{i=1}^{n} (1 - \frac{1}{p^i})$$

8
With the above proof we can now calculate the probability that a determinant of an \( n \times n \) matrix is divisible by \( p \), namely
\[
D_n(p) = 1 - \prod_{i=1}^{n}(1 - \frac{1}{p^i}).
\]
With that done we prove the equivalent of Theorem 3.1 for \( n \times n \) determinants.

**Theorem 3.3** Given \( k \) matrices \( A_1, A_2, \ldots, A_k \in M_n(\mathbb{Z}) \) we have that
\[
P((|A_1|, |A_2|, \ldots, |A_k|) = 1) = \prod_{p \text{ prime}} (1 - D_n(p)^k)
\]

**Proof** In Theorem 3.1 we notice that it’s pretty clear that the events of being divisible by \( p \) and being divisible by \( q \), for coprime \( p \) and \( q \), are independent. Likewise one would guess that \(|A|\) being divisible by \( p \) does not depend on whether or not it is divisible by \( q \). We use the definition of conditional probability to state this formally.

**Lemma 3.4** For a matrix \( A \), if \( x \) and \( y \) are any pair of relatively prime integers, then
\[
P((x, |A|) = 1) = \frac{P((x, |A|) = 1 \cap (y, |A|) = 1)}{P((y, |A|) = 1)}
\]
Or, equivalently,
\[
P((|A|, xy) = 1) = P((x, |A|) = 1)P((y, |A|) = 1).
\]

**Proof** For any \( m \in \mathbb{Z} \), let \( M_n(\mathbb{Z}_m) \) denote the ring of all \( n \times n \) matrices with entries in \( \mathbb{Z}_m \), and let \( M_n(\mathbb{Z}_m)^* \) denote the units in this ring; that is, the matrices whose determinant is a unit in \( \mathbb{Z}_m \). According to this terminology, we wish to find a one-to-one correspondence between \( M_n(\mathbb{Z}_{xy})^* \) and \( M_n(\mathbb{Z}_x)^* \times M_n(\mathbb{Z}_y)^* \).

Let \( \phi : \mathbb{Z}_{xy} \rightarrow \mathbb{Z}_x \times \mathbb{Z}_y \) be defined by \( \phi(k) = (k \mod x, k \mod y) \). \( \phi \) is one-to-one and onto by the Chinese Remainder Theorem. In fact, it is not hard to see that \( \phi \) is a ring homomorphism, hence an isomorphism. For a matrix \( A \in M_n(\mathbb{Z}_{xy}) \), let \( \phi(A) \) denote the matrix obtained by applying \( \phi \) to each entry of \( A \). Notice that that \(|\phi(A)| = \phi(|A|) \) where on the left we’re taking the determinant of each resulting matrix. This makes \( \phi \) into an isomorphism from \( M_n(\mathbb{Z}_{xy})^* \) to \( M_n(\mathbb{Z}_x)^* \times M_n(\mathbb{Z}_y)^* \). In particular, this implies:
\[
|M_n(\mathbb{Z}_{xy})^*| = |M_n(\mathbb{Z}_x)^*| \times |M_n(\mathbb{Z}_y)^*|.
\]
The stated result about probabilities follows directly.

\( D_n(p) \) is the probability that a determinant is divisible by \( p \) and \( 1 - D_n(p)^k \) is the probability that \( k \) determinants don’t all share \( p \) as a factor. Using the result of Lemma 3.4 the probability that these \( k \) determinants don’t all share any prime factor, or in other words, that they’re relatively prime, is simply
\[
\prod_{p \text{ prime}} (1 - D_n(p)^k)
\]
Table 1: Probabilities

<table>
<thead>
<tr>
<th></th>
<th>n=1</th>
<th>n=2</th>
<th>n=3</th>
<th>n=4</th>
<th>n=5</th>
<th>n=6</th>
<th>n=7</th>
<th>n=8</th>
<th>n=9</th>
<th>n=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>k=2</td>
<td>0.609</td>
<td>0.453</td>
<td>0.397</td>
<td>0.374</td>
<td>0.364</td>
<td>0.359</td>
<td>0.356</td>
<td>0.355</td>
<td>0.354</td>
<td>0.354</td>
</tr>
<tr>
<td>k=3</td>
<td>0.832</td>
<td>0.692</td>
<td>0.629</td>
<td>0.6</td>
<td>0.587</td>
<td>0.577</td>
<td>0.575</td>
<td>0.575</td>
<td>0.574</td>
<td></td>
</tr>
<tr>
<td>k=4</td>
<td>0.924</td>
<td>0.821</td>
<td>0.766</td>
<td>0.739</td>
<td>0.726</td>
<td>0.716</td>
<td>0.715</td>
<td>0.714</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k=5</td>
<td>0.964</td>
<td>0.894</td>
<td>0.85</td>
<td>0.827</td>
<td>0.815</td>
<td>0.81</td>
<td>0.807</td>
<td>0.805</td>
<td>0.804</td>
<td></td>
</tr>
<tr>
<td>k=6</td>
<td>0.983</td>
<td>0.936</td>
<td>0.902</td>
<td>0.883</td>
<td>0.874</td>
<td>0.869</td>
<td>0.867</td>
<td>0.865</td>
<td>0.864</td>
<td></td>
</tr>
<tr>
<td>k=7</td>
<td>0.992</td>
<td>0.961</td>
<td>0.936</td>
<td>0.921</td>
<td>0.913</td>
<td>0.909</td>
<td>0.907</td>
<td>0.906</td>
<td>0.905</td>
<td></td>
</tr>
<tr>
<td>k=8</td>
<td>0.996</td>
<td>0.976</td>
<td>0.957</td>
<td>0.946</td>
<td>0.94</td>
<td>0.937</td>
<td>0.935</td>
<td>0.934</td>
<td>0.934</td>
<td>0.933</td>
</tr>
<tr>
<td>k=9</td>
<td>0.998</td>
<td>0.985</td>
<td>0.972</td>
<td>0.963</td>
<td>0.958</td>
<td>0.955</td>
<td>0.954</td>
<td>0.954</td>
<td>0.953</td>
<td>0.953</td>
</tr>
<tr>
<td>k=10</td>
<td>0.999</td>
<td>0.991</td>
<td>0.981</td>
<td>0.974</td>
<td>0.971</td>
<td>0.969</td>
<td>0.968</td>
<td>0.967</td>
<td>0.967</td>
<td>0.967</td>
</tr>
<tr>
<td>k=11</td>
<td>1.</td>
<td>0.994</td>
<td>0.987</td>
<td>0.982</td>
<td>0.979</td>
<td>0.978</td>
<td>0.977</td>
<td>0.977</td>
<td>0.977</td>
<td>0.976</td>
</tr>
<tr>
<td>k=12</td>
<td>1.</td>
<td>0.996</td>
<td>0.991</td>
<td>0.988</td>
<td>0.986</td>
<td>0.984</td>
<td>0.984</td>
<td>0.983</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k=13</td>
<td>1.</td>
<td>0.998</td>
<td>0.994</td>
<td>0.992</td>
<td>0.99</td>
<td>0.989</td>
<td>0.989</td>
<td>0.988</td>
<td>0.988</td>
<td></td>
</tr>
<tr>
<td>k=14</td>
<td>1.</td>
<td>0.999</td>
<td>0.996</td>
<td>0.994</td>
<td>0.993</td>
<td>0.992</td>
<td>0.992</td>
<td>0.992</td>
<td>0.992</td>
<td></td>
</tr>
<tr>
<td>k=15</td>
<td>1.</td>
<td>0.999</td>
<td>0.997</td>
<td>0.996</td>
<td>0.995</td>
<td>0.995</td>
<td>0.994</td>
<td>0.994</td>
<td>0.994</td>
<td>0.994</td>
</tr>
</tbody>
</table>

Though we were not able to get useful and rigorous estimates for the lower bound on this probability we did calculate some values up to a point where it 'looked' like the product was quickly converging. The following values are evaluations of expression (2) up to 30 distinct primes. The table is arranged by number of matrices picked determined by the row, and number of dimensions determined by the column. The calculations were done with Mathematica, and go up to 15 matrices picked, and 30 dimensions, though we’re only showing it up to 10 dimensions. It’s fairly clear how even at a fairly low dimension and small number of matrices picked, the probabilities are already very close to 1. Notice that the first column, since it’s simply $1 \times 1$ matrices should look like $\zeta(n)^{-1}$.

Now we get to why picking random matrices and knowing the probability that their determinants are relatively prime is important.

**Lemma 3.5** Let $a_1, a_2, \ldots, a_n$ represent the standard basis vectors of $\mathbb{Z}^n$. Given $n$ row vectors in $\mathbb{Z}^n$ whose determinant is $k$ the vectors $ka_1, ka_2, \ldots, ka_n$ are all in the generated subspace.

**Proof** Let $A$ be the matrix composed of the $n$ given vectors represented as columns. A given vector $a$ is a linear combination of our given vectors if and only if there exists a column vector $w$ such that $Aw = a$. Now let $\hat{A}$ denote the adjunct of the matrix $A$, that is the matrix such that $\hat{A}A = kI$ where $I$ is the identity matrix. From basic linear algebra we know that if $A \in M_n(\mathbb{Z})$ that also $\hat{A} \in M_n(\mathbb{Z})$ so we can solve for $w_i$ in the equation $Aw_i = ka_i$ for $i = 1..n$, namely

\[
\begin{align*}
    Aw_i &= ka_i \\
    \hat{A}Aw_i &= k\hat{A}a_i \\
    kw_i &= k\hat{A}a_i \\
    w_i &= \hat{A}a_i
\end{align*}
\]
Theorem 3.6 Assume the same notation as in the previous lemma. If given 2 sets of \( n \) vectors \( \{v_i\} \) and \( \{w_i\} \) such that the constructed matrices have determinants \( k_1 \) and \( k_2 \) respectively, and \( (k_1, k_2) = 1 \), then together the \( 2n \) vectors generate all of \( \mathbb{Z}^n \).

Proof By Lemma 3.5 we know that the \( v_i \)'s and \( w_i \)'s generate \( k_1a_i \) and \( k_2a_i \) for all \( i \)'s. From basic number theory we also know that if \( (k_1, k_2) = 1 \) then there exists a linear combination such that \( xk_1 + yk_2 = 1 \) for \( x, y \in \mathbb{Z} \). Therefore for all \( i \) \( a_i \) lies in the space generated by \( \{v_i\} \) and \( \{w_i\} \) and therefore this space is all of \( \mathbb{Z}^n \).

Corollary 3.7 Proposition 3.6 can be generalized to \( m \) sets of \( n \) vectors as long as the determinants of the matrices determined by each set of vectors are all relatively prime, in other words \( (k_1, k_2, \ldots, k_m) = 1 \).

Proof The generalization follows from the fact that if \( (k_1, k_2, \ldots, k_m) = 1 \) then there exists a linear combination such that \( \sum_{i=1}^{m} x_ik_i = 1 \).

3.2 The Algorithm

First we introduce our assumptions, which are not all that different from the finite case of our problem. Say our abelian group looks like \( \mathbb{Z}^n \). Then we assume that we’re given a function \( \phi \) that can break down an element into its components. Since the group is infinite and therefore choosing an element randomly is impossible we assume that we’re given a large subset of the group that we know generates it, and we are going to find elements of this subset that form a minimal generating set. The algorithm once again relies on randomly choosing elements and is explained through the following proposition and theorem.

We randomly pick a set of \( m \) elements in \( \mathbb{Z}^n \), with \( m > n \), and we wish to reduce them to a set of only \( n \) vectors which generate the same subgroup. \( m \) will always be a multiple of \( n \) using our algorithm, but for the purposes of this section we can ignore this fact. Let \( A \) be the \( n \times m \) matrix whose columns are formed by the \( m \) vectors we have picked so far (in any order). For the remainder of this section, we will use the word “matrix” to mean “matrix with integer entries.” Our strategy will be to transform \( A \) to a matrix \( A' \) that is in upper-right triangular form, and whose columns generate the same subgroup as the columns of \( A \). Once this is accomplished, only the rightmost \( n \) columns may be nonzero, so these columns will become our minimal generating set (Note that th.

Proposition 3.8 If \( A \) is an \( n \times m \) matrix, and \( B \) is an \( m \times m \) matrix whose determinant is \( \pm 1 \), then the columns of \( AB \) generate the same subgroup of \( \mathbb{Z}^n \) as the columns of \( A \).

Proof:

The subgroup of \( \mathbb{Z}^n \) generated by the columns of any \( n \times m \) matrix \( X \) is the set of all \( \mathbb{Z} \)-linear combinations of the columns, which is just the set \( \{Xz \mid z \in \mathbb{Z}^m\} \). It is clear that \( \{ABz, z \in \mathbb{Z}\} \subseteq \{Ay, y \in \mathbb{Z}\} \), just take \( y = Bz \). However, the reverse inclusion is also true, because \( B^{-1} \) exists and has integer entries, so we can take \( z = B^{-1}y \).

We will make use of the following proposition, which allows us to insert zeroes into \( A \).
Theorem 3.9 If $A$ is an $n \times m$ matrix and $i, j$ are integers with $0 \leq i < n$ and $0 \leq j < m - 1$, then there exists an $m \times m$ matrix $B$ such that:

(i) $|B| = 1$.
(ii) $(AB)_{ij} = 0$.
(iii) Multiplication on the right by $B$ does not alter any of the columns of $A$ besides the $j^{th}$ and $(j + 1)^{st}$ columns.
(iv) If the entries $A_{ij}$ and $A_{i'(j+1)}$ are both zero for any $i'$ with $0 \leq i' < n$, then $AB$ also has zeros there.

Proof:

Let $A_{ij} = a$ and $A_{i(j+1)} = b$. If $a = 0$ then just let $B = I$. Otherwise, let $g = \gcd(a, b)$ (hence we know that $g \neq 0$), and let $u, v \in \mathbb{Z}$ be integers such that $au + bv = g$. (Note; $u$ and $v$ can be determined efficiently using Euclid’s Algorithm).

We define $B$ using block notation in the diagram below, where $I_k$ represents the $k \times k$ identity matrix.

$B = \begin{pmatrix}
I_j & 0 & 0 \\
0 & -b/g & u \\
0 & -a/g & v \\
0 & 0 & I_{m-j-1}
\end{pmatrix}$

It should be noted that $|B| = \frac{bv}{g} + \frac{au}{g} = 1$, and that right multiplication of $A$ by $B$ only changes the $j^{th}$ and $(j + 1)^{st}$ columns of $A$; it leaves the other columns unchanged.

In fact, the matrix product $AB$ will give the following: (where $\cdots$ represents arbitrary integers that we need not concern ourselves with):

$AB = \begin{pmatrix}
\cdots & \cdots & \cdots \\
\cdots & a & b \\
\cdots & \cdots & \cdots \\
0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
I_j & 0 & 0 \\
0 & -b/g & u \\
0 & -a/g & v \\
0 & 0 & I_{m-j-1}
\end{pmatrix} = \begin{pmatrix}
\cdots & \cdots & \cdots \\
0 & g \\
\cdots & \cdots & \cdots \\
0 & 0 & \cdots
\end{pmatrix}$

Therefore the proposition is proved with $B$ defined as above. □

Let $A$ be an $n \times m$ matrix, and let $B_{ij}$ denote the $m \times m$ matrix, defined above, that zeroes out the entry $A_{ij}$. We can put $A$ into upper right triangular form by computing the following matrix product:

$A' = A(B_{(n-1)0}B_{(n-1)1} \cdots B_{(n-1)(m-2)})(B_{(n-2)0} \cdots B_{(n-2)(m-3)}) \cdots (B_{10} \cdots B_{0(m-n-1)})$

That is, we zero out the whole bottom row except for the rightmost entry; then zero out the next row up except for the rightmost two entries; and continue in this manner up to the top row, except for the rightmost $n$ entries. Parts (iii) and (iv) of the above proposition guarantee that none of the zeros created by the previous $B_{ij}$s will be disturbed by the ones later on. Therefore $A'$ is a matrix whose columns generate the same subgroup of $\mathbb{Z}^n$ as the columns of $A$, and only the rightmost $n$ columns of $A'$ can be nonzero. If we identify $A'$ with its rightmost $n$ columns, then we can easily compute its determinant as the product
along its main diagonal, since it is in upper triangular form. If \(|A'| = \pm 1\), then the columns of \(A'\) form a minimal generating set for \(\mathbb{Z}^n\), and we are done. If any other determinant is obtained, we know that the \(m\) elements that we began with do not generate \(\mathbb{Z}^n\), so we must go back and choose some more.

### 3.3 Running Time

Each multiplication by \(B_{ij}\) requires finding the greatest common divisor of two integers, which we can assume is computable in constant time, and also requires four multiplications of columns, which amounts to \(4n\) integer multiplications. The number of \(B_{ij}\)s required less than \(mn\), and recall that \(m = kn\), where \(k\) is a constant. Therefore, the whole elimination algorithm runs in time \(O(n^3)\). If \(k\) is chosen large enough, we know that the probability of generating all of \(\mathbb{Z}^n\) is high (say, greater than \(1/2\)), so we expect to run this elimination routine less than two times. This means that the entire algorithm is \(O(n^3)\) expected running time.

### 4 Ideas for Further Research

Further research could be done on ways to make our algorithms deterministic. As it stands all of the algorithms have good expected running times, yet because they’re all based on random choosing of elements there’s always a chance that a particular case might take a very long time. Another problem one could consider is to construct an efficient algorithm for finding generators for a finitely-generated abelian group, something we did not finish. One algorithm is to first find the generators for the torsion free part of the group using our method of Gaussian elimination, and then to employ the finite group algorithms on the finite group components of elements to get generators for the torsion group. However since the sample set of elements given is so large, in fact orders of magnitude larger than the order of the torsion group, the finite group algorithms wouldn’t be efficient. A valid course after this would then be to design a more efficient algorithm which could combine to solve the more general finitely generated abelian group.
References


