Abstract. In this paper we study the conjugacy problem in polycyclic groups. Our main result is that we create polycyclic groups $G_n$ whose conjugacy problem is at least as hard as the subset sum problem with $n$ indeterminates. As such, the conjugacy problem over the groups $G_n$ is NP-complete where the parameters of the problem are taken in terms of $n$ and the length of the elements given on input.

1. Introduction

Given a finitely presented group $G$, the conjugacy decision problem for $G$ asks if two elements $u, v \in G$ are conjugate. Along with the word problem and isomorphism problem, it was one of the original group theoretic problems introduced by Dehn in 1911. In the context of non-commutative group based cryptography, one often studies the search variant: given two conjugate elements in $G$, find a third element of $G$ conjugating one to the other. In 1999 Anshel, Anshel, and Goldfeld created a key exchange protocol that relied on solving the conjugacy search problem multiple times and proposed braid groups as the platform group. In the years since, different parties have shown that heuristic attacks can in fact break the protocol when it is done over braid groups. Other cryptographic protocols using the conjugacy search problem include.

In 2004 Eick and Kahrobaei proposed polycyclic groups as a secure platform for AAG and offered computational evidence. Later Garber, Kahrobaei, and Lam experimentally showed that polycyclic groups were resistant to many of the heuristic attacks that are strong against braid groups. In this paper we offer theoretical evidence that the conjugacy decision and search problems over polycyclic groups are difficult. We construct polycyclic groups $G_n$ whose conjugacy search and decision problems are at least as hard as the subset sum search and decision problems in $n$ indeterminates which is well known to be NP-complete. The $G_n$ also have the additional property that algebraic computations such as conjugacy and collection can be performed quickly when group elements are represented by exponent vectors. In this way, a polynomial time algorithm that solves either conjugacy problem in these groups would imply P = NP.

We devote sections 2 and 3 to preliminaries on polycyclic groups and the subset sum problem. In section 4 we explicitly construct the groups $G_n$ that we will be working over and introduce a variant of the subset sum problem that we call the twisted subset sum problem ($TSSP$). We further show that the $TSSP$ reduces to the conjugacy decision problem in the $G_n$. In section 5 we show that the conjugacy
problem in the $G_n$ is in NP. We also note that the same computations can also be used to show that multiplication and collection can also be performed in polynomial time. In section 6 we reduce the subset sum problem to the TSSP which implies that the conjugacy problem in the $G_n$ is NP-complete.

2. Poly-$\mathbb{Z}$ Groups

A group $G$ is polycyclic if it has a subnormal series with cyclic quotients. Namely, $G$ has subgroups $G_0, G_1, \cdots, G_n$ such that

\[ \{1\} = G_0 \triangleleft G_1 \triangleleft \cdots \triangleleft G_n = G \]

and $G_{i+1}/G_i$ is a cyclic group.

In this section, we will summarize a variety of results on polycyclic groups that can be found in [3, 4] that will be used in the remainder in this paper.

Given a subnormal series as in (1), one can find a polycyclic generating set, $\{g_1, \cdots, g_n\}$ where $G_i = \langle g_i, G_{i-1} \rangle$ for $1 \leq i \leq n$. With respect to this generating set, each group element $g \in G$ can be represented as $g_{k_1} \cdots g_{k_n}$ and such a representation is called its normal form. We also call each $g_{k_i}$ a syllable. For every polycyclic group, there exists a polycyclic generating set such that any word has a unique normal form. The process of converting an arbitrary word to its normal form is called collection.

Of specific interest to us in this paper are polycyclic groups where $G_i/G_{i-1} \simeq \mathbb{Z}$ for each $i$. Such a group is called poly-$\mathbb{Z}$ as each quotient is isomorphic to $\mathbb{Z}$. If this is the case, then $G$ is obtained from the final non-trivial group in the subnormal series $G_1 \simeq \mathbb{Z}$ by successive semi-direct products with $\mathbb{Z}$ as follows. It is a standard result (see [3] for instance) that if $G/G_{n-1} \simeq \mathbb{Z}$ then $G \simeq G_{n-1} \rtimes \phi \mathbb{Z}$. If we take $g_n$ as the generator of $\mathbb{Z}$, then $\phi$ is given by conjugating elements of $G_{n-1}$ by $g_n$. For reference, $G_{n-1} \rtimes \phi \mathbb{Z}$ is the group with elements $wg_n^k$ with $w \in G_{n-1}$ and multiplication given by:

\[(wg_n^k)(w'g_n^{l}) = w\phi^k(w')g_n^{k+l} \]

Proceeding inductively, we see that $G$ can be written as:

\[ (\cdots (\mathbb{Z} \rtimes \phi_1 \mathbb{Z}) \rtimes \phi_2 \mathbb{Z}) \rtimes \phi_3 \cdots \rtimes \phi_{n-1} \mathbb{Z} \]

where $\phi_j$ is conjugation by $g_{j+1}$.

The groups we will be interested in will be constructed in this fashion by explicitly describing the different $\phi_i$. See [2] in which the authors use the same construction for more details. In the following, we will take $g_i$ as the generator of the $i^{th}$ $\mathbb{Z}$ in the semi-direct product form. From the multiplication rules of the semi-direct product, one can see that $g_jg_i = \phi_{j-1}(g_i)g_j$ for $i < j$. By using this identity, it is possible to then put any arrangement of letters into normal form so that any $g_i$ appears to the left of $g_j$ when $i < j$. We also define the Hirsch length as the number of $\mathbb{Z}$’s in
A FAMILY OF POLYCYCLIC GROUPS OVER WHICH THE CONJUGACY PROBLEM IS NP-COMPLETE

the semi-direct product formulation of the poly-$Z$ group. The Hirsch length is an isomorphism invariant, so while different automorphisms in the construction of the poly-$Z$ group may lead to isomorphic groups, the number of factors is necessarily the same.

Any word $w = g_1^{k_1} \cdots g_n^{k_n}$ can be represented uniquely by its exponent vector, $[k_1, \cdots, k_n]$. We can then take the length of $w$ to be the length of its corresponding exponent vector, $l(w) = \sum_{i=1}^n \log(|k_i|) = O(n \log(K))$ where $K$ is an upper bound of the absolute value of all of the exponents. This measure of length, is somewhat different than many of the standard ones used when studying algorithms in groups. Most often, one would take the length of the word to be its distance from the identity in the Cayley graph equipped with the word metric. In this scenario, we are considering normal forms of group elements, rather than geodesic forms, because they are often used for cryptographic and other algorithmic applications in polycyclic groups. For instance, in a practical setting, one cannot necessarily generate random words that are of the shortest length so it might be better to evaluate computation with respect to the length in normal form. Additionally, it is more practical when possible to work with group elements as a tuple of exponents rather than generator by generator, making the length of the exponent vector a natural measure. In the groups we will be working with, algebraic operations (multiplication, collection, conjugacy) can be computed effectively with a Turing machine (see section 5) when inputs are taken in terms of exponent vectors. This is not necessarily the case in other scenarios where group elements are dealt with generator by generator.

3. Subset Sum Problem

The subset sum problem, or $SSP$, is the following: given a set of integers, $L = \{k_1, k_2, \cdots, k_n\}$, and an integer, $M$, determine if there exists subset of $L$ that sums to $M$. This can also be rephrased as determining if there is a solution to the equation:

$$k_1 x_1 + \cdots + k_n x_n = M \text{ where } x_i \in \{0, 1\}.$$  

We can bound the size of the problem from above by considering the length of the list, $N$, and an upper bound on the absolute value of the entries, $K$. In doing so, the length of the problem can be seen to have length $O(n \log(K))$. We will also label this instance of the $SSP$, $SSP(L, M)$, or just $(L, M)$ when there is no ambiguity.

The $SSP$ is NP-complete, meaning that the existence of a deterministic polynomial-time Turing machine that solves it would imply that $P = NP$. In fact, it was originally introduced by Karp as one of his 21 NP-complete problems. Despite it being NP-complete, there exists a pseudo-polynomial algorithm via dynamic programming (see [13]). Namely there exists a deterministic algorithm that runs in polynomial time when the length of the problem is taken in terms of the actual numerical entries of the list rather than the number of digits needed to represent them. As such, the existence of such an algorithm and the NP-completeness of the problem doesn’t imply that $P = NP$.  


For the purposes of cryptography, we consider the search version of the SSP: given that a subset of $L$ sums to $M$, actually find such a subset. From the outset, it is not immediately clear how the two problems are related, but one can show that a polynomial time algorithm for one would lead to a polynomial time algorithm for the other. First we show how an algorithm for the decision problem can be applied at most $n - 1$ times to make an algorithm for the search problem. This can be done by first checking if the SSP, $(L \setminus \{k_n\}, M)$ has a solution. If not, then we know $k_n$ is a part of our solution and proceed by checking $(L \setminus \{k_n-1, k_n\}, M - k_n)$. Otherwise, we know we can create a solution without $k_n$ and proceed by checking $(L \setminus \{k_n-1, k_n\}, M)$. By doing this repeatedly, we will have eventually found a subset summing to $M$. In the worst case scenario, we will have reached the end of the list and performed the decision algorithm $n - 1$ times. Note that since we are in an instance of the search problem, a solution is assumed to exist, so it is not necessary to run the decision algorithm on $(L, M)$.

On the other hand, if there were a polynomial time algorithm that solved the search version of the SSP, we could use it to prove existence of a polynomial time algorithm for the decision problem. Rather than give a formal proof, we will just sketch one omitting certain details. Given a polynomial time Turing machine, $M$, for the SSP search problem, there exists a polynomial $P(n)$ such that for any input $x$, the number of steps $M$ takes on input $x$ is less than or equal to $P(|x|)$. We can then create another Turing machine, $M'$, that on input $y$ performs the same steps as $M$ for $P(|y|)$ steps. If $M'$ has not yet finished, it then hits its final state and outputs “no” as the answer. $M'$ is then a polynomial time Turing machine for any instance of the SSP decision problem. Either it terminates in less than $P(|y|)$ steps in which case $M'$ has found a solution for $y$ and outputs “yes” or $M'$ takes longer than $M$ would if there were a solution, implying that there isn’t one, and so $M'$ outputs “no”. As such, a polynomial time algorithm for either the SSP or its search variant would imply existence of a polynomial time algorithm for the other.

We also will use the notion of a polynomial time reduction. We say that a decision problem, $Q$, can be reduced to a decision problem, $R$, in polynomial time if there exists a polynomial time mapping, $f$, from instances of $Q$ to instances of $R$, such that an instance $x$ is a “yes” instance of $Q$ if and only if $f(x)$ is a “yes” instance of $R$. Such a mapping also must only increase the lengths of instances at most polynomially. We also write $Q \leq_p R$ to say that $Q$ polynomial time reduces to $R$.

If such an $f$ exists, then a polynomial time algorithm for $R$ would imply a polynomial time algorithm for $Q$. Given an instance $x$ of $Q$, we can compute $f(x)$ and then perform our polynomial time decision algorithm for $R$. As such, a decision problem, $A$, is NP-complete if $B \leq_p A$ for all $B \in$ NP and one can prove a problem $C$ is NP-complete if $A \leq_p C$ where $A$ is NP-complete. Finally, note that polynomial time reductions are transitive: $A \leq_p B$ and $B \leq_p C$ imply $A \leq_p C$. 
4. The conjugacy problem over the groups $G_n$

Given a list $L = \{k_1, \cdots, k_n\}$ and an integer $M$, the twisted subset sum problem ($TSSP$) is determining if the following equation has a solution:

$$k_n x_n (-1)^{x_1 + x_2 + \cdots + x_{n-1} + k_{n-1} x_{n-1} (-1)^{x_1 + x_2 + \cdots + x_{n-2} + \cdots + k_2 x_2 (-1)^{x_1 + k_1 x_1} = M$$

where $x_i \in \{0, 1\}$. Note that we could trivially let $x_i$ be any number and replace $x_i$ with $x_i \mod 2$ in the above equation. For the remainder of the paper, we write $x_i = x_i \mod 2$.

In this section, we show how any instance of the $TSSP$ where the set of integers has length $n$, can be turned into an instance of the conjugacy problem of a polycyclic group with Hirsch length $2n + 1$. Since the reduction is polynomial, we prove that $TSSP$ polynomial time reduces to the conjugacy problem over the $G_n$. Also note that checking certificates of $TSSP$ can be done in polynomial time, hence it is in NP.

The group $G_n$ will be constructed as follows:

$$(\cdots ((\mathbb{Z} \times \phi_1 \mathbb{Z}) \times \phi_2 \mathbb{Z}) \times \phi_3 \cdots) \times \phi_{n-1} \mathbb{Z}$$

where

$$\phi_{2i-1}(g_j) = \begin{cases} g_1^{-1} & \text{if } j = 1 \\ g_j & \text{otherwise} \end{cases}$$

and

$$\phi_{2i}(g_j) = \begin{cases} g_1 g_j & \text{if } j = 2i \\ g_j & \text{otherwise} \end{cases}$$

The multiplicative structure of $G_n$ can then be seen as such: if $j$ is even, $g_j g_1 = g_1^{-1} g_j$ and $g_j g_{j+1} = g_1 g_j g_{j+1}$. The following lemma, is a consequence of these multiplicative identities and will assist us in collecting words throughout the paper.

**Lemma 4.1.** Let $j$ be even and $a, b \in \mathbb{Z}$. Then $g_j^a g_1^b = g_1^{b(-1)^n} g_j^a$ and $g_{j+1}^a g_j^b = g_1^{b'(-1)^{n+1}} g_j^a$.

**Proof.** To prove the first identity, just check that $g_j^a g_1^b = \phi_{j-1}^1(g_1) g_j^a = (\phi_{j-1}^1(g_1)) g_j^a = (g_1^{-1})^b g_j^a = g_1^{b(-1)^n} g_j^a$. For the second identity, we first check the case $a = 1$. We then have $g_{j+1}^a g_j^b = (\phi_j(g_j))^b g_{j+1} = (g_1 g_j)^b g_{j+1}$. Now note that $(g_1 g_j)^2 = g_1 g_j g_1 g_j = g_1 g_1^{-1} g_1 g_1 g_j = g_j^2$ from the first part of the lemma. Therefore if $b$ is even, then $(g_1 g_j)^b = ((g_1 g_j)^2)^l = (g_j^2)^l = g_j^b$. Using the same logic, if $b$ is odd, $(g_1 g_j)^b = g_1 g_j^b$. Therefore, $g_{j+1}^a g_j^b = (g_1 g_j)^b g_{j+1} = g_1^{b'(-1)^{n+1}} g_j^a$.

To continue with the more general case, $g_{j+1}^a g_j^b = g_{j+1}^{a(-1)^n} g_j^{b'(-1)^{n+1}} g_j g_{j+1} = g_{j+1}^{a(-1)^{n+1}} g_j g_{j+1}$ and after iterating this computation one obtains $g_{j+1}^a g_j^b = g_1^{a b'(-1)^n} g_j^a$.
We now show that the problem \( TSSP(\{k_1, k_2, \ldots, k_n\}, M) \), is equivalent to the instance of the conjugacy problem \( g_1^{-1} g_3 k_1 g_2 \cdots g_{2n+1}^{-1} \). Our general strategy will be as follows: conjugate \( g_1^{-1} g_3 k_1 g_2 \cdots g_{2n+1}^{-1} \) by a generic word in \( G_n \) and collect. First, note that from the multiplication rules above, it can be seen that any \( g_i \) where \( i \) is odd, commutes with \( g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} \). It can also be seen from the lemma that conjugating by one of the generators with even index does not introduce any generators with even index in the collected word. Therefore, without loss of generality, we can assume that our generic word is of the form \( g_2^{-1} x_2 \cdots g_{2n}^{-1} \) because adding in any generators with odd index doesn’t affect the conjugated product. In doing so we find that

\[
(g_2^{-1} x_2 \cdots g_{2n}^{-1})(g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1})(g_2^{-1} x_2 \cdots g_{2n}^{-1})^{-1} = g_1^{-p(k_1, \ldots, k_n, x_1, x_2, \ldots, x_n)} g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1}
\]

where \( p(k_1, \ldots, k_n, x_1, x_2, \ldots, x_n) = k_n x_n^{-1} + x_{n-1} + \cdots + x_1 \).

Notice then, that finding \( x_i \) such that \( p(k_1, \ldots, k_n, x_1, x_2, \ldots, x_n) = M \) is exactly the \( TSSP(\{k_1, k_2, \ldots, k_n\}, M) \). As such, the two words are conjugate if and only if \( TSSP(\{k_1, k_2, \ldots, k_n\}, M) \) has a solution. Additionally, the length of the inputs to the problems are only off by a polynomial. It can be seen that the length of both the \( TSSP \) and the conjugacy problem are \( O(n\log(K)) \) where \( K \) and \( n \) are chosen as before. Additionally, it is clear that the transformation is efficient to compute. Since the general conjugacy problem in \( G_n \) includes all of these instances we then have that the conjugacy problem in \( G_n \) in polynomial time reducible to the \( TSSP \) with \( n \) indeterminates.

**Theorem 4.2.** \( TSSP \) with \( n \) indeterminants is polynomial time reducible to the conjugacy problem in the group \( G_n \) of Hirsch length \( 2n+1 \).

**Proof.** To prove this, we show that solving the conjugacy problem, \( g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} \sim g_1^{-1} g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} \), in \( G_n \) yields a solution to \( TSSP(\{k_1, k_2, \ldots, k_n\}, M) \).

We proceed by induction on \( l \) where we conjugate by the last \( l \) syllables of the generic word. Rather than starting with \( l = 1 \) it may clarify the computation to start with \( l = 2 \). In this case we collect:

\[
(g_{2n}^{-1} g_{2n+1}^{-1})(g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1})(g_{2n}^{-1} g_{2n+1}^{-1})^{-1} =
\]

Conjugating first by \( g_{2n+1}^{-1} \), we find:

\[
\begin{align*}
g_{2n}^{-1} (g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1}) g_{2n}^{-1} &= g_{2n}^{-1} g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} \\
g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} (g_{2n+1}^{-1} g_{2n}^{-1}) &= g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} \\
g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} (g_{2n+1}^{-1} g_{2n}^{-1}) &= g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} \\
g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1} &= g_3^{-1} g_5 k_2 \cdots g_{2n+1}^{-1}
\end{align*}
\]
Negt we conjugate by \( g_{2n-1}^{-x_n} \):
\[
g_{2n-1}^{-x_n}(g_1^{-k_n x_n} g_3^{k_3} g_5^{k_5} \cdots g_{2n-1}^{k_{2n-1}}) g_{2n-1}^{-x_n} =
\]
\[
g_1^{-k_n x_n} g_3^{k_3} g_5^{k_5} \cdots g_{2n-1}^{k_{2n-1}} g_2 g_{2n-2} g_2 g_{2n-1} =
\]
\[
g_1^{-k_n x_n} g_3^{k_3} g_5^{k_5} \cdots g_{2n-1}^{k_{2n-1}} g_2 g_{2n-2} g_2 =
\]
\[
g_1^{-k_n x_n} g_3^{k_3} g_5^{k_5} \cdots g_{2n-1}^{k_{2n-1}} g_2 g_{2n-2} g_2 =
\]
\[
g_1^{-p(k_{n-1}, k_n, x_{n-1}, x_n)} g_3^{k_3} g_5^{k_5} \cdots g_{2n-1}^{k_{2n-1}} g_2 g_{2n-2} g_2 =
\]
\[
g_1^{-p(k_{n-1}, k_n, x_{n-1}, x_n)} g_3^{k_3} g_5^{k_5} \cdots g_{2n-1}^{k_{2n-1}} g_2 g_{2n-2} g_2 =
\]
\[
g_1^{-p(k_{n-1}, k_n, x_{n-1}, x_n)} g_3^{k_3} g_5^{k_5} \cdots g_{2n-1}^{k_{2n-1}} g_2 g_{2n-2} g_2 =
\]

We now induct and assume the result holds for \( l = n - 1 \) and show it holds for \( l = n \). In this case we have:
\[
(g_2^{-x_n} g_3^{k_2} \cdots g_{2n}^{k_{2n}})(g_3^{k_3} g_5^{k_5} \cdots g_{2n+1}^{k_{2n+1}})(g_2^{-x_n} g_3^{k_2} \cdots g_{2n}^{k_{2n}})^{-1} =
\]
\[
g_2^{-x_n}(g_1^{-p(k_2, \cdots, k_n, x_{n-1}, x_n)} g_3^{k_3} \cdots g_{2n+1}^{k_{2n+1}}) g_2^{-x_n} =
\]
Conjugating by \( g_2^{x_n} \) then yields:
\[
g_2^{x_n}(g_1^{-p(k_2, \cdots, k_n, x_{n-1}, x_n)} g_3^{k_3} \cdots g_{2n+1}^{k_{2n+1}}) g_2^{x_n} =
\]
\[
g_2^{x_n}(g_1^{-p(k_2, \cdots, k_n, x_{n-1}, x_n)} g_3^{k_3} \cdots g_{2n+1}^{k_{2n+1}}) g_2^{x_n} =
\]
\[
g_2^{x_n}(g_1^{-p(k_2, \cdots, k_n, x_{n-1}, x_n)} g_3^{k_3} \cdots g_{2n+1}^{k_{2n+1}}) g_2^{x_n} =
\]
\[
g_2^{x_n}(g_1^{-p(k_2, \cdots, k_n, x_{n-1}, x_n)} g_3^{k_3} \cdots g_{2n-1}^{k_{2n+1}}) g_2^{x_n} =
\]
\[
g_2^{x_n}(g_1^{-p(k_2, \cdots, k_n, x_{n-1}, x_n)} g_3^{k_3} \cdots g_{2n-1}^{k_{2n+1}}) g_2^{x_n} =
\]

It is now enough to note that any possible solution to the above conjugacy problem would give you a solution to the equivalent TSSP instance by eliminating all the \( g_j \) with \( j \) odd and reducing all the exponents modulo 2. Therefore a “yes” answer to the conjugacy decision problem using any algorithm would imply the existence of a solution to the TSSP. \( \square \)

5. The Conjugacy Problem In \( G_n \) Is In NP

In this section we show that the conjugacy problem in the groups \( G_n \) can in fact be checked efficiently. To do this we will find closed form expressions for conjugating a word by a power of a single generator. These closed form expressions will be effectively computable with group elements in their normal form. Since conjugating by a single syllable can be done in polynomial time, conjugating by \( 2n+1 \) of them is also polynomial time. Therefore, checking conjugacy is efficient. These methods can also be used to create closed form expressions for multiplying and collecting elements in normal form.

When conjugating elements in \( G_n \) there are three cases to consider: conjugation by powers \( g_1 \), conjugation by powers of \( g_j \) with \( j \) even, and conjugation by powers of \( g_l \) where \( l \) is odd and larger than 1.

For the first case we collect
\[
g_1^{k_1, k_2, \cdots, k_{2n+1}} g_1^{-k} =
\]
Since each of the even \( g_i \) invert \( g_1 \), when we bring the \( g_i^{-k} \) to the left we switch the sign of the exponent according to the parity of the exponents of the even indexed \( g_j \). Also, the odd \( g_i \) commute with \( g_1 \), and do not affect the collection process. Therefore we end up with:

\[
g_1^{k+k_1-k(-1)^{k_2+k_4+\cdots+k_{2n}}} g_2^{k_2} \cdots g_{2n+1}^{k_{2n+1}}
\]

(3)

The second case is then collecting

\[
g_j^{k} (g_1 \cdots g_{2n+1}^{k_{2n+1}}) g_j^{-k}
\]

where \( j \) is even.

We first move the \( g_j^k \) right. Hopping over the \( g_1^{k_1} \) may change the sign of the exponent, but after that, each \( g_i \) commutes with \( g_j \) for \( i < j \). Therefore as a first step we end up with:

\[
(g_1^{k_1(-1)^{\cdot \cdot \cdot j}} g_2^{k_2} \cdots g_j^{k_j+1} g_{j+1}^{k_{j+1}} g_{2n+1}^{k_{2n+1}})^{-k} g_j
\]

In moving the \( g_j^{-k} \) to the left, the only thing that doesn’t commute is \( g_{j+1} \). To hop over \( g_{j+1}^{k_{j+1}} \) we can use Lemma 4.1 and get

\[
g_1^{k_1(-1)^{k}} g_2^{k_2} \cdots g_j^{k_j+1} g_{j+1}^{k_{j+1}} g_{2n+1}^{k_{2n+1}} =
\]

\[
(g_1^{k_1(-1)^{j}} g_2^{k_2} \cdots g_j^{k_j+1} g_{j+1}^{k_{j+1}} g_{2n+1}^{k_{2n+1}})^{-k} g_j
\]

Finally, we move the \( g_{j+1}^{k_{j+1} k'} \) to the left to end up with:

\[
(g_1^{k_1(-1)^{j}} g_2^{k_2} \cdots g_j^{k_j+1} g_{j+1}^{k_{j+1}} g_{2n+1}^{k_{2n+1}})^{-k} g_j
\]

(4)

The third case is dealt with similarly to the second. When \( l > 1 \) is odd:

\[
g_l^{k} (g_1^{k_1} \cdots g_{2n+1}^{k_{2n+1}}) g_l^{-k} =
\]

\[
g_1^{k_1} \cdots (g_l^{k_l} g_{l-1}^{k_{l-1}}) g_l^{k_l-k} \cdots g_{2n+1}^{k_{2n+1}} =
\]

\[
g_1^{k_1} \cdots (g_l^{k_l} g_{l-1}^{k_{l-1}}) g_l^{k_l-k} \cdots g_{2n+1}^{k_{2n+1}} =
\]

\[
(g_1^{k_1} \cdots (g_l^{k_l} g_{l-1}^{k_{l-1}}) g_l^{k_l-k} \cdots g_{2n+1}^{k_{2n+1}})^{-k} g_l
\]

(5)

Since conjugation is done by successively conjugating elements of the form of those in (3), (4), and (5) these closed forms can iteratively perform a general conjugation. Such a computation can be performed in polynomial time in terms of \( n \log(K) \) because computing the normal form after conjugation by each syllable can be done in polynomial time using the closed forms, and need only be performed \( n \) times. This means that we can create a polynomial time verifier for the conjugacy problem in the \( G_n \).
These normal forms also provide us with the following corollaries that describe conjugation in the group. The proofs for both statements can be seen directly by inspecting the closed forms above.

**Corollary 5.1.** Let \( u, v \in G_n \) where \( u = g_1^{e_1} \cdots g_{2n+1}^{e_{2n+1}} \) and \( v = g_1^{f_1} \cdots g_{2n+1}^{f_{2n+1}} \).

i. \( u \sim v \) implies \( e_i = f_i \) for \( i \geq 2 \).

ii. Let \( e_i = f_i \) for \( i \geq 2 \). If there exists an even \( l - 1 \) such that \( e_{l-1} = f_{l-1} \) is odd, the \( u \) and \( v \) are conjugate. In fact, one such conjugator is

\[
g_l^{(f_l-e_l)(-1)^{k_2+k_4+\cdots+k_{l-3}}}
\]

Note that part ii of the above corollary does not include the difficult cases of the conjugacy problem that we saw in section 4.

Now we check that there exists a certificate that is of polynomial length with respect to any instance of the conjugacy problem \( G_n, u \sim v \). Let \( u = g_1^{e_1} g_2^{e_2} \cdots g_{2n+1}^{e_{2n+1}} \) and \( v = g_1^{f_1} g_2^{f_2} \cdots g_{2n+1}^{f_{2n+1}} \) and \( w = g_1^{f_1} \cdots g_{2n+1}^{f_{2n+1}} \) such that \( wuw^{-1} = v \). In the case that there exists an odd exponent above an even indexed generator, we have a certificate of polynomial length from part ii of Corollary 5.1. Therefore, we can assume that for all \( j \) even, \( e_j \) is odd.

In this case, we also know there exists a conjugator where \( x_l = 0 \) for all \( l \) odd and greater than 1 by inspecting the closed forms from (5). Additionally we can take \( x_j \) to be 0 and 1 for \( j \) even by looking at the closed forms from (4). It remains to put bounds on \( x_1 \).

Let \( y \) be the exponent above \( g_1 \) conjugating by \( g_2^{x_2} \cdots g_{2n+1}^{x_{2n+1}} \). Then by repeated applications of (4), \( |y| \leq |e_1| + |e_3| + \cdots + |e_{2n+1}| \). From (3), conjugating by \( g_1^{x_1} \) either increases the exponent by \( 2x_1 \) or leaves it unchanged. Therefore if \( f_1 = y \), we can take \( x_1 \) to be 0 and then clearly a certificate has length \( O(n) \). Otherwise, \( f_1 = y + 2x_1 \) implying that \( |x_1| \leq |k_1| + |y| \leq |f_1| + |e_1| + |e_3| + \cdots + |e_{2n+1}| \). This means that the length of a certificate is bounded from above by \( \log(|f_1| + |e_1| + |e_3| + \cdots + |e_{2n+1}|) + n \) which is of polynomial size in the length of the original conjugacy problem. This now shows that the conjugacy problem in \( G_n \) is in NP.

One could also compose these operations to find a single closed form for conjugation in general. Such a closed form, would be not unlike the one computed in the previous section, but altogether much more complicated. If instead we consider right or left multiplication by syllables, we can obtain closed forms for multiplication of normal forms. By using these closed forms, we can also perform these algebraic operations with elements represented by exponent vectors in polynomial time.

6. Reduction of TSSP to SSP

In this section we show that \( SSP \leq_p TSSP \). To make this easier we introduce another problem \( SSP' \) that is similar to \( SSP \) and in fact show that \( SSP \leq_p SSP' \leq_p TSSP \). We define \( SSP' \) as follows: given a list of integers \( \{k_1, \cdots, k_n\} \) and an integer \( M \), decide if there exists a solution to the equation:
\( k_1 x_1 + \cdots + k_n x_n = M \) where \( x_1, \ldots, x_n \in \{-1, 0, 1\} \)

We first show that \( \text{SSP}' \leq_p \text{TSSP} \). Consider \( \text{SSP}'(\{k_1, \cdots, k_n\}, M) \), an instance of \( \text{SSP}' \). Notice that this is equivalent to \( \text{TSSP}(\{k_1, 0, k_2, 0, \cdots, 0, k_n\}, M) \). If \( (x_1, \cdots, x_n) \) is a solution for the \( \text{SSP}' \), then \( (y_1, \cdots, y_{2n-1}) \) is a solution for the corresponding \( \text{TSSP} \) where \( y_{2j-1} = x_j \) and \( y_{2j} = 1 \) if \( x_{j-1} = 1 \) and \( x_j = -1 \) or if \( x_{j-1} = -1 \) and \( x_j = 1 \) and is otherwise 0. Altogether, what we are doing is adding in extra values in the list that don’t affect the sum, but allow you control if the following variable is added or subtracted in the sum. Conversely, we can take any solution to the \( \text{TSSP} \) and get a solution for \( \text{SSP}' \) by “forgetting” what values the even \( y_j \) take and only pay attention to whether each \( k_i \) appears as itself or as \(-k_i \) in sum. In this vein, any solution \( (y_1, \cdots, y_{2n-1}) \) gives you the solution \( (x_1, \cdots, x_n) \) where \( x_1 = y_1 \) and for \( i > 1 \), \( x_i = y_{2i-1} - y_{2i-2} \). Since we are only increasing the length of the list by a factor of 2, we have \( \text{SSP}' \leq_p \text{TSSP} \). Thus we have proved:

**Lemma 6.1.** \( \text{SSP}' \leq_p \text{TSSP} \).

It is more work to then show that \( \text{SSP} \leq_p \text{SSP}' \). We adapt our proof from the appendix of [13]. Consider the following systems of equations:

\[
\begin{align*}
1. & \quad \sum_{i=1}^{n} k_i x_i = M \\
2. & \quad x_i \in \{0, 1\}
\end{align*}
\]

(6)

\[
\begin{align*}
1. & \quad \sum_{i=1}^{n} k_i x_i = M \\
2. & \quad x_i + y_i = 1 \quad \text{for} \quad i = 1, \cdots, n \\
3. & \quad x_i, y_i \in \{-1, 0, 1\}
\end{align*}
\]

(7)

\[
\begin{align*}
1. & \quad \sum_{i=1}^{n} (4^{n-i} + 4^n k_i) x_i + \sum_{i=1}^{n} 4^{n-i} y_i = 4^n M + 4^n - 1 \\
2. & \quad x_i, y_i \in \{-1, 0, 1\}
\end{align*}
\]

(8)

First, note that (6) and (7) have equivalent solutions: any set of \( x_i \) that satisfies one will satisfy the other. The constraints \( x_i + y_i = 1 \) and \( x_i, y_i \in \{0, 1\} \) prevent \( x_i \) from ever being \(-1\). What is less apparent is that (7) and (8) have the same solution set. If this is the case, we can solve any instance of \( \text{SSP} \), (6), using an algorithm that solves the equivalent \( \text{SSP}' \) (8). If we also show that the size of (8) is only polynomially larger than (6) then we will have in fact shown that \( \text{SSP} \leq_p \text{SSP}' \) and proving that both \( \text{SSP} \) and \( \text{TSSP} \) are NP-complete.

**Proposition 6.2.**

i. The following systems of equations have the same set of solutions:

\[
\begin{align*}
1. & \quad \sum_{i=1}^{n} k_i x_i = M \\
2. & \quad x_1 + y_1 = 1 \\
3. & \quad x_i, y_i \in \{-1, 0, 1\}
\end{align*}
\]

(9)

\[
\begin{align*}
1. & \quad x_1 + y_1 + 4 \sum_{i=1}^{n} k_i x_i = 4M + 1 \\
2. & \quad x_i, y_i \in \{-1, 0, 1\}
\end{align*}
\]

(10)
Moving from (9) to (10) can be done in polynomial time and space.

Proof. i. First note that anything that is a solution to (9) is a solution to (10). In the other direction, assume that \((x_1, \ldots, x_n, y_1)\) is a solution to (10). Note that that is equivalent to saying that:

\[
4\left(\sum_{i=1}^{n} k_i x_i - M\right) = 1 - x_1 - y_1
\]

Using the fact that \(-1 \leq 1 - x_1 - y_1 \leq 3\) we then get

\[
-1 \leq 4\left(\sum_{i=1}^{n} k_i x_i - M\right) \leq 3
\]

Finally, since \(\sum_{i=1}^{n} k_i x_i - M\) is an integer, (12) can only be satisfied if \(\sum_{i=1}^{n} k_i x_i = 0\) implying that \((x_1, \ldots, x_n)\) is a solution for (9).

ii. Let the length of (6) be \(O(n \log(K))\) where \(K\) is an upper bound of the absolute values of all the coefficients. This is because we have \(n + 1\) indeterminates and any coefficient can be expressed in at most \(log(K)\) digits. Since we will need to eventually see that iterating this process is polynomial in \(n \log(K)\), we bound our multiplicative constant, in this case 4, from above by \(nK\) which we can assume for \(n\) and \(K\) both greater than or equal to 2. Note that if either is less than 2, the problem becomes trivial. In doing so, we can bound the absolute values of the coefficients from above by \(nK^2\) making the size of (7) \(O(n \log(nK))\) since we have \(n + 1\) variables with coefficients of length at most \(log(nK^2)\). Since \(O(n \log(nK))\) is polynomial in \(O(n \log(K))\) and since the calculations to turn (6) into (7) can be done in polynomial time we have a polynomial time reduction.

Proposition 6.3. The systems of equations (7) and (8) have the same set of solutions.

Proof. As we did in the previous proposition, we combine the conditions \(x_1 + y_1 = 1\) to the equation \(\sum_{i=1}^{n} k_i x_i = M\) to obtain an instance of \(SSP'\) whose solution will yield a solution to the corresponding instance of the \(SSP\).

We then continue as in the proposition, merging our system of equations into just one, by performing the same steps beginning with \(x_1 + y_1 = 1\) and ending with \(x_n + y_n = 1\). Note, that as we perform each step, we are not changing the solution set. After we have performed the first two steps we have the equation:

\[
x_2 + y_2 + 4x_1 + 4y_1 + 4^2 \sum_{i=1}^{n} k_i x_i = 4(4M + 1) + 1
\]

and then after \(n\) steps, we have obtained:

\[
\sum_{i=1}^{n} 2^{n-1} x_i + \sum_{i=1}^{n} 2^{n-1} y_i + 4^n \sum_{i=1}^{n} k_i x_i = 4^n M + 4^{n-1} + 4^{n-2} + \cdots + 1
\]

After collecting like terms on the left and summing the geometric series on the right we have (8).
Note that from the Proposition 5.2 adding each $x_i + y_i = 1$ only increases the length of the problem by an amount polynomial in $n \log(K)$, so therefore after combining all the $x_i + y_i = 1$, the problem will still have only increased by a polynomial amount. As such, $SSP \leq_p SSP'$ and we have our main result. To summarize:

**Theorem 6.4.** $SSP \leq_p SSP' \leq_p TSSP$ implying that the TSSP is NP-complete and furthermore so is the conjugacy decision problem in the $G_n$.

Furthermore, from the argument in section 3, a polynomial time algorithm for the conjugacy search problem over the $G_n$ would imply $P = NP$.

7. **Acknowledgments and Support**

The authors would like to thank Vladimir Shpilrain for many useful conversations.

Delaram Kahrobaei is partially supported by the Office of Naval Research grant N00014120758 and also supported by PSC-CUNY grant from the CUNY research foundation, as well as the City Tech foundation.

**References**

[1] Iris Anshel, Michael Anshel, and Dorian Goldfeld. An algebraic method for public-key cryptography. *Mathematical Research Letters*, 6:287–292, 1999.
[2] Bren Cavallo, Jorge Delgado, Delaram Kahrobaei, Ha Lam, and Enric Ventura. Tit’s alternative for the automorphism group of a rigid poly-$Z$ group. *Preprint*.
[3] Cornelia Drutu and Michael Kapovich. Lectures on geometric group theory. *Preprint*, 2013.
[4] B. Eick. *Algorithms for Polycyclic Groups*. Habilitationsschrift, Universität Kassel, 2001.
[5] Bettina Eick and Delaram Kahrobaei. Polycyclic groups: A new platform for cryptography? *arXiv preprint math/0411077*, 2004.
[6] David Garber, Delaram Kahrobaei, and Ha T Lam. Analyzing the length-based attack on polycyclic groups. *arXiv preprint arXiv:1305.0548*, 2013.
[7] David Garber, Shmuel Kaplan, Mina Teicher, Boaz Tsaban, and Uzi Vishne. Length-based conjugacy search in the braid group. *Contemporary Mathematics*, 418:75, 2006.
[8] Derek F. Holt, Bettina Eick, and Eamonn A. O’Brien. *Handbook of Computational Group Theory*. CRC Press, 2005.
[9] Delaram Kahrobaei and Michael Anshel. Decision and search in non-abelian Cramer-Shoup public key cryptosystem. *Groups Complexity Cryptology*, 1(2):217–225, 2009.
[10] Delaram Kahrobaei and Bilal Khan. A non-commutative generalization of ElGamal key exchange using polycyclic groups. In *GLOBECOM*, 2006.
[11] Delaram Kahrobaei and Charalampos Koupparis. Non-commutative digital signatures using non-commutative groups. *Groups, Complexity Cryptology*, 4:377–384, 2012.
[12] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.
[13] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.
[14] Ki Hyoung Ko, Sang Jin Lee, Jung Hee Cheon, Jae Woo Han, Ju-sung Kang, and Choonsik Park. New public-key cryptosystem using braid groups. In *Advances in Cryptology, CRYPTO 2000*, pages 166–183. Springer, 2000.
[15] Alex D Myasnikov and Alexander Ushakov. Length based attack and braid groups: cryptanalysis of anshel-anshel-goldfeld key exchange protocol. In *Public Key Cryptography–PKC 2007*, pages 76–88. Springer, 2007.
[16] Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in groups. *To appear in Mathematics of Computation*. Available at http://arxiv.org/abs/1302.5671.
[17] Alexei Myasnikov, Vladimir Shpilrain, and Alexander Ushakov. *Group-based Cryptography*. Springer, 2008.
A FAMILY OF POLYCYCLIC GROUPS OVER WHICH THE CONJUGACY PROBLEM IS NP-COMPLETE

Bren Cavallo, Department of Mathematics, CUNY Graduate Center, City University of New York
E-mail address: bcavallo@gc.cuny.edu

Delaram Kahrobaei, CUNY Graduate Center, PhD Program in Computer Science and NYCCT, Mathematics Department, City University of New York
E-mail address: dkahrobaei@gc.cuny.edu