6-2017

Solving Algorithmic Problems in Finitely Presented Groups via Machine Learning

Jonathan Gryak
The Graduate Center, City University of New York

Solving Algorithmic Problems in Finitely Presented Groups via Machine Learning

by

Jonathan Gryak

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2017

Solving Algorithmic Problems in Finitely Presented Groups via Machine Learning

by

Jonathan Gryak

This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Date	Delaram Kahrobaei **Chair of Examining Committee**
Date	Robert Haralick **Executive Officer**
	Benjamin Fine Robert Haralick Delaram Kahrobaei Vladimir Shpilrain Xiaowen Zhang **Supervisory Committee**

The City University of New York

Abstract

Solving Algorithmic Problems in Finitely Presented Groups via Machine Learning

by

Jonathan Gryak

Advisor: Professor Delaram Kahrobaei

Machine learning and pattern recognition techniques have been successfully applied to algorithmic problems in free groups. In this dissertation, we seek to extend these techniques to finitely presented non-free groups, in particular to polycyclic and metabelian groups that are of interest to non-commutative cryptography.

As a prototypical example, we utilize supervised learning methods to construct classifiers that can solve the conjugacy decision problem, i.e., determine whether or not a pair of elements from a specified group are conjugate. The accuracies of classifiers created using decision trees, random forests, and N-tuple neural network models are evaluated for several non-free groups. The very high accuracy of these classifiers suggests an underlying mathematical relationship with respect to conjugacy in the tested groups.

In addition to testing these techniques on several well-known finitely presented groups, we introduce a new family of metabelian groups for which we analyze the computational complexity of the conjugacy search problem. We prove that for the family in general the time complexity of the conjugacy search problem is exponential, while for a subfamily the problem is polynomial. We also show that for some of these groups the conjugacy search problem is an instance of the discrete logarithm problem.

We also apply machine learning techniques to solving the conjugacy search problem. For each platform group we train a N-tuple regression network that can produce a candidate conjugator for a pair of conjugate elements. This candidate is then used as the initial state of a local search for a conjugator in the Cayley graph, in what we call regression-based conjugacy search (RBCS). RBCS can be applied to groups such as polycyclic groups for which other heuristic approaches, such as the length-based attack, are ineffective.

Acknowledgments

First and foremost I would like to thank my advisor Delaram Kahrobaei, who gave me the opportunity to study at the Graduate Center. She has tirelessly advocated on my behalf, enabled me to interact and collaborate with a global cadre of researchers, and provided me with many interesting problems for study. I would like to thank Benjamin Fine, my fellow speaker of Brooklynese, who showed me through his example the expansive and gratifying nature of academic life, and who convinced me to further my education. He has supported me throughout my entire graduate studies, and remains an inspiration to me to this day.

I give thanks to Robert Haralick, who provided a great deal of guidance concerning machine learning and with whom I had many stimulating conversations on computer science, philosophy, and life in general. I thank him also for making the Computer Science program such a richly rewarding experience. I thank Vladimir Shpilrain for his scholarship and his helpful suggestions and guidance throughout my studies. To Xiaowen Zhang, I thank you for your enthusiastic support and for the many great conversations had at the various colloquia we attended together. A special thank you goes to Conchita Martinez-Perez, who served on my second exam committee and with whom I collaborated on my first paper along with Delaram. I would like to thank all of my committee members again for their useful comments and suggestions throughout my examinations.

I would like to thank Dilvania Rodriguez, who performs a yeoman's service in supporting the CS department, and has provided me with personal support and encouragement. I also thank the Computer Science faculty for their scholarship and their support of us graduate students, and in particular Susan Epstein and Sergei Artemov for their excellent pedagogy.

I would like to thank my wife, family, and friends, who have supported me throughout the years with their love and encouragement.

Finally, I acknowledge support from the Office of Naval Research through a grant awarded to Delaram Kahrobaei and Vladimir Shpilrain.

J.A.G.

To my wife, Rachel

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Group theory has been a rich source of decision problems dating back to Max Dehn, who in 1911 [11] articulated the word, conjugacy, and isomorphism problems for finitely generated groups. The exploration of solutions to these problems gave rise to combinatorial group theory - the study of groups via their presentations, i.e., sets of generators and relations. In 1955, the word problem was proven by Novikov [42] to be undecidable in general, one of the first undecidable problems to be found outside of mathematical logic. Much subsequent work in combinatorial group theory has determined the decidability of the word, conjugacy, and (to a lesser extent) the isomorphism problem for many classes of groups.

Each of the above classic decision problems has a dual in the form of a search or "witness" problem; the latter so called due to the requirement that a valid solution produces a group element that acts as a witness to the affirmative answer of the decision problem. For the word search problem, a word w known to be equivalent to the group identity is rewritten as a product of the group's elements and relators, while for the conjugacy search problem, an element z is produced for a known conjugate pair x, y such that $x^z = y$. Beyond the classic problems, there are many other algorithmic decision/search problem pairs that have been

studied in group theory, including the subgroup membership problem and various problems concerning morphisms.

Spurred by the 1994 publication of Shor's quantum algorithm [49] for solving discrete logarithm and integer factorization problems, upon which the security of most commercial cryptographic systems rely, mathematicians and cryptographers sought new hardness assumptions distinct from those aforementioned number-theoretic standards. The wellspring of algorithmic problems from group theory was put to good use through the development of non-commutative cryptography, the first modern instance of which was the commutator key exchange introduced [2] by Anshel, Anshel, and Goldfeld (AAG).

The security of the AAG key exchange is based in part on the conjugacy search problem. As new cryptographic schemes are developed, there is a natural parallel advancement in cryptanalysis. The original AAG protocol operating over braid groups was found to be susceptible to a heuristic algorithm known as the length-based attack [26]. This has motivated the search for other potential platform groups. In general, groups suitable for use in non-commutative cryptography must be well-known and possess the following properties: a solvable word problem, a computationally difficult group-theoretic problem, a "fast" word growth rate, and the namesake non-commutativity [39].

In 2004, Eick and Kahrobaei [12] investigated the complexity of the word and conjugacy problems in polycyclic groups: groups possessing a finite subnormal series of subgroups with cyclic factors. Their experiments showed that while the time complexity of the conjugacy problem grew exponentially with increased Hirsch length (the number of subgroups isomorphic to $\mathbb{Z}$ in the subnormal series), the word problem remained efficiently solvable. This suggested the suitability of polycyclic groups as platform groups, and stimulated further investigation into their use in non-commutative cryptosystems.

Algorithmic problems in polycyclic groups can be studied in a variety of ways, including through their linear representations, multiplication polynomials, or through polycyclic presentations [20]. In [23], Haralick et al. suggested a machine learning approach to solving algorithmic problems in free groups. In this dissertation we seek to extend these results to non-free groups in general and polycyclic groups in particular. As a prototypical example, we will use machine learning techniques to solve the conjugacy search and decision problems in a variety of groups, including a family of polycyclic and metabelian groups that have an exponentially bounded conjugacy search problem.

In **Chapter 2** we provide the group theory and non-commutative cryptography background needed to facilitate the reader's comprehension of our exposition. In **Chapter 3**, we introduce the aforementioned family of polycyclic groups, proving the upper exponential bound on the conjugacy search problem and the relationship between it and the discrete logarithm problem. In **Chapter 4** we present our method for implementing a machine learning system for non-free groups, including methods for feature extraction, model selection, data generation, and system evaluation. In **Chapter 5** we apply this methodology to solving the conjugacy decision problem in a variety of non-free finitely presented groups. In **Chapter 6**, we present a framework for applying these techniques to the conjugacy search problem, as well as suggest additional improvements and uses of our machine learning method.

Chapter 2

A Primer on Group Theory and Non-Commutative Cryptography

2.1 Group Theory

In this section we provide an overview of group theory that is relevant to this dissertation and the work contained herein. We assume that the reader has a basic understanding of group theory gleaned from an undergraduate course in the subject, from a graduate course in theoretical computer science, or from a familiarity with traditional cryptographic protocols such as RSA or Diffie-Hellman key exchange. The content here is not exhaustive; additional concepts will be introduced in context as needed.

2.1.1 Conjugacy

Definition 2.1.1. For a given group G, two elements $u, v \in G$ are *conjugate*, denoted $u \sim v$, if there exists an element $z \in G$ such that

$$zuz^{-1} = v.$$

As with any group action, conjugation can be defined equivalently as either a left or right action. The definition above uses the *left action convention*. However, throughout this dissertation we will denote conjugation exponentially as

$$u^z = zuz^{-1}.$$

Conjugation by a fixed element of G is an *inner automorphism*, i.e., a bijection from G to itself that is also a group homomorphism.

2.1.2 Semidirect Products

One of the standard ways in which to construct non-commutative groups is through the use of the semidirect product. The semidirect product of two groups is a generalization of the direct product, wherein only one of the groups is normal in the resultant group.

Definition 2.1.2. Given two groups H and K, along with a homomorphism $\phi : K \to \mathrm{Aut}(H)$, we can construct a new group G called the *semidirect product* and denoted by

$$G = H \rtimes_\phi K.$$

Multiplication in the semidirect product G is defined as

$$(h_1, k_1)(h_2, k_2) = (h_1\phi(k_1) \cdot h_2, k_1k_2)$$

with $(h_1, k_1), (h_2, k_2) \in G$ and $\cdot$ denoting the action of $\phi(k_1)$ on h_2. The $\rtimes$ symbol is used to indicate that H is a normal subgroup of G.

G contains subgroups $H' = \{(h, 1) \mid h \in H\}$ and $K' = \{(1, k) \mid k \in K\}$ that are isomorphic to H and K respectively. It follows that the group action $\phi(k) \cdot h$ is equivalent to h^k, i.e.,

$$\phi(k) \cdot h = h^k = khk^{-1}.$$

Note that G will be non-abelian provided that ϕ is not equivalent to the trivial homomorphism.

2.1.3 Torsion

The concepts of torsion elements and subgroups are instrumental to the proof in section 3.2 concerning the computational complexity of the conjugacy search problem in $\mathcal{F}$.

Definition 2.1.3. Let G be an abelian group. An element $g \in G$ has *finite order* n if $\exists n \in \mathbb{Z} \setminus \{0\}$ such that $ng = 0$. If no such n exists, the element is said to have *infinite order*. For each $n \in \mathbb{Z}$, the elements that have the same order n form a subgroup of G, denoted $G[n]$. The set of all elements with finite order forms a subgroup $T \leq G$ called the *torsion subgroup of* G.

Definition 2.1.4. Let G be an abelian group. For $p \in \mathbb{Z}$, with p a fixed prime, the subgroup G_p is called a *p-primary component* of G.

Definition 2.1.5. The *exponent of a torsion subgroup* T, denoted $\exp(T)$, is the smallest positive integer k such that $kv = 0$ for any $v \in T$. If no such integer exists, then by convention $\exp(T) = \infty$.

Theorem 2.1.1 (Primary Decomposition Theorem)**.** *Let G be an abelian group with torsion subgroup T, then*

$$T = \oplus_p G_p,$$

with G_p the $p-$primary components of G.

2.1.4 Free Groups, Group Presentations, and Generic Sets

Analogous to the use of set-builder notation for sets, groups can be defined via group presentations. Before defining presentations, we must first be explicit about what we mean by "words" in the context of free groups. Let X be a set of symbols, and let X^{-1} denote the set of inverses of X, e.g., if $a \in X$, then $a^{-1} \in X^{-1}$. A *word* on $Y = X \cup X^{-1}$ is a finite string of elements of Y. A *reduced word* w is a word on Y such that no subwords of the form $x^{-1}x$ or xx^{-1} exist in w. We are now ready to define a free group:

Definition 2.1.6. The *free group on* X, denoted F_X, is the set of all reduced words on Y. The group operation is string concatenation, with the reduction of any subwords of the form $x^{-1}x$ or xx^{-1} to the empty symbol as necessary. The *rank* r of the free group F_X is the cardinality of X, and up to isomorphism there is precisely one free group of rank r.

We need one additional concept before defining group presentations, that of the normal closure:

Definition 2.1.7. Let $R \leq G$ be a subset of the group G. The *normal closure of* R, denoted R^G, is the intersection of all normal subgroups of G that contain R. Or, more constructively, R^G is defined as the group generated from conjugating R by all elements of G:

$$R^G = \langle gRg^{-1} \mid g \in G \rangle.$$

Definition 2.1.8. Let X be a set of symbols and R a set of freely reduced words from F_X. The group $G \cong F_X/R^G$ is said to be given by a *presentation*

$$\langle X \mid R \rangle.$$

The corresponding mapping $\phi : F(X) \to G$ with $\ker(\phi) = R^G$ is called the *canonical epimorphism*.

The elements of X are called the *generators* of G, while elements of R are called *relators*. The cardinality of the sets X and R determine different classes of groups. If X and R are finite, G is said to be *finitely presented*. A special subclass of finitely presented groups have presentations with $|R| = 1$, these are called *one-relator groups*. If only X is finite, then G is *finitely generated*. In this dissertation, we will only be considering finitely presented groups.

We conclude our discussion of free groups with an additional concept that is relevant to our analysis of length-based conjugacy search in section 6.1.1, that of the generic free basis property. We first require the notion of a generic set, whose definition we take from [28].

Given a finite alphabet X, let $(X^*)^k$ denote the set of all k-tuples of words on X. The *length* of a k-tuple is defined as the length of the words within it. For all $n \geq 0$, let B_n be the set of all k-tuples of length n.

Definition 2.1.9. Let S be a subset of $(X^*)^k$. The *asymptotic density* of S, denoted $\rho(S)$, is defined as

$$\rho(S) = \limsup_{n\to\infty} \frac{|S \cap B_n|}{|B_n|}.$$

A set S is considered *generic* in $(X^*)^k$ if $\rho(S) = 1$. Conversely, a set S is considered *negligible* in $(X^*)^k$ if $\rho(S) = 0$.

If a subset S of a free group G possesses a property $\mathcal{P}$, and if S can be shown to be generic in G, then probabilistically the property $\mathcal{P}$ holds for all words in G. Akin to this concept, we can now define the (generic) free basis property, following [40]:

Definition 2.1.10. Let G be given by a presentation $\langle X \mid R\rangle$. A k-tuple of words $(w_1, \ldots, w_k)$ from the free group on X has the *free basis property* in G if it freely generates a free subgroup of G. The group G has the *generic free basis property* if the free basis property is generic for every $k \geq 1$ and finite generating set X of G.

2.1.5 Algorithmic Problems

As mentioned in the introduction, group theory is an abundant source of decision and other algorithmic problems. Below we provide the explicit definitions of the classic decision problems given by Dehn, along with those for the search problems relevant to non-commutative cryptography.

Decision Problems

In 1911, Max Dehn introduced [11] three decision problems on finitely presented groups - the word problem, the conjugacy problem, and the isomorphism problem. In the definitions below, let G be a finitely presented group:

- *Word Decision Problem* - For any $g \in G$, determine if $g = 1_G$, the identity element of G.
- *Conjugacy Decision Problem* - Determine for any $u, v \in G$ if u is conjugate to v.
- *Isomorphism Decision Problem* - Given groups G and G' with respective finite presentations $\langle X \mid R \rangle$ and $\langle X' \mid R' \rangle$, determine if G is isomorphic to G'.

An additional decision problem called the *subgroup membership decision problem* (also called the *generalized word decision problem*) asks for any $g \in G$ and subgroup $H \leq G$, determine if $g \in H$.

Search Problems

Let G be a group with elements $a_1, \ldots, a_n, b_1, \ldots, b_n$ such that $a_i \sim b_i$. The problem of finding a $c \in G$ such that for all i, $a_i^c = b_i$ is called the *(single) conjugacy search problem* for $i = 1$ and the *multiple conjugacy search problem* for $1 < i \leq n$.

2.1.6 Normal Forms

The *normal form* for elements of a group can in general be construed as the unique and most concise representation of each element in the group. For free groups the normal form of an element is its reduced word representation. Normal forms need not be words; they can be numbers, sequences, or some other formal representation. Note that in some contexts the uniqueness of normal forms may be relaxed.

We can give a general but precise definition of normal forms by using the language of rewriting systems. Let X be a set and $\rightarrow$ be a binary relation on X, with $\stackrel{*}{\rightarrow}$ its reflexive, transitive closure. The relation $\rightarrow$ is said to be *terminating* if there exist no infinite sequences $x_0 \rightarrow x_1 \rightarrow \ldots$ for any $x_i \in X$. The relation $\rightarrow$ is *confluent* if for all $x, y, z \in X$ with $x \stackrel{*}{\rightarrow} y$ and $x \stackrel{*}{\rightarrow} z$, there exists a $w \in X$ such that $y \stackrel{*}{\rightarrow} w$ and $z \stackrel{*}{\rightarrow} w$. An element $x \in X$ is *irreducible* if no $y \in X$ exists such that $x \stackrel{*}{\rightarrow} y$. We can now give the following definition:

Definition 2.1.11. Let X be a set and $\rightarrow$ be terminating and confluent. For every element $x \in A$ there exists a unique, irreducible element $n_x \in X$ called the *normal form of* x, with $x \stackrel{*}{\rightarrow} n_x$. Moreover, x and y are equivalent under $\stackrel{*}{\rightarrow}$, that is $x \stackrel{*}{\rightarrow} y$ and $y \stackrel{*}{\rightarrow} x$, if and only if $n_x = n_y$.

A group may have zero, one, or many normal forms. The existence of at least one normal form for a group implies the decidability of the word problem. For some finitely presented groups (e.g., automatic groups) the Knuth-Bendix algorithm [30] can be used to create a confluent, terminating rewriting systems with respect to the generating set X. In other classes of groups, such as polycyclic groups, the collection algorithm can be used (see section 2.1.9).

2.1.7 Growth Rate

Let G be a finitely generated group. The growth rate of a group is specified by its *growth function* $\gamma : \mathbb{N} \longrightarrow \mathbb{R}$ defined as $\gamma(n) = \#\{w \in G : l(w) \leq n\}$, where $l(w)$ is the length of w as a word in the generators of G. The growth rate of a group imposes restrictions on its algebraic structure, e.g., whether or not it is virtually nilpotent. In non-commutative cryptography, growth rate is important as it affects the size of the *key space*, the set of all possible candidate keys available to a cryptosystem.

2.1.8 Nilpotent Groups

Nilpotent groups are a superclass of abelian groups. Before defining them we will need some additional terminology. For any elements $x, y \in G$, the *commutator of x and y* is $[x, y] = xyx^{-1}y^{-1}$. The set of all commutators of all elements $x, y \in G$, is a normal subgroup of G denoted by $[G, G]$, and is called either the *commutator subgroup* or *derived subgroup* of G.

Using the commutator one can define a series of subgroups called the *lower central series*,

$$G = G_1 \trianglerighteq G_2 \trianglerighteq \cdots \trianglerighteq G_n \trianglerighteq \cdots$$

with $G_{i+1} = [G_i, G]$. We can now use the lower central series to define a nilpotent group:

Definition 2.1.12. A *nilpotent group* is a group with a finite lower central series that terminates with the trivial group, i.e.,

$$G = G_1 \trianglerighteq G_2 \trianglerighteq \cdots \trianglerighteq G_n = \{1\}.$$

Nilpotent groups are contained within the class of polycyclic groups, which we define in

the next subsection. The use of nilpotent groups in this dissertation will be restricted to differentiating between subclasses of polycyclic groups.

2.1.9 Polycyclic Groups

Polycyclic groups are natural generalizations of cyclic groups that are finitely presented and can be readily represented computationally via their eponymous presentations. This ease of representation, amongst other properties, have motivated their use in group-based cryptography. We describe here the aspects of polycyclic groups that are salient to this dissertation. For further details on their algebraic theory and use in group-based cryptography, see [20], a joint work from which this section is derived.

Polycyclic Sequences and Hirsch Length

A group G is said to be *polycyclic* if it has a subnormal series $G = G_1 \triangleright \cdots \triangleright G_{n+1} = \{1\}$ such that the quotient groups G_i/G_{i+1} are cyclic. This series is called a *polycyclic series*. The *Hirsch length* of a polycyclic group G is the number of infinite groups in its polycyclic series. Though a polycyclic group can have more than one polycyclic series, as a consequence of the Schreier Refinement Theorem, its Hirsch length is independent of the choice of series.

Polycyclic Presentations

Every polycyclic group can be described by a polycyclic presentation:

$$\begin{aligned} \langle g_1, \ldots, g_n \mid \quad g_i^{g_j} &= u_{j,i} \quad \text{for } 1 \le j < i \le n, \\ g_i^{g_j^{-1}} &= v_{i,j} \quad \text{for } 1 \le j < i \le n, \\ g_i^{r_i} &= w_{i,i} \quad \text{for } i \in I \rangle, \end{aligned}$$

where $u_{j,i}, v_{i,j}$ are words in the generators $g_{j+1}, \ldots, g_n$; $w_{i,i}$ are words in the generators $g_{i+1}, \ldots, g_n$; and I is the set of indices $i \in \{1, \ldots, n\}$ such that $r_i = [G_i : G_{i+1}]$ is finite.

In a polycyclic group G with polycyclic sequence X, any element g can be represented uniquely in *normal form* as a product of powers of the generators of G:

$$g = x_1^{e_1} \cdots x_n^{e_n},$$

with $e_i \in \mathbb{Z}$. The sequence $(e_1, \ldots, e_n)$ is called the *exponent vector* of g with respect to X.

This special type of finite presentation reveals the polycyclic structure of the underlying group, see [24, Chapter 10] for details. Unlike general finite presentations, a polycyclic presentation enables the word problem to be solved using an algorithm called *collection*. While the collection algorithm is generally effective in practice, its precise computational complexity remains unknown. For finite groups, collection from the left was shown to be polynomial by Leedham-Green and Soicher [33]. For infinite groups, the complexity of the collection algorithm with respect to word length remains unknown.

Matrix Representation

It is well-known that every polycyclic group can be embedded into $GL(n, \mathbb{Z})$ for some $n \in \mathbb{N}$. For groups that are additionally torsion-free and nilpotent, a matrix representation can be computed from the polycyclic presentation using the algorithm of Lo and Ostheimer [34]. Multiplication of group elements in their matrix form is polynomial in the dimension n of the representation.

Algorithmic Problems in Polycyclic Groups

For polycyclic groups all three of the classic problems in Section 2.1.5 are decidable. The conjugacy decision problem for polycyclic groups is decidable by the results of Remeslennikov [44] and Formanek [15]. That the word problem is decidable can be observed from its

formulation as a special case of the conjugacy decision problem (where $g = u, v = 1_G$), or by observing that every word has a unique normal form induced by a polycyclic presentation. The isomorphism decision problem for polycyclic groups is solvable by a result of Segal [48].

In polycyclic groups, the multiple conjugacy search problem for n elements reduces to n independent solutions of single conjugacy search [12]. Mal'cev showed [36] that the subgroup membership search problem is solvable for polycyclic groups.

2.1.10 Metabelian and Solvable Groups

Definition 2.1.13. A group G is *metabelian* if and only if it has an abelian normal subgroup A such that G/A is abelian. Alternatively, it is a group with a subnormal series of length 2:

$$\{1\} = G_0 \triangleleft G_1 \triangleleft G_2 = G,$$

and G_2/G_1 is abelian.

Example 2.1.1. The dihedral group of order 8 is metabelian.

Metabelian groups are a special case of *solvable* groups, whose definition removes the restriction on the length of the subnormal series. Note that all polycyclic and metabelian groups are solvable, but not all metabelian groups are polycyclic:

Example 2.1.2. The Baumslag-Solitar group

$$BS(1,2) = \langle a, b \mid bab^{-1} = a^2 \rangle$$

is metabelian but not polycyclic.

The word problem is decidable in finitely generated metabelian groups [5], as is the

conjugacy decision problem [41]. The decidability of the isomorphism problem is currently unknown.

2.2 Non-Commutative Cryptography

As mentioned in the introduction, non-commutative cryptographic systems replace algorithmic problems from number theory with group-theoretic problems for their hardness assumptions. In this section we include background material from standard and non-commutative cryptography that is referenced throughout the remainder of the text.

The discrete logarithm hardness assumption undergirds well-known cryptosystems such as Diffie-Hellman key exchange and ElGamal encryption. As we will show in chapter 3, there are particular groups for which solving the conjugacy search problem is equivalent to finding a discrete logarithm. The depiction of the seminal Anshel, Anshel, and Goldfeld (AAG) key exchange protocol provides a concrete example of how non-commutative cryptosystems operate. The protocol's use of the conjugacy search problem gave rise to the length-based attack algorithm, which can be modified to perform conjugacy search instead of cryptanalysis; we provide the pseudocode for such a modification below.

2.2.1 Security and the Discrete Logarithm Problem

A cryptosystem exhibiting *perfect security* means that, from an information-theoretical standpoint, no information can be obtained from the encrypted data without the encryption key. Note that this property is irrespective of the computational power of any adversary wishing to obtain the unencrypted information. As such systems are often impractical, modern cryptographic systems replace perfect security with the notion of *semantic security*, whereby a cryptosystem is deemed secure if any probabilistic, polynomial time algorithm

that can decrypt information without the key does so with negligible probability. Negligible functions [29, Def 3.4] are a convenient way of formalizing this probability:

Definition 2.2.1. A function f is *negligible* if for every polynomial $p(x)$ there exists a N such that for all integers $n > N$, $f(n) < \frac{1}{p(n)}$. A negligible function can be denoted $negl$.

Given a cyclic group G and elements $g, y \in G$, with $y \in \langle g \rangle$, recall that the *discrete logarithm problem* is to find an integer x such that $g^x = y$. Using the concept of semantic security, we can now formally define a hardness assumption based upon the discrete logarithm:

Definition 2.2.2. Let $\mathcal{A}(G, g, y)$ be any probabilistic, polynomial time algorithm $\mathcal{A}$ that, for a specified cyclic group G with elements g and y, outputs 1 if an x is found such that $g^x = y$ and 0 otherwise. The *discrete logarithm assumption* is that there exists a G such that, for any probabilistic, polynomial time algorithm $\mathcal{A}$, the following holds:

$$Pr[\mathcal{A}(G, g, y) = 1] \leq negl(n),$$

with $n = |G|$ being the order of the cyclic group.

There is no known efficient algorithm for computing discrete logarithms for arbitrary groups on conventional (i.e., non-quantum) computers. Exhaustive search for a discrete logarithm takes $O(n)$ time, where n is the group order. For scale, note that the order of a group $\mathbb{Z}_p^*$ based on the common key size of 2048-bit primes is approximately 10^{617}. The *baby-step, giant-step* algorithm by Shanks is currently the most efficient for arbitrary groups, at $O(\sqrt{n} \cdot \text{polylog}(n))$ [29, pg. 306]. For groups of the form $\mathbb{Z}_p^*$ with p prime, the *general number field sieve* is the most efficient at $2^{O(n^{1/3} \cdot (\log n)^{2/3})}$ [29, pg. 307].

With a quantum computer of sufficient size, Shor's algorithm [49] can solve the discrete logarithm problem efficiently. The algorithm runs in bounded-error quantum polynomial

time (BQP), that is, it runs in polynomial time and the probability that the algorithm produces an incorrect answer can be made arbitrarily small.

2.2.2 The Anshel-Anshel-Goldfeld Key-Exchange Protocol

In their 1999 paper [2], Anshel, Anshel, and Goldfeld introduced the *commutator key exchange protocol*, which is also referred to as AAG key exchange or Arithmetica. The group-based version of the key exchange described below is in the style of [38]. Prior to the key exchange, the protocol parameters $N_1, N_2, L_1, L_2, L \in \mathbb{N}$, with $1 \leq L_1 \leq L_2$, are chosen and made public:

1. Alice chooses a set $\bar{A} = \{a_1, \ldots, a_{N_1}\}$, with Bob choosing $\bar{B} = \{b_1, \ldots, b_{N_2}\}$, where $a_i, b_j \in G$ are words of length in $[L_1, L_2]$. Note that $\bar{A}$ and $\bar{B}$ both generate subgroups of G. These sets are then exchanged publicly with each other.

2. Alice constructs her private key as $A = a_{s_1}^{\epsilon_1} \ldots a_{s_L}^{\epsilon_L}$, with $a_{s_k} \in \bar{A}$ and $\epsilon_k \in \{-1, 1\}$. Similarly, Bob computes as his private key $B = b_{t_1}^{\delta_1} \ldots b_{t_L}^{\delta_L}$, with $b_{t_k} \in \bar{B}$ and $\delta_k \in \{-1, 1\}$.

3. Alice then computes $b'_j = A^{-1} b_j A$ for $1 \leq j \leq N_2$ and sends this collection to Bob, while Bob computes and sends Alice $a'_i = B^{-1} a_i B$ for $1 \leq i \leq N_1$.

4. Alice and Bob can now compute a shared key $\kappa = A^{-1}B^{-1}AB$, which is the *commutator* of A and B, denoted $[A, B]$. Alice computes (using only the a'_i which correspond to some s_i of her private key):

$$\begin{aligned}\kappa_A &= A^{-1}a'^{\epsilon_1}_{s_1}\cdots a'^{\epsilon_L}_{s_L} \\ &= A^{-1}B^{-1}a^{\epsilon_1}_{s_1}B\cdots B^{-1}a^{\epsilon_L}_{s_L}B \\ &= A^{-1}B^{-1}a^{\epsilon_1}_{s_1}(BB^{-1})a^{\epsilon_2}_{s_2}B\cdots B^{-1}a^{\epsilon_{L-1}}_{s_{L-1}}(BB^{-1})a^{\epsilon_L}_{s_L}B \\ &= A^{-1}B^{-1}a^{\epsilon_1}_{s_1}a^{\epsilon_2}_{s_2}\cdots a^{\epsilon_{L-1}}_{s_{L-1}}a^{\epsilon_L}_{s_L}B \\ &= A^{-1}B^{-1}AB.\end{aligned}$$

Analogously, Bob computes $\kappa_B = B^{-1}A^{-1}BA$. The shared secret is then $\kappa = \kappa_A = \kappa_B^{-1}$.

As noted in [51], the security of AAG is based on both the simultaneous conjugacy search problem and the subgroup membership search problem.

2.2.3 Length-based Attack and Conjugacy Search

The length-based attack (LBA) is an incomplete, local search that attempts to solve the conjugacy search problem (or its generalized version) by using the length of a word as a heuristic. It was first introduced by Hughes and Tannenbaum [26] as a means to attack the AAG key exchange protocol over braid groups. In [17], Garber et al. explored the use of length functions based on the Garside normal form of braid group elements. They demonstrated experimentally that the length-based attack in this context could break the AAG protocol, albeit inefficiently.

As the length-based attack is an iterative improvement search, it is susceptible to failing at peaks and plateaux in the search space. In [38], Myasnikov and Ushakov identified where these peaks occurred and were able to make successive refinements to the algorithm to yield a high success rate. More recently, the authors of [16] analyzed the LBA against AAG over polycyclic groups, finding that the success rate of the LBA decreased as the Hirsch length of the platform group increased.

Any version of the LBA algorithm can be readily adapted to solve the single conjugacy search problem in a finitely presented group. Such algorithms will be referred to as *length-based conjugacy search*. In what follows we provide the pseudocode for the "LBA with Memory 2" from [16], the most effective algorithm from their paper, adapted to solving the single conjugacy problem. In this variation, one maintains a set S full of conjugates of our initial element, y. Each element of S is conjugated by each generator and the results are stored in a set S'. After every element of S has been conjugated by every generator, the T smallest elements of S are kept, with the rest discarded. The algorithm terminates when the problem has been solved or after a user-specified time-out. We assume that the group G has a length function, $|\cdot|$, such that $|g| < |xgx^{-1}|$, and that the set S generates G. As input we take $x, y \in G$ such that $|y| > |x|$ and X such that $\langle X \rangle = G$. For convenience, we assume that X is closed under the inversion of elements:

Algorithm 1 LBCS with Memory 2 (Single Conjugacy Problem)

$S \leftarrow \{(|y|, y, \mathrm{id}_G)\}$
while not time-out **do**
  for $(|z|, z, a) \in S$ **do**
    Remove $(|z|, z, a)$
    for $h \in X$, $e = \pm 1$ **do**
      $g \leftarrow h^e$
      if $gzg^{-1} = x$ **then**
        Return ga as a conjugator of x to y
      else
        Save $(|gzg^{-1}|, gzg^{-1}, ga)$ in the set S'
      end if
    end for
  end for
  Sort potential conjugators in S' by their length
  Copy the shortest T elements into S and delete the rest of S'
end while
Upon time-out, return FAIL

Chapter 3

The Conjugacy Search Problem in a Family of Polycyclic and Metabelian Groups

The exposition in this section is a condensed and annotated version of [21], a joint work.

3.1 Split Metabelian Groups of Finite Prüfer Rank

We consider the conjugacy search problem for a certain family $\mathcal{F}$ of finitely presented metabelian groups given by the following presentation:

$$G = \langle q_1, \ldots, q_n, b_1, \ldots, b_s \mid [q_l, q_t] = 1, [b_i, b_j] = 1, b_i^{q_l} = b_1^{m_{l(1,i)}} b_2^{m_{l(2,i)}} \ldots b_s^{m_{l(s,i)}} \rangle$$

with $1 \leq l, t \leq n$, $1 \leq i, j \leq s$ and the $m_{l(j,i)}$ suitable integers so that the actions of the q_l commute.

Observe that $q_1, \ldots, q_n$ generate a free abelian group which we denote by Q and that $b_1, \ldots, b_s$ and their Q-conjugated elements generate a torsion-free abelian group B such that $G = B \rtimes Q$, with B a normal subgroup of G. Throughout this chapter we will consider B as a Q-module with left action and will denote conjugation as $b_i^{q_l} = q_l b_i q_l^{-1}$.

As B is torsion-free abelian there is an embedding $B \hookrightarrow \mathbb{Q}^s$ that maps $b_1, \ldots, b_s$ to a free basis of $\mathbb{Q}^s$. This means that the group G has finite Prüfer rank $n + s$. Recall that a group has finite Prüfer rank if the number of generators needed to generate any finitely generated subgroup is bounded. Observe that the action of Q on B can be described using integral matrices: the action of q_l is encoded by the $s \times s$ matrix M_l with entries $m_{l(j,i)}$. As these matrices commute pairwise, Q maps onto an abelian subgroup of $\mathrm{GL}(s, \mathbb{Q})$. Our group G need not be polycyclic: in fact, it is polycyclic if and only if the matrices M_l have integral inverses [3].

Elements of these groups exhibit the following normal form:

$$q_1^{-\alpha_1} \ldots q_n^{-\alpha_n} b_1^{\beta_1} \ldots b_s^{\beta_s} q_1^{\gamma_1} \ldots q_n^{\gamma_n},$$

with $\alpha_l, \beta_i, \gamma_l \in \mathbb{Z}$ and $\alpha_1, \ldots, \alpha_n \geq 0$. Collection from the left [33] can be employed to convert any word written in the generators of G into its unique normal form.

3.1.1 Generalized Metabelian Baumslag-Solitar Groups

Of particular interest is the subfamily of $\mathcal{F}$ that we call *generalized metabelian Baumslag-Solitar groups*. Let $m_1, \ldots, m_n$ be positive integers. These groups are given by the following presentation:

$$G = \langle q_1, \ldots, q_n, b \mid b^{q_i} = b^{m_i},\ 1 \leq i, j \leq n, [q_i, q_j] = 1 \rangle.$$

They are constructible metabelian groups of finite Prüfer rank and G is isomorphic to $B \rtimes Q$ with $Q = \langle q_1, \ldots, q_n \rangle \cong \mathbb{Z}^n$ and $B = \mathbb{Z}[m_1^{\pm 1}, \ldots, m_k^{\pm 1}]$ (as additive groups).

Example 3.1.1. The group GMBS(2,3) given by the presentation

$$\mathrm{GMBS}(2,3) = \langle q_1, q_2, b \mid b^{q_1} = b^2, b^{q_2} = b^3, [q_1, q_2] = 1 \rangle$$

is a generalized metabelian Baumslag-Solitar group. Note that GMBS(2,3)$\cong \mathbb{Z}\left[\frac{1}{2}, \frac{1}{3}\right] \rtimes \mathbb{Z}^2$.

Let us examine how collection works for these groups. Consider an uncollected word in GMBS(2,3):

$$q_1^{-1} q_2 b^{-1} q_1 q_2^{-1}.$$

As the elements q_i commute we have

$$q_1^{-1} q_2 b^{-1} q_2^{-1} q_1.$$

We then apply the negated form of the relation $b^{q_2} = b^3$ to yield

$$q_1^{-1} q_2 q_2^{-1} b^{-3} q_1,$$

that, after cancellation, gives us the reduced word in normal form:

$$q_1^{-1} b^{-3} q_1.$$

3.1.2 Galois Extensions

Let $L : \mathbb{Q}$ be a Galois extension of degree n and fix an integral basis $\{u_1, \ldots, u_s\}$ of L over $\mathbb{Q}$, then $\{u_1, \ldots, u_s\}$ freely generates the maximal order $\mathcal{O}_L$ as a $\mathbb{Z}$-module. Now, choose integral elements, $q_1, \ldots, q_n$, that generate a free abelian multiplicative subgroup of $L - \{0\}$.

Each q_l acts on L by left multiplication and using the basis $\{u_1, \ldots, u_s\}$, we may represent this action by means of an integral matrix M_l. Let B be the smallest sub $\mathbb{Z}$-module of L closed under multiplication with the elements $q_l^{\pm 1}$ such that $\mathcal{O}_L \subseteq B$, i.e.,

$$B = \mathcal{O}_L[q_1^{\pm 1}, \ldots, q_n^{\pm 1}].$$

We may then define $G = B \rtimes Q$, where the action of Q on B is given by multiplication by the elements q_l. The generalized metabelian Baumslag-Solitar groups of the previous example are a particular case of this situation when $L = \mathbb{Q}$. If the elements q_l lie in $\mathcal{O}_L^\times$, the group of units of $\mathcal{O}_L$, then the group G is polycyclic.

3.1.3 Linear Representations

Recall that each $q_l \in Q$ can be represented linearly as a matrix $M_l \in \mathbb{Q}^{s \times s}$. As $B \subset \mathbb{Q}^s$, elements of G can be converted from words to linear representations, where a word $g = bx \in G$ is mapped to a vector $vX \in \mathbb{Q}^s$, with X a product of the matrices M_l. If g is in normal form, i.e.,

$$q_1^{-\alpha_1} \ldots q_n^{-\alpha_n} b_1^{\beta_1} \ldots b_s^{\beta_s} q_1^{\gamma_1} \ldots q_n^{\gamma_n},$$

then the following word also yields g:

$$q_1^{-\alpha_1} \ldots q_n^{-\alpha_n} b_1^{\beta_1} \ldots b_s^{\beta_s} q_1^{\alpha_1} \ldots q_n^{\alpha_n} q_1^{\gamma_1 - \alpha_1} \ldots q_n^{\gamma_n - \alpha_n}.$$

In the semidirect representation of we have $g = bx$ with $x = q_1^{\gamma_1 - \alpha_1} \ldots q_n^{\gamma_n - \alpha_n}$ and additively

$$b = (q_1^{-\alpha_1} \ldots q_n^{-\alpha_n}) \cdot (\beta_1 b_1 + \ldots + \beta_s b_s).$$

To represent b as a vector $v \in \mathbb{Q}^s$, recall that the action of each q_l is encoded by the integral matrix M_l, then

$$v = M_1^{-\alpha_1} \cdots M_n^{-\alpha_n} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_s \end{bmatrix}.$$

The complexity of the above procedure using Gaussian elimination for inverses, standard matrix multiplication, and efficient exponentiation is:

$$O((n-1)[s^3 + s^3 \log \max_l(\alpha_l) + s^3 \log \max_l(\gamma_l - \alpha_l)] + s^2 + s^3).$$

Thus, we have shown that we can, in polynomial time, convert from a word representation to a linear one. Now, consider the converse, in which we have vx with v given as a vector in $\mathbb{Q}^s$. In order to convert v into its normal form, we first show that B is embedded in a particular subset of $\mathbb{Q}^s$. Testing for membership in this subset will then yield an element $b \in B$ in normal form as desired. In the following discussion, we identify B with its image in $\mathbb{Q}^s$ and the group generated by $b_1 \ldots, b_s$ with $\mathbb{Z}^s$.

Let $d = \prod_l d_l$, with d_l is the lowest common denominator of the entries of M_l. Observe that for any $v \in B$,

$$d^{\alpha_1 + \ldots + \alpha_n} v \in \mathbb{Z}^s,$$

thus $v \in \mathbb{Z}[\frac{1}{d}]^s$. In other words, we have

$$B \subseteq \mathbb{Z}[\tfrac{1}{d}]^s \subset \mathbb{Q}^s.$$

Lemma 3.1.1. *Let $M = \prod_l M_l$. There is some α depending on G only such that for any i, $B \cap \frac{1}{d^i}\mathbb{Z}^s \subseteq M^{-i\alpha}\mathbb{Z}^s$. Moreover, $\alpha \leq s \log d$.*

Proof. Consider first the case when $i = 1$. We have $\mathbb{Z}^s \subseteq \frac{1}{d}\mathbb{Z}^s$ and

$$\mathbb{Z}^s \subseteq M^{-1}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s \subseteq \ldots \subseteq M^{-j}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s \subseteq M^{-j-1}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s \subseteq \ldots \subseteq \frac{1}{d}\mathbb{Z}^s.$$

As the quotient $\frac{1}{d}\mathbb{Z}^s/\mathbb{Z}^s$ is the finite group $\mathbb{Z}_d \times \ldots \times \mathbb{Z}_d$ of order d^s, this sequence stabilizes at some degree, say α. Then $B \cap \frac{1}{d}\mathbb{Z}^s = M^{-\alpha}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s$ and

$$B \cap \frac{1}{d}\mathbb{Z}^s \subseteq M^{-\alpha}\mathbb{Z}^s$$

as desired. Moreover, we claim that it stabilizes precisely at the first α such that

$$M^{-\alpha}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s = M^{-\alpha-1}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s.$$

To demonstrate, let $b \in M^{-\alpha-2}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s$. Then $Mb \in M^{-\alpha-1}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s = M^{-\alpha}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s$ thus $b \in M^{-\alpha-1}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s = M^{-\alpha}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s$. Repeating the argument implies that for all $\beta > \alpha$,

$$M^{-\alpha}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s = M^{-\beta}\mathbb{Z}^s \cap \frac{1}{d}\mathbb{Z}^s.$$

As a consequence, α is bounded by the length of the longest chain of proper subgroups in $\mathbb{Z}_d \times \ldots \times \mathbb{Z}_d$, i.e., $\alpha \leq \log(d^s) = s \log d$.

Now we argue by induction. Let $b \in B \cap \frac{1}{d^i}\mathbb{Z}^s$, then $db \in B \cap \frac{1}{d^{i-1}}\mathbb{Z}^s$ and by induction we may assume that $db \in M^{-(i-1)\alpha}\mathbb{Z}^s$, thus $M^{(i-1)\alpha}db = v \in \mathbb{Z}^s$. Then

$$\frac{1}{d}v \in B \cap \frac{1}{d}\mathbb{Z}^s \subseteq M^{-\alpha}\mathbb{Z}^s.$$

Therefore

$$M^{\alpha}M^{(i-1)\alpha}b = \frac{1}{d}M^{i\alpha}v \in \mathbb{Z}^s$$

and $b \in M^{-i\alpha}\mathbb{Z}^s$. □

To glimpse how the above lemma can be used, consider the group $G \in \mathcal{F}$ given by the following presentation:

$$G = \langle b_i, q_i \mid b_1^{q_1} = b_1^2, b_2^{q_2} = b_2^4, b_3^{q_3} = b_3^{16}, b_i^{q_j} = b_i \text{ for } i \neq j, [b_i, b_j] = 1, [q_i, q_j] = 1\rangle,$$

with $1 \leq i, j \leq 3$.

From the presentation above $s = 3$. The linear representations of the elements q_l (and their product M) are then:

$$M_1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix} M_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 16 \end{bmatrix}; M = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 16 \end{bmatrix}.$$

From visual inspection of M it is clear that $d = 16$. Moreover, it is easy to check that $\frac{1}{16}\mathbb{Z}^s \subseteq B$ and that

$$\frac{1}{16}\mathbb{Z}^s = \frac{1}{16}\mathbb{Z}^s \cap B \subseteq M^{-4}\mathbb{Z}^s,$$

with $\alpha = 4$ the smallest value that satisfies the above equation.

There are two cases to considered in determining the word representation of v. If $v \in \mathbb{Z}^s$, then $v \in B$ and the coordinates of v are the coefficients of the elements b_i. However, if $v \in \frac{1}{d^i}\mathbb{Z}^s$, then from Lemma 3.1.1 there is some $i > 0$ such that $d^i v$ is integral. To determine this, we only have to check whether the vector

$$[(M_1 M_2 \dots M_n)]^{is\lfloor \log d \rfloor} \equiv M^{i\alpha}$$

is integral, which can be achieved within the complexity bounds of the following lemma:

Lemma 3.1.2. *Let $v \in \mathbb{Z}[\frac{1}{d}]^s$ and i the smallest possible integer such that $d^i v$ is integral.*

Then $v \in B$ if and only if

$$M^{is\lfloor \log d \rfloor} v \in \mathbb{Z}^s$$

where $M = M_1 M_2 \dots M_n$. The complexity of this computation is polynomial, specifically $O((n-1)s^3 \log is\lfloor \log d \rfloor)$. (Alternatively, the same result holds true for α instead of $s\lfloor \log d \rfloor$.)

Proof. Lemma 3.1.1 implies that $v \in B$ if and only if $M^{i\alpha} v$ is integral. Thus if $v \in B$,

$$M^{is\lfloor \log d \rfloor} v = M^{(is\lfloor \log d \rfloor - i\alpha)} M^{i\alpha} v$$

is integral because $is\lfloor \log d \rfloor - i\alpha \geq 0$. The converse is obvious.

Regarding the time complexity, we have to compute the $(is\lfloor \log d \rfloor v)$-th power of the matrix M. The complexity estimation is obtained using standard matrix multiplication and efficient exponentiation. □

Note that the exponent $is\lfloor \log d \rfloor$ is merely an upper bound; often a much smaller value suffices to obtain an expression of a given $v \in B$ as a product of the conjugated b_i's. Consider the example group from Lemma 3.1.1 above and the following vector $v \in \mathbb{Q}^3$:

$$v = \left[\frac{1}{32}, \frac{3}{64}, \frac{5}{16} \right].$$

Here, $i = 2$, $s = 3$, and $d = 16$, thus $is\lfloor \log d \rfloor = 24$, but note that $M^5 v$ is already integral.

3.1.4 Solving Linear Systems

In developing our algorithm to solve the conjugacy search problem in $\mathcal{F}$ we will need the ability to solve linear systems. Let N be an integral $s \times s$ matrix that is some product of the matrices M_l and $u \in \mathbb{Q}^s$ an integral column vector.

We wish to determine if the linear system

$$NX = u \tag{3.1}$$

has some solution $v \in \mathbb{Q}^s$ that lies in $B \subseteq \mathbb{Z}[\frac{1}{d}]^s$. To solve this problem, we will first use a standard technique to solve this type of system in $\mathbb{Z}$. The *Smith normal form* of a matrix A is a diagonal matrix D with entries $k_1, \ldots, k_r, 0, \ldots, 0$, such that k_j divides k_{j+1} for $j \in [1, r)$, with r being the rank of A. Moreover, there are invertible matrices P and Q in $\mathrm{SL}(s, \mathbb{Z})$ such that $D = QAP$.

Let $a = \max\{|a_{ij}| \mid a_{ij} \text{ entry of } N\}$, then we have the following bound on the product of the diagonal entries $k_1, \ldots, k_r$:

Lemma 3.1.3.

$$\prod_{j=1}^{r} k_j \leq \sqrt{s} a^s$$

Proof. It is well known that the product $k_1 \cdots k_r$ is the greatest common divisor of the determinants of the nonsingular $r \times r$ minors of the matrix N. Let N_1 be one of those minors, then

$$k_r \leq k_1 \cdots k_r \leq |\det N_1|.$$

Now, the determinant of the matrix N_1 is bounded by the product of the norms of the columns $c_1, \ldots, c_r$ of the matrix (this bound is due to Hadamard, see for example [25]) so we have

$$|\det N_1| \leq \prod_{j=1}^{r} \|c_j\| \leq (\sqrt{r} a)^r \leq \sqrt{s} a^s.$$

□

Let $D = PNQ$ be the Smith normal form of N, with P and Q as above. If r is the rank of N, then the columns of D can be rearranged such that the first $(s-r)\times(s-r)$ diagonal block is zero, and that the kernel of N, Ker$N$, is generated by the first $s-r$ columns of P. Let D_2 be the last $r \times r$ diagonal block. Our system (3.1) can then be transformed into

$$D\tilde{X} = \begin{bmatrix} 0 & 0 \\ 0 & D_2 \end{bmatrix} \tilde{X} = Qu \tag{3.2}$$

with $\tilde{X} = P^{-1}X$. At this point, we see that this reduced system has some solution if and only if the first $s-r$ entries of Qu vanish.

Assume that this is the case and let v_2 be the unique solution to the system

$$D_2\tilde{X}_2 = (Qu)_2 \tag{3.3}$$

where the subscript 2 in $\tilde{X}$ and Qu means that we take the last r coordinates only. Then

$$v_2 = D_2^{-1}(Qu)_2.$$

The set of all the rational solutions to (3.1) is

$$\left\{ P \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \mid v_1 \in \mathbb{Q}^{s-r} \right\}.$$

Equivalently, this set can be written as

$$v + \text{Ker}N \text{ where } v = P \begin{bmatrix} 0 \\ v_2 \end{bmatrix}.$$

As P is invertible, it has full rank, thus yielding a basis for $\mathbb{Z}^s$. Each M_l can be rewritten in this new basis as $P^{-1}M_lP$. The fact that N commutes with each M_l implies that M_l leaves Ker$N$ (set-wise) invariant. By construction, Ker$N$ is generated by the first $s-r$ columns of P and therefore each $P^{-1}M_lP$ has the following upper triangular block form:

$$P^{-1}M_lP = \begin{bmatrix} A_l & B_l \\ 0 & C_l \end{bmatrix}.$$

Moreover, C_l is just the $r \times r$ matrix associated with the action of q_l in the quotient $\mathbb{Q}^s/\mathrm{Ker}N$, written in the new basis obtained from the last r columns of P.

Proposition 3.1.1. *A solution to the system (3.2) exists in B if and only if $v_2 \in \mathbb{Z}[\frac{1}{d}]^r$ and*

$$C^{ir\lfloor \log d \rfloor}v_2 \in \mathbb{Z}^r,$$

with $C = \prod_l C_l$ and i the smallest possible integer such that d^iv_2 is integral. (We can use s instead of r.)

Proof. Assume first that $C^{ir\lfloor \log d \rfloor}v_2 \in \mathbb{Z}^r$, with i as above. We have

$$P^{-1}M^{i\alpha}P = \begin{bmatrix} A & S \\ 0 & C^{ir\lfloor \log d \rfloor} \end{bmatrix}$$

for a certain $(s-r) \times r$ matrix S and a certain $(s-r) \times (s-r)$ invertible matrix A, with $M = \prod_l M_l$ as before. Therefore

$$P^{-1}M^{ir\lfloor \log d \rfloor}P\tilde{X} = \begin{bmatrix} A & S \\ 0 & C^{ir\lfloor \log d \rfloor} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} Av_1 + Sv_2 \\ C^{ir\lfloor \log d \rfloor}v_2 \end{bmatrix}.$$

This means that we only have to find a $v_1 \in \mathbb{Q}^{s-r}$ such that $Av_1 + Sv_2 \in \mathbb{Z}^s$. To do so, observe that it suffices to take $v_1 = -A^{-1}Sv_2'$.

Conversely, assume that some $P \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ lies in B. Then some product of positive powers of the matrices M_l transforms $P \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ into an integral vector, thus there is a product of the blocks C_l that transforms v_2 into an integral vector. We may use now Lemma 3.1.2 applied to $\mathbb{Q}^r = \mathbb{Q}^s/\mathrm{Ker}N$ with respect to the action of the blocks C_l to conclude that $v_2 \in \mathbb{Z}[\frac{1}{d}]^r$ and

$$C^{ir\lfloor \log d \rfloor}v_2 \in \mathbb{Z}^r,$$

with i the smallest possible integer such that $d^i v_2$ is integral. (Note that dC_l^{-1} is integral so we can use the same d for this quotient as for the original group.)

□

Remark 3.1.1. Observe that, as N is integral, a necessary condition for the system (3.1) to have some solution in B is for u to lie in $\mathbb{Z}[\frac{1}{s}]$. Let i_0 be such that $d^{i_0}u$ is integral. Then $d^{i_0}\det(D_2)v_2$ is also integral. If this lies in $\mathbb{Z}[\frac{1}{d}]^s$, it means that for some i_1 such that $d^{i_1} \leq \det(D_2)$, we have that $d^{i_0+i_1}v_2$ is integral. By Lemma 3.1.3, $\det(D_2) \leq \sqrt{s}a^s$, thus $i_1 \leq \sqrt{s}a^s$. As a consequence, if i is as in Proposition 3.1.1, we have

$$i \leq i_0 + \sqrt{s}a^s.$$

We now have an algorithm to solve this system:

1. Compute the Smith normal form of N.
2. Compute v_2 from the product of D_2^{-1} and $(Qu)_2$.
3. Find a $v_1 \in \mathbb{Q}^{s-r}$ such that $Av_1 + Sv_2 \in \mathbb{Z}^s$. This can be computed directly: $v_1 = -A^{-1}Sv_2'$.

The complexity of the algorithm is stated formally in the following proposition:

Proposition 3.1.2. *There is an algorithm to decide whether the system (3.1) has some solution in B and to compute that solution. The complexity of this algorithm is polynomial, specifically*

$$O(s^6 \log sa + (s-r)^5 + (s-r)^3 + (n-1)[s^3 \log is \log d + 1] + r^3).$$

where $i \leq i_0 + \sqrt{s}a^s$ and i_0 is such that $d^{i_0}u$ is integral and a is the maximum absolute value of the entries of N.

Proof. The algorithm has been described above. In summary, we have to transform the original system using the Smith normal form for N, compute v_2 and the matrices C_l and $C = C_1 \dots C_n$, and then check whether v_2 lies in $\mathbb{Z}[\frac{1}{d}]$. If it does, we may either compute i such that $d^i v_2$ is integral or estimate i as $i_0 + i_1$ (see Remark 3.1.1). We then compute

$$C^{ir\lfloor \log d \rfloor} v_2$$

and check whether it is integral or not.

For a proof of this fact see [27] in the non-singular case and [55] for the singular one. Once we have the Smith normal form, to compute v_2 we only have to perform the product of D_2^{-1} and $(Qu)_2$, which is $O(r^3)$. Next, we have to compute the matrices C_l, which requires $n-1$ matrix multiplications, taking time $O((n-1)s^3)$. We then check whether $C^{ir\lfloor \log d \rfloor} v_2$ is integral which takes at most $O((n-1)s^3 \log is \log d)$ time. Solving for v_2 and v_1' via Gaussian elimination takes $O(r^3)$ and $O((s-r)^3)$, respectively, and calculating v_1 is $O((s-r)^5$.

The overall time complexity is then the sum of of the above operations and is denoted in the proposition. Note that the lower order terms involving s and r are dominated by the complexity of calculating the Smith normal form. □

3.2 The Complexity of the Conjugacy Search Problem in $\mathcal{F}$

Given $g, g_1 \in G$ and $g \sim g_1$, the conjugacy search problem is to find $h \in G$ such that $g^h = g_1$. Let $g = bx$, $g_1 = b_1 x$ and $h = cy$, with $b, c \in B$ and $x, y \in Q$, then:

$$\begin{aligned} g^h &= hgh^{-1} \\ &= cybxy^{-1}c^{-1} \\ &= cyby^{-1}xc^{-1} \quad (Q \text{ is abelian}) \\ &= cb^y(c^{-1})^x x \end{aligned}$$

The component $cb^y(c^{-1})^x$ belongs to the abelian group B. We can write it additively as:

$$\begin{aligned} & cb^y(c^{-1})^x \\ &= c(c^{-1})^x b^y && (B \text{ is abelian}) \\ &= cc^{-x}b^y && (\text{definition of semidirect product}) \\ &= c - x \cdot c + y \cdot b && (\cdot \text{ denotes the action}) \\ &= y \cdot b + (1-x) \cdot c. && (\text{B is a } \mathbb{Z}Q\text{-module}) \end{aligned}$$

Consequently, the conjugacy search problem above is equivalent to the problem of finding $c \in B$, $y \in Q$ such that $b_1 = y \cdot b + (1-x) \cdot c$.

The equation above can be further reduced. Notice that the subgroup $(1-x) \cdot B$ is invariant under any action $y \in Q$:

$$\begin{aligned} y \cdot (1-x) \cdot B &= (y(1-x)) \cdot B && (\text{compatibility}) \\ &= ((1-x)y) \cdot B && (Q \text{ is abelian}) \\ &= (1-x) \cdot b', \end{aligned}$$

with $b' = y \cdot b \in B$. Thus, Q acts on the quotient group $\bar{B} = B/(1-x) \cdot B$. Let $\bar{b}$ be the coset in $\bar{B}$ associated with the element b. The conjugacy equation is then:

$$\bar{b}_1 = y \cdot \bar{b}.$$

In solving the modified conjugacy search problem above, we will need to utilize the linear representation of $\bar{B}$. Let M_x be the rational matrix associated with the action of x on B (with respect to the set $b_1, \ldots, b_s$), $N = I - M_x$, and let NB to denote $(1-x) \cdot B$. Using this notation, $\bar{B} = B/NB$. Let T be the torsion subgroup of $\bar{B}$. Recall that the *torsion subgroup* is the subgroup of an abelian group that contains elements of finite order. T is also

Q-invariant, so Q acts on $\bar{B}/T$. As $\bar{B}/T$ is torsion-free and of finite Prüfer rank, it can be embedded in $\mathbb{Q}^s/N\mathbb{Q}^s$.

3.2.1 An Algorithm for the Conjugacy Search Problem in $\mathcal{F}$

The idea of the algorithm is to decompose the problem of finding the original conjugator h into two problems - one is an instance of the multiple orbit problem in a vector space, while the other is a type of discrete logarithm problem. For the first we take advantage of the polynomial time solution of Babai et al. [4]. For the latter, we provide an upper bound for its complexity that is essentially dependent upon the size of T, the torsion subgroup of $\bar{B}$.

The algorithm proceeds as follows:

1. Construct the quotient group $\bar{B}/T$, where $\bar{B} = (1-x) \cdot B < B$ and T is the torsion subgroup of $\bar{B}$. We then consider the projections $\bar{b}+T$ and $\bar{b}_1+T$ of b and b_1 in $\bar{B}/T$, where $\bar{b}$ are the cosets. Using the algorithm from [4], we solve the multiple torsion-free orbit problem

$$y \cdot (\bar{b}+T) = \bar{b}_1 + T$$

with $y \in Q$. In fact, this algorithm finds the full lattice of solutions (i.e., set of linear combinations):

$$\Lambda = \{q \in Q \mid q \cdot \bar{b} - \bar{b_1} \in T\},$$

2. For some fixed $h \in \Lambda$, we compute a basis $y_1, \ldots, y_m$ of $Q_1 \leq Q$ where

$$Q_1 = \{h^{-1}q \mid q \in \Lambda\}.$$

Order the elements of Q_1 with respect to their word length and for each $q \in Q_1$ deter-

mine whether $q \cdot b - b_1 \in NB$ (the linear representation of T).

More precisely, we must determine whether the linear system

$$u = NX$$

with $u = q \cdot b - b_1$ has some solution c in B; this can be done by using the algorithm in Section 3.1.4. The number of linear systems that must be checked is dependent upon $t = |T|$, the order of T.

To verify that the algorithm above does indeed solve the modified conjugacy search problem, and that it, in fact, halts, we must show respectively that the lattice generated by the multiple orbit algorithm generates all possible coset representatives (i.e., solutions), and that $t = |T|$ is finite. The lemma below ensures that we can generate all coset representatives:

Lemma 3.2.1. *Let $Q_2 \leq Q_1$ with Q_1 free abelian with generators $x_1, \ldots, x_m$, and assume that the group Q_1/Q_2 is finite of order t. Then the set*

$$\Omega = \{x_1^{\alpha_1} \ldots x_m^{\alpha_m} \mid \sum_{j=1}^{m} |\alpha_j| < t\}$$

has order bounded by $(2t)^m$ and contains a full set of representatives of the cosets of Q_2 in Q_1.

Proof. Let $v_1, \ldots, v_m$ be generators of the subgroup Q_2, which can be viewed as points in $\mathbb{Z}^m$. Consider the parallelogram

$$P = \{t_1 v_1 + \ldots + t_m v_m \mid t_j \in \mathbb{R}, 0 \leq t_j < 1\}.$$

Then $\mathbb{Z}^m \cap P$ is a set of representatives of the cosets of Q_2 in Q_1 and we claim that $P \subseteq \Omega$.

Observe that for any point $p = (\alpha_1, \ldots, \alpha_m)$ in $\mathbb{Z}^m \cap P$ there is a path in $\mathbb{Z}^m \cap P$ from $(0, \ldots, 0)$ to p. We may assume that the path is simple and therefore its length is bounded by t. On the other hand, the length of the path is greater than or equal to $\sum_{j=1}^{m} |\alpha_j|$ thus

$$\sum_{j=1}^{m} |\alpha_j| \leq t.$$

□

3.2.2 On the Subgroup T

In determining the order of T, we will first need two results concerning the order of torsion subgroups in general. If T' is some torsion subgroup of finite Prüfer rank s, then $|T'|$ is bounded by k^s, where $k = \exp(T')$, the *exponent* of T', i.e., the smallest natural number such that $kv = 0$ for $v \in T'$. We then need a bound on k, which is provided by the following lemma:

Lemma 3.2.2. *Let N be a square $s \times s$ integer matrix and T' the torsion subgroup of the group $\mathbb{Z}^s/N\mathbb{Z}^s$. Then*

$$exp(T') \leq \sqrt{s}a^s$$

with

$$a = max\{|a_{ij}| \mid a_{ij} \text{ entry of } N\}.$$

Proof. Let $D = \mathrm{diag}(k_1, \ldots, k_r, 0, \ldots, 0)$ be the Smith normal form of N. Then

$$\exp(T) = k_r \leq k_1, \ldots, k_r$$

so it suffices to apply Lemma 3.1.3. □

Finally, we must show that our particular T has finite order, which is confirmed by the

theorem below. Recalling our previous notation, for $1 \leq l \leq n$, let d_l be the smallest positive integer such that $d_l M_l^{-1}$ is an integral matrix and let d be the product of all the integers $d_1, \ldots, d_n$.

Theorem 3.2.1. *Let T be the torsion subgroup of the abelian group $\bar{B} = B/(1-x)\cdot B$. Then T is finite and*

$$|T| \leq \sqrt{s}^{s} d^{\mathcal{L}s^2} (a+1)^{s^2}$$

where $\mathcal{L}$ is the length of the element x as a word in the generators of Q, a is the maximum absolute value of an entry in M_x, the matrix associated with the action of x on B.

Proof. Let $N = I - M_x$. Assume first that M_x is an integral matrix, so the same happens with N. We want to relate the exponent of T with the exponent of the torsion subgroup of $\mathbb{Z}^s/N\mathbb{Z}^s$. Let k be this last exponent and choose $b \in B$ such that $1 \neq \bar{b}$ lies in T. Denote by $m > 0$ the order of $\bar{b}$. Observe that $mb = Nc$ for some $c \in B$ and that m is the smallest possible under these conditions.

Next, choose $q \in Q$ such that $q \cdot b$ and $q \cdot c$ both lie in $\mathbb{Z}^s$. To find such a q it suffices to write b and c multiplicatively using their normal forms and take as q a product of the q_l's with big enough exponents.

Then we have $m(q \cdot b) = q \cdot Nc = N(q \cdot c) \in N\mathbb{Z}^s$ thus $q \cdot b + N\mathbb{Z}^s$ lies in the torsion subgroup of $\mathbb{Z}^s/N\mathbb{Z}^s$. Therefore, $k(q \cdot b) \in N\mathbb{Z}^s$. Now, let m_1 be the greatest common divisor of m and k and observe that the previous equations imply $m_1(q \cdot b) \in N\mathbb{Z}^s$. This means that for some $c_1 \in \mathbb{Z}^s$ we have $m_1(q \cdot b) = Nc_1$, thus

$$m_1 b = q^{-1} N c_1 = N q^{-1} c_1 = N c_2$$

with $c_2 = q^{-1} \cdot c_1 \in B$. By the minimality of m we must have $m \leq m_1$. As m_1 divides both k and m we can conclude $m = m_1 \mid k$. This implies that k is also the exponent of T.

Next, we consider the general case when N could be non-integral. As M_x is the product of $\mathcal{L}$ matrices in the set $\{M_1^{\pm 1}, \ldots, M_n^{\pm 1}\}$ we see that the matrix $d^{\mathcal{L}} M_x$ is integral and therefore so is $d^{\mathcal{L}} N$. Obviously, the group $NB/d^{\mathcal{L}} NB$ is torsion thus

$$\exp(T) \leq \exp(\text{torsion subgroup of } B/d^{\mathcal{L}} NB).$$

The matrix $d^{\mathcal{L}} N$ also commutes with the Q-action so the results above imply that this last exponent equals the exponent of the torsion subgroup of $\mathbb{Z}^s/d^{\mathcal{L}} N\mathbb{Z}^s$. These statements, together with Lemma 3.2.2 and the fact that the largest absolute value of an entry of $d^{\mathcal{L}}$ is bounded by $d^{\mathcal{L}} N$, yield

$$\exp(T) \leq \sqrt{s} d^{\mathcal{L}s} (a+1)^s.$$

Finally, as the group $\bar{B}$ has finite Prüfer rank, so too does T. □

We can now show that bound on the order of T is exponential with respect to the word length $\mathcal{L}$ of x:

Proposition 3.2.1. *With the previous notation, there is a constant K, depending on G only such that for T, the torsion subgroup of $B/NB = (1 - M_x)B$,*

$$|T| \leq K^{\mathcal{L}}$$

where $\mathcal{L}$ is the length of x.

Proof. By Theorem 3.2.1 and the observation above

$$|T| \leq \sqrt{s}^s d^{\mathcal{L}s^2} (a+1)^{s^2} \leq \sqrt{s}^s d^{\mathcal{L}s^2} (s^{\mathcal{L}-1} h^{\mathcal{L}} + 1)^{s^2} \leq (\sqrt{s} dsh + \sqrt{s} d)^{s^2 \mathcal{L}}$$

so we only have to take $K = (\sqrt{s}dsh + \sqrt{s}d)^{s^2}$. □

3.2.3 Complexity Results

Using the results of the previous section, we can now state the following results concerning the conjugacy search problem:

Theorem 3.2.2. *The time complexity of the conjugacy search problem in $\mathcal{F}$ is at most exponential with respect to word length.*

Proof. Recall that we seek to determine h where $g^h = g_1$. Assume that g and g_1 are given as words in normal form. Observe that Step 1 of the algorithm only requires polynomial time. As for Step 2, we have to consider an exponential (in $\mathcal{L}$) number of systems of linear equations of the form

$$u = NX$$

with $u = q \cdot b - b_1$. Moreover, we may find (by writing u in its normal form) some $z \in Q$ such that $z \cdot u$ is in the group generated by $b_1 \ldots, b_s$. If Z is the matrix representing the action of z, this is equivalent to the vector Zu being integral. As Z and N commute our system can be transformed into

$$NZX = Zu.$$

Obviously, X lies in B if and only if ZX does, thus the problem is equivalent to deciding whether

$$d^{\mathcal{L}} NX_1 = d^{\mathcal{L}} Zu$$

has some solution X_1 in B.

Using Proposition 3.1.1 and the complexity computation of Proposition 3.1.2 we see that this can be done in a time that is polynomial in the logarithm of the maximum absolute value of an entry in $d^{\mathcal{L}}N$. Observe that our integrality assumption on Zu implies that the

integer denoted i_0 in Proposition 3.1.2 can be taken to be 0. As the maximum absolute value of an entry in $d^{\mathcal{L}}N$ is exponential with respect to $\mathcal{L}$, this time is polynomial on $\mathcal{L}$. The exponential bound follows, as we must potentially solve a number of linear systems that is exponential with respect to $\mathcal{L}$. □

We also have a second result for groups in $\mathcal{F}$ whose matrix actions are of a particular form:

Theorem 3.2.3. *Fix $s_1, s_2 \geq 0$, with $s = s_1 + s_2$ and assume that for $1 \leq i \leq n$,*

$$M_i \in \{\textit{Matrices } \begin{bmatrix} I_{s_1} & M \\ 0 & I_{s_2} \end{bmatrix} \textit{ with } M \in \textit{Mat}(s_2 \times s_1, \mathbb{Z})\}.$$

Then the time complexity of the conjugacy search problem is polynomial.

Proof. Consider the matrices Γ_{s_1,s_2}, given by

$$\Gamma_{s_1,s_2} := \left\{ \text{Matrices } \begin{bmatrix} I_{s_1} & A \\ 0 & I_{s_2} \end{bmatrix} \right\} \leq SL(s, \mathbb{Z}).$$

As these matrices are invertible in $SL(s, \mathbb{Z})$, we can choose $d = 1$. Assume that for $l = 1, \ldots, n$,

$$M_l \in \Gamma_{s_1,s_2}.$$

Then there is some constant K depending on G only such that for T, the torsion subgroup of $B/NB = (1 - M_x)B$,

$$|T| \leq K\mathcal{L}^{s^2}$$

where $\mathcal{L}$ is the length of x. We consider the bound of Theorem 3.2.1 for $d = 1$ (see above)

$$|T| \leq \sqrt{s}(a+1)^{s^2},$$

where a is the maximum absolute value of an entry in A. Observe that A is a product of matrices in Γ_{s_1,s_2} and that

$$\begin{bmatrix} I_{s_1} & A_1 \\ 0 & I_{s_2} \end{bmatrix} \begin{bmatrix} I_{s_1} & A_2 \\ 0 & I_{s_2} \end{bmatrix} = \begin{bmatrix} I_{s_1} & A_1 + A_2 \\ 0 & I_{s_2} \end{bmatrix}.$$

Therefore, if we let h be the maximum absolute value of an entry in each of the matrices $A_1, \ldots, A_n$, then $a \leq \mathcal{L}h$ and therefore

$$|T| \leq \sqrt{s}(a+1)^{s^2} \leq \sqrt{s}(\mathcal{L}h+1)^{s^2} \leq \sqrt{s}(2\mathcal{L}h)^{s^2}$$

so it suffices to take $K = \sqrt{s}(2h)^{s^2}$.

This result together with the algorithm above (recall that $d = 1$ in this case) produces the desired bound. □

3.3 The Relationship of the Conjugacy Search Problem to the Discrete Logarithm

By using the algebraic machinery constructed in the previous section, we can show that for certain members of $\mathcal{F}$ the conjugacy search problem is an instance of the discrete logarithm problem.

For this section, we restrict ourselves to the subfamily from section 3.1.2, where Q is a multiplicative subgroup of a field L such that $L : \mathbb{Q}$ is a Galois extension and B is the additive group of the subring $\mathcal{O}_L[q_1^{\pm}, \ldots, q_n^{\pm}]$ that is sandwiched between $\mathbb{Q}$ and L. In particular, this means that the only element in Q with an associated matrix having an eigenvalue of 1 is the identity matrix: the eigenvalues of the matrix representing an element $h \in L$ are precisely h itself and its Galois conjugates and thus cannot be 1 if $h \neq 1$. Recall also that this subfamily includes the generalized metabelian Baumslag-Solitar groups of section 3.1.1.

We will keep the notation of the previous section, with elements bx, $b_1x \in G$ such that there is some $cy \in G$ with (additively)

$$b_1 = y \cdot b + (1 - x) \cdot c.$$

We may consider y and $1 - x$ as elements in the field L. From now on we omit the $\cdot$ from our notation and use juxtaposition to denote the action. Now, B also has a ring structure and $(1 - x)B$ is an ideal in B. Moreover, in this case the quotient ring $\bar{B} = B/(1 - x)B$ is finite (because the matrix associated with $1 - x$ is invertible). In this finite quotient ring we wish to solve the equation

$$y\bar{b} = \bar{b}_1.$$

Let $y = q_1^{t_1} \dots q_k^{t_k}$, then solving the discrete log problem in $B/(1 - x)B$ consists of finding $t_1, \dots, t_k$ so that

$$q_1^{t_1} \dots q_k^{t_k} \bar{b} = \bar{b}_1$$

in the finite ring $\bar{B}$.

This is a special type of discrete logarithm problem, as one can observe by recalling what happens when Q is cyclic: $x = q_1^s$ for some s thus we have to solve

$$q_1^{t_1} \bar{v} = \bar{w}$$

in $\bar{B} = B/(1 - q_1^s)B$. To solve it s trials are sufficient (see [9]). In general, as $\bar{h} = 1$ in $\bar{B}$, $q_1^{l_1} \dots q_k^{l_k} = 1$. Assume that we choose $x = q_1$. Then $\bar{q}_1 = 1$ in $\bar{B}$ thus the problem is to find $t_2, \dots, t_k$ such that

$$q_2^{t_2} \dots q_k^{t_k} \bar{b} = \bar{b}_1$$

in $\bar{B}$.

Let us restrict ourselves further to the case of generalized metabelian Baumslag-Solitar groups. We identify the elements q_l with the integers m_l encoding their action. Assume that each m_l is coprime with $1 - m_1$. As before let $y = m_1^{t_1} \dots m_k^{t_k}$ and choose $x = m_1$. Then as each m_l is coprime with $1 - m_1$

$$B/(1-x)B = \mathbb{Z}[m_1^{\pm}, \dots, m_k^{\pm}]/(1-x)\mathbb{Z}[m_1^{\pm}, \dots, m_k^{\pm}] = \mathbb{Z}/(1-x)\mathbb{Z} = \mathbb{Z}_{1-x}.$$

We then have to find $t_2, \dots, t_k$ such that

$$m_2^{t_2} \dots m_k^{t_k} \bar{b} = \bar{b_1}$$

in the ring of integers modulo $1 - m_1$.

If $k = 2$ this is an instance of the ordinary discrete logarithm problem.

Chapter 4

A Machine Learning Approach to Algorithmic Problems in Group Theory

4.1 Related Work

In [23], Haralick et al. posited that pattern recognition techniques are an appropriate methodology for solving problems in combinatorial group theory. To demonstrate, they constructed a machine learning system for discovering effective heuristics for the Whitehead automorphism problem, a search problem in free groups that uses the successive application of the namesake automorphisms to reduce a word to its minimal length.

As mentioned in [23], every machine learning system must contend with the following tasks: data generation, feature extraction, model selection, and evaluation. Once the system is constructed, analysis of the system's performance can yield insights into the nature of the problem at hand, and potentially be used to improve upon it. In this chapter we will

delve into each of these aforementioned tasks, showing in the process how these techniques can be extended from free groups to finitely presented groups. The primary difference in the construction of machine learning systems for free and not-free groups is in feature extraction, which is the focus of the next section.

4.2 Feature Extraction

One of the most important aspects of creating a machine learning system is the process of feature extraction, the means by which relevant information is distilled from the raw dataset and presented to the learning algorithm. If the raw dataset is unstructured, subsets of data may first be aggregated into units of observation, from which the features will be extracted. Some datasets may come with an intrinsic structure, such as that of a list, a matrix, or a string of text. Regardless of the data's inherent structure, the ability to extract features from the underlying data that provide information relevant to the learning process requires domain-specific knowledge.

Finitely presented groups, in addition to their representation as generators and relators, have a combinatorial structure that is manifested by their Cayley graphs. A *Cayley graph* is a rooted, labeled digraph, with a vertex for every word in the group and each edge labeled by a generator or an inverse generator. The root of the graph is the identity element. If the group is infinite then so is its Cayley graph. The graph is connected, and the label of every path from the root to a vertex represents a word in the group. Circuits from the root represent elements that are equivalent to the identity and are therefore in the normal closure of the set of relators.

The Cayley graph also enables groups to be considered as metric spaces. Let G be a finitely generated group with generating set X and $\phi : F(X) \to G$ the canonical epimorphism. If w

is a word over X representing the element $g \in G$, the *geodesic length* of g over X is defined as

$$l_X(g) = \min\{|w| \mid w \in F(X), \phi(w) = g\}.$$

The geodesic length of g corresponds to the shortest path in the Cayley graph whose label corresponds to w. If every edge in the Cayley graph is given unit length, then $l_X(g)$ corresponds to the number of edges in the shortest path labeled by w. We then define the *word length* (or word norm) of w, denoted $|w|$, as $|w| = l_X(g)$. Given words u and v representing elements g and h respectively, we can now define the *word metric* $d_X(u, v) = |u^{-1}v|$ that satisfies the axioms required of a metric function. Note that as the notation implies, $d_X(u, v)$ is dependent upon the choice of the generating set X.

Assigning unit lengths to edges is but one way to extract numerical information from the Cayley graph. In [23], Haralick et al. utilize a weighted variant of the Cayley graph to extract features related to the Whitehead problem. Given a word w and an edge (x, y) in the Cayley graph with label v, the weight of an edge is associated with a *counting function*, $C(w, xvy)$, that counts the number of occurrences of the subword xvy in w. Various counting functions can be employed, and there is a duality between a counting function and a subgraph of the Cayley graph of a word w.

More generally, for a finitely presented group $G = \langle X|R\rangle$, with words $w, v_1, \ldots, v_k \in G$ and subsets $U_1, \ldots, U_{k+1}$, the *counting function* $C(w, U_1v_1U_2v_2 \ldots v_kU_{k_+1})$ counts the number of occurrences of subwords of the form $u_1v_1u_2v_2 \ldots v_ku_{k_+1}$, with $u_1, \ldots, u_{k+1}$ in $U_1, \ldots, U_{k+1}$ respectively. Every sequence of counting functions $C_1, \ldots, C_N$, when normalized by the word length $|w|$, gives rise to a real-valued feature vector $v \in \mathbb{R}^N$:

$$v = \frac{1}{|w|}\langle(C_1(w, \cdot), \ldots, C_N(w, \cdot)\rangle.$$

The origin of the counting functions defined previously can be illuminated by examining group presentations. Recall from section 2.1.4 that a finitely presented group $G = \langle X \mid R \rangle$ is isomorphic to the quotient of the free group F_X over the alphabet X and the normal closure R^G, with R^G consisting of all products of conjugates of all relators $r \in R$. Note that we need only consider the cyclically reduced versions of r, as this represents the minimal (but not unique) form of each relator. A word w is *cyclically reduced* if and only if every cyclic permutation of w is reduced. Consequently, a word w over X is a element of N if and only if it can be expressed in the form

$$w = u_1 r_1 u_2 \dots r_k u_{k+1},$$

with $u_1 u_2 \dots u_{k+1} = 1$ in F and where r_k is a cyclically reduced permutation of a relator in $R^{\pm 1}$[46][5.1.1]. One can compare the expression above to the previous definition of a counting function and observe their similarity.

Relators of the form r_k are said to be *symmetrized*. More generally, let R_S denote the *symmetrization* of R, where every $r_s \in R_S$ is cyclically reduced and

- $R \subseteq R_S$;
- R_S contains every cyclic permutation of $r \in R$;
- $r^{-1} \in R_S \ \forall r \in R$.

That trivial words can be written as products of cyclic permutation of relators can be interpreted topologically. First let us consider the embedding of the Cayley graph of G into $\mathbb{R}^2$. The vertices are then simply points in $\mathbb{R}^2$, while the edges become bounded subsets homeomorphic to the interval $(0, 1)$. In addition to the graph (or *1-skeleton*, in topological parlance), we now have two-dimensional regions or *cells* of the Euclidean plane that are

enclosed by the embedded edges of the Cayley graph. These cells are homeomorphic to an open disk, i.e., a circle without a boundary. The edges that form the boundary of these cells retain their labels and orientation from the original Cayley graph. The points and bounded subsets of the embedded graph can be considered 0-dimensional and 1-dimensional cells respectively.

A *van Kampen diagram* for a word w over the group G is this collection of aforementioned cells $\mathcal{D}$ with the following properties:

i. No edge is labeled by 1_G;

ii. $\mathcal{D}$ is connected and simply connected;

iii. A distinguished vertex O exists on $\partial\mathcal{D}$, the boundary of $\mathcal{D}$;

iv. Any region of $\mathcal{D}$ has a label on its boundary w' which is equivalent to 1_G, that is, $w' \in N$.

Thus, from a topological perspective, a word is G trivial (i.e., belongs to N) if and only if its homotopic to a loop in the Cayley graph, or equivalently, it can be identified with the label of the boundary of a van Kampen diagram. This is the basis of the van Kampen theorem, which states that for every reduced word in G a van Kampen diagram $\mathcal{D}$ exists, and vice versa. Van Kampen diagrams were originally used to explore instances of the word problem in free groups, and were then extended to other types of groups. There is an analogue of van Kampen diagrams for conjugate elements; these are called *annular* or *Schupp* diagrams, after Paul Schupp, who used them to extend results for the conjugacy problem in small cancellation theory [47].

We now have at our disposal a bevy of mathematical machinery to extract information concerning group elements: combinatorial, geometric, and topological. As noted previously,

not all groups with decidable word problems have normal forms that are efficient to calculate. We first consider feature vectors that are applicable to finitely presented groups that possess an efficient normal form:

- n_0 (*Normal Form*) - Let G be a finitely generated group with generating set X and possessing a normal form. Let $Y = X \cup X^{-1}$. If w is a word in normal form, then w is of the form

  $$y_1^{e_1} \cdots y_N^{e_N}$$

  with $y_i \in Y$ and $e_i \in \mathbb{Z}$. The feature vector n_0 is then

  $$n_0 = \langle e_1, \ldots, e_N \rangle.$$

- n_1 (*Weighted Normal Form*) - The feature vector n_1 is the same as n_0 above, except it is weighted by the word length $|w|$:

  $$n_1 = \frac{1}{|w|} \langle e_1, \ldots, e_N \rangle.$$

The features below were introduced in [23]. They apply to finitely presented groups in general and do not require a normal form:

- f_0 (*Generator Count*) - Let the generator set X be given a fixed order and let $x_i \in X$ be the ith generator. The counting function $C(w, x_i) = |\{w_j \mid w_j = x_i \vee x_i^{-1}\}|$, that is, the number of occurrences of the generator x_i (and its inverse) in the word w. The feature vector f_0 is then

  $$f_0 = \langle C(w, x_1), \ldots, C(w, x_N) \rangle.$$

- f_1 (*Weighted Generator Count*) - The feature vector f_1 is the same as f_0 above, except

it is weighted by the word length $|w|$:

$$f_1 = \frac{1}{|w|}\langle C(w, x_1), \ldots, C(w, x_N)\rangle.$$

- f_2 through f_7 (*Cayley Subgraphs*) - These features count subwords of length 1, 2, and 3. For each subword length there is a weighted and non-weighted variant:

$$\begin{aligned}
f_2 &= \quad \langle C(w, x_1 s x_2) \mid x_1, x_2 \in X; s \in G \wedge |s| = 1\rangle \\
f_3 &= \tfrac{1}{|w|}\langle C(w, x_1 s x_2) \mid x_1, x_2 \in X; s \in G \wedge |s| = 1\rangle \\
f_4 &= \quad \langle C(w, x_1 s x_2) \mid x_1, x_2 \in X; s \in G \wedge |s| = 2\rangle \\
f_5 &= \tfrac{1}{|w|}\langle C(w, x_1 s x_2) \mid x_1, x_2 \in X; s \in G \wedge |s| = 2\rangle \\
f_6 &= \quad \langle C(w, x_1 s x_2) \mid x_1, x_2 \in X; s \in G \wedge |s| = 3\rangle \\
f_7 &= \tfrac{1}{|w|}\langle C(w, x_1 s x_2) \mid x_1, x_2 \in X; s \in G \wedge |s| = 3\rangle
\end{aligned}$$

4.3 Model Selection

In the context of machine learning, a *model* or *learning algorithm* is the means by which a set of training inputs can be used to predict the output on future, unseen inputs. The choice of a learning algorithm is informed by the type and structure of the training data, such as whether the data is discrete or continuous, or is comprised of feature vectors like those described above. A particular learning algorithm in turn determines the *hypothesis space*; the set of functions that can be learned from the data. The class of available learning algorithms that have been developed is too numerous to describe here. Instead, we will focus on a set of models that will be applied to group-theoretic problems: decision trees, random forests, and N-tuple networks.

4.3.1 Decision Trees

Decision tree learning is a model that utilizes a tree structure to encode the learned function. Trees can be used for either classification or regression analysis; we will focus on those used for classification, and in particular binary classification trees, where each node can have a maximum of two children. Every node in the decision tree corresponds to a unique partition of the measurement space. Leaf nodes correspond to the assignment of a particular class, while at internal nodes a test is performed to distinguish between data points. This distinction is encoded in the node's children and further partitions the measurement space. Trees can distinguish by feature, by combining features via a discriminant function (such as a linear discriminant), or by other means.

There are a number of tests available that can be used to partition the space at each internal node. *Gini impurity* measures the frequency at which the remaining data points would be misclassified, with the best split being that which minimizes this impurity. With *information gain*, the entropy of the parent node and the remaining data are calculated, and a partition is chosen that reduces the entropy the most, i.e., that which maximizes the information that can be obtained from the partition. The probability of misclassification can also be used.

As with many learning algorithms, decision trees are prone to overfitting. As decision trees do not have a fixed size representation (i.e., they are considered a non-parametric model), the tree-making algorithm can create large trees or ones with complex branching that do not generalize well. This can be combatted by *pruning*, whereby a subtree of the learned decision tree is replaced with a leaf node whose class is the most common one of the data points contained in the pruned subtree. Pruning can be performed by testing a subtree's classification performance against a separate data set, and keeping the subtree if improves

the performance of the classifier. Pruning can also be achieved by employing a statistical significance test such as the χ^2 test that can determine if a subtree results in a meaningful split in the data, and pruning if the subtree does not meet some threshold of significance. Yet another technique is to limit the depth to which the tree can grow during the training process, a form of *pre-pruning*.

4.3.2 Random Forests

Random forest classifiers [7] are an example of an *ensemble* method, in which multiple classifiers are combined into a single classifier. As the name implies, random forests are comprised of several decision trees that are constructed from a random sampling of the training set. Additionally, the best split at each node in a particular tree is determined not by the single best component of the feature vector, but instead by choosing the best feature among a randomly sampled subset of the feature vector's components. Using multiple trees trained on the sampled training set reduces overfitting, while using subsets of the feature vector for choosing partitions in the measurement space reduces variance. Once trained, the classification of a new sample can be determined by either averaging the classifications of each tree in the forest, or by having each tree vote for the sample's class and assigning the sample to the class with a plurality of the votes.

4.3.3 N-tuple Neural Networks

Another model that we will investigate is the *N-tuple neural network*. N-tuple classifiers were introduced in 1959 by Bledsoe and Browning [6] as a means of performing printed character recognition using specialized table lookup hardware. In its original implementation, the positions of a binary number s are sampled using a total of M random patterns of size N for each class C. The sampled positions produce another binary number b_m. These samplings

are stored in tables T_{mc} (one for each class c and random sample m), and the value of the table entry $T_{mc}(b_m)$ is the total number of times a sample s of class c was mapped by m to the binary number b_m. A new sample s' is classified by choosing the class for which the sum of $T_{mc}(b'_m)$ is maximal. If no maximum exists, the classifier does not choose a class and is said to reserve decision.

With the advent of radial basis function networks and other forms of artificial neural networks in the late 1980s and early 1990s, N-tuple classifiers were revisited, being recast as a type of weightless, single-layer artificial neural network (e.g., the "single layer lookup perceptrons" of Tattersall et al. [52]). In a series of papers, Allinson and Kołcz extended N-tuple neural networks further, developing a binary encoding scheme based on CMAC (cerebellar model arithmetic computer, an older form of artificial neural network) and Gray codes [31], as well as using NTNNs for regression analysis [32] instead of classification. In [45], Rohwer performed a series of experiments on standard pattern recognition databases, finding that for most data sets the best results were achieved with N-tuples of size 8 and the total number of patterns M around 1000.

NTNNs can be generalized beyond classifying binary data by utilizing the framework of relational algebra. Consider a feature vector s of length N and class c. Let $J_1, \ldots, J_M$ be *index sets*, that is, subsets of the set $\{1, \ldots, N\}$ that represent the indices at which to sample s. For each class c and index set J_m, form the table T_{mc}, and for each training sample s, store the number of times the projection of s onto T_{mc} via the index set J_m occurs.

Various classification criteria [22] have been devised for use with NTNNs. For all criteria listed below, if there is no unique class that satisfies the criterion the NTNN reserves decision:

- *Voting Majority* – Assign s to class c' if $\sum_{m \in M} T_{mc'}(s) > \sum_{m \in M} T_{mc}(s)$ for all classes $c \neq c'$.

- *Logarithm Voting Majority* – Assign s to class c' if $\sum_{m \in M} \log T_{mc'}(s) > \sum_{m \in M} \log T_{mc}(s)$ for all classes $c \neq c'$.

- *Least Votes* – Assign s to class c' if $\min_{m \in M} T_{mc'}(s) < \min_{m \in M} T_{mc}(s)$ for all classes $c \neq c'$.

- *Minimum Probability* – Assign sample s to class c' when

$$\min_{c'} P_{c'}(T_{mc'}(s) > 0 \mid c')P(c') \geq \min_{c} P_c(T_{mc}(s) > 0 \mid c)P(c)$$

  for all $c \neq c'$.

4.3.4 N-tuple Regression Networks

The N-tuple neural network classifiers of the previous section can be modified so that they perform regression analysis rather than classification. N-tuple regression networks (NTRNs) were introduced by Tattersall et al. in [52] as single-layer lookup perceptrons (SLLUPs). In that paper, the authors demonstrated that SLLUPs could be considered non-linear function interpolators that operate by convolving a low-pass filter with the input data. This is similar to how radial basis function (RBF) interpolation works, in that the network's output is a weighted sum of each RBF, except that the N-tuple patterns represent implicit functions rather than explicit ones.

In [32], Kołcz and Allinson showed that SLLUPs can be considered as a type of NTRN. They then showed that NTRNs are equivalent to a form of non-parametric kernel regression analysis. Let X and Y be random variables of which the input/output pairs (x, y) of our training set are respective realizations. By assuming the existence of a joint probability distribution $P(X, Y)$, a trained NTRN learns the conditional expectation $\mathbb{E}(y \mid x)$ of an unknown regression function f, i.e., the expected value of the function output y given x.

In our implementation of the NTRN, let N be the dimension of the feature vector and D the dimension of the output vector. For each $d \in D$ we maintain a distinct set of M patterns of size P. Thus, for each d and $m \in M$ we populate tables T_{md} with projections of each input x. As in the NTNN case, each table entry k contains a count c_k of the number of times elements of the training set were projected onto entry k of T_{md}. In addition, an estimate w_k for the dth output of the network response is stored, which is initialized to zero. Given an N-tuple table entry containing a previous value of w_k and the observed output y_d, the updated value w'_k is calculated as

$$w'_k = w_k + y_d.$$

During the output phase, the NTRN is given a new (unseen) input x from which it constructs a D-dimensional response vector $\hat{y}$. For each $d \in [1, D]$ and $m \in M$, the NTRN produces an estimate of the dth value of $\hat{y}$ by first projecting x onto T_{md}, and then producing an average of the estimates w_k stored in each projected entry k of T_{md}:

$$\hat{y_d} = \frac{\sum_k w_k}{\sum_k c_k}.$$

The average as specified above is the *arithmetic mean*. This mean is not a particularly robust statistic: it is overly sensitive to outliers (i.e., unusually small and large values) in the training data. To mitigate this behavior, one can instead use the *geometric mean*. During the training phase, the updated value w'_k is calculated as

$$w'_k = w_k + \ln y_d.$$

Using the same notation as before, the network's response using the geometric mean is calculated as:

$$\hat{y_d} = \exp\left(\frac{\sum_k w_k}{\sum_k c_k}\right).$$

For both means, the output vector $\hat{y}$ will need to be made integral by applying some form of rounding function (e.g., floor, ceiling) component-wise. As as alternative, the network response can be configured to calculate the median of the estimates for each output component. For each $d \in [1, D]$ and $m \in M$, the NTRN produces an estimate of the dth value of $\hat{y}$ by first projecting x onto T_{md}. For each entry k, a set $\{\hat{y_d}\}$ is constructed by inserting the estimate w_k a total of c_k times. If $|\{\hat{y_d}\}|$ is odd, then $\hat{y_d}$ is the median of the set, otherwise $\hat{y_d}$ is the floor (or ceiling) of the mean of the two middle values.

In producing the output vector $\hat{y}$ above, we are making an implicit independence assumption on the joint probability distribution, specifically that

$$P(X, Y) = P(X)(Y).$$

This assumption may prove too restrictive to produce an effective response. Therefore, we construct an additional set of tables T_{qdi} that can be used to estimate the values of X and Y via an unknown random variable Z. These tables will operate under the weaker assumption of conditional independence of X and Y from Z, namely

$$P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z).$$

Let $Q < D$ be the number of 1-dimensional patterns (i.e., indices) that will be stored for each dimension $d \in D$, and let $i \in \{x, y\}$ indicate whether the table is to record entries for input x or output y. Given a training sample (x, y) and the pattern q for dimension d,

the table T_{qdx} will store the projection (x_d, y_q), while the table T_{qdy} will store the projection (y_d, x_q). As before, each table entry k will contain a count c_k of the number of times that samples of the training set were projected onto entry k of T_{qdi}.

For a concrete example, let $N = 4$ and $D = 2$. The possible values of q are then $\{1, 2\}$ (we will assume $Q = 1$ for this demonstration). Consider the sample $(x, y) = (\langle a, b, c, d\rangle, \langle r, s\rangle)$. Let $d = 1$ and $q = 2$, then the projections of (x, y) are

$$\begin{array}{rcl} \pi_{qdx}((x, y)) & = & (a, s) \\ \pi_{qdy}((x, y)) & = & (r, b) \end{array},$$

which are stored in tables T_{qdx} and T_{qdy} respectively.

To generate the network response for a new input x using the tables T_{qdi}, we proceed component-wise as before. For each $d \in [1, D]$ and $q \in Q$, the NTRN produces an estimate of the dth value of $\hat{y}$ by first projecting x onto T_{qdx}. This produces a tuple (x_d, z), and for each entry $k = (y_k, z_k)$ in T_{qdy} such that $z = z_k$, the median of the values y_k is evaluated as above by adding c_k copies of each y_k to the set $\{\hat{y}_d\}$, and taking the median of this set if its cardinality is odd, or the rounded mean of the two middle values if its cardinality is even.

4.4 Generating Data

The primary goal of a supervised machine learning system is to predict classes or values on new, unseen members of the data domain. As a consequence, the system's construction requires at least two sets of data - one for training and one for evaluation. When data is scarce the method of *cross-validation* can be employed, whereby the data set is partitioned into separate sets that are used exclusively for training or evaluation. Depending on the learning model chosen, a third data set may be required for use during the optimization process that occurs between the training and evaluation phases. Our approach is to produce

three, independently generated sets that are used in each phase of the system's construction. The sets used for training, optimization, and verification are respectively referred to as S_i, S_o, and S_v.

Note that the data generation process for a finitely presented group is dependent upon the group under consideration and the learning task at hand.

4.5 Evaluation and Analysis

There are various means by which the performance of a machine learning system can be evaluated. Regardless of which method is chosen, it is imperative that an independent data set (e.g., the verification set S_v from the previous section) is used to measure the system's performance. We will focus on methods available to the different learning tasks of classification and regression analysis.

In classification, we are primarily concerned with the *accuracy* $\mathcal{A}$ of a model $\mathcal{M}(f)$, trained with respect to the feature vector f, over the verification set S_v:

$$\mathcal{A}(\mathcal{M}(f), S_v) = \frac{|\text{True Positives}(S_v)| + |\text{True Negatives}(S_v)|}{|S_v|}.$$

If performance metrics other than accuracy are required, a *confusion matrix* can be calculated. The matrix is a 2×2 table that indicates the classification of the samples in the verification set relative to a particular class C. For instance, if a sample $s_0 \in S_v$ was labeled $C' \neq C$ but the classifier assigned s_0 to C, the classification of s_0 would be considered a *false positive*. The full table is depicted below:

		Assigned	
		C	$\neg C$
True	C	True Positives	False Negatives
	$\neg C$	False Positives	True Negatives

Table 4.1: Values of a Confusion Matrix for Class C

The confusion matrix enables us to derive a whole host of performance measures. For instance, we can define *precision*, or positive predictive value, as

$$\text{Precision}(\mathcal{M}(f), S_v) = \frac{|\text{True Positives}(S_v)|}{|S_v|}.$$

In regression analysis, we are given a feature vector $\vec{x}$ with true output $\vec{y}$ of dimension D and are looking to produce an estimated value $\hat{y}$. There are many methods of measuring regression error; we emphasize two that are relevant to the NTRN method of section 4.3.4. The *mean squared error* for each sample is calculated as

$$MSE = \frac{1}{D}\sum_{i=1}^{D}\left(\hat{y}_i - y_i\right)^2.$$

Using the same notation as above, the *mean absolute error* (MAE) for each sample is calculated as

$$MAE = \frac{1}{D}\sum_{i=1}^{D}\left|\hat{y}_i - y_i\right|.$$

The performance of the regression network is then the average of the MSE or MAE over the entire verification set S_v.

Beyond verifying that a machine learning system performs as intended, analysis can provide may benefits, including greater performance, elimination of irrelevant features, and the discovery of heretofore unknown mathematical relationships in the data domain.

Chapter 5

Solving the Conjugacy Decision Problem via Machine Learning

Recall that the *conjugacy decision problem* for a group G is to determine for any $u, v \in G$ if u is conjugate to v. With respect to computability, the conjugacy decision problem is in fact *two* problems - each concerned with determining positive or negative solutions exclusively. The positive conjugacy decision problem in any recursively presented group is computable, as for any element in the group its conjugates can be recursively enumerated [39]. The negative solution is not guaranteed to be computable for non-finite groups. There are classes of groups for which both parts of the conjugacy decision problem are computable, including finitely generated polycyclic and metabelian groups.

Computability, however, does not imply efficiency. The efficient algorithms that do exist are often restricted in some sense, such as answering only one part of the decision problem or being solely applicable to a specific class of groups. For instance, a polynomial algorithm exists [35] for the full conjugacy decision problem in the Grigorchuk groups, while for non-cyclic finitely generated groups of infinite abelianization, a linear algorithm was found [28]

that can only be used to solve the negative conjugacy decision problem.

Given the limitations of existing algorithms, we turn to the framework of the previous chapter for a machine learning solution. In the sections below we outline how we adapt the general framework to constructing machine learning systems for the conjugacy decision problem. We use the aforementioned supervised learning methods to train classifiers for several classes of finitely presented group. These classifiers can determine whether a pair of elements from their respective groups are conjugate or not, and do so with very high accuracy. We analyze the performance of different models and feature vectors for each group, as well as provide methods for visualizing the NTNN classifiers.

The results in this chapter represent joint work with Robert Haralick and Delaram Kahrobaei.

5.1 Machine Learning System for the Conjugacy Decision Problem

As outlined in the previous chapter, constructing a machine learning system for the conjugacy decision problem requires us to consider feature extraction, model selection, data generation, and evaluation criteria. We delineate our solutions to each of these tasks below.

5.1.1 Feature Extraction

Given a group G and words $u, v \in G$, we concatenate (denoted by $\|$) the unit feature vectors n_0 and n_1 from section 4.2 to create two derived feature vectors for the conjugacy decision problem:

$$c_0 = \langle n_0(u) \parallel n_0(v) \rangle$$

$$c_1 = \langle n_1(u) \parallel n_1(v) \rangle$$

The default feature vector used with tree-based classifiers is c_1 (weighted normal forms), while for NTNNs it is c_0 (unweighted normal forms). Additional feature vectors and normal forms that are used for a particular group are included in that group's experimental results section.

5.1.2 Model Selection

For the conjugacy decision problem we can utilize all three classifiers defined in section 4.3: decision trees, random forests, and NTNNs. The parameters and implementation of each model are discussed below:

Decision Trees

Using the `DecisionTreeClassifier` from Scikit-learn [43], we trained a decision tree classifier for each group on its respective training set S_i. Each leaf node was required to contain at least one sample, and a split could only occur if there were at least two samples at that node. The accuracy of each classifier was calculated by classifying the data in each group's verification set S_v. Both Gini impurity and information gain were used to determine the best split. For the depth limit, we tested not having a limit, as well as limiting the number of levels to $\log_2 S_i - 1$.

Random Forests

Using the `RandomForestClassifier` from Scikit-learn [43], we trained a random forest classifier for each of group on its respective training set S_i. As in the case of single decision trees, each leaf node was required to contain at least one sample, and a split could only occur if there were at least two samples at that node. Additionally, the number of trees in the forest was 10, and the size of the random subset of features was $\sqrt{|c_1|}$, i.e., the square

root of the length of the feature vector. The same combinations of split criteria and depth limits for decision trees were used for random forests as well.

NTNNs

For learning a NTNN classifier, the total number of patterns M was varied, with M taking on values from the set $\{10, 20, 30, 50, 100\}$. The initial size of the patterns was set to 3, and where applicable, sizes in the range $[3, 5]$ were tested. We used both the "Voting Majority" and "Logarithm Voting Majority" criteria for classification in our tests.

For a given feature vector of dimension N, the total number of patterns of size $P \leq N$ is $\binom{N}{P}$. When initializing a NTNN classifier that uses pattern sets of size M (a set of M patterns of size P), the list of $\binom{N}{P}$ patterns is generated, and a separate permutation of each list is kept for each class C (in our case, $C = 2$, as we are performing binary classification). Before training the NTNN on the set S_i, each class is assigned the first M patterns from its pattern list.

As the accuracy of the initial random selection of patterns varies considerably, a random restart was implemented, in which a new NTNN was initiated with a new random permutation of all possible patterns. Each NTNN's performance was tested against the set S_o, with the NTNN proceeding to the optimization stage only when its accuracy was greater than the *starting threshold* θ_α, which was set to 60%.

During the optimization phase, the algorithm alternates between each classes' list of patterns in choosing the next test pattern. Each pattern in M is swapped out with the test pattern, and the NTNN is evaluated against the optimization set S_o. The algorithm keeps track of the pattern m whose replacement with the test pattern improves accuracy the most over all $m \in M$, and makes that pattern swap permanent if a new best accuracy is achieved.

The algorithm will continue this process until all patterns have been exhausted or the *goal threshold* θ_ω is reached, which was set to 97%. The NTNN classifier and the current location in the pattern list are then saved, so that optimization can be continued at a later time if desired.

5.1.3 Generating Data

Each dataset consists of 20,000 words in normal form, with two 10,000 word halves that are generated via the following procedures:

1. *Random Non-Conjugate Words in Normal Form* - For each $n \in [5, 1004]$ we generate two words $u, v \in G$ with $|u| = |v| = n$. A word w is generated uniformly and randomly by starting with the identity element $w = 1_G$, then selecting a generator g from X and performing the product $w = w \cdot g$. The element is then converted into its normal form w', and the length $|w'|$ is computed. Additional products are computed until $|w'| = n$.

   After generating each u, v pair, an additional step is required to verify that $u \not\sim v$. Using the method from [28], we construct the derived (or commutator) subgroup of G, denoted $[G, G]$, and an epimorphism $\phi : G \to G/[G, G]$. We then look at the images $\phi(u)$ and $\phi(v)$ and reject the pair if they map to the same representative in the quotient $G/[G, G]$. This process is repeated until 10 non-conjugate pairs are generated for each n.

2. *Random Conjugates in Normal Form* - For $n \in [5, 1004]$ we generate a pair of words $v, z \in G$ with $|v| = |z| = n$. Each word v, z is generated uniformly and randomly as above. After v and z are generated, the word $u = v^z$ is formed, and the tuple (u, v, z) is added to the dataset. This process is repeated 10 times for each n.

Note that the minimal length of two conjugate words written in their trivial form is 4, as this corresponds to a pair of elements that are abelian (i.e., they commute under the group

operation).

Three disjoint datasets were generated using the procedure above:

1. *Training Set* S_i - The set S_i was used to train all three classifiers.

2. *Optimization Set* - The set S_o was used to optimize the choice and number of patterns for each NTNN.

3. *Verification Set* - The set S_v was used to evaluate the performance of our trained and/or optimized classifier.

One instance of each of the three data sets was generated for each group.

5.1.4 Evaluation and Analysis

For the conjugacy decision problem over a group G, we are primarily concerned with the accuracy of the model over the verification set, $\mathcal{A}(\mathcal{M}(f), S_v)$, as defined in section 4.5.

5.2 Experimental Results

In this section we present experimental results from the application of our machine learning system to the conjugacy decision problem in several groups. We note the following conventions that are used throughout this section:

- All confusion matrices are from the perspective of the class of conjugate elements.

- Accuracy for the NTNN classifier is with respect to "Voting Majority" unless otherwise noted.

- When both "Voting Majority" and "Logarithm Voting Majority" accuracies are available for a NTNN classifier, they are denoted in the experimental results tables as "Accuracy (Σ)" and "Accuracy (log)" respectively.

5.2.1 The Baumslag-Solitar Group BS(1,2)

Recall that one-relator groups are finitely presented groups of the form $\langle X \mid R \rangle$, where $|R|$ is strictly one. These groups played an important role in the development of combinatorial and geometric group theory. The Baumslag-Solitar groups are a well-known class of one-relator groups that include among their number instances of non-Hopfian groups. A group G is *non-Hopfian* if there exists an epimorphism from G to itself that is not an isomorphism. We will consider the non-hyperbolic Baumslag-Solitar group of Example 2.1.2:

$$BS(1,2) = \langle a, b, \mid bab^{-1}a^{-2} \rangle.$$

Note that the conjugacy decision problem over BS(1,2) resides in the complexity class TC^0 [53], where TC^0 is the class of constant-depth arithmetic circuits using AND, OR, NOT, and majority gates.

Normal Form and Feature Vectors

Elements in BS(1,2) can be uniquely written in the following normal form:

$$n_0 = b^{e_1}a^{e_2}b^{e_3},$$

with $e_1 \leq 0$ and $e_3 \geq 0$. Collection from the left can transform any element of BS(1,2) into this normal form. The dimension of feature vectors c_0 and c_1 is 6.

Decision Trees and Random Forests

Method, Split Criterion, Depth	Accuracy
Tree, Gini Impurity, No Depth Limit	91.24%
Tree, Gini Impurity, Depth Limit	91.85%
Tree, Entropy, No Depth Limit	91.53%
Tree, Entropy, Depth Limit	92.00%
Random Forest, Gini Impurity, No Depth Limit	93.72%
Random Forest, Gini Impurity, Depth Limit	93.17%
Random Forest, Entropy, No Depth Limit	**93.64%**
Random Forest, Entropy, Depth Limit	93.16%

Table 5.1: Decision Tree and Random Forest Results for BS(1,2)

For BS(1,2), the random forest classifier using entropy as the split criterion and with no depth limit performed the best, with an overall accuracy of 93.64%. The confusion matrix for this classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9713	287
	Non-Conjugate	969	9031

Table 5.2: Confusion Matrix for the Best Performing Tree Classifier of BS(1,2)

NTNN

For BS(1,2) the NTNN classifier did not perform well using the feature vector c_0. It may be that the relatively low dimension of the feature vector for BS(1,2) ($N = 6$) provides insufficient information to the classifier. Therefore, we also tested features vectors c_2 and c_4, defined as the concatenation of unit vectors f_2 and f_4 respectively. The feature vector c_2 is of dimension 48, while c_4 has dimension 96. The use of the feature vector c_2 produced a marked improvement in accuracy as compared to c_0. We ran the full array of tests over all (M, P) pairs for c_2, but ran only three additional tests using the c_4 feature vector, as in

these initial tests we did not see any improvement in the performance of the NTNN classifier as compared to using c_2.

Group	**M**	**P**	**Accuracy (Σ)**	**Accuracy (log)**
BS(1,2)	6	1	66.46%	73.40%
BS(1,2)	6	2	71.26%	76.37%
BS(1,2)	6	3	68.84%	71.38%
BS(1,2)	10	2	72.23%	73.96%
BS(1,2)	10	3	69.18%	71.42%
BS(1,2)	15	2	70.96%	71.77%
BS(1,2)	15	3	71.17%	71.36%
BS(1,2)	20	3	70.83%	73.29%

Table 5.3: NTNN Results for BS(1,2) Using Feature Vector c_0

Group	**M**	**P**	**Accuracy (log)**
BS(1,2)	10	3	87.76%
BS(1,2)	10	4	87.98%
BS(1,2)	10	5	82.04%
BS(1,2)	20	3	92.08%
BS(1,2)	20	4	91.10%
BS(1,2)	20	5	89.33%
BS(1,2)	30	3	90.15%
BS(1,2)	30	4	**92.41%**
BS(1,2)	30	5	89.97%
BS(1,2)	50	3	88.71%
BS(1,2)	50	4	90.47%
BS(1,2)	50	5	89.34%
BS(1,2)	100	3	84.94%
BS(1,2)	100	4	90.53%
BS(1,2)	100	5	90.42%

Table 5.4: NTNN Results for BS(1,2) Using Feature Vector c_2

Group	**M**	**P**	**Accuracy (log)**
BS(1,2)	30	3	85.58%
BS(1,2)	30	4	92.37%
BS(1,2)	50	3	92.33%

Table 5.5: NTNN Results for BS(1,2) Using Feature Vector c_4

For BS(1,2), the NTNN classifier using feature vector c_2 and parameters $M = 30$, $P = 4$ performed the best, with an overall accuracy of 92.41%. The confusion matrix for this classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	8817	1183
	Non-Conjugate	336	9664

Table 5.6: Confusion Matrix for the Best Performing NTNN Classifier of BS(1,2)

Improving Performance

Despite achieving an accuracy of 92.41%, the performance of all classification models for BS(1,2) is relatively low compared to the other groups tested. Given the high dimensionality of the feature vectors c_2 and c_4, it may be possible to improve performance by utilizing *feature selection*, whereby a portion of the components of the feature vector are discarded. Suggestions for the best components may be found by using the NTNN visualizations described later in the chapter.

Another option is to utilize a faithful linear representation of BS(1,2), with generators a and b corresponding respectively to the matrices A and B below:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

The feature vector using this representation would be similar to f_m for $\mathrm{SL}(2, \mathbb{Z})$ in section 5.2.4, and can be used with decision trees and random forests.

5.2.2 Non-Nilpotent Polycyclic Groups

Polycyclic groups that are non-virtually nilpotent have exponential word growth ([54],[37]) and remain promising candidates for use as platform groups [20]. One method of constructing

such groups is through the use of algebraic number fields, as outlined in [24, §8.2.2].

Given an algebraic number field F with degree $[F : \mathbb{Q}] > 1$, one can define two substructures, the *maximal order* $\mathcal{O}(F)$ and the *unit group* $U(F)$. The maximal order is the largest ring of integers of F, and consists of those elements in F that are a root of some monic polynomial over F with integral coefficients. The multiplicative group $U(F)$ consists wholly of the non-zero elements of $\mathcal{O}(F)$ that have a multiplicative inverse, i.e., are units. Given these two structures and the aforementioned degree criterion, the semidirect product $\mathcal{O}(F) \rtimes U(F)$ results in an infinite, non-virtually nilpotent polycyclic group.

Below are three specific instances of the $\mathcal{O}(F) \rtimes U(F)$ family of polycyclic groups. The conjugacy search problem over the first two groups was studied in [16], in the context of the length-based attack (see section 2.2.3 for further details). The groups can be constructed by using the `MaximalOrderByUnitsPcpGroup` function of the GAP Polycyclic package [13]. The function takes a polynomial that is irreducible over $\mathbb{Q}$ (thereby defining a field extension of $\mathbb{Q}$) and returns a group of the form $\mathcal{O}(F) \rtimes U(F)$:

- $\mathcal{O} \rtimes U_{14}$ - Given the polynomial $f = x^9 - 7x^3 - 1$, `MaximalOrderByUnitsPcpGroup` returns a group of the form $\mathcal{O}(F) \rtimes U(F)$ with a Hirsch length of 14.
- $\mathcal{O} \rtimes U_{16}$ - Given the polynomial $f = x^{11} - x^3 - 1$, `MaximalOrderByUnitsPcpGroup` returns a group of the form $\mathcal{O}(F) \rtimes U(F)$ with a Hirsch length of 16.
- $\mathcal{O} \rtimes U_{34}$ - Given the polynomial $f = x^{23} - x^3 - 1$, `MaximalOrderByUnitsPcpGroup` returns a group of the form $\mathcal{O}(F) \rtimes U(F)$ with a Hirsch length of 34.

Normal Form

Recall from section 2.1.9 that every polycyclic group has a normal form in terms of the generators in its polycyclic sequence. The feature vector n_0 for a polycyclic group element g

simply corresponds to the exponent vector of g in normal form. Thus the feature vectors c_0 and c_1 are readily computable for polycyclic group elements. The dimension of these feature vectors for groups of the form $\mathcal{O}(F) \rtimes U(F)$ is $2(H+1)$, where H is the Hirsch length of the group.

Decision Trees and Random Forests

Method, Split Criterion, Depth	**Accuracy**
Tree, Gini Impurity, No Depth Limit	98.05%
Tree, Gini Impurity, Depth Limit	98.20%
Tree, Entropy, No Depth Limit	98.52%
Tree, Entropy, Depth Limit	98.49%
Random Forest, Gini Impurity, No Depth Limit	98.42%
Random Forest, Gini Impurity, Depth Limit	98.61%
Random Forest, Entropy, No Depth Limit	**98.69%**
Random Forest, Entropy, Depth Limit	98.63%

Table 5.7: Decision Tree and Random Forest Results for $\mathcal{O} \rtimes U_{14}$

For $\mathcal{O} \rtimes U_{14}$, the random forest classifier using entropy as the split criterion and with no depth limit performed the best, with an overall accuracy of 98.69%. The confusion matrix for this classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9952	48
	Non-Conjugate	215	9785

Table 5.8: Confusion Matrix for the Best Performing Tree Classifier of $\mathcal{O} \rtimes U_{14}$

Method, Split Criterion, Depth	Accuracy
Tree, Gini Impurity, No Depth Limit	96.43%
Tree, Gini Impurity, Depth Limit	96.64%
Tree, Entropy, No Depth Limit	97.23%
Tree, Entropy, Depth Limit	97.21%
Random Forest, Gini Impurity, No Depth Limit	97.74%
Random Forest, Gini Impurity, Depth Limit	97.99%
Random Forest, Entropy, No Depth Limit	98.01%
Random Forest, Entropy, Depth Limit	**98.19%**

Table 5.9: Decision Tree and Random Forest Results for $\mathcal{O} \rtimes U_{16}$

For $\mathcal{O} \rtimes U_{16}$, the random forest classifier using entropy as the split criterion and with the standard depth limit performed the best, with an overall accuracy of 98.19%. The confusion matrix for this classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9974	26
	Non-Conjugate	335	9665

Table 5.10: Confusion Matrix for the Best Performing Tree Classifier of $\mathcal{O} \rtimes U_{16}$

Method, Split Criterion, Depth	Accuracy
Tree, Gini Impurity, No Depth Limit	97.99%
Tree, Gini Impurity, Depth Limit	98.02%
Tree, Entropy, No Depth Limit	98.34%
Tree, Entropy, Depth Limit	98.47%
Random Forest, Gini Impurity, No Depth Limit	98.61%
Random Forest, Gini Impurity, Depth Limit	98.83%
Random Forest, Entropy, No Depth Limit	**98.89%**
Random Forest, Entropy, Depth Limit	98.82%

Table 5.11: Decision Tree and Random Forest Results for $\mathcal{O} \rtimes U_{34}$

For $\mathcal{O} \rtimes U_{34}$, the random forest classifier using entropy as the split criterion and with no

depth limit performed the best, with an overall accuracy of 98.89%. The confusion matrix for this classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9989	11
	Non-Conjugate	212	9788

Table 5.12: Confusion Matrix for the Best Performing Tree Classifier of $\mathcal{O} \rtimes U_{34}$

NTNN

For the group $\mathcal{O} \rtimes U_{14}$ the best parameters are $M = 20, P = 3$ for both classification criteria.

Group	**M**	**P**	**Accuracy (Σ)**	**Accuracy (log)**
$\mathcal{O} \rtimes U_{14}$	10	3	90.32%	91.49%
$\mathcal{O} \rtimes U_{14}$	10	4	95.93%	97.41%
$\mathcal{O} \rtimes U_{14}$	10	5	97.12%	97.59%
$\mathcal{O} \rtimes U_{14}$	20	3	**97.88%**	**98.77%**
$\mathcal{O} \rtimes U_{14}$	20	4	97.17%	96.57%
$\mathcal{O} \rtimes U_{14}$	20	5	97.37%	97.19%
$\mathcal{O} \rtimes U_{14}$	30	3	96.70%	97.34%
$\mathcal{O} \rtimes U_{14}$	30	4	96.15%	96.00%
$\mathcal{O} \rtimes U_{14}$	30	5	96.98%	77.91%
$\mathcal{O} \rtimes U_{14}$	50	3	95.52%	91.93%
$\mathcal{O} \rtimes U_{14}$	50	4	93.68%	89.28%
$\mathcal{O} \rtimes U_{14}$	50	5	97.37%	91.87%
$\mathcal{O} \rtimes U_{14}$	100	3	96.75%	93.77%
$\mathcal{O} \rtimes U_{14}$	100	4	96.26%	96.75%
$\mathcal{O} \rtimes U_{14}$	100	5	97.44%	96.60%

Table 5.13: NTNN Results for $\mathcal{O} \rtimes U_{14}$

The confusion matrix for the best performing NTNN classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9995	5
	Non-Conjugate	242	9758

Table 5.14: Confusion Matrix for the Best Performing NTNN Classifier of $\mathcal{O} \rtimes U_{14}$

For the group $\mathcal{O} \rtimes U_{16}$ the best parameters are $M = 20, P = 5$ using voting majority, and $M = 10, P = 3$ for the logarithm voting majority.

Group	**M**	**P**	**Accuracy (Σ)**	**Accuracy (log)**
$\mathcal{O} \rtimes U_{16}$	10	3	94.36%	**94.25%**
$\mathcal{O} \rtimes U_{16}$	10	4	96.56%	88.82%
$\mathcal{O} \rtimes U_{16}$	10	5	98.01%	80.68%
$\mathcal{O} \rtimes U_{16}$	20	3	97.30%	93.58%
$\mathcal{O} \rtimes U_{16}$	20	4	96.16%	86.21%
$\mathcal{O} \rtimes U_{16}$	20	5	**98.46%**	88.06%
$\mathcal{O} \rtimes U_{16}$	30	3	97.68%	92.12%
$\mathcal{O} \rtimes U_{16}$	30	4	96.54%	90.74%
$\mathcal{O} \rtimes U_{16}$	30	5	98.41%	90.85%
$\mathcal{O} \rtimes U_{16}$	50	3	97.45%	87.31%
$\mathcal{O} \rtimes U_{16}$	50	4	97.52%	90.41%
$\mathcal{O} \rtimes U_{16}$	50	5	97.75%	91.94%
$\mathcal{O} \rtimes U_{16}$	100	3	97.32%	91.06%
$\mathcal{O} \rtimes U_{16}$	100	4	96.60%	92.00%
$\mathcal{O} \rtimes U_{16}$	100	5	97.15%	90.39%

Table 5.15: NTNN Results for $\mathcal{O} \rtimes U_{16}$

The confusion matrix for the best performing NTNN classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9950	26
	Non-Conjugate	247	9741

Table 5.16: Confusion Matrix for the Best Performing NTNN Classifier of $\mathcal{O} \rtimes U_{16}$

The classifier reserved decision for 24 conjugate pairs and 12 non-conjugate pairs.

For the group $\mathcal{O} \rtimes U_{34}$ the best parameters are $M = 50, P = 5$ using voting majority, and $M = 100, P = 3$ for the logarithm voting majority.

Group	**M**	**P**	**Accuracy (Σ)**	**Accuracy (log)**
$\mathcal{O} \rtimes U_{34}$	10	3	94.82%	93.15%
$\mathcal{O} \rtimes U_{34}$	10	4	96.90%	95.91%
$\mathcal{O} \rtimes U_{34}$	10	5	96.36%	96.68%
$\mathcal{O} \rtimes U_{34}$	20	3	95.02%	95.37%
$\mathcal{O} \rtimes U_{34}$	20	4	96.61%	98.09%
$\mathcal{O} \rtimes U_{34}$	20	5	96.74%	95.54%
$\mathcal{O} \rtimes U_{34}$	30	3	98.12%	99.15%
$\mathcal{O} \rtimes U_{34}$	30	4	97.14%	97.87%
$\mathcal{O} \rtimes U_{34}$	30	5	84.13%	97.37%
$\mathcal{O} \rtimes U_{34}$	50	3	97.00%	95.73%
$\mathcal{O} \rtimes U_{34}$	50	4	96.77%	97.00%
$\mathcal{O} \rtimes U_{34}$	50	5	**98.71%**	98.99%
$\mathcal{O} \rtimes U_{34}$	100	3	96.35%	**99.50%**
$\mathcal{O} \rtimes U_{34}$	100	4	97.00%	97.46%
$\mathcal{O} \rtimes U_{34}$	100	5	91.80%	93.57%

Table 5.17: NTNN Results for $\mathcal{O} \rtimes U_{34}$

The confusion matrix for the best performing NTNN classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9914	86
	Non-Conjugate	14	9986

Table 5.18: Confusion Matrix for the Best Performing NTNN Classifier of $\mathcal{O} \rtimes U_{34}$

5.2.3 Generalized Metabelian Baumslag-Solitar Groups

Generalized metabelian Baumslag-Solitar groups are the family of polycyclic and metabelian groups defined in Section 3.1.1. We tested the group GMBS(2,3) of Example 3.1.1, whose

presentation we reproduce here for convenience:

$$\mathrm{GMBS}(2,3) = \langle q_1, q_2, b \mid b^{q_1} = b^2, b^{q_2} = b^3, [q_1, q_2] = 1 \rangle.$$

Normal Form

Elements in GMBS(2,3) can be uniquely written in the following normal form:

$$n_0 = q_1^{e_1} q_2^{e_2} b^{e_3} q_1^{e_4} q_2^{e_5},$$

with $e_1, e_2 \leq 0$ and $e_4, e_5 \geq 0$. Collection from the left can transform any element of GMBS(2,3) into this normal form. The dimension of feature vectors c_0 and c_1 is 10.

Decision Trees and Random Forests

Method, Split Criterion, Depth	**Accuracy**
Tree, Gini Impurity, No Depth Limit	93.88%
Tree, Gini Impurity, Depth Limit	95.43%
Tree, Entropy, No Depth Limit	94.25%
Tree, Entropy, Depth Limit	95.32%
Random Forest, Gini Impurity, No Depth Limit	96.31%
Random Forest, Gini Impurity, Depth Limit	96.35%
Random Forest, Entropy, No Depth Limit	**96.49%**
Random Forest, Entropy, Depth Limit	96.40%

Table 5.19: Decision Tree and Random Forest Results for GMBS(2,3)

For GMBS(2,3), the random forest classifier using entropy as the split criterion and with no depth limit performed the best, with an overall accuracy of 96.49%. The confusion matrix for this classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9931	69
	Non-Conjugate	632	9368

Table 5.20: Confusion Matrix for Best Performing Tree Classifier of GMBS(2,3)

NTNN

For the group GMBS(2,3), the best parameters are $M = 30, P = 4$ using voting majority.

Group	**M**	**P**	**Accuracy** (Σ)	**Accuracy** (log)
GMBS(2,3)	10	3	84.89%	87.39%
GMBS(2,3)	10	4	94.73%	82.05%
GMBS(2,3)	10	5	92.28%	85.26%
GMBS(2,3)	20	3	84.51%	82.03%
GMBS(2,3)	20	4	93.21%	83.47%
GMBS(2,3)	20	5	93.59%	87.80%
GMBS(2,3)	30	3	84.45%	83.55%
GMBS(2,3)	30	4	**96.13%**	67.98%
GMBS(2,3)	30	5	93.43%	87.85%
GMBS(2,3)	50	3	84.08%	85.25%
GMBS(2,3)	50	4	94.73%	82.02%
GMBS(2,3)	50	5	93.17%	87.25%
GMBS(2,3)	100	3	81.50%	82.16%
GMBS(2,3)	100	4	94.54%	81.55%
GMBS(2,3)	100	5	93.16%	86.75%

Table 5.21: NTNN Results for GMBS(2,3)

The confusion matrix for the best performing NTNN classifier on the verification set S_v is below. Note that the classifier reserved decision for 42 conjugate pairs and 23 non-conjugate pairs.

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9737	221
	Non-Conjugate	489	9488

Table 5.22: Confusion Matrix for the Best Performing NTNN Classifier of GMBS(2,3)

5.2.4 $\mathbf{SL}(2,\mathbb{Z})$

Recall that $\mathrm{SL}(2,\mathbb{Z})$ is the set of 2×2 integral matrices with determinant 1. This set forms a group under matrix multiplication, and is a discrete subgroup of $\mathrm{SL}(2,\mathbb{R})$.

Representation

The group $\mathrm{SL}(2,\mathbb{Z})$ was implemented in `GAP` with a dual representation: for each element $x \in \mathrm{SL}(2,\mathbb{Z})$ we have a pair (m, w) of the form

$$m = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, w = w_1 \cdots w_n, w_i \in \{S^{\pm 1}, R^{\pm 1}\},$$

with $a, b, c, d \in \mathbb{Z}$ such that $ad - bc = 1$, and S and R corresponding to the matrices below that generate $\mathrm{SL}(2,\mathbb{Z})$:

$$S = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, R = \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}.$$

In this formulation, $\mathrm{SL}(2,\mathbb{Z})$ is an amalgamated free product given by the presentation

$$\mathrm{SL}(2,\mathbb{Z}) \cong \langle S, R \mid S^4 = 1, S^2 = R^3 \rangle.$$

These generators and attendant presentation were chosen so that a confluent rewriting system could be constructed in `GAP` via the Knuth-Bendix algorithm.

Word Length

In generating the data sets, the length of an element x was taken to be the length of the word representation of the element, i.e., $|x| = |w|$, as suggested in [50]. When the matrix form m of the element x is needed, the norm of the matrix, $\|m\|$, can be used as a length measure. We utilized the *Frobenius norm*, which is calculated as

$$\|m\| = \sqrt{a^2 + b^2 + c^2 + d^2}.$$

Note that words of relatively small length can have corresponding matrix forms with large integral entries and, consequently, a large norm. Consider a word w of length $|w| = 50$ from the training set:

$$w = (RS)^2R^{-1}SR^{-1}(SR^{-1}(SR)^2)^2SR^{-1}SR(SR(SR^{-1})^2SRSR^{-1})^2(SR^-1)^3S^{-1}.$$

It has the corresponding matrix form m below, with $\|m\| \approx 40360$:

$$m = \begin{bmatrix} 3336 & -15761 \\ -7663 & 36204 \end{bmatrix}.$$

Normal Forms

Given that there are two representations for each element, there are multiple normal forms that can be considered. Let $x = (m, w) \in \mathrm{SL}(2, \mathbb{Z})$. The *matrix normal form* is simply the "flattened" matrix, i.e., a vector in $\mathbb{Z}^4$:

$$f_m = \langle a, b, c, d \rangle.$$

In the presentation of $\mathrm{SL}(2,\mathbb{Z})$ provided above, every word can be written uniquely in its *word normal form* as

$$(-I_2)^a R^{i_0} S R^{i_1} S \cdots R^{i_{n-1}} S R^{i_n},$$

with I_2 the identity matrix, $a \in \{0,1\}$, and $i_j \not\equiv 0 \bmod 3$ for $0 < j < n$ [10].

Decision Tree and Random Forest Results

For the decision tree classifier we used the normalized matrix normal form as the feature vector, i.e., for a word $u = (m_u, w_u)$ we have

$$f_m = \frac{1}{\|m_u\|}\langle a, b, c, d\rangle,$$

and for a pair of words u, v with respective matrix representations m_u, m_v, we concatenate the two unit feature vectors together to form a feature vector for the conjugacy decision problem:

$$c_m = \langle f_m(m_u) \parallel f_m(m_v)\rangle$$

Below are the results for testing both the decision tree and random forest classifiers with various parameters:

Method, Split Criterion, Depth	**Accuracy**
Tree, Gini Impurity, No Depth Limit	95.80%
Tree, Gini Impurity, Depth Limit	95.25%
Tree, Entropy, No Depth Limit	96.26%
Tree, Entropy, Depth Limit	95.27%
Random Forest, Gini Impurity, No Depth Limit	97.16%
Random Forest, Gini Impurity, Depth Limit	95.42%
Random Forest, Entropy, No Depth Limit	**97.47%**
Random Forest, Entropy, Depth Limit	95.37%

Table 5.23: Decision Tree and Random Forest Results for $\mathrm{SL}(2,\mathbb{Z})$

The random forest classifier using entropy as the split criterion and with no depth limit performed the best, with an overall accuracy of 97.47%. The confusion matrix for this classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9634	366
	Non-Conjugate	141	9859

Table 5.24: Confusion Matrix for the Best Performing Tree Classifier of $\mathrm{SL}(2,\mathbb{Z})$

NTNN Results

For the NTNN classifier we are looking to train on discretely valued data, thus the previous feature vector of normalized matrix entries is not applicable. Attempting to use the *unnormalized* matrix normal form would be a poor choice, as the frequency distribution of the integral values that comprise the matrix entries is highly skewed. For instance, in the training set, we have the following relative frequencies f_i for the matrix entries:

Sample Class	$f_i = 1$	$f_i > 1$
Conjugate Pairs	99.64%	.36%
Non-Conjugate Pairs	99.46%	.64%

Thus in lieu of the matrix representation for an element we will use its word representation. However, the word normal form as denoted above does not have a fixed length. Consequently, we will use the Cayley subgraph features to transform each element to a fixed length feature vector. In particular, we will utilize the feature vector c_2 that was used for BS(1,2). Note however that for $\mathrm{SL}(2,\mathbb{Z})$ the dimension of c_2 is 40.

For $\mathrm{SL}(2,\mathbb{Z})$ the best parameters are $M = 10, P = 5$ using voting majority and $M = 50, P = 4$ using logarithm voting majority.

Group	M	P	Accuracy (Σ)	Accuracy (log)
$\mathrm{SL}(2,\mathbb{Z})$	10	3	50.00%	99.13%
$\mathrm{SL}(2,\mathbb{Z})$	10	4	74.03%	75.95%
$\mathrm{SL}(2,\mathbb{Z})$	10	5	**97.72%**	88.51%
$\mathrm{SL}(2,\mathbb{Z})$	20	3	77.90%	98.37%
$\mathrm{SL}(2,\mathbb{Z})$	20	4	96.88%	96.27%
$\mathrm{SL}(2,\mathbb{Z})$	20	5	97.11%	96.21%
$\mathrm{SL}(2,\mathbb{Z})$	30	3	71.19%	51.19%
$\mathrm{SL}(2,\mathbb{Z})$	30	4	64.85%	50.11%
$\mathrm{SL}(2,\mathbb{Z})$	30	5	97.21%	98.77%
$\mathrm{SL}(2,\mathbb{Z})$	50	3	74.88%	99.60%
$\mathrm{SL}(2,\mathbb{Z})$	50	4	71.33%	**99.81%**
$\mathrm{SL}(2,\mathbb{Z})$	50	5	97.64%	95.57%
$\mathrm{SL}(2,\mathbb{Z})$	100	3	75.56%	99.78%
$\mathrm{SL}(2,\mathbb{Z})$	100	4	68.19%	90.66%
$\mathrm{SL}(2,\mathbb{Z})$	100	5	68.68%	83.68%

Table 5.25: NTNN Results for $\mathrm{SL}(2,\mathbb{Z})$

The confusion matrix for the best performing NTNN classifier on the verification set S_v is below:

		Assigned	
		Conjugate	Non-Conjugate
True	Conjugate	9987	13
	Non-Conjugate	25	9975

Table 5.26: Confusion Matrix for the Best Performing NTNN Classifier of $\mathrm{SL}(2,\mathbb{Z})$

5.3 Evaluation and Analysis

5.3.1 Decision Tree and Random Forest Optimizations

In optimizing the performance of the decision tree-based classifiers, different combinations of tree depth limits and splitting criteria were considered. For nearly all test groups, using information gain (equivalently, greatest reduction in entropy) resulted in the most accurate classifier. Only for GMBS(2,3) did using Gini impurity result in a higher accuracy, and only by .1%.

For all groups tested, the random forest classifier performed better than a single decision tree, and again information gain resulted in the most accurate classification. Limiting the depth of the tree (or trees in the case of random forests) to $\log_2 N - 1$, where N is the total number of samples, slightly improved the results when using Gini impurity as the splitting criterion, but did not do so when using information gain.

The generalization error for random forests approaches zero as additional trees are included in the forest. For example, the table below lists the classification accuracy for random forest classifiers on the group $\mathcal{O} \rtimes U_{34}$ with different numbers of trees. Note the diminishing marginal increases in accuracy as the number of trees increases (as this is a stochastic process, increases in accuracy are not strictly monotonic):

# Trees	**Accuracy**
10	98.89%
15	99.17%
20	99.07%
30	99.20%
50	99.31%
100	99.39%
200	99.41%

Table 5.27: Accuracy of Random Forest Classifiers for $\mathcal{O} \rtimes U_{34}$ with Increasingly Large Forests

5.3.2 Accuracy with Respect to Word Length

The length of a word with respect to a generating set is a crucial measurement throughout group theory. Word length, rather than bit length, is the standard input size parameter for group-theoretic algorithms. As mentioned in section 2.1.7 the growth rate of a group, which depends on word length, can determine algebraic properties such as nilpotency. Recall that in non-commutatively cryptography, the word length corresponds to key size. Thus, it is important to consider how well our system performs with respect to the word length.

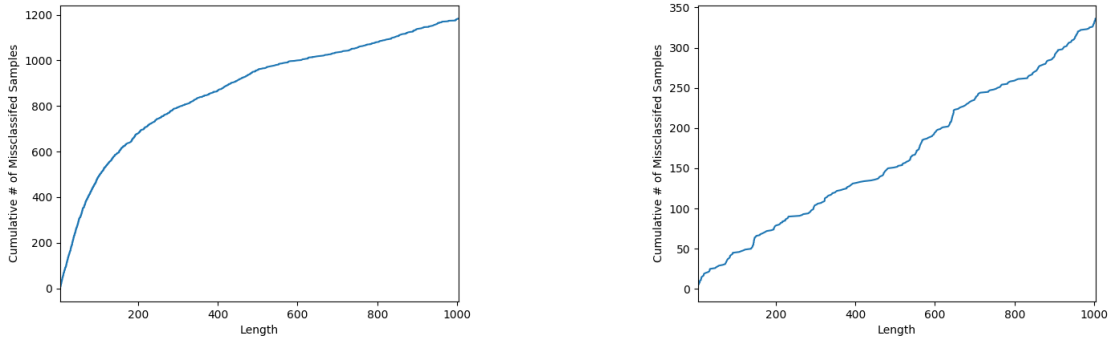
In analyzing the performance of our classifiers, we looked for a length threshold L that would provide the greatest difference in accuracy between words below and above this demarcation. To calculate L for each group, we first calculated the accuracy of the best performing NTNN classifier for each length and class over the verification dataset S_v. We then determined the inflection points in this data via second order finite differences. The threshold L was then set to the length that resulted in the greatest difference in accuracy. The results for each class and group are listed in the table below:

			Accuracy	
Group	Class	L	$\|w\| < L$	$\|w\| \geq L$
BS(1,2)	Conjugate	16	30.00%	88.82%
	Non-Conjugate	14	84.44%	96.75%
$\mathcal{O} \rtimes U_{14}$	Conjugate	10	94.00%	99.98%
	Non-Conjugate	11	80.00%	97.69%
$\mathcal{O} \rtimes U_{16}$	Conjugate	7	95.00%	99.51%
	Non-Conjugate	21	86.25%	97.59%
$\mathcal{O} \rtimes U_{34}$	Conjugate	7	55.00%	99.23%
	Non-Conjugate	36	97.74%	99.93%
GMBS(2,3)	Conjugate	17	88.33%	97.48%
	Non-Conjugate	9	100%	94.86%
$\mathrm{SL}(2,\mathbb{Z})$	Conjugate	17	90.83%	99.98%
	Non-Conjugate	7	90.00%	99.77%

Table 5.28: Accuracy with Respect to Word Length and Class for Tested Groups

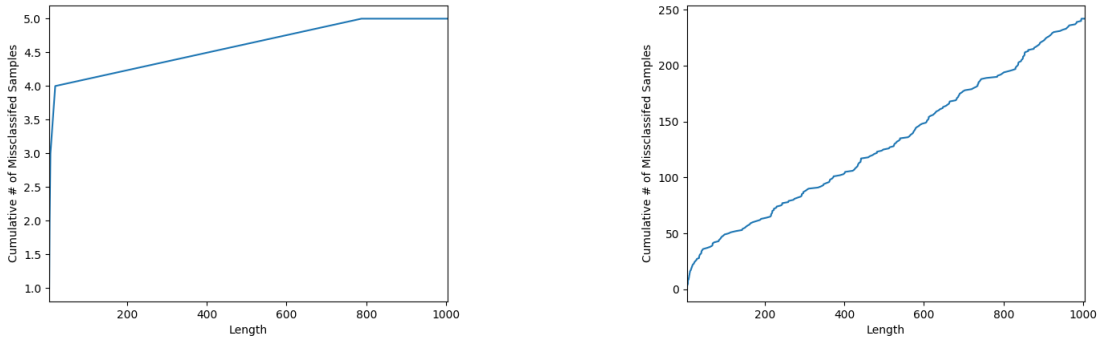
From the above table one can readily observe that classification is more accurate on longer words than shorter ones, with the only exception being the non-conjugate elements of GMBS(2,3). For BS(1,2) the classifier performed very poorly on short conjugate pairs. For the non-virtually nilpotent polycyclic groups, conjugate pair accuracy was over 99% for words of length greater than 10, while for non-conjugate pairs the length threshold required to achieve this performance level increased along with Hirsch length. The $\mathrm{SL}(2,\mathbb{Z})$ classifier performed very well in both classes with words greater than 17 in length.

Choosing a single length threshold for each group masks lesser changes in accuracy with respect to word length. To provide a fuller sense of the accuracy-length relationship, we present figures below that depict the cumulative rate of misclassification as word length increases. For each group we have two graphs, one for each class of elements. For non-conjugate elements the graphs display a roughly linear rate of misclassification with respect to increasing word length, while for conjugate elements the rates are less uniform. Note that in the graphs below, the x-axis ranges over the lengths of words in the verification set ($[5, 1004]$), while the y-axis range is dependent upon the cumulative error rate for each group and class:



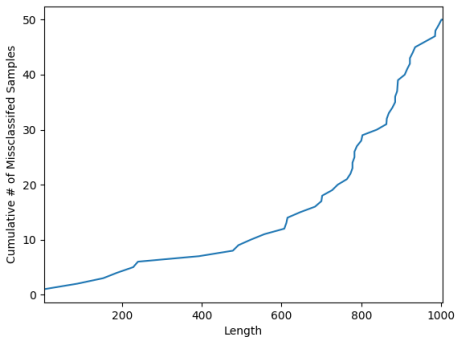
(a) Conjugate Elements (b) Non-Conjugate Elements

Figure 5.1: Cumulative Misclassifications by Word Length, BS(1,2)

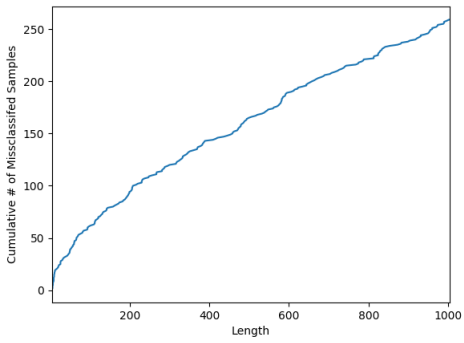


(a) Conjugate Elements (b) Non-Conjugate Elements

Figure 5.2: Cumulative Misclassifications by Word Length, $\mathcal{O} \rtimes U_{14}$

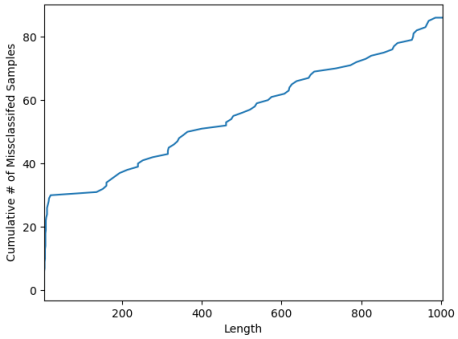


(a) Conjugate Elements

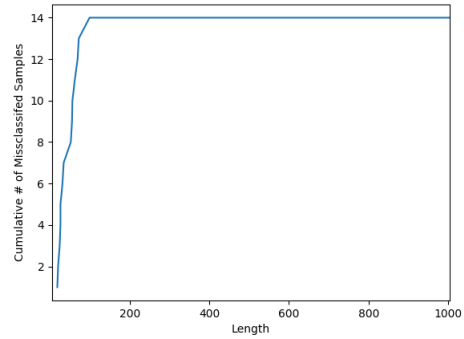


(b) Non-Conjugate Elements

Figure 5.3: Cumulative Misclassifications by Word Length, $\mathcal{O} \rtimes U_{16}$

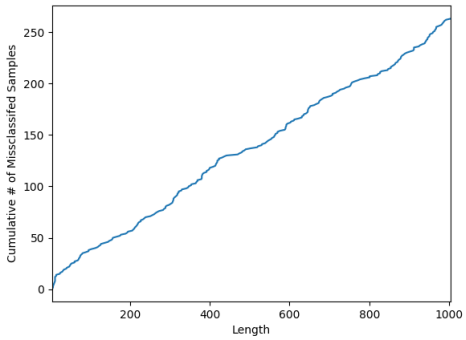


(a) Conjugate Elements

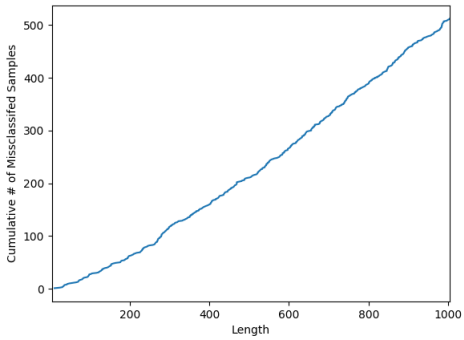


(b) Non-Conjugate Elements

Figure 5.4: Cumulative Misclassifications by Word Length, $\mathcal{O} \rtimes U_{34}$

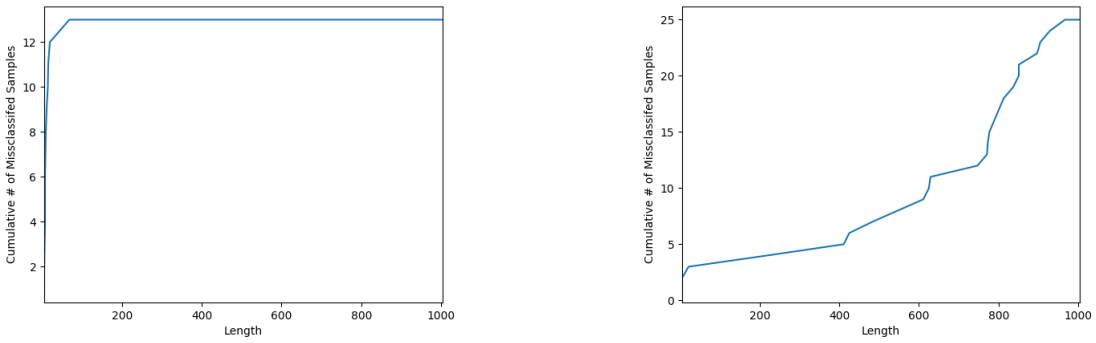


(a) Conjugate Elements



(b) Non-Conjugate Elements

Figure 5.5: Cumulative Misclassifications by Word Length, GMBS(2,3)



(a) Conjugate Elements (b) Non-Conjugate Elements

Figure 5.6: Cumulative Misclassifications by Word Length, SL$(2, \mathbb{Z})$

5.3.3 Accuracy with Respect to Class

By examining the confusion matrices for the best classifier for each group, we can observe the accuracy for each class of elements in our data set. The accuracies depicted in the table below are for the best performing NTNN classifier for each group, which was unanimously the best classifier for all groups. All classifiers achieved higher accuracy on the class of conjugate elements than the class of non-conjugate elements, with the exception of BS(1,2), which had the lowest accuracy results of all groups tested.

	Accuracy by Class	
Group	Conjugate	Non-Conjugate
BS(1,2)	88.17%	96.64%
$\mathcal{O} \rtimes U_{14}$	99.95%	97.58%
$\mathcal{O} \rtimes U_{16}$	99.50%	97.41%
$\mathcal{O} \rtimes U_{34}$	99.14%	99.86%
GMBS(2,3)	97.37%	94.88%
SL$(2, \mathbb{Z})$	99.87%	99.75%

Table 5.29: Accuracy by Class for Tested Groups

5.3.4 Visualizations for N-Tuple Neural Networks

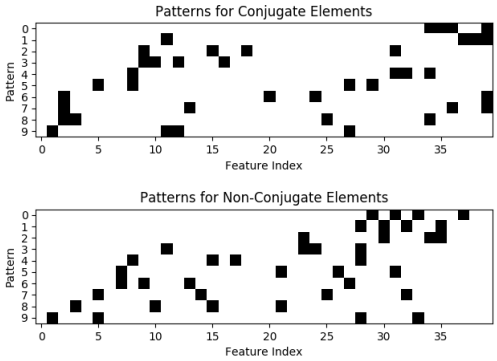
A visualization of a classifier can aid in the analysis of its performance. We present two different visualizations for the patterns used by the NTNN classifier. These visualizations, when used in sequence, can illuminate how the patterns change over the course of the optimization process. Examples for the NTNN classifier for $\mathrm{SL}(2,\mathbb{Z})$ are presented below. The classifier was trained using 10 patterns of size 4. The feature vector used was c_2, which is of dimension 40.

Pattern Grids and Heat Maps

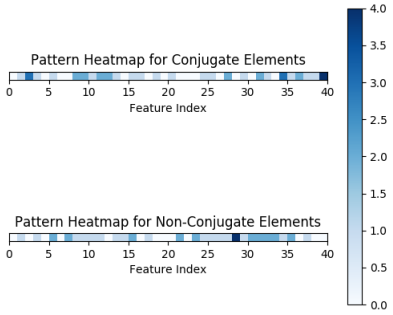
The left-hand graphic is a 10×40 (10 patterns, 40-dimensional feature vector) grid, with white and black squares representing 0 and 1 values in the pattern respectively. Note that the patterns and indices are indexed starting from 0. The patterns are sorted lexicographically as binary numbers. For example, if we had a set of patterns {111,101,011,010}, this set sorted lexicographically yields {010,011,101,111}.

The right-hand graphic is a 1-dimensional heat map of the patterns. Darker colored indices in the heat map correspond to more patterns sampling that component of the feature vector. The bar on the right provides a mapping of colors to sampling frequency.

In the figures below, there are two patterns grids on the left and two heat maps on the right, one for each class. The figures depict the classifier during the optimization process at accuracy levels of 74%, 84%, 94%, and 97%. One can see from the heat maps that at 97% accuracy, feature vector component 14 is the most heavily sampled for conjugate elements, while for non-conjugate elements component 22 is the most sampled.

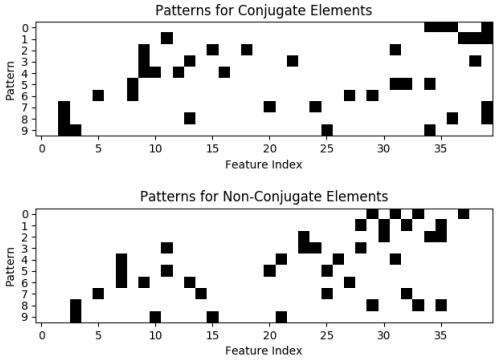


(a) Patterns

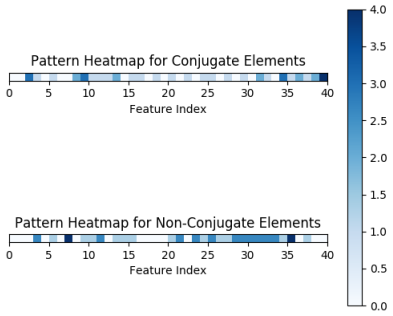


(b) Heat Map

Figure 5.7: Patterns and Heat Map for $\mathrm{SL}(2,\mathbb{Z})$, 74% Accuracy

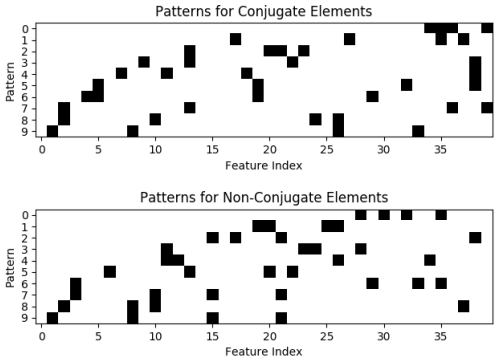


(a) Patterns

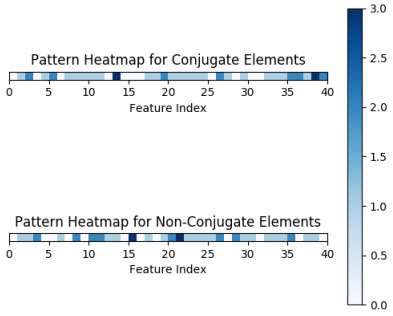


(b) Heat Map

Figure 5.8: Patterns and Heat Map for $\mathrm{SL}(2,\mathbb{Z})$, 84% Accuracy

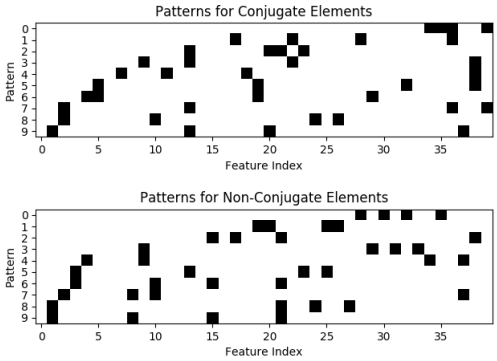


(a) Patterns

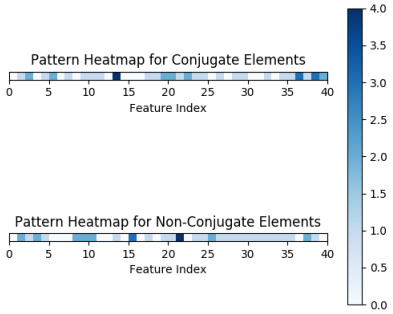


(b) Heat Map

Figure 5.9: Patterns and Heat Map for $\mathrm{SL}(2,\mathbb{Z})$, 94% Accuracy



(a) Patterns



(b) Heat Map

Figure 5.10: Patterns and Heat Map for $\mathrm{SL}(2,\mathbb{Z})$, 97% Accuracy

NTNN Optimization Progress

We can also use visualizations to evaluate the optimization progress of a NTNN classifier. As classifiers with different numbers of patterns and pattern sizes require different amounts of computation time, we used the percentage of patterns tested as our means of measuring progress. In general, the optimization algorithm tends to plateau between 95%-97% for most pattern parameters and groups. Depicted below is the optimization progress for the same NTNN classifier for $\mathrm{SL}(2,\mathbb{Z})$ as used in the visualizations above:

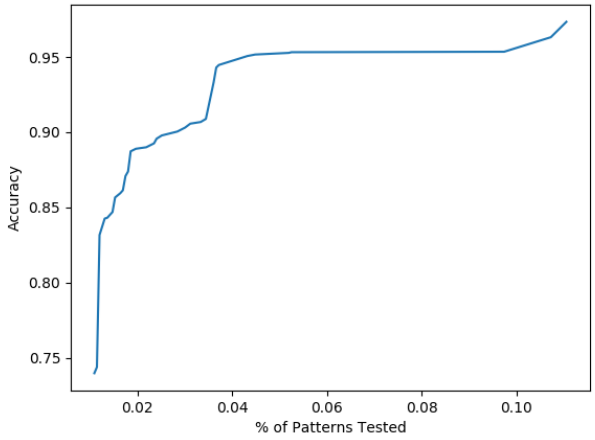


Figure 5.11: Optimization Progress for $\mathrm{SL}(2,\mathbb{Z})$ with 10 Patterns of Size 4

Chapter 6

Future Applications and Conclusion

Having shown that the conjugacy decision problem can be solved using our machine learning methods, we turn to the most natural subsequent application of our system: the conjugacy search problem. As the solution of the conjugacy search problem requires the production of a group element, we must perform regression analysis rather than classification, and will make use of the N-tuple regression networks of section 4.3.4.

Beyond this application, we present additional groups for which the performance of their trained machine learning systems is of immediate interest. We also suggest potential enhancements to our system with respect to feature extraction and model selection. Finally, we conclude with a discussion of the mathematical implications of our experimental results concerning the conjugacy problem, and more broadly on how we hope the methods described within this dissertation will be of use to the community at large.

6.1 Solving the Conjugacy Search Problem Using Machine Learning

Recall that the *conjugacy search problem* for a group G is to determine for any conjugate elements $x, y \in G$ if there exists a $z \in G$ such that $x^z = y$. The conjugacy search problem (CSP), and its high computational complexity in certain groups, is one of the central cryptographic hardness assumptions in non-commutative cryptography. Analogous to the conjugacy decision problem, algorithms for the search variant are often limited to particular groups and specific representations. For example, in braid groups the quotient attack [40] can solve the conjugacy search problem in time $O(n^2)$ (with respect to word length). For polycyclic groups given by their polycyclic presentations and with low Hirsch length, the algorithm of Eick and Ostheimer [14] can solve the CSP efficiently.

The success of length-based conjugacy search (LBCS) in braid groups, along with its strictly combinatorial nature, suggests that it may provide a more generally applicable solution for the conjugacy search problem. Unfortunately, the experimental results presented below show that this method does not yield the same level of success for other classes of finitely presented groups, particularly those that lack the generic free basis property.

Another potential approach is to utilize techniques from geometric group theory, specifically those that have been developed for hyperbolic groups. However, of the groups we are considering only $\mathrm{SL}(2, \mathbb{Z})$ is hyperbolic, and even relaxed forms of hyperbolicity are not satisfied by our non-virtually nilpotent polycyclic groups.

Having been stymied in our attempt to use known methodologies, we apply our machine learning system to the conjugacy search problem. For each platform group we train a N-tuple regression network (NTRN) that can produce a candidate conjugator for a pair of

group elements known to be conjugate. This candidate is then used as the initial state of a local search for a conjugator in the Cayley graph, in what we call regression-based conjugacy search.

The results in this section represent joint work with Robert Haralick and Delaram Kahrobaei.

6.1.1 Experimental Results for LBCS

We begin our exploration of the conjugacy search problem with a series of trial runs of LBCS over the same groups that were tested for the conjugacy decision problem. Tests were performed on a computer running Ubuntu 16.04 LTS with an Intel Core i7-4770K CPU. The GAP version used was 4.8.5 [1] with 9 GB of memory allowance. We utilized the algorithm "LBCS with Memory 2" as defined in section 2.2.3. In each test run for the group G, two random elements $x, z \in G$ in normal form were chosen whose lengths are in the range $[L_1, L_2]$ such that, for the element $y = x^z$, $|y| > |x|$. For each range of $[L_1, L_2]$ values, 50 tests were run with a timeout of 30 minutes per test.

BS(1,2) and Non-Virtually Nilpotent Polycyclic Groups

Group	$[10, 15]$	$[20, 23]$	$[40, 43]$
BS(1,2)	96%	90%	74%
$\mathcal{O} \rtimes U_{14}$	36%	12%	8%
$\mathcal{O} \rtimes U_{16}$	28%	2%	2%
$\mathcal{O} \rtimes U_{34}$	56%	8%	2%

Table 6.1: LBCS Results for BS(1,2) and Non-Virtually Nilpotent Polycyclic Groups

GMBS Groups and SL$(2, \mathbb{Z})$

Group	$[10, 15]$	$[20, 23]$	$[40, 43]$
GMBS(2,3)	96%	86%	54%
GMBS(3,5)	92%	86%	76%
GMBS(17,19)	56%	88%	56%
SL$(2, \mathbb{Z})$	78%	66%	62%

Table 6.2: LBCS Results for GMBS Groups and SL$(2, \mathbb{Z})$

In analyzing the above test results it is instructive to consider what makes length an effective heuristic for conjugacy search in some groups but not others. Hughes and Tannebaum suggested [26] using a length function $l(x)$, with the generic property that $l(yxy^{-1}) \geq l(x)$, for the purpose of selecting candidate solutions for the conjugacy search problem. Geodesic length, the length in the Cayley graph, was chosen by default for braid groups, as no more specific function was known. However, the optimized LBA in [38] worked well over braid groups, which are not free. This performance was further analyzed by Myasnikov and Ushakov in [40]. They found that the success of the LBA over non-free groups was due to such groups possessing the *generic free basis property*, in which a random choice of elements generates a free subgroup with high probability. The consequences of this property on the efficacy of LBCS are validated by the experimental results above. LBCS was much more successful in BS(1,2), the GMBS groups, and SL$(2, \mathbb{Z})$, which contain free subgroups, than in the tested polycyclic groups, which do not.

For many non-free groups, developments in geometric group theory have provided additional insight into the relationship between group elements and their geodesic representation in the Cayley graph. This is particularly true for δ-hyperbolic groups, introduced by Gromov [19]. A *δ-hyperbolic group* is a group whose Cayley graph is a δ-hyperbolic space, i.e., a space in which every triangle is "thinner" than in standard Euclidean space, thus implying that

the space is negatively curved. Hyperbolic groups have an efficiently solvable word problem, a polynomial time complexity conjugacy decision problem, and other nice group-theoretic properties.

In particular, hyperbolic groups have the topological property that they act *properly* on other metric spaces (i.e., standard hyperbolic space), so that compact sets in the mapped space have pre-images in the Cayley graph that are also compact. Results concerning hyperbolic groups have been extended to other groups by relaxing the requirements of hyperbolicity, producing classes of groups such as relatively hyperbolic groups and semi-hyperbolic groups. Polycyclic groups that are virtually abelian can be considered semi-hyperbolic, and act properly on $CAT(0)$ (Hadamard) spaces [8]. Unfortunately, the non-virtually nilpotent polycyclic groups under consideration are not virtually abelian. Therefore, rather than looking for new length functions, we will utilize supervised learning to determine potential conjugators from known conjugate pairs.

6.1.2 N-tuple Regression Networks and Conjugacy Search

N-tuple regression networks (NTRNs), as defined in section 4.3.4, can be used to estimate a candidate conjugator z from a given conjugate pair of elements $x, y \in G$. Once a NTRN has been trained and optimized for a particular group G, we can use the NTRN to provide an alternative means of performing conjugacy search. Rather than building a candidate conjugator from the identity element, we adapt the LBCS algorithm to attempt to produce a conjugator from an initial seed $\hat{z}$, the NTRN response to the feature vector of x and y. In this regression-based conjugacy search (RBCS), we perform local searches in the Cayley graph by permuting $\hat{z}$ via multiplication of positive and negative generators. Like LBCS, we impose a timeout on our search, and in local beam search fashion, maintain a set number T of shortest candidate conjugators after each iteration:

Algorithm 2 Regression-based Conjugacy Search

```
ẑ ← NTRN(FV(x, y))                    ▷ FV(x, y) is the feature vector of x and y
if ẑxẑ^{-1} = y then
    Return ẑ as a conjugator of x to y
else
    S ← {(|ẑ|, ẑ)}
    while not time-out do
        for (|z|, z) ∈ S do
            Remove (|z|, z)
            for h ∈ X, e = ±1 do
                for g ∈ {zh^e, h^e z} do
                    if gxg^{-1} = y then
                        Return g as a conjugator of x to y
                    else
                        Save (|g|, g) in the set S′
                    end if
                end for
            end for
        end for
        Sort potential conjugators in S′ by their length
        Copy the shortest T elements into S and delete the rest of S′
    end while
    Upon time-out, return FAIL
end if
```

6.1.3 NTRN Setup

We will train NTRNs with the same three data sets: S_i, S_o, and S_v, that were generated in Section 5.1.3. The set S_i will be used to train the NTRN as described in section 4.3.4. The set S_o will be used as in the NTNN classifier case to optimize the choice of pattern sets. The set S_v will be used for evaluation as described below.

6.1.4 Evaluation and Analysis

In evaluating the performance of the NTRN, we will utilize two different metrics, one for each type of network response. For the mean outputs (arithmetic and geometric) we will

calculate the average mean squared error (MSE) over the optimization set S_o. The closer the average MSE is to zero the more accurate the NTRN can be considered. For the median output, we will calculate the average mean absolute error (MAE) over the optimization set. Once the NTRN is trained to a sufficient level of accuracy, the performance of the RBCS algorithm will be evaluated by both its rate of success on the verification set S_v, as well as the speed at which it arrives at a solution relative to LBCS.

6.2 Additional Applications and Modifications

In this section we suggest additional groups that our machine learning systems can be applied to, as well some further means of enhancing the performance of our systems.

6.2.1 Additional Test Groups

While there are many groups to which our machine learning solutions can be applied, the two included below are of particular interest. Braid groups received considerable attention in non-commutative cryptography as the original platform group for the AAG key exchange. The group $\mathrm{PSL}(2,\mathbb{Z})$ is of interest as it is related to both the tested group $\mathrm{SL}(2,\mathbb{Z})$ and the braid group B_3.

Braid Groups

The *braid group on n strands* is given by the following presentation:

$$B_n = \left\langle x_1, \ldots, x_{n-1} \;\middle|\; \begin{array}{rl} x_i x_j x_i = x_j x_i x_j & \text{if } |i-j| = 1 \\ x_i x_j = x_j x_i & \text{if } |i-j| > 1 \end{array} \right\rangle.$$

Intuitively, elements of a braid groups represent the crossings of strings or braids that are non-trivial (i.e., they do not become uncrossed after pulling on both ends of a braid) and that do not result in knots. Braid groups have a number of normal forms, including

DeHornoy and Garside.

PSL$(2, \mathbb{Z})$

The *modular group* is the quotient group $\mathrm{SL}(2, \mathbb{Z})/\langle -I_2 \rangle$, where $-I_2$ is the additive inverse of the 2×2 identity matrix. The group is denoted by $\mathrm{PSL}(2, \mathbb{Z})$, as it is a member of the family of projective special linear groups. The group can be identified with the group of rational functions over $\mathbb{C}$ of the form $\frac{az+b}{cz+d}$, where $a, b, c, d \in \mathbb{Z}$ and $ad - bc = 1$. There are a number of presentations for $\mathrm{PSL}(2, \mathbb{Z})$, the one below is for the free product of $\mathbb{Z}_2$ and $\mathbb{Z}_3$:

$$\mathrm{PSL}(2, \mathbb{Z}) = \langle S, T \mid S^3 = 1, T^2 = 1 \rangle.$$

$\mathrm{PSL}(2, \mathbb{Z})$ is also isomorphic to the quotient of the braid group B_3 with its center.

6.2.2 Modifications to the Machine Learning System

Higher Order Features

All of the machine learning systems tested in the previous chapters rely on *first order* features, that is, features extracted directly from the training data. *Higher order* features are those extracted from feature vectors themselves. For instance, in the case of NTNNs, we can create an additional layer of pattern tables that would sample the pattern tables learned from the training set.

Decision Tree/NTNN Ensemble Method

For many applications the performance of a machine learning system can be improved by combining multiple learning models together into an ensemble method (e.g., the random forest model). We may be able to enhance the performance of our solution for the conjugacy decision problem by using decision tree, random forest, and NTNN classifiers together. The

output of an odd number of classifiers greater than 3 can be used to achieve a majority voting classification criterion, say by having two random forests classifiers and one NTNN. Another option would be to identify a commonality between the training samples that a classifier misclassifies, and use another type of classifier on these samples, so as to extract any remaining information out of the data.

6.3 Conclusion

In conclusion, we have shown how the pattern recognition techniques for free groups developed in [23] can be extended to non-free groups. We demonstrated that the conjugacy decision problem in a variety of groups can be solved with very high accuracy using random forests and n-tuple neural networks, and presented a framework for solving the conjugacy search problem in a similar fashion. We also introduced a family of metabelian groups that may serve as a potential platform group for non-commutative cryptography. Moreover, the relationship between the conjugacy search problem and the discrete logarithm problem in that family links the cryptographic hardness assumptions of non-commutative and number-theoretic cryptography.

As suggested in [23], the successful application of pattern recognition techniques to group-theoretic problems can provide experimental evidence for new conjectures in group theory. The decisions made by the decision trees and n-tuple neural network models used in this dissertation are readily interpretable, thus enabling a computational group theorist to link the results in the model back to their corresponding algebraic inputs.

We in fact have such a potential conjecture at hand. From the high accuracy of the classifiers across the tested groups it is apparent that there is some underlying mathematical relationship with respect to conjugacy that is responsible for the classifiers' performance.

We will make use of the visualizations and other methods of analysis presented within this dissertation to tease out what exactly this mathematical relationship is; a forthcoming paper will bring these additional results to light.

Our machine learning approach to solving the conjugacy search problem is a template for a general method of cryptanalysis that can be applied to other platform groups as they arise. Moreover, these methods can also be applied to cryptographic systems that utilize group-theoretic hardness assumptions other than the conjugacy search problem. For example, the general non-linear function interpolation of N-tuple neural networks may be applied to protocols that utilize the endomorphism search problem, such as the Grigoriev-Shpilrain authentication scheme [18].

Bibliography

[1] GAP – Groups, Algorithms, and Programming, Version 4.8.5. `http://www.gap-system.org`, 2016.

[2] Iris Anshel, Michael Anshel, and Dorian Goldfeld. An algebraic method for public-key cryptography. *Math. Res. Let.*, 6:287–291, 1999.

[3] Louis Auslander. On a problem of Philip Hall. *Annals of Mathematics*, 86(1):pp. 112–116, 1967.

[4] Laszlo Babai, Robert Beals, Jin-yi Cai, Gabor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proc. 3rd ACM-SIAM SODA (Symp. on Discrete Algorithms)*. Citeseer, 1996.

[5] Gilbert Baumslag, Frank B. Cannonito, and Derek J.S. Robinson. The algorithmic theory of finitely generated metabelian groups. *Transactions of the American Mathematical Society*, 344(2):629–648, 1994.

[6] Woodrow W. Bledsoe and Iben Browning. Pattern recognition and reading by machine. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, pages 225–232. ACM, 1959.

[7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] Martin R. Bridson and André Haefliger. *Metric Spaces of Non-positive Curvature*, volume 319. Springer-Verlag Berlin Heidelberg, 1999.

[9] Bren Cavallo and Delaram Kahrobaei. A polynomial time algorithm for the conjugacy problem in $\mathbb{Z}^n \rtimes \mathbb{Z}$. *Reports@ SCM*, 1(1), 2014.

[10] Keith Conrad. $\mathrm{SL}(2,\mathbb{Z})$. `http://www.math.uconn.edu/~kconrad/blurbs/grouptheory/SL(2,Z).pdf`.

[11] Max Dehn. Über unendliche diskontinuierliche gruppen. *Mathematische Annalen*, 71(1):116–144, 1911.

[12] Bettina Eick and Delaram Kahrobaei. Polycyclic groups: A new platform for cryptology? *arXiv preprint math/0411077*, 2004.

[13] Bettina Eick, Werner Nickel, and Max Horn. Polycyclic, computation with polycyclic groups, Version 2.11. `http://www.icm.tu-bs.de/ag_algebra/software/polycyclic/`, Mar 2013. Refereed GAP package.

[14] Bettina Eick and Gretchen Ostheimer. On the orbit-stabilizer problem for integral matrix actions of polycyclic groups. *Math. Comp.*, 72(243):1511–1529 (electronic), 2003.

[15] Edward Formanek. Conjugate separability in polycyclic groups. *Journal of Algebra*, 42(1):1–10, 1976.

[16] David Garber, Delaram Kahrobaei, and Ha T. Lam. Length-based attack for polycyclic groups. *Journal of Mathematical Cryptology, De Gruyter*, pages 33–44, 2015.

[17] David Garber, Shmuel Kaplan, Mina Teicher, Boaz Tsaban, and Uzi Vishne. Length-based conjugacy search in the braid group. *Contemp. Math. 418*, pages 75–87, 2006.

[18] Dima Grigoriev and Vladimir Shpilrain. Zero-knowledge authentication schemes from actions on graphs, groups, or rings. *Ann. Pure Appl. Logic*, 162:194–200, 2010.

[19] Mikhail Gromov. Asymptotic invariants of infinite groups: "geometric group theory, vol. 2" london math. soc. lecture note ser. no. 182, 1993.

[20] Jonathan Gryak and Delaram Kahrobaei. The status of polycyclic group-based cryptography: A survey and open problems. *Groups Complexity Cryptology*, 8(2):171–186, 2016.

[21] Jonathan Gryak, Delaram Kahrobaei, and Conchita Martinez-Perez. On the conjugacy problem in certain metabelian groups. *arXiv preprint arXiv:1610.06503*, 2016.

[22] Robert Haralick. N-tuple method. University Lecture, 2015.

[23] Robert Haralick, Alex D. Miasnikov, and Alexei G. Myasnikov. Pattern recognition approaches to solving combinatorial problems in free groups. *Computational and Experimental Group Theory: AMS-ASL Joint Special Session, Interactions Between Logic, Group Theory, and Computer Science, January 15-16, 2003, Baltimore, Maryland*, 349:197–213, 2004.

[24] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005.

[25] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, 1985.

[26] James Hughes and Allen Tannenbaum. Length-based attacks for certain group based encryption rewriting systems. *Workshop SECI02 Securite de la Communication sur Internet*, page 5, 2002.

[27] Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *SIAM J. Compt. 8*, 8(4):499–507, 1979.

[28] Ilya Kapovich, Alexei G. Myasnikov, Paul Schupp, and Vladimir Shpilrain. Generic-case complexity, decision problems in group theory, and random walks. *Journal of Algebra*, 264(2):665–694, 2003.

[29] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC Press, 2014.

[30] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. *Computational Problems in Abstract Algebra*, pages 263–297, 1970.

[31] Aleksander Kołcz and Nigel M. Allinson. Application of the CMAC input encoding scheme in the N-tuple approximation network. *IEE Proceedings-Computers and Digital Techniques*, 141(3):177–183, 1994.

[32] Aleksander Kołcz and Nigel M. Allinson. N-tuple regression network. *Neural networks*, 9(5):855–869, 1996.

[33] Charles R. Leedham-Green and Leonard H. Soicher. Collection from the left and other strategies. *J. Symbolic Comput.*, 9(5-6):665–675, 1990. Computational group theory, Part 1.

[34] Eddie Lo and Gretchen Ostheimer. A practical algorithm for finding matrix representations for polycyclic groups. *J. Symbolic Comput.*, 28(3):339–360, 1999.

[35] Igor Lysenok, Alexei G. Myasnikov, and Alexander Ushakov. The conjugacy problem in the Grigorchuk group is polynomial time decidable. *Groups, Geometry, and Dynamics*, 4(4):813–833, 2010.

[36] Anatoly Mal'cev. On homomorphisms onto finite groups. *Trans. Amer. Math. Soc*, 119:67–79, 1983.

[37] John Milnor. Growth of finitely generated solvable groups. *J. Differential Geom.*, 2(4):447–449, 1968.

[38] Alex D. Myasnikov and Alexander Ushakov. Length-based attack and braid groups: cryptanalysis of Anshel-Anshel-Goldfeld key-exchange protocol. *PKC 2007, LNCS 4450*, pages 76–88, 2007.

[39] Alexei G. Myasnikov, Vladimir Shpilrain, Alexander Ushakov, and Natalia Mosina. *Non-commutative cryptography and complexity of group-theoretic problems*, volume 177. American Mathematical Society Providence, RI, USA, 2011.

[40] Alexei G. Myasnikov and Alexander Ushakov. Random subgroups and analysis of the length-based and quotient attacks. *Journal of Mathematical Cryptology 2(1)*, pages 29–61, 2008.

[41] Gennady Andreevich Noskov. Conjugacy problem in metabelian groups. *Mathematical Notes*, 31(4):252–258, 1982.

[42] Petr Sergeevich Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Matematicheskogo Instituta imeni V.A. Steklova*, 44:3–143, 1955.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[44] Vladimir Remeslennikov. Conjugacy in polycyclic groups. *Algebra and Logic*, 8(6):404–411, 1969.

[45] Richard Rohwer and Michal Morciniec. The theoretical and experimental status of the n-tuple classifier. *Neural Networks*, 11(1):1–14, 1998.

[46] Mark V. Sapir, Victor Guba, and Mikhail Volkov. *Combinatorial Algebra: Syntax and Semantics*. Springer Monographs in Mathematics. Springer International Publishing, 2014.

[47] Paul Schupp. On Dehn's algorithm and the conjugacy problem. *Mathematische Annalen*, 178(2):119–130, 1968.

[48] Daniel Segal. Decidable properties of polycyclic groups. *Proc. London Math. Soc*, 3:61–497, 1990.

[49] Peter Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[50] Vladimir Shpilrain. Randomness and complexity in matrix groups. `http://www.sci.ccny.cuny.edu/~shpil/matrix_complex.pdf`, 2017.

[51] Vladimir Shpilrain and Alexander Ushakov. The conjugacy search problem in public key cryptography: unnecessary and insufficient. *Applicable Algebra in Engineering, Communication and Computing*, 17(3-4):285–289, 2006.

[52] Graham D. Tattersall, S. Foster, and Robert D. Johnston. Single-layer lookup perceptrons. In *IEE Proceedings F-Radar and Signal Processing*, volume 138, pages 46–54. IET, 1991.

[53] Armin Weiß. A logspace solution to the word and conjugacy problem of generalized baumslag-solitar groups. *Algebra and Computer Science*, 677:185, 2016.

[54] Joseph A. Wolf. Growth of finitely generated solvable groups and curvature of Riemannian manifolds. *Journal of Differential Geometry*, pages 421–446, 1968.

[55] Chee K. Yap. Lecture notes on Sylvester identity, Lecture X, linear systems. University Lecture, 1999.