Computing the Rank of Braids

Justin Meiners

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Mark Hughes, Chair
Curtis Kent
Stephen Humphries

Department of Mathematics

Brigham Young University

ABSTRACT

Computing the Rank of Braids

Justin Meiners
Department of Mathematics, BYU
Master of Science

We describe a method for computing rank (and determining quasipositivity) in the free group using dynamic programming. The algorithm is adapted to computing upper bounds on the rank for braids. We test our method on a table of knots by identifying quasipositive knots and calculating the ribbon genus. We consider the possibility that rank is not theoretically computable and prove some partial results that would classify its computational complexity. We then present a method for effectively brute force searching band presentations of small rank and conjugate length.

Keywords: braid group, free group, rank, quasipositive, algorithm, ribbon genus.

Contents

Contents **iii**

List of Figures **v**

1 Introduction **1**

1.1 Objectives 2

1.2 Motivation 3

2 Structure of Braids **5**

2.1 More about Rank 6

2.2 Lower bounds from Quotients 6

2.3 Symmetric Group 7

2.4 Exponential Sum 11

2.5 Quasipositivity 13

2.6 Upper Bounds from Quotients 14

2.7 Rank Parity 15

2.8 Examples of Computing Rank 16

3 Survey of Word Problem Algorithms For Braids **17**

3.1 Lawrence-Krammer Representation 18

3.2 Action on the Free Group. 20

3.3 Dehornoy Handle Reduction 22

4 Computing Rank in the Free Group **24**

4.1 Inductive Structure of Rank 24

4.2 Optimal Splitting and Conjugating 26

4.3 Dynamic Programming Algorithm 34

5 Computing Rank Upper Bounds in Braid Group **35**

5.1 Identifying Solvable Subwords 36

5.2 Identifying Conjugate Words 37

5.3 Commuting Words 38

5.4 Example of a Word it Does Not Solve 39

5.5 Testing Against KnotInfo Database 40

6 Brute Force Search Theory **41**

6.1 Decision Problems 41

6.2 The Naive Search 43

6.3 Comparison with Halting Problem 44

7 Practical Search with Templates **48**

7.1 Analysis of Naive Search 48

7.2 Enumerating Braids 49

7.3 Templated Search 50

7.4 Permutation Templates 52

7.5 Exponential Sum Templates 55

7.6 Signed Permutation Templates 57

7.7 Implementation Notes 57

7.8 Example of Computation 58

A Implementations of Selected Algorithms in Common Lisp **59**

A.1 Rank in the Free Group 60

A.2 Rank Upper Bound in the Braid Group 60

B Table of Computed Knot Results **62**

Bibliography **75**

List of Figures

1.1 Ribbon singularity 4

2.1 Topological braid in $D^2 \times I$ 8

2.2 Standard generator σ_1 in B_3 9

2.3 σ_1^2 in B_3 9

4.1 Expression tree for $abca^{-1}b$ 28

4.2 Associating two split nodes in an expression tree. 30

4.3 Enumeration of cases for Theorem 4.14 33

7.1 Graph of exponential sum speedup factor. 56

7.2 Graph of signed permutation speedup factor for $n = 5$ strands. 57

Chapter 1. Introduction

Definition 1.1. The **braid group** on n strands is defined by the following standard presentation:

$$B_n = \langle \sigma_1, \ldots \sigma_{n-1} \colon \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}, \sigma_j\sigma_i = \sigma_i\sigma_j \text{ whenever } |i-j| \geq 2 \rangle$$

. Each σ_i is referred to as a **positive standard generator**. Elements of B_n are called **(n-strand) braids**.

Definition 1.2. In the braid group a **positive band** is a conjugate of a positive standard generator. Thus a positive band is a braid in B_n of the form $\alpha\sigma_i\alpha^{-1}$ where α is any braid. A **negative band** is the inverse of a positive band.

Definition 1.3. A **braid word** w is an element of the free group on $n-1$ generators: $F_{n-1} = \langle \sigma_1, \ldots, \sigma_{n-1} \rangle$ (assumed to be reduced).

We can consider B_n as a quotient of F_{n-1}. Each *braid word* $w \in F_{n-1}$ is distinguished from its equivalence class $[w] \in B_n$ which is the *braid* it represents. When it is clear we are talking about words which represent braids, and the operations we are working with are well-defined on braids, the brackets will sometimes be dropped.

Definition 1.4. A **band presentation** for a braid x is a tuple of bands $(\alpha_1\delta_1\alpha_1^{-1}, \ldots, \alpha_k\delta_k\alpha_k^{-1}) \in B_n^k$ of any length $k \in \mathbb{N}$, such that $\prod_{i=1}^{k} \alpha_i\delta_i\alpha_i^{-1}$ where each $\delta_i \in \{\sigma_1^{\pm}, \ldots, \sigma_{n-1}^{\pm}\}$.

Every generator is a band, simply by conjugating by the empty word. Thus band presentations trivially exist for every braid.

Definition 1.5. The **rank** of a braid is the minimum number of bands occurring in any of its band presentations. Rank was introduced in [Rudolph, 1983]. Since band presentations always exist, the rank must be defined and we will denote it by a function $rk \colon B_n \to \mathbb{Z}_{\geq 0}$.

Whenever we write a braid $x \in B_n$ in the form $x = \prod_{i=1}^{k} \alpha_i \delta_i \alpha_i^{-1}$ then this implicitly gives us a band presentation which is the tuple of its terms.

Closely related to rank, and more well known is the notion of quasipositivity:

Definition 1.6. A braid is **quasipositive** if it has a band presentation consisting only of positive bands [Rudolph, 1983]. A braid is **quasinegative** if it has a band presentation consisting only of negative bands.

1.1 Objectives

With this preliminary understanding of the problem, we outline our main results. The purpose of this research is to develop methods for computing the rank of braids. For many braids, we will find that we are able to compute rank directly, using our tools. But, in general rank is difficult to compute. Our most general method settles for computing lower and upper bounds. This is useful as close bounds can often serve the same obstruction purposes as the real invariant, or give partial information.

One bounding method we discuss is solving an analogous rank problem in the free group. We introduce a new algorithm for doing so based on dynamic programming. This algorithm additionally solves the quasipositivity problem for free groups (see [Orevkov, 2004]). This method is specialized and extended for estimating rank in the braid group, providing a flexible framework for further additions.

After considering bounds we examine whether braid rank is in theory computable. We prove some partial results which could be used to determine its computability and classify its computational complexity.

Lastly, we develop a method for brute force searching for band presentations, which has practical runtime for short ranks and short conjugating words. Bounds are also relevant to the method for constraining the search space.

An additional research goal is to actually implement programs for computing rank. With this consideration in mind we will often discuss practical implementation and performance

concerns of various algorithms which we often do informally.

1.2 Motivation

We want to briefly give an overview of research questions which the rank problem relates to. To understand these references in detail requires further study beyond this work. These results are not relied upon.

Rank tells us several elementary properties of braids:

- It solves the word problem $rk(x) = 0 \Leftrightarrow x = e$
- It is an invariant of conjugacy classes.
- It tells us whether a braid is quasipositive. This can in turn help detect quasipositive knots.

These facts contain generally useful algebraic information, but of particular interest for us is the relationship between slice genus, ribbon genus, and rank. Rank can sometimes tell us about these otherwise difficult to compute invariants. We will give a brief of this relationship here.

Definition 1.7. A knot is a smooth embedding of the circle S^1 into the 3-sphere S^3. Knots are considered up to isotopy.

Definition 1.8. Consider the 3-sphere S^3 as the boundary of the 4-ball D^4. A **slice surface** for a knot $K \subseteq S^3$ is an orientable surface smoothly embedded in D^4 whose boundary is K.

Definition 1.9. The **slice genus** of a knot K is the minimum genus of a slice surface having K as it's boundary. The slice genus will be denoted by g_s.

Definition 1.10. A **ribbon surface** for a knot $K \subseteq S^3$ is an orientable surface immersed in S^3 whose boundary is K and whose singularities are all of *ribbon type*. Ribbon type singularities are "nice" self intersections as depicted in Figure 1.1. Additional details can be found in [Grigsby, 2018].

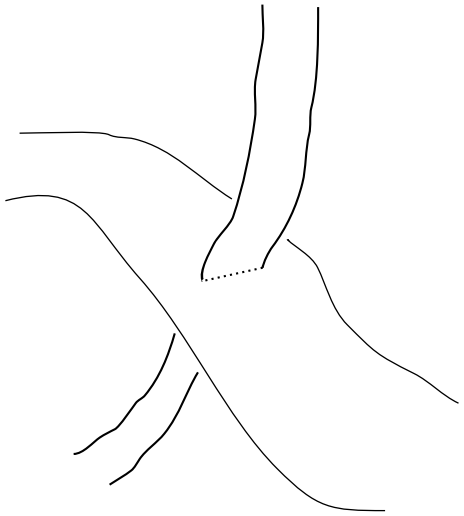

Figure 1.1: Ribbon singularity
The solid lines represent the boundary K of the ribbon surface. The dotted line is the self intersection lying within the surfaces interior.

Definition 1.11. The **ribbon genus** of a knot K is the minimum genus of a ribbon surface having K as it's boundary. Ribbon genus will be denoted by g_r.

Requiring the ribbon surface to be in S^3 limits the degrees of freedom of a ribbon surface in comparison to a slice surface. But requiring an immersion instead of an embedding simultaneously relaxes them. So we might wonder if either surface type requires a larger minimal genus than the other to compensate for either restriction. One relationship is clear:

Proposition 1.12. *For every ribbon surface of K of genus g there is slice surface for K having the same genus g.*

After pushing a ribbon surface into D^4, the ribbon singularities can be carefully removed (see Proposition 1.4 of [Rudolph, 1983]). Therefore

$$g_s(K) \leq g_r(K) \tag{1.1}$$

Proposition 1.13. *(Section 5 of [Rudolph, 1983]) Anytime K is represented by the closure of a braid $\widehat{x}$ and $x \in B_n$ has a band presentation: $(\alpha_1\delta_1\alpha_1^{-1}, \ldots, \alpha_k\delta_k\alpha_k^{-1})$, then a ribbon*

surface can be constructed for K by reading off the bands, whose Euler characteristic χ depends only on the number of bands k. Specifically: $\chi = n - k$.

The relationship between Euler the characteristic χ and the genus g of orientable surfaces is $g = \frac{2-\chi-C}{2}$ where C is the number of boundary components. For a knot, C is just 1. So finding band presentations with fewer bands yields smaller genus. Specifically:

$$g_r(K) \leq \frac{1 + rk(x) - n}{2}. \tag{1.2}$$

So rank can be used in a few important ways:

- If we don't know the slice genus or ribbon genus we can get an upper bound on it from the rank with (1.2).
- If we know the slice genus and the rank, then the ribbon genus must be between them because $g_s \leq g_r$ and g_r is bounded by (1.2).
- If we know (1.2) is equal to the slice genus, then we know the ribbon genus and slice genus are equal.

  In particular when $g_s = g_r = 0$, we can conclude that K has slice and ribbon surfaces which are both discs. Such knots have special names:

  Definition 1.14. A **slice knot** is a knot with slice genus 0.

  Definition 1.15. A **ribbon knot** is a knot with ribbon genus 0.

The last case provides a way to rule out potential counterexamples to the **Slice-Ribbon conjecture** which hypothesizes that every slice knot is also ribbon. Additional details about the relationship between rank and the Slice-Ribbon conjecture are described in [Grigsby, 2018].

Chapter 2. Structure of Braids

In this chapter we review basic background material on braid groups, including basic geometric and algebraic properties. It will differ from other surveys of this material by focusing on rank. Readers familiar with braids and quasipositivity may skip this chapter.

Whenever we wish to write a concrete braid we will use a more concise notation. Standard generators will be denoted with an integer, as in $\sigma_i = i$. A bar over top will be used to indicate inverse, so $\sigma_i^{-1} = \bar{i}$. For example, $\sigma_1\sigma_2\sigma_2^{-1}$ will be written as $12\bar{2}$.

2.1 More about Rank

Proposition 2.1. *[Rudolph, 1983] In the braid group every positive standard generator is in the same conjugacy class as σ_1.*

Proof. Assume this is true for the group B_{n+1} where $n \in \mathbb{N}$. The base case $n = 1$ is trivial. Then

$$\sigma_n\sigma_{n+1}\sigma_n\sigma_{n+1}^{-1}\sigma_n^{-1} = \sigma_{n+1}\sigma_n\sigma_{n+1}\sigma_{n+1}^{-1}\sigma_n^{-1} = \sigma_{n+1}.$$

So it holds for B_{n+2} and the result follows by induction. □

This fact tells us that only conjugates of $\sigma_1^{\pm 1}$ need to be considered when searching for band presentations. However, in practice we can shorten the conjugate lengths by using other generators. When we consider the analogous rank problem in other groups, this proposition typically does not hold.

2.2 Lower bounds from Quotients

Braid groups are very complex and can be difficult to study directly. However, questions of interest can often be answered by looking at quotients, which are abstractions of simpler structures in the braid group.

We will often use the same terms *band presentation* and *rank* in other groups besides the braid group. The definition for rank is analogous; the only ambiguity is in what constitutes a band.

Definition 2.2. Suppose ϕ is a surjective homomorphism $\phi\colon B_n \to H$. A **band** in H is a conjugate of the image of a standard generator. Thus a band is an element of the form $\alpha\phi(\delta)\alpha^{-1}$ where $\alpha \in H$ and $\delta \in \{\sigma_1^{\pm}, \ldots, \sigma_{n-1}^{\pm}\}$.

Definition 2.3. A **band presentation** for an element $y \in H$ is tuple of bands $(\alpha_1\phi(\delta_1)\alpha_1^{-1}, \ldots, \alpha_k\phi(\delta_k)\alpha_k^{-1}) \in H^k$, of any length $k \in \mathbb{N}$ such that $y = \prod_{i=1}^{k} \alpha_i\phi(\delta_i)\alpha_i^{-1}$.

Rank for another group H is denoted by a subscript rk_H. Note that rk_H also depends on the choice of homomorphism ϕ.

Proposition 2.4. *Suppose $\phi\colon B_n \to H$ is a surjective homomorphism and $x \in B_n$. Then $rk_H(\phi(x)) \leq rk(x)$.*

Proof. Suppose $\prod_i^k \alpha_i\sigma_i\alpha_i^{-1}$ and $k = rk(x)$. Then

$$\phi(x) = \phi(\prod_i^k \alpha_i\sigma_i\alpha_i^{-1}) = \prod_i^k \phi(\alpha_i)\phi(\sigma_i)\phi(\alpha_i)^{-1}.$$

Since each term in this product is a band in H, this gives a band presentation for $\phi(x)$ in H. Since rank in H gives the shortest such presentation, $rk_H(\phi(x)) \leq k$. □

To apply this fact, we want to find quotients where the rank can be computed easily.

2.3 Symmetric Group

Braid groups arise as an algebraic description of topological braids, which have nice geometric images that can give us clues about properties of rank.

Definition 2.5. A **topological braid on n strands** is a submanifold M with boundary of $D^2 \times I$ of dimension 1, having n components, and so that the projection map $\pi_I\colon M \times I \to I$ is a covering map. See Figure 2.1.

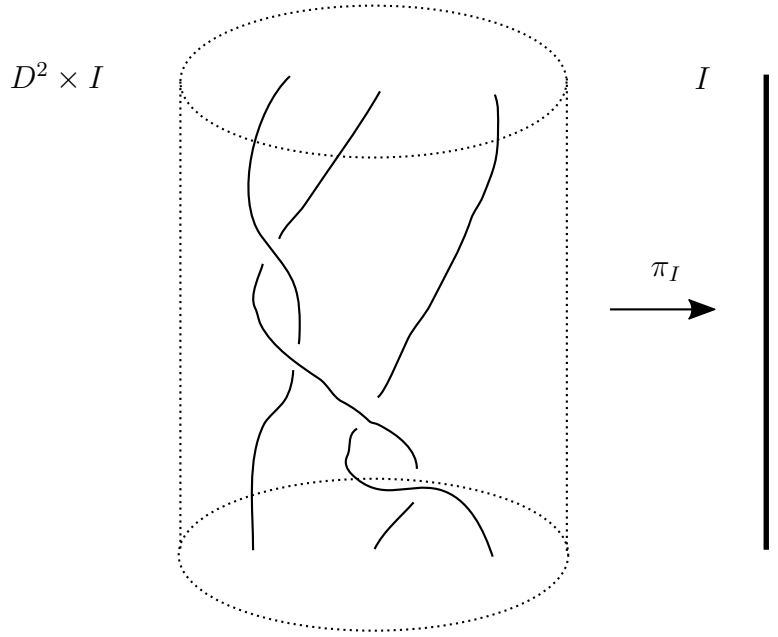

Figure 2.1: Topological braid in $D^2 \times I$

Elements of the braid group correspond to smooth isotopy classes of topological braids relative to the manifold boundary, where the number of strands is the index of the covering map.

Usually we think about this geometry a bit more intuitively. The cylinder is projected onto a 2D rectangle, from the side view. On the top of the rectangle the n strands are fixed with n distinct starting points. The strands move downward to distinct points on the bottom. As the strands actually lie in 3D, they occasionally cross, one passing over it's neighbor. We require strands to "monotonically" move downward, so they do not knot around themselves. Formally, the covering map π_I must be locally injective.

In this picture, a standard generator σ_i represents a crossings of the ith strand underneath the $(i+1)$th strand (see Figure 2.2). The inverse generator is a crossing of the same strands, but with the opposite layering; the ith strand crosses over the $(i+1)$th strand. Multiplying braids corresponds to concatenating these pictures.[1]

Based on this description, we can observe that generators have the effect of *permuting*

[1]We follow a right-to-left convention like function composition. The right-most generator in a braid word corresponds to the first crossing of the braid. But this has no implications for the algebra.

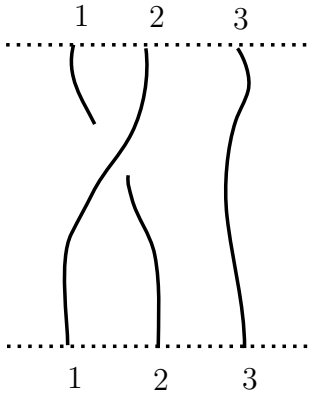

Figure 2.2: Standard generator σ_1 in B_3

the endpoints of a strand. Specifically, $\sigma_i^{\pm}$ permutes i and $i+1$. For an arbitrary braid we can determine the resulting permutation by tracing a strand from it's start point i, to it's end point j.

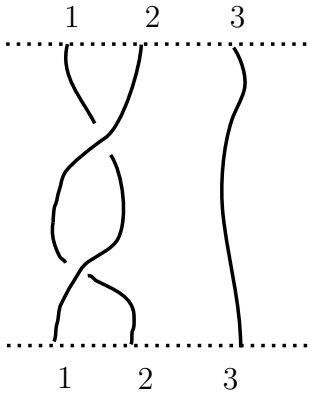

Figure 2.3: σ_1^2 in B_3

When a standard generator is repeated twice, it winds around itself, making the braid appear more complex, but the end point of the strands return to where they began. So the resulting permutation is trivial (see Figure 2.3).

If we form a quotient of the braid group including this relation for every generator, then this gives us the symmetric group S_n on n elements:

$$S_n \simeq B^n / \langle\langle \sigma_i^2 \text{ for all } i \in \{1, \ldots, n-1\} \rangle\rangle. \tag{2.1}$$

A homomorphism $\phi\colon B_n \to S_n$ can be determined by sending each generator to it's corre-

sponding transposition $\phi(\sigma_i) = (i\ i+1)$.

2.3.1 Computing rank. We would like to be able to compute the rank in the symmetric group. We first look at how bands behave under the homomorphism.

Lemma 2.6. *In the symmetric group, a conjugate of a transposition is a transposition.*

Proof. Let $(i,j), f \in S_n$ and consider $\varphi = f^{-1}(ij)f$. Pick $k \in \{1,\dots,n\}$. If $f(k) \notin \{i,j\}$ then $\varphi(k) = f^{-1}(ij)f(k) = f^{-1}f(k) = k$, so k is fixed. So the only elements that are permuted are $f^{-1}(i)$ and $f^{-1}(j)$. Because $\varphi(f^{-1}(i)) = f^{-1}(j)$, we conclude $\varphi = (f^{-1}(i)f^{-1}(j))$. □

Definition 2.7. A transposition of the form $(i\ i+1)$ is called an **adjacent transposition**.

Lemma 2.8. *(See exercise 3.5.3 of [Dummit and Foote, 2003].) Every transposition (ij) can be written as a conjugate of an adjacent transpositions by products of adjacent transpositions.*

Proof. Assume we can write $(i\ \ i+m) \in S_n$ as such a conjugate where $i+m<n$. The base case when $m=1$ is empty conjugation. Then $(i+m\ \ i+m+1)(i\ \ i+m)(i+m\ \ i+m+1) = (i\ \ i+m+1)$. □

Proposition 2.9. *The bands in the symmetric group are precisely the transpositions.*

Proof. The image of every band is a conjugate of an adjacent transposition, and thus a transposition. Specifically for $\alpha \in B_n$, $\phi(\alpha\sigma_i^{\epsilon}\alpha^{-1}) = \phi(\alpha)\phi(\sigma_i^{\epsilon})\phi(\alpha)^{-1} = \phi(\alpha)(i\ i+1)\phi(\alpha)^{-1}$. Furthermore, we can pick α so $\phi(\alpha)$ is an arbitrary product of adjacent transpositions. Thus the image of bands is surjective on transpositions. □

Proposition 2.10. *Suppose $\varphi = c_1 \cdots c_l$ is a decomposition of $\varphi \in S_n$ into disjoint cycles. Assume each c_i is a cycle of length l_i. Then $rk_{S_n}(\varphi) = \sum_{i=1}^{m} l_i - 1$.*

Proof. The number of bands required to write φ is the smallest number of transpositions needed to write φ. Each $c_i = (a_1 a_2 \dots a_{l_i})$ can be written minimally as a product of $l_i - 1$ transpositions as: $c_i = (a_1 a_{l_i}) \cdots (a_1 a_2)$. □

2.3.2 Example of braid whose rank is undetected. A braid can have tremendous complexity and still have a trivial symmetric group representation, as long as the strands eventually end up where they began. Braids having a trivial image (elements of the kernel) are called *pure braids*, and one can form a quotient of the braid group, which contains only braids of this form. For example $x = 1^{2k} \in B_3$ is Figure 2.3 repeated, and has rank as large as desired, but is a pure braid. So $rk_{S_3}(x) = 0$, but $rk(x) = 2k$.

2.4 Exponential Sum

We would like to find a quotient which captures the "winding" overlooked by the permutation representation. One way it can be measured is by simply counting up the number of generators, while considering signs. Positive generators increase winding in one direction. Negative generators decrease positive winding, or increase winding in the opposite direction.

For a braid word w define $\gamma(w)$ to be the sum of the exponents in the standard generators of w. So $\gamma(\sigma_i^\epsilon) = \epsilon$, and we extend to all words by defining $\gamma(wv) = \gamma(w) + \gamma(v)$.

Proposition 2.11. *The function $\gamma\colon B_n \to \mathbb{Z}$ is a well-defined surjective homomorphism.*

Proof. To see it is surjective observe $\gamma(\sigma_1^k) = k$ for any $k \in \mathbb{Z}$. We just need to show it is well defined in B_n. To see that γ is invariant under free reduction consider a word containing a canceling pair: $w_1 s_i s_i^{-1} w_2$ where each $w_j \in B_n$ and s_i is a generator.

$$\begin{aligned}
&\gamma(w_1 s_i s_i^{-1} w_2) \\
&= \gamma(w_1) + \gamma(s_i s_i^{-1}) + \gamma(w_2) \\
&= \gamma(w_1) + \gamma(w_2) \\
&= \gamma(w_1 w_2).
\end{aligned}$$

Also, γ is invariant under conjugacy:

$$\gamma(sws^{-1}) = \gamma(s) - \gamma(s) + \gamma(w) = \gamma(w).$$

So now suppose $w = v$ where w and v are braid words. Then $v^{-1}w$ is a word in the normal subgroup generated by the braid relations. So we just show $\gamma(r) = 0$ for each relation r.

- $\gamma(\sigma_i\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1}\sigma_i^{-1}\sigma_{i+1}^{-1}) = 0$
- $\gamma(\sigma_i\sigma_j\sigma_j^{-1}\sigma_i^{-1}) = 0$ where $|i - j| \geq 2$.

So $\gamma(v^{-1}w) = 0$. □

Definition 2.12. For a braid $x \in B_n$, $\gamma(x)$ is referred to as the **exponential sum**. Some sources refer to it as the **writhe**. $\gamma(B_n) \simeq \mathbb{Z}$ represents the abelianization of the braid group.

2.4.1 Computing rank. The exponential sum behaves conveniently on bands, so it turns out the rank in $\mathbb{Z}$ is just the value of ϕ, making it trivial to compute.

Proposition 2.13. *Bands have exponential sum* 1 *or* -1*. Furthermore, if* $\alpha\sigma_i^\epsilon\alpha^{-1}$ *is a band where* $\epsilon = \pm 1$*, the exponential sum is* ϵ*.*

Proof. $\phi(\alpha\sigma_i^\epsilon\alpha^{-1}) = \phi(\alpha) + \phi(\alpha)^{-1} + \phi(\sigma_i^\epsilon) = 0 + \epsilon$. □

Corollary 2.14. $rk_{\mathbb{Z}}(\phi(x)) = |\phi(x)|$ *for any* $x \in B_n$*.*

Proof. Since a band has rank ± 1, the rank of $k = \gamma(x)$ is the smallest j such that $k = \sum_{i=1}^{j} \pm 1$. If $k \geq 0$, then $k = \sum_{i=1}^{k} 1$ is minimal. If $k < 0$, then $k = \sum_{i=1}^{-k} -1$ is minimal. □

Corollary 2.15. *If* $x = \prod_{i=1}^{n} \alpha\sigma_{j_i}^{\epsilon_i}\alpha^{-1}$ *then* $\gamma(x) = \sum_{i=1}^{n} \epsilon_i$*.*

2.4.2 Example of braid whose rank is undetected. The following braid has trivial exponential sum but as large a rank as one wishes, simply by adjusting k. It can involve any number of generators depending on the number of strands (just stop the sequence when desired), but they are chosen to be index 2 apart:

$$\bar{1}^k 3^k \bar{5}^k 7^k \dots$$

Because the rank lower bounds provided by exponential sum and permutations detect very different features, they complement each other well in estimating the rank.

2.4.3 When exponential sum is a tight bound. It's worth examining when the inequality $|\gamma(x)| \leq rk(x)$ is strict. Let $x = \prod_{i=1}^{k} \alpha\sigma_{j_i}^{\epsilon_i}\alpha^{-1}$ where $k = rk(x)$. Then $\gamma(x) = \sum_{i=1}^{k} \epsilon_i$. Whenever every exponent is positive, this sum will be k and we will have equality (similarly if all the exponents are negative). It's only when there are "mixed signs", or in other words "canceling pairs" that we get inequality. This is actually a property of integers:

Lemma 2.16. *Suppose $m \in \mathbb{Z}$ is written as sum of n signed ones. So $m = \sum_{i=1}^{n} \epsilon_i$ where each $\epsilon_i = \pm 1$. Then the number of positive ones is $\frac{m+n}{2}$*

Proof. Let P denote the number of $+1$s in the sum and N denote the number of -1's in the sum. Then $m = P - N$ and $n = P + N$. So $m + n = 2P$. □

Proposition 2.17. *In any band presentation for $x \in B_n$ of length k, the number of positive bands is $\frac{k+\gamma(x)}{2}$.*

Proof. Suppose $x = \prod_{i=1}^{k} \alpha\sigma_{j_i}^{\epsilon_i}\alpha^{-1}$. In the sum $\gamma(x) = \sum_{i=1}^{k} \epsilon_i$ the number of positive ϵ_i's is the number of positive bands. So the result follows from the previous lemma. □

Define $Pb\colon B_n \to \mathbb{Z}_{\geq 0}$ to be the number of positive bands in any minimal band presentation of $x \in B_n$.

Corollary 2.18. *Pb is well defined and $Pb(x) = \frac{rk(x)+\gamma(x)}{2}$.*

Proof. Since γ and rk are well defined on braids, so is Pb. □

2.5 Quasipositivity

The case when the exponential sum is equal to the rank naturally leads to the concept of quasipositivity for braids. Rank is a more general property than quasipositivity, but quasipositivity for braids has been studied extensively, especially in its relation to knots.

Proposition 2.19. *A braid is quasipositive if and only if it's rank is equal to it's exponential sum.*

Proof. Suppose $x \in B_n$ is quasipositive so for some positive bands $x = \prod_i^k \alpha_i \sigma_i \alpha_i^{-1}$. Then $rk(x) \leq k$ as the product of positive bands also gives a band presentation of length k. However we also know $\gamma(x) \leq rk(x)$ and $\gamma(x) = k$ so $rk(x) = k$.

Suppose $rk(x) = k = \gamma(x)$. Then x has a band presentation of length k. By the previous corollary $2Pb(x) = \gamma(x) + rk(x)$, so $Pb(x) = k$. □

Corollary 2.20. *If a braid x is not-quasipositive than $rk(x) \geq 2 + \gamma(x)$.*

This corollary is worth observing because there are topological obstructions to a knot K being quasipositive. If any of these obstructions are found to hold, the lower bounds on the rank can be immediately improved for any braid representing K.[2]

2.6 Upper Bounds from Quotients

Similar to finding quotients of B_n we can also consider groups for which B_n is a quotient. Without considering the details of the group we can't define what the bands are, other than requiring their images to be bands in B_n.

Proposition 2.21. *Suppose $\psi \colon G \to B_n$ is a surjective homomorphism. Suppose $x \in B_n$ and $w = \prod_i^L h_i g_i h_i^{-1} \in G$ where $\phi(w) = x$ and each $\psi(g_i)$ is a standard generator in B_n. Then $rk(x) \leq L$.*

Proof. The image $\psi(w) = x = \prod_i^L \psi(h_i)\psi(g_i)\psi(h_i)^{-1}$ gives a band presentation for x, but may not be minimal. □

The natural candidate to apply this proposition to is the free group.

Definition 2.22. Suppose $F_n = \langle a_1, \ldots a_n \rangle$ so each generator a_i has been fixed. A **band** in the free group F_n is a conjugate of a free generator (positive or negative).

.

[2] KnotInfo[Livingston and Moore, 2021] cites a few examples, such as one based on the Homfly polynomial.

Definition 2.23. A **band presentation** for $h \in F_{n-1}$ is a tuple of bands $(g_1 s_1 g_1^{-1}, \ldots, g_k s_k g_k^{-1})$ such that $h = \prod_{i=1}^{k} g_i s_i g_i^{-1}$ where each s_i is a positive or negative generator.

However, computing the rank in the free group, is not as straightforward as it was in the symmetric group or the abelianization of B_n. In Chapter 4 we present a non-trivial algorithm for solving this problem.

Unlike computing lower bounds in the quotient, we don't actually need to solve the rank problem to get a bound. Any band decomposition, whether it's minimal or not, gives us an upper bound on the rank in the braid group. So, partial results can still be useful. Other useful groups could perhaps be formed by gradually adding in relations of the braid group to the free group.

2.7 Rank Parity

Corollary 2.24. *For a braid word $w \in F_{n-1}$, the parity of its length is the same as the parity of $rk([w])$ and $\gamma([w])$.*

Proof. We first apply Proposition 2.17: $rk([w]) + \gamma([w]) = 2P([w])$ so is even. Therefore $rk([w])$ must be odd whenever $\gamma([w])$ is odd. Similarly, $rk([w])$ must be even whenever $\gamma([w])$ is even.

To see the parity of length also matches we consider the homomorphism $\phi\colon B_n \to S_n$. Every permutation is even or odd. This is an invariant on the ways you can write a permutation as a product of transpositions. If the braid word is written as $w = \delta_1 \cdots \delta_n$ where each δ_i is a standard generator. Then $\phi([w]) = \phi(\delta_1) \cdots \phi(\delta_n)$ writes $\phi([w])$ as a product of n transpositions. □

Given lower and upper bounds on rank, this fact cuts the possible values for the rank in half.

2.8 Examples of Computing Rank

The theorems we have proved are simple and straightforward, but already they provide us with great insight and useful tools for computing rank. The following examples demonstrate how they can be combined to prove rank.

Example 2.25. The braid word $w = \bar{2}312$ has length 4 so $rk([w]) \in \{0, 2, 4\}$. The exponential sum is 2 so we can conclude $rk([w]) \geq 2$. We can see $(\bar{2}32)(\bar{2}12)$ is a band presentation so $rk([w]) = 2$.

Example 2.26. The braid word $w = 123\bar{1}$ has length 4 so $rk([w]) \in \{0, 2, 4\}$. The exponential sum is 2 so $rk([w]) \geq 2$.

$$(123\bar{2}\bar{1})(12\bar{1}) = (123\bar{1}) = (12\bar{1})(13\bar{1})$$

are two band presentations showing $rk([w]) = 2$.

This example reveals the following fact:

Proposition 2.27. *Minimal band presentations are not unique. Eg. if $x = ab = cd$ where a, b, c, d are bands, it's possible that $a \neq c$ and/or $b \neq d$.*

Proof. In the previous example: $(123\bar{2}\bar{1}) \neq (13\bar{1})$ as shown by the permutation representation:

$$\begin{aligned}
&\phi(123\bar{2}\bar{1})\phi(13\bar{1})^{-1} \\
&= (12)(23)(34)(23)(12) * (12)(34)(12) \\
&= (12)(23)(34)(23)(34)(12).
\end{aligned}$$

This is not the identity permutation because 1 maps to 4. □

We provide an additional counterexample which reveals why this is the case.

Example 2.28. The braid word $w = 121$ has length 3 so $rk([w]) \in \{1, 3\}$. The exponential sum is 3 so $rk([w]) = 3$. (1)(2)(1) is one possible band presentation. (2)(1)(2) is another due to the braid group relationship $212 = 121$. They are distinct because $2 \neq 1$.

Chapter 3. Survey of Word Problem Algorithms For Braids

In group theory, the word problem is to construct an algorithm to determine whether a given word in the group generators is equal to the identity. This can then be used to determine if two elements are equal. For x, y in a group, $x = y \Leftrightarrow y^{-1}x = e$. Although the word problem for groups with finite presentation is not in general solvable, it is solvable in the braid group. Having a fast solution to the word problem is an important building block for our algorithms so that we can identify band presentations and conjugations.

A variety of different methods exist, all with various strengths. To discuss their complexities we will need the following definition:

Definition 3.1. [Cormen et al., 2009] For a functions $f, g \colon \mathbb{R} \to \mathbb{R}$ we say f **has asymptotic upper bound** g if there exists constants $c, d \in \mathbb{R}$ so that $0 \leq f(x) \leq cg(x)$ for all $x \geq d$. The set of all functions that have asymptotic upper bound g is denoted $O(g(x))$.

Definition 3.2. For an input parameter n we say an algorithm is $O(g(n))$ if the number of steps required to execute the algorithm as a function of n is in $O(g(n))$. This is sometimes called the **big O** complexity of the algorithm.

However, there is wide gap between the existence of an algorithm in the theoretical sense, and having a practical implementation. Algorithms which optimize big O are useful for classifying complexity, but sometimes are slower or more cumbersome in practice.

In this chapter, we survey a few methods which are practical to implement and compare these characteristics informally and with a touch of big O analysis. The complexity results

mentioned here are relevant for Chapter 6. This chapter is also optional and can be referenced as needed. It is included to record and justify the specific methods we use in our algorithm. Additionally, we hope it can help bridge the gap between the theory and actual implementation of the code.

3.1 Lawrence-Krammer Representation

The Lawrence-Krammer representation allows braid generators to be transformed into matrices, and word concatenation to become matrix multiplication; operations which have straightforward computer implementations.

Proposition 3.3. *[Bigelow, 2002]. (This specific formula is from Chapter 4.2.) Consider the free module of rank* $\binom{n}{2}$ *over the ring* $\Lambda = \mathbb{Z}[q^{\pm 1}, t^{\pm 1}]$. *The basis for this module is denoted by the set* $\{e_{i,j} \colon 1 \leq i < j \leq n\}$. *The braid group* B_n *acts on this module in the following manner:*

$$\sigma_i * e_{j,k} = \begin{cases} e_{j,k} & i \notin \{j-1, j, k-1, k\} \\ qe_{i,k} + (q^2 - q)e_{i,j} + (1-q)e_{j,k} & i = j-1 \\ e_{j+1,k} & i = j \neq k-1 \\ qe_{j,i} + (1-q)e_{j,k} + (1-q)qte_{i,k} & i = k-1 \neq j \\ e_{j,k+1} & i = k \\ -tq^2 e_{j,k} & i = j = k-1. \end{cases}$$

This action is faithful. (This is a hard result.)

Faithfulness is necessary to be able to check if a word is the identity.

3.1.1 Method description. Implementing this action in a computer requires some translation. The ring Λ is a subring of the field $\mathbb{R}$, provided we choose q and t to be algebraically independent real numbers which are not integers. For example, one could choose

$q = \pi$ and $t = e^{\pi}$. For each σ_i the action it performs on vectors corresponds with a linear map in $GL(\binom{n}{2}, \mathbb{R})$ which has a matrix representation. The matrix for σ_i is cumbersome to write explicitly, but is constructed immediately from the action. Each column denotes the image of the basis element $e_{i,j}$ under σ_i. For each row the coefficients for the image can be read off from the action.

Algorithm 3.4.

Input: Braid word $w = s_1 \cdots s_m \in F_{n-1}$.

Output: Whether the word w represents the identity braid.

(i) Translate $\sigma_1, \ldots, \sigma_{n-1}$ to matrices $A_1, \ldots, A_{n-1}$.

(ii) Compute $A_1^{-1}, \ldots, A_{n-1}^{-1}$ by inverting matrices and store the results.

(iii) Send each s_i to it's corresponding matrix by sending $\sigma_i^{\pm} \to A_i^{\pm 1}$.

(iv) Multiply the chain of matrices and terminate. If the result is the identity matrix, then w is the identity braid.

3.1.2 Analysis.

Proposition 3.5. *The Lawrence-Krammer solution to the word problem is $O(ln^6)$ in the number of field operations where n is the number of strands and l is the length of the word w.*

Proof. Each generator is sent to a matrix of dimension $\binom{n}{2} = n(n-1)/2$. Standard matrix multiplication uses dim^3 field operations. So a multiplication is $n^3(n-1)^3/8$ operations. We apply this operation $l-1$ times to multiply the l generators. So the resulting polynomial $(l-1)n^3(n-1)^3/8$ is $O(ln^6)$. □

This analysis shows the method scales well in the length of the braid, but not in the number of strands. In addition, computing inverse matrices in the setup is slow due to the

large matrix dimension. This work can be cached and does not scale in length, so is not considered in the big O analysis.

In our implementation, we used straightforward matrix implementations. It's likely there are better numerical linear algebra solutions to this problem and these could strengthen this method.

An additional limitation is that most implementations introduce floating point calculation because we define q and t to be specific real numbers. So there is arithmetic error accumulation, and one must choose a tolerance level for checking whether a matrix is the identity. Alternatively, if the algorithm was implemented in a computer algebra system it could represent q and t symbolically. The matrix entries would then be rational functions of q and t with integer coefficients, but this introduces its own overhead.

3.2 Action on the Free Group.

Proposition 3.6. *[Bardakov and Bellingeri, 2014] The braid group B_n acts on the free group on n generators $F_n = \langle t_1, \ldots, t_n \rangle$ in the following manner:*

$$\sigma_i * t_j = \begin{cases} t_j & j \notin \{i, i+1\} \\ t_{i+1} & j = i \\ t_{i+1} t_i t_{i+1}^{-1} & j = i+1. \end{cases}$$

Furthermore, this action is faithful.

The action of σ_i^{-1} can be determined directly from this definition, although we will state it for completeness:

$$\sigma_i^{-1} * t_j = \begin{cases} t_j & j \notin \{i, i+1\} \\ t_i^{-1} t_{i+1} t_i & j = i \\ t_i & j = i+1. \end{cases}$$

Similar to Lawrence-Krammer, we can use these actions to transform the standard generators into maps on the free group. However, the action gives us a general automorphism of the group free which can't always be represented in a matrix. The principle is still the same, in that only identity elements give identity maps, but we must do more work to detect identity maps.

3.2.1 Method description. This procedure verifies that a given braid word acts as the identity element.

Algorithm 3.7.

Input: A braid word $w = s_1 \cdots s_m \in F_{n-1}$.

Output: Whether w is an identity braid.

(i) For each free generator $t \in \{t_1, \ldots, t_n\}$:

  (a) Define a free word $x = t$.

  (b) Apply the action of each generator s_i to x in order, replacing x with the result each time. (Braids are right to left, so the rightmost braid generator acts first.) Between each action application, free reduce x.

  (c) Verify that $x = t$. If not, the braid word is not the identity, so terminate.

(ii) If the algorithm has not previously terminated, every free generator maps to itself, so we have an identity braid.

The free reduction step is done to avoid having the intermediate free word x grow unnecessarily. Without this adjustment the action can potentially triple the length each time, so it grows rather quickly. But, most often free reduction helps curtail growth.

3.2.2 Analysis. Xu has found a family of braids for which free reduction doesn't help, and the runtime is exponential in it's length [Xu, 2011]. For an arbitrary braid the performance heavily depends on the probability of such braids appearing as subwords of the braid in question, as well as how long the offending pieces actually are.

Proposition 3.8. *The free group action method is $O(3^l n)$, where l is the length of the word and n is the number of strands.*

Proof. First consider a single generator $x = t$. We apply the action l times to x. Each application is linear in the length of x, and can at most triple the length of the output. So the length of x is at most $\sum_{i=1}^{l} 3^i = 3^{l+1}/2 - 3/2 \in O(3^l)$. Free reduction is linear in the length of x. We must apply this n times for the number of strands. So the total time is in $O(3^l n)$. □

Thus, this method has the opposite scaling characteristic as Lawrence-Krammer. It scales excellently in the number of strands, and poorly in the length. However, note that this is the absolute worst case. For words where the growth in length of x is only by a constant factor, the algorithm has complexity $O(l^2 n)$. In practice we found it to be faster than Lawrence-Krammer in most every case, especially for long words (which is counterintuitive).

3.3 Dehornoy Handle Reduction

Switching focus a bit, Dehornoy's handle reduction [Dehornoy, 1997] provides a family of algorithms which not only solve the word problem, but also compare braids. In fact, it gives a total ordering on each braid group B_n. Which is an essential prerequisite for sorting and sorted data-structures such as search trees. We will restate some results from [Dehornoy, 1997].

Definition 3.9. A braid word $w \in F_{n-1}$ is called **reduced** if w is the identity word, or if the generator with lowest index in w occurs only positively, or only negatively.

Proposition 3.10. *A nonempty reduced braid word is not the identity*

This appears plausible, but is not an easy result to show. This fact can be used as a fast obstruction to being the identity, and suggests an algorithm, try to write a word in reduced form.

Proposition 3.11. *For braids x, y we say $x < y$ whenever $b^{-1}a$ has a reduced form such that the generator of lowest index occurs only positively. Then the binary function $<$ is a total ordering for each braid group B_n.*

Definition 3.12. A **σ_j-handle** is a braid word of the form $\sigma_j^{\epsilon} w \sigma_j^{-\epsilon}$ where the word w contains only generators $\sigma_k^{\pm 1}$ where $k < j-1$ or $k > j$.

Handles have a particular geometry which allow them to be manipulated in a way that leads to reduction. This is expressed by the following homomorphism which can be applied to a handle:

$$\phi_{j,\epsilon}(\sigma_i^{\delta}) = \begin{cases} e & i = j \\ \sigma_{j+1}^{-\epsilon} \sigma_j^{\delta} \sigma_{j+1}^{\epsilon} & i = j+1 \\ \sigma_i^{\delta} & i \notin \{j, j+1\} \end{cases}$$

(j and ϵ are determined by the handle).

Notice that this homomorphism is similar to the braid group action on the free group, and it similarly can have problems with growth. In fact, there are some words that will grow infinitely if this is applied (see Chapter 1 of [Dehornoy, 1997] for an example). However, these cases can be avoided by being careful about reducing handles in a safe order.

3.3.1 Method description.

The following algorithm is one such method which reduces handles safely. It is guaranteed to terminate with a reduced word. Applying this algorithm with Proposition 3.11 gives a comparison algorithm. Combining with Proposition 3.10 gives an algorithm for solving the word problem.

Algorithm 3.13. (titled "FullHRed" in [Dehornoy, 1997])

Input: Braid word $w = s_1 \cdots s_m \in F_{n-1}$ to handle reduce.

Output: A *reduced* word w' such that $[w'] = [w]$.

(i) Find the leftmost handle of w. That is, the one that ends at the minimal possible position (from left to right). This handle can be specified by a start index k and a length l. If there are no more handles, terminate. The word is reduced.

(ii) Modify w by replacing the handle we found $s_k \cdots s_{k+l} = \sigma_j^{\epsilon} s_{l+1} \cdots s_{k+l-1} \sigma_j^{-\epsilon}$, with it's image under the homomorphism $\phi_{j,\epsilon}$.

(iii) Free reduce w. Go back to i.

The free reduction is included as an intermediate step to curtail growth, similar to the previous algorithm. Since it's difficult to predict how many handles there will be in a given braid word, and when new handles will be introduced while reducing, we will forgo a big O analysis.

As a word problem solution, this algorithm is very similar to the braid action on the free groups. In practice we found the latter to be faster for most purposes, but have not done a rigorous test. There are alternative ways of picking handles which may be more efficient (see Chapter 4 of [Dehornoy, 1997]). In our view, the primary reason to use this method is to get both linear ordering and a word problem solution, in one algorithm.

Chapter 4. Computing Rank in the Free Group

In this chapter we develop an efficient solution for computing rank in the free group based on dynamic programming. This method and it's proof of correctness are original. Because of Proposition 2.21. This immediately gives us an upper bound on the rank in the braid group. Later, this approach will be modified specifically for the braid group to compute closer upper bounds. In [Orevkov, 2004] Orekov describes an algorithm for detecting quasipositivity in the free group. Our method additionally solves this problem.

4.1 Inductive Structure of Rank

Algorithms are often designed using mathematical induction. A problem may at first appear intractable due to too many cases and variations. One first considers how to divide up a problem into smaller instances. Assuming those smaller instances can be solved, can they

be combined in a way to solve the a full problem? If so, then a solution is complete, just recursively combine smaller solutions down to a trivial base case. (See chapter 2.3 of [Cormen et al., 2009].)

We would like to apply this technique to computing rank in the free group: rk_F. But this is made difficult by the fact that rank is a somewhat non-local property (especially for braids). Naturally we might consider how the rank of a subword relates to the full word. But, adding or removing a single generator can add or remove an entire band.

If we weaken our desired result a bit, we can find an induction relationship between subwords, and the rank of the full word. This is a straightforward observation:

Proposition 4.1. *If $w, v \in F_n$ have band presentations: $w = \prod_{i=1}^{n} g_i s_i g_i^{-1}$ and $v = \prod_{i=1}^{m} h_i t_i h_i^{-1}$, then their product $(\prod_{i=1}^{n} g_i s_i g_i^{-1})(\prod_{i=1}^{m} h_i t_i h_i^{-1})$ gives a band presentation for wv.*

Corollary 4.2. *If $w, v \in F_n$, then $rk_F(wv) \leq rk_F(w) + rk_F(v)$.*

Proof. Appending minimal band presentations for w and v gives a band presentation for wv, but it may not be minimal. □

On it's own, this doesn't tell us much. Applying this relationship recursively down to each individual generators would just give the length of the word. But if there are subwords for which the rank be improved or solved, then those improvements can be applied to the larger word. An obvious case is when a subword is a conjugated.

Proposition 4.3. *For any $v, w \in F_n$, $rk_F(vwv^{-1}) = rk_F(w)$.*

Proof. Suppose $w = \prod_{i=1}^{k} g_i s_i g_i^{-1}$ and $rk_F(w) = k$. Then

$$vwv^{-1} = v(\prod_{i=1}^{k} g_i s_i g_i^{-1})v^{-1} = \prod_{i=1}^{k} v g_i s_i g_i^{-1} v^{-1}$$

as the middle terms $v^{-1}v$ cancel in the product. This gives a band presentation for vwv^{-1}, but it may not be minimal. So $rk_F(vwv^{-1}) \leq rk_F(w)$.

Now suppose $vwv^{-1} = \prod_{i=1}^{m} g_i s_i g_i^{-1}$ and $m = rk_F(vwv^{-1})$. Then similarly

$$w = v^{-1}(\prod_{i=1}^{m} g_i s_i g_i^{-1})v = \prod_{i=1}^{m} v^{-1} g_i s_i g_i^{-1} v.$$

gives a band presentation, but may not be minimal. So $rk_F(w) \leq rk_F(vwv^{-1})$. □

Normally this relationship can only be applied to conjugates of entire words. But, combined with the splitting relationship (Corollary 4.2) it can apply to subwords.

Example 4.4. $abba^{-1}b$ clearly contains a conjugate subword and can be split into pieces: $rk_F(abba^{-1}b) \leq rk_F(abba^{-1}) + rk_F(b) = rk_F(bb) + rk_F(b) \leq 2 + 1 = 3$.

Without a closer examination we don't know that this upper bound is the best we could have found. How do we know there wasn't a better way to split up the word? What if there are conjugate subwords that overlap: $a^{-1} \cdots b \cdots a \cdots b^{-1}$?

However, we claim that the rank can be computed simply by finding the optimal way to apply these relationships. In other words, if we choose all the right splits and conjugates, the resulting upper bound is actually equal to the rank. This next section makes this precise and provides a proof of its correctness.

4.2 Optimal Splitting and Conjugating

Definition 4.5. Let W_n be the set of finite strings on a set of n letters $\{t_1, \ldots, t_n\}$ and their inverses $\{t_1^{-1}, \ldots, t_n^{-1}\}$ including the empty string. Members of W_n are called **free words**.

Forgetting the group structure on F_n, we can consider F_n as a subset of W_n. There is a well defined function $fr\colon W_n \to F_n$ taking w to its free reduced word $fr(w)$

Let $w = s_1 s_2 \cdots s_m \in W_n$. and define mutually recursive functions $C\colon W_n \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$ and $\rho\colon W_n \to \mathbb{Z}_{\geq 0}$ by

$$C(w) = \begin{cases} \rho(s_2 \cdots s_{m-1}) & s_1 = s_m^{-1} \\ \infty & \text{otherwise} \end{cases} \tag{4.1}$$

$$\rho(w) = \begin{cases} m & m \leq 1 \\ \min(C(w), \min_{1<i<m}(\rho(s_1 \cdots s_{i-1}) + \rho(s_i \cdots s_m))) & \text{otherwise.} \end{cases} \tag{4.2}$$

C is defined separately from ρ to indicate when a conjugation application has been chosen in proofs.

Proposition 4.6. *For any free word $w \in W_n$, ρ gives an upper bound on the rank. That is $rk_F(fr(w)) \leq \rho(w)$.*

Proof. ρ and C terminate for all inputs because each recursive call reduces the length of the input by one or more.

For a single generator s, $\rho(s) = 1$. For larger words ρ does nothing more than apply splitting (Proposition 4.2) and conjugating (Proposition 4.3) in a structured order. Each of these relationships preserves upper bounds. □

Definition 4.7. An **expression tree for** $w \in W_n$ is a tree where each node is labeled with a free word in W_n, satisfying the following constraints:

- The root node is labeled w.
- If a node is labeled v and length of v is 1 or 0. then it must be a leaf.
- Otherwise a node labeled v must have exactly one or two children.
  - If v has one child v' then $v = sv's^{-1}$ for some generator s. v is called a **conjugate node**.
  - If v has two children w_1, w_2 then $w = w_1w_2$ and w_1 and w_2 must not be empty words. v is called a **split node**.

(See Figure 4.2)

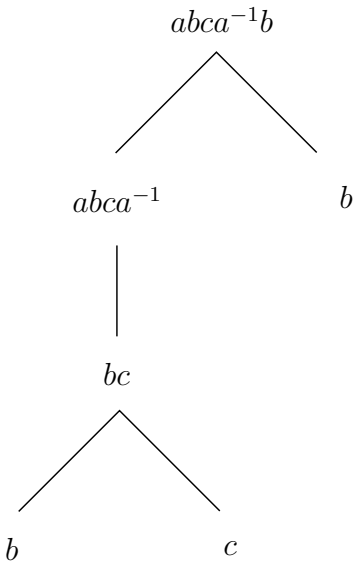

Figure 4.1: Expression tree for $abca^{-1}b$

Expression trees represent potential evaluation paths for ρ where each node corresponds with a recursive call (this will be made formal). We will use them to describe how ρ responds to changes in its input.

Let $\mathcal{T}_n$ be the set of all expression trees for words in W_n. Then define $L\colon \mathcal{T}_n \to \mathbb{Z}_{\geq 0}$ so $L(T)$ is the number of leaves in the tree T with nonempty labels.

Definition 4.8. An expression tree T for w is called **optimal** if for any other expression tree T' for w, $L(T) \leq L(T')$.

Proposition 4.9. *Suppose $w = w_1 v w_2 \in W_n$ and $w' = w_1 v' w_2 \in W_n$ and v' has an expression tree U'. If T is an expression tree for w and v is a node with subtree U. Then we can form an expression tree T' for w' where $L(T) - L(T') = L(U') - L(U)$.*

Proof. Form T' by replacing the subtree U with v's tree U'. Let p be the parent for v'. Suppose p is a split node. Then $p = w_1 v$ or $p = w_1 v$ and the other child for p is w_1. Assuming v' has nonzero length, replace p with $w_1 v'$ or $v' w_1$, and p is now a valid node. Otherwise v' is an empty string, remove the split node p and replace it with its child w_1. Suppose p is a conjugate node. Then $p = svs^{-1}$. Replace $p = sv's^{-1}$ and p is now a valid node.

We can continue similar replacements for every ancestor of v' to form T'. The resulting tree is an expression tree for w'. All leaves were unchanged except those in U were replaced with those in U'. Therefore $L(T) - L(T') = L(U') - L(U)$. □

Proposition 4.10. *If T is an expression tree for $w \in W_n$. Then T is optimal if and only if for every node v in T and subtree U rooted at v, $\rho(v) = L(U)$.*

Proof. If v is leaf, then $\rho(v) = L(v) = |v|$. If v is a split node, then $L(v) = L(w_1) + L(w_2)$. If v is a conjugate node with child v' then $L(v) = L(v')$. ρ is defined by the same recursive relationships as L, but is defined to be the minimal such way to apply them. So $L(T) = \rho(w)$ if and only if T is optimal.

To see it holds for any node v, let U be the subtree rooted at v. If U is optimal, then $L(U) = \rho(v)$. If U is not optimal say $L(U') \leq U$ for some other expression tree U' for v, then by the previous proposition we can form a tree T' for w where $L(T) - L(T') \geq 0$, contradicting that T was optimal. □

Corollary 4.11. *Suppose w_1vw_2 has an optimal expression tree T containing v as a node and $\rho(v') \leq \rho(v)$. Then $\rho(w_1v'w_2) \leq \rho(w_1vw_2)$.*

Proof. Let U' be an optimal tree for v' and U be the subtree of T rooted at v. Then $L(U') \leq L(U)$. So we can form a new expression tree T' by replacing the subtree rooted at v with U and relabeling as in (Proposition 4.9). The result is an expression tree T' for $w_1v'w_2$ where $L(T') \leq L(T)$. so $\rho(w_1v'w_2) \leq \rho(w_1vw_2)$ □

Proposition 4.12. *Suppose T is an optimal tree for $w = \cdots v_1v_2 \cdots$ containing v_1 and v_2 as nodes. Then there exists an optimal tree T' for w with a split node v_1v_2 and it's children are v_1 and v_2.*

Proof. First note that split nodes can be associated. If uvw has children uv and w and uv is a split node with children u and v. Then we can make a new expression tree by letting uvw have children u and vw and making children of vw v and w. Any nodes descending from u, v, and w are unchanged.

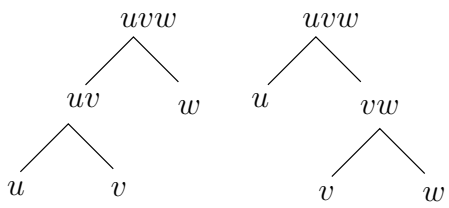

Figure 4.2: Associating two split nodes in an expression tree.

Every ancestor of v_1 contains v_1 as a subword. So the first common ancestor v of v_1 and v_2 is labeled $w_1v_1v_2w_2$. If w_1 and w_2 are empty, then we are done.

Otherwise let p be the parent of v_1. It cannot be a conjugate node, otherwise it's parent would not contain v_1. If it is a split node, it must be of the form u_1v_1 where u_1 is a subword of w, otherwise v_1 and v_2 would not be adjacent in the original word. Similarly the parent of v_2 is v_2u_1'. Continuing up each ancestor to v we find $w_1 = u_k \cdots u_1$ and $w_2 = u_1' \cdots u_l'$ where each u_i and u_i' appears as a node.

By repeatedly associating these split nodes we can replace T with a tree T' where $L(T) = L(T')$ and the children of v are $u_k \cdots u_1$ and $v_1v_2u_1' \cdots u_k'$ and the children of $v_1v_2u_1' \cdots u_l'$ are v_1v_2 and $u_1' \cdots u_k'$. □

Proposition 4.13. *For a word $w = s_1 \cdots s_n \in W_n$ and an expression tree T, each letter s_i (the specific letter at that index) appears in exactly one leaf node, or it appears at the beginning or end of exactly one conjugate node: $s_iws_i^{-1}$ (or swap s_i and s_i^{-1})*

Proof. Every letter in a split node appears in one of its children. So the letters which are not removed by conjugate nodes will appear as leaves. □

Theorem 4.14. *Suppose $w \in W_n$ and w' is the same word free reduced, then $\rho(w') \leq \rho(w)$.*

Proof. Suppose $w = s_1 \cdots s_is_{i+1} \cdots s_n$ and $s_i = s_{i+1}^{-1}$. Let $w' = s_1 \cdots s_{i-1}s_{i+2} \cdots s_n$ It suffices to show $\rho(w') \leq \rho(w)$ as free reduction is just a finite repetition of removing canceling pairs.

Let T be an optimal expression tree for w. Using Proposition 4.13 we will enumerate all possible ways s_i and s_{i+1} can appear in T. From their location we will infer that T contains a node of interest, $v = w_1s_is_{i+1}w_2$ containing the canceling pair (where each $w_i \in W_n$). Let $v' = w_1w_2$. If we can show $\rho(v') \leq \rho(v)$ then by Corollary 4.11, $\rho(w') \leq \rho(w)$.

Case 1 s_i and s_{i+1} are removed by the same conjugate node. Then $v = s_i s_{i+1}$ and

$$\rho(v) = C(s_i s_{i+1}) = 0 = \rho(v').$$

Case 2 s_i and s_{i+1} are removed by distinct conjugate nodes, and neither is a descendant of the other. Then T contains $s_i^{-1} w_1 s_i$ and $s_{i+1} w_2 s_{i+1}^{-1}$. By Proposition 4.12 $v = s_i^{-1} w_1 s_i s_{i+1} w_2 s_{i+1}^{-1}$ and

$$\rho(v) = C(s_i^{-1} w_1 s_i) + C(s_{i+1} w_2 s_{i+1}^{-1}) = \rho(w_1) + \rho(w_2).$$

After reduction:

$$\begin{aligned} \rho(v') &\leq \rho(s_i^{-1} w_1 w_2 s_{i+1}^{-1}) \\ &\leq C(s_i^{-1} w_1 w_2 s_{i+1}^{-1}) \\ &= \rho(w_1 w_2) \\ &\leq \rho(w_1) + \rho(w_2). \end{aligned}$$

Case 3: s_i and s_{i+1} are both leaves. By Proposition 4.12 $v = s_i s_{i+1}$ and

$$\rho(v) = \rho(s_i) + \rho(s_{i+1}) = 2.$$

But, $\rho(v) \leq C(s_i s_{i+1}) = 0$. contradicting that T is an optimal tree. So this case cannot happen.

Case 4: s_i is a leaf, s_{i+1} is removed by a conjugate node, and s_i is not a descendant of s_{i+1}. By Proposition 4.12 $v = s_i s_{i+1} w_1 s_{i+1}^{-1}$ and

$$\rho(v) = \rho(s_i) + C(s_{i+1} w_1 s_{i+1}^{-1}) = 1 + \rho(w_1).$$

After reduction:

$$\rho(v') = \rho(w_1 s_{i+1}^{-1}) \leq \rho(w_1) + \rho(s_{i+1}^{-1}) = \rho(w_1) + 1.$$

Case 5: s_{i+1} is a leaf, s_i is removed by a conjugate node, and s_{i+1} is not its descendant. Then $v = s_i^{-1} w_1 s_i s_{i+1}$ and the proof is similar to case 4.

Case 6: s_{i+1} is removed by a conjugate node and and s_i is its descendant. Then $v = s_{i+1}^{-1} w_1 s_i s_{i+1}$ and

$$\rho(v) = C(s_{i+1}^{-1} w_1 s_i s_{i+1}) = \rho(w_1 s_i).$$

Subcase A: s_i is a leaf. Therefore:

$$\rho(v) = \rho(w_1) + \rho(s_i) = \rho(w_1) + 1$$

After reduction:

$$\rho(v') = \rho(s_{i+1}^{-1} w_1) \leq \rho(s_{i+1}^{-1}) + \rho(w_1) = 1 + \rho(w_1).$$

Subcase B: s_i is removed by a conjugate node. $v = s_{i+1}^{-1} w_1 s_i^{-1} w_2 s_i s_{i+1}$ (the w_is have been relabeled.) and

$$\begin{aligned}
\rho(v) &= C(s_{i+1}^{-1} w_1 s_i^{-1} w_2 s_i s_{i+1}) \\
&= \rho(w_1 s_i^{-1} w_2 s_i) \\
&= \rho(w_1) + C(s_i^{-1} w_2 s_i) \\
&= \rho(w_1) + \rho(w_2).
\end{aligned}$$

After reduction:

$$\begin{aligned}\rho(v') &= \rho(s_{i+1}^{-1} w_1 s_i^{-1} w_2) \\ &\leq \rho(s_{i+1}^{-1} w_1 s_i^{-1}) + \rho(w_2) \\ &\leq C(s_{i+1}^{-1} w_1 s_i^{-1}) + \rho(w_2) \\ &= \rho(w_1) + \rho(w_2).\end{aligned}$$

Case 7: s_i is removed by a conjugate node and s_{i+1} is its descendant. Then $v = s_i s_{i+1} w_2 s_i^{-1}$ and the proof is similar to case 6. □

To check that all cases have been considered see Figure 4.3.

s_i	s_{i+1}	Does either descend from the other node?	Case
leaf	leaf	no	3
leaf	conjugate	s_i descends	6a
leaf	conjugate	no	4
conjugate	leaf	s_{i+1} descends	7a
conjugate	leaf	no	5
conjugate	conjugate	in the same node	1
conjugate	conjugate	s_i descends	6b
conjugate	conjugate	s_{i+1} descends	7b
conjugate	conjugate	no	2

Figure 4.3: Enumeration of cases for Theorem 4.14

Lemma 4.15. *Suppose* $w = \prod_{i=1}^{k} w_i s_i w_i^{-1} \in W_n$ *and* $k = rk_F(fr(w))$, *then* $\rho(w) = rk_F(fr(w))$.

Proof. Apply ρ to the product of bands:

$$\begin{aligned}\rho(w) &\leq \rho(w_1 s_1 w_1^{-1}) + \rho(\prod_{i=2}^{k} w_i s_i w_i^{-1}) \\ &\leq \rho(w_1 s_1 w_1^{-1}) + \cdots + \rho(w_k s_k w_k^{-1}) \\ &\leq 1 + 1 + \cdots + 1 = k.\end{aligned}$$

So $\rho(w) \leq rk_F(fr(w))$. By Proposition 4.6 we have equality. □

Corollary 4.16. *For any $w \in F_n$, $\rho(w) = rk_F(w)$.*

Proof. Let $(w_1 s_1 w_1^{-1}, \ldots, w_k s_k w_k^{-1}) \in F_n^k$ be a band presentation for w where $k = rk_F(w)$. By the previous lemma and Theorem 4.14, $\rho(w) \leq \rho(\prod_{i=1}^{k} w_i s_i w_i^{-1}) = rk_F(w)$. By Proposition 4.6 we have equality. □

4.3 Dynamic Programming Algorithm

In [Orevkov, 2004] Orekov describes and an algorithm for detecting positive bands by "plucking" generators out of a free words. This same idea could possibly be adapted to compute rank. But, our framing and the definition of ρ has the important advantage of being structured as a dynamic programming problem. This allows it to be computed very efficiently.

Dynamic programming problems are those with inductive structure as described in the introduction: optimal results of sub-problems can be combined to form an overall optimal result. In addition, the sub-problems overlap, So results of smaller computations can be cached and reused. (See chapter 15 of [Cormen et al., 2009].)

To see that this occurs for ρ consider the computation for the subwords $\rho(a_2 \cdots a_{n-1} a_n)$ and $\rho(a_1 \cdots a_{n-1})$. In their recursive calls, both must compute $\rho(a_2 \cdots a_{n-1})$, so instead of doing the work twice, we can store this result and reuse it.

A matrix is a convenient tool for storing and organizing these intermediate results. We first compute ρ for the subwords of length one (which require no recursion), fill in entries of the matrix corresponding to them. Then fill in the matrix for larger subwords up to the entire word. This is the opposite of how we tend to think of recursive functions. In detail, the process is:

Algorithm 4.17.

Input: A word $w = s_1 \cdots s_k \in F_n$

Output: $rk_F(w)$.

(i) Allocate an upper triangular matrix $\{m_{i,j}\} \in M(k \times k, \mathbb{Z})$. Each entry $m_{i,j}$ will be used to store $\rho(s_i \cdots s_j)$.

(ii) Fill in $m_{i,i} = 1$ for $i \in \{1, \ldots, k\}$ to store ρ for subwords of length 1 (base case).

(iii) For each $l \in \{2, \ldots k\}$ and $i \in \{1, \ldots, k-(l-1)\}$ fill in $m_{i,i+(l-1)}$ by computing ρ but referencing the entries of matrix M in place of recursive calls: $\rho(s_i \cdots s_j) = m_{i,j}$.

(iv) The algorithm terminates when $l > k$. The top right entry $m_{1,k}$ will contain the rank.

An implementation is provided in Appendix A.1.

Example 4.18. Following this algorithm for the word $bcab^{-1}aba^{-1}c^{-1} \in F_3$ gives the following matrix:

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 3 & 4 & 3 & \mathbf{2} \\ 0 & 1 & 2 & 3 & 4 & 3 & 2 & 1 \\ 0 & 0 & 1 & 2 & 3 & 2 & 1 & 2 \\ 0 & 0 & 0 & 1 & 2 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The rank 2 can be read off from the top right entry.

Chapter 5. Computing Rank Upper Bounds in Braid Group

In this chapter we adapt the rank algorithm for computing rank in free groups to be used for braid groups. This modified algorithm does not solve the rank problem for all braids, but computes close upper bounds for many. It's not uncommon that it computes the rank

exactly, but it is hard to verify when it does. The algorithm is very flexible and encourages future improvements.

To demonstrate it's effectiveness we use it in an empirical test of recognizing quasipositive braids and knots for which their ribbon genus and slice genus coincide

5.1 Identifying Solvable Subwords

The basic relationships used to construct the algorithm for chapter 4 hold for braid groups as well, where their proofs are the same.

Proposition 5.1. *For* $x, \alpha \in B_n$ $rk(\alpha x \alpha^{-1}) = rk(x)$

Proof. Same as Proposition 4.3. □

Proposition 5.2. *For* $x, y \in B_n$, $rk(xy) \leq rk(x) + rk(y)$

Proof. Same as Corollary 4.2, because band presentations for x and y can be concatenated to form a presentation for xy. □

However, braids are much more complex and their intrinsic structure is less tightly coupled with their word representations.

Recall the main idea which led to the algorithm for free groups. Whenever a word contains a subword that we can compute the rank of, this computation can be combined with the splitting rule to improve our rank upper bound. It's not even necessary for the rank to be computed exactly. Any improvement in upper bound for the rank of a subword will lead to an overall improvement.

Choosing which methods and splits to use can be done in a similar way as was done with ρ. So we can specifically focus on solving special cases, and allow the dynamic programming approach to pick the optimal way to use them.

Specifically we will define a modified $\rho' \colon F_{n-1} \to \mathbb{Z}_{\geq 0}$ which operates on braid words. Let $w = s_1 \dots s_m$. Then define

$$\rho'(w) = \begin{cases} m & m \leq 1 \\ \min(\min_{1<i<m}(\rho'(s_1 \cdots s_{i-1}) + \rho'(s_i \cdots s_m)), M_1(w), \dots M_m(w)) & \text{otherwise} \end{cases} \tag{5.1}$$

where each M_i is some special case we can improve upon, just as the conjugacy rule C was applied in the free group.

5.2 Identifying Conjugate Words

Our first heuristic for improvement is to focus less on individual generators in the representation and more on the structure of the braids. In the free group we identify conjugates by comparing the first and last generators, and conjugation by longer words is just a repetition of this rules. In the braid group most conjugates are not identifiable from looking at a single letter. For example $(121)1(\bar{2}\bar{1}\bar{2})$ is clearly a band, but the first letter is not the inverse of the last.

Given a word we want to find a head and tail that are inverses of each other under braid group equivalence. We will define a function $\Omega \colon F_{n-1} \to \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$. Let $w = s_1 s_2 \cdots s_k \in F_{n-1}$ be a braid word. Define $R = \{(i,j) \colon 0 \leq i < j \leq k+1\}$ and define $R' = \{(i,j) \in R \colon [s_1 \cdots s_i] = [s_j \cdots s_k]\}$. Let $m = \min_{(i,j)\in R'}(j-i)$. Define $\Omega(w)$ to be the element $(i,j) \in R'$ where $j - i = m$ having the smallest i value. (When $i = 0$ the left side is the empty word, and similarly for the right when $j > n$).

Now we can define a new conjugation rule:

$$C'(w) = \begin{cases} \rho'(s_{i+1} \cdots s_{j-1}) & j - i - 1 < k, \text{ where } (i,j) = \Omega(w) \\ \infty & \text{otherwise.} \end{cases} \tag{5.2}$$

If w happens to represent the identity braid, $C'(w)$ will also detect this.

The function Ω is defined to give the indices of the longest conjugating word. Choosing the longest is somewhat of a heuristic. There could possibly be a better choice which would lead to an overall smaller result for ρ', but theoretically it's supported by the fact that rank is a conjugacy invariant,

5.3 Commuting Words

The free group does not have any commuting relations, but the braid group does. Specifically when generators are index 2 or more apart they commute.

Proposition 5.3. *If $w = \sigma_{i_1}^{\epsilon_1} \cdots \sigma_{i_m}^{\epsilon_m} \in F_{n-1}$ where each σ_{i_k} is a standard generator, and $|i_l - i_k| \in \{0, 2\}$ for $l, k \in \{1, \ldots, m\}$. Then $rk([w]) \leq \sum_{j=1}^{n-1} |\sum_{\sigma_j = \sigma_{i_k}} \epsilon_k|$ (the total magnitude of the sum of exponents for each generator).*

Proof. Because each generator is at least index 2 apart, the commuting relations can be applied to obtain a word of the form: $[w] = [\sigma_1^{c_1} \cdots \sigma_{n-1}^{c_{n-1}}]$ where each $c_j \in \mathbb{Z}$ is the sum of exponents for the jth generator: $c_j = \sum_{\sigma_j = \sigma_{i_k}} \epsilon_k$. (Note that $c_i = 0$ for every even i, or for every odd i.) This word gives a band presentation considering every generator as a trivial band. □

Note that this is bound does not follow from previous techniques.

Example 5.4. Consider the word $46\bar{2}6\bar{4}62$. It's sum of distinct exponents is 3. But, $\rho'(46\bar{2}6\bar{4}62) = \rho'(46) + \rho'(\bar{2}6\bar{4}62) = 2 + \rho'(6\bar{4}6) = 5$

A possible algorithm for detecting this is to sort and compare the indices of adjacent neighbors, or to count generators in a table and compare adjacent table indices.

5.3.1 Precomputing. One can always precompute ranks for a set of words and utilize those in computing ranks of other words. Whenever a subword is equal to a precomputed word, we can consider it's rank as an additional rule M_i in ρ'.

A bank of precomputed braids could be stored in a sorted table using Dehornoy ordering, Binary search can be used to find braids that match a subword, which is $O(log(n))$ in the number of precomputed words. Precomputing can only improve the algorithm as ρ will not choose to use it if it does not lead to an optimal overall rank estimate.

We think this approach could be particularly effective for targeting the algorithm to a particularly challenging braid. Subwords of interest can be analyzed on paper, or using our later search algorithms. Improvements in those parts would in turn improve the rank estimate.

5.4 Example of a Word it Does Not Solve

The special cases we have reviewed for ρ' greatly improve the rank upper bounds for braids in comparison with ρ for the free group. But ρ' doesn't perfectly compute rank.

Example 5.5. Consider the braid word $w = \bar{2}\bar{2}1\bar{2}1$[1]. w does not contain any commuting subwords (other than commuting $\bar{2}$ with itself). It also contains no subwords with are inverses of each other. So conjugacy and commuting cannot be applied. Therefore $\rho'(w) = 5$.

However:

$$\begin{aligned}
&\bar{2}\bar{2}1\bar{2}1 \\
&= \bar{2}\bar{2}1(121\bar{1}\bar{2}\bar{1})\bar{2}1 \\
&= \bar{2}\bar{2}1(212\bar{1}\bar{2}\bar{1})\bar{2}1 \\
&= \bar{2}\bar{2}(212)(2\bar{1}\bar{2})(\bar{1}\bar{2}1) \\
&= (\bar{2}12)(2\bar{1}\bar{2})(\bar{1}\bar{2}1).
\end{aligned}$$

Therefore $rk(w) \leq 3$.

This example is short but striking because there is no obvious pattern, like in the free group. Inserting a seemingly unrelated identity word and applying some relations gave a

[1]This braid is a subword of the irreducible braid in [Morton, 1983].

better rank estimate.

5.5 Testing Against KnotInfo Database

To test the effectiveness of our bounding methods, we applied them to the KnotInfo database [Livingston and Moore, 2021] which contains invariants and properties for 2977 knots.

One caveat to these tests is that a given knot can be represented by many distinct braids, due to the well-known theorem of Markov. We simply used the braids available in our database. Thus our rank upper bounds may be worse or better, depending on the braid that was chosen. The braids chosen in the KnotInfo database may also be biased in certain ways.

This database does not include rank, so there is no easy way to know exactly how accurate the values are, except in cases where it can be inferred from other invariants. We will describe two such cases.

5.5.1 Quasipositivity. Whenever a braid's exponential sum (lower bound) is equal to it's rank upper bound we can conclude that it is quasipositive (Proposition 2.19). Similarly, we can conclude it is quasinegative if the exponential sum is equal to the negation of the rank upper bound. KnotInfo indicates which knots are quasipositive and quasinegative[2], so we can compare our findings with theirs. In our test **75%** of knots indicated to be quasipositive or quasinegative were found with our method. This includes several knots for which this is known to be difficult. (See Appendix B for details.)

5.5.2 Knots with equal ribbon genus and slice genus. Recall our brief overview of the relationship between the slice genus and rank found in Section 1.2. From the slice genus and (1.2) we can obtain a lower bound on the rank. KnotInfo provides the slice genus for all but 21 of the 2977 knots. So when our computed rank upper bound for a braid is equal to the lower bound from combining (1.1) and (1.2), we can conclude:

[2]No distinction is made between quasipositive and quasinegative in their database.

- The rank upper bound is actually equal to the rank.
- The knot has equal slice genus and ribbon genus.

Our results demonstrate that 498 of the 2956 knots, with slice genus provided, had equal slice genus and ribbon genus. Ribbon genus is not recorded in the KnotInfo database so our findings contribute new information. Appendix B contains a full table of results and additional details about these findings.

Chapter 6. Brute Force Search Theory

A possible critique of the upper bound methods is that they rely too much on the specific word representation. They examine the word and try to "see" bands within it. "Seeing" works in the free group, but the complex interaction of the braid relations make the bands difficult to find. (See Example 5.5.) Imagine a word $w \in F_{n-1}$ having a minimal band presentation which is very dissimilar, having almost none of the same generators in corresponding locations (it's worth thinking about a Cayley graph for this situation as well). It might be difficult to construct the band presentation from subwords, or by applying a sequence of minor transformations to the word. Given these observations and the fact that representation can vary greatly, we might be concerned that this approach is limited.

An alternative approach to the problem is to ignore facts about the specific representation of the braid word, and instead try to generate a minimal band presentation for it. In this chapter we examine this approach from a complexity theory perspective.

6.1 Decision Problems

Definition 6.1. A function f is **computable**, if there exists an algorithm which computes its output for any given input in its domain.

Definition 6.2. A **decision problem** is a subset A of a countable set D (D is referred

to as the **domain**). Every decision problem corresponds with a **membership predicate** $Q\colon D \to \{0,1\}$ where $Q(x) = 1$ if $x \in A$ and $Q(x) = 0$ otherwise.

Decision problems represent mathematical questions which can be solved by answering a predicate. As a set, A is the set of things for which the predicate is true. The domain is countable so the inputs can be encoded as natural numbers (or equivalently binary strings).

Definition 6.3. A decision problem is called **decidable** if it's membership predicate is computable. Decidable problems are also called **recursive sets**.

Definition 6.4. A decision problem $A \subseteq D$ is **semi-decidable** if there exists an algorithm which returns "true", for any input $x \in A$, but for any input $x \in D \setminus A$ it does not terminate. Semi-decidable problems are also called **recursively enumerable sets**. (See Chapter 4.3 of [Pudlak, 2013].)

Because complexity and computability results focus on decision problems, we would like to characterize the rank problem in this form. We construct a family of sets by defining a function $BA\colon \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathcal{P}(B_n)$ so that

$$BA(n,k) = \{x \in B_n \colon x \text{ has a band presentation of } k \text{ bands }\}. \tag{6.1}$$

For a fixed n and k, $BA(n,k)$ is a decision problem. It's corresponding predicate is denoted by $BP\colon B_n \times \mathbb{Z}_{\geq 0} \to \{0,1\}$.

Although computing rank appears to require more information, (existence and minimality), computationally they are the same difficulty.

Proposition 6.5. *Assuming either one is computable, rk and BP can be computed in linear time of each other.*

Proof. Assume we can compute $BP(x,k)$. Find the first non-negative integer k for which $BP(x,k) = 1$. We can conclude $rk(x) = k$. This search is bounded above by the length of any braid word for x, so is linear in the input length.

If we know the rank $rk(x) = k$ then $BP(x, j) = 0$ for all $j < k$ (there are no presentations of lower rank). Furthermore $BP(x, k) = 1$ for all $k + 2m$ where $m \in \mathbb{Z}_{\geq 0}$ because we can always insert any even number of canceling pairs of bands into a minimal band presentation. We also know $BP(x, k + 2m + 1) = 0$ for $m \in \mathbb{Z}_{\geq 0}$ because the parity of the length of words representing x must match the parity of the rank (Corollary 2.24). □

With this fact in mind, we will focus our complexity results on the band presentation existence problem, as rank follows. We naturally want to know how difficult it is and whether it's computable at all.

6.2 The Naive Search

Given a solution to the word problem, such as those from Chapter 3, there is a straightforward semi-algorithm for deciding $BP(x, k)$, and it's runtime is improved by using efficient lower and upper bounds, which we have good estimates for from Chapters 2 and 5.

Algorithm 6.6.

Input: A braid word $w \in F_{n-1}$. The number k of bands in a desired presentation k.

Output: Affirmative, if $[w] \in BA(n, k)$.

(i) Initialize $l = l_0$. For now we will let $l_0 = 1$.

(ii) Generate all band words having conjugate length l or less, store those in a set B_l.

(iii) Generate all possible products of k bands from B_l. These are potential band presentations for $[w]$.

(iv) Use a word problem algorithm to check whether each potential presentation is equal to $[w]$. If one of them is, then terminate. The result is affirmative.

(v) Otherwise, increment l and go back to (ii).

This algorithm is incredibly inefficient and duplicates work each time l increases. The number of bands to check is arbitrarily large, and grows exponentially in the conjugate length l of the band.

Despite impractical performance, the algorithm is guaranteed to terminate whenever $w \in BA(n,k)$. If there is no band presentation with k bands then the algorithm will run forever. The existence of a such an algorithm proves:

Proposition 6.7. *For a given braid group order n and number of bands k, $BA(n,k)$ is semi-decidable.*

6.3 Comparison with Halting Problem

Being semi-decidable tells us some useful information, but we might hope for a better classification. For one, it's not useful for proving a band presentation is actually minimal, so we can't verify rank. If the band presentation search terminates with a successful result, there might still be a band presentation having fewer bands.

Furthermore, the set of Turing machines which terminate on a given input is also semi-decidable. The halting problem is the problem of determining whether a given Turing machine is in this set, and it is a well known undecidable problem. (See Chapter 8 of [Minsky, 1967].)

There are some similarities between our algorithm and the halting problem. In the halting problem one is given a Turing machine and input, but cannot predict whether it will terminate. Perhaps it will halt, but only after a significant amount of time, or perhaps it is stuck in a loop and will run forever. In general, there is no a way to know.

Similarly in the band presentation search algorithm, if it terminates, we know there is a band presentation with the desired number of bands. However, if it runs for a long time, one can never be sure if it just hasn't run for a sufficient amount of time to find the band presentation, or if none exist.

An apparent difference in the halting problem is that there really are Turing machines that run forever, whereas each braid has a finite band presentation. But as we shall see, this doesn't appear to make a difference.

What would it take for the problem to be decidable? Recall the parameter l which is the maximum conjugate length of a band. This parameter was introduced to organize the enumeration of braids in a reasonable order, but if one could predict a bound for l, then the search space would be finite and the band presentation search would terminate on false results.

We expect that l depends in some way on the length of the word to start with. Braids which require longer words to write probably require longer bands to describe as well.

Define $L\colon B_n \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}$ by the following: To determine $L(x,k)$, let S_k denote the set of band presentations for x consisting of k bands. This set may be empty. If it is empty assign $L(x,k) = -1$. Otherwise assign:

$$L(x,k) = \inf_{\prod_i^k (\alpha_i \delta_i \alpha_i^{-1}) \in S_k} \sup |\alpha_i| \tag{6.2}$$

where $|\alpha_i|$ is the length of the conjugating word α_i.

In other words, if we consider any band presentation of k bands for x, its largest conjugate must at least as big as $L(x,k)$. Furthermore, there is at least one whose largest conjugate is $L(x,k)$ (if band presentations of such a length exist).

Theorem 6.8. *If L is bounded by a computable function f then $BA(n,k)$ is decidable for every $n, k \in \mathbb{Z}_{\geq 0}$.*

Proof. We can construct a modified version of Algorithm 6.6 by assigning $l_0 = f(x,k)$. Instead of increasing l after a failed search, it terminates with a negative result. This algorithm decides $BA(n,k)$ for all braids. □

6.3.1 Why f may not computable. Consider the following bound on $L(x,k)$. Given a braid word $w \in F_{n-1}$ the set:

$$S = \{y \in B_n \colon \text{ there exists a word for } y \text{ with length } |w|\}$$

is finite and nonempty. Therefore we can define $f(w) = \max_{y \in S} L(y,k)$.

It might appear Algorithm 6.6 could initialize l_0 to the value of f based on the input length. However, since we have no idea how large those values are, we can't claim there is a variation of our algorithm that actually does that. To construct such an algorithm would require f to be *computable* otherwise we would be implicitly relying on a countably infinite table of input/output pairs, which do not constitute an algorithm.

In fact, an analog of everything we have said in this section is also true of the halting problem. In chapter 8 of [Minsky, 1967], Minsky defines a finite function f similar to L but which bounds the number of steps a Turing machine must run before we can conclude it will never halt. The function f depends on the the number of states Q, the number of non-blank input tape entries N , and the number of symbols S: $f(Q,N,S)$. There are finitely many Turing machines for given values of Q, N, S. For the subset that halt, we can take the upper bound of their runtimes and define f to be this number. We could imagine an algorithm which runs a Turing machine for f steps, and then terminates. However, we know the halting problem is not solvable. The conclusion is that f must grow too quickly to be computable.

6.3.2 Complexity based on word length. The function L might not be bounded by a computable function and it's possible that the halting problem can be reduced to the rank problem. However we suspect there is some relationship between word length and band presentation length with reasonable growth. The following partial results would result in a better classification, if some computable bounds could be constructed.

Let $\Lambda \colon B_n \to \mathbb{Z}_{\geq 0}$ be defined to be the minimum length of a representation for a braid

x. So

$$\Lambda(x) = \min_{w \in F_{n-1} \text{ and } [w]=x} |w|. \tag{6.3}$$

Corollary 6.9. *Suppose there exists a computable function $f\colon \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ so that $L(x,k) \leq f(\Lambda(x),k)$, then BA is decidable.*

Proof. This follows directly from Theorem 6.8 and the fact that $f(W(x),k)$ is a function of x and k. □

If the bounds f from this corollary turn out to reasonably optimistic we can classify it further.

Theorem 6.10. *Fix $k \in \mathbb{Z}_{\geq 0}$. Suppose there exists a polynomial p such that $L(x,k) \leq p(\Lambda(x))$ for all braids $x \in B_n$. Then the set $BA(n,k)$ is in **NP** (the class of decision problems that can be verified in polynomial time).*

Proof. Let $R\colon W_{n-1} \times W_{n-1} \to \{0,1\}$ be defined so $R(w,v) = 1$ iff $[fr(w)] = [fr(v)]$ and v is a concatenation of k bands. The equality can be decided in polynomial time of the lengths of w and v since free redution is linear and braid equality test is polynomial due to Lawrence-Krammer (Proposition 3.5). Bands in v can be recognized with a greedy scanning algorithm, but we could also require v to have a separator symbol between bands for computation purposes.

Let $w \in F_{n-1}$ where $[w] = x$. Let q be a monotonic polynomial where $q(m) \geq 2kp(m)+k$. We will show:

$$[w] \in BA(n,k) \Leftrightarrow \text{ exists a string } v \in W_n \text{ where } R(w,v) = 1 \text{ and } |v| \leq q(|w|).$$

Assume $R(w,v) = 1$ for some string v and $|v| \leq q(\Lambda(w))$. Then $|v| \leq q(|w|)$ because $\Lambda(w) \leq |w|$. Then v gives a band presentation of length k for w so $w \in BA(n,k)$.

Assume $[w] \in BA(n,k)$. Then there exists a band presentation for $[w]$ say $(\alpha_1\delta_1\alpha_1^{-1}, \ldots, \alpha_k\delta_k\alpha_k^{-1})$ where $|\alpha_i| \leq L(x,k)$ of k bands. Let v be its concatenation so $v = \prod_{i=1}^{k} \alpha_i\delta_i\alpha_i^{-1}$. Then $R(w,v) = 1$ and

$$|v| \leq \sum_{i=1}^{k}(2L(x,k)+1) \leq 2kp(\Lambda(x)) + k \leq q(\Lambda(x)). \qquad \square$$

To complete the classification one might find a bound on the conjugate length or demonstrate a sequence of braids whose L values grow faster than any polynomial.

Chapter 7. Practical Search with Templates

In this chapter we use the algebraic structure of braids to develop a band presentation search which is much more efficient than the naive search. This method has been practically implemented and solves the band presentation existence problem for braids with small rank and with band presentations having short conjugate lengths. However, the algorithm is overall the same structure and has the same complexity theory properties. In particular, it only terminates on affirmative results and runs forever otherwise.

7.1 Analysis of Naive Search

In the naive search we generate every band word and try every different way to put them together. This is incredibly inefficient.

Proposition 7.1. *In the group of braid words F_{n-1} there are*

$$2(n-1)(2n-1)^L \tag{7.1}$$

non-trivial bands with maximum conjugate length L.

Proof. First we pick the central generator. There are $n-1$ positive generators and $n-1$ negative generators, so $2n-2$ choices. Next we construct a conjugating word of length L, which can additionally include the identity element (empty word). So there are $(2(n-1)+1)^L = (2n-1)^L$ conjugating words. $\square$

Corollary 7.2. *In the group of braid words F_{n-1} there are*

$$2^k(n-1)^k(2n-1)^{kL} \tag{7.2}$$

band presentations of k bands where each band has maximum conjugate length L.

Fundamentally the growth is too large to be practical. Considering modest inputs such as, $n = 3$, $k = 3$ and $L = 4$ this number is already well beyond the range of a 32 bit integer. Optimizing the word problem solution and using early obstructions can make a significant difference in runtime due to the number of iterations. However, straightforward optimization cannot overcome this issue alone. Rather, we need a way to reduce the set of possible solutions in the first place.

7.2 Enumerating Braids

Clearly there are far more distinct words than distinct braids, due to the braid relations. Just consider bands with canceling pairs or words such as:

Example 7.3. The three bands $13\bar{3}\bar{1} = 1\bar{1} = 1$ are all equal and have maximum conjugate length 2.

It follows that our algorithm generates many more braid words than we need to consider. In practice, there is enormous duplication. For B_3 and length $L = 4$ there are 14406 band words and only 604 distinct band braids.

The ideal way to to avoid duplicating work would be to *enumerate braids*, rather than simply words. But, we don't know of a way to do this. For the purposes of our method however, we can settle for simply generating all band words and then removing duplicates, as the main growth is actually constructing band presentations (raising to the kth power), not generating bands.

The straightforward approach to removing duplicates is to test equality of every band with every other band. This algorithm is $O(n^2)$ in the number of comparisons. The better

approach is to sort and then test equality of adjacent elements. This algorithm is $O(n \log(n))$ in the worst case. In order to sort, we need a total ordering on braids that can be computed efficiently. We have successfully used the Dehornoy ordering, which was sufficiently fast for this purpose.

7.3 Templated Search

Even though we don't know how to enumerate braids, we can do better at *enumerating band presentations*, to reduce the number of ways to assemble those bands.

In the braid group, it is often not clear how concatenating a band will affect it. For a given braid, it is difficult to predict which bands could be use to form a band presentation for it, and thus we blindly try all of them. However, in quotients of the braid group, the effect of adding a band is often very predictable. Furthermore, quotients many only contain a handful of bands, and even fewer ways to arrange them so that the result is equal to the image of the original braid. This idea is made precise by the following definition.

Definition 7.4. Let $\phi\colon B_n \to H$ be a homomorphism and define $\beta_\phi = \{\phi(\alpha\delta\alpha^{-1})\colon \alpha \in B_n, \delta \in \{\sigma_1^{\pm} \dots \sigma_{n-1}^{\pm}\}\}$. A k-**template** for x is a tuple $(b_1, b_2, \dots b_k) \in \beta_\phi^k$. Such that $b_1 b_2 \cdots b_k = \phi(x)$.

Thus if ϕ is surjective a k-template is the same as band presentation for $\phi(x)$ in the quotient group H having a fixed length k (see definition 2.3). Clearly the image of any minimal band presentation is a k-template.

When $|\beta_\phi|$ is finite, we can construct all k-templates and partition the set of bands based on their images, as in the following algorithm:

Algorithm 7.5.

Input: A braid x. The number of bands in a potential presentation k. A maximum conjugate length l_0. A homomorphism ϕ such that $|\beta_\phi| < \infty$.

Output: A set of all band presentations for x within the conjugate length limit.

(i) Initialize $l = l_0$

(ii) Generate all band words with maximum conjugate length l. Place each band word β in a set based on it's image: $C_{\phi([\beta])}$. These sets are referred to as *band buckets*.

(iii) Remove band words that represent the same braid from each bucket $C_{\phi([\beta])}$. Each word in a bucket now represents a distinct braid.

(iv) Generate all k-templates for x. This is the set $T_k = \{(b_1, \ldots, b_k) \in \beta_\phi \colon b_1 \cdots b_k = \phi(x)\}$.

(v) For each template $(b_1, \ldots, b_k)$ consider every $(w_1, \ldots, w_k) \in C_{b_1} \times \cdots \times C_{b_k}$. If $w_1 \cdots w_k = x$ then terminate. The result is affirmative.

(vi) Increment l and go back to (ii).

The reason why duplicate braids are removed after after placing them into sets, instead of before, is the placement acts like a "radix" sort. Braids with distinct images under ϕ are never equal, so by sorting them into "buckets" first, there are fewer words to compare.

7.3.1 Analysis. This complicates the search quite a bit, so it may not be immediately clear that constructing templates saves any work. First observe that the amount of time to create templates does not scale in the length parameter l, so it isn't the significant time in our algorithm.

There is some additional time to put words into buckets, but we already need to remove duplicates, and evaluating a homomorphism to partition the work is already a smart optimization. So what really matters is whether it reduces the number of band presentations we must enumerate, and it's not difficult for this to be the case.

Proposition 7.6. *Suppose M is an upper bound on the bucket size. So for each band $b \in \beta_\phi$, $|C_b| \leq M$. Then Algorithm 7.5 $|T_k|M^k$ band presentations.*

Proof. The precise number of band presentations considered is:

$$\sum_{(b_1,\ldots,b_k)\in T_k} |C_{b_1}| \cdots |C_{b_k}| \leq \sum_{(b_1,\ldots,b_k)\in T_k} M^k = |T_k|M^k. \qquad \square$$

Corollary 7.7. *Assuming bands are evenly distributed among buckets, Algorithm 7.5 will compare $\frac{|T_k|}{|\beta_\phi|^R}$ as many band presentations as the naive search.*

Proof. If bands are evenly distributed then each bucket should have $M = N/|\beta_\phi|$ items where N is the total number of bands. So by the previous proposition the total number of band presentations considered is $|T_k|N^k/|\beta_\phi|^k$. $\square$

Definition 7.8. The reciprocal

$$\frac{|\beta_\phi|^R}{|T_k|} \tag{7.3}$$

will be referred to as the **speedup factor**.

A speedup factor of 3 would mean the algorithm should be roughly 3x faster because it only needs to check 1/3 as many band presentations. It must be greater than 1 to represent an improvement. Ideally it will increase with the rank and number of strands, to counteract the increase in the number of band presentations.

7.4 Permutation Templates

We will now make this concrete by considering specific quotients. We will start with the symmetric group and the homomorphism $\phi\colon B_n \to S_n$ (from Section 2.3).

In the symmetric group, the bands β_ϕ are precisely the transpositions (Proposition 2.9). Every permutation can be written as a product of at most n transpositions in only finitely many ways. These possible ways tell us which bands to consider and in which positions, relative to one another.

In terms of the concepts just introduced:

Definition 7.9. A k**-template** for $x \in B_n$ in the symmetric group is a tuple of transpositions $((i_1 j_1), \ldots, (i_k, j_k))$ such that $\phi(w) = (i_1 j_1) \cdots (i_k j_k)$.

Although the symmetric group may allow commuting of disjoint transpositions, we must consider them as distinct templates, which is why tuples are used. They represent images of band presentations and the braid group is non-abelian.

Example 7.10. The braid $x = \bar{1}211$ can be trivially decomposed into bands $\bar{1}21$ and 1. Suppose however, we couldn't see them and suspected there was a presentation of 2 bands. We can examine the image of x in the symmetric group: $\phi(\bar{1}211) = (123)$. There are several ways to represent (123) as a product of 2 transpositions, each of which is a 2-template. They are: $((13), (12))$, $((23), (13))$, and $((12), (23))$.

We consider the first template: $((13), (12))$. Any band b where $\phi(b) \notin \{(12), (13)\}$ can be ignored. If $\phi(b) = (13)$ we know to place it on the left, and otherwise on the right. If we apply this search to all such bands having maximum conjugate length 1, we would find our original band presentation: $\phi(\bar{1}21)\phi(1) = (13)(12)$.

This is just one template. If a presentation was not found, we could continue the search with the remaining templates.

7.4.1 Generating templates. For a braid $x \in B_n$ we want to generate the set of all templates T_k. In other words we want to find all the ways to write $\phi(x)$ as a product of k transpositions.

Our first inclination might be to use clever facts about the symmetric group. If we write $\phi(x)$ as a product of disjoint cycles, then cycling and commuting these enumerates some of the ways of writing it as a product of transpositions. However, we must also consider inserting canceling pairs. This is complicated enough to prefer the brute force approach:

Algorithm 7.11. (i) Generate every transposition $\beta_\phi = \{(i, j) \colon 1 \leq i < j \leq n\}$.

(ii) Generate the set β_ϕ^k and find the subset equal to $\phi(x)$.

7.4.2 Analysis. Regardless of the algorithm chosen to generate them, what ultimately matters is how many templates are produced.

Lemma 7.12. *Every permutation $\varphi \in S_n$ can be represented by at most $\binom{n}{2}^{k-1}$ words of length k where the letters of the words are the transpositions.*

Proof. Let $A_{\varphi,l}$ be the set of words of length l in the transpositions of S_n equal to φ. Multiplying each element of S_n by a fixed element gives an automorphism of S_n. Observe that for each transposition (a, b) we have $\{(ab)w \colon w \in A_{(ab)^{-1}\varphi,l}\} \subseteq A_{\varphi,l+1}$ and whenever $\varphi' \neq (ab)^{-1}\varphi$ we have $\{(ab)w \colon w \in A_{\varphi',l}\} \cap A_{\varphi,l+1} = \emptyset$. Therefore

$$\begin{aligned} |A_{\varphi,l+1}| &= \sum_{(cd) \text{ is a transposition}} |A_{(cd)^{-1}\varphi,l}| \\ &\leq \binom{n}{2} \max_{\psi \in S_n} |A_{\psi,l}|. \end{aligned}$$

Now $|A_{\varphi,1}|$ is at most 1 if φ is odd, and $|A_{\varphi,1}| = 0$ if φ is even.

Assume that $|A_{\varphi,k}| \leq \binom{n}{2}^{k-1}$ if k and φ have the same parity, and it's 0 otherwise. Then for φ with the same parity as k: $|A_{\varphi,k+1}| \leq \binom{n}{2}|A_{\varphi,k}| = \binom{n}{2}^{k}$ and for φ with differing parity as k, $|A_{\varphi,k+1}| \leq 0$. The result follows by induction. □

Theorem 7.13. *The speedup factor is at least $\binom{n}{2}$.*

Proof. There are $\binom{n}{2}$ transpositions in S_n and this is the number of band buckets. So $|\beta_\phi|^k = \binom{n}{2}^k$. The number of templates is precisely the number of words of length k equal to $\phi(x)$ which we have a bound on: Therefore:

$$\frac{|\beta_\phi|^k}{|T_k|} \geq \frac{\binom{n}{2}^k}{\binom{n}{2}^{k-1}} = \binom{n}{2}$$ □

This is a good speedup factor[1], But the assumption that braid permutations are evenly distributed, doesn't actually hold. Empirically, they appear to distribute very close to evenly

[1]This is a conservative estimate because counting these words is hard. The exact number would be described by a sequence such as OEIS A029698[oei, 2021].

though.

7.5 Exponential Sum Templates

The next obvious quotient to consider is the exponential sum. The image of a band under γ is just 1 or -1 depending on whether it's a positive band or negative band.

Definition 7.14. A $k-$**template** for $x \in B_n$ in $\mathbb{Z}$, is a tuple $(s_1, \dots s_k) \in \{1, -1\}^k$, such that $\sum_{i=1}^{k} s_i = \gamma(x)$. This can alternatively be represented by a string of signs: $\{+, -\}^k$.

Proposition 7.15. *The set of $k-$templates for $x \in B_n$ in $\mathbb{Z}$. is the set of tuples of length k having $p = \frac{k+\gamma(x)}{2}$ entries of 1, and $n = \frac{k-\gamma(x)}{2}$ entries of -1.*

Proof. This follows from Proposition 2.17 □

Example 7.16. Recall the word $w = \bar{2}\bar{2}1\bar{2}1$ from the end of Chapter 5. This has rank 3, although it is difficult to see. We can compute $\gamma(w) = -1$. So there must be $(-1+3)/2 = 1$ positive band, and 2 negative bands in a presentation.

The possible 3-templates for w are: $+--$, $-+-$, $--+$. The band presentation we found $(\bar{2}12)(2\bar{1}\bar{2})(\bar{1}\bar{2}1)$ satisfies the template $+--$.

7.5.1 Analysis.

Proposition 7.17. *For a braid x, rank k, and positive bands $p = \frac{k+\gamma(x)}{2}$ there are $\binom{k}{p}$ $k-$templates in T_k.*

Proof. Counting the number of ways to arrange p positive signs and $k-p$ negative signs is the same as counting permutations of k elements, but with duplicates. Ignoring duplicate letters there are $k!$ permutations. The negative sign duplicates produce a factor of $(k-p)!$ equivalent rearrangements and similarly the positive signs produce a factor of $p!$ equivalent rearrangements. So the total is:

$$|T_k| = \frac{k!}{p!(k-p)!} = \binom{k}{p}.$$ □

.

To generate them, we just modify the relationship $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ to be an inductive construction of sets instead of counting them.

Theorem 7.18. *The speedup factor is at least*

$$\frac{2^k}{\binom{k}{\lfloor \frac{k+1}{2} \rfloor}}.$$

Proof. Because there are two buckets, $|\beta_\phi|^k = 2^k$ and the largest $|T_k|$ can can get is when $p = \lfloor \frac{k+1}{2} \rfloor$. □

To see this this speedup factor is actually an improvement apply the binomial theorem: $2^k = (1+1)^k = \sum_{i=1}^{k} \binom{k}{i}$. Therefore:

$$\binom{k}{p} = 2^k - \sum_{i \in \{1,\ldots,k\} \setminus \{p\}} \binom{k}{i} < 2^k.$$

Figure 7.1 is graph comparing their growths. Unlike permutations, the signs of braids are evenly distributed.

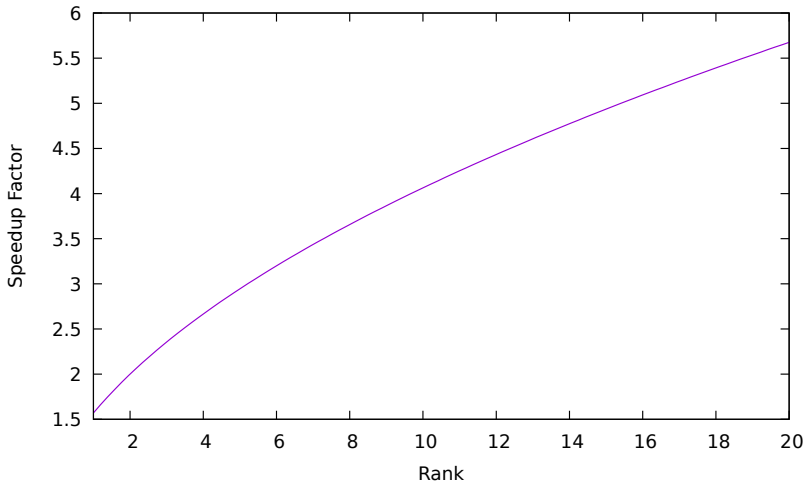

Figure 7.1: Graph of exponential sum speedup factor.
(Plotted continuously with the Γ function.)

7.6 Signed Permutation Templates

The exponential sum seems less useful as it only partitions the bands into 2 buckets. But both methods can be combined with improved results. Define a homomorphism $\psi\colon B_n \to S_n \times \mathbb{Z}$ by $\psi(x) = (\phi(x), \gamma(x))$. The k-templates for ψ are tuples of signed transpositions. This set T_k is the Cartesian product of the permutation templates and the exponential sum templates.

Proposition 7.19. *The speedup factor for using signed permutation templates is just the product of each separate method's speedup factor.*

$$\frac{2^k \binom{n}{2}}{\binom{k}{\lfloor \frac{k+1}{2} \rfloor}}$$

Trying out our original test values $n = 3$ and $k = 3$, and assuming p is the worst case $\lfloor \frac{k+1}{2} \rfloor$, we improve the runtime from naive search by a factor of 8.

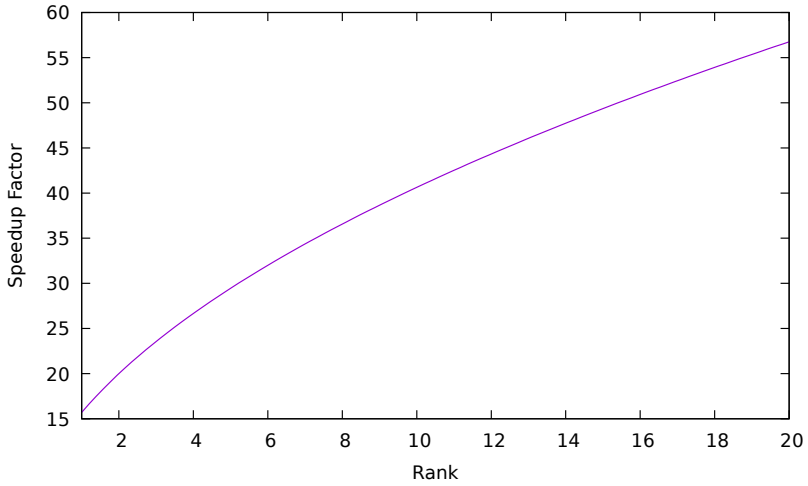

Figure 7.2: Graph of signed permutation speedup factor for $n = 5$ strands. (Plotted continuously with the Γ function.)

7.7 Implementation Notes

Several steps of Algorithm 7.5 describe how to generate sets. Sometimes these sets are very large and care should be taken to enumerate them one at a time, rather then generating

them all at once. In our implementation we chose to generate all the templates and bands beforehand, and then to only enumerate the various band presentations.

7.7.1 Parallelization/Threading. Sorting and merging to remove duplicates from each band bucket can be done in parallel.

Each band presentation comparison with the braid word can be considered independently. However, iterating them independently is probably not convenient. A more natural way to divide the work is to process each template in parallel.

7.7.2 GPU computing. We also considered whether computations using a GPU would make sense, as the band presentation search resembles hash/nonce searches used to crack cryptography. We wrote a word problem implementation ideal for a GPU. However, after further analysis realized the templated search wouldn't do well. The issue is sharing the buckets of bands with the various compute units. Each GPU compute unit has a tiny memory cache. The words from the buckets must be pulled into the cache in order to do their word problem check. Thus, a lot of the time would be spent waiting for data to arrive, instead of performing computations

7.8 Example of Computation

Recall the braid which our upper bound algorithm could not solve: $x = \bar{2}\bar{2}1\bar{2}1$ (see example 5.5). Using our bounding methods we conclude that $1 \leq rk(x) \leq 5$. We suspect there is a band presentation of 3 bands having maximum conjugate length $l \leq 3$. There are 1029 distinct bands in the 3 strand braid group with this maximum conjugate length. Therefore the naive search (after removing duplicates) woud compare $1029^3 = 1,089,547,389$ band presentations.

The signed permutation buckets each have about 10 bands after removing duplicates:

- $C_{(12)}$ has 9 bands

- $C_{-(12)}$ has 9
- $C_{(13)}$ has 10
- ...

We enumerate each 3-template for x. There are 27 total:

- $-(13), -(12), (12)$ has 810 presentations to check
- $(12), -(13), -(23)$ has 810
- ...

After checking all of these (fewer than $27,000$ total band presentations), we find 5 band presentations for x:

$$\bar{2}\bar{2}1\bar{2}\bar{1}22, 3\bar{1}2\bar{3}\bar{2}1\bar{3}, \bar{1}\bar{1}\bar{1}2111$$

$$\bar{2}\bar{2}1\bar{2}\bar{1}22, \bar{2}\bar{2}1\bar{2}\bar{1}22, 3123\bar{2}\bar{1}\bar{3}$$

$$3\bar{1}2\bar{3}\bar{2}1\bar{3}, \bar{1}\bar{1}\bar{1}2111, 3\bar{1}2\bar{3}\bar{2}1\bar{3}$$

$$\bar{2}\bar{2}1\bar{2}\bar{1}22, 3123\bar{2}\bar{1}\bar{3}, 3\bar{1}2\bar{3}\bar{2}1\bar{3}$$

$$3123\bar{2}\bar{1}\bar{3}, 3\bar{1}2\bar{3}\bar{2}1\bar{3}, 3\bar{1}2\bar{3}\bar{2}1\bar{3}.$$

Appendix A. Implementations of Selected Algorithms in Common Lisp

The full source code for a program that computes bounds on braid rank, as well as a program for brute force searching with the template method, can be found at the GitHub page: `https://github.com/justinmeiners/braid-rank-thesis`.

This appendix includes a selection of generally useful algorithms that are self contained and concise.

A.1 Rank in the Free Group

```
(defun rank-rule (word start end memo)
  (min
    (if  (= (aref word start)
            (- (aref word (- end 1))))
         (funcall memo (+ start 1) (- end 1))
         (+ 1 (- end start)))

    (loop for k from (+ start 1) below end minimize
          (+ (funcall memo start k)
             (funcall memo k end)))))

(defun rank (free-word)
  "free-word is a sequence of integers. eg: (1 2 -1)"
  (let* ((word (coerce free-word 'vector))
         (N (length word))
         (matrix (make-array (list N N)
                             :element-type 'fixnum
                             :initial-element 0))
         (memo-lookup (lambda (start end)
                        (aref matrix start (- end 1)))))

    (loop for i from 0 below N do
          (setf (aref matrix i i) 1))

    (loop for length from 2 to N do
          (do ((i 0 (+ i 1)))
              ((> (+ i length) N) nil)

              (let ((j (+ i (- length 1))))
                (setf (aref matrix i j)
                      (rank-rule word i (+ j 1) memo-lookup)))))

    (values (aref matrix 0 (- N 1)) matrix)))
```

A.2 Rank Upper Bound in the Braid Group

The **identity-p** function used in this sample is not included. This function is an implementation of one of the word problem solutions discussed in Chapter 3. Several implementations can be found on the GitHub page.

```
(defun commutable-p (braid)
  (let ((min-step 2))
    (reduce
      (lambda (a b)
        (let ((step (abs (- a b))))
          (if (and (not (= step 0)) (< step min-step ))
              (setf min-step step)))
        b)
      (sort (copy-seq braid) #'<))
    (= min-step 2)))

(defun find-conjugate (braid)
  (let* ((N (length braid))
         (best N)
         (start 0)
         (end N))
    (loop for i from 1 below (- N 1) do
          (loop for j from (+ i 1) below N do
                (when (identityp (nconc (subseq braid 0 i)
                                        (subseq braid j)))
                  (when (< (- j i) best)
                    (setf best (- j i))
                    (setf start i)
                    (setf end j)))

                ))
    (values start end)))

(defun upper-bound-part (braid start end memo)
  (let ((part (subseq braid start end)))
    (multiple-value-bind (i j)
      (find-conjugate part)

      (cond ((identityp part) 0)
            ((commutable-p part) (group:total-generators part))
            (t
             (min
               (if (< (- j i) (- end start))
                   (funcall memo (+ start i) (+ start j))
                   (- end start))

               (loop for k from (+ start 1) below end minimize
                   (+ (funcall memo start k)
                      (funcall memo k end))))
             )))))
```

```
(defun rank-upper-bound (braid)
  "rank upper bound: modified version of free group rank."
  (let* ((N (length braid))
         (matrix (make-array (list N N)
                             :element-type 'fixnum
                             :initial-element 0))
         (memo-lookup (lambda (start end)
                        (aref matrix start (- end 1)))))

    (loop for i from 0 below N do
          (setf (aref matrix i i) 1))

    (loop for length from 2 to N do
          (do ((i 0 (+ i 1)))
              ((> (+ i length) N) nil)

              (let ((j (+ i (- length 1))))
                (setf (aref matrix i j)
                      (upper-bound-part braid i (+ j 1) memo-lookup)))))

    (values (aref matrix 0 (- N 1)) matrix)))
```

Appendix B. Table of Computed Knot Results

The following table lists knots from the KnotInfo database [Livingston and Moore, 2021], for which our rank method proved interesting properties. The properties listed are the following:

- P: The knot is quasipositive. This is shown when the exponential sum γ is equal to computed rank rk.
- N: The knot is quasinegative. The exponential sum is equal to the negation of the computed rank.
- R: The knot has equal slice and ribbon genus. The l column is the lower bound on the

rank genus predicted by the slice genus g_s and the number of strands n

$$l = 2g_s - 1 + n \leq rk$$

. When $rk = 2g_s - 1$ then it's know that the slice genus and the ribbon genus of the corresponding knots are equal (see the discussion for (1.2)).

- *: According to KnotInfo, showing this knot is quasipositive or quasinegative requires advanced techniques. Our method may offer a simpler alternative.

Name	Chosen Braid	n	g_s	l	γ	rk	Properties
3 1	111	2	1	3	3	3	R P
4 1	$1\bar{2}1\bar{2}$	3	1	4	0	4	R
5 1	11111	2	2	5	5	5	R P
5 2	$1112\bar{1}2$	3	1	4	4	4	R P
7 1	1111111	2	3	7	7	7	R P
7 2	$1112\bar{1}23\bar{2}3$	4	1	5	5	5	R P
7 3	$11111\bar{2}\bar{1}2$	3	2	6	6	6	R P
7 4	$112\bar{1}22\bar{3}\bar{2}3$	4	1	5	5	5	R P
7 5	$11112\bar{1}22$	3	2	6	6	6	R P
8 1	$112\bar{1}23\bar{2}\bar{4}3\bar{4}$	5	1	6	2	6	R
8 3	$112\bar{1}\bar{3}2\bar{3}\bar{4}3\bar{4}$	5	1	6	0	6	R
8 15	$11\bar{2}132223$	4	2	7	7	7	R P
8 19	11121112	3	3	8	8	8	R P
8 20	$111\bar{2}\bar{1}\bar{1}\bar{1}\bar{2}$	3	0	2	-2	2	R N *
8 21	$1112\bar{1}\bar{1}22$	3	1	4	4	4	R P *
9 1	111111111	2	4	9	9	9	R P
9 2	$1112\bar{1}23\bar{2}34\bar{3}4$	5	1	6	6	6	R P
9 3	$1111111\bar{2}\bar{1}2$	3	3	8	8	8	R P
9 4	$11111\bar{2}1\bar{2}3\bar{2}3$	4	2	7	7	7	R P
9 5	$112\bar{1}223\bar{2}34\bar{3}4$	5	1	6	6	6	R P
9 6	$1111112\bar{1}22$	3	3	8	8	8	R P
9 7	$11112\bar{1}23\bar{2}33$	4	2	7	7	7	R P
9 9	$111112\bar{1}222$	3	3	8	8	8	R P
9 10	$112\bar{1}2222\bar{3}\bar{2}3$	4	2	7	7	7	R P
9 13	$11112\bar{1}22\bar{3}\bar{2}3$	4	2	7	7	7	R P
9 16	$111122\bar{1}222$	3	3	8	8	8	R P
9 18	$1112\bar{1}222\bar{3}\bar{2}3$	4	2	7	7	7	R P
9 23	$1112\bar{1}223\bar{2}33$	4	2	7	7	7	R P
9 35	$112\bar{1}223\bar{2}\bar{2}4\bar{3}243$	5	1	6	6	6	R P
9 42	$\bar{1}\bar{1}\bar{1}211\bar{3}2\bar{3}$	4	1	5	-1	5	R

9 44	$1112\bar{1}\bar{1}\bar{3}2\bar{3}$	4	1	5	1	5	R
9 45	$\bar{1}\bar{1}\bar{2}1\bar{2}\bar{1}\bar{3}2\bar{3}$	4	1	5	-5	5	R N *
9 46	$\bar{1}2\bar{1}2\bar{3}\bar{2}1\bar{2}\bar{3}$	4	0	3	-3	3	R N
10 1	$112\bar{1}23\bar{2}34\bar{3}\bar{5}4\bar{5}$	6	1	7	3	7	R
10 49	$1111\bar{2}132223$	4	3	9	9	9	R P
10 53	$112\bar{1}2\bar{3}243334$	5	2	8	8	8	R P
10 55	$1112\bar{1}3243334$	5	2	8	8	8	R P
10 63	$11\bar{2}132223434$	5	2	8	8	8	R P
10 66	$111\bar{2}1322233$	4	3	9	9	9	R P
10 80	$111\bar{2}1132223$	4	3	9	9	9	R P
10 101	$1112\bar{1}3\bar{2}1322434$	5	2	8	8	8	R P
10 120	$112\bar{1}3\bar{2}14322334$	5	2	8	8	8	R P
10 124	1111121112	3	4	10	10	10	R P
10 125	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}21112$	3	1	4	0	4	R
10 126	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{2}1112\bar{2}$	3	1	4	-4	4	R N *
10 127	$111112\bar{1}\bar{1}22$	3	2	6	6	6	R P *
10 128	$11121122\bar{3}\bar{2}3$	4	3	9	9	9	R P
10 130	$111\bar{2}\bar{1}\bar{1}\bar{2}\bar{2}\bar{3}2\bar{3}$	4	1	5	-3	5	R
10 131	$1112\bar{1}\bar{1}22\bar{3}\bar{2}3$	4	1	5	5	5	R P *
10 132	$111\bar{2}\bar{1}\bar{1}\bar{2}\bar{3}2\bar{3}\bar{3}$	4	1	5	-3	5	R
10 133	$1112\bar{1}\bar{1}23\bar{2}33$	4	1	5	5	5	R P *
10 134	$11121123\bar{2}33$	4	3	9	9	9	R P
10 139	1111211122	3	4	10	10	10	R P
10 140	$\bar{1}\bar{1}\bar{1}21112\bar{3}\bar{2}3$	4	0	3	3	3	R P *
10 141	$\bar{1}\bar{1}\bar{1}\bar{1}211122$	3	1	4	2	4	R
10 142	$11121112\bar{3}\bar{2}3$	4	3	9	9	9	R P
10 143	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{2}11\bar{2}\bar{2}$	3	1	4	-4	4	R N *
10 145	$\bar{1}\bar{1}\bar{2}\bar{1}\bar{2}\bar{1}\bar{3}\bar{2}1\bar{2}\bar{3}$	4	2	7	-7	7	R N *
10 147	$111\bar{2}1\bar{2}\bar{3}\bar{2}\bar{1}2\bar{3}$	4	1	5	1	5	R
10 148	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{2}11\bar{2}1\bar{2}$	3	1	4	-4	4	R N *
10 149	$1111\bar{2}1\bar{2}\bar{1}22$	3	2	6	6	6	R P
10 150	$111\bar{2}113\bar{2}\bar{1}32$	4	2	7	5	7	R
10 152	1112211222	3	4	10	10	10	R P
10 154	$112\bar{1}2132223$	4	3	9	9	9	R P
10 156	$\bar{1}\bar{1}\bar{1}211\bar{3}\bar{2}1\bar{2}\bar{3}$	4	1	5	-3	5	R
10 157	$\bar{1}\bar{1}\bar{1}\bar{2}\bar{2}1\bar{2}1\bar{2}\bar{2}$	3	2	6	-6	6	R N *
10 159	$1112\bar{1}2\bar{1}\bar{1}22$	3	1	4	4	4	R P *
10 161	$1112\bar{1}21122$	3	3	8	8	8	R P
10 164	$11\bar{2}1\bar{2}\bar{2}\bar{3}2\bar{1}2\bar{3}$	4	1	5	-1	5	R
11a 43	$112\bar{1}21\bar{3}2433344$	5	3	10	10	10	R P
11a 94	$111122\bar{1}223\bar{2}33$	4	3	9	9	9	R P
11a 95	$1112\bar{1}223\bar{2}34\bar{3}44$	5	2	8	8	8	R P
11a 186	$1112\bar{1}22223\bar{2}33$	4	3	9	9	9	R P
11a 191	$111112\bar{1}223\bar{2}33$	4	3	9	9	9	R P
11a 192	$1112\bar{1}223\bar{2}334\bar{3}4$	5	2	8	8	8	R P
11a 200	$1112\bar{1}223\bar{2}\bar{2}4\bar{3}2443$	5	2	8	8	8	R P
11a 211	$\bar{1}\bar{1}\bar{2}1\bar{2}\bar{3}2\bar{3}4\bar{3}\bar{5}4\bar{5}$	6	2	9	-5	9	R
11a 234	$11111111\bar{2}\bar{1}22$	3	4	10	10	10	R P

11a 235	$1112\bar{1}22222\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 236	$11112\bar{1}222\bar{3}\bar{2}33$	4	3	9	9	9	R P
11a 237	$1112\bar{1}222\bar{3}\bar{2}\bar{2}4\bar{3}243$	5	2	8	8	8	R P
11a 238	$1112\bar{1}222\bar{3}\bar{2}34\bar{3}4$	5	2	8	8	8	R P
11a 240	$11111122\bar{1}222$	3	4	10	10	10	R P
11a 241	$111122\bar{1}222\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 242	$1111112\bar{1}23\bar{2}33$	4	3	9	9	9	R P
11a 243	$1112\bar{1}23\bar{2}333\bar{4}\bar{3}4$	5	2	8	8	8	R P
11a 245	$11112\bar{1}233\bar{2}333$	4	3	9	9	9	R P
11a 246	$11112\bar{1}23\bar{2}34\bar{3}44$	5	2	8	8	8	R P
11a 247	$1112\bar{1}23\bar{2}34\bar{3}4\bar{5}45$	6	1	7	7	7	R P
11a 263	$111122\bar{1}322233$	4	4	11	11	11	R P
11a 334	$1111112\bar{1}2222$	3	4	10	10	10	R P
11a 335	$111112\bar{1}222\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 336	$1111112\bar{1}22\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 337	$112\bar{1}222\bar{3}\bar{2}33\bar{4}\bar{3}4$	5	2	8	8	8	R P
11a 338	$11111222\bar{1}222$	3	4	10	10	10	R P
11a 339	$111112\bar{1}23\bar{2}333$	4	3	9	9	9	R P
11a 340	$1111222\bar{1}22\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 341	$11112\bar{1}23\bar{2}33\bar{4}\bar{3}4$	5	2	8	8	8	R P
11a 342	$111112\bar{1}23\bar{2}34\bar{3}4$	5	2	8	8	8	R P
11a 343	$112\bar{1}223\bar{2}34\bar{3}4\bar{5}45$	6	1	7	7	7	R P
11a 355	$11111112\bar{1}222$	3	4	10	10	10	R P
11a 356	$11112\bar{1}2222\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 357	$11112\bar{1}223\bar{2}333$	4	3	9	9	9	R P
11a 358	$1111112\bar{1}2\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 359	$112\bar{1}2222\bar{3}\bar{2}34\bar{3}4$	5	2	8	8	8	R P
11a 360	$11112\bar{1}223\bar{2}34\bar{3}4$	5	2	8	8	8	R P
11a 361	$11112\bar{1}223\bar{2}\bar{2}4\bar{3}243$	5	2	8	8	8	R P
11a 362	$112\bar{1}223\bar{2}\bar{2}4\bar{3}245\bar{4}35$	6	1	7	7	7	R P
11a 363	$112\bar{1}23\bar{2}334\bar{3}4\bar{5}45$	6	1	7	7	7	R P
11a 364	$111111111\bar{2}\bar{1}2$	3	4	10	10	10	R P
11a 365	$112\bar{1}222222\bar{3}\bar{2}3$	4	3	9	9	9	R P
11a 366	$112\bar{1}2222\bar{3}\bar{2}\bar{2}4\bar{3}243$	5	2	8	8	8	R P
11a 367	11111111111	2	5	11	11	11	R P
11n 1	$\bar{1}\bar{1}\bar{1}\bar{2}\bar{1}3\bar{2}\bar{3}\bar{2}4\bar{3}4$	5	1	6	-6	6	R N
11n 10	$\bar{1}\bar{1}\bar{1}\bar{2}\bar{1}\bar{2}\bar{2}\bar{1}\bar{3}\bar{2}\bar{3}$	4	2	7	-7	7	R N
11n 12	$11\bar{2}1\bar{2}\bar{2}32\bar{1}23$	4	1	5	3	5	R
11n 14	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{2}1\bar{2}\bar{1}\bar{3}\bar{2}\bar{3}$	4	2	7	-7	7	R N
11n 17	$\bar{1}\bar{1}\bar{2}1\bar{2}\bar{3}\bar{2}\bar{3}\bar{2}4\bar{3}4$	5	1	6	-6	6	R N
11n 18	$112\bar{1}223\bar{2}\bar{2}4\bar{3}4$	5	1	6	2	6	R
11n 19	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}211\bar{3}\bar{2}\bar{3}$	4	2	7	-3	7	R
11n 20	$\bar{1}\bar{1}\bar{2}1\bar{2}\bar{2}32\bar{2}4\bar{3}4$	5	1	6	-2	6	R
11n 28	$1112\bar{1}\bar{1}23\bar{2}\bar{4}3\bar{4}$	5	1	6	2	6	R
11n 35	$11\bar{2}1\bar{2}132233$	4	2	7	7	7	R P
11n 38	$111\bar{2}\bar{1}3\bar{2}\bar{1}2\bar{3}\bar{2}$	4	1	5	-1	5	R
11n 40	$1\bar{2}1\bar{2}\bar{2}132223$	4	1	5	5	5	R P
11n 43	$11\bar{2}1\bar{2}132223$	4	2	7	7	7	R P

11n 48	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{2}111\bar{3}\bar{2}\bar{3}$	4	1	5	-1	5	R
11n 51	$\bar{1}\bar{1}\bar{1}211\bar{3}\bar{2}132$	4	1	5	3	5	R
11n 54	$112\bar{1}\bar{1}213\bar{2}\bar{3}33$	4	1	5	5	5	R P
11n 59	$111\bar{2}1\bar{2}32223$	4	2	7	7	7	R P
11n 62	$1112\bar{1}\bar{1}32\bar{3}\bar{4}3\bar{4}$	5	1	6	0	6	R
11n 63	$112\bar{1}213\bar{2}34\bar{3}4$	5	1	6	6	6	R P
11n 65	$\bar{1}2\bar{1}\bar{2}\bar{2}32\bar{1}22\bar{3}$	4	1	5	-3	5	R
11n 72	$11\bar{2}\bar{2}1322233$	4	2	7	7	7	R P
11n 75	$\bar{1}\bar{1}222\bar{1}3\bar{2}\bar{2}\bar{3}$	4	1	5	-5	5	R N
11n 77	11221322233	4	4	11	11	11	R P
11n 79	$111\bar{2}\bar{1}\bar{1}3\bar{2}34\bar{3}4$	5	1	6	2	6	R
11n 82	$1111\bar{2}\bar{1}\bar{1}3\bar{2}3$	4	1	5	1	5	R
11n 84	$\bar{1}\bar{1}2\bar{1}2\bar{3}\bar{2}1\bar{2}\bar{3}$	4	1	5	-5	5	R N
11n 85	$\bar{1}\bar{1}\bar{1}21112\bar{3}\bar{2}\bar{3}$	4	1	5	1	5	R
11n 87	$\bar{1}\bar{1}\bar{2}1\bar{2}\bar{1}32\bar{1}\bar{3}2$	4	1	5	-5	5	R N
11n 89	$\bar{1}\bar{1}\bar{1}2\bar{1}\bar{3}\bar{2}1\bar{2}\bar{3}$	4	2	7	-7	7	R N
11n 91	$\bar{1}\bar{1}\bar{2}132\bar{4}\bar{3}2\bar{3}\bar{3}\bar{4}$	5	1	6	-6	6	R N
11n 95	$1112\bar{1}232\bar{1}23$	4	2	7	7	7	R P
11n 99	$\bar{1}2\bar{1}\bar{2}\bar{2}\bar{3}2\bar{1}2\bar{3}$	4	1	5	-5	5	R N
11n 105	$11\bar{2}1\bar{2}322233$	4	2	7	7	7	R P
11n 106	$\bar{1}\bar{1}\bar{1}2111\bar{3}2\bar{3}\bar{3}$	4	1	5	-1	5	R
11n 108	$\bar{1}\bar{1}2\bar{1}\bar{1}\bar{3}\bar{2}1\bar{2}\bar{3}$	4	2	7	-7	7	R N
11n 109	$\bar{1}\bar{1}\bar{1}2\bar{1}\bar{1}\bar{3}\bar{2}1\bar{2}\bar{3}$	4	2	7	-7	7	R N
11n 110	$1112\bar{1}\bar{1}32\bar{1}2\bar{3}$	4	1	5	1	5	R
11n 113	$\bar{1}\bar{1}\bar{2}13\bar{2}\bar{2}\bar{4}\bar{3}2\bar{3}\bar{4}$	5	1	6	-6	6	R N
11n 118	$11121\bar{3}\bar{2}1\bar{2}32$	4	2	7	7	7	R P
11n 122	$\bar{1}\bar{1}\bar{1}2\bar{1}2\bar{3}\bar{2}1\bar{2}\bar{3}$	4	1	5	-5	5	R N
11n 127	$\bar{1}\bar{1}\bar{2}11\bar{2}\bar{2}\bar{1}\bar{3}2\bar{3}$	4	1	5	-5	5	R N
11n 134	$\bar{1}\bar{1}\bar{2}1\bar{2}\bar{2}32\bar{1}2\bar{3}$	4	1	5	-5	5	R N
11n 136	$1112\bar{1}3\bar{2}113222$	4	3	9	9	9	R P
11n 138	$\bar{1}2\bar{1}\bar{2}\bar{3}2\bar{1}222\bar{3}$	4	1	5	-1	5	R
11n 139	$112\bar{1}3\bar{2}34\bar{3}2\bar{3}4$	5	0	4	4	4	R P
11n 142	$112\bar{1}3\bar{2}\bar{4}3\bar{2}3\bar{4}\bar{3}$	5	1	6	-2	6	R
11n 143	$11\bar{2}1\bar{2}\bar{1}3\bar{2}\bar{1}2\bar{3}$	4	1	5	1	5	R
11n 144	$11\bar{2}1132\bar{1}233$	4	2	7	7	7	R P
11n 162	$112\bar{1}213\bar{2}\bar{1}324\bar{3}4$	5	1	6	6	6	R P
11n 174	$11\bar{2}13\bar{2}13223$	4	2	7	7	7	R P
11n 176	$\bar{1}\bar{1}2\bar{1}2\bar{1}\bar{3}\bar{2}1\bar{2}\bar{3}$	4	1	5	-5	5	R N
11n 180	$11112\bar{1}3\bar{2}11322$	4	3	9	9	9	R P
11n 183	$1\bar{2}132213223$	4	3	9	9	9	R P
11n 185	$1\bar{2}13\bar{2}132223$	4	2	7	7	7	R P
12a 43	$\bar{1}2231341223\bar{2}44$	5	3	10	10	10	R P
12a 52	$1223131111\bar{2}31$	4	4	11	11	11	R P
12a 53	$12231313\bar{4}\bar{3}\bar{2}433$	5	3	10	10	10	R P
12a 55	$12234141\bar{4}\bar{3}\bar{2}433$	5	3	10	10	10	R P
12a 56	$1\bar{2}12344\bar{5}3\bar{4}252\bar{3}5$	6	2	9	9	9	R P
12a 82	$\bar{1}2211233423\bar{2}44$	5	3	10	10	10	R P
12a 93	$1112231311\bar{2}31$	4	4	11	11	11	R P

12a 94	$12231313\bar{2}34\bar{3}43$	5	3	10	10	10	R	P
12a 96	$1223\bar{4}3141\bar{2}3333$	5	3	10	10	10	R	P
12a 97	$123445352\bar{4}5\bar{3}21\bar{2}$	6	2	9	9	9	R	P
12a 143	$111112231\bar{3}\bar{2}31$	4	4	11	11	11	R	P
12a 144	$1223112\bar{3}334\bar{3}43$	5	3	10	10	10	R	P
12a 145	$1223112\bar{3}444\bar{3}43$	5	3	10	10	10	R	P
12a 152	$12344535\bar{4}532\bar{1}\bar{2}\bar{3}$	6	2	9	9	9	R	P
12a 255	$\bar{1}2345545\bar{4}3\bar{4}2\bar{1}\bar{2}\bar{3}$	6	2	9	5	9	R	
12a 270	$1\bar{2}12\bar{3}4\bar{3}45\bar{4}3\bar{4}3\bar{5}22\bar{3}$	6	1	7	5	7	R	
12a 276	$122223131\bar{1}\bar{2}31$	4	4	11	11	11	R	P
12a 277	$1\bar{2}123333422\bar{3}44$	5	3	10	10	10	R	P
12a 293	$\bar{1}2231\bar{3}4122232\bar{3}44$	5	3	10	10	10	R	P
12a 295	$\bar{1}2211\bar{2}\bar{3}4243334$	5	3	10	10	10	R	P
12a 319	$\bar{1}2231312\bar{3}4222\bar{3}43$	5	3	10	10	10	R	P
12a 320	$112\bar{1}23\bar{2}\bar{4}325433345$	6	2	9	9	9	R	P
12a 344	$1\bar{2}123342424\bar{3}44$	5	3	10	10	10	R	P
12a 345	$1\bar{2}12334\bar{5}4252\bar{3}44$	6	2	9	9	9	R	P
12a 355	$1\bar{2}313342432444$	5	3	10	10	10	R	P
12a 356	$1\bar{2}12\bar{3}4522333\bar{4}54$	6	2	9	9	9	R	P
12a 367	$111222231\bar{3}\bar{2}31$	4	4	11	11	11	R	P
12a 368	$1222231\bar{1}\bar{2}34\bar{3}43$	5	3	10	10	10	R	P
12a 386	$\bar{1}2\bar{1}2\bar{3}\bar{3}\bar{4}\bar{2}\bar{3}\bar{2}\bar{4}\bar{2}3$	5	2	8	-6	8	R	
12a 391	$\bar{1}2231313323\bar{4}2\bar{3}43$	5	3	10	10	10	R	P
12a 392	$112\bar{1}23\bar{2}\bar{4}325433455$	6	2	9	9	9	R	P
12a 420	$1\bar{2}123342\bar{2}\bar{3}4444$	5	3	10	10	10	R	P
12a 421	$1\bar{2}12334223\bar{4}5\bar{4}54$	6	2	9	9	9	R	P
12a 432	$\bar{1}2231\bar{3}\bar{1}234222\bar{3}43$	5	3	10	10	10	R	P
12a 442	$1\bar{2}341122222\bar{3}43$	5	3	10	10	10	R	P
12a 444	$1\bar{2}3\bar{4}51\bar{3}524\bar{3}2445$	6	2	9	7	9	R	
12a 542	$\bar{1}2\bar{1}23\bar{2}34\bar{3}\bar{4}54\bar{3}4352$	6	2	9	5	9	R	
12a 556	$\bar{1}2211232\bar{3}4\bar{3}22\bar{3}\bar{4}$	5	2	8	6	8	R	
12a 563	$\bar{1}22221123\bar{2}\bar{3}43\bar{2}3\bar{4}$	5	2	8	6	8	R	
12a 564	$1\bar{2}12\bar{3}23\bar{4}34\bar{5}4\bar{3}4\bar{3}\bar{5}2$	6	1	7	5	7	R	
12a 568	$\bar{1}2211232\bar{3}334\bar{3}2\bar{3}\bar{4}$	5	2	8	6	8	R	
12a 574	$122222231\bar{3}\bar{2}31$	4	4	11	11	11	R	P
12a 575	$1\bar{2}311222334\bar{3}43$	5	3	10	10	10	R	P
12a 586	$\bar{1}2231312223\bar{4}2\bar{3}43$	5	3	10	10	10	R	P
12a 610	$1112\bar{1}3\bar{2}1322\bar{4}3\bar{4}5\bar{4}5$	6	2	9	9	9	R	P
12a 615	$\bar{1}2231312\bar{3}\bar{4}24334\bar{4}$	5	3	10	10	10	R	P
12a 623	$\bar{1}2211232\bar{3}4443\bar{2}3\bar{4}$	5	2	8	6	8	R	
12a 638	$\bar{1}22\bar{3}2231312\bar{3}4\bar{3}\bar{2}4$	5	2	8	6	8	R	
12a 647	$1122231112\bar{1}33$	4	4	11	11	11	R	P
12a 648	$1\bar{2}313342422\bar{3}24$	5	3	10	10	10	R	P
12a 653	$1112\bar{1}23\bar{2}\bar{4}32433\bar{5}\bar{4}5$	6	2	9	9	9	R	P
12a 677	$\bar{1}2211\bar{2}3\bar{4}\bar{3}2\bar{3}\bar{2}3\bar{4}34$	5	1	6	4	6	R	
12a 679	$123\bar{4}5352414\bar{2}45\bar{3}$	6	2	9	9	9	R	P
12a 685	$\bar{1}2211232\bar{3}44\bar{3}22\bar{3}\bar{4}$	5	2	8	6	8	R	
12a 735	$1\bar{2}123\bar{2}3\bar{4}34\bar{5}4\bar{3}4\bar{3}\bar{5}2$	6	1	7	5	7	R	

12a 750	$122\bar{3}\bar{2}\bar{1}3\bar{4}3\bar{4}54\bar{3}4\bar{2}\bar{5}322$	6	1	7	5	7	R
12a 787	$1\bar{2}12\bar{3}4\bar{3}4\bar{5}4\bar{3}4\bar{3}\bar{5}22\bar{3}$	6	2	9	-1	9	R
12a 803	$1\bar{2}3\bar{4}5\bar{6}135\bar{6}24\bar{5}4\bar{3}2$	7	1	8	4	8	R
12a 811	$1223131\bar{1}\bar{2}3133$	4	4	11	11	11	R P
12a 813	$1112231\bar{3}\bar{2}3133$	4	4	11	11	11	R P
12a 817	$1122313\bar{1}\bar{2}3133$	4	4	11	11	11	R P
12a 876	$1112231333\bar{2}31$	4	4	11	11	11	R P
12a 877	$\bar{1}23342\bar{3}1412444$	5	3	10	10	10	R P
12a 900	$\bar{1}22\bar{3}42231312\bar{3}234$	5	3	10	10	10	R P
12a 908	$1\bar{2}12\bar{3}4\bar{3}223\bar{4}3\bar{4}22\bar{3}$	5	2	8	4	8	R
12a 938	$\bar{1}2\bar{3}\bar{2}3\bar{2}3131334\bar{3}24$	5	2	8	6	8	R
12a 953	$\bar{1}2\bar{3}32\bar{1}12\bar{3}2334\bar{3}43$	5	2	8	6	8	R
12a 973	$1\bar{2}3122\bar{3}4\bar{3}\bar{2}324444$	5	3	10	10	10	R P
12a 995	$1\bar{2}3122223\bar{4}3\bar{2}3244$	5	3	10	10	10	R P
12a 996	$112\bar{1}23\bar{2}\bar{4}\bar{3}25433445$	6	2	9	9	9	R P
12a 1004	$1\bar{2}3134242\bar{4}3\bar{2}32\bar{4}3$	5	3	10	10	10	R P
12a 1035	$\bar{1}2\bar{3}4231122333\bar{3}\bar{2}4$	5	3	10	10	10	R P
12a 1037	$112\bar{1}223\bar{2}4\bar{3}2433\bar{5}\bar{4}5$	6	2	9	9	9	R P
12a 1097	$112\bar{1}2\bar{3}\bar{2}14\bar{3}432234\bar{5}\bar{4}5$	6	2	9	9	9	R P
12a 1112	$\bar{1}2\bar{3}4231313323\bar{3}234$	5	3	10	10	10	R P
12a 1166	$1\bar{2}3\bar{4}5\bar{6}135\bar{6}24\bar{5}4\bar{3}2$	7	1	8	2	8	R
12a 1185	$1\bar{2}\bar{3}41\bar{3}4223\bar{4}3\bar{2}\bar{3}24$	5	2	8	4	8	R
12a 1287	$\bar{1}2\bar{3}456\bar{1}\bar{3}\bar{5}6\bar{2}\bar{4}5432\bar{2}$	7	1	8	0	8	R
12n 5	$\bar{1}2\bar{1}234\bar{3}4\bar{3}\bar{4}2\bar{4}\bar{2}\bar{3}$	5	1	6	-6	6	R N
12n 13	$\bar{1}2\bar{1}2\bar{3}45\bar{3}\bar{4}\bar{2}5\bar{2}\bar{3}$	6	1	7	-3	7	R
12n 25	$\bar{1}234\bar{1}\bar{2}34\bar{3}\bar{2}\bar{3}4$	5	1	6	0	6	R
12n 46	$\bar{1}23\bar{1}\bar{2}3245\bar{3}\bar{4}54$	6	1	7	3	7	R
12n 53	$\bar{1}2\bar{1}2\bar{3}433\bar{2}\bar{2}\bar{3}433$	5	1	6	2	6	R
12n 54	$\bar{1}2\bar{3}\bar{3}\bar{2}1\bar{2}\bar{2}\bar{2}312\bar{1}$	4	2	7	-3	7	R
12n 58	$\bar{1}2\bar{1}2\bar{3}\bar{4}4\bar{3}\bar{2}\bar{2}\bar{3}4\bar{3}\bar{4}$	5	2	8	-8	8	R N
12n 74	$11122\bar{1}2322133$	4	4	11	11	11	R P
12n 77	$1\bar{2}123434342\bar{4}2\bar{3}$	5	3	10	10	10	R P
12n 79	$1\bar{2}123\bar{4}3\bar{4}342\bar{4}2\bar{3}$	5	1	6	6	6	R P
12n 89	$1\bar{2}33221112\bar{3}\bar{2}1$	4	3	9	7	9	R
12n 91	$1221112\bar{3}22133$	4	4	11	11	11	R P
12n 93	$1223134\bar{1}\bar{2}\bar{2}\bar{3}244$	5	3	10	8	10	R
12n 96	$1\bar{2}12334342423\bar{4}\bar{3}$	5	3	10	10	10	R P
12n 97	$\bar{1}2\bar{3}\bar{1}3\bar{2}\bar{3}43334\bar{4}$	5	1	6	-2	6	R
12n 103	$12222313\bar{1}\bar{2}1\bar{2}\bar{3}$	4	3	9	7	9	R
12n 105	$1112212\bar{3}22133$	4	4	11	11	11	R P
12n 110	$12234\bar{3}14312232\bar{2}$	5	3	10	10	10	R P
12n 113	$\bar{1}2\bar{1}2\bar{1}2\bar{1}\bar{2}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R N *
12n 114	$1112\bar{2}\bar{1}122222$	3	3	8	8	8	R P *
12n 117	$1\bar{2}122\bar{2}\bar{3}322333$	4	2	7	7	7	R P
12n 120	$\bar{1}\bar{1}223\bar{1}\bar{2}\bar{2}\bar{3}\bar{3}\bar{2}\bar{1}2$	4	2	7	-5	7	R
12n 121	$\bar{1}\bar{2}11\bar{2}\bar{3}2\bar{1}\bar{2}\bar{2}\bar{3}\bar{3}2$	4	1	5	-5	5	R N
12n 123	$1\bar{2}1\bar{2}3322\bar{3}43344$	5	2	8	8	8	R P
12n 128	$1\bar{2}12\bar{3}342423\bar{4}\bar{3}$	5	2	8	8	8	R P

12n 133	$\bar{1}22112233\bar{2}12222\bar{3}$	4	3	9	9	9	R	P
12n 134	$1122\bar{3}13\bar{1}\bar{2}3133$	4	3	9	7	9	R	
12n 136	$1122112\bar{3}22133$	4	4	11	11	11	R	P
12n 138	$122\bar{3}411\bar{2}\bar{2}33244$	5	3	10	8	10	R	
12n 143	$1\bar{2}\bar{3}41242\bar{3}\bar{2}4$	5	2	8	4	8	R	
12n 148	$\bar{1}\bar{2}\bar{2}\bar{3}2\bar{1}\bar{2}\bar{2}\bar{2}\bar{3}\bar{3}2$	4	3	9	-9	9	R	N
12n 149	$\bar{1}2\bar{1}\bar{2}\bar{3}\bar{3}43\bar{2}\bar{3}\bar{3}\bar{2}43$	5	2	8	-8	8	R	N
12n 153	$1212\bar{3}22313212$	4	4	11	11	11	R	P
12n 154	$\bar{1}21\bar{2}3\bar{2}\bar{2}\bar{3}1\bar{3}\bar{2}12$	4	1	5	-1	5	R	
12n 155	$122\bar{1}23\bar{2}\bar{2}33221$	4	2	7	7	7	R	P
12n 157	$1\bar{2}3412\bar{2}\bar{3}423\bar{2}$	5	1	6	6	6	R	P
12n 159	$\bar{1}21\bar{2}3\bar{2}\bar{2}\bar{3}1\bar{3}\bar{2}\bar{1}2$	4	1	5	-5	5	R	N
12n 166	$12222\bar{1}2322133$	4	4	11	11	11	R	P
12n 169	$12211\bar{2}\bar{3}2334\bar{3}43$	5	3	10	10	10	R	P
12n 176	$12211\bar{2}\bar{3}\bar{2}34\bar{3}2\bar{3}4$	5	1	6	6	6	R	P
12n 177	$1\bar{2}123443223\bar{3}\bar{4}3$	5	3	10	10	10	R	P
12n 183	$1\bar{2}3412\bar{4}2\bar{3}423$	5	2	8	8	8	R	P
12n 185	$11\bar{2}\bar{3}122313\bar{1}\bar{2}1$	4	3	9	7	9	R	
12n 187	$1222212\bar{3}22133$	4	4	11	11	11	R	P
12n 190	$1\bar{2}\bar{1}2\bar{1}\bar{1}\bar{1}\bar{2}1\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R	N *
12n 191	$1112222\bar{1}\bar{1}222$	3	3	8	8	8	R	P
12n 194	$111\bar{2}31312\bar{3}2\bar{1}2$	4	2	7	7	7	R	P
12n 199	$\bar{1}2\bar{1}\bar{2}\bar{3}\bar{3}\bar{2}\bar{2}\bar{2}333$	4	2	7	-5	7	R	
12n 200	$1\bar{2}\bar{3}42\bar{3}4\bar{3}\bar{2}3\bar{1}4\bar{1}\bar{2}$	5	1	6	-4	6	R	
12n 203	$12231312\bar{3}42\bar{3}43$	5	3	10	10	10	R	P
12n 211	$1\bar{2}31\bar{2}34\bar{3}222\bar{3}\bar{2}\bar{4}$	5	1	6	2	6	R	
12n 215	$\bar{1}223131\bar{2}34\bar{3}4\bar{2}3$	5	2	8	6	8	R	
12n 217	$12211\bar{2}34222\bar{3}43$	5	3	10	10	10	R	P
12n 221	$1\bar{2}31334\bar{2}\bar{3}\bar{3}\bar{3}4$	5	1	6	2	6	R	
12n 222	$12231341\bar{2}32\bar{4}\bar{3}\bar{2}$	5	2	8	8	8	R	P
12n 224	$1\bar{2}3133\bar{4}\bar{2}\bar{3}\bar{3}\bar{3}4$	5	1	6	-2	6	R	
12n 233	$1\bar{2}\bar{1}2\bar{1}\bar{2}\bar{1}\bar{2}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R	N *
12n 234	$1\bar{2}\bar{2}112222222$	3	3	8	8	8	R	P *
12n 237	$1\bar{2}\bar{2}3322222\bar{1}\bar{2}3$	4	2	7	7	7	R	P
12n 240	$111\bar{2}\bar{2}33222\bar{1}\bar{2}3$	4	2	7	7	7	R	P
12n 242	122112222222	3	5	12	12	12	R	P
12n 243	$1223322222\bar{1}\bar{2}3$	4	4	11	11	11	R	P
12n 244	$1112233222\bar{1}\bar{2}3$	4	4	11	11	11	R	P
12n 245	$\bar{1}2342\bar{3}43231312$	5	3	10	10	10	R	P
12n 248	$1\bar{2}\bar{3}4\bar{3}4\bar{3}424\bar{1}\bar{3}\bar{1}\bar{2}$	5	1	6	-4	6	R	
12n 249	$1\bar{2}\bar{2}1122234\bar{2}\bar{3}43$	5	1	6	6	6	R	P
12n 250	$12211\bar{2}\bar{2}\bar{3}\bar{4}23\bar{4}\bar{3}$	5	1	6	0	6	R	
12n 251	$1221122234\bar{2}\bar{3}43$	5	3	10	10	10	R	P
12n 254	$122311\bar{2}\bar{2}3321\bar{2}$	4	2	7	7	7	R	P
12n 259	$12223\bar{3}\bar{1}222\bar{3}\bar{1}21$	4	3	9	9	9	R	P
12n 262	$\bar{1}2\bar{1}\bar{2}\bar{3}\bar{3}2\bar{3}\bar{4}2333\bar{4}$	5	1	6	-2	6	R	
12n 269	$\bar{1}2\bar{1}2344\bar{3}\bar{2}\bar{2}3\bar{4}\bar{3}\bar{4}$	5	2	8	0	8	R	
12n 270	$\bar{1}2\bar{1}234\bar{3}\bar{3}2\bar{2}3\bar{4}\bar{3}\bar{3}$	5	1	6	-6	6	R	N

12n 273	$1\bar{2}1\bar{2}\bar{3}4222\bar{3}\bar{4}$	5	2	8	4	8	R	
12n 275	$1\bar{2}3\bar{4}1\bar{2}\bar{4}\bar{3}2\bar{3}\bar{4}3$	5	1	6	-2	6	R	
12n 276	$\bar{1}\bar{2}\bar{2}1\bar{2}\bar{2}3\bar{1}\bar{3}\bar{2}1\bar{2}\bar{3}$	4	3	9	-9	9	R	N
12n 282	$\bar{1}2\bar{3}\bar{4}\bar{1}\bar{2}3\bar{2}\bar{3}\bar{4}\bar{3}2$	5	1	6	-4	6	R	
12n 286	$\bar{1}2\bar{3}\bar{1}2\bar{3}43\bar{2}324$	5	1	6	2	6	R	
12n 289	$1\bar{2}123342423\bar{3}\bar{4}3$	5	3	10	10	10	R	P
12n 292	$1221112\bar{3}22133$	4	4	11	11	11	R	P
12n 303	$1\bar{2}12\bar{3}\bar{3}2233333$	4	2	7	7	7	R	P
12n 305	$1\bar{2}12332233333$	4	4	11	11	11	R	P
12n 306	$1\bar{2}12\bar{3}\bar{3}22333\bar{4}\bar{3}4$	5	1	6	6	6	R	P
12n 308	$1\bar{2}123322333\bar{4}\bar{3}4$	5	3	10	10	10	R	P
12n 310	$1\bar{2}123322\bar{3}\bar{3}\bar{4}3\bar{4}\bar{4}$	5	1	6	2	6	R	
12n 316	$1\bar{2}12\bar{3}22\bar{3}22333$	4	2	7	7	7	R	P
12n 320	$1\bar{2}\bar{3}1\bar{3}22\bar{1}2\bar{3}\bar{2}1\bar{2}$	4	1	5	-1	5	R	
12n 321	$\bar{1}23132\bar{1}231\bar{2}12$	4	2	7	7	7	R	P
12n 322	$1\bar{2}3134\bar{2}\bar{3}24\bar{3}\bar{4}$	5	1	6	0	6	R	
12n 328	$1231322\bar{1}23212$	4	4	11	11	11	R	P
12n 329	$\bar{1}\bar{2}\bar{2}\bar{3}\bar{1}\bar{1}2\bar{1}\bar{2}\bar{2}\bar{3}\bar{3}2$	4	3	9	-9	9	R	N
12n 331	$\bar{1}\bar{2}\bar{2}\bar{1}\bar{1}\bar{2}32\bar{3}43\bar{2}34$	5	2	8	-4	8	R	
12n 332	$\bar{1}2\bar{1}\bar{2}34\bar{3}\bar{3}\bar{2}\bar{2}3\bar{4}\bar{3}\bar{3}$	5	2	8	-8	8	R	N
12n 334	$1\bar{2}\bar{2}\bar{1}\bar{1}\bar{2}3\bar{2}34\bar{3}2\bar{3}4$	5	1	6	-2	6	R	
12n 336	$\bar{1}2\bar{3}4\bar{1}\bar{3}\bar{1}2\bar{3}\bar{2}\bar{4}\bar{2}3\bar{4}$	5	2	8	-4	8	R	
12n 338	$1\bar{2}12333322333$	4	4	11	11	11	R	P
12n 340	$1\bar{2}12\bar{3}\bar{3}\bar{3}\bar{3}22333$	4	1	5	3	5	R	
12n 341	$1\bar{2}1234\bar{3}4322333$	5	3	10	10	10	R	P
12n 342	$1\bar{2}\bar{2}\bar{1}\bar{1}\bar{2}34222\bar{3}43$	5	1	6	2	6	R	
12n 344	$\bar{1}\bar{1}\bar{1}\bar{2}\bar{2}\bar{2}\bar{2}\bar{1}\bar{1}222$	3	2	6	-6	6	R	N *
12n 347	$\bar{1}2\bar{1}\bar{2}\bar{3}\bar{3}\bar{3}\bar{2}\bar{2}333$	4	1	5	-5	5	R	N
12n 349	$\bar{1}2\bar{3}\bar{1}2\bar{3}\bar{2}\bar{4}\bar{2}\bar{2}3\bar{4}$	5	2	8	-6	8	R	
12n 355	$\bar{1}\bar{2}34\bar{2}\bar{3}\bar{1}2\bar{3}24\bar{3}$	5	1	6	-4	6	R	
12n 366	$\bar{1}2\bar{3}\bar{3}2\bar{2}3\bar{1}32\bar{3}\bar{2}\bar{2}$	4	3	9	-9	9	R	N
12n 368	$1112231312\bar{1}2\bar{3}$	4	3	9	7	9	R	
12n 371	$1\bar{2}\bar{3}1\bar{3}22\bar{1}232\bar{1}2$	4	1	5	3	5	R	
12n 373	$1223\bar{1}32211\bar{2}\bar{3}1$	4	2	7	7	7	R	P
12n 374	$1\bar{2}12332233223$	4	4	11	11	11	R	P
12n 375	$1\bar{2}123\bar{2}3222\bar{3}\bar{2}3$	4	2	7	7	7	R	P
12n 379	$\bar{1}2\bar{1}\bar{2}3\bar{2}\bar{2}\bar{3}1\bar{3}\bar{2}\bar{1}\bar{2}$	4	1	5	-5	5	R	N
12n 381	$1\bar{2}3412\bar{4}\bar{3}2\bar{3}43$	5	1	6	6	6	R	P
12n 383	$1\bar{2}3411\bar{2}\bar{3}243\bar{4}$	5	1	6	6	6	R	P
12n 386	$122\bar{3}112233212$	4	4	11	11	11	R	P
12n 395	$1\bar{2}\bar{3}\bar{4}\bar{2}3\bar{1}\bar{1}\bar{2}\bar{2}\bar{3}\bar{3}2\bar{4}$	5	2	8	-6	8	R	
12n 398	$\bar{1}234\bar{1}\bar{2}43\bar{2}\bar{3}\bar{3}\bar{4}$	5	2	8	-6	8	R	
12n 402	$\bar{1}\bar{1}\bar{1}\bar{2}3\bar{2}\bar{3}\bar{1}23\bar{2}\bar{1}\bar{3}$	4	3	9	-9	9	R	N
12n 406	$1\bar{2}12333423\bar{2}433$	5	3	10	10	10	R	P
12n 407	$1\bar{2}123\bar{2}32\bar{3}2333$	4	2	7	7	7	R	P
12n 414	$\bar{1}2\bar{3}4\bar{1}\bar{3}2\bar{3}\bar{2}34\bar{3}\bar{2}4$	5	0	4	-2	4	R	
12n 417	$\bar{1}22112221122$	3	4	10	10	10	R	P
12n 419	$122112233\bar{2}1\bar{2}\bar{3}$	4	3	9	7	9	R	

12n 426	$123132\bar{1}231212$	4	4	11	11	11	R	P
12n 432	$12\bar{3}4\bar{2}23\bar{1}3\bar{4}1\bar{2}\bar{3}4$	5	2	8	-8	8	R	N
12n 436	$\bar{1}23\bar{4}\bar{1}\bar{2}3\bar{2}\bar{2}\bar{3}\bar{3}\bar{4}$	5	2	8	-6	8	R	
12n 438	$1\bar{2}33222\bar{1}2\bar{3}\bar{1}\bar{2}1$	4	2	7	3	7	R	
12n 439	$\bar{1}\bar{2}\bar{3}1\bar{3}2\bar{1}2\bar{3}1\bar{2}\bar{1}2$	4	1	5	-3	5	R	
12n 441	$11\bar{2}\bar{3}1\bar{2}1223322$	4	2	7	7	7	R	P
12n 443	$\bar{1}\bar{2}\bar{3}1\bar{3}2\bar{1}231\bar{2}\bar{1}2$	4	1	5	-1	5	R	
12n 444	$1\bar{2}3\bar{4}131\bar{2}\bar{3}4\bar{3}\bar{4}$	5	1	6	0	6	R	
12n 451	$\bar{1}2\bar{3}\bar{1}2\bar{3}\bar{2}3\bar{2}\bar{3}2\bar{3}\bar{2}$	4	1	5	-5	5	R	N
12n 452	$\bar{1}2313\bar{2}1\bar{2}\bar{3}12\bar{1}2$	4	1	5	3	5	R	
12n 453	$12233223411\bar{2}\bar{3}4$	5	3	10	10	10	R	P
12n 454	$\bar{1}2\bar{1}\bar{2}3\bar{2}3\bar{2}\bar{3}\bar{2}\bar{3}\bar{3}$	4	1	5	-5	5	R	N
12n 456	$\bar{1}\bar{2}\bar{2}\bar{1}\bar{1}\bar{2}34222\bar{3}43$	5	1	6	0	6	R	
12n 464	$1\bar{2}332\bar{1}2\bar{1}22\bar{3}\bar{2}1$	4	1	5	3	5	R	
12n 466	$1\bar{2}\bar{2}\bar{2}1\bar{2}\bar{1}2\bar{1}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R	N *
12n 467	$1\bar{2}\bar{2}\bar{2}\bar{2}1122222$	3	1	4	4	4	R	P *
12n 468	$12222112\bar{2}\bar{2}\bar{2}\bar{2}$	3	1	4	2	4	R	
12n 469	$\bar{1}2\bar{1}\bar{2}\bar{2}3\bar{1}3\bar{2}3\bar{2}\bar{1}2$	4	1	5	-5	5	R	N
12n 472	122221122222	3	5	12	12	12	R	P
12n 473	$1222233222\bar{1}\bar{2}3$	4	4	11	11	11	R	P
12n 474	$1233222223\bar{2}1\bar{2}$	4	4	11	11	11	R	P
12n 477	$\bar{1}234\bar{3}422323112$	5	3	10	10	10	R	P
12n 478	$1\bar{2}12\bar{3}2\bar{3}\bar{4}\bar{4}32433$	5	1	6	2	6	R	
12n 483	$\bar{1}233\bar{2}\bar{2}11\bar{2}\bar{3}2\bar{1}2$	4	1	5	1	5	R	
12n 487	$\bar{1}\bar{2}33221123\bar{2}\bar{1}2$	4	1	5	5	5	R	P
12n 488	$1\bar{2}123\bar{2}32\bar{3}2\bar{3}\bar{3}\bar{3}$	4	1	5	1	5	R	
12n 491	$1\bar{2}\bar{3}\bar{4}134\bar{2}\bar{3}\bar{2}\bar{3}22\bar{3}$	5	1	6	-2	6	R	
12n 496	$1\bar{2}3412\bar{3}24233$	5	2	8	8	8	R	P
12n 502	$123131112\bar{3}212$	4	4	11	11	11	R	P
12n 503	$\bar{1}2323231343\bar{1}24$	5	3	10	10	10	R	P
12n 505	$\bar{1}2\bar{3}4\bar{1}4\bar{3}2\bar{3}4\bar{3}4$	5	2	8	-4	8	R	
12n 510	$\bar{1}2\bar{3}\bar{1}\bar{3}\bar{4}\bar{2}3\bar{2}\bar{4}\bar{2}\bar{3}$	5	2	8	-8	8	R	N
12n 511	$\bar{1}223\bar{1}3\bar{2}\bar{2}1\bar{2}\bar{1}\bar{2}\bar{3}$	4	1	5	-1	5	R	
12n 515	$\bar{1}2231312\bar{3}4\bar{2}\bar{3}43$	5	2	8	6	8	R	
12n 518	$\bar{1}223132223122$	4	4	11	11	11	R	P
12n 519	$\bar{1}2\bar{1}\bar{2}\bar{3}2\bar{3}2\bar{3}\bar{4}3\bar{4}$	5	1	6	-2	6	R	
12n 520	$\bar{1}2\bar{3}4\bar{1}\bar{3}\bar{4}\bar{2}3\bar{2}\bar{4}\bar{3}$	5	1	6	-6	6	R	N
12n 522	$\bar{1}2\bar{1}\bar{2}\bar{3}2\bar{3}\bar{2}\bar{3}\bar{2}\bar{2}\bar{3}$	4	1	5	-5	5	R	N
12n 523	$1\bar{2}\bar{3}\bar{4}3\bar{4}22\bar{3}\bar{2}3\bar{1}\bar{1}\bar{2}$	5	1	6	-4	6	R	
12n 526	$1122\bar{3}1\bar{2}1\bar{2}\bar{2}332$	4	2	7	5	7	R	
12n 528	$\bar{1}\bar{1}\bar{2}\bar{2}\bar{3}\bar{1}2\bar{1}\bar{2}\bar{2}\bar{3}\bar{3}2$	4	3	9	-9	9	R	N
12n 549	$1\bar{2}12\bar{3}22313\bar{2}1\bar{2}$	4	2	7	5	7	R	
12n 554	$\bar{1}2\bar{2}345\bar{4}32\bar{1}4\bar{2}\bar{5}2\bar{3}2\bar{4}$	6	1	7	-3	7	R	
12n 570	$\bar{1}\bar{2}\bar{2}\bar{2}\bar{2}\bar{2}\bar{1}\bar{1}222$	3	2	6	-6	6	R	N *
12n 572	$\bar{1}2\bar{3}\bar{1}\bar{3}2\bar{3}\bar{2}\bar{1}\bar{2}1\bar{2}$	4	1	5	-5	5	R	N
12n 574	122222211222	3	5	12	12	12	R	P
12n 575	$123322232221\bar{2}$	4	4	11	11	11	R	P
12n 576	$1\bar{2}31332223332$	4	4	11	11	11	R	P

12n 577	$1\bar{2}123\bar{2}3222\bar{3}2\bar{3}$	4	1	5	5	5	R P
12n 581	$1234\bar{2}\bar{3}43211222$	5	3	10	10	10	R P
12n 582	$1\bar{2}1234\bar{2}432\bar{3}4\bar{2}\bar{3}$	5	0	4	4	4	R P
12n 585	$1\bar{2}123424323\bar{4}23$	5	3	10	10	10	R P
12n 591	$1233221123\bar{2}1\bar{2}$	4	4	11	11	11	R P
12n 592	$\bar{1}2\bar{3}\bar{1}\bar{3}\bar{3}\bar{3}\bar{2}\bar{3}\bar{2}33$	4	2	7	-5	7	R
12n 593	$1\bar{2}1\bar{2}\bar{3}424332\bar{4}3\bar{4}$	5	2	8	6	8	R
12n 594	$1233221\bar{1}2\bar{3}2\bar{1}\bar{2}$	4	3	9	9	9	R P
12n 600	$1\bar{2}3132234\bar{3}4232$	5	3	10	10	10	R P
12n 603	$123\bar{1}3\bar{2}\bar{1}\bar{2}\bar{3}1212$	4	3	9	7	9	R
12n 604	$\bar{1}22\bar{1}\bar{1}2\bar{1}\bar{1}\bar{2}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R N *
12n 608	$\bar{1}2\bar{1}\bar{2}32\bar{3}\bar{2}3\bar{4}\bar{3}4$	5	1	6	-2	6	R
12n 609	$1\bar{2}313\bar{2}322\bar{3}22\bar{3}$	4	2	7	5	7	R
12n 610	$1\bar{2}313\bar{2}\bar{2}32\bar{3}2\bar{3}\bar{3}$	4	1	5	1	5	R
12n 617	$\bar{1}2\bar{3}\bar{1}\bar{3}\bar{4}\bar{2}3\bar{2}\bar{4}3\bar{4}$	5	2	8	-6	8	R
12n 626	$\bar{1}2\bar{3}\bar{4}\bar{1}\bar{3}\bar{2}\bar{2}3\bar{2}3\bar{4}\bar{3}\bar{3}$	5	2	8	-8	8	R N
12n 629	$\bar{1}2\bar{3}\bar{1}23\bar{2}\bar{4}3\bar{2}3\bar{4}$	5	1	6	-4	6	R
12n 631	$\bar{1}2\bar{3}\bar{1}2\bar{3}243\bar{2}34$	5	1	6	2	6	R
12n 638	$122\bar{3}112233\bar{2}1\bar{2}$	4	3	9	9	9	R P
12n 639	$1233\bar{2}111\bar{2}\bar{3}121$	4	3	9	7	9	R
12n 640	$\bar{1}22112222112$	3	4	10	10	10	R P
12n 641	$\bar{1}2\bar{3}\bar{1}\bar{3}\bar{3}2\bar{3}\bar{2}\bar{2}33$	4	2	7	-5	7	R
12n 643	$\bar{1}233211\bar{2}34\bar{3}4\bar{2}3$	5	2	8	6	8	R
12n 644	$122231312\bar{3}21\bar{2}$	4	3	9	9	9	R P
12n 647	$\bar{1}22112211222$	3	4	10	10	10	R P
12n 649	$1\bar{2}33221123\bar{2}1\bar{2}$	4	2	7	5	7	R
12n 650	$1\bar{2}3322\bar{1}\bar{1}2\bar{3}\bar{2}1\bar{2}$	4	1	5	1	5	R
12n 654	$1\bar{2}1\bar{2}\bar{3}424323\bar{4}2\bar{3}$	5	2	8	6	8	R
12n 655	$1\bar{2}12332233223\bar{3}$	4	3	9	9	9	R P
12n 658	$1\bar{2}31332223\bar{2}3\bar{3}$	4	2	7	5	7	R
12n 660	$\bar{1}2\bar{3}\bar{3}\bar{2}\bar{2}3\bar{1}\bar{3}\bar{2}3\bar{2}\bar{2}$	4	3	9	-9	9	R N
12n 666	$1\bar{2}1\bar{2}1\bar{2}122222$	3	2	6	6	6	R P *
12n 668	$1\bar{2}3132221\bar{3}\bar{2}1\bar{2}$	4	2	7	5	7	R
12n 671	$1\bar{2}\bar{2}1112322133$	4	3	9	9	9	R P
12n 674	$111\bar{2}\bar{2}1122222$	3	3	8	8	8	R P *
12n 679	111221122222	3	5	12	12	12	R P
12n 680	$1\bar{2}12223322333$	4	4	11	11	11	R P
12n 682	$11\bar{2}\bar{2}112322133$	4	3	9	9	9	R P
12n 683	$\bar{1}2\bar{1}2\bar{1}\bar{1}\bar{1}2\bar{1}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R N *
12n 684	$\bar{1}2\bar{1}2\bar{1}\bar{1}\bar{1}2\bar{1}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R N *
12n 688	111222211222	3	5	12	12	12	R P
12n 689	$122231311\bar{2}\bar{3}21$	4	4	11	11	11	R P
12n 691	$123321112\bar{3}221$	4	4	11	11	11	R P
12n 692	$1233221112\bar{3}21$	4	4	11	11	11	R P
12n 693	$122\bar{3}1122334\bar{2}\bar{3}4$	5	3	10	8	10	R
12n 694	$1223112233\bar{2}1\bar{2}$	4	4	11	11	11	R P
12n 707	$1\bar{2}\bar{2}\bar{1}2\bar{1}2\bar{1}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R N *
12n 708	$1\bar{2}\bar{2}\bar{2}1\bar{2}12\bar{1}222$	3	0	2	2	2	R P *

12n 719	$1\bar{2}123\bar{2}32\bar{3}2333$	4	1	5	5	5	R P
12n 720	$\bar{1}2\bar{3}42313\bar{1}2\bar{3}\bar{2}34$	5	2	8	6	8	R
12n 722	$1\bar{2}\bar{2}111122222$	3	3	8	8	8	R P *
12n 724	$1\bar{2}31311\bar{2}32\bar{1}\bar{2}$	4	2	7	7	7	R P
12n 725	122111122222	3	5	12	12	12	R P
12n 726	$\bar{1}2\bar{3}23\bar{2}3134\bar{3}124$	5	1	6	6	6	R P
12n 740	$1\bar{2}3\bar{1}32\bar{1}2\bar{3}\bar{1}\bar{2}1\bar{2}$	4	1	5	-1	5	R
12n 744	$\bar{1}233211\bar{2}34\bar{2}\bar{3}43$	5	2	8	6	8	R
12n 747	$1\bar{2}\bar{2}\bar{1}\bar{1}22\bar{1}\bar{1}\bar{2}\bar{2}\bar{2}$	3	2	6	-6	6	R N *
12n 749	$122112211\bar{2}\bar{2}\bar{2}$	3	2	6	6	6	R P
12n 750	$1\bar{2}\bar{2}112211222$	3	3	8	8	8	R P
12n 756	$\bar{1}2\bar{1}\bar{2}3342\bar{3}\bar{2}4\bar{2}3$	5	2	8	2	8	R
12n 758	$\bar{1}2\bar{3}42311223324$	5	3	10	10	10	R P
12n 764	$1122\bar{3}1\bar{2}122332$	4	3	9	9	9	R P
12n 767	$1\bar{2}111\bar{2}\bar{2}\bar{1}\bar{1}\bar{1}\bar{1}\bar{2}$	3	1	4	-4	4	R N *
12n 768	$\bar{1}2\bar{1}\bar{2}\bar{3}\bar{3}22\bar{3}\bar{2}\bar{2}33$	4	0	3	-3	3	R N
12n 797	$\bar{1}2\bar{3}23\bar{2}4\bar{3}14312\bar{3}23$	5	1	6	6	6	R P
12n 799	$\bar{1}2\bar{3}23\bar{2}411\bar{2}3\bar{2}34$	5	1	6	4	6	R
12n 801	$12\bar{1}23\bar{2}132\bar{3}233$	4	2	7	7	7	R P
12n 807	$1\bar{2}1232\bar{1}32\bar{3}233$	4	2	7	7	7	R P
12n 811	$\bar{1}21\bar{2}3\bar{2}132\bar{3}233$	4	1	5	5	5	R P
12n 812	$\bar{1}\bar{2}1\bar{2}32\bar{1}32\bar{3}233$	4	1	5	1	5	R
12n 820	$1\bar{2}\bar{2}\bar{1}\bar{1}2\bar{1}\bar{1}\bar{2}\bar{2}\bar{2}\bar{2}$	3	3	8	-8	8	R N *
12n 821	$122\bar{1}\bar{1}211\bar{2}\bar{2}\bar{2}\bar{2}$	3	1	4	0	4	R
12n 822	$1\bar{2}\bar{2}112\bar{1}\bar{1}2222$	3	1	4	4	4	R P *
12n 823	$\bar{1}2\bar{1}\bar{2}3\bar{2}\bar{2}\bar{3}\bar{3}\bar{2}\bar{2}3$	4	2	7	-7	7	R N
12n 825	$1\bar{2}1\bar{2}\bar{2}3132\bar{3}2\bar{1}\bar{2}$	4	1	5	3	5	R
12n 830	$1\bar{2}\bar{2}112221122$	3	3	8	8	8	R P
12n 850	$\bar{1}22221122112$	3	4	10	10	10	R P
12n 851	$1\bar{2}12233223\bar{2}32$	4	3	9	9	9	R P
12n 868	$\bar{1}23\bar{2}311\bar{2}3\bar{2}\bar{1}2\bar{3}$	4	1	5	1	5	R
12n 881	$\bar{1}23\bar{2}324141323\bar{2}34$	5	2	8	8	8	R P
12n 882	$1\bar{2}\bar{2}\bar{1}\bar{1}2\bar{1}\bar{1}\bar{2}\bar{2}\bar{2}$	3	3	8	-8	8	R N *
12n 884	$1\bar{2}123\bar{2}34\bar{2}4\bar{3}2\bar{3}4$	5	1	6	2	6	R
12n 887	$1\bar{2}1\bar{2}1\bar{2}112221$	3	2	6	6	6	R P *
12n 888	111222111222	3	5	12	12	12	R P

According to KnotInfo, whether the knots **12n512** and **12n239** are quasipositive is unknown. Using our bounds method we found both have rank in $\{5, 7\}$. A rank of 5 would show they are quasipositive. Using the templated search method, we were unable to find a band presentation of short conjugate length. Once again, due to Markov's theorem about the variety of braid representations for a given knot, we may need to examine a different

braid to show quasipositivity.

Bibliography

[oei, 2021] (2021). The on-line encyclopedia of integer sequences. URL: `https://oeis.org`. 2021-04-06.

[Bardakov and Bellingeri, 2014] Bardakov, V. and Bellingeri, P. (2014). On representations of braids as automorphisms of free groups and corresponding linear representations.

[Bigelow, 2002] Bigelow, S. (2002). The lawrence-krammer representation. *arXiv: Geometric Topology*.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

[Dehornoy, 1997] Dehornoy, P. (1997). A fast method for comparing braids. *Advances in Mathematics*, 125(2):200–235.

[Dummit and Foote, 2003] Dummit, D. and Foote, R. (2003). *Abstract Algebra*. Wiley.

[Grigsby, 2018] Grigsby, J. E. (2018). On braided, banded surfaces and ribbon obstructions.

[Livingston and Moore, 2021] Livingston, C. and Moore, A. H. (2021). Knotinfo: Table of knot invariants. URL: `https://knotinfo.math.indiana.edu`.

[Minsky, 1967] Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., USA.

[Morton, 1983] Morton, H. R. (1983). An irreducible 4-string braid with unknotted closure. *Mathematical Proceedings of the Cambridge Philosophical Society*, 93(2):259–261.

[Orevkov, 2004] Orevkov, S. (2004). Quasipositivity problem for 3-braids. *Turkish Journal of Mathematics*, 28.

[Pudlak, 2013] Pudlak, P. (2013). *Logical Foundations of Mathematics and Computational Complexity*. Springer.

[Rudolph, 1983] Rudolph, L. (1983). Braided surfaces and seifert ribbons for closed braids. *Commentarii Mathematici Helvetici*, 58(1):1–37.

[Xu, 2011] Xu, C. (2011). Word problem for braid group using a representation. URL: `https://chaoxuprime.com/posts/2011-06-23-word-problem-for-braid-group-using-a-representation.html`. (accessed: 02-05-2021).