

Braid Compression

Martin Hock

December 15, 2004

Abstract

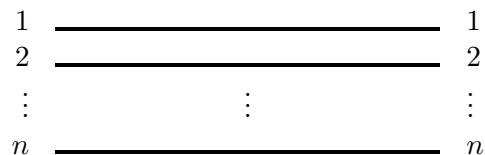
In this paper, we describe the braid group, first mentioned in [2]. We give applications to cryptography mentioned in [1] and [8]. We then describe an apparently new method for shortening braid descriptions which could be useful for transmitting braids over a limited bandwidth channel.

1 Introduction

Braid groups were first described by Artin in 1947 [2]. Artin originally described the mathematical concept of braids in terms of their geometric or topological nature. The braiding of hair or rope may give some intuition as to the nature of the mathematical braid. Alternately, consider an elevator full of n people. As the elevator descends, the people move around in the elevator, along the two dimensional floor. The path traced by these people over time in three-dimensional space represents a braid, with each individual path called a strand. (A strand is a curve in 3-space parameterized by z . Some sources call it a string, but we reserve this word to mean a concatenation of symbols.)

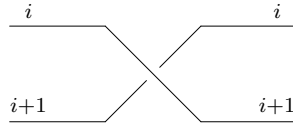
Two braids are considered equivalent if they are *isotopic*, which means that the points of one braid can be mapped to the points of the other in a continuous fashion. In other words, if the braids were physically manifested with the strands made out of a flexible material like thread, we could push the threads of one braid around to match the other braid, leaving the ends fixed. We can also consider the concatenation of braids by first performing the crossings of one braid and then the crossings of the other on the same set of strands. This of course requires that the two braids match at the ends, but if they have the same number of strands, we can use isotopic equivalence to move them so that they match up, and there is a unique way to do this that does not cause strands to cross each other. In this way, we can think of the braid group on n strands, denoted B_n , as being a geometric generalization of the symmetric group S_n . Specifically, when we permute the elements in the case of B_n , we are interested when strands pass in front of or behind each other.

Soon, Artin found that an algebraic representation was superior for analysis, and if you found the above descriptions too airy for rigor yet too hairy for formalization, you will too. He described a set of generators, now referred to as Artin generators, which produce all possible braids up to isotopy. First, we will picture the n strands stretching from left to right, sequenced from top to bottom, as in the following diagram:

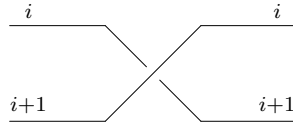


We will consider this the identity element. We will build all possible braids out of a set of generators. There are two kinds of generators, positive and negative.

Definition (Positive Generator). The *positive generator* σ_i signifies that strand i crosses over strand $i + 1$:



Definition (Negative Generator). The *negative generator* σ_i^{-1} signifies that strand i crosses under strand $i + 1$:



The generators do *not* follow the individual strands as they work their way around the braid; the generator σ_i signifies that the i th strand from the top crosses over the $i + 1$ st strand, thus in essence mapping the i th strand to the $i + 1$ st and vice versa.

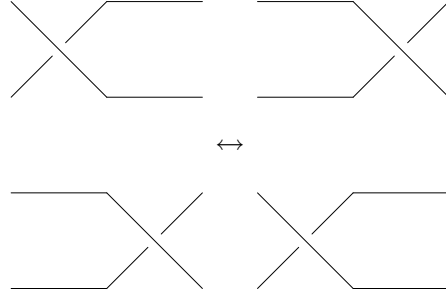
Remark. This presentation is doubly reversed from that of other papers such as [4], where strands are numbered with 1 starting at the bottom, and σ_i signifies that strand $i + 1$ crosses over strand i . There is no difference in the algebra of the two presentations, of course - we could view the braid from the back instead of from the front, through a mirror, and get the opposite interpretation.

We define the group B_n of $n \in \mathbb{N}, n > 1$ strands as having the generators $\sigma_1, \dots, \sigma_{n-1}$ and $\sigma_1^{-1}, \dots, \sigma_{n-1}^{-1}$. (Recall that σ_i involves strands i and $i + 1$, so we do not include σ_n .) Multiplication in this group consists of concatenation, which is obviously associative. We are given inverses by definition, and we have an identity: the empty string. These generators are known as the Artin generators as they come from [2]; later, we will consider an alternate presentation. We will now give two relations for the generators which, when combined with the axioms which hold for all groups, will complete the presentation of B_n .

1.1 Swap relation

A string (as in symbolic concatenation) of these generators signifies a sequence of crosses. Dividing the braid along the horizontal axis into discrete parts, each of which contains exactly one adjacent crossing, the j th position in the string of generators signifies what occurs at the j th part of the braid. The fact that we can in fact divide up the braids into such a sequence was shown in [2], but it is intuitively obvious. If a strand i crosses another strand j , it must cross the strands in between i and j . If more than one crossing occurs at the same horizontal position, simply separate the strands by pulling them apart from each other.

So, for example, $\sigma_1\sigma_3$ appears as the picture on the left, but $\sigma_3\sigma_1$ appears as the picture on the right:



It is fairly obvious that these are isotopic, since we can simply move these separated crossovers past each other. We could do this as long as the crossovers are sufficiently separated – if they are only one strand apart, they will interfere. We call the following the *swap relation*, as it involves the swapping of two generators:

Definition (Swap relation).

$$\sigma_i \sigma_j = \sigma_j \sigma_i \text{ if } |i - j| \geq 2.$$

This is the first of the two relations.

1.1.1 Generalized swap relation

We can use the swap relation to derive the corresponding identities for negative crossings as well as mixed positive and negative crossings:

$$\begin{aligned}
\sigma_i \sigma_j &= \sigma_j \sigma_i && \Rightarrow \\
\sigma_i^{-1} \sigma_i \sigma_j &= \sigma_i^{-1} \sigma_j \sigma_i && \Rightarrow \\
\sigma_j &= \sigma_i^{-1} \sigma_j \sigma_i && \Rightarrow \\
\sigma_j^{-1} \sigma_j &= \sigma_j^{-1} \sigma_i^{-1} \sigma_j \sigma_i && \Rightarrow \\
1 &= \sigma_j^{-1} \sigma_i^{-1} \sigma_j \sigma_i && \Rightarrow \\
\sigma_i^{-1} &= \sigma_j^{-1} \sigma_i^{-1} \sigma_j \sigma_i \sigma_i^{-1} && \Rightarrow \\
\sigma_i^{-1} &= \sigma_j^{-1} \sigma_i^{-1} \sigma_j && \Rightarrow \quad (*) \\
\sigma_i^{-1} \sigma_j^{-1} &= \sigma_j^{-1} \sigma_i^{-1} \sigma_j \sigma_j^{-1} && \Rightarrow \\
\sigma_i^{-1} \sigma_j^{-1} &= \sigma_j^{-1} \sigma_i^{-1} && \Rightarrow
\end{aligned}$$

Starting from line (*), we can get the following:

$$\begin{aligned}
\sigma_i^{-1} &= \sigma_j^{-1} \sigma_i^{-1} \sigma_j && \Rightarrow \\
\sigma_j \sigma_i^{-1} &= \sigma_j \sigma_j^{-1} \sigma_i^{-1} \sigma_j && \Rightarrow \\
\sigma_j \sigma_i^{-1} &= \sigma_i^{-1} \sigma_j && \Rightarrow
\end{aligned}$$

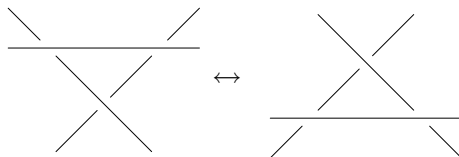
Flipping the sign of σ_j is similar. Thus, we have the more generalized form of the relation:

Theorem (Generalized swap relation).

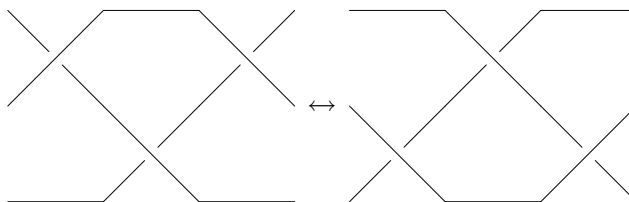
$$\sigma_i^x \sigma_j^y = \sigma_j^y \sigma_i^x \text{ if } |i - j| \geq 2.$$

1.2 Shift relation

There is one other relation we need. Examine the following diagram:



This is known as the third Reidemeister move from knot theory. Assuming the strands are infinitely long, if we shift the horizontal strand down in the first picture, we get the second picture, so the two are isotopic. This corresponds to the following braid situation, where instead of sliding the first strand down (which would compromise equivalence because it would no longer be considered the first strand), we merely push the middle part of the strand down, and push the outer parts up:



In other words, $\sigma_1^{-1} \sigma_2 \sigma_1 = \sigma_2 \sigma_1 \sigma_2^{-1}$. In general, one would state this as $\sigma_i^{-1} \sigma_{i+1} \sigma_i = \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1}$. However, there is a more beautiful way to write this equivalence:

$$\begin{aligned} \sigma_i^{-1} \sigma_{i+1} \sigma_i &= \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} && \Rightarrow \\ \sigma_i \sigma_i^{-1} \sigma_{i+1} \sigma_i &= \sigma_i \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} && \Rightarrow \\ \sigma_{i+1} \sigma_i &= \sigma_i \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} && \Rightarrow \\ \sigma_{i+1} \sigma_i \sigma_{i+1} &= \sigma_i \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} \sigma_{i+1} \end{aligned}$$

which, due to the topological shifting of the strands, we refer to as the *shift relation*:

Definition (Shift relation).

$$\sigma_{i+1} \sigma_i \sigma_{i+1} = \sigma_i \sigma_{i+1} \sigma_i.$$

1.2.1 Generalized shift relation

We can also derive a couple of identities out of this relation:

$$\begin{aligned}
\sigma_{i+1}\sigma_i\sigma_{i+1} &= \sigma_i\sigma_{i+1}\sigma_i && \Rightarrow \\
\sigma_i\sigma_{i+1} &= \sigma_{i+1}^{-1}\sigma_i\sigma_{i+1}\sigma_i && \Rightarrow \\
\sigma_{i+1} &= \sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i\sigma_{i+1}\sigma_i && \Rightarrow \\
1 &= \sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i\sigma_{i+1}\sigma_i
\end{aligned}$$

Similarly, $\sigma_i\sigma_{i+1}\sigma_i\sigma_i^{-1}\sigma_{i+1}^{-1} = \sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i^{-1}\sigma_{i+1}\sigma_i\sigma_{i+1} = \sigma_{i+1}\sigma_i\sigma_{i+1}\sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i^{-1} = 1$.

Using the techniques above, we can give the following general form:

Theorem (Generalized shift relation).

$$\sigma_i^x\sigma_j^y\sigma_i^z = \sigma_j^z\sigma_i^y\sigma_j^x \text{ where } |i-j| = 1, \ x, y, z \in \{1, -1\}, \text{ and } (x-y)(y-z) = 0.$$

Remark. In particular, we do not allow σ_j 's sign to be different from both σ_i and σ_k 's signs, as then if we follow the strands they do not have a total order: in $\sigma_i\sigma_{i+1}^{-1}\sigma_i$ for example, the first strand overlaps the second, the third strand overlaps the first, but the second strand overlaps the third.

These are the only two relations that define the group. We showed that they make sense physically and that they needed only be defined on the positive generators. We did not, however, show that they were completely sufficient to equate all isotopic braids. The proof is outside the scope of this paper, so we refer to [2] for the details. We conclude with the complete Artin presentation of B_n :

Definition (Artin presentation). The group B_n has a presentation with generators $\{\sigma_i; n > i \geq 1\}$ and with defining relations

$$\sigma_i\sigma_j = \sigma_j\sigma_i \text{ if } |i-j| \geq 2$$

and

$$\sigma_{i+1}\sigma_i\sigma_{i+1} = \sigma_i\sigma_{i+1}\sigma_i.$$

1.3 Observations

We now make a few observations about various B_n groups. With only one strand, there is nothing interesting to say; we have no defined generators, so there are no elements, and by definition a group must contain at least one element, so B_1 isn't really a group. If there are exactly two strands, one thing we can see immediately is that neither identity comes into play, but if the only generators we have are σ_1 and σ_1^{-1} , B_2 is isomorphic to \mathbb{Z} , where σ_1 is successor and σ_1^{-1} is predecessor, and $i \in \mathbb{Z} \simeq \sigma_1^i \in B_2$.

The groups become more interesting when they include at least three strands. At this point, the group is no longer abelian. Due to the associative law, σ_i^x and σ_i^y always commute, and due to the first relation, σ_i^x and σ_j^y always commute if $|i-j| \geq 2$, but due to the second relation, σ_i^x and σ_j^y never commute if $|i-j| = 1$. Suppose we assume that they commute; then the following is true:

$$\begin{aligned}
\sigma_i \sigma_{i+1} \sigma_i &= \sigma_{i+1} \sigma_i \sigma_{i+1} && \Rightarrow \\
\sigma_i^{-1} \sigma_i \sigma_{i+1} \sigma_i &= \sigma_i^{-1} \sigma_{i+1} \sigma_i \sigma_{i+1} && \Rightarrow \quad (\text{using commutative property}) \\
\sigma_i^{-1} \sigma_i \sigma_{i+1} \sigma_i &= \sigma_{i+1} \sigma_i^{-1} \sigma_i \sigma_{i+1} && \Rightarrow \\
\sigma_{i+1} \sigma_i &= \sigma_{i+1} \sigma_{i+1} && \Rightarrow \\
\sigma_{i+1}^{-1} \sigma_{i+1} \sigma_i &= \sigma_{i+1}^{-1} \sigma_{i+1} \sigma_{i+1} && \Rightarrow \\
\sigma_i &= \sigma_{i+1} &&
\end{aligned}$$

But if $\sigma_i = \sigma_{i+1}$, then we can use the first identity to commute σ_{i+1} with σ_{i+2} (since σ_i commutes with σ_{i+2}), or σ_i with σ_{i-1} (since σ_{i+1} commutes with σ_{i-1}). Thus, we can inductively apply the above argument to equate all the generators aside from the outermost ones, implying that if any group B_n has two commuting adjacent generators, then $B_n \simeq B_2$.

2 Cryptography

The concept of public key cryptography can be conveyed by the following scenario. Suppose Alice (A) wishes to communicate a message M with Bob (B) over an unsecured line. (Don't confuse communicator B with braid group B_n !) They would like to, based on some public knowledge and some private knowledge, engage in a public conversation which will allow them to share knowledge without allowing eavesdroppers to easily reconstruct that knowledge. It almost sounds like a paradox, but there are in fact two different schemes which allow this to happen.

2.1 Encryption

The first scheme is used for encryption of messages. The following steps ensue:

1. B announces his public key K in a manner which hopefully will not be tampered with (such tampering is known as a man-in-the-middle attack). This key can also be considered to be prior knowledge – it does not change from one message to the next.
2. A uses K to encode a message, $K(M)$ and broadcasts it to B .
3. B uses his private key K' to decode $K(M)$ back into M .

Repeat steps 2 and 3 for each individual message transmitted. B can respond to A if A has her own public-private key pair. An eavesdropper is faced with the task of calculating M given $K(M)$ and K .

The encryption scheme was presented in a paper by Rivest, Shamir, and Adleman [10], which was the first successful public key cryptosystem and is the most widely used one today. It uses the fact that determining primality is easy (which allows for key generation) and exponentiation on the field $\text{GF}(p)$ is easy (which allows for computing f), but factoring the product of two large primes is not known to be easy (which means that, given the public key, it is difficult to find the private key).

2.2 Key exchange

The second scheme is used for key exchange, which has the following steps:

1. B and A are made aware of a public key K (perhaps by convention B announces it).
2. A has a private key a and sends $f(K, a)$ to B .
3. B has a private key b and sends $f(K, b)$ to A . (This step does not depend on knowledge from step 2.)
4. A can compute $f(f(K, a), b)$ and B can compute $f(f(K, b), a)$.

The concept of a public key cryptosystems and key exchange in particular was first publically described by Diffie, Hellman, and Merkle. The first successful implementation was described in [5]. (I say publically described because apparently this scheme as well as the RSA scheme were known to the British intelligence agency GCHQ by 1974; see [6].)

Here, we have the specific need that $f(f(K, a), b) = f(f(K, b), a)$; in other words, f must commute. This value will be used as a common private key for the duration of the conversation. Another scheme will be necessary for the transmission of messages with this key. In the case of the Diffie-Hellman system [5], f is exponentiation on a large cyclic group. In this case (as before), the exponentiation is easy, but the inverse, the discrete log, is believed to be difficult.

Note that with key exchange, A and B learn a common secret, but they can't choose what that secret is, so it's not useful for transmitting a message. However, the secret can be used for the initialization of another system which can do this, such as a symmetric block cipher.

2.3 Anshel-Anshel-Goldfeld scheme

We now exhibit two candidates for key exchange based on braid groups. The first is known as the Anshel-Anshel-Goldfeld scheme and was described in [1] among other places.

1. B and A are made aware of two public keys: $x = (x_1, \dots, x_\ell) \in B_n^\ell$ and $y = (y_1, \dots, y_m) \in B_n^m$.
2. A has a private key $a \in [2\ell]^*$. That is, A 's private key is a string of arbitrary length on an alphabet of 2ℓ characters. A uses a to compute a product of braids based on x . The character $2i$, where $1 \leq i \leq \ell$, is replaced with x_i and the character $2i - 1$ is replaced with x_i^{-1} . We call this product α (the i th character being α_i) and it is kept private. A instead sends B the conjugates $\alpha y_1 \alpha^{-1}, \alpha y_2 \alpha^{-1}, \dots, \alpha y_m \alpha^{-1}$, which we refer to as y' .
3. B has a private key $b \in [2m]^*$. B computes $\beta \in B_n$ by substituting elements of y into b . B then sends the conjugates $\beta x_1 \beta^{-1}, \beta x_2 \beta^{-1}, \dots, \beta x_\ell \beta^{-1}$, which we refer to as x' .
4. A computes α' , which is computed in the same way as α except substituting x' into a . Thus, the i th character $\alpha'_i = \beta \alpha_i \beta^{-1}$. A then computes $\alpha(\alpha')^{-1}$.
5. B computes β' by substituting y' into b ($\beta'_i = \alpha \beta_i \alpha^{-1}$) and then computes $\beta'(\beta)^{-1}$.

It turns out that $\alpha(\alpha')^{-1} = \beta'(\beta)^{-1} = \alpha\beta\alpha^{-1}\beta^{-1}$:

$$\begin{aligned}
\alpha(\alpha')^{-1} &= \alpha(\beta\alpha_1\beta^{-1}\beta\alpha_2\beta^{-1}\dots\beta\alpha_\ell\beta^{-1})^{-1} && \text{(definition of } \alpha' \text{ along with note of its indices)} \\
&= \alpha(\beta\alpha_1\dots\alpha_\ell\beta^{-1})^{-1} && \text{(cancellation of } \beta\beta^{-1}\text{)} \\
&= \alpha(\beta\alpha\beta^{-1})^{-1} && \text{(definition of } \alpha\text{)} \\
&= \alpha\beta\alpha^{-1}\beta^{-1} && \text{(inverse of } \beta\alpha\beta^{-1}\text{)} \\
&= \alpha\beta_1\dots\beta_m\alpha^{-1}\beta^{-1} && \text{(definition of } \beta\text{)} \\
&= \alpha\beta_1\alpha^{-1}\alpha\beta_2\alpha^{-1}\dots\alpha\beta_m\alpha^{-1}\beta^{-1} && \text{(introduction of } \alpha\alpha^{-1}\text{)} \\
&= \beta'_1\beta'_2\dots\beta'_m\beta^{-1} && \text{(definition of } \beta'_i\text{)} \\
&= \beta'\beta^{-1} && \text{(definition of } \beta'\text{)}
\end{aligned}$$

So, $\alpha\beta\alpha^{-1}\beta^{-1}$ is the shared secret.

The eavesdropper must somehow reconstruct $\alpha\beta\alpha^{-1}\beta^{-1}$ from the traffic. Recall that we stated that multiplication consists of concatenation. With the eavesdropper seeing all the x' and y' traffic, if the individual strings simply consist of the form $\alpha y_i \alpha^{-1}$ and $\beta x_i \beta^{-1}$, plucking out α merely consists of looking for a common pattern at the start of each of A 's strings, α^{-1} from looking at the end of A 's strings, and β and β^{-1} can be seen from looking at B 's strings. There may be some additional overlap, but there are only a linear number of possibilities that the eavesdropper must try to shave down his strings to the correct alternatives, and in practice probably very few guesses would be needed. Therefore, we will need some way of “tangling” a concatenated braid into an unidentifiable jumble which still represents the same braid. We can call this the *obfuscation problem*.

While we're at it, if something is being done with the common key $\alpha\beta\alpha^{-1}\beta^{-1}$ other than braid group operations, we would want A and B to be looking at the same representation of the string. But since they go about calculating it in different ways, it is likely to look much different. We would like a way to convert a braid into some kind of canonical form. We can call this the *normalization problem*. Solving normalization might also solve the obfuscation problem at least partially, because the original concatenated representation of the braids sent is probably not the canonical representation.

Next, note that the braids we send might be kind of big. We would like to reduce traffic as much as possible because bandwidth is a finite resource. To do so, we would like to find a small representation of a given braid. First, notice that there will almost always be more than one minimal representation for a given braid, so solving this won't immediately solve the normalization problem. Second, it has been shown that determining whether an arbitrary braid is minimal is co-NP complete, which means that if you can solve it in sub-exponential time, you will become very famous among both mathematicians and computer scientists. However, this does not preclude efficiently finding a small form which is provably within some factor of the optimal. Advancements in the field of probabilistically checkable proofs have helped us show how well we can do; see [12] for more information.

Finally, note that the scheme is not exactly symmetric in that A and B operate on different parts of the public key, one on x and the other on y . However, there is no technical limitation to setting $x = y$, and it would not seem to hurt us - in this case, for example, A 's first part of x' sent will be $\alpha x_1 \alpha^{-1}$, while B 's first part of y' sent will be $\beta x_1 \beta^{-1}$. But it was already public knowledge what α and β were being conjugated with. The fact that they are conjugated with the same thing does not appear to give us any advantages.

2.4 Ko et al scheme

A second key exchange procedure given by Ko et al in [8] more closely resembles that in [5]. Again, a is A 's private key and b is B 's private key. In this case, we want to make a and b commute, so we have $a, b \in B_n$ and the public key $p \in B_{2n+1}$.

1. A and B are made aware of a public braid $p \in B_{2n+1}$.
2. A computes apa^{-1} and sends it to b .
3. B produces b' , which is b shifted down: σ_i is mapped to σ_{i+n+1} . B computes $b'p(b')^{-1}$ and sends it to A .
4. A computes $a(b'p(b')^{-1})a^{-1}$.
5. B computes $b'(apa^{-1})(b')^{-1} = a(b'p(b')^{-1})a^{-1}$ due to the fact that a and b' commute, since the closest strands are 2 apart (σ_n and σ_{n+2}). (This is why we chose B_{2n+1} instead of B_{2n} .)

The shared secret is thus $a(b'p(b')^{-1})a^{-1}$. The asymmetry here is somewhat disappointing, but since one person distinguishes themselves by starting the conversation, A and B can decide their roles and act accordingly.

This scheme, too, has the problem that a and b are broadcast in concatenated form. [11] note that an adversary need not find the original a, b but rather simply a_1, a_2, b_1, b_2 such that $a_1pa_2 = apa^{-1}$ and $b_1pb_2 = bpb^{-1}$. In a search context, introducing additional variables inflates the search space, but in an algebraic context, it can introduce many additional solutions, ultimately simplifying the search. However, no techniques are provided to take advantage of this approach.

3 The Word Problem

Suppose we have two braids $b_1, b_2 \in B_n$ constructed out of a string of Artinian generators. We want some way of comparing them. There can be many ways of representing a given braid using generators – just a single application of the two equivalence relations, or even adding an extra $\sigma_i\sigma_i^{-1}$ for any i , will make the braid different.

All we really need is an algorithm for inverting a braid and an algorithm for seeing if a braid is the identity. The inverse for a braid is easy: take the horizontal mirror image and invert the individual generators. Then, we can perform elimination starting at the middle and easily observe that this will cancel the braid.

One idea for seeing if a braid is the identity is the following. First, make sure that all strands map to their starting places by tracing them. Then, starting with the top strand, “tease apart” each strand one at a time, making sure that it is free of the other strands. Every time a strand crosses on top of the top strand, follow it and make sure that it eventually comes back and passes on top of the first strand again. Similarly, if a strand passes beneath, make sure it comes back and passes beneath again. If this is true, we can remove the top strand as it is free of the other strands. Keep repeating with the top strand until the procedure fails, which means that it’s not the identity, or all strands have been deleted.

This procedure is not too efficient, but it does take time polynomial in the number of strands and length of the braid. A good question to ask is whether there is a more efficient algorithm. The typical solution is to kill two birds with one stone: solve the normalization problem. Once two braids are normalized, seeing if they are equivalent is as simple as a bit comparison.

The procedure used for normalization of braids can be compared to the procedure for normalizing rational numbers: converting them to lowest terms. If you have the rational number $\frac{a}{b}$, and $c = \gcd(a, b)$, then the lowest term representation is $\frac{a/c}{b/c}$. We will also use a form of gcd for finding the normal form of a braid; however, instead of making it the smallest braid, the normal form will typically increase a braid's size.

In [7], Garside showed that every braid $b \in B_n$ can be written in the form $b_1^{-1}b_2$ where $b_1, b_2 \in B_n^+$, i.e., contain no negative generators. This can be thought of as a fractional presentation of B_n , where B_n^+ takes on the role of \mathbb{Z} . To get the concept of a gcd, we will have to generalize the idea of less than or equal, for which we'll use the sign \preceq . We will say that $a \preceq b$ if there exists c such that $ac = b$ (these elements all being in B_n^+). Also, we have a generalized idea of greater than or equal: $a \succeq b$ if there exists c such that $a = cb$. Note that because the side of c has been switched and we are not working with an abelian group, here $a \preceq b$ does not imply $b \succeq a$ and vice versa, and indeed is false unless the elements commute. For example, $\sigma_1 \preceq \sigma_1\sigma_2$ because σ_2 suffices for c , but $\sigma_1\sigma_2 \not\preceq \sigma_1$ because we can't concatenate a positive element to the left of σ_1 to produce $\sigma_1\sigma_2$. (Recall that the only way to delete elements is to introduce negative elements which we don't allow because we are working with positive braids, so introducing two elements won't help us, and introducing one element to the right of σ_1 won't help us either.)

As with the comparison operators, the generalized notion of gcd will take into account the non-abelian nature of the groups. There are both left and right gcds. Suppose we have elements a, b ; the left gcd c is such that $c \preceq a$, $c \preceq b$ (which means that c is a divisor of a and b), and $\forall x$ $x \preceq a$ and $x \preceq b$ implies $x \preceq c$ (which means that c is the greatest divisor of a and b). Notice that we used only \preceq to define the left gcd; the right gcd is similar only using only \succeq and changing the order.

We can also define a least common multiple, or lcm, for both left and right. Suppose we have a, b : the left lcm c is such that $c \succeq a$, $c \succeq b$ (which means that c is a multiple of a and b), and $\forall x$ $x \succeq a$ and $x \succeq b$ implies $x \succeq c$ (which means that c is the least multiple of a and b). This is actually the right lcm, despite it appearing on the same side as the left gcd. The relations \preceq and \succeq along with existence of left and right gcds and lcms (also known as greatest lower bounds and least upper bounds, or infimums and supremums) define what is known as a lattice. Garside showed that B_n^+ is a lattice. He also defined Δ_n , known as the fundamental braid:

$$\Delta_1 = 1, \quad \Delta_{n+1} = \Delta_n \sigma_n \cdots \sigma_1.$$

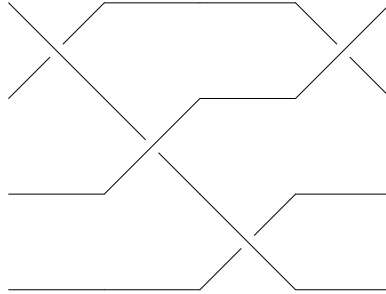
Δ_n is thought of as being fundamental because $\Delta_n \succeq \sigma_i$ and $\sigma_i \preceq \Delta_n$ for all $\sigma_i \in B_n^+$.

We say that a braid $b \in B_n^+$ is simple if there exists $c \in B_n^+$ such that $\Delta_n = bc$. Now consider some braid $a \in B_n$. Garside showed that a can be uniquely written in the following way, which is the normal form we have been seeking:

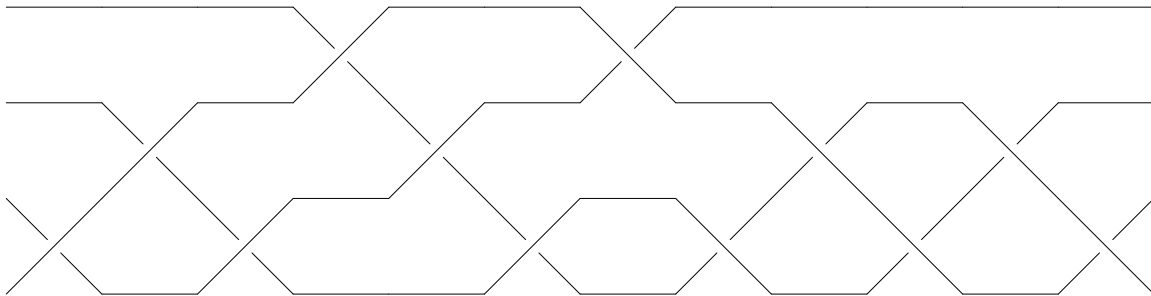
Definition (Greedy normal form). *The greedy normal form of a braid $a \in B_n$ is a braid $\Delta_n^k b = a$ with $k \in \mathbb{Z}$ and $b \in B_n^+$ if we maximize k . Furthermore, b is written as a product $b_1 \cdots b_r$ such that for each b_i , $\Delta_n = b_i c_i$ for some $c_i \in B_n^+$.*

To actually produce this representation, we start with a braid $a \in B_n$. We can then try representing a in the $\Delta_n^k b$ form, for progressively smaller k . k begins as 0; smaller k is only necessary if a contains negative elements. The actual concept of left gcd comes in when we find a normal form for the positive part b . We can find the maximal simple braid that divides b , which is the left gcd of b and δ_n . Suppose $b = b_1 b'$ where $b_1 = \gcd(b, \delta_n)$. Then we can further factor $b' = b_2 b''$ where $b_2 = \gcd(b', \delta_n)$. This provides the decomposition as b_1, \dots, b_r as desired.

The normal form allows us to compare braids easily, but it tends to make the braid increase in size quadratically because of the growth rate of Δ_n^k . Consider the following depiction of $\sigma_1\sigma_2^{-1}\sigma_3\sigma_1^{-1}$ in B_4 :



Its normal form is $\sigma_3^{-1}\sigma_2^{-1}\sigma_3^{-1}\sigma_1^{-1}\sigma_2^{-1}\sigma_3^{-1}\sigma_1\sigma_3\sigma_2\sigma_3\sigma_2\sigma_3$:



Interestingly, even a braid as simple as $\sigma_1\sigma_2^{-1}$ can have a very large normal form. In fact, the normal form for this braid in B_n is (exactly) of size $n^2 - n$ for $n > 3$; even though the optimal representation consists of two crossings for any number of braids, the normal form grows quadratically with the number of braids, a huge loss of efficiency.

One technique which has been studied for a more efficient representation is to increase the number of generators. This was described by Birman et al in [3]. We refer to the original representation as the Artin form and the new representation as the BKL form. Instead of simply providing generators for adjacent crossings, we allow any two strands to cross. By convention, the two cross over all intervening strands. The authors showed that their algorithms are somewhat more computationally efficient than those on the Artin generators, but they also indicate that their normal form is typically much smaller. This is because the fundamental braid they use, δ (which plays the same role as the Artinian Δ), rather than growing quadratically, only grows linearly. However, note that the bit representation of the generators will take up roughly twice as much space. This is because instead of $2n$ generators, which take up $\log(2n) = 1 + \log(n)$ bits apiece, there are $2n(n-1)$ generators, which take up $\log(2n(n-1)) = 1 + \log(n) + \log(n-1) > 2\log(n)$ bits apiece. We now give a formal definition of the BKL presentation:

Definition (BKL presentation). The group B_n has a presentation with generators $\{a_{ts}; n \geq t > s \geq 1\}$ and with defining relations

$$a_{ts}a_{rq} = a_{rq}a_{ts} \text{ if } (t-r)(t-q)(s-r)(s-q) > 0$$

and

$$a_{ts}a_{sr} = a_{tr}a_{ts} = a_{sr}a_{tr} \text{ for all } t, s, r \text{ with } n \geq t > s > r \geq 1.$$

4 Compression

Here we present a different approach than the BKL normal form to make a smaller but still canonical braid. The idea behind the compression technique is simple. We come up with an evaluation function for Artin braids which is used to score braids for how “good” they are, $f : B_n \rightarrow \mathbb{N}$. We want better braids to have a smaller score, so the identity braid is mapped to 0 and we want to score braids which are harder to reduce to a smaller form with bigger scores. Then, we simply try applying a single “move”: we try applying the generalized swap relation to every adjacent pair and the generalized shift relation to every adjacent triple. For each single move, we evaluate the braid and compare its score to the original. We then select the first transformed braid which the evaluation function produces the minimal result for. If that braid is the same as the original, we simply return the original: we could not find an improvement. Otherwise, we apply that transform, perform free reduction, and try to find the best single move again.

Definition (Free reduction). *Free reduction* is the process of removing the maximum number of generator-inverse pairs while preserving equality, e.g. by first eliminating all strings of the form $(\sigma_i \sigma_i^{-1})^*$ and then all strings of the form $(\sigma_i^{-1} \sigma_i)^*$, where x^* is the Kleene (star) closure of x .

Note that if you have a string of the form $\sigma_i \sigma_i^{-1} \sigma_i$, either the first two or last two symbols may be eliminated, but not all three, but that in either case, the result is the same: σ_i .

This procedure will not find an optimal representation in general. If it did, then $P=NP$, as shown by Paterson and Razborov in [9]. Instead, it finds a representation which is locally optimal, that a single move won’t improve. However, we can still use the output of the algorithm as a special normal form, if our algorithms are deterministic and if we start with a normal form, since we will get identical output for identical input, which will be true in the case of equivalent braids as the normal forms will be identical.

We settle on a function which is easy enough to compute, yet analyzes the whole string. We compute the distance from each position in the braid to its closest inverse and return the sum of these distances. (If a given position has no inverses, we add 0; also, if a generator is adjacent to its inverse, we consider it to have distance 0.) A naïve solution might take quadratic time, but it can be done in linear time assuming constant time array access. Refer to Algorithm 1 for pseudocode. Note that if $b_i = \sigma_j^k$, $\text{LOOKUP}(b_i)$ returns the pair (i, j) .

There are two tuples we use to aid in computation of BLOCKEDEVALUATION , $V = (v_1, \dots, v_m)$ with one entry for each index in B , and $S = (s_{1,1}, s_{1,-1}, s_{2,1}, s_{2,-1}, \dots, s_{n,1}, s_{n,-1})$ with one entry for each generator in B_n .

The key to BLOCKEDEVALUATION is in the maintenance of two invariants in the for loop found between lines 5 and 12. Our inductive hypothesis is that at the start of the i th iteration, for all j , v_j holds the index of the closest inverse $< i$ to b_j , or $-(m-i)$ if there is no such index. Furthermore, for all j, k , $s_{j,k}$ holds the maximum index $< i$ such that $b_i = \sigma_j^k$ (i.e., $\text{LOOKUP}(b_i) = (j, k)$), or 0 if there is no such index. The algorithm begins by initializing each v_i to $-(m-i)$ and each $s_{i,j}$ to 0. Note that the initialization of v_i is such that its phantom closest neighbor is $i - (m-i) = m$

Algorithm 1: Blocked evaluation function

Input: A braid $B \in B_n$ consisting of m symbols b_1, \dots, b_m

Output: The sum of the distances from each b_i to its closest inverse which does not appear before another intervening copy of b_i

BLOCKEVALUATION(B)

```

(1)  foreach  $b_i \in B$ 
(2)       $v_i \leftarrow -(m - i)$ 
(3)  foreach  $\sigma_i^j \in B_n$ 
(4)       $s_{i,j} \leftarrow 0$ 
(5)  for  $i = 1$  to  $m$ 
(6)       $(j, k) \leftarrow \text{LOOKUP}(b_i)$ 
(7)       $s_{j,k} \leftarrow i$ 
(8)       $z \leftarrow s_{j,-k}$ 
(9)      if  $z \neq 0$ 
(10)          $v_i = z$ 
(11)         if  $i - z < z - v_z$ 
(12)             $v_z \leftarrow i$ 
(13)   $y \leftarrow 0$ 
(14)  for  $i = 1$  to  $n$ 
(15)       $y \leftarrow y + |i - v_i|$ 
(16)  return  $y$ 

```

positions away from it, which is farther than any possible neighbor and normalizes all distances to the same value. We can see that inductively, the base case is complete: we have not seen anything so far.

Now consider iteration i of the loop, which will set up the invariants for $i + 1$. In lines 6–7, we immediately set $s_{j,k}$ to i , where $b_i = \sigma_j^k$. This follows the definition of $s_{j,k}$: if $b_i = \sigma_j^k$, then i is the rightmost index $< i + 1$ such that $b_i = \sigma_j^k$. Now we attempt to look up the index of the closest inverse of σ_j^k seen so far, which is s_j^{-k} . Let $z = s_j^{-k}$. If $z = 0$, by the inductive hypothesis, we haven't seen σ_j^{-k} yet, so we leave s_j^k as is. Otherwise, we have seen it. Immediately, we know that the closest inverse of σ_j^k seen so far is b_z , so we assign $v_i = z$. Now we need to see if b_i is the closest inverse to b_z . We can calculate this by comparing differences. We don't need absolute value signs because $z < i$ and if $v_z > z$, then $i > v_z$ and is thus even farther away than v_z . If b_i is closest, we reassign v_z to be i . With that, we have maintained the invariants and thus the inductive hypothesis is proved. To calculate the return value in lines 14–15, we simply add up for each i the distance between v_i and i , since v_i is the index of the closest inverse of b_i .

Or is it? Note that in the i th iteration, if we have two copies of σ_j^{-k} , we will only check the closest one $< i$ to see if b_i is closer and update it if necessary. So, after the algorithm completes, for all i , v_i only contains the closest inverse of $b_i = \sigma_j^k$ which is not separated by another copy of σ_j^k . In fact, this is a desirable trait, because we are only concerned with the distance between a symbol and its closest accessible inverse. A symbol will never be able to access the inverse through the other copy of itself. For this reason, we call this the “blocked version” of the evaluation function. Note that it only takes loops which perform a constant number of operations and which run for either $O(n)$ or $O(m)$ iterations, for a total runtime of $O(n + m)$.

If we do want the solution to do what we says it does (the “unblocked version” of the evaluation function), we also need to keep track of a symbol's closest copy that we have seen so far, and every

time we see an inverse, we must trace this list and potentially update every copy. We do this in Algorithm 2. c_i here stores the index of the closest copy of b_i 's symbol which is $< i$, or 0 if no such copy is known.

However, a given c_i position may only be updated in this manner once, and since there are only $O(m)$ c_i s, this does not increase the overall time complexity of the algorithm beyond $O(m + n)$. The position need only update once because an inverse which is even farther to the right of the first match found will be farther away, so we can stop tracing the c_i s as soon as a closer match is found (lines 20–21). To see why this variable time spent does not result in an asymptotic time increase, visualize each position holding a single counter which is only consumed if it is updated via a trace (lines 15–21 of the algorithm). Some counters may be consumed and others may not, but there are only $O(m)$ counters total. While it is true that the trace will run across one redundant element (the first one which is too far away to update), we can add this constant cost as part of the total loop cost (the loop from lines 6–21).

Algorithm 2: Unblocked evaluation function

Input: A braid $B \in B_n$ consisting of m symbols b_1, \dots, b_m

Output: The sum of the distances from each b_i to its closest inverse

```

UNBLOCKEDEVALUATION( $B$ )
(1)  foreach  $b_i \in B$ 
(2)     $v_i \leftarrow -(m - i)$ 
(3)     $c_i \leftarrow 0$ 
(4)  foreach  $\sigma_i^j \in B_n$ 
(5)     $s_{i,j} \leftarrow 0$ 
(6)  for  $i = 1$  to  $m$ 
(7)     $(j, k) \leftarrow \text{LOOKUP}(b_i)$ 
(8)     $s_{j,k} \leftarrow i$ 
(9)     $z \leftarrow s_{j,-k}$ 
(10)   if  $z \neq 0$ 
(11)      $v_i = z$ 
(12)     if  $i - z < z - v_z$ 
(13)        $v_z \leftarrow i$ 
(14)        $z \leftarrow c_z$ 
(15)     while  $z \neq 0$ 
(16)        $v_z \leftarrow i$ 
(17)       if  $i - z < z - v_z$ 
(18)          $v_z \leftarrow i$ 
(19)          $z \leftarrow c_z$ 
(20)     else
(21)        $z \leftarrow 0$ 
(22)   $y \leftarrow 0$ 
(23)  for  $i = 1$  to  $n$ 
(24)     $y \leftarrow y + |i - v_i|$ 
(25)  return  $y$ 

```

Rather than run the entirety of one of the evaluation functions every time we perform a local update, it is tempting to do only a minimal update of the tuples they maintain. To do this, we will need to store more information: backward as well as forward links. This involves maintaining

a sorted list, which in practice will probably be more trouble than it's worth, so we will not pursue this idea further.

4.1 Performance

The algorithm appears to do well. The normal form of the earlier depicted example of $\sigma_1\sigma_2^{-1}\sigma_3\sigma_1^{-1}$ is reduced to its original form. The quadratic blowup example of $\sigma_1\sigma_2^{-1}$ is reduced to 4 symbols, or the optimal 2 if the optimization function is suitably tweaked (without compromising the other results). Some experimentation was done with attempting to make the evaluation function nonlinear in the distance of the closest accessible inverse, with negative results. The following are statistics regarding the length of random elements of the form $a^{-1}b$ where $a, b \in B_6^+$. The BKL values have been doubled to represent the approximate number of bits taken up.

Initial	Normal Artin	Blocked	Unblocked	Normal BKL
1092	732	724	730	688
1014	730	716	726	912
984	614	596	600	432
531	381	357	375	542
363	377	339	337	154

As expected, the blocked version usually outperforms the unblocked version. However, the compression algorithm typically does not do as well as normalized BKL. Therefore, it seems worth implementing our algorithm for BKL. Essentially the same evaluation procedure can be used and the local search procedure needs only a few modifications. However, with BKL's quadratically larger number of generators, there will be far fewer collisions. Still, BKL's second defining relation allows the transformation of one generator into another, which could allow more inverses to be created. I will probably follow up on this in the coming weeks for curiosity's sake.

We should mention one last point which partially undermines the value of this work. Recall that the Artin normal form of a braid will be of the form $\Delta_n^k b$ where $b \in B_n^+$. This means that if we know n , we can encode the braid simply by encoding (k, b) where b is a string on $[1, n - 1]$. This will only take up $\log(k) + |b|\log(n)$ bits, which is significantly shorter than the lengths given above. However, for the compression procedure to work, it will have to destroy the regular form of Δ_n^k , so we can't encode the braid as efficiently. Still, the modified form might still be helpful for making other operations (which can't take advantage of the form of Δ_n^k) more efficient.

5 Acknowledgements

David Koop and Sarah Knoop (no relation) both provided invaluable editorial assistance. In particular, Dave suggested braid groups as a topic of study and provided several significant insights into their workings. I would also like to thank Nigel Boston, who provided some initial guidance, and without whose applied algebra class this paper would not be written.

References

- [1] Iris Anshel, Michael Anshel, Benji Fisher, Dorian Goldfeld. New Key Agreement Protocols in Braid Group Cryptography. *CT-RSA 2001*, 13–27.
- [2] E. Artin. Theory of Braids. *The Annals of Mathematics*, 2nd Ser., Vol. 48, No. 1 (Jan., 1947), 101–126.

- [3] J. Birman, K.H. Ko, and S.J. Lee. A New Approach to the Word Problem in the Braid Groups. *Advances in Math.*, 139-2 (1998), 322–353.
- [4] Patrick Dehornoy. Braid-based cryptography. *Contemporary Mathematics*, 360 (2004) 5–33.
- [5] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, Vol. 22, No. 6 (Nov., 1976), 644–654.
- [6] James H. Ellis. The History of Non-Secret Encryption. Manuscript, currently available at <http://www.cesg.gov.uk/site/publications/media/ellis.pdf>.
- [7] G. A. Garside. The Braid Group and Other Groups. *Quart. J. Math. Oxford* 20-78 (1969), 235–254.
- [8] K.H. Ko, S.J. Lee, J.H. Cheon, J.W. Han, J.S.Kang, and C. Park. New Public-Key Cryptosystem Using Braid Groups. *Crypto 2000*, Springer Lect. Notes in Comput. Sci., 1880 (2000), 166–184.
- [9] M. S. Paterson and A. A. Razborov. The Set of Minimal Braids Is Co-NP-complete. *Journal of Algorithms*, Vol. 12 (1991), 393–408.
- [10] Ronald Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, Vol. 21, No. 2, 1978, 120–126.
- [11] Vladimir Shpilrain and Alexander Ushakov. The Conjugacy Search Problem in Public Key Cryptography: Unnecessary and Insufficient. Preprint, currently available at <http://www.sci.ccny.cuny.edu/~shpil/csp.pdf>.
- [12] Luca Trevisan. Inapproximability of Combinatorial Optimization Problems. Preprint, currently available at <http://www.cs.berkeley.edu/~luca/pubs/inapprox.pdf>.