Solutions to the conjugacy search and decision problems in the braid group using finite conjugacy class invariants

Sané Erasmus

Dissertation presented for the degree

Master of Science

in Applied Mathematics

University of Cape Town

February 2022

Abstract

For two braids, $A, B \in B_n$, the conjugacy decision problem asks whether another braid $X \in B_n$ exists such that $X^{-1} A X = B$. If we know $A, B \in B_n$ are indeed conjugate, the conjugacy search problem asks us to find a braid $Y \in B_n$ such that $Y^{-1} A Y = B$. In this dissertation we investigate a number of solutions to the conjugacy search problem and conjugacy decision problem in the braid group, all of which use finite invariant subsets of the conjugacy class. In particular, we study the summit set, the super summit set, the improved super summit set algorithm which utilises minimal simple elements, the ultra summit set, improvements to the ultra summit set solution using graph theory, and lastly the set of sliding circuits. As part of this investigation, we also study normal forms of braids, partial orders on the braid group, and the Garside group which generalises the braid group.

Under the supervision of

Dr Claire Blackman
Maastricht Science Programme
Maastricht University

&

A/Prof. David Erwin
Laboratory for Discrete Mathematics and Theoretical Computer Science
Department of Mathematics and Applied Mathematics
University of Cape Town

Acknowledgements

This work is based on the research supported in part by the National Research Foundation of South Africa.

I would like to thank Irene Pasternak and Richard Smith from the Bodleian Libraries at the University of Oxford for their kind assistance in accessing Frank Garside's thesis remotely.

Contents

Chapter 1

Mathematical background

This dissertation focuses on the evolution of solutions to the conjugacy search and decision problems for arbitrary braids. In particular, I focus on the use of finite invariant subsets of the conjugacy class to solve the conjugacy problem. Although it was the source of my initial interest in this topic, we will not look at the specific variations on this problem introduced for cryptographic purposes. My goal is to present, in an understandable way, what I have learned by studying the key papers about these invariants.

A note on notation: I encountered many variations in notation, word choice, and (equivalent) definitions across the literature. For the sake of clarity and consistency, I unify these notations and choices, but for the sake of historical interest, I mention what appears in different papers, as it is interesting to see how these trends evolve.

In this chapter I introduce mathematical braids, conjugation, and the braid conjugacy search and decision problems. In Chapter 2, we look at two normal forms for braids, as these normal forms are essential to the first two solutions to the conjugacy search and decision problems. After that, we dive into the solutions and add further relevant knowledge as needed. Chapter 3 deals with the summit set and the super summit set (for which those normal forms are necessary). In Chapter 4, we look at a partial order on braids which allows the definition of least common multiples and greatest common divisors for braids, and introduce a more general structure, Garside groups, as later solutions are given in the language of Garside groups. We also look at an improvement to the super summit set solution which uses greatest common divisors. Chapter 5 is concerned with the ultra summit set and an improvement to this solution, and Chapter 6 is about solving the conjugacy problem using the set of sliding circuits. Chapter 7 concludes the dissertation. Examples of a variety of computations are given in Appendix A, and Appendix B contains a small proof and a brief motivation for corrections I made to two of the papers we discuss.

1.1 Mathematical braids

Braids are topological objects, which can be fully represented algebraically. Although they are topological and have been studied from a topological viewpoint for many years, they

were originally introduced using the term "geometric", and we retain that legacy. We begin by describing geometric braids and introducing the geometric braid group, before defining the algebraic braid group. After this, we use algebraic expressions to represent braids, with geometric illustrations where appropriate.

The notion of a mathematical braid was first defined by Emil Artin in his 1925 paper, *Theorie der Zöpfe* [Art25], and further explored in *Theory of Braids* [Art47b] and *Braids and Permutations* [Art47a]. Wei-Liang Chow [Cho48] and H. Frederic Bohnenblust [Boh47] also contributed much to the initial groundwork for braids. The description I provide below has not been derived from one particular source – rather, these are the facts I have absorbed and the conventions I have adopted as I have been researching braids over the past four years, presented in what I hope to be the clearest way. The two sources I used most when I started investigating braids during my undergraduate studies are Jean-Luc Thiffeault's lecture notes on braids and dynamics [Thi08] and Ester Dalvit's PhD thesis on popularising braid theory [Dal11].[1]

1.1.1 Geometric braids

It is easy to illustrate what a braid is via examples, but for the sake of rigour, we need a confusion-free description. Thus, consider the below example, and the description that follows it.

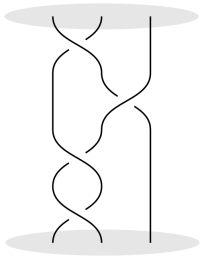


Figure 1.1

For positive integers n, we define a **braid** on n strands (sometimes called an n-braid) as a collection of n strands which adhere to the following conditions. The strands are fixed between two parallel discs[2], which are the unit distance apart. In particular, there are n fixed points on each disc, to which these strands are attached. Each fixed point has one and only one strand attached to it. The n fixed points on each disc are collinear, and spread apart such that the points divide the line segment they span into $n-1$ equal parts. Furthermore, if one considers the two line segments spanned by each set of fixed points on the two separate discs, these line segments will be in the same plane; the plane containing these line segments

[1] Although it is relatively advanced, I spent a decent amount of time with [Thi08], as it concerns applications of braids to dynamical problems, which was my initial interest in braids, and it is quite thorough. [Dal11] introduces braids in a very straight-forward way, which I appreciated, as the concern of her thesis is using braids as a tool to excite interest in mathematics.

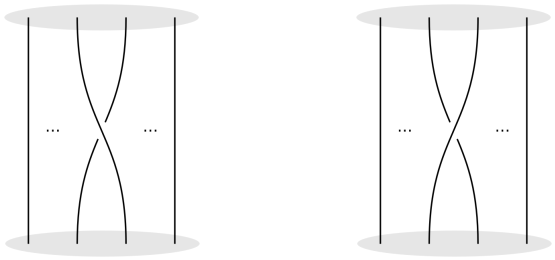
[2] Some authors use planes, some use bars, but I find the discs the best way to capture the 3-dimensionality of braids without making them frustrating to draw.

is perpendicular to the discs and parallel to the page when we draw a braid on paper.

Most importantly, the strands cross over and under one another, and the whole braid exists in 3 dimensions. Since the strands cross over and under each other, the endpoint of a strand need not be directly below its starting point. Furthermore, the strands can never intersect, and never "turn back" – in other words, any plane parallel to and between the discs will intersect each strand exactly once.

A note on convention: We can orient our drawings of braids either vertically or horizontally (diagonally is, of course, possible as well, but I have yet to encounter it in the literature). We must also decide which set of fixed points is the start and which is the end of the braid, as we need to have an established order in which crossings occur (top to bottom or bottom to top in the vertical case, left to right or right to left in the horizontal case) to enable us to describe braids consistently. We will use the convention that our braids are oriented vertically, and the start is at the top of the braid. Thus, when considering a sequence of crossings, we will start our description at the top and work downwards.

When a strand crosses over its right-hand (as viewed from our perspective) neighbour, we call it a **positive crossing**, and draw it as indicated in figure 1.2a. When a strand goes behind its right-hand neighbour, we call it a **negative crossing** and draw it as in figure 1.2b. Some authors use the opposite convention – that left *under* right is a positive crossing. These conventions do not affect the crux of the mathematics, as long as we are consistent in our chosen convention.



(a) (b)

Figure 1.2

We can, of course, have a braid with no crossings, which we call the **identity braid**. We will denote the identity braid by 1. Here all the strands go straight down (see figure 1.3).

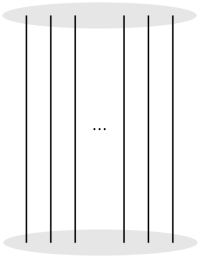


Figure 1.3

We consider two n-braids to be equivalent if they are **isotopic**. This means that we can deform the one braid into the other by continuously moving the strands, but never cutting a strand, letting two strands touch, or moving a start point or end point. We do not care about the length of the strands during this deformation: they can stretch and retract as much as necessary. We do, however, care that the discs at the start and end of the braid are always the unit distance apart.

There are infinitely many ways to deform a particular braid into an equivalent braid, so when we consider a braid, we are actually considering an equivalence class. Unless there is a reason to distinguish, we will use the word "braid" to mean "the equivalence class of a braid".

We can **concatenate** two n-braids to produce a third n-braid. Given braids A and B we obtain the concatenated braid $A \cdot B$ (or AB), by first placing B beneath A, then removing the discs at the end of A and start of B, connecting the strands at the meeting points, and finally squeezing this new braid so that its discs are again the unit distance apart. Figure 1.4 illustrates concatenation – I have colour-coded the discs to make the process more obvious; the green dashes indicate where the two green discs were deleted.

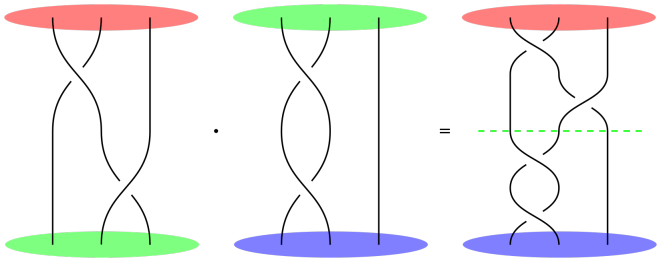


Figure 1.4

We will omit the $\cdot$ when concatenating braids, unless we use it to convey specific information, like a particular factorisation.

Since we require that the resulting braid is also of unit length, the set of all possible braids on n strands is closed under braid concatenation. Furthermore, concatenating braids is an associative operation. We can easily see that the identity braid is the identity element for the concatenation operation. We will shortly see how inverse braids are constructed, allowing us to conclude that the set of all n-braids forms a group under braid concatenation.

We've already seen a positive and negative crossing. Notice that if we concatenate a positive and a negative crossing which occur on the same strands, as illustrated in figure 1.5, we obtain a braid isotopic to the identity braid. Thus, as long as they occur on the same strands, a negative crossing is the inverse of a positive crossing, and vice versa.

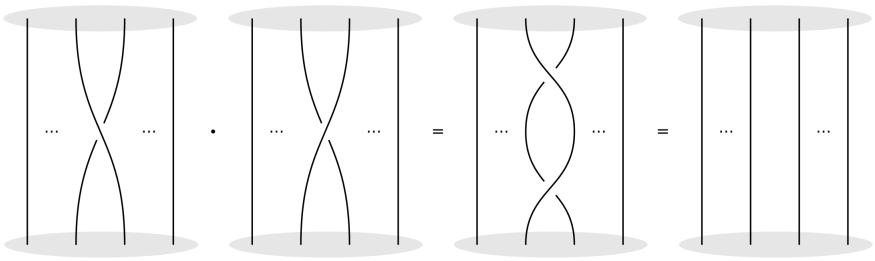
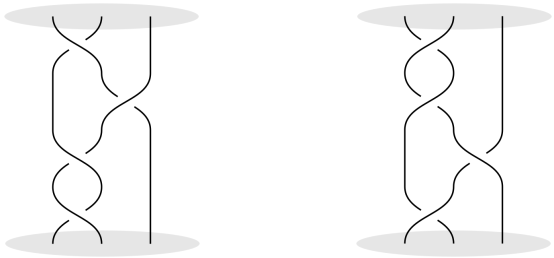


Figure 1.5

If we have a braid with any number of positive and negative crossings, we can construct its inverse by starting at the bottom of the braid and working upwards. We turn each crossing we encounter, as we work upwards, into its inverse counterpart, and write these counterparts down from top to bottom to form our resulting braid. It is important that we write the last crossing's inverse first, and then the second-last crossing's inverse, and so forth, because braid concatenation is not blessed with commutativity. A quicker way to find the inverse of a geometric braid, is simply to reflect it across the horizontal axis, as this provides the same outcome (and is highly convenient when using illustration software). Below we have a braid (figure 1.6a) and its inverse (figure 1.6b). I leave it as an exercise to the reader to check that concatenating these braids forms the identity braid on 4 strands.



(a) (b)

Figure 1.6

Finally, we can conclude that the set of all possible braids on n strands forms a group under braid concatenation. We call this the **geometric braid group**, and denote it by B_n. As mentioned when first describing braid concatenation, we can concatenate an n-braid only to another n-braid – we cannot mix the number of strands when concatenating. So, just to be clear: B_n and B_m are two different braid groups when $n \neq m$. Since there are an infinite number of positive integers, there are an infinite number of braid groups.

There are a number of ways to describe braids without drawing them as we have done so far. In particular, Dale Rolfsen lists six distinct descriptions in [Rol03]. We can describe an n-braid as n specified curves in $\mathbb{R}^3$, or as the time history for $0 \leq t \leq 1$ of a collection of n trajectories (i.e., $\beta(t) = (\beta_1(t), \ldots, \beta_n(t))$ where $\beta_i(t) \in \mathbb{C}$). In both these cases, the curves or trajectories must adhere to the requirements we have laid out. We can also describe B_n as the mapping class group of D_n, the disk D with n punctures, or as the fundamental group of the configuration space of orbits of the obvious action of Σ_n upon $\mathbb{C}^n \backslash \delta$, where

$\delta = \{(z_1, \ldots, z_n); z_i = z_j \text{ for some } i < j\} \subset \mathbb{C}^n$, the so-called big diagonal in complex n-space, and Σ_n is the group of permutations on n elements. Furthermore, we can describe B_n as a group of automorphisms: in particular, a mapping class $[h]$, where $h : D_n \to D_n$, gives rise to an automorphism $h_* : F_n \to F_n$ of free groups; this defines an injective homomorphism $B_n \to Aut(F_n)$. Lastly, we can describe B_n purely algebraically, using generators and relations upon these generators. This is the description I introduce in the next subsection, and the one I use throughout this dissertation.

1.1.2 Algebraic braids

We now investigate how to work with braids without their geometric representations. We introduce algebraic braids, and give the presentation of the algebraic braid group. This algebraic presentation is proposed by Emil Artin in his first two papers on braids, and he proves via geometric considerations that it is a valid presentation. However, H. Frederic Bohnenblust proves from a purely algebraic perspective in [Boh47] that the algebraic braid group indeed shares the same fundamental features as the geometric braid group, and hence that the algebraic braid group is isomorphic to the geometric braid group.

A bit of background on group presentations: a **presentation** of a group G consists of a set of generators S and a set of relations R among these generators[3], such that any element in the group can be written as a product of powers of any number of these generators. The relations tell us how to determine equivalence between elements which may have multiple ways of being written in the generators.

Consider the braid introduced in figure 1.1 (and used again in figure 1.6a). We can see that this braid on 3 strands, which we now call X, consists of 4 crossings. We can consider each crossing (with the remaining strand fixed) as its own braid, so that this braid is obtained by concatenating 4 braids, as illustrated in figure 1.7.

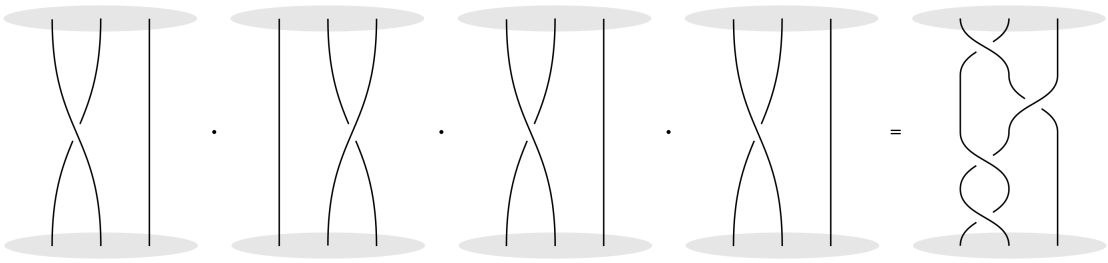


Figure 1.7

Let's name the 4 braids which we concatenated to obtain X, braids A, B, C, and D, so that $ABCD = X$. In braid A, we see that the first strand crosses over the second strand while the third strand remains fixed. In braid B, the second strand crosses under the third strand while the first strand remains fixed. In braids C and D we have the same situation as for braid A: strand 1 goes over strand 2, while strand 3 remains fixed. Clearly, we actually have

[3]Formally, G has the presentation given by S and R if it is isomorphic to the quotient of a free group on S by the normal subgroup generated by the relations R.

$ABAA = X$ since $C = D = A$.

Braids such as A and B, where only one crossing occurs, can be viewed as braid generators. However, naming them A and B is not very helpful; instead we name them in such a way that it is clear from the name which strand crosses over/under its right-hand neighbour. In order to know which strands we're discussing, we number the strands from 1 to n from left to right. Thus, strand $k+1$ is to the right of strand k.

In general, we write σ_i, for $1 \leq i < n$, to indicate the braid where strand i crosses strand $i+1$ positively, and the remaining $n-2$ strands stay fixed. Thus, $A = \sigma_1 \in B_3$, since strand 1 goes over strand 2, and strand 3 is fixed. On the other hand, if strand i crosses strand $i+1$ negatively, while the remaining $n-2$ strands are fixed, we write this as σ_i^{-1} for $1 \leq i < n$. So, $B = \sigma_2^{-1} \in B_3$, since strand 2 goes under strand 3, and strand 1 stays fixed.

If we write a braid using this notation, it is important to indicate the value of n somewhere else, as this information is not captured in σ_i. For example, $\sigma_1 \in B_3$ and $\sigma_1 \in B_4$ are completely different braids. Also notice that we cannot have σ_n in B_n, as that would require the existence of a strand $n+1$. Hence, σ_i and σ_i^{-1} are only defined for $1 \leq i < n$.

With this knowledge, we can rewrite X as $X = \sigma_1\sigma_2^{-1}\sigma_1\sigma_1 \in B_3$, and we have all the necessary information about X, without having to look at its geometric representation. Notice that the numbering of the strands is reset after each crossing – in the second crossing of X, we say strand 2 crosses under strand 3, even though what we now call strand 2 used to be strand 1 in the previous crossing.

Any braid can be written using σ_i and σ_i^{-1} where $1 \leq i < n$, and so we call the elements in the set $\{\sigma_1, \sigma_2, \ldots, \sigma_{n-1}\}$ the **generators of the algebraic braid group on n strands**. When referring to some σ_i, we call it a "positive generator" or just a "generator". We call σ_i^{-1} an "inverse generator" or a "negative generator".

As with any generators of a group, we use exponents to indicate repeated occurrences of generators, so we could write $X = \sigma_1\sigma_2^{-1}\sigma_1^2 \in B_3$. Similarly, we could rewrite $\sigma_k^{-1}\sigma_k^{-1} = \sigma_k^{-2}$ in some B_n. We can also use exponents on whole braids, e.g., $X^2 = XX = \sigma_1\sigma_2^{-1}\sigma_1^2 \cdot \sigma_1\sigma_2^{-1}\sigma_1^2 = \sigma_1\sigma_2^{-1}\sigma_1^3\sigma_2^{-1}\sigma_1^2$. When we write a braid as an ordered sequence in the generators and their inverses, we call it a **braid word**. Some authors let $X^Y = Y^{-1}XY$ where X and Y are braids, but we do not use this notation. Any symbol written as an exponent will always refer to an integer.

There are two relations we need to impose in order to have a complete presentation of the algebraic braid group. As mentioned earlier, these relations are useful if we want to rewrite one braid word as another equivalent braid word. The **relations for the algebraic braid group** are

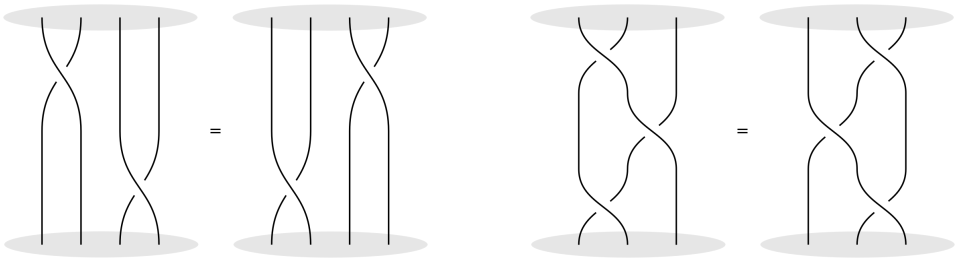
$$\sigma_i\sigma_j = \sigma_j\sigma_i \text{ if } |i-j| \geq 2,$$

and

$$\sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j \text{ if } |i-j| = 1.$$

The first relation tells us that when the strands are far enough apart, the generators commute. The second relation gives an almost-commutativity for adjacent generators of a particular

form. It is easy to check geometrically that these relations are indeed true. Figure 1.8a illustrates the first relation, and figure 1.8b illustrates the second.



(a) (b)

Figure 1.8

Since the algebraic braid group on n strands is isomorphic to the geometric braid group on n strands, we will denote it by B_n as well. Thus, B_n has presentation given by the generators $S = \{\sigma_1, \sigma_2, \ldots, \sigma_{n-1}\}$ and relations $R = \{\sigma_i\sigma_j = \sigma_j\sigma_i \text{ for } |i-j| \geq 2, \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j \text{ if } |i-j| = 1\}$.

As before, we indicate the identity element by 1 for the algebraic braid group. It will be clear from context when 1 refers to the braid and when 1 refers to the integer.

The braid group B_1 on 1 strand contains only one element – the single strand which goes straight down, as there are no other strands to cross. Thus, B_1 is called the trivial braid group. B_2 has only one generator, σ_1, and so is an infinite cyclic group, isomorphic to the group of integers under addition.

1.2 The braid conjugacy problem(s)

In this section I give a brief overview of conjugation and introduce the braid conjugacy search and decision problems. These definitions are given in every paper that discusses these conjugacy problems.

Let $a, b, x \in G$, where G is a non-commutative group. We can **conjugate** a by x, which means to determine the element

$$x^{-1} \cdot a \cdot x$$

where $\cdot$ indicates the group operation. This element will also be in G.

We say that a is **conjugate to** b (or that a and b are conjugate) if there exists an element $x \in G$ such that

$$x^{-1} \cdot a \cdot x = b.$$

In this case, we call x a **conjugator** from a to b or a **conjugating element** between a and b. We may also say that x conjugates a to b. Notice that if a is conjugate to b, then b is also conjugate to a, since we can write

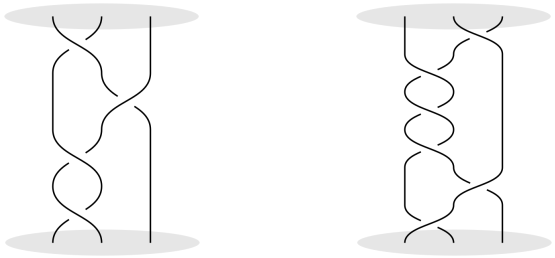
$$a = x \cdot b \cdot x^{-1} = (x^{-1})^{-1} \cdot b \cdot (x^{-1}).$$

We define the **conjugacy class** of a as the set of all elements which are conjugate to a, i.e.,

$$C(a) = \{y^{-1} \cdot a \cdot y \mid y \in G\}.$$

Conjugacy is an equivalence relation and all elements in the conjugacy class of a will have the same conjugacy class, all of which are equal to the set $C(a)$. Thus, two elements are conjugate in G if and only if they have the same conjugacy class.

We can conjugate two n-braids to form a third n-braid. Below is an example of the braid $X = \sigma_1\sigma_2^{-1}\sigma_1^2$ and the braid $Y = \sigma_2\sigma_1^3\sigma_2^{-1}\sigma_1^{-1}$. Do you think they are conjugate? If so, what is the conjugating element?



(a) (b)

Figure 1.9

These are indeed conjugate, with conjugating element $\sigma_2^{-1}\sigma_1^{-1}$. This is a relatively simple example, but even so it is not immediately obvious that these two braids are conjugate.

We can now define the two computational problems of conjugacy in the braid group:

The braid conjugacy decision problem: Given two braids $A, B \in B_n$, determine whether A is conjugate to B. In other words, determine whether there exists some $X \in B_n$ such that $X^{-1}\,A\,X = B$.

The braid conjugacy search problem: Given two conjugate braids $A, B \in B_n$, find $Y \in B_n$ such that $Y^{-1}\,A\,Y = B$.

I refer to these two problems collectively as the **braid conjugacy problems**, or just the conjugacy problems, as they are solved together in the solutions we investigate. Artin states these problems in his initial papers, but the first solution was only proposed 20 years later. There is no general closed-form solution to these problems for arbitrary braids, and the algorithmic solutions can be slow if one is unlucky, since they are not in polynomial time. However, in most cases the algorithms can be performed quickly despite their non-polynomial nature.

The conjugacy decision problem can be restated as a problem in the equivalence of closed braids. If we wrap a braid once around a cylinder, connect the fixed points of the strands where they meet, and delete the discs, we obtain a closed braid. In this case, we would call our usual braids "open braids". Two closed braids are equivalent if we can deform the one into the other without cutting any strands, and without any of the strands crossing the axis of the cylinder. Artin shows in [Art25] that two closed braids are equivalent if and only if their corresponding open braids are conjugate.

Chapter 2

Representing braids uniquely

Two equivalent braids may be drawn and written very differently from one another. We need a unique form for braids, so that equivalent braids can be drawn and written identically. Such a form is often called a **normal form**. The question of whether two algebraic words are equivalent is called the **word problem**.

Artin proposes a normal form for braids in [Art47b]. With this we can determine whether two given braids are equal by putting both in normal form and comparing them; if they are indeed equal, their normal forms are identical. Artin calls the process of turning a braid into this normal form the **combing operation**, and quips in the final paragraph of his paper that "...the writer is convinced that any attempt to carry this out on a living person would only lead to violent protests and discrimination against mathematics. He would therefore discourage such an experiment." Indeed, it is quite convoluted! Figure 2.1 contains an illustration from [Art47b] of what the normal form of a certain braid looks like (Artin does not give an expression for this particular braid in terms of the usual generators, although he does provide enough information for the reader to work out what the original braid is).

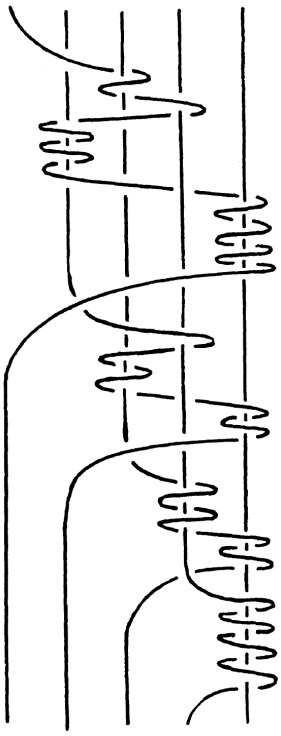


Figure 2.1

In [Gar69], Frank A. Garside provides the first solution to the conjugacy problem, and proposes his own normal form (called the "standard form" in his paper) which plays a fundamental role in his solution. William P. Thurston improves upon Garside's standard form in his chapter on the braid group in [Eps92] and proposes the right-greedy canonical form and similar left-greedy canonical form as his normal forms. El-Sayed A. El-Rifai and Hugh R. Morton then use the left-greedy canonical form (simply called the "left-canonical form" in their paper) to improve upon Garside's solution to the conjugacy problem in [ERM94]. The left- and right-canonical forms appear widely in the literature surrounding the conjugacy problem, and they are often referred to as the "left- and right-normal forms".

As these two normal forms (the standard form and the left-canonical form) are so important to understanding the first two solutions to the conjugacy problem, we begin with a discussion of the two relevant papers introducing them. This will also introduce us to much of the prerequisite knowledge for the respective solutions. This latter reason is why I introduce the left-canonical form in the mathematical language that El-Rifai and Morton laid out, rather than in the language of Thurston. We do, however, revisit Thurston's chapter later, as he proposes a partial order on braids which allows us to find greatest common divisors and least common multiples of braids. El-Rifai and Morton propose a different partial order, which does not allow for the calculation of greatest common divisors or least common multiples of braids.

2.1 Garside's standard form

The first paper we consider is *The braid group and other groups* by Frank A. Garside [Gar69]. This paper is mainly based upon his 1965 D.Phil. thesis, *The theory of knots and associated problems* [Gar65], with some improvements[1].

A historical note: Garside denotes the braid group on $n+1$ strands by B_{n+1} with generators $a_1, a_2, \ldots, a_n$. For consistency with the rest of the literature, I replace these with B_n and $\sigma_1, \sigma_2 \ldots, \sigma_{n-1}$, and all references to $n+1$ with n, without loss of generality.

The two theorems we focus on are the following:

Theorem 2.1.1 (Theorem 5 in [Gar69])**.** *In B_n, every word W can be expressed uniquely in the form $\Delta^m \bar{A}$.*

Theorem 2.1.2 (Theorem 6 in [Gar69])**.** *The necessary and sufficient condition that two words in B_n are equal is that their standard forms are identical.*

The form described in Theorem 2.1.1 is called the **standard form**. In order to understand Theorem 2.1.1 and the standard form, we start by defining a word and associated definitions like word equality, before diving into what is meant by Δ, Δ^m, and $\bar{A}$.

Recall that a **word in the generators** is some ordered sequence in the generators and their inverses. A **positive word** is a sequence consisting only of positive generators, no inverses; geometrically, a positive braid is one that consists of only positive crossings. For a word which consists of more than one generator, we can (usually) use the group relations to transform it into another equivalent word. For example, we can transform $\sigma_1\sigma_2\sigma_1$ into $\sigma_2\sigma_1\sigma_2$, and we can turn $\sigma_1\sigma_3$ into $\sigma_3\sigma_1$. Some words, such as $\sigma_1\sigma_2$, cannot be transformed into any other equivalent positive word using the group relations. If we can transform word A into word B, through however many applications of the group relations, we write $A = B$. If A and B are exactly the same word, i.e. **identically equal**, we write $A \equiv B$. Thus, if we draw A and B as geometric braids when $A = B$, we would have two different pictures, but the braids would be isotopic. If we draw A and B when $A \equiv B$, we would have identical geometric braids.

[1] Interestingly, this paper contains no drawings of braids, yet there are multiple examples of geometric braids in his thesis.

Two words are **positively equal** (written[2] $A \cong B$) if they are both positive, and either identically equal (i.e. $A \equiv B$), or we can transform one into the other such that no intermediate word (obtained through some number of applications of the group relations) contains the inverse of a generator. Of course, if we simply encounter the statement $A \cong B$, then it implies that both A and B are positive.

Garside states that "the **word-length** of a word W is denoted by $L(W)$". No definition is given of what exactly word-length means. However, based on common definitions in algebra and how Garside uses the word-length, we define it as the total number of generators and inverse generators used in a word. We replace the notation $L(W)$ with $||W||$, to avoid confusion with later notation.

Notice that if $A \cong B$, then $||A|| = ||B||$, since we can only rearrange the generators in a positive word, and cannot get rid of any generators through cancellation. This is of course not the case for equal words in general: $||\sigma_1\sigma_1^{-1}|| = 2$, but $\sigma_1\sigma_1^{-1} = 1$ and $||1|| = 0$, since the identity is the empty word. The fact that $A \cong B \Rightarrow ||A|| = ||B||$ is simply stated without proof in Garside's paper, shortly after he defines $\cong$. Only much later does he give the necessary theorem to solidify this fact:

Theorem 2.1.3 (Theorem 4 in [Gar69], the embedding theorem)**.** *In B_n, if two positive words are equal they are positively equal.*

Next we define the **fundamental word** on n strands:

$$\Delta_n = (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1\sigma_2)(\sigma_1).$$

Garside[3] abbreviates Δ_n in B_n as Δ, a convention which is followed in subsequent papers by other authors.

Geometrically, the **fundamental braid** for a group is what we obtain if we take the identity braid in that group and rotate the bottom disc by 180° such that the crossings induced by this motion are positive, to obtain a "positive half twist". Figure 2.2 illustrates Δ_4:

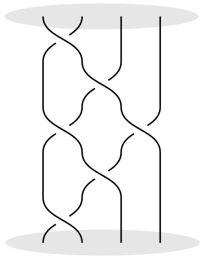


Figure 2.2

[2]Garside write $A \doteq B$, but the dot is easily lost to the eye, so I replace it with $\cong$.

[3]Garside actually defines $\Delta_r \in B_n$ for any $2 \leq r \leq n$, where $\Delta_r = (\sigma_1 \dots \sigma_{r-1})(\sigma_1 \dots \sigma_{r-2}) \dots (\sigma_1\sigma_2)(\sigma_1)$. Geometrically, we would obtain Δ_r by drawing Δ on r strands, and then putting additional strands to the right, going straight down, so that the entire braid is on n strands. While the notion of Δ_r for $r < n$ is useful for some of Garside's proofs, it is not necessary for our discussion.

This is not explicitly defined, but we can safely assume that

$$\Delta^m = \underbrace{\Delta\Delta\ldots\Delta}_{m \text{ times}},$$

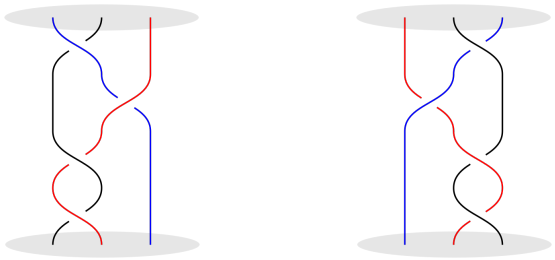
since Δ is a group element and m represents an integer.

Let us explore some further features of Δ, as given in the paper, which are useful going forward. Garside defines a map from the set of generators onto itself, which he calls the reflection in B_n. He denotes it by $\mathfrak{R}$ and it is given by $\mathfrak{R}\sigma_i = \sigma_{n-i}$. This extends to an automorphism on B_n. This name and notation did not last, however: some authors call this the "twisting" automorphism [BGGM08], El-Rifai and Morton describe its action as "turning over" a braid, and calls it a "flip". Furthermore, Thurston denotes it by $\tilde{\sigma}_i = \sigma_{n-i}$, but the notation used by El-Rifai and Morton is the notation that is now in common use:

$$\text{the automorphism } \tau : B_n \to B_n, \text{ defined by } \tau(\sigma_i) = \sigma_{n-i}.$$

Based on its action on an arbitrary generator, and the descriptions given in the above mentioned sources, we can deduce that this map shows us the braid from the back, as if the entire braid was rotated 180° around a vertical axis. In fact, I do not agree with Garside's choice to call it a "reflection", as mirroring/reflecting a braid along the vertical axis does emphatically not produce the same result as $\mathfrak{R}$ (the orientation of the crossings change when we mirror a braid). Nonetheless, the facts presented about this map do indeed hold.

As mentioned, τ is an automorphism on B_n. A group automorphism is an isomorphism from the group to itself and an isomorphism, in turn, is a bijective homomorphism. From this, we conclude that $\tau(AB) = \tau(A)\tau(B)$ and $\tau(B^{-1}) = \tau(B)^{-1}$. Also, $\tau(1) = 1$. Figure 2.3a shows the braid $X = \sigma_1\sigma_2^{-1}\sigma_1^2 \in B_3$ and figure 2.3b shows $\tau(X) = \sigma_2\sigma_1^{-1}\sigma_2^2 \in B_3$. The strands are coloured for the sake of highlighting the rotation.



(a) (b)

Figure 2.3

τ does not affect the positivity of a word; if $P \cong Q$ then $\tau(P) \cong \tau(Q)$. This map allows us to utilise Δ in a number of ways, as laid out in Lemma 2 and Theorem 2 of the paper as discussed below. In particular, these two results allow us to move Δ or Δ^{-1} closer to the start of a braid, which is necessary in order to find the standard form. Later lemmas and theorems in Garside's paper explore the properties of Δ further.

Lemma 2.1.4 (Lemma 2 in [Gar69]). *In B_n, where $i = 1, 2, \ldots, n-1$,*

(*i*) $\sigma_i \Delta \cong \Delta \tau(\sigma_i)$,

(*ii*) $\sigma_i^{-1} \Delta = \Delta (\tau(\sigma_i))^{-1}$,

(*iii*) $\sigma_i \Delta^{-1} = \Delta^{-1} \tau(\sigma_i)$, *and*

(*iv*) $\sigma_i^{-1} \Delta^{-1} = \Delta^{-1} (\tau(\sigma_i))^{-1}$.

Figure 2.4 illustrates statement (*i*) for $\sigma_3 \Delta = \Delta \tau(\sigma_3) = \Delta \sigma_2$ in B_5. I like to think of Δ as a twisted ribbon which delivers σ_i from the top to $\tau(\sigma_i) = \sigma_{n-i}$ at the bottom.

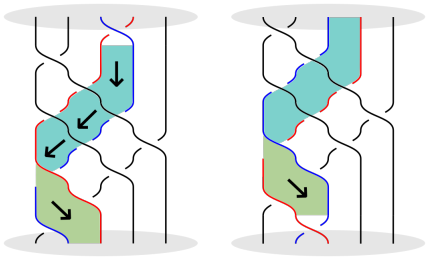


Figure 2.4

Theorem 2.1.5 (Theorem 2 in [Gar69]). *In B_n,*

(*i*) $P \Delta^{2m} \cong \Delta^{2m} P$ *and* $P \Delta^{2m+1} \cong \Delta^{2m+1} \tau(P)$ *for all positive words P and $m \geq 0$,*

(*ii*) $Q \Delta^{2m} = \Delta^{2m} Q$ *and* $Q \Delta^{2m+1} = \Delta^{2m+1} \tau(Q)$ *for all words Q and m any integer.*

The proof of Theorem 2.1.5 relies on repeated applications of Lemma 2.1.4 and another feature of τ: $\tau^2(Q) \equiv Q$ for all words Q. Notice that by the original definition, $\tau(\tau(\sigma_i)) = \tau(\sigma_{n-i}) = \sigma_{n-(n-i)} = \sigma_i$. We can say more about multiple applications of τ; this is not discussed in Garside's paper, but is relevant to later work. Before doing so, I need to list a rather significant result about Δ, given after the standard form theorems in Garside's paper:

Theorem 2.1.6 (Theorem 7 in [Gar69], but first proved in [Cho48]).

(*i*) *When $n = 2$, the centre of B_n is generated by Δ.*

(*ii*) *When $n > 2$, the centre of B_n is generated by Δ^2.*

Recall that the centre of a group is the set of elements which commute with any element of the group, usually denoted by $Z(G)$, where G is whichever group we are considering. So, from the above theorem, for the braid group on 3 or more strands, any even power of Δ will commute with arbitrary braids. Notice that $\Delta_2 = \sigma_1$, and commutes with any braid in B_2, since σ_1 is the only generator of said group.

We can see from statement (*ii*) of Theorem 2.1.5 that $\Delta^{-(2m+1)} Q \Delta^{2m+1} = \tau(Q)$, where m is any integer. This can be rewritten as $\Delta^{-2m} \Delta^{-1} Q \, \Delta^{2m} \Delta$, and since $2m$ is even for all integers m, we have by Theorem 2.1.6 that $\tau(Q) = \Delta^{-2m} \Delta^{2m} \Delta^{-1} Q \, \Delta = \Delta^0 \Delta^{-1} Q \, \Delta = \Delta^{-1} Q \, \Delta$. In

fact, statement (i) of Lemma 2.1.4 confirms that $\tau(\sigma_i) = \Delta^{-1}\sigma_i\Delta$. We use this definition of τ as conjugation by Δ throughout this dissertation.

We can consider multiple applications of τ. For a non-negative integer k, we write $\tau^k(Q)$ to mean $\underbrace{\tau(\tau(\ldots(\tau}_{k \text{ times}}(Q))))$. Thus, $\tau^k(Q) = \underbrace{\Delta^{-1}\Delta^{-1}\ldots\Delta^{-1}}_{k \text{ times}}\, Q\, \underbrace{\Delta\ldots\Delta\Delta}_{k \text{ times}} = \Delta^{-k}\, Q\, \Delta^k$. Furthermore, by Theorem 2.1.6, we have that $\tau^k(Q) = \tau^{k \mod 2}(Q)$ for all braids Q.

Let's think about this geometrically: one application of τ allows us to view the back of the braid; a second application will let us view the back of the back – i.e. the front of the original braid! Clearly, a third application of τ will show us the back again. We can keep this up, and see that any even number of applications of τ gives us the original braid, as we're turning the braid over an even number of times. Any odd number of applications returns the back of the braid, which is the same as one application of τ. Of course, 0 applications of τ just leaves the braid as it is.

What happens when k is a negative integer? Later authors often use τ^{-m}, for any value of m. We can think of τ^{-1} as the inverse map of the map $\tau : B_n \to B_n$. If τ turns a braid over, then "undoing" τ (i.e., applying τ^{-1}) will simply show the front of the braid, which is accomplished by turning the braid over again. In other words, $\tau^{-1}(\tau(Q)) = Q$. To get from $\tau(Q) = \Delta^{-1}\, Q\, \Delta$ to Q, we need to conjugate by Δ^{-1}, to get $\Delta\Delta^{-1}\, Q\, \Delta\Delta^{-1} = Q$. Thus $\tau^{-1}(A) = \Delta\, A\, \Delta^{-1}$ for $A \in B_n$. We also have that $\tau^{-1}(\tau(Q)) = Q = \tau(\tau(Q))$, and by our geometric argument, τ^{-1} performs the same action as τ. In fact, Thurston tells us in [Eps92] that τ is an involution, that is, a map which is its own inverse. Thus,

$$\Delta^{-1}\, Q\, \Delta = \tau(Q) = \tau^{-1}(Q) = \Delta\, Q\, \Delta^{-1}$$

for all $Q \in B_n$. The astute reader may notice that statement (iii) of Lemma 2.1.4 actually tells us that $\tau(\sigma_i) = \Delta\, \sigma_i\, \Delta^{-1}$, confirming that $\tau = \tau^{-1}$. However, we still define τ^{-1} as conjugation by Δ^{-1}.

Thus, in general, for any braid $Q \in B_n$ and integer m, we write

$$\tau^m(Q) = \Delta^{-m}\, Q\, \Delta^m,$$

from which we have the following corollary:

Corollary 2.1.7. *For any braid $Q \in B_n$ and any integer m, we can rewrite $Q\Delta^m$ as $\Delta^m\tau^m(Q)$ and $\Delta^m Q$ as $\tau^{-m}(Q)\Delta^m$.*

Before we can look at the next lemma, we need one more definition. If $P \equiv x_1x_2\ldots x_t$ is any word, where each x_i is a generator or the inverse of a generator, the x_i not necessarily distinct, then by the **reverse** of P (written $\operatorname{rev} P$) we mean the word $x_tx_{t-1}\ldots x_2x_1$. Note that $\operatorname{rev} PQ = \operatorname{rev} Q \operatorname{rev} P$ and if $P \cong Q$ then $\operatorname{rev} P \cong \operatorname{rev} Q$.

Lemma 2.1.8 (Lemma 3 in [Gar69])**.** *In B_n,*

(i) $\tau(\Delta) \cong \Delta$,

(ii) $\operatorname{rev}\Delta \cong \Delta$.

The following lemma is elementary to the standard form:

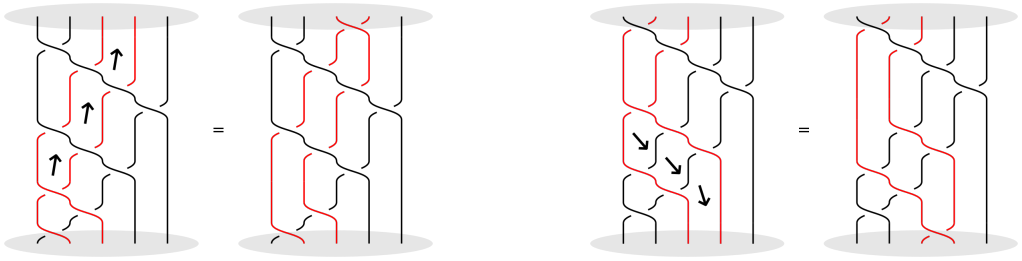
Lemma 2.1.9 (Lemma 4 in [Gar69]). *In B_n, there exist positive words X_i, Y_i such that for $i = 1, 2, \ldots, n-1$,*

$$\sigma_i X_i \cong \Delta \cong Y_i \sigma_i.$$

This lemma also tells us that the word Δ can be written such that it starts or ends with any chosen generator of the group.

Knowing how to find the words X_i and Y_i for a particular generator is very useful when we convert a word into its standard form. Unfortunately, Garside's proof of this lemma does not make it clear how to find such words, only that they exist. However, we can see how to do it by first inspecting Δ as a geometric braid.

If we want to pull σ_k up to the start, we consider the strands which start at fixed points k and $k+1$ and follow these down the braid until they cross. This crossing we then pull up. If we want to pull σ_k down to the end, we consider the strands which end at fixed point k and $k+1$ and follow these upwards until they cross. This crossing is then pulled down. This leaves us with the factorisation of Δ as desired. Figure 2.5a illustrates pulling σ_3 up in Δ_5, and figure 2.5b illustrated pulling σ_3 down in Δ_5. I have drawn these braids so the crossings occur in exactly the same order as given in the definition of Δ, as this makes the purely algebraic description of this process clearer.



(a) (b)

We can turn this into an algebraic process. Recall that

$$\Delta = (\sigma_1 \ldots \sigma_{n-1})(\sigma_1 \ldots \sigma_{n-2}) \ldots (\sigma_1\sigma_2)(\sigma_1).$$

If we want to take σ_k to the start, we consider the kth factor in the word Δ (as written above), and delete σ_1 from this factor. Then, we write σ_k at the beginning of the word. Thus, we get

$$\Delta = \sigma_k \cdot (\sigma_1 \ldots \sigma_{n-1})(\sigma_1 \ldots \sigma_{n-2}) \ldots (\sigma_2 \ldots \sigma_{n-k}) \ldots (\sigma_1\sigma_2)(\sigma_1) = \sigma_k X_k.$$

This exact expression holds for $2 < k < n-2$ in particular, for the sake of illustration. The method described holds for any $1 \leq k \leq n-1$.

To take σ_k to the end, we will proceed in a similar fashion, but "backwards". We delete σ_1 from the factor which is kth from last, and then write σ_k at the end. Thus,

$$\Delta = (\sigma_1 \ldots \sigma_{n-1})(\sigma_1 \ldots \sigma_{n-2}) \ldots (\sigma_2 \ldots \sigma_k) \ldots (\sigma_1\sigma_2)(\sigma_1) \cdot \sigma_k = Y_k \sigma_k.$$

Again, this exact expression is for illustration purposes.

Since we know exactly how to find X_i and Y_i for arbitrary σ_i, we can write $\sigma_i = \Delta X_i^{-1} = Y_i^{-1}\Delta$, and so $\sigma_i^{-1} = X_i\Delta^{-1} = \Delta^{-1}Y_i$. Thus, we can replace any inverse generator with an inverse Δ and a positive braid, and then use τ to move the Δ^{-1} closer to the start, which is a key part of the standard form.

We have now said enough about Δ. It remains for us to investigate the term $\bar{A}$ in Theorem 2.1.1, to understand the standard form. In order to do this, we discuss Section 3.2 of [Gar69], as this introduces us to the remaining prerequisite knowledge for understanding and proving Theorem 2.1.1.

Let W be any positive word, and let $W, W_1, W_2, \ldots, W_p$ be the complete set of distinct words which are positively equal to W. Garside calls this set the **diagram**[4] of W, denoted by $D(W)$. The individual words $W, W_1, \ldots, W_p$ are called the **routes** of $D(W)$. The process of enumerating the routes of $D(W)$ is called **drawing the diagram** of $D(W)$.

If $W \cong AXB$ ($||A|| \geq 0, ||B|| \geq 0$), we say that X is a **subroute** of $D(W)$. If $||A|| = 0$, then X is an **initial subroute** of $D(W)$. If any subroute of $D(W)$ is the fundamental word Δ (in other words, if $W \cong A\Delta B$ where $||A|| \geq 0, ||B|| \geq 0$), we say Δ is a **factor** of W or W **contains** Δ. From Theorem 2.1.5, if W contains Δ, then $W \cong \Delta X$ for some (positive) X. Thus, if Δ is a factor of W, there is an equivalent positive word with Δ at the start. If W is a positive word which does *not* contain Δ, we say W **is prime to** Δ.

Finally, we get to understanding the meaning of $\bar{A}$. In B_n, suppose W has word-length L, and suppose $D(W)$ consists of t words $W_1 \equiv \sigma_{j_1}\sigma_{j_2}\sigma_{j_3}\ldots\sigma_{j_L}$, $W_2 \equiv \sigma_{k_1}\sigma_{k_2}\sigma_{k_3}\ldots\sigma_{k_L}$, $\ldots$, $W_t \equiv \sigma_{q_1}\sigma_{q_2}\sigma_{q_3}\ldots\sigma_{q_L}$. Then there is a one-to-one correspondence between the words W_1, W_2, $\ldots$, W_t and the set of numbers $P_1 = j_1j_2j_3\ldots j_L$, $P_2 = k_1k_2k_3\ldots k_L$, $\ldots$, $P_t = q_1q_2q_3\ldots q_L$ where each number P_i is expressed, as Garside says, "in the scale of n", and consists of L digits. We may take "in the scale of n" to mean the more commonly-understood "in base n". Garside uses the word "base" as part of a definition we are about to see, which may explain his use of "scale".

The numbers P_i are all distinct. Suppose the smallest number is P_r. Then the corresponding word W_r, which is uniquely defined, will be called the **base** of $D(W)$. If A is a positive word which is prime to Δ, then we denote the base of A by $\bar{A}$. When we see the notation $\bar{A}$, it implies that A is positive and prime to Δ. As an example, let $W_1 \equiv \sigma_1\sigma_2\sigma_1\sigma_3$. Equivalent to this are $W_2 \equiv \sigma_2\sigma_1\sigma_2\sigma_3$ and $W_3 = \sigma_1\sigma_2\sigma_3\sigma_1$. The base of the diagram of this braid will be $\sigma_1\sigma_2\sigma_1\sigma_3$ since $1213 < 1231 < 2123$ (working "in the scale of" 4).

Recall that the standard form of a word W is the unique form $\Delta^m\bar{A}$. The value m is called the **power** of W in Garside's work. The proof of Theorem 2.1.1, which gives us the standard form, makes it clear how we are to find m and A, and we just saw how we can find $\bar{A}$ given A. Probably the slowest part is that we have to find the entire $D(W)$ (i.e., all possible words positively equal to W) in order to accomplish anything.

We are now ready to consider the proof of the core theorem of this section.

Theorem (Theorem 5 in [Gar69], the standard form)**.** *In B_n, every word W can be expressed*

[4] In his paper, Garside uses Cayley diagrams to illustrate all the different ways one could "build" the word W; I leave this and related notions out for the sake of brevity.

uniquely in the form $\Delta^m \bar{A}$.

Proof of Theorem 5 in [Gar69], the standard form. The following algorithm[5] finds the standard form of a positive braid word P:

Algorithm 2.1.10.
Input: P*, a positive braid word.*
Output: $\Delta^t \bar{A}$*, the standard form of* P*.*

1. Compute the set $D(P)$.
2. Select any route starting with as many consecutive subroutes Δ as possible, equal to, say, t ($t \geq 0$).
3. Write $P \cong \Delta^t A$.
4. Determine $\bar{A}$, the base of A.
5. Return $\Delta^t \bar{A}$.

Note that A found in Step 3 is prime to Δ, as otherwise there would be a route of $D(P)$ starting with more than t consecutive subroutes Δ.

The following algorithm finds the standard form of a general braid word W in B_n:

Algorithm 2.1.11.
Input: W*, a general braid word.*
Output: $\Delta^m \bar{A}$*, the standard form of* W*.*

1. Write W such that
$$W \equiv W_1 (x_1)^{-1} W_2 (x_2)^{-1} \dots (x_s)^{-1} W_{s+1},$$
where each W_i is positive and satisfies $||W_i|| \geq 0$, and the x_i are generators.
2. Use Lemma 2.1.9 to rewrite each x_i^{-1} as $(x_i)^{-1} = X_i \Delta^{-1}$, so we have $W = W_1 X_1 \Delta^{-1} W_2 X_2 \Delta^{-1} \dots W_s X_s \Delta^{-1} W_{s+1}$.
3. Use Corollary 2.1.7 to move all factors Δ^{-1} to the start, and write this resulting word as $\Delta^{-s} P$ where P is positive.
4. Use Algorithm 2.1.10 to find the standard form of P as $\Delta^t \bar{A}$.
5. Set $m = -s + t$.

[5]Note that Garside does not give any algorithms in step-by-step pseudocode. His algorithms have been reworked for clarity and consistency with the rest of this dissertation.

6. Return $\Delta^m \bar{A}$

In Step 1 we group together any consecutive positive generators into positive words, and keep track of all the inverse generators in the word. Note that any of the positive W_i can be 1 as needed.

It remains to show that this form above is unique. Suppose

$$\Delta^m \bar{A} = \Delta^p \bar{B}.$$

First suppose that $p < m$, and let $m-p = u$, so $u > 0$. Then, from $\Delta^m \bar{A} = \Delta^p \bar{B}$ we have that $\Delta^u \bar{A} = \bar{B}$, and thus by Theorem 2.1.3, $\Delta^u \bar{A} \cong \bar{B}$. Hence, $\bar{B}$ contains Δ, which is impossible. Thus, $p \not< m$. By a similar argument we can show that $m \not< p$. Hence, we can conclude that $m = p$, and so the statement becomes $\Delta^m \bar{A} = \Delta^m \bar{B}$, which in turn means that $\bar{A} = \bar{B}$. By Theorem 2.1.3, this means that $\bar{A} \cong \bar{B}$. However, any positive word has one and only one base, and therefore $\bar{A} \equiv \bar{B}$. Thus, the form $W = \Delta^m \bar{A}$ is unique. □

The power of a word is of the greatest importance when it comes to Garside's solution to the conjugacy problem. But before we get to that, it is time to look at the left- and right-canonical forms, as laid out by El-Rifai and Morton.

2.2 The canonical form(s)

The paper we are considering is *Algorithms for positive braids* by El-Sayed A. El-Rifai and Hugh R. Morton [ERM94]. This is based on the first chapter of El-Rifai's D.Phil thesis, *Positive braids and Lorenz links* [ER88]. If we compare Garside's standard form to the left-canonical form that we are about to see, the main difference is that $\bar{A}$ in $W = \Delta^m \bar{A}$ gets factorised in a very particular way. In fact, the main theorem tells us how to find a unique factorisation for arbitrary positive braids. Getting an arbitrary word W into the form $\Delta^{-s} P$ where P is positive will be done in the same way as in the proof of the standard form in [Gar69], but the word algorithm will allow us to write P as $\Delta^t A$ where A is positive and prime to Δ without drawing the whole diagram $D(P)$ of P, and it allows us to factorise A uniquely. What's the point? The factorisation gives us more information about the braid, which leads to the improvement of the conjugacy solution. The factorisation also allows for easier representation of braids in computer memory.

Below is the key theorem we need to understand for the left-canonical form, for which we will introduce the notation shortly:

Theorem 2.2.1 (Theorem 2.9 in [ERM94])**.** *There is a unique expression for $P \geq 1$ as $P = A_1 A_2 \dots A_s$ with $A_i \in [0, 1]$, $A_s \neq 1$ and $S(A_{i+1}) \subset F(A_i)$ for each i.*

In their paper, El-Rifai and Morton define a partial order[6] on braids, using positive braids. Later in their paper they identify B_n^+, the **set of all positive braids**, as a semigroup, which

[6] A (non-strict) partial order is a relation that satisfies the reflexivity, antisymmetry, and transitivity axioms.

is a set equipped with a binary operation which obeys the closure and associativity axioms (no requirements of an identity element or invertibility). However, in later sources we see B_n^+ identified as a **monoid**, as it does indeed have an identity element. In fact, the very first feature the authors point out after defining the partial order requires $1 \in B_n^+$. Let's take a look.

For $A, B \in B_n$, we write $A \leq B$ when $B = C_1 A C_2$ for some $C_1, C_2 \in B_n^+$. It is worth noting that the authors later use $\geq$ as well, but do not explicitly define it. We can assume $B \geq A$ simply means $A \leq B$. This is not always the case; in Chapter 4, we discuss a partial order which cannot simply be "flipped" as here.

After defining the partial order, the authors state that

$$B \in B_n^+ \Leftrightarrow 1 \leq B \tag{2.1}$$

and

$$A \leq B \Leftrightarrow B^{-1} \leq A^{-1}. \tag{2.2}$$

These are not proven in the paper, but we will check this as an exercise to deepen our understanding of the partial order and the monoid B_n^+

Proof of statement 2.1. [$\Leftarrow$] If $1 \leq B$, then $B = D_1 1 D_2$ for some $D_1, D_2 \in B_n^+$, by definition of $\leq$. B_n^+ is closed under braid concatenation[7], so if indeed $1 \in B_n^+$, then $B \in B_n^+$ too.
[$\Rightarrow$] If $B \in B_n^+$, we know that it is a positive braid. Then we can write $B = B11 = C_1 1 C_2$, and since $C_1 = B \in B_n^+$ and $C_2 = 1 \in B_n^+$ too, we have that $1 \leq B$, by definition of $\leq$.
Hence, $B \in B_n^+ \Leftrightarrow 1 \leq B$. □

The authors often use statement 2.1 in their writing, as they prefer to write $P \geq 1$ instead of $P \in B_n^+$. For consistency with later papers, we write $P \in B_n^+$ to indicate P is a positive braid.

Proof of statement 2.2. If $A \leq B$ (where $A, B \in B_n$), we have $B = C_1 A C_2$ for $C_1, C_2 \in B_n^+$. Inverting both sides of the equality, we get that $B^{-1} = C_2^{-1} A^{-1} C_1^{-1}$. Thus $C_2 B^{-1} C_1 = C_2 C_2^{-1} A^{-1} C_1^{-1} C_1 = A^{-1}$. Hence $A^{-1} = C_2 B^{-1} C_1$ so $B^{-1} \leq A^{-1}$. We can show similarly that $B^{-1} \leq A^{-1} \Rightarrow A \leq B$. Thus, $A \leq B \Leftrightarrow B^{-1} \leq A^{-1}$. □

This partial order essentially tells us that A is a factor of B (if indeed $A \leq B$), but it also tells us that if we remove the factor of A from wherever it appears in the factorisation, whatever remains will be positive. Comparing arbitrary braids does not hold much value in what we are aiming to do, but comparison to the fundamental braid is, well, fundamental. We consider Lemmas 1.1-1.3 of the paper:

Lemma 2.2.2 (Lemma 1.1 in [ERM94])**.** *Each generator σ_i satisfies $1 \leq \sigma_i \leq \Delta$.*

[7] Concatenating a positive braid with another positive braid will never introduce a negative crossing, hence B_n^+ is closed under braid concatenation

This is shown very easily from the fact that every generator of B_n appears in the word Δ and the fact that the generators are positive.

Lemma 2.2.3 (Lemma 1.2 in [ERM94]). *If $A \leq \Delta^s$ then $\Delta^s = D_1 A = A D_2$ for some $D_1, D_2 \in B_n^+$.*

Lemma 2.2.4 (Lemma 1.3 in [ERM94]). *If $\Delta^p \leq A$ then $A = E_1 \Delta^p = \Delta^p E_2$ for some $E_1, E_2 \in B_n^+$.*

These lemmas are easily proven using Corollary 2.1.7, i.e., that $\Delta^k B = \tau^k(B)\Delta^k$ and $B\Delta^k = \Delta^k \tau^k(B)$ for any braid B and any integer k.

From Lemmas 2.2.3 and 2.2.4, we obtain the following corollary:

Corollary 2.2.5 (Corollary 1.4 in [ERM94]). *If $\Delta^{p_1} \leq B \leq \Delta^{s_1}$ and $\Delta^{p_2} \leq C \leq \Delta^{s_2}$ then*

$$\Delta^{p_1+p_2} \leq BC \leq \Delta^{s_1+s_2}.$$

Finally, we can place every braid somewhere between powers of Δ, by using Corollary 2.2.5, and the fact that $1 \leq \sigma_i \leq \Delta$ and $\Delta^{-1} \leq \sigma_i^{-1} \leq 1$:

Theorem 2.2.6 (Theorem 1.5 in [ERM94]). *Every braid B satisfies $\Delta^p \leq B \leq \Delta^s$ for some $p, s \in \mathbb{Z}$ with $p \leq s$.*

The following is not shown in the paper, but is useful later:

Proposition 2.2.7. *If $A \leq B$ for $A, B \in B_n$, then $\tau^m(A) \leq \tau^m(B)$ for any integer m. Moreover, if $\Delta^p \leq C \leq \Delta^s$ for $C \in B_n$ and some integers p and s, then $\Delta^p \leq \tau^l(C) \leq \Delta^s$ for any integer l.*

Proof. Suppose $A \leq B$, so that $B = C_1 A C_2$ for $C_1, C_2 \in B_n^+$. Then $\tau^m(B) = \Delta^{-m} B \Delta^m = \Delta^{-m} C_1 A C_2 \Delta^m = \tau^m(C_1)\Delta^{-m} A \Delta^m \tau^m(C_2) = \tau^m(C_1)\tau^m(A)\tau^m(C_2)$. Since $\tau^m(C_1), \tau^m(C_2) \in B_n^+$, we have that $\tau^m(A) \leq \tau^m(B)$.

Since $\tau^l(\Delta^k) = \Delta^{-l}\Delta^k\Delta^l = \Delta^{-l+k+l} = \Delta^k$ for any integer k, it follows that if $\Delta^p \leq C \leq \Delta^s$, then $\Delta^p \leq \tau^l(C) \leq \Delta^s$ for any $C \in B_n$, by the first result of this proposition. □

We have the notation where $[p, s] \subset B_n$ is the proper subset $\{B \in B_n : \Delta^p \leq B \leq \Delta^s\}$. If we now look back at Theorem 2.2.1, we can see that $A_i \in [0, 1]$ means that $1 \leq A_i \leq \Delta$. It remains for us to see what is special about $[0, 1]$, and to investigate what $S(A_{i+1}) \subset F(A_i)$ means.

The authors define the starting sets and finishing sets for a positive braid $P \in B_n^+$ as follows: The **starting set**, $S(P) \subset \{1, \ldots, n-1\}$, is the set

$$S(P) = \{i \mid \exists P_i \in B_n^+ \text{ s.t. } P = \sigma_i P_i\}.$$

Similarly, the **finishing set**, $F(P)$, is the set

$$F(P) = \{i \mid \exists P_i \in B_n^+ \text{ s.t. } P = P_i \sigma_i\}.$$

In essence, these sets tell us which crossings could be "pulled" to the start (respectively end) of the braid, without breaking the braid's positivity. For example, $\sigma_1\sigma_3\sigma_2 \in B_4$ can also be written as $\sigma_3\sigma_1\sigma_2$, and so its starting set will be $\{1, 3\}$ and its finishing set is $\{2\}$. Notice that Lemma 2.1.9 tells us that $S(\Delta) = F(\Delta) = \{1, \ldots, n-1\}$.

From here, we can define the key notion we will use to factorise P uniquely in $W = \Delta^m P$. A positive factorisation $P = AB$ with $A, B \in B_n^+$ is a **left-weighted factorisation** if $S(B) \subset F(A)$, and is **right-weighted** if $S(B) \supset F(A)$. The authors use left-weightedness extensively, but this is simply a preference. All proofs and definitions involving left-weightedness can be shown for right-weightedness as well. Later the authors define the left-canonical form for a braid, but we could just as well use the right-canonical form.

We can think about left-weightedness as follows: if AB is a left-weighted factorisation, then for every possible generator we could pull to the start of the second factor, B, the same generator could be pulled to the end of A. If a factorisation is not left-weighted, there will be a generator, say σ_i, that could come to the start of B but not to the end of A. In this case, moving said generator over to A, to make the new factorisation $A\sigma_i \cdot \sigma_i^{-1}B$ (which is still positive, as σ_i^{-1} will cancel with σ_i at the start of B) is part of turning the factorisation into a left-weighted one; if we do this for all generators that are in $S(B)$ but not in $F(A)$, eventually we will reach the point where indeed $S(B) \subset F(A)$.

A key lemma towards our goal, Theorem 2.2.1, is the following:

Lemma 2.2.8 (Lemma 2.2 in [ERM94])**.** *Every $P \in B_n^+$ has a unique left-weighted factorisation $P = A_1P_1$ with $A_1 \in [0, 1]$ and $P_1 \in B_n^+$. Every other positive factorisation $P = AB$, with $A \in [0, 1]$ and $B \in B_n^+$, satisfies $AQ = A_1$ for some $Q \in B_n^+$.*

Thus, we can think of A_1 as the "greatest" factor in $[0, 1]$ that P could start with – any other factor in $[0, 1]$ that P could start with is, in fact, a factor of A_1.

Following Lemma 2.2.8, the authors spend some time investigating braids in $[0, 1]$, which I summarise briefly, mentioning some key statements, before moving on to the unique form for braids, and the algorithm to find it for arbitrary braids.

A braid $A \in B_n^+$ is called a **positive permutation braid**[8] if it can be drawn as a positive geometric braid in which every pair of strands crosses at most once. The authors denote the set of positive permutation braids by S_n^+. Thurston shows in [Eps92] that there is a bijective correspondence between S_n^+ and Σ_n, the set of permutations on n elements, so the reason for the name is clear. Furthermore, there are $n!$ possible positive permutation braids on n strands. We can take any braid in B_n and find an associated permutation – if we number the fixed points at the top and bottom of the braid from 1 to n and see how each strand connects a number at the top to a number at the bottom, we get a permutation on n elements. In, for example, $\sigma_1\sigma_2$ the strands connect 1 to 3, 2 to 1, and 3 to 2. Thus, this braid has associated permutation (132). However, multiple different braids can have the same associated permutation. Thus, the map from the permutations to braids needs to be restricted to special braids, so that there is a unique braid for each permutation. Positive braids in which every pair of strands crosses at most once work as required. Later authors

[8] Thurston first introduced these kinds of braids and showed the connection to permutations, but he called them "non-repeating braids" in his work.

call positive permutation braids **simple braids**, and we will use this terminology. We will also let the set of simple braids on n strands be denoted by S_n, instead of S_n^+.

If we want to draw a simple braid which corresponds to the permutation $\pi \in \Sigma_n$, we start with a braid outline (two discs the unit distance apart with n fixed points on each). Draw a line between each i at the top and associated $\pi(i)$ at the bottom, so that the pairs of lines cross at most once. This is a familiar way to visualise the permutation π. Turn it into a braid diagram by separating the lines at every crossing to make a positive crossing.

The computational advantage of using simple braids is clear: we can refer to a simple braid uniquely using only a permutation – no braid generators required!

Theorem 2.2.9 (Theorem 2.6 in [ERM94]). *The subsets* $[0,1]$ *and* S_n *of* B_n *are identical.*

Thus, the set of simple braids is $\{B \in B_n \mid 1 \leq B \leq \Delta\}$. Notice that $B \leq \Delta$ means $\Delta = BA_1 = A_2B$ where $A_1, A_2 \in B_n^+$. Thus, simple braids are initial or final factors of Δ. Furthermore, except for Δ itself, simple braids do not "contain" Δ as a factor. Notice that if Δ is a factor of a positive braid which itself is not Δ, then that braid cannot be a simple braid: every pair of strands crosses exactly once in Δ; if we attach more crossings before or after Δ, there would be pairs of strands crossing for a second time, and so the braid would not be simple. However, also take note that while "simple braid" $\Rightarrow$ "does not contain Δ as a factor", the other direction does not hold. A non-simple braid can also not contain any Δ factor. When we look at a geometric example of the left-canonical form at the end of this section, we work with a braid which is positive, not simple, and does not contain Δ as a factor.

We also encounter the following corollary:

Corollary 2.2.10 (Corollary 2.8 in [ERM94]). *Let* $P \in B_n^+$ *have left-weighted factorisation* $P = A_1P_1$, *with* $A_1 \in [0,1]$. *Then* $S(A_1) = S(P)$.

Thus, every generator that could be pulled to the start of P can be pulled to the start of $A_1 \in [0,1]$ in the left-weighted factorisation $P = A_1P_1$. Using this, we can prove Theorem 2.2.1, which gives the **left-canonical form for positive braids**, as stated at the start of this section:

Theorem (Theorem 2.9 in [ERM94]). *There is a unique expression for* $P \in B_n^+$ *as* $P = A_1A_2 \dots A_s$ *with* $A_i \in [0,1]$, $A_s \neq 1$ *and* $S(A_{i+1}) \subset F(A_i)$ *for each* i. *In other words, all factors are simple, and the factorisation is left-weighted.*

Proof of Theorem 2.9 in [ERM94]. Let $P_i = A_{i+1} \dots A_s$ for each $0 \leq i \leq s$ (so $P_0 = P$). Then A_iP_i is the unique left-weighted factorisation of P_{i-1}. This follows by downwards induction on i since $S(P_i) = S(A_{i+1})$ from Corollary 2.2.10. □

The process of turning P into its left-canonical form is called the **word algorithm**.

Notice that any number of the A_i at the start could all be Δ. If there are any Δs, they will all be on the left (by left-weightedness), so we can collect them to get $P = \Delta^q P'$ such that $P' \in B_n^+$ but $P' \not\geq \Delta$, i.e. P' is positive but does not contain Δ as a factor. In Garside's

language, we would say that P' is prime to Δ. P' might be simple but is not necessarily so. Once we have the positive braid in this form, we can turn P' into its left-weighted factorisation.

For a general braid $B \geq \Delta^p$ we can write $B = \Delta^p B'$ with $B' \in B_n^+$, using Lemma 2.2.4 , which means $B' = \Delta^{-p} B \in B_n^+$. Then we write $B' \in B_n^+$ in left-canonical form and use Corollary 2.1.7 collect all the Δs to the start to get, say, $\Delta^q A_1 \dots A_r$, so that finally the **left-canonical form of the general braid** B, where B is an arbitrary braid, is $\Delta^{p+q} A_1 \dots A_r$ where $A_1 \neq \Delta$. In computer memory, we would store this as an integer indicating the power of Δ, and a sequence of permutations indicating each of the factors A_1 to A_r.

Writing B as $\Delta^p B'$ is done the same way as done for the standard form. In the worst case scenario, we rewrite each σ_i^{-1} as $\Delta^{-1} Y_i$ where $Y_i \in [0,1]$ (by Lemma 2.1.9) and then we use Corollary 2.1.7 to collect all the powers of Δ to the left. However, El-Rifai and Morton state that it is generally possible to handle consecutive inverse generators too. This is more efficient than working with one inverse generator at a time, as it reduces the number of times we have to shift Δ^{-1} towards the start, and leads to a shorter remaining positive braid to convert to left-canonical form. They do not, however, specify how this would be done. The way I would do it, is to look for the longest negative subword such that its inverse is a simple braid. In particular, we know every simple braid $S \in [0,1]$ satisfies $\Delta = SC_1 = C_2 S$ where $C_1, C_2 \geq 1$ (in fact $C_1, C_2 \in [0,1]$ because $SC_1 1 = \Delta \Rightarrow C_1 \leq \Delta \Rightarrow C_1 \in [0,1]$ and similarly for C_2), so $S^{-1} = C_1 \Delta^{-1} = \Delta^{-1} C_2$. It is easy to find $C_1 = S^{-1}\Delta$ and $C_2 = \Delta S^{-1}$, and so if we have S^{-1} (where $S \in [0,1]$) in our braid word, we can replace it with a positive braid word and an inverse Δ, just as we would do for a single inverse generator.

I find the geometric description of the word algorithm very intuitive, so I will explain that, before giving the purely algebraic description. For a positive braid, consider its geometric representation. Work down this braid to first partition it into simple braids – to do so, follow the strands down the braid until a pair of strands that are about to cross for a second time are encountered. Before this second crossing, indicate the start of the next simple braid. Continue creating subsequent partitions until the bottom of the braid is reached. Spotting the second time a pair of strands is about to cross is made easier by colouring the strands.

Once the positive braid has been factorised into simple braid factors, we can work on the left-weightedness aspect. Look at adjacent pairs of simple braids, and see if any pair of strings crosses in the lower but not the upper braid. If so, and if both factors will still be simple after moving the crossing, then move the crossing up. It is possible for a crossing of two strands to occur in the lower braid and not the upper braid but in such a way that moving it up will pull another crossing along, leading to the new upper braid not being simple; this possibility is illustrated in the example to come. Continue checking adjacent simple braids. Otherwise, we can stop. Eventually we'll have "pushed up" as many crossings as possible, while keeping the factors simple. Delete any factors which have been turned into the identity braid. Note that if we were putting the braid into right-canonical form, we would be proceeding much the same, but pushing crossings down instead of up. Figure 2.6 shows an example of the braid $P = \sigma_1 \sigma_3 \sigma_2^2 \sigma_3 \sigma_1 \sigma_3 \sigma_2 \sigma_3 \sigma_2$ being turned into its left-weighted factorisation, as illustrated in the paper.

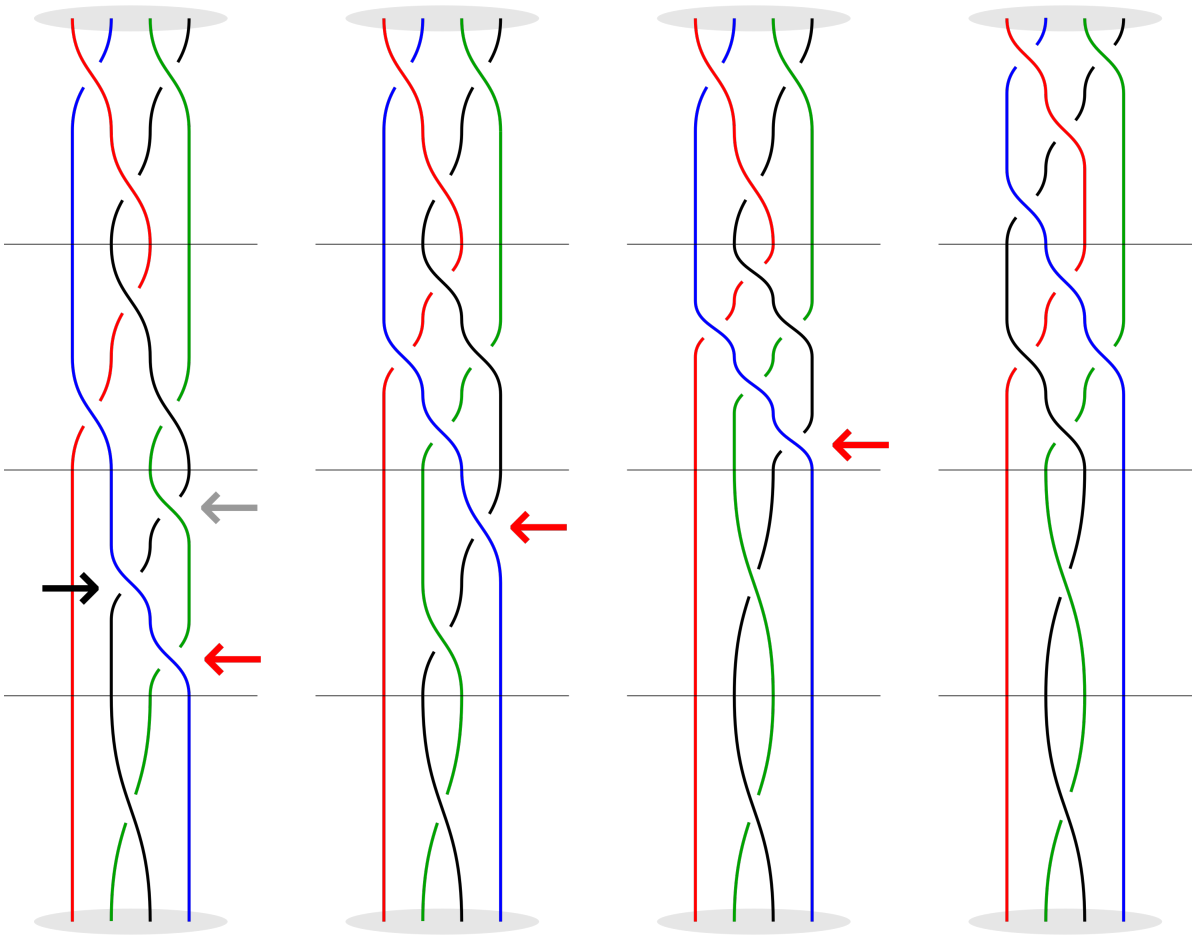


Figure 2.6

In this illustration, we can see that the initial braid (on the left of figure 2.6) has been turned into four simple braids. For the sake of illustration, this braid has been stretched so that each simple braid is of unit length. From here we make the factorisation left-weighted by moving up the crossing that is indicated by the red arrow (and similarly at the next stages). Notice the crossing marked with a black arrow. This crossing of the blue and black strands does not occur earlier in the braid, but if we were to move it up, we would also drag along the crossing marked by the grey arrow, hence obtaining a non-simple braid, which we do not want.

We are left with $P = \sigma_1\sigma_3\sigma_2\sigma_1 \cdot \sigma_2\sigma_1\sigma_3\sigma_2 \cdot \sigma_2 \cdot \sigma_2$ or, as an integer indicating the power of Δ and a sequence of permutations on n elements: $0, (134) \cdot (13)(24) \cdot (23) \cdot (23)$.

The authors give the algorithm[9] in purely algebraic terms too. Before we can consider this, we need an algorithm to find the starting- and finishing set of a given simple braid. To do so requires the following lemma, which allows us to find the starting set of a simple braid with incredible ease:

[9]El-Rifai and Morton also do not give any of their algorithms in step-by-step pseudocode. Their algorithms have been reworked for consistency with the rest of this dissertation.

Lemma 2.2.11 (Lemma 2.4 in [ERM94])**.** *For $A \in [0,1]$ with associated permutation π, the following are equivalent:*

(*i*) $i \in S(A)$.

(*ii*) *strands i and $i+1$ cross in A.*

(*iii*) $\pi(i+1) < \pi(i)$.

In particular, i is in the starting set if and only if strands i and $i+1$ cross in the simple braid. More importantly, if we have the associated permutation for the braid on hand, we can check the output of the permutation for each adjacent pair of strands, and obtain the starting set in this way: clearly if strand i crosses strand $i+1$, then strand i will "end up" at a fixed point further to the right than where strand $i+1$ "ends up" at; this information is completely captured in the comparison between $\pi(i)$ and $\pi(i+1)$.

To find the finishing set, we simply consider the reverse of the simple braid (this is stated in the remark following the proof of Lemma 2.5 in [ERM94]). Thus, we have:

Corollary 2.2.12. *For $A \in [0,1]$, consider* rev $A \in [0,1]$ *and the permutation ψ associated to* rev A*. The following are equivalent:*

(*i*) $i \in F(A)$.

(*ii*) *strings i and $i+1$ cross in* rev A.

(*iii*) $\psi(i+1) < \psi(i)$.

With this in mind, I have written the following algorithm to find the starting and finishing set of a simple braid:

Algorithm 2.2.13.
Input: $A \in [0,1]$ and n, the number of strands.
Output: The starting set $S(A)$ and the finishing set $F(A)$ of A.

1. Set $S = \emptyset$ and $F = \emptyset$
2. Determine π and ψ, the permutations associated to A and rev A, respectively.
3. For $i = 1, \ldots, n-1$:
   (a) If $\pi(i) > \pi(i+1)$, set $S = S \cup \{i\}$.
   (b) If $\psi(i) > \psi(i+1)$, set $F = F \cup \{i\}$.
4. Return S and F.

The following algorithm checks whether a given factorisation is in left-canonical form:

Algorithm 2.2.14.
Input: $A_1 \dots A_r$ *such that each* $A_i \in [0,1]$.
Output: `true` *(if* $A_1 \dots A_r$ *is in left-canonical form) or* `false` *(if* $A_1 \dots A_r$ *is not in left-canonical form).*

1. Use Algorithm 2.2.13 to find $S(A_i)$ and $F(A_i)$ for each A_i.
2. For $i = 1, \dots, r-1$, do:
   (a) If $\exists j$ such that $j \in S(A_{i+1})$ and $j \notin F(A_i)$, then return `false`.
3. Return `true`.

We are now ready to find the left-canonical form for an arbitrary braid algebraically:

Algorithm 2.2.15.
Input: $W \in B_n$.
Output: $\Delta^p A_1 \dots A_r$ *in left-canonical form such that* $A_1 \neq \Delta$ *and* $A_r \neq 1$.

1. Write W such that $W \equiv W_1(T_1)^{-1}W_2(T_2)^{-1}\dots(T_q)^{-1}W_{q+1}$, where each $W_i \in B_n^+$ and $||W_i|| \geq 0$, and each T_i is a simple braid.
2. Rewrite each $(T_i)^{-1}$ as $\Delta^{-1}C_i$ where $C_i = \Delta(T_i)^{-1}$.
3. Use Corollary 2.1.7 to move all factors of Δ^{-1} to the start, and write this resulting word as $\Delta^{-q}P$ where $P \in B_n^+$.
4. Partition P into simple braid factors such that $P = A_1 \dots A_s$ (where each $A_i \in [0,1]$).
5. While Algorithm 2.2.14 applied to $A_1 \dots A_s$ returns `false`, do:
   (a) For $i = 1, \dots, s-1$, do:
      i. Use Algorithm 2.2.13 to find $S(A_{i+1})$ and $F(A_i)$.
      ii. If $\exists j$ such that $j \in S(A_{i+1})$ and $j \notin F(A_i)$, set $A_i = A_i\sigma_j$ and $A_{i+1} = \sigma_j^{-1}A_{i+1}$
6. In $A_1 \dots A_s$, count all the initial Δs (say there are t) and count all final factors of 1 (say there are k).
7. Return $\Delta^{-q+t}A_{t+1}\dots A_{s-k}$.

Note that any of the W_i in Step 1 can be 1 as needed. Furthermore, note that $A_i\sigma_j$ and $\sigma_j^{-1}A_{i+1}$ in Step 5(a)ii will still be in $[0,1]$.

I have not been able to find a time complexity estimate for this exact method for finding the left-canonical form. Thurston gives an equivalent left-canonical form (using his own definitions), and his algorithm for finding it has complexity $O(||W||^2 n \log n)$ where W is the word we are converting to left-canonical form. See Section 9.2 of [Eps92] for details.

Chapter 3

Using the normal forms

Garside uses the standard form to solve the conjugacy problem, and El-Rifai and Morton use the left-canonical form to improve upon Garside's solution to the conjugacy problem. We investigate both solutions in this chapter. We also have a brief historical interlude during which we consider a result that is a precursor to a key result of El-Rifai and Morton.

3.1 The summit set

We now continue with the relevant work from [Gar69]. We will see that two braids are conjugate[1] if and only if their summit sets are identical.

Recall that for a word W in standard form $\Delta^m \bar{A}$, the value m is called the **power** of W. The summit set $\mathrm{SS}(W)$ of W consists of all words which are conjugate to W and have maximal power such.

Garside defines the **index length** of a word as the "algebraic sum of its indices". He gives the following example: $\sigma_1^{-3}\sigma_3^4\sigma_2$ is of index length 2. Since $-3+4+1=2$, we can deduce that "index" refers to the exponents of these generators. Geometrically, this would correspond to the difference between the total number of positive crossings and the total number of negative crossings. El-Rifai and Morton define this operation at the start of [ERM94], but call it **weight**. They also specify that it is the abelianisation group homomorphism from B_n to $\mathbb{Z}$, which encapsulates much more information. We will use $\mathrm{wt}(A)$ to denote the weight of a braid $A \in B_n$ and call it the **weight map**. This map is well-defined because the relations in B_n are homogeneous[exa22]. For any positive word, the word-length and weight coincide, i.e., $||P|| = \mathrm{wt}(P)$ for all $P \in B_n^+$.

The following result later allows us to conclude that the summit set is finite:

Lemma 3.1.1 (Lemma 9 in [Gar69])**.** *In B_n, the number of words in standard form of weight t and power $\geq p$ is finite.*

[1] Garside introduces the notation that $A \sim B$ means A is conjugate to B. I have not encountered this $\sim$ notation elsewhere; most authors simply say "A is conjugate to B", which we will also do.

Proof. Let $\Delta^m \bar{A}$ be a word in standard form satisfying the conditions, i.e, $\text{wt}(\Delta^m \bar{A}) = t$ and $m \geq p$ (where t and p are fixed). Let $||\Delta|| = d$. Then $t = md + ||\bar{A}||$. Notice that $m = \frac{t-||\bar{A}||}{d} = \frac{t}{d} - \frac{||\bar{A}||}{d}$ and since $||\bar{A}|| \geq 0$ and $d > 0$, this means $m \leq \frac{t}{d}$. Thus, $p \leq m \leq \frac{t}{d}$. Since p, t, and d are fixed, this means that the number of possible values for m is finite. Furthermore, for any fixed m, the word-length $||\bar{A}||$ is fixed (due to t being fixed), and so the number of possible words $\bar{A}$ is also finite. Hence, the result follows. □

Recall the definitions associated to the diagram of a word, given on page 20. Garside introduces conjugation by very particular braids. If we consider the diagram $D(\Delta)$ in B_n, let α be any initial subroute so that $\Delta \cong \alpha X$ with $0 \leq ||X|| \leq ||\Delta||$. Then, such a subroute is called an **α-route**. In the language of El-Rifai and Morton's partial order, this can be read as $1 \leq \alpha \leq \Delta$, as the use of $\cong$ ensures $1 \leq \alpha$ and the factorisation ensures $\alpha \leq \Delta$. If we take any word W in B_n and consider the word $\alpha^{-1} W \alpha$, reduced to its standard form, then this is called an **α-transformation** of W. In other words, we are considering conjugation by simple braids, with the result written in standard form.

Garside also introduces conjugation by the base $\bar{\alpha}$ of α, but this notion is not necessary for the solution to the conjugacy problem. In fact, he states that it is clear that any α-transformation is equal to the corresponding $\bar{\alpha}$-transformation. This is of course due to the fact that α and $\bar{\alpha}$ represent the same braid, just written differently. The key thing to remember is that we are considering conjugation by an initial subroute of Δ, i.e., by a simple braid, with the result written in standard form.

The summit set of $X \in B_n$ (which we will denote by $\text{SS}(X)$) is found as follows. This algorithm computes the elements of $\text{SS}(X)$, and keeps track of the conjugators between X and the elements of $\text{SS}(X)$.

Algorithm 3.1.2.
Input: $X = \Delta^m \bar{A} \in B_n$ *in standard form.*
Output: $\text{SS}(X)$.

1. List all $n!$ α-routes, say $Q_1, \ldots, Q_{n!}$.
2. Set $\mathcal{V} = \{(X, 1)\}$ (where $1 \in B_n$).
3. For every $(U, C) \in \mathcal{V}$, do:
   (a) For every $Q \in \{Q_1, \ldots, Q_{n!}\}$, do:
      i. Compute $W = Q^{-1} U Q$ in standard form.
      ii. If (power of W) $<$ (power of U), then continue to next Q.
      iii. If (power of W) $>$ (power of U), then set $\mathcal{V} = \emptyset$.
      iv. If W is not in $\mathcal{V}$, set $\mathcal{V} = \mathcal{V} \cup \{(W, C \cdot Q)\}$.
4. Return $\mathcal{V}$.

In other words, we conjugate X with α-routes (simple elements) and discard any with lower power than X. Any new elements we find, we also conjugate with simple elements. If we encounter a higher power during this process, we discard all previous elements and search for conjugates with this higher power. The algorithm terminates when no new elements are found via conjugation with simple braids, i.e., when all elements in $\mathcal{V}$ have been checked..

Each element we find that is conjugate to X is of the same weight as X, and so by Lemma 3.1.1, the sequence must be finite. Thus, ultimately a stage must be reached when further applications of the process will yield no new words. The claim that each word in the sequence has the same weight as the original word, X, is not explained in the paper, but we can do so simply. We know all the words we find are conjugate to X by positive braids (the first few conjugates we find are conjugate by simple braids in particular, but as we conjugate those words with simple braids again, we build up longer conjugators between X and the new words, as concatenations of simple braids, which are just positive braids). Thus, every word we find, regardless of its power, can be written as $P^{-1}\,X\,P$ for some positive P, and so $\mathrm{wt}(P^{-1}\,X\,P) = \mathrm{wt}(P^{-1}) + \mathrm{wt}(X) + \mathrm{wt}(P) = -\,\mathrm{wt}(P) + \mathrm{wt}(X) + \mathrm{wt}(P) = \mathrm{wt}(X)$.

Any $V_i \in \mathrm{SS}(X)$ is said to be a **summit form** of X; some representative of the summit set of X, i.e., a summit form of X, is often denoted by $\tilde{X}$ in later works. The power of any summit form is called the **summit power** of X.

Now that we know what the summit set is, we can look at the lemmas which lead up to the core theorem that solves the conjugacy problem. Firstly, for any pair of conjugate braids, we can find a conjugating braid which is positive:

Lemma 3.1.3 (Lemma 11 in [Gar69])**.** *In B_n, if W is conjugate to V, then there exists a positive word X such that $X^{-1}\,W\,X = V$.*

This can be seen very easily, and I include the proof of this lemma, as it is crucial to the validity of the summit set algorithm:

Proof. Suppose $B^{-1}\,W\,B = V$, where $B \in B_n$. If $B = \Delta^m \bar{A}$ in standard form, then $B^{-1}\,W\,B = \bar{A}^{-1}\Delta^{-m}\,W\,\Delta^m\bar{A}$. If m is even, this becomes $\bar{A}^{-1}\,W\,\bar{A}$, and $\bar{A}$ is positive. If m is odd, we rewrite $B^{-1}\,W\,B$ as $\bar{A}^{-1}\Delta^{-1}(\Delta^{-m+1}\,W\,\Delta^{m-1})\Delta\bar{A}$, and this becomes $(\Delta\bar{A})^{-1}\,W\,\Delta\bar{A}$, where $\Delta\bar{A}$ is positive. Thus, the result follows. □

Furthermore,

Lemma 3.1.4 (Lemma 12 in [Gar69])**.** *In B_n, suppose that $W \equiv \Delta^p\bar{P}$ is a summit form of any given word A (so $W \in \mathrm{SS}(A)$), that X is any positive word such that $X^{-1}\,W\,X = \Delta^q\bar{Q}$, where $q \geq p$, and that $X = UY$ where U is an α-route (simple braid) of maximum length. Then $U^{-1}\,W\,U$, reduced to standard form, is also a summit form of A, i.e., $U^{-1}\,W\,U \in \mathrm{SS}(A)$*

Notice that if $X = UY$, where U is the longest possible simple braid that X could start with, then in El-Rifai and Morton's language, U would be the first factor in the left-canonical form of X. In fact, we will see a similar statement in Section 3.2, on El-Rifai and Morton's solution to the conjugacy problems.

Also, notice that this lemma does not tell us that $U^{-1} W U = Q$, but it does imply the **convexity property**[2] for the summit set:

Corollary 3.1.5. *Let W and Q be summit forms of some word A. Then there is a sequence $W = W_0, W_1, \ldots, W_k = Q$ of summit forms such that each element is conjugate to the next by an α-route (simple braid).*

Finally,

Theorem 3.1.6 (Theorem 10 in [Gar69])**.** *In B_n, A is conjugate to B if and only if their summit sets are identical.*

The crux of the proof of this theorem lies in showing that any summit form of B is also a summit form of A, and vice versa. Thus, if A and B are conjugate and $\tilde{A}$ and $\tilde{B}$ represent summit forms of A and B, respectively, then $\tilde{B} \in \mathrm{SS}(A)$ and $\tilde{A} \in \mathrm{SS}(B)$. The equality of their summits sets is due to the summit set only depending on the conjugacy class of the element we are considering, not on the element itself – it is the subset of the conjugacy class containing elements of maximal power.

The following algorithm solves the conjugacy search and decision problems. Note that this process is not explicitly listed by Garside, but this is the approach in later solutions, and it will work here too.

Algorithm 3.1.7.
Input: $X = \Delta^m \bar{A}, Y = \Delta^p \bar{B} \in B_n$, both in standard form.
Output: C such that $C^{-1} X C = Y$, or `fail` *if X and Y are not conjugate.*

1. Use Algorithm 3.1.7 to compute $\mathrm{SS}(X)$ along with the conjugators between X and the elements of $\mathrm{SS}(X)$.
2. Use Algorithm 3.1.7 to compute $\mathrm{SS}(Y)$ along with the conjugators between Y and the elements of $\mathrm{SS}(Y)$.
3. If (summit power of X) $\neq$ (summit power of Y), return `fail`.
4. If $\exists\ U$ such that $(U, A) \in \mathrm{SS}(X)$ and $(U, B) \in \mathrm{SS}(Y)$ (i.e., if $\mathrm{SS}(X) \cap \mathrm{SS}(Y) \neq \emptyset$), then return $A \cdot B^{-1}$.
5. Return `fail`.

If X and Y have the same summit set, we take an element in both, and since A conjugates X to that element, and B conjugates Y to that element, the conjugator from X to Y is $A \cdot B^{-1}$. If $\mathrm{SS}(X) \cap \mathrm{SS}(Y) = \emptyset$, then X and Y are not conjugate, in which case we reach Step 5. Of course, if X and Y do not even have the same summit power, we know they do not have the same summit set, and so we can return `fail` (Step 3).

[2] Garside does not refer to "convexity" in his paper, but later authors name this feature as such.

Notice that in generating the summit set, we have to conjugate with $n!$ (where n is the number of strands) simple braids over and over. The exact complexity of Garside's algorithm is not discussed in subsequent papers, but we can be certain there is a factor of $n!$ in the big-$\mathcal{O}$ notation, which of course gets extremely large as n increases.

3.1.1 Improvement from Joan Birman's seminal braid textbook

As part of her textbook, *Braids, Links, and Mapping Class Groups* [Bir75], which is the first textbook dedicated to braids, Joan Birman discusses the conjugacy problem and works through Garside's approach to solving it. She also offers a new theorem (and relevant lemmas), which I would like to mention for the sake of historical interest. Birman introduces the notion of a "cyclic diagram" and the "cyclic permutation" of a braid, which I view as a precursor to the cycling operation introduced by El-Rifai and Morton (we see this in the next section). The cycling operation of El-Rifai and Morton is widely used in subsequent solutions and inspired related operations which allowed for even more improvements.

The main theorem that Birman shows in her discussion on conjugacy is as follows:

Theorem 3.1.8 (Theorem 2.7 in [Bir82])**.** *Let $W = \Delta^m P$ be in standard form in B_n. Then W has summit power $> m$ if and only if its cyclic diagram contains Δ.*

Note that the theorem in the original textbook has a stronger statement but an errata [Bir82] was published to correct this to a slightly weaker version, which is stated above. Nonetheless, let's discuss the cyclic diagram.

Let $W = \Delta^m P$ be a braid. If $P \cong AB$, then the positive word BA will be called a **cyclic permutation** of P, and the positive word $B\tau(A)$ is a **reflected**[3] **cyclic permutation** of P. The **cyclic diagram** $\mathcal{C}(\Delta^m P)$ of $\Delta^m P$ is a list of positive words in B_n such that

(*i*) The list $\mathcal{C}(\Delta^m P)$ includes the word P,

(*ii*) For every word $P_j \in \mathcal{C}(\Delta^m P)$, the list also includes every word in $D(P_j)$, the diagram of P_j, and

(*iii*) For every word $P_j \in \mathcal{C}(\Delta^m P)$, the list also includes all cyclic permutations of P_j if m is even, or all reflected cyclic permutations of P_j if m is odd.

If one of the elements in the cyclic diagram of W contains a factor of Δ, we say, "the cyclic diagram contains Δ," as in the statement of the theorem.

Birman gives the example of $P = \sigma_1\sigma_3\sigma_1 \in B_4$. In terms of factorising P as AB, we have the following possibilities: $1 \cdot \sigma_1\sigma_3\sigma_1$, $\sigma_1 \cdot \sigma_3\sigma_1$, $\sigma_1\sigma_3 \cdot \sigma_1$, or $\sigma_1\sigma_3\sigma_1 \cdot 1$. Now, if m is even, it is quite easy to swap AB for BA in each of these options and get the set

$$\mathcal{C}(\Delta^m P) = \{\sigma_1\sigma_3\sigma_1,\, \sigma_3\sigma_1^2,\, \sigma_1^2\sigma_3\}.$$

[3] "reflected" likely comes from Garside's use of "reflection" for the map τ.

In this example, finding the cyclic permutations of P implicitly satisfies rules (i) and (ii) – clearly $P \in \mathcal{C}(\Delta^m P)$, and $D(\sigma_1\sigma_3\sigma_1) = D(\sigma_3\sigma_1^2) = D(\sigma_1^2\sigma_3) = \{\sigma_1\sigma_3\sigma_1, \sigma_3\sigma_1^2, \sigma_1^2\sigma_3\}$, and the cyclic permutations of each of these is already in our list.

If m is odd, more work is necessary to complete the set according to the three rules. Just finding $B\tau(A)$ for each of the factorisations of P we listed, gives us $\{\sigma_1\sigma_3\sigma_1, \sigma_3\sigma_1\sigma_3\}$, but then we still need to ensure requirement (ii) and (iii) hold. To do so, we add the elements of $D(\sigma_1\sigma_3\sigma_1)$ and $D(\sigma_3\sigma_1\sigma_3) = \{\sigma_3\sigma_1\sigma_3, \sigma_1\sigma_3^2, \sigma_3^2\sigma_1\}$ to the list, and then find the reflected cyclic permutation of any new elements too. In particular, the reflected cyclic permutation of $\sigma_3 \cdot \sigma_1^2$ and $\sigma_3^2 \cdot \sigma_1$ both give σ_1^3, and the reflected cyclic permutation of $\sigma_1^2 \cdot \sigma_3$ and $\sigma_1 \cdot \sigma_3^2$ both give σ_3^3. Thus, we have

$$\mathcal{C}(\Delta^m P) = \{\sigma_1\sigma_3\sigma_1,\ \sigma_3\sigma_1^2,\ \sigma_1^2\sigma_3,\ \sigma_3\sigma_1\sigma_3,\ \sigma_1\sigma_3^2,\ \sigma_3^2\sigma_1,\ \sigma_1^3,\ \sigma_3^3\}.$$

Note the following, which makes the later connection even clearer:

If m in $\Delta^m P$ is even, we swap AB to become BA. However, we could rewrite this as $B\tau^m(A)$, which is the exact same word, since m is even (an even number of applications of τ returns the original input). On the other hand, if m is odd, $B\tau(A)$ could be rewritten as $B\tau^m(A)$, since an odd number of applications of τ produces the same result as one application of τ.

Thus, both kinds of permutations of AB could be written as $B\tau^m(A)$. The cycling operation which we see in Section 3.2 looks very similar, the factorisation is just more particular.

This example we considered does not lead to a factor of Δ showing up in the cyclic diagram. However, if we have an element of the form $C_1\Delta C_2$ for some positive C_1, C_2 in the cyclic diagram of $W = \Delta^m P$, it would mean that there is an element conjugate to W such that $W = \Delta^m C_1\Delta C_2$, and thus $W = \Delta^m\Delta\tau(C_1)C_2 = \Delta^{m+1}\tau(C_1)C_2$, so we have found a conjugate of W with a higher power. Why is this element conjugate to W? Notice that $\tau^m(A)^{-1}\, W\, \tau^m(A) = \tau^m(A^{-1}) \cdot \Delta^m AB \cdot \tau^m(A) = \Delta^m A^{-1}AB\tau^m(A) = \Delta^m B\tau^m(A)$, and so finding the cyclic diagram finds elements which are conjugate to W.

3.2 The super summit set

We now discuss the rest of the relevant results from [ERM94].

The super summit set, which we discuss in this section, is a subset of the summit set; it is also a finite invariant of the conjugacy class, and two braids are conjugate if and only if their super summit sets are equal. Later authors denote the super summit set of a braid X by $\mathrm{SSS}(X)$. Besides this set being smaller than the summit set, the authors also circumvent the issue where we find elements only to later discard them. This can be used to improve the summit set algorithm too.

We already saw some of the necessary definitions as part of Section 2.2 on the left-canonical form. We now consider the other necessary definitions and statements.

Recall that $[p, s] \subset B_n$ is the subset $\{B \in B_n : \Delta^p \leq B \leq \Delta^s\}$. The authors note that there will clearly be a shortest braid interval $[p, s]$ containing a given braid. This is due to the

well-ordering principle. For $B \in B_n$, we define the **infimum** of B as

$$\inf B = \max\{p : \Delta^p \leq B\}$$

and the **supremum** of B as

$$\sup B = \min\{s : B \leq \Delta^s\}.$$

Furthermore, we call $\ell(B) = \sup B - \inf B$ the **canonical length** of B.

Notice that $\inf B$ is the same as the power of B in Garside's work, because $p = \inf B$ if and only if $B = \Delta^p B'$ with $B' \in B_n^+$ and $\Delta \not\leq B'$; this last condition is what Garside refers to as "B' is prime to Δ".

From the left-canonical form of a positive braid P as $P = A_1 A_2 \dots A_s$ (with $A_i \in [0,1], A_s \neq 1$, and $S(A_{i+1}) \subset F(A_i)$ for each i), we can find $\inf P$ immediately. This is because $P \geq \Delta$ if and only if $A_1 = \Delta$. Furthermore, if $A_i = \Delta$ in the left-canonical form, then $A_j = \Delta$ for all $j < i$. Thus, based on the left-canonical form of a positive braid,

$$\inf P = \max\{i : A_i = \Delta\}.$$

We also have the following theorem regarding sup for positive braids:

Theorem 3.2.1 (Theorem 2.11 in [ERM94])**.** *Let $P \in B_n^+$ have left-canonical form $P = A_1 A_2 \dots A_s$. Then $\sup P = s$.*

From this, we have that the canonical length $\ell(P) = \sup P - \inf P$ is the number of factors that remain once all Δs have been collected to the left in the left-canonical form of a positive braid.

For a general braid $B \geq \Delta^p$, Lemma 2.2.4 tells us that $B = \Delta^p B'$ with $B' \geq 1$ and if we write B' in left-canonical form and collect all Δs to the left, we get $B = \Delta^p \Delta^{\inf B'} A_1 \dots A_r$ where $A_1 \neq \Delta$. Clearly

$$\inf B = p + \inf B'$$

and

$$\ell(B) = \ell(B') = r.$$

Thus, using the expression for canonical length, we get

$$\begin{aligned} \sup B &= \ell(B) + \inf B \\ &= \ell(B') + p + \inf B' \\ &= \sup B' - \inf B' + p + \inf B' \\ &= p + \sup B' \end{aligned}$$

Hence, we have identified $\inf B$ and $\sup B$ for a general braid B in terms of the left-canonical form of a suitable positive braid.

Now we return to the authors' discussion of the solution to the conjugacy problem. Their conjugacy algorithm depends on listing all the conjugates of the given braid B in the interval

$[p, s]$ where p is maximal and s is minimal given p. The authors show that a process of conjugating with braids in $[0, 1]$ will eventually yield the complete list and terminate. The algorithm relies on the **convexity property** of Corollary 3.2.3, which is a consequence of the following theorem, which, in turn, is an adaptation of Lemmas 3.1.3 and 3.1.4.

Theorem 3.2.2 (Theorem 4.1 in [ERM94]). *Let $X, Y \geq \Delta^p$ be conjugate. We may suppose that $A^{-1}XA = Y$ for some $A \in B_n^+$. Then $A_1^{-1}XA_1 \geq \Delta^p$ where $A_1 \in [0, 1]$ is the first factor in the left-canonical form of A.*

Corollary 3.2.3 (Corollary 4.2 in [ERM94]). *Let $X, Y \in [p, s]$ be conjugate. Then there is a sequence $X = X_0, X_1, \ldots, X_k = Y$ of braids all in $[p, s]$, such that each element is conjugate to the next by an element of $[0, 1]$.*

Now we may define the super summit set, $\mathrm{SSS}(B)$ of a braid B. This set consists of the elements in the conjugacy class of B which have minimal canonical length (or equivalently, have maximal inf and minimal sup within these; see Corollary 3.2.5).

This maximal value of inf on the conjugacy class, which is the value of inf for all elements in $\mathrm{SSS}(B)$, is called the **summit infimum** of B and we denote it by $\inf_* B$. Similarly, the value of sup for all elements in $\mathrm{SSS}(B)$ is called the **summit supremum** of B and we denote it by $\sup_* B$. Lastly, the minimal canonical length on the conjugacy class of B, which is the canonical length of all the elements in $\mathrm{SSS}(B)$, is called the **summit (canonical) length**[4] of B, denoted by $\ell_*(B)$. These definitions are not given in El-Rifai and Morton's paper, but are given and used in later papers.

To find the set $\mathrm{SSS}(X)$, we conjugate repeatedly with elements of $[0, 1]$, and discard elements for which inf decreases or sup increases. To solve the conjugacy problem for arbitrary braids X and Y, we determine $\mathrm{SSS}(X)$, the super summit set of X, find one element, say $\tilde{Y}$ of $\mathrm{SSS}(Y)$, and check whether $\tilde{Y} \in \mathrm{SSS}(X)$. If so, X and Y are conjugate, otherwise they are not. However, it is possible to save time by checking if $\tilde{Y}$ appears as we construct $\mathrm{SSS}(X)$, instead of checking at the end. This is the approach taken in [GGM10b] (just with a different invariant subset of the conjugacy class).

This is not the final version of the algorithm, however. The authors further show that there is a relatively easy way to determine the summit infimum of a braid, and similarly to find the summit supremum. While these methods do not, unfortunately, generate the whole super summit set, it does help to make the algorithm more efficient, as we do not need to find and store elements only to discard them later. We only retain those which have the correct inf and sup. Not only can we use this method to find $\inf_*$, $\sup_*$, and ℓ_*, we can also use it to find at least one element we know is in the super summit set, giving us a starting point for finding the rest of the set

This is accomplished using cycling and decycling[5], which are special kinds of conjugation.

Cycling allows us to find $\inf_*$ of a braid – if any conjugate of a braid X has a higher infimum, then one such will be found by repeatedly cycling X. This higher value will be recognised as such when cycling starts to repeat conjugates that have already been found. Note, however,

[4] The word "canonical" is often left out when discussing the summit length.

[5] Decycling is called "reverse cycling" in El-Rifai and Morton's paper, but later authors call it decycling.

that at each stage of cycling, we need to ensure the new braid is in left-canonical form before checking inf and continuing, since the cycling operation is defined using the left-canonical form.

Let $X = \Delta^p P_1 P_2 \dots P_r$ where $p = \inf X$ and $P_1 \dots P_r$ is the left-canonical form of the positive braid $\Delta^{-p} X$. Then $P_1 \neq \Delta$ and we can form the conjugate

$$\mathbf{c}(X) = \Delta^p P_2 \dots P_r \tau^{-p}(P_1)$$

which is given by **cycling** X. El-Rifai and Morton[6] actually define cycling as $\mathbf{c}(X) = \Delta^p P_2 \dots P_r \tau^p(P_1)$, which is equivalent to what is given, in the case of braids. However, when we later consider the more general Garside group, we will see that we cannot interchange τ^p and τ^{-p} as we can for braids, so we preemptively use τ^{-p} for consistency. Notice the similarity to Birman's cyclic permutation!

We can see that

$$\begin{aligned}
\tau^{-p}(P_1)^{-1}\, X\, \tau^{-p}(P_1) &= \tau^{-p}(P_1^{-1}) \cdot \Delta^p P_1 P_2 \dots P_r \cdot \tau^{-p}(P_1) \\
&= \Delta^p P_1^{-1} \Delta^{-p} \cdot \Delta^p P_1 P_2 \dots P_r \cdot \tau^{-p}(P_1) \\
&= \Delta^p P_1^{-1} P_1 P_2 \dots P_r \tau^{-p}(P_1) \\
&= \Delta^p P_2 \dots P_r \tau^{-p}(P_1) \\
&= \mathbf{c}(X).
\end{aligned}$$

Since $P_1 \in [0, 1]$, cycling a braid conjugates it by the simple braid, $\tau^{-p}(P_1)$.

If any conjugate of X has a higher value of inf, then one such will be found by repeatedly cycling X.

Lemma 3.2.4 (Lemma 4.3 in [ERM94])**.** *Suppose that X is conjugate to Y with $\inf Y > \inf X$. Then repeated cycling will produce $\mathbf{c}^j(X)$ with $\inf \mathbf{c}^j(X) > \inf X$, for some j.*

Note that if $\inf X$ is already maximal, no such Y would exist. Also notice that cycling will not necessarily increase inf immediately. From this lemma, we have the corollary:

Corollary 3.2.5 (Corollary 4.4 in [ERM94])**.** *In every conjugacy class the maximum value of* inf *and the minimum value of* sup *can be achieved simultaneously. Thus the super summit set for a braid is the subset of its conjugacy class on which the canonical length ℓ is minimum.*

The authors state earlier in the paper that $\sup X = -\inf X^{-1}$. This is not proven, but we can see it easily. If $\Delta^p \leq X \leq \Delta^s$ for some integers p and s, then statement 2.2 implies $\Delta^{-s} \leq X^{-1} \leq \Delta^{-p}$, and so $\sup X = s = -\inf X^{-1}$. Thus, we can find the extreme values for inf and sup on the conjugacy class of X by cycling both X and X^{-1} until no new conjugates arise.

However, there is another way to determine $\sup_* X$ – repeated applications of **decycling**, which is denoted by $\mathbf{d}(X)$. Let $X = \Delta^p P_1 P_2 \dots P_r$ where $p = \inf X$ and $P_1, \dots P_r$ is the left-canonical form of the positive braid $\Delta^{-p} X$. Then

$$\mathbf{d}(X) = \Delta^p \tau^p(P_r) P_1 P_2 \dots P_{r-1} = P_r \Delta^p P_1 \dots P_{r-1}.$$

[6] In this paper cycling is denoted by $c(X)$, but we will use $\mathbf{c}$ to be consistent with later literature.

Clearly $P_r\, X\, P_r^{-1} = P_r \cdot \Delta^p P_1 \ldots P_{r-1} P_r \cdot P_r^{-1} = \mathbf{d}(X)$, and so decycling X is conjugation by P_r^{-1}.

Lemma 4.5 in [ERM94] shows that $\mathbf{d}(X) = \tau(\mathbf{c}(X^{-1}))^{-1}$, and so $\sup \mathbf{d}(X) = \sup \tau(\mathbf{c}(X^{-1}))^{-1} = -\inf \tau(\mathbf{c}(X^{-1}))$. From Proposition 2.2.7 we know that $\inf \tau(X) = \inf X$ for any $X \in B_n$, so we have $\sup \mathbf{d}(X) = -\inf \mathbf{c}(X^{-1})$. Hence, $\sup \mathbf{d}^j(X) = -\inf \mathbf{c}^j(X^{-1})$, and since we can find the largest value of inf on the conjugacy class of X^{-1} by considering only $\mathbf{c}^j(X^{-1})$, we can thus find the smallest value of sup among the conjugates of X by considering only successive decyclings $\mathbf{d}^j(X)$. Thus, we can identify at least one element of the super summit set of X by cycling and decycling, without calculating the whole super summit set of X. We cycle until we find the maximum value of inf (identified once cycling produces a conjugate already found as a previous cycling), and then decycle this result until we get the minimum value of sup (identified when decycling produces repeats). See Section A.1 for two examples illustrating that cycling indeed increases inf and decycling indeed decreases sup.

Although it is not stated as such in this paper, it is clear that if $Y \in \mathrm{SSS}(X)$ for some $X \in B_n$, then we also have $\mathbf{c}(Y) \in \mathrm{SSS}(X)$ and $\mathbf{d}(Y) \in \mathrm{SSS}(X)$, since cycling can only increase inf or stay the same, and inf is already maximal in $\mathrm{SSS}(X)$; similarly for decycling and decreasing sup. It is possible in some cases for cycling to also decrease sup and decycling to increase inf, but that will not happen either when we are in $\mathrm{SSS}(x)$. Once we are inside $\mathrm{SSS}(X)$, there is no possible conjugation that could decrease the summit canonical length.

A further result not stated in this paper, but which is of use later, is the following:

Theorem 3.2.6 (Derived from Theorem 1.13(d) in [Geb05])**.** *For all $X \in B_n$, $\tau(\boldsymbol{c}(X)) = \boldsymbol{c}(\tau(X))$ and $\tau(\boldsymbol{d}(X)) = \boldsymbol{d}(\tau(X))$.*

I could not find a proof of this in the references given with this statement, so I have written a quick proof in Section B.1.

For the sake of consistency, and to make the process crystal clear, I have written out a number of algorithms not given in this paper. I refer back to these in some of the later algorithms I state, both written by myself and the later authors (in which case I have changed some of their steps to refer to a specific algorithm).

The following algorithm returns $\mathbf{c}(X)$ of X and the conjugating element from X to $\mathbf{c}(X)$:

Algorithm 3.2.7.
Input: $X \in B_n$.
Output: $\boldsymbol{c}(X)$ *and* $C \in B_n$ *such that* $C^{-1}\, X\, C = \boldsymbol{c}(X)$.

1. Compute the left-canonical form of X as $\Delta^p X_1 \ldots X_r$.
2. If $X = \Delta^p$, then return Δ^p and 1.
3. Compute $C = \tau^{-p}(X_1)$
4. Compute the left-canonical form of $C^{-1}\, X\, C$ and store it as Y.
5. Return Y and C.

The next algorithm returns $\mathbf{d}(X)$ of X and the conjugating element from X to $\mathbf{d}(X)$:

Algorithm 3.2.8.
Input: $X \in B_n$.
Output: $\boldsymbol{d}(X)$ *and* $C \in B_n$ *such that* $C^{-1} X C = \boldsymbol{d}(X)$.

1. Compute the left-canonical form of X as $\Delta^p X_1 \dots X_r$.
2. If $X = \Delta^p$, then return Δ^p and 1.
3. Compute $C = X_r^{-1}$
4. Compute the left-canonical form of $C^{-1} X C$ and store it as Y.
5. Return Y and C.

El-Rifai and Morton do not define what cycling and decycling do to a braid which is a power of Δ; it is stated in [BGGM07a] that $\mathbf{c}(X) = X = \mathbf{d}(X)$ when $\ell(X) = 0$.

A few years after El-Rifai and Morton's paper, Joan Birman, Ki Hyoung Ko, and Sang Jin Lee showed the following result in [BKL01], which gives us an upper bound for how many times we would have to cycle (respecitvely, decycle) before inf increases (resp. sup decreases).

Theorem 3.2.9 (Theorem 1 in [BKL01])**.** *Let* $W \in B_n$. *If* $\inf W$ *is not maximal for the conjugacy class of* W, *then* $\inf \boldsymbol{c}^{||\Delta||-1}(W) > \inf W$. *If* $\sup W$ *is not minimal for the conjugacy class of* W, *then* $\sup \boldsymbol{d}^{||\Delta||-1}(W) < \sup W$.

Thus, if we cycle (respectively decycle) $||\Delta|| - 1$ times without any change to inf (resp. sup), we know we are already in the super summit set. On the other hand, each time inf increases (resp. sup decreases), we must restart our count and continue cycling (resp. decyling) to see if we encounter another increase (resp. decrease) before $||\Delta|| - 1$ cyclings (resp. decyclings) occur. As established in [ERM94], the other way to know we have reached maximal inf (resp. minimal sup) is when we encounter a repeated element under cycling (resp. decycling).

The following algorithm allows us to find a representative $\tilde{X}$ of the summit set of X using cycling, which thus finds the maximum value of inf on the conjugacy class. It also keeps track of the conjugators and returns the conjugating element from X to $\tilde{X} \in \mathrm{SS}(X)$.

Algorithm 3.2.10.
Input: $X \in B_n$.
Output: $\tilde{X} \in \mathrm{SS}(X)$ *and* $C \in B_n$ *such that* $C^{-1} X C = \tilde{X}$.

1. Set $\tilde{X} = X$, $C = 1 \in B_n$, and $\texttt{max} = \frac{n(n-1)}{2} - 1$.
2. Loop:

(a) Set $\mathcal{T} = \emptyset$, $\texttt{inf} = \inf \tilde{X}$, and $\texttt{count} = 0$.

(b) While $\inf \tilde{X} = \texttt{inf}$, do:

i. If $\tilde{X} \notin \mathcal{T}$ and $\texttt{count} < \texttt{max}$, then:

A. Set $\mathcal{T} = \mathcal{T} \cup \{\tilde{X}\}$.

B. Use Algorithm 3.2.7 to compute $Z = \mathbf{c}(\tilde{X})$ and A such that $A^{-1}\,\tilde{X}\,A = Z$.

C. Set $\tilde{X} = Z$, $C = C \cdot A$, and $\texttt{counter} = \texttt{counter} + 1$.

ii. Else:

A. Return $\tilde{X}$ and C.

Notice that each time the infimum increases, we exit the while-loop and update the current value of inf, reset the counter, and reset the set, $\mathcal{T}$, which keeps track of which elements we have found for this particular value of inf. Once we do reach the maximal value of inf, we cycle until we encounter a repeated element under cycling, or have cycled $||\Delta|| - 1$ many times, and so we know that we are at the maximal value of inf, and can output the relevant braids.

The next algorithm finds a representative of the super summit set using the previous algorithm and decycling, thus finding the minimum canonical length on the conjugacy class of X. It is very similar to the previous algorithm, and also uses Theorem 3.2.9 to avoid decycling for longer than necessary.

Algorithm 3.2.11.
Input: $X \in B_n$.
Output: $\tilde{X} \in \mathrm{SSS}(X)$ *and* $C \in B_n$ *such that* $C^{-1}\,X\,C = \tilde{X}$.

1. Use Algorithm 3.2.10 to find $\tilde{X} \in \mathrm{SS}(X)$ and B such that $B^{-1}\,X\,B = \tilde{X}$
2. Set $C = 1 \in B_n$, and $\texttt{max} = \frac{n(n-1)}{2} - 1$.
3. Loop:

(a) Set $\mathcal{T} = \emptyset$, $\texttt{sup} = \sup \tilde{X}$, and $\texttt{count} = 0$.

(b) While $\sup \tilde{X} = \texttt{sup}$, do:

i. If $\tilde{X} \notin \mathcal{T}$ and $\texttt{count} < \texttt{max}$, then:

A. Set $\mathcal{T} = \mathcal{T} \cup \{\tilde{X}\}$.

B. Use Algorithm 3.2.8 to compute $Z = \mathbf{d}(\tilde{X})$ and A such that $A^{-1}\,\tilde{X}\,A = Z$.

C. Set $\tilde{X} = Z$, $C = C \cdot A$, and $\texttt{counter} = \texttt{counter} + 1$.

ii. Else:

A. Return $\tilde{X}$ and $B \cdot C$.

Putting these together, we have the following algorithm to compute the entire super summit set of an element. The structure of this algorithm is based on Algorithm 4.3.14 in [FGM03].

Algorithm 3.2.12.
Input: $X \in B_n$.
Output: $\mathrm{SSS}(X)$.

1. Use Algorithm 3.2.11 to find $\tilde{X} \in \mathrm{SSS}(X)$.
2. Set $\ell = \ell(\tilde{X})$, $\mathcal{V} = \{\tilde{X}\}$, and $\mathcal{W} = \emptyset$.
3. List all $n!$ simple braids, say $Q_1, \ldots, Q_{n!}$.
4. While $\mathcal{V} \neq \mathcal{W}$, do:
   (a) Take $V \in \mathcal{V} \backslash \mathcal{W}$.
   (b) For every $Q \in \{Q_1, \ldots, Q_{n!}\}$, do:
      i. Set $W = Q^{-1} V Q$ and compute the left-canonical form of W.
      ii. If $\ell(W) = \ell$ and $W \notin \mathcal{V}$, then set $\mathcal{V} = \mathcal{V} \cup \{W\}$.
   (c) Set $\mathcal{W} = \mathcal{W} \cup \{V\}$.
5. Return $\mathcal{V}$.

Step 3 is easy to do, as we know how to find the simple braid associated to a permutation, and there are many programs available to find all possible permutations on n elements. The set $\mathcal{V}$ tracks all the elements we find, and $\mathcal{W}$ tracks which elements have already been conjugated by all simple elements. Since the super summit set is finite, we will reach a point where no new conjugates are found, and thus exit the while-loop.

We could use this algorithm to solve the conjugacy problem for X and Y, by finding the entire $\mathrm{SSS}(X)$ and then looking for the representative $\tilde{Y}$ of $\mathrm{SSS}(Y)$ in $\mathrm{SSS}(X)$, but there is a more efficient way. Along the same logic as Algorithm 6.27 in [GGM10b], we can find $\tilde{Y}$ and then look for at it as we build $\mathrm{SSS}(X)$. Thus, only if X and Y are not conjugate do we need to construct the whole of $\mathrm{SSS}(X)$ (or if we are unlucky and $\tilde{Y}$ is the very last element of $\mathrm{SSS}(X)$ that we find). The relevant algorithm follows below. We denote the conjugating element between $\tilde{X}$ and another element of $\mathrm{SSS}(X)$, say R, as C_R.

Algorithm 3.2.13.
Input: $X, Y \in B_n$
Output: $C \in B_n$ *such that* $C^{-1} X C = Y$ *or* `fail` *if* X *and* Y *are not conjugate.*

1. Use Algorithm 3.2.11 to compute $\tilde{X} \in \mathrm{SSS}(X)$ and A such that $A^{-1} X A = \tilde{X}$, and $\tilde{Y} \in \mathrm{SSS}(Y)$ and B such that $B^{-1} Y B = \tilde{Y}$.

2. If $\ell(\tilde{X}) \neq \ell(\tilde{Y})$, then return `fail`.

3. List all $n!$ simple braids, say $Q_1, \ldots, Q_{n!}$.

4. Set $\ell = \ell(\tilde{X})$, $\mathcal{V} = \{\tilde{X}\}$, $\mathcal{W} = \emptyset$, and $C_{\tilde{X}} = 1$.

5. While $\mathcal{V} \neq \mathcal{W}$, do:

   (a) Take $V \in \mathcal{V} \backslash \mathcal{W}$.

   (b) For every $Q \in \{Q_1, \ldots, Q_{n!}\}$, do:

      i. Set $W = Q^{-1} V Q$ and compute the left-canonical form of W.

      ii. If $W = \tilde{Y}$, then set $C_{\tilde{Y}} = C_V \cdot Q$. Return $A \cdot C_{\tilde{Y}} \cdot B^{-1}$.

      iii. If $\ell(W) = \ell$ and $W \notin \mathcal{V}$, then set $C_W = C_V \cdot Q$ and $\mathcal{V} = \mathcal{V} \cup \{W\}$.

   (c) Set $\mathcal{W} = \mathcal{W} \cup \{V\}$.

6. Return `fail`.

If $\tilde{X}$ and $\tilde{Y}$ do not have the same canonical length, they cannot have the same super summit sets, and so we can stop (Step 2). If we find $\tilde{Y}$ in $\mathrm{SSS}(X)$ in Step 5(b)ii, we return $A \cdot C_{\tilde{Y}} \cdot B^{-1}$ as the conjugator between X and Y, since A conjugates X to $\tilde{X}$, $C_{\tilde{Y}}$ conjuates $\tilde{X}$ to $\tilde{Y}$, and B^{-1} conjugates $\tilde{Y}$ to Y (since B conjugates Y to $\tilde{Y}$). We only reach Step 6 if $\tilde{Y}$ is not a member of $\mathrm{SSS}(X)$, which we construct as part of the while-loop.

Note that Algorithms 3.2.12 and 3.2.13 are not as efficient as they could be. In particular, it would be quicker to work with permutations instead of simple braids. However, in the next chapter, we consider a more general group (the Garside group), and these algorithms translate trivially to this more general group as it stands. They would not translate trivially if we write this using permutations. In [FGM03], it is stated that finding $\mathrm{SSS}(X)$ for $X \in B_n$, using El-Rifai and Morton's method (presumably using permutations in stead of braids), runs in $\mathcal{O}(|\mathrm{SSS}(X)| \cdot ||X||^2 (n!) n \log n)$. Solving the conjugacy problem for X and Y will have a similar time complexity.

Chapter 4

A different partial order, with stronger structure

In this chapter we start by investigating the partial orders that William P. Thurston suggests in Chapter 9 of *Word processing in groups* [Eps92]. After this, we look at the generalisation of braid groups, called Garside groups, since the rest of the solutions to the conjugacy problem are given for the problem in Garside groups. With this out of the way, we look at an improved version of the super summit set solution which utilises the partial orders introduced by Thurston.

4.1 Thurston's partial order on braids

Thurston defines two partial orders on B_n^+. Suppose $AB = C$ for positive braids $A, B, C \in B_n^+$. Then we write $A \preccurlyeq C$ and call A a **prefix** of C (Thurston calls it a head), and we write $C \succcurlyeq B$ and call B a **suffix** of C (called a tail by Thurston). We can think of a prefix of a positive braid as a left factor (i.e. the left-hand factor in a factorisation) of said braid, while a suffix is a right factor. Some authors (including Thurston) use $\prec$ and $\succ$ instead of $\preccurlyeq$ and $\succcurlyeq$. We will use $\prec$ and $\succ$ to indicate that the two elements being compared are not equal. $\preccurlyeq$ and $\succcurlyeq$ has the option of equality.

Notice that for arbitrary $X, Y \in B_n^+$, $X \preccurlyeq Y$ is not equivalent to $Y \succcurlyeq X$. The former implies that $XZ_1 = Y$ (for $Z_1 \in B_n^+$) while the latter implies that $Z_2X = Y$ (for $Z_2 \in B_n^+$). Since arbitrary braids do not commute, these are not equivalent. Recall that El-Rifai and Morton use $A \leq B$ and $B \geq A$ interchangeably, a clear difference between the partial orders.

Thurston remarks that any prefix or suffix of a simple braid is also a simple braid, since if a simple braid is the concatenation of two positive subbraids, the two subbraids must also be simple. I would like to highlight parts of two results[1] that Thurston shows (the rest of these results are not new to us at this stage). Notice the similarities that arise between the partial order $\leq$ from [ERM94] and these two partial orders $\preccurlyeq$ and $\succcurlyeq$, when Δ is on either side of the order symbol:

[1] Regarding notation: Thurston denotes Δ by Ω, S_n by D, and B_n^+ by P.

Lemma 4.1.1 (Last statement of Lemma 9.1.14 in [Eps92]). *A positive braid A is in S_n if and only if $\Delta \succcurlyeq A$ or, equivalently, $A \preccurlyeq \Delta$ in B_n^+.*

We saw that simple braids $A \in S_n$ satisfy $1 \leq A \leq \Delta$, which means $\Delta = AC_1 = C_2A$ where $C_1, C_2 \in B_n^+$. Notice that $\Delta = AC_1$ means $A \preccurlyeq \Delta$ and $\Delta = C_2A$ means $\Delta \succcurlyeq A$.

Lemma 4.1.2 (Last statement of Lemma 9.1.16 in [Eps92]). *If Δ is a factor of $C \in B_n^+$, i.e., $C = A\Delta B$ with $A, B \in B_n^+$, then $C \succcurlyeq \Delta$ and $\Delta \preccurlyeq C$.*

By the definition of $\leq$, $C = A\Delta B$ (where $A, B \in B_n^+$) would mean that $\Delta \leq C$, which in turn is shown to imply that $C = E_1\Delta = \Delta E_2$ for $E_1, E_2 \in B_n^+$ (Lemma 2.2.4). However, $C = E_1\Delta$ means $C \succcurlyeq \Delta$ and $C = \Delta E_2$ means $\Delta \preccurlyeq C$.

Suppose $A \preccurlyeq B$. We say A is maximal w.r.t. $\preccurlyeq$ is any other element X satisfying $X \preccurlyeq B$ also satisfies $X \preccurlyeq A$.

The factorisation $W = W_1W_2 \dots W_r$ for $W \in B_n^+$ is in right-canonical form if each W_i is in S_n and W_i is the maximal element (w.r.t. $\succcurlyeq$) of S_n satisfying $W_1W_2 \dots W_i \succcurlyeq W_i$. Thus, any other simple element X satisfying $W_1W_2 \dots W_i \succcurlyeq X$ also satisfies $W_i \succcurlyeq X$. The equivalent definition for the left-canonical form is not given in the text, but it is easy to translate it to the left: the factorisation $V = V_1V_2 \dots V_r$ for $V \in B_n^+$ is in left-canonical form if each V_j is in S_n and V_j is the maximal element (w.r.t $\preccurlyeq$) of S_n satisfying $V_j \preccurlyeq V_jV_{j+1} \dots V_r$. Thus, any other simple element Y satisfying $Y \preccurlyeq V_jV_{j+1} \dots V_r$ also satisfies $Y \preccurlyeq V_j$.

The partial orders $\preccurlyeq$ and $\succcurlyeq$ can be extended to the whole braid group. Thurston only states this for $\succcurlyeq$, but later papers (like [FGM03],[BGGM07a], and [GGM10b]) discuss this extension for $\preccurlyeq$ too.

For $Q, R, S, T \in B_n$, we say that $Q \preccurlyeq R$ if $QC = R$ for some $C \in B_n^+$ or, equivalently, if $Q^{-1}R \in B_n^+$. We say that $T \succcurlyeq S$ if $DS = T$ for some $D \in B_n^+$ or, equivalently, if $TS^{-1} \in B_n^+$. Note that the terms "prefix" and "suffix" specifically refer to positive braids, so we do not use those terms when discussing general braids. In the case of general braids, we can say that $Q \preccurlyeq R$ means that Q is a left factor of R such that the remaining factor is positive, and similarly $T \succcurlyeq S$ means that S is a right factor of T such that the remaining factor is positive.

$\preccurlyeq$ is invariant under left multiplication – if $Q \preccurlyeq R$ and $X \in B_n$ is another braid, then $XQ \preccurlyeq XR$ too. Similarly, $\succcurlyeq$ is invariant under right multiplication, so if $T \succcurlyeq S$ and $Y \in B_n$ is another braid, then $TY \succcurlyeq SY$. Both of these statements are very easy to verify using the definitions of $\preccurlyeq$ and $\succcurlyeq$ on B_n.

Thurston shows that the braid group is a lattice with respect to $\succcurlyeq$, i.e., for any two braids, we can find a unique braid that is smaller (w.r.t. $\succcurlyeq$) than both and the greatest such, as well as a unique braid that is greater (w.r.t. $\succcurlyeq$) than both and the smallest such. The former, which we will call the **right greatest common divisor** or **right gcd** is denoted by $A \wedge^{\urcorner} B$ for two braids A and B, and satisfies $A \succcurlyeq A \wedge^{\urcorner} B$ and $B \succcurlyeq A \wedge^{\urcorner} B$ while being the greatest braid such (if any other element, say P, satisfies $A \succcurlyeq P$ and $B \succcurlyeq P$, then $A \wedge^{\urcorner} B \succcurlyeq P$). The latter, which we will call the **right least common multiple** or **right lcm** is denoted by $A \vee^{\urcorner} B$ for two braids A and B, and satisfies $A \vee^{\urcorner} B \succcurlyeq A$ and $A \vee^{\urcorner} B \succcurlyeq B$ while being the smallest braid such (any other element, say M, satisfying $M \succcurlyeq A$ and $M \succcurlyeq B$ will also satisfy $M \succcurlyeq A \vee^{\urcorner} B$).

The braid group is also a lattice with respect to $\preccurlyeq$. This is not shown in Thurston's chapter, but is discussed in later papers. The **left greatest common divisor** or **left gcd** is denoted by $C \wedge D$ for two braids C and D, and satisfies $C \wedge D \preccurlyeq C$ and $C \wedge D \preccurlyeq D$ while being the greatest (w.r.t $\preccurlyeq$) braid such. The **left least common multiple** or **left lcm** is denoted by $C \vee D$ for two braids C and D, and satisfies $C \preccurlyeq C \vee D$ and $D \preccurlyeq C \vee D$ while being the smallest (w.r.t. $\preccurlyeq$) braid such.

Thurston mainly works with $\succcurlyeq$, but we see $\preccurlyeq$ and the left gcd and left lcm much more often in later papers, which is why I use $\wedge$ and $\vee$ for the "usual" (left) gcd and lcm, but indicate the "unusual" right gcd and right lcm by $\wedge^{\text{r}}$ and $\vee^{\text{r}}$.

Thurston gives a method for finding the right lcm and right gcd, and we can use his method to construct a method to find the left lcm and left gcd. First, we need another definition:

A braid word is in **mixed canonical form** if it is a braid word of minimal word-length of the form $U^{-1}V$ where U and V are both positive and in left-canonical form.

After defining the mixed canonical form, Thurston gives the following lemma:

Lemma 4.1.3 (Lemma 9.3.5 in [Eps92])**.** *Every element of B_n has exactly one representative satisfying the defining properties of mixed canonical form. If $U = U_1 \dots U_m$ and $V = V_1 \dots V_k$ are braids in left-canonical form, then $U^{-1}V$ is in mixed canonical form if and only if*

$$\operatorname{rev} U_1 \wedge^{\text{r}} \operatorname{rev} V_1 = 1.$$

In other words, the greatest (and only) suffix that both $\operatorname{rev} U_1$ and $\operatorname{rev} V_1$ have in common is 1, the identity braid, so they have no non-trivial right factor in common. If $\operatorname{rev} U_1$ and $\operatorname{rev} V_1$ did indeed have a non-trivial right factor in common, say $Y \in B_n^+$, then $\operatorname{rev} U_1 = U_Y Y$ and $\operatorname{rev} V_1 = V_Y Y$ where $U_Y, V_Y \in B_n^+$. Thus, $U_1 = Y U_Y$ and $V_1 = Y V_Y$. Then where U^{-1} and V meet in the mixed canonical form, we would encounter $U_m^{-1} \dots U_2^{-1} U_Y^{-1} Y^{-1} \cdot Y V_Y V_2 \dots V_k$, and so Y^{-1} and Y would cancel, which contradicts the requirement that $U^{-1}V$ is of minimal word-length. It is worth noting that we could equivalently state this Lemma with $U_1 \wedge V_1 = 1$ [exa22], but Thurston uses $\operatorname{rev} U_1 \wedge^{\text{r}} \operatorname{rev} V_1 = 1$ (as is consistent with his use of $\succcurlyeq$), and so this is what we discuss under this Lemma.

As part of the proof of Lemma 4.1.3, when showing the "existence" aspect of the lemma, Thurston says that any element of B_n can be expressed as a product $U^{-1}V$ with U and V positive and in left-canonical form – for example, we can let U be a power of Δ. We just have to choose such an expression having minimal length. It is not made clear how exactly to ensure the expression we have chosen is indeed of minimal length.

When Thurston says that we can let U be a power of Δ, this is just for the sake of proving existence, as we know we can write any braid in left-canonical form, which starts with an integer power of Δ. However, it is not my belief that we *have* to use powers of Δ to put a purely positive or purely negative braid into mixed canonical form; we can let $U = 1$ or $V = 1$, the identity braid. In fact, if we do insist that neither U or V can be the identity, the method for finding the lcm and gcd is not correct.

Thus, if we want to write a positive braid, P, in mixed canonical form, we put it in left-canonical form, say $\Delta^p P_1 \dots P_r$, and concatenate a factor of 1^{-1} at the start. Thus U will be 1 and $V = \Delta^p P_1 \dots P_r$, which are both in left-canonical form, to get the form $P = U^{-1}V$.

To write a purely negative braid, N, in mixed canonical form, we take its inverse, which is positive, write that in left-canonical form, and invert it back to being negative. After this, we concatenate 1 to the end, to get the mixed canonical form $N1 = U^{-1}V$, so $U = N^{-1}$ and $V = 1$.

The tricky case is, of course, general braids consisting of both positive and negative crossings. Sure, we can write it in left-canonical form, which will have a power (which may be negative, but not necessarily) of Δ at the start, followed by a positive braid in its left-weighted factorisation, but that would not necessarily give the smallest word-length. For example, $\sigma_1^{-1}\sigma_2\sigma_1\sigma_3 \in B_4$ is already in mixed canonical form, with word-length 4. In left-canonical form, it would be $\Delta^{-1}\sigma_1\sigma_2\sigma_3\sigma_1\sigma_2 \cdot \sigma_2\sigma_1\sigma_3 \in B_4$ which has word-length 14. Thus, to put this in mixed canonical form, we consider the left-canonical form, and cancel crossings, until all the negative crossings are together at the start, and all the positive crossings are together at the end, and no more crossings can be cancelled [exa22]. It is an easy exercise to see that the left-canonical form of this particular braid indeed leads to the mixed canonical form upon cancelling as many crossings as possible. That being said, it is often possible to find the mixed canonical form without first finding the left-canonical form, simply by using the braid relations to move the crossings as desired, or considering the geometric representation. See Section A.2 for such examples.

We need one more definition before seeing how to find the gcd and lcm of two braids. The **negation** B^- of a braid B is what we obtain if we replace every negative generator with its positive counterpart, and vice versa. For example, $(\sigma_1\sigma_2^{-1})^- = \sigma_1^{-1}\sigma_2$. We could also think of it as applying rev to the inverse of the braid in question, i.e. $B^- = \operatorname{rev} B^{-1}$.

To find the **right least common multiple** of braids A_1 and A_2, we start by finding the mixed canonical form $U_1^{-1}U_2$ of the braid $A_1A_2^{-1}$ and then, $A_1 \vee^{\curvearrowleft} A_2 = U_2A_2 = U_1A_1$.

To find the **right greatest common divisor** of braids A_1 and A_2, we start by finding the mixed canonical form $U_1^{-1}U_2$ of the braid $(A_1A_2^{-1})^-$ and then $A_1 \wedge^{\curvearrowleft} A_2 = U_1^-A_1 = U_2^-A_2$.

For a full proof of why this is valid, see Corollary 9.3.7 of [Eps92].

Due to Lemma 4.2.2 in Section 4.2, we know that $(A_1 \wedge A_2)^{-1} = A_1^{-1} \vee^{\curvearrowleft} A_2^{-1}$ and $(A_1 \vee A_2)^{-1} = A_1^{-1} \wedge^{\curvearrowleft} A_2^{-1}$. Thus, we can use what we know of finding $\vee^{\curvearrowleft}$ and $\wedge^{\curvearrowleft}$ to find $\wedge$ and $\vee$ of two braids: in the methods above, replace A_1 and A_2 with A_1^{-1} and A_2^{-1}, respectively, and invert the results.

To find the **left least common multiple** of braids A_1 and A_2, we start by finding the mixed canonical form $U_1^{-1}U_2$ of the braid $(A_1^{-1}A_2)^-$ and then $A_1 \vee A_2 = A_1(U_1^-)^{-1} = A_2(U_2^-)^{-1}$.

To find the **left greatest common divisor** of braids A_1 and A_2, we start by finding the mixed canonical form $U_1^{-1}U_2$ of the braid $A_1^{-1}A_2$ and then $A_1 \wedge A_2 = A_1U_1^{-1} = A_2U_2^{-1}$.

The operations $\wedge, \vee, \wedge^{\curvearrowleft}$, and $\vee^{\curvearrowleft}$ are commutative.

Examples of all four of these operations are presented in Section A.2.

4.2 Garside groups

We now have all the knowledge we need to discuss a more general class of groups, of which the braid group is an example, called Garside groups. The next solutions to the conjugacy problem that we investigate are all given in terms of Garside groups, and thus also hold for braid groups. The notion of a Garside group is introduced by Patrick Dehornoy and Luis Paris in *Gaussian groups and Garside groups, two generalisations of Artin groups* [DP99] and Matthieu Picantin shows that the conjugacy solutions provided thusfar hold for these general groups in [Pic01]. For our summary of Garside groups, we use the discussion in the papers regarding solutions to the conjugacy problem, in particular [FGM03], [Geb05], [BGGM07a], and [GGM10b]. As we work through this summary, keep the braid group in mind, and check how these descriptions hold for what we know of braids. Additional definitions of algebraic terminology is added in the footnotes as needed. Thusfar we have denoted braids by upper case Latin letters; now that we generalise, we use lower case Latin letters.

Let M be a monoid[2]. We call $a \in M$ an **atom** if $a \neq 1$ and if $a = xy$ implies $x = 1$ or $y = 1$, i.e. a cannot be factorised non-trivially. M is an **atomic monoid** if it is generated by its atoms and for every $x \in M$, there exists an integer $N_x > 0$ such that x cannot be written as a product of more than N_x atoms (i.e. there is an upper bound on the number of atoms used to write an arbitrary element of M as a product of atoms).

We call a monoid M a **Gaussian monoid** if it is atomic, (left and right) cancellative[3], and if every pair of elements in M admits a left- and right least common multiple (lcm) and a left- and right greatest common divisor (gcd).

When we have a cancellative monoid M, with no non-trivial invertible elements, we can define two different partial orders on its elements. Given $a, b \in M$, we write $a \preccurlyeq b$ if there exists $p \in M$ such that $ap = b$, and we say a is a **prefix** or **left divisor** of b. On the other hand, we write $b \succcurlyeq a$ is there exists $q \in M$ such that $qa = b$, and we say a is a **suffix** or **right divisor** of b.

Let $a, b \in M$, with M a Gaussian monoid. We denote the **left lcm** of a and b by $a \vee b \in M$. This is the unique minimal element with respect to $\preccurlyeq$ such that both $a \preccurlyeq a \vee b$ and $b \preccurlyeq a \vee b$. We denote the **left gcd** of a and b by $a \wedge b \in M$. This is the unique maximal element with respect to $\preccurlyeq$ such that both $a \wedge b \preccurlyeq a$ and $a \wedge b \preccurlyeq b$.

Similarly, the **right lcm** of a and b, denoted by $a \vee^{\curvearrowleft} b$ is the unique minimal element with respect to $\succcurlyeq$ such that both $a \vee^{\curvearrowleft} b \succcurlyeq a$ and $a \vee^{\curvearrowleft} b \succcurlyeq b$. The **right gcd** of a and b, denoted $a \wedge^{\curvearrowleft} b$ is the unique maximal element with respect to $\succcurlyeq$ such that both $a \succcurlyeq a \wedge^{\curvearrowleft} b$ and $b \succcurlyeq a \wedge^{\curvearrowleft} b$.

We can now define a **Garside monoid**, which is a Gaussian monoid that has a Garside element. A **Garside element**, denoted $\Delta \in M$, is defined by its divisors: its left divisors (i.e. the set $\{s \in M \mid s \preccurlyeq \Delta\}$) must coincide with its right divisors (the set $\{t \in M \mid \Delta \succcurlyeq t\}$), these divisors must form a finite set, and the divisors must generate M. We call these divisors

[2] A set equipped with a binary operation is a monoid if it has an identity element and the operation is associative. If every element in the monoid also has an inverse, then it is a group.

[3] A monoid M is left cancellative if $ab = ac$ implies $b = c$ for all $a, b, c \in M$. Similarly, a monoid is right cancellative if $ba = ca$ implies $b = c$ for all $a, b, c \in M$. If a monoid is both left- and right cancellative, it is simply called cancellative.

of Δ in a Garside monoid M **simple elements**, and we denote this set by S. Moreover, Theorem 1.4 in [Geb05] tells us that if a is an atom of M, then $a \preccurlyeq \Delta$.

When we think back to the braid group, it is clear that the fundamental braid is the Garside element for B_n^+, which is a Garside monoid. The braid generators are the atoms generating B_n^+, and the word-length $||X|||$ of $X \in B_n$ is N_X. The left- and right divisors of the fundamental braid are simple braids, i.e. braids in which every pair of strands cross at most once. Also, the generators of B_n^+ are included in the set of simple braids S_n, so we can say that simple braids generate the positive braid monoid.

Lastly, a group G is called a **Garside group** if it is the group of fractions[4] of a Garside monoid. Every Garside monoid admits a group of fractions. Furthermore, every Garside monoid embeds into its corresponding Garside group, so positive elements which are equal in the Garside group are also equal in the Garside monoid. Clearly, the braid group on n strands, B_n, is a Garside group. Other examples of Garside groups include the free abelian groups of finite rank, spherical type Artin-Tits groups, and torus knot groups.

The partial orders $\preccurlyeq$ and $\succcurlyeq$ defined on the monoid extend to the Garside group, as well as the notion of left and right lcms and gcds. Let G be a Garside group, and $a, b \in G$. Then $a \preccurlyeq b$ means $ap = b$ for some $p \in M$ where M is the Garside monoid. Equivalently, $a \preccurlyeq b$ means $a^{-1}b \in M$. $b \succcurlyeq a$ means $qa = b$ for some $q \in M$, or equivalently, $ba^{-1} \in M$. The left lcm $\vee$, left gcd $\wedge$, right lcm $\vee^{\curvearrowleft}$ and right gcd $\wedge^{\curvearrowleft}$ are defined the same way as earlier, but for elements of G, with the result of each of these operations being also in G.

With all of this in mind, we can say that G admits a lattice order $(G, \preccurlyeq, \wedge, \vee)$ which is invariant under left multiplication. Similarly, $(G, \succcurlyeq, \wedge^{\curvearrowleft}, \vee^{\curvearrowleft})$ is also a lattice order, invariant under right multiplication. We can also say that M admits a lattice order $(M, \preccurlyeq, \wedge, \vee)$, since every pair of elements $a, b \in M$ has a unique (left) lcm and a unique (left) gcd, and similarly for $(M, \succcurlyeq, \wedge^{\curvearrowleft}, \vee^{\curvearrowleft})$.

Notice that $M = \{p \in G \mid 1 \preccurlyeq p\} = \{q \in G \mid q \succcurlyeq 1\}$. The set of simple elements can be written as $S = \{s \in G \mid 1 \preccurlyeq s \preccurlyeq \Delta\} = \{t \in G \mid \Delta \succcurlyeq t \succcurlyeq 1\}$. Furthermore, the set S forms a sublattice of M, with the identity being the minimum, and Δ the maximum. Figure 4.1 is provided in [BGGM07a]. It illustrates the sublattice of simple elements for the braid group on 4 strands, B_4, induced by the $\preccurlyeq$ partial order. An element a is joined by a line to an element b in the row above if and only if $a \preccurlyeq b$. Furthermore, the type of line that connects a and b tells us how to get from a to b by right multiplication of an atom. Thus, $a\sigma_i = b$, where a single line indicates σ_1, a double line indicates σ_2, and a dotted line indicates σ_3. So, for example, $\sigma_3\sigma_2\sigma_1 \preccurlyeq \sigma_1\sigma_3\sigma_2\sigma_1$, via right multiplication with σ_2, since $\sigma_3\sigma_2\sigma_1 \cdot \sigma_2 = \sigma_3\sigma_1\sigma_2\sigma_1 = \sigma_1\sigma_3\sigma_2\sigma_1$. We can also compare elements which are not one row apart. For example, $\sigma_2 \preccurlyeq \sigma_2\sigma_1\sigma_3\sigma_2$.

Furthermore, we can use this diagram to find the left gcd (respectively left lcm) of any two simple braids. Follow the lines down (resp. up) to find the possible common divisors (resp. common multiples), and look for the divisor (resp. multiple) which is the highest (resp. lowest) in the diagram.

[4]To obtain the group of fractions of a monoid, we use the generators and relations of the monoid to generate a group, hence allowing for inverses.

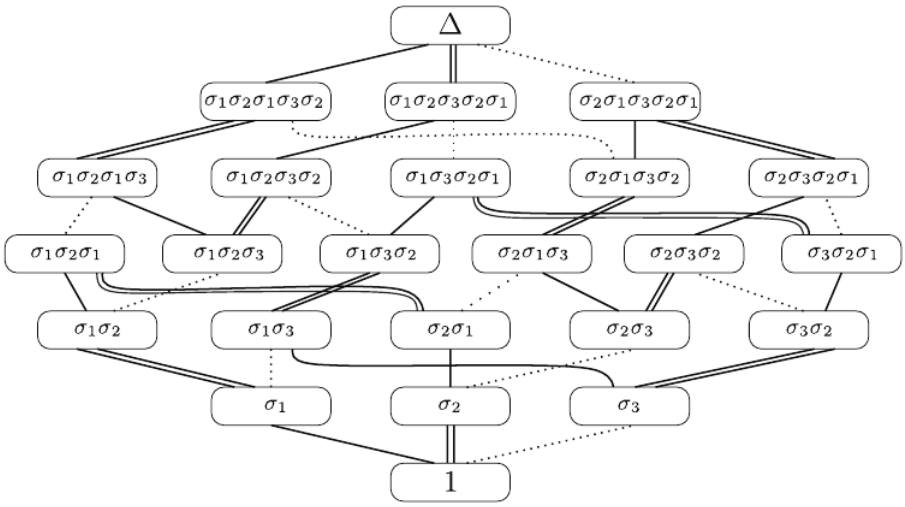


Figure 4.1

Now that we have a solid grasp on the two partial orders, I would like to point out some interesting and useful results.

Theorem 4.2.1 (Theorems 1.4(c) and 1.6 in [Geb05]). *Let M be a Garside monoid with Garside element Δ and group of fractions G.*

(*i*) *M is invariant under conjugation by Δ, i.e. for all elements $p \in M$, $\Delta^{-1} p \Delta \in M$ too.*

(*ii*) *For every $x \in G$ there are integers p and s such that $\Delta^p \preccurlyeq x \preccurlyeq \Delta^s$.*

(*iii*) *There is an integer k such that Δ^k is central in G.*

As for braids, we can refer to conjugation by Δ in general Garside groups as the map $\tau : G \to G$ defined by $\tau(x) = \Delta^{-1} x \Delta$. As before, τ is an automorphism on G. However, τ is not necessarily an involution. When we first discussed τ in the context of braids, we observed that $\tau^m = \tau^n$ if and only if $m \equiv n \mod 2$ (i.e., m and n have the same parity). This is because Δ^2 is central in the braid group. Now that we are generalising to Garside groups, we have that $\tau^m = \tau^n$ if and only if $m \equiv n \mod k$ where k is the smallest integer such that Δ^k is central in G [exa22]. For braids, we had that $\tau^p = \tau^{-p}$, but this is not the case for general Garside groups, and hence we cannot assume τ is an involution.

Both $\preccurlyeq$ and $\succcurlyeq$ are invariant under τ, i.e. if $a \preccurlyeq b$ and $d \succcurlyeq c$ for $a, b, c, d \in G$, then $\tau(a) \preccurlyeq \tau(b)$ and $\tau(d) \succcurlyeq \tau(c)$. Notice that statement (*ii*) in Theorem 4.2.1 is an extension of Theorem 2.2.6 to Garside groups, since the partial order $\leq$ is equivalent to $\preccurlyeq$ when (a power of) Δ is the element on either side of the partial order symbol.

The following lemma gives us more information on the relationship between $\preccurlyeq$ and $\succcurlyeq$:

Lemma 4.2.2 (Lemma 1.3 in [GGM10b]). *For G a Garside group and any $a, b \in G$, the following hold:*

(*i*) $a \preccurlyeq b \Leftrightarrow a^{-1} \succcurlyeq b^{-1}$.

(*ii*) $a \preccurlyeq b \Leftrightarrow \tau(a) \preccurlyeq \tau(b)$.

(*iii*) $(a \wedge b)^{-1} = a^{-1} \vee^{\curvearrowleft} b^{-1}$.

(*iv*) $(a \vee b)^{-1} = a^{-1} \wedge^{\curvearrowleft} b^{-1}$.

(*v*) $\tau(a \wedge b) = \tau(a) \wedge \tau(b)$.

(*vi*) $\tau(a \vee b) = \tau(a) \vee \tau(b)$.

We used (*iii*) and (*iv*) in Section 4.1 to help us find the left gcd and left lcm of two braids, from the method for finding the right lcm and right gcd.

For a simple element $s \in S$, the **right complement** of s is defined by $\partial(s) = s^{-1}\Delta$. Notice that $s\partial(s) = \Delta$, which is why it is the *right* complement – it tells us what to multiply on the right of s to obtain Δ. The **left complement** of s is defined by $\partial^{-1}(s) = \Delta s^{-1}$. Again, notice that $\partial^{-1}(s)s = \Delta$, as $\partial^{-1}(s)$ tells us what to multiply on the *left* of s to obtain Δ.

The inverse notation for the left complement makes sense when we think of ∂ as a map from S to S. $\partial^{-1}(\partial(s)) = \partial^{-1}(s^{-1}\Delta) = \Delta(s^{-1}\Delta)^{-1} = \Delta\Delta^{-1}s = s$. By a very similar argument, $\partial(\partial^{-1}(s)) = s$. In fact, the map $\partial : S \to S$ is a bijection on the finite set S. Furthermore, $\partial^2(s) = \partial(\partial(s)) = \partial(s^{-1}\Delta) = (s^{-1}\Delta)^{-1}\Delta = \Delta^{-1}s\Delta$. Thus, $\partial^2(s) = \tau(s)$ for all $s \in S$.

We can now give new definitions of left- and right-weightedness. For two simple elements $a, b \in S$, the factorisation $a \cdot b$ is **left-weighted** if $\partial(a) \wedge b = 1$, or equivalently, if $ab \wedge \Delta = a$. The factorisation $a \cdot b$ is **right-weighted** if $a \wedge^{\curvearrowleft} \partial^{-1}(b) = 1$, or equivalently, if $ab \wedge^{\curvearrowleft} \Delta = b$. For an element $x \in G$, we say the factorisation $x = \Delta^p x_1, \ldots, x_r$, where $p \in \mathbb{Z}$ and $r \geq 0$, is the **left-normal form** of x if, for each $i = 1, \ldots, r$, $x_i \in S \backslash \{1, \Delta\}$ and $x_i x_{i+1}$ is a left-weighted factorisation. These definitions are equivalent to the definitions we have seen for braids.

We can use this definition of left-weightedness, and see that for $u \in M$, the first factor in the left-normal form of u will be $u \wedge \Delta$. In fact, if we have an element in left-normal form as $\Delta^p x_1 \ldots x_r$, then $x_i x_{i+1} \ldots x_r \wedge \Delta = x_i$ for each $1 \leq i \leq r$. Furthermore, we can use this to turn a factorisation which is not left-weighted into its left-weighted form: we replace $a \cdot b$ (a and b simple) with $(as) \cdot (s^{-1}b)$ where $s = \partial(a) \wedge b = a^{-1}\Delta \wedge b = \Delta\tau(a^{-1}) \wedge b$. Similarly, to put $a \cdot b$ into right-weighted form, we write it as $(as^{-1}) \cdot (sb)$ where $s = a \wedge^{\curvearrowleft} \partial^{-1}(b)$.

The right complement is useful in finding the **left-normal form of the inverse** of an element. If $x = \Delta^p x_1 \ldots x_r$ with $\inf x = p$ is the left-normal form of x, then the left-normal form of x^{-1} is

$$x^{-1} = \Delta^{-p-r} x'_r \ldots x'_1 \text{ where } x'_i = \tau^{-p-i}(\partial(x_i)).$$

The definitions of the infimum, supremum, and canonical length translate trivially from the language of braids to that of Garside groups. However, when working with general Garside elements, we will call it the normal length and left- and right-normal forms, instead of the canonical length and left- and right-canonical forms. The summit infimum, summit supremum, and summit length also translate trivially to Garside groups. Cycling and decycling works the same way; the only change to our earlier algorithms is to replace B_n with G, and any occurrence of $\frac{n(n-1)}{2}$ with $||\Delta||$.

4.3 Minimal simple elements

Nuno Franco and Juan González-Meneses offer the next major improvement to the conjugacy problem solution in *Conjugacy problem for braid groups and Garside groups* [FGM03]. The core of their improvement lies in whittling down the set of simple elements to a much smaller subset, which the authors call minimal simple elements. Then, it is possible to determine the summit set and the super summit set of an element by conjugating with minimal simple elements, instead of all simple elements as in the original solutions. Recall that for the braid group on n strands, the number of simple elements corresponds to the number of permutations on n elements, which is $n!$. The number of minimal simple elements, on the other hand, will be at most the number of atoms, which is $n-1$ for braids.

For the rest of this dissertation, M will denote a Garside monoid, G its group of fractions, and Δ the corresponding Garside element. Given that $M \subset G$, we call the elements in M positive elements of G. Let the set of simple elements be denoted by S.

Recall that for $x \in G$ the conjugacy class of x is denoted by $C(x) = \{y^{-1}xy \mid y \in G\}$. The authors introduce the set $C^{\geq m}(x) = \{v \in C(x) \mid \inf v \geq m\}$. Notice that if m is the summit infimum (or summit power, in Garside's terminology), then $C^{\geq m}(x) = \mathrm{SS}(x)$. The authors denote the super summit set (the subset of $C(x)$ containing all elements of minimal normal length) by $C^{\mathrm{sum}}(x)$. This notation does make it clear that these sets are subsets of the conjugacy class, but it is not commonly used. We will continue using $\mathrm{SSS}(x)$ to denote the super summit set of $x \in G$. However, for the discussion of this paper, we will use $C^{\geq m}(x)$ as the authors do, as the algorithms involving this set are not limited to the case where m is the summit infimum of x – in fact, the authors remark after the algorithm for computing $C^{\geq m}(x)$ that we can find all positive elements conjugate to x if we set $m = 0$. As mentioned when we discussed cycling in Section 3.2, we can use the cycling operation defined in [ERM94] to easily find the value of m which would make $C^{\geq m}(x) = \mathrm{SS}(x)$, as well as a representative of the summit set.

Franco and González-Meneses start by showing that a certain class of properties on simple elements will lead to a reduction in the number of elements necessary to find the summit set and super summit set. Let $\mathcal{P}$ be some property for simple elements. Let $S_{\mathcal{P}}$ denote the set of simple elements satisfying this property $\mathcal{P}$. The set of **minimal simple elements** for the property $\mathcal{P}$, $\min(S_{\mathcal{P}})$, is the set of elements in $S_{\mathcal{P}}$ which are minimal with respect to $\preccurlyeq$. We will use "minimal" to mean "minimal with respect to $\preccurlyeq$" for the rest of this dissertation. For every atom $a \in M$, let $mult(a) = \{s \in S \mid a \preccurlyeq s\} \subset S$, i.e., the set of all simple elements for which a is a prefix.

We will insist that the property $\mathcal{P}$ is closed under gcd, i.e., for two simple elements satisfying $\mathcal{P}$, the gcd of these elements must also satisfy $\mathcal{P}$. Thus, if $s_1, s_2 \in S_{\mathcal{P}}$, then $s_1 \wedge s_2 \in S_{\mathcal{P}}$. With this in mind, the authors show two results:

Lemma 4.3.1 (Lemma 4.2 in [FGM03])**.** *Suppose $\mathcal{P}$ is closed under gcd and let a be an atom of M. If the set $S_{\mathcal{P}} \cap mult(a)$ is non-empty, then it has a unique minimal element, which we will denote by ρ_a.*

Note that $S_{\mathcal{P}} \cap mult(a)$ is all the simple elements which satisfy the property $\mathcal{P}$ while also having the atom a as a prefix.

Corollary 4.3.2 (Corollary 4.3 in [FGM03]). *Suppose M has k atoms. Then $|\min(S_{\mathcal{P}})| \leq k$ if $\mathcal{P}$ is closed under gcd.*

I include the proof of this corollary as it is useful to a later discussion of the algorithms the authors provide.

Proof. Every element in M must have an atom as a prefix. Take $s \in \min(S_{\mathcal{P}})$ and consider an atom $a \preccurlyeq s$. Since s is minimal in $S_{\mathcal{P}}$, it is also minimal in $S_{\mathcal{P}} \cap mult(a)$. Thus, $s = \rho_a$, and so $\min(S_{\mathcal{P}}) \subset \{\rho_a \mid a \text{ is an atom}\}$. Hence, $|\min(S_{\mathcal{P}})| \leq k$, where k is the number of atoms in M. □

Note that for a set A, $|A|$ is the cardinality (number of elements) of A. This corollary makes it clear why we would want to work with minimal simple elements. As mentioned earlier, in the braid group, there are $n-1$ generators and $n!$ simple braids. According to this corollary, if $\mathcal{P}$ is a property closed under gcd, then there are at most $n-1$ minimal simple elements satisfying $\mathcal{P}$.

Now that we know what minimal simple elements are, we can look at the suitable properties, closed under gcd, which allow for the computation of $C^{\geq m}(x)$ for $x \in G$.

Let $x \in G$ and $v \in C^{\geq m}(x)$ for some $m \in \mathbb{Z}$. A simple element s satisfies the property $\mathcal{P}_v^{\geq m}$ if it conjugates v to another element in $C^{\geq m}(x)$, i.e., $s^{-1}\, v\, s \in C^{\geq m}(x)$. In other words, conjugating v with s results in an element with the same or higher inf as v.

Proposition 4.3.3 (Proposition 4.7 in [FGM03]). *If $v \in C^{\geq m}(x)$, one can write $v = \Delta^m w$ where $w \in M$. Then a simple element $s \in S$ satisfies the property $\mathcal{P}_v^{\geq m}$ if and only if $\tau^m(s) \preccurlyeq ws$.*

It is powerful to be able to check how conjugation by a specific simple element will affect the value of inf. If $\tau^m(s) \not\preccurlyeq ws$, then $\inf s^{-1}\, v\, s < m$. Notice that $s^{-1}\, v\, s = s^{-1}\Delta^m ws = \Delta^m \tau^m(s^{-1})ws = \Delta^m(\tau^m(s))^{-1}ws$. As long as $(\tau^m(s))^{-1}ws \in M$, $\inf s^{-1}\, v\, s \geq m$, which means s satisfies $\mathcal{P}_v^{\geq m}$. Lastly, $(\tau^m(s))^{-1}ws \in M \Leftrightarrow \tau^m(s) \preccurlyeq ws$.

The next proposition has been adjusted, as it is correct in the paper, but is inconsistent with later claims. Section B.2.1 contains a proof of this adjusted statement, and motivation for the adjustment.

Proposition 4.3.4 (Proposition 4.8 in [FGM03], adjusted). *For every $v \in C^{\geq m}(x)$ and every $m \in \mathbb{Z}$, the property $\mathcal{P}_v^{\geq m}$ is closed under gcd.*

To put it another way, this tells us that if $s^{-1}\, v\, s \in C^{\geq m}(x)$ and $t^{-1}\, v\, t \in C^{\geq m}(x)$ (where $s, t \in S$ and $v \in C^{\geq m}(x)$), then $(s \wedge t)^{-1}\, v\, (s \wedge t) \in C^{\geq m}(x)$ too.

For every $v \in C^{\geq m}(x)$ we define the set $S_v^{\geq m} = \min(S_{\mathcal{P}_v^{\geq m}})$. $S_v^{\geq m}$ is the set of minimal simple element among those which conjugate v to an element in $C^{\geq m}(x)$.

The cardinality of $S_v^{\geq m}$ for every $v \in C^{\geq m}(x)$ is at most the number of atoms in M, by Corollary 4.3.2 and Proposition 4.3.4. Lastly, we have this result, which is analogous to Corollary 3.2.3, and gives a **convexity property** for $C^{\geq m}(x)$ with minimal simple elements.

Proposition 4.3.5 (Proposition 4.10 in [FGM03]). *Given $u, v \in C^{\geq m}(x)$ for some $x \in G$, there exists a sequence $u = u_1, u_2, \ldots, u_k = v$ of elements in $C^{\geq m}(x)$ such that for $i = 1, \ldots, k-1$, the elements u_i and u_{i+1} are conjugate by an element in $S^{\geq m}_{u_i}$.*

Thus, in order to compute $C^{\geq m}(x)$ for some $x \in G$, it suffices to conjugate every $v \in C^{\geq m}(x)$ by the elements in the small set $S^{\geq m}_v$, instead of S as in previous solutions.

Next, we investigate minimal simple elements to compute $\mathrm{SSS}(x)$.

Let $x \in G$ and let $v \in \mathrm{SSS}(x)$. A simple element s satisfies the property $\mathcal{P}^{\mathrm{SSS}}_v$ if it conjugates v to an element in $\mathrm{SSS}(x)$. In other words, conjugating v with s will result in an element $s^{-1}\, v\, s$ with the same normal length as v (which is the summit length of x).

Proposition 4.3.6 (Proposition 4.12 in [FGM03]). *For every $v \in \mathrm{SSS}(x)$, the property P^{SSS}_v is closed under gcd.*

As before, if $s^{-1}\, v\, s \in \mathrm{SSS}(x)$ and $t^{-1}\, v\, t \in \mathrm{SSS}(x)$ (where $s, t \in S$ and $v \in \mathrm{SSS}(x)$), then $(s \wedge t)^{-1}\, v\, (s \wedge t) \in \mathrm{SSS}(x)$ too.

For every $v \in \mathrm{SSS}(x)$, we define $S^{\mathrm{SSS}}_v = \min(S_{\mathcal{P}^{\mathrm{SSS}}_v})$. S^{SSS}_v is the set of minimal simple elements among those which conjugate v to an element in $\mathrm{SSS}(x)$. Similarly to the case for $S^{\geq m}_v$, the cardinality of S^{SSS}_v for every $v \in \mathrm{SSS}(x)$ is at most the number of atoms in M, by Corollary 4.3.2 and Proposition 4.3.6.

Again, we have the following result, analogous to Corollary 3.2.3 and Proposition 4.3.5, which gives a **convexity property** for $\mathrm{SSS}(x)$ with minimal simple elements:

Proposition 4.3.7 (Proposition 4.14 in [FGM03]). *For u, v conjugate elements in $\mathrm{SSS}(x)$, there exists a sequence $u = u_1, u_2, \ldots, u_k = v$ of elements in $\mathrm{SSS}(x)$ such that, for $i = 1, \ldots, k-1$, the elements u_i and u_{i+1} are conjugate by an element in $S^{\mathrm{SSS}}_{u_i}$.*

Thus, in order to compute $\mathrm{SSS}(x)$ for $x \in G$, it suffices to conjugate every $v \in \mathrm{SSS}(x)$ by the elements in S^{SSS}_v.

From this discussion, it should be clear that there is not one fixed set of minimal simple elements which allow us to compute $C^{\geq m}(x)$ or $\mathrm{SSS}(x)$ quicker than if we were to use the whole set S. The appropriate set of minimal simple elements depends on the element of $C^{\geq m}(x)$ or $\mathrm{SSS}(x)$ we have on hand. Nonetheless, once we have a representative of $C^{\geq m}(x)$ or $\mathrm{SSS}(x)$, it is much quicker to compute the rest of the set.

We now look at the algorithms the authors give which allow us to compute these special subsets of S, as well as $C^{\geq m}(x)$ and $\mathrm{SSS}(x)$ with this new knowledge.

The first algorithm computes the left lcm $s \vee v$ of a simple element $s \in S$ and a positive element $v \in M$; in particular, it finds $s' \in S$ such that $s \vee v = vs'$. The authors note that it is well known how to find the lcm and gcd of two simple elements, as well as the normal forms of any element in G, a Garside group. We already know how to compute all of this for the case of braids[5], but I include this algorithm for the sake of completeness.

[5] In the case of the braid group, we already know how to find s' such that $s \vee v = vs'$ – the method at the end of Section 4.1 gives $A_1 \vee A_2 = A_1(U_1^-)^{-1} = A_2(U_2^-)^{-1}$ for arbitrary braids $A_1, A_2 \in B_n$, so setting $A_1 = s$ and $A_2 = v$, we have $s' = (U_2^-)^{-1}$.

Algorithm 4.3.8 (Algorithm 1 in [FGM03])**.**
Input: $s \in S$, $v \in M$.
Output: $s' \in S$ *such that* $s \vee v = vs'$.

1. Compute the normal form of v as $v_1 \dots v_t$.
2. Set $s_0 = s$.
3. For every $i = 1, \dots, t$, compute $s_{i-1} \vee v_i$, and write it as $v_i s_i$.
4. Return s_t.

The authors do not specify whether the normal form in Step 1 is the left- or right-normal form. Given that we are finding the left lcm, it is most likely the left-normal form.

For $x \in G$ and $m \in \mathbb{Z}$, we have seen that the main problem in computing $C^{\geq m}(x)$ is finding $S_v^{\geq m}$ for every $v \in C^{\geq m}(x)$. Now, take $v \in C^{\geq m}(x)$, let a be an atom of M, and consider the set

$$S_{\mathcal{P}_v^{\geq m}} \cap mult(a)$$

(i.e., the set of simple elements satisfying $\mathcal{P}_v^{\geq m}$ while also having the atom a as a prefix). Notice that this set is always non-empty, since Δ is an element in this set: conjugation by Δ does not decrease inf, so Δ satisfies $\mathcal{P}_v^{\geq m}$ for every v (and Δ is a simple element); furthermore, every atom is a prefix of Δ, by definition of the Garside element. Thus, as this set is non-empty, Lemma 4.3.1 tells us that it has a unique minimal element, which will be denoted by r_a in this situation where we are working with $C^{\geq m}(x)$. Thus, r_a is the unique minimal element satisfying $a \preccurlyeq r_a \preccurlyeq \Delta$ and $r_a^{-1}\, v\, r_a \in C^{\geq m}(x)$.

From the proof of Corollary 4.3.2, we know that $S_v^{\geq m} \subset \{r_a \mid a \text{ is an atom}\}$. Thus, the next algorithm computes r_a for an atom a, and we use this over all atoms, in the subsequent algorithm, to compute $S_v^{\geq m}$.

Let $\inf v = p \geq m$ so that $v = \Delta^m w$ where $w \in M$ (i.e. $\inf w$ could be ≥ 0). From Proposition 4.3.3 we know that a simple element s satisfies $\mathcal{P}_v^{\geq m}$ if and only if $\tau^m(s) \preccurlyeq ws$.

Algorithm 4.3.9 (Algorithm 2 in [FGM03])**.**
Input: $a \in M$ *an atom,* $m \in \mathbb{Z}$, $v \in C^{\geq m}(x)$ *(for some* $x \in G$*).*
Output: r_a*, the minimal element in* $S_{\mathcal{P}_v^{\geq m}} \cap mult(a)$.

1. Compute the left-normal form of v as $\Delta^p w_1 \dots w_t$.
2. If $p > m$, then return a.
3. Set $w = w_1 \dots w_t$ and $s = a$.

4. Compute $\tau^m(s)$.

5. Use Algorithm 4.3.8 to compute s' such that $\tau^m(s) \vee ws = wss'$.

6. If $s' = 1$, then return s.

7. Set $s = ss'$; go to Step 4.

We can stop at Step 2, since $\inf v = p > m$ means that conjugating by the atom a would definitely lead to $\inf \geq m$ (as required), and since a is an atom, it is already minimal. We need to do more if $p \not> m$, i.e. $p = m$. In this case, we either have that a satisfies $\mathcal{P}_v^{\geq m}$ because $\tau^m(a) \vee wa = wa$, which means $\tau^m(a) \preccurlyeq wa$, or we need to concatenate more factors to the right of a until $\mathcal{P}_v^{\geq m}$ is indeed satisfied.

Now, we can compute $S_v^{\geq m}$:

Algorithm 4.3.10 (Algorithm 3 in [FGM03])**.**
Input: $m \in \mathbb{Z}$, $v \in C^{\geq m}(x)$ *(for some* $x \in G$*).*
Output: $S_v^{\geq m}$*.*

1. List the atoms of M, say $a_1, \ldots, a_\lambda$. Set $R = \emptyset$.

2. For $i = 1, \ldots, \lambda$, do:

   (a) Use Algorithm 4.3.9 to compute r_{a_i}.

   (b) Compute $J_i = \{j \mid j \in R \text{ and } a_j \preccurlyeq r_{a_i}\}$ and $K_i = \{j \mid j > i \text{ and } a_j \preccurlyeq r_{a_i}\}$.

   (c) If $J_i = K_i = \emptyset$, then set $R = R \cup \{i\}$.

3. Return $\{r_{a_i} \mid i \in R\}$.

Steps 2b and 2c is quite important, as it determines which elements we have found are indeed minimal. The authors remark that we could also find $S_v^{\geq m}$ by considering the set $\{r_{a_i} \mid i = 1, \ldots, \lambda\}$, and taking the minimal elements in this set. However, this would require comparing all the r_{a_i}, and it is much faster to check if an atom divides an element, than to compare two elements, which is why the above algorithm is preferable.

Why does this exact method work? Remember that r_{a_i} is the *unique* minimal simple element in the set $S_{\mathcal{P}_v^{\geq m}} \cap mult(a_i)$. R keeps track of the indices i for which r_{a_i} is a minimal simple element. Suppose indices $j < k < l$ have already been added to R, and we are now considering r_{a_m} as computed from a_m (where $m > l$). If, say, $a_j \preccurlyeq r_{a_m}$ (the test performed via set J_i), it means that $r_{a_j} \preccurlyeq r_{a_m}$, by the uniqueness and minimality of r_{a_j}, and so we do not add m to R. On the other hand, if $a_p \preccurlyeq r_{a_m}$ for $p > m$ (the test performed via set K_i), then the unique minimal element for a_p, r_{a_p}, will satisfy $r_{a_p} \preccurlyeq r_{a_m}$, and so r_{a_m} is not minimal (or if

$r_{a_m} = r_{a_p}$, it is minimal, but will be computed when $i = p$) and m is not added to R. It may be that two different atoms lead to the same minimal element, in which case the test performed via the set K_i ensures the same element is not added twice.

Finally, we can find $C^{\geq m}(x)$ for some $x \in G$ and $m \in \mathbb{Z}$, thanks to Proposition 4.3.5.

Algorithm 4.3.11 (Algorithm 4 in [FGM03])**.**
Input: $m \in \mathbb{Z}$, $x \in G$.
Output: the set $C^{\geq m}(x)$.

1. Compute the left-normal form of x.
2. Use Algorithm 3.2.10 to compute $\tilde{x} \in C^{\geq \inf_* x}(x)$.
3. If $\inf \tilde{x} < m$ (i.e. $\tilde{x} \notin C^{\geq m}(x)$), then return $\emptyset$.
4. Set $V = \{\tilde{x}\}$ and $W = \emptyset$.
5. While $V \neq W$, do:
   (a) Take $v \in V \backslash W$.
   (b) Use Algorithm 4.3.10 to compute $S_v^{\geq m}$.
   (c) For every $r \in S_v^{\geq m}$, do:
      i. Set $w = r^{-1} v r \in C^{\geq m}(x)$ and compute the left-normal form of w.
      ii. If $w \notin V$, then set $V = V \cup \{w\}$.
   (d) Set $W = W \cup \{v\}$.
6. Return V.

The sets V and W serve the same logical purpose as $\mathcal{V}$ and $\mathcal{W}$ in Algorithm 3.2.12.

We can find m such that $C^{\geq m}(x) = \mathrm{SS}(x)$ by repeatedly cycling x to find $\tilde{x} \in \mathrm{SS}(x)$. In that case, the algorithm can start at Step 4.

Now, we can look at the algorithms to compute $\mathrm{SSS}(x)$ for $x \in G$. Take $v \in \mathrm{SSS}(x)$ and let $\inf v = p$ and $\ell(v) = t$ (so $\sup v = t + p$). Consider an atom $a \in M$. Similarly to the discussion before Algorithm 4.3.9, we need to find the minimal element $\rho_a \in S_{\mathcal{P}_v^{\mathrm{SSS}}} \cap mult(a)$ (which exists because Δ is in this set, and is unique by Lemma 4.3.1). ρ_a is the unique minimal element satisfying $a \preccurlyeq \rho_a \preccurlyeq \Delta$ and $\rho_a^{-1} v \rho_a \in \mathrm{SSS}(x)$. With this, we can find S_v^{SSS}, and finally $\mathrm{SSS}(x)$.

Algorithm 4.3.12 (Algorithm 5 in [FGM03])**.**
Input: $a \in M$ *an atom,* $v \in \mathrm{SSS}(x)$ *(for some* $x \in G$*),* $p = \inf v$*, and* $t = \ell(v)$.
Output: ρ_a*, the minimal element in* $S_{\mathcal{P}_v^{\mathrm{SSS}}} \cap mult(a)$.

1. Use Algorithm 4.3.9 to compute r_a, minimal in $S_{\mathcal{P}_v^{\geq p}} \cap mult(a)$.
2. Set $s = r_a$.
3. Compute the right-normal form of $s^{-1}\, v\, s$ as $w_1 \dots w_k \Delta^p$.
4. If $k = t$, then return s.
5. Set $s = sw_1$; go to Step 3.

We know that conjugating v by r_a will not change inf (it cannot decrease by definition of r_a and cannot increase because $\inf v$ is the summit infimum). We need to check, however, whether it would lead to an appropriate normal length, for it to conjugate v to elements in the super summit set. Thus, we start by finding $r_a^{-1} v r_a$ in right-normal form, say $w_1 \dots w_k \Delta^p$. We know the power of Δ will be p; we want to check what happens to k, which is the normal length of the resulting element. If $k = t = \ell(v)$, then r_a indeed conjugates v to an element of the super summit set, and we can stop (Step 4). However, if $k \neq t$, we know that the normal length has increased, so we need to decycle $r_a^{-1}\, v\, r_a$ to decrease the normal length. This is achieved by conjugating v with $r_a w_1$. Thus, $w_1^{-1} r_a^{-1}\, v\, r_a w_1 = w_1^{-1} \cdot w_1 \dots w_k \Delta^p \cdot w_1 = w_2 \dots w_k \Delta^p w_1$, which is how we decycle in the right-normal form. We keep checking k and decycling as necessary until $k = t$, and in the process we find ρ_a. We can also see from this algorithm that $r_a \preccurlyeq \rho_a$, since we multiply simple elements to the right of r_a in order to obtain ρ_a (or we make no changes, in which case $r_a = \rho_a$, but this still means $r_a \preccurlyeq \rho_a$.)

S_v^{SSS} is found in a very similar fashion to $S_v^{\geq m}$:

Algorithm 4.3.13 (Algorithm 6 in [FGM03])**.**
Input: $v \in \mathrm{SSS}(x)$ *(for some* $x \in G$*),* $p = \inf v$*, and* $t = \ell(v)$*.*
Output: S_v^{SSS}*.*

1. List the atoms of M, say $a_1, \dots, a_\lambda$. Set $R = \emptyset$.
2. For $i = 1, \dots, \lambda$, do:
   (a) Use Algorithm 4.3.12 to compute ρ_{a_i}.
   (b) Compute $J_i = \{j \mid j \in R \text{ and } a_j \preccurlyeq \rho_{a_i}\}$ and $K_i = \{j \mid j > i \text{ and } a_j \preccurlyeq \rho_{a_i}\}$.
   (c) If $J_i = K_i = \emptyset$, then set $R = R \cup \{i\}$.
3. Return $\{\rho_{a_i} \mid i \in R\}$.

Finally, we can compute $\mathrm{SSS}(x)$ for any $x \in G$, also by a very similar process to finding $C^{\geq m}(x)$:

Algorithm 4.3.14 (Algorithm 7 in [FGM03])**.**
Input: $x \in G$.
Output: $\mathrm{SSS}(x)$.

1. Use Algorithm 3.2.11 to compute $\tilde{x} \in \mathrm{SSS}(x)$.
2. Set $V = \{\tilde{x}\}$, and $W = \emptyset$.
3. While $V \neq W$, do:
   (a) Take $v \in V \backslash W$.
   (b) Use Algorithm 4.3.13 to compute S_v^{SSS}.
   (c) For every $r \in S_v^{\mathrm{SSS}}$, do:
      i. Set $w = r^{-1} v r \in \mathrm{SSS}(x)$ and compute the left-normal form of w.
      ii. If $w \notin V$, then set $V = V \cup \{w\}$.
   (d) Set $W = W \cup \{v\}$.
4. Return V.

For proofs of these algorithms, see Section 5 of [FGM03]. Note that I adjusted the logic of Algorithms 4.3.11 and 4.3.14 slightly, for consistency with other algorithms in this dissertation. The authors also study the complexity of their algorithms in Section 6 of the paper, which we will not discuss here. I will only mention that for $W \in B_n$ as a word of length l, the complexity of computing $\mathrm{SSS}(W)$ is $\mathcal{O}(kl^2n^4)$ where k is the number of elements in $\mathrm{SSS}(W)$. In comparison, according to the methods employed by Franco and González-Meneses, the El-Rifai and Morton algorithm to find the super summit set takes $\mathcal{O}(kl^2(n!)n\log n)$. The improved algorithm is clearly faster, but the magnitude of k is hidden. In [GGM10b] it is stated that $|\mathrm{SSS}(W)| \leq |S|^m$ where S is the set of simple elements; if we write W as a product of simple elements and inverses of simple elements, m is the number of such factors. Thus, k has upper bound $(n!)^m$, and so the improved algorithm is still not in polynomial-time.

The authors do not provide an algorithm to solve the conjugacy problem, but we can follow a very similar process to Algorithm 3.2.13. As before, we denote the conjugating element between $\tilde{x}$ (the representative of $\mathrm{SSS}(x)$) and another element of $\mathrm{SSS}(x)$, say q, as c_q.

Algorithm 4.3.15.
Input: $x, y \in G$.
Output: $c \in G$ *such that* $c^{-1} x c = y$ *or* `fail` *if* x *and* y *are not conjugate.*

1. Use Algorithm 3.2.11 to compute $\tilde{x} \in \mathrm{SSS}(x)$ and a such that $a^{-1} x a = \tilde{x}$, and $\tilde{y} \in \mathrm{SSS}(y)$ and b such that $b^{-1} y b = \tilde{y}$.
2. If $\ell(\tilde{x}) \neq \ell(\tilde{y})$, then return `fail`.

3. Set $V = \{\tilde{x}\}$, $W = \emptyset$, and $c_{\tilde{x}} = 1$.

4. While $V \neq W$, do:

   (a) Take $v \in V \backslash W$.

   (b) Use Algorithm 4.3.13 to compute S_v^{SSS}.

   (c) For every $s \in S_v^{\mathrm{SSS}}$, do:

      i. Set $w = s^{-1}\, v\, s$ and compute the left-normal form of w.

      ii. If $w = \tilde{y}$, then set $c_{\tilde{y}} = c_v \cdot s$. Return $a \cdot c_{\tilde{y}} \cdot b^{-1}$.

      iii. If $w \notin V$, then set $c_w = c_v \cdot s$ and $V = V \cup \{w\}$.

   (d) Set $W = W \cup \{v\}$

5. Return `fail`.

Chapter 5

Incorporating graph theory

In this chapter we consider the ultra summit set as presented in [Geb05], which is a subset of the super summit set, and we consider an improvement to the ultra summit set solution given in [BGGM08].

5.1 The ultra summit set

The paper we consider is *A new approach to the conjugacy problem in Garside groups* by Volker Gebhardt [Geb05]. We will see an algorithm to compute the ultra summit set of an element, as well as an algorithm to determine a conjugating element when we know two elements are indeed conjugate. Both of these rely on a third algorithm, which finds minimal simple elements for the ultra summit set.

Let $x \in G$, and consider $\mathrm{SSS}(x)$. We define the **ultra summit set**, denoted by $\mathrm{USS}(x)$, of x as

$$\mathrm{USS}(x) = \{y \in \mathrm{SSS}(x) \mid \mathbf{c}^k(y) = y \text{ for some integer } k > 0\}.$$

Two elements $x, y \in G$ are conjugate in G if and only if they have the same ultra summit set, i.e., $\mathrm{USS}(x) = \mathrm{USS}(y)$, or equivalently, if and only if $\mathrm{USS}(x) \cap \mathrm{USS}(y) \neq \emptyset$.

We can think of the ultra summit set in another way. Recall that for every $y \in \mathrm{SSS}(x)$, $\mathbf{c}(y) \in \mathrm{SSS}(x)$ and $\mathbf{d}(y) \in \mathrm{SSS}(x)$ too. Using the super summit set, we construct a finite directed graph Γ_x, where $\mathrm{SSS}(x)$ is the set of vertices of the graph, and the set of directed edges is $\{(y, \mathbf{c}(y)) : y \in \mathrm{SSS}(x)\}$. The subset of vertices which are contained in a circuit[1] of Γ_x forms the ultra summit set.

[1] In graph theory, a circuit is a sequence of vertices and edges which starts and ends at the same vertex, and allows repeated vertices, but no repeated edges. A graph theoretic cycle is a sequence of vertices and edges which starts and ends at the same vertex but allows no repeated edges or vertices (except the initial/final vertex, of course). It would be more accurate to describe the ultra summit set as the subset of vertices contained in a cycle of Γ_x, as the existence of a circuit implies that the cycling operation $\mathbf{c}$ is not well-defined. However, "cycle" and "cycling" are already commonly-used terms in this field, so I assume Gebhardt uses "circuit" to avoid confusion, a convention we follow too.

By definition, all the elements in the ultra summit set will be of the same normal length, since $\mathrm{USS}(x) \subseteq \mathrm{SSS}(x)$. If we have an element $y \in \mathrm{USS}(x)$, then $\mathbf{c}^m(y) = y$, for some positive integer m. Notice that $\mathbf{c}^{m+1}(y) = \mathbf{c}(y)$, so $\mathrm{USS}(x)$ will include each of the intermediate $\mathbf{c}^i(y)$ where $0 \leq i < m$.

For an element $y \in \mathrm{SSS}(x)$, the **trajectory** of y is $T_y = \{\mathbf{c}^k(y) \mid k \geq 0\}$. We can find a representative of $\mathrm{USS}(x)$ by computing the trajectory of an arbitrary element of $\mathrm{SSS}(x)$ (since $\mathrm{SSS}(x)$ is finite, T_y must also be finite, and so cycling y must eventually produce an element already found). Furthermore, we do not need to have the entire super summit set on hand to find a representative of the ultra summit set – we can use cycling and decycling on x to find a representative $\tilde{x}$ of $\mathrm{SSS}(x)$ and then use the trajectory of $\tilde{x}$ to find a representative of $\mathrm{USS}(x)$. To find a representative, we take some $y \in \mathrm{SSS}(x)$ and compute $\mathbf{c}^i(y)$, starting from $i = 0$, and continuing until we find a repeated element, say $\mathbf{c}^m(y) = \mathbf{c}^n(y)$ where $m < n$. Then the set $\{\mathbf{c}^i(y) \mid i = m, \ldots, n-1\}$ will be a subset of $\mathrm{USS}(x)$, and any element of this set is a representative of $\mathrm{USS}(x)$. Furthermore, we can keep track of the conjugating element at each step (by the definition of cycling), so that we know how get from y to any element in its trajectory.

With this in mind, we have the following algorithms to, firstly, determine the trajectory of an element and the conjugators between the element and every element in its trajectory, and, secondly, to find a representative of the ultra summit set. These algorithms are not given in the paper, but are useful towards building understanding and making later algorithms extra clear.

The structure of this first algorithm is inspired by Algorithms 6.6 and 6.27 in [GGM10b]. We denote the conjugator between x and another element in its trajectory, say $z \in T_x$, by c_z. The set of these conjugators is denoted by $C_x = \{c_z \mid z \in T_x \text{ and } c_z^{-1}\, x\, c_z = z\}$

Algorithm 5.1.1.
Input: $x \in G$.
Output: The sets $T_x = \{\mathbf{c}^k(x) \mid k \geq 0\}$ *and* $C_x = \{c_z \mid z \in T_x \text{ and } c_z^{-1}\, x\, c_z = z\}$.

1. Set $\tilde{x} = x$, $c_{\tilde{x}} = 1$, $T = \emptyset$, and $C = \emptyset$.
2. While $\tilde{x} \notin T$, do:
   (a) Set $T = T \cup \{\tilde{x}\}$, and $C = C \cup \{c_{\tilde{x}}\}$.
   (b) Use Algorithm 3.2.7 to compute $y = \mathbf{c}(\tilde{x})$ and a such that $a^{-1}\, \tilde{x}\, a = y$.
   (c) Set $\tilde{x} = y$ and $c_{\tilde{x}} = c_{\tilde{x}} \cdot a$.
3. Return T and C.

As we compute consecutive cyclings of x, we incorporate the conjugator between x and the previous element and the conjugator between the previous element and the current element, via right multiplication, to find the conjugator between x and the current element. Notice that the while-loop will stop when we find a repeated element, and that repeated element is

not stored, so all the elements of T_x are distinct. The elements of C_x are also distinct, as having repeated conjugators would lead to repeated elements in T_x.

The next algorithm finds a representative $\tilde{x} \in \mathrm{USS}(x)$, and the conjugating element between x and $\tilde{x}$.

Algorithm 5.1.2.
Input: $x \in G$.
Output: $\tilde{x} \in \mathrm{USS}(x)$ *and* $c \in G$ *such that* $c^{-1} x c = \tilde{x}$.

1. Use Algorithm 3.2.11 to find $\tilde{x} \in \mathrm{SSS}(x)$ and b such that $b^{-1} x b = \tilde{x}$.
2. Use Algorithm 5.1.1 to find $T_{\tilde{x}}$ and $C_{\tilde{x}}$, and enumerate these sets as $T_{\tilde{x}} = \{t_1, \ldots, t_\lambda\}$ and $C_{\tilde{x}} = \{c_{t_1}, \ldots, c_{t_\lambda}\}$.
3. Use Algorithm 3.2.7 to find $y = \mathbf{c}(t_\lambda)$ and a such that $a^{-1} t_\lambda a = y$.
4. Set $t_{\lambda+1} = y$.
5. For $t \in \{t_1, \ldots, t_\lambda\}$, do:
   (a) If $t_{\lambda+1} = t$, then return $b \cdot c_t$.

By the design of Algorithm 5.1.1, cycling the last element in the trajectory of $T_{\tilde{x}}$ must lead to an element already in $T_{\tilde{x}}$. Hence, this repeated element, t, will be in $\mathrm{USS}(x)$, and so the conjugating element from x to $\tilde{x} \in \mathrm{USS}(x)$ is $b \cdot c_t$

With the ultra summit set and trajectory of an element defined, Gebhardt states the following results:

Theorem 5.1.3 (Theorem 1.18 in [Geb05])**.** *Let* $x \in G$, $y \in \mathrm{USS}(x)$ *and let* $u, v \in M$ *be such that* $u^{-1} y u \in \mathrm{USS}(x)$ *and* $v^{-1} y v \in \mathrm{USS}(x)$. *Then* $(u \wedge v)^{-1} y (u \wedge v) \in \mathrm{USS}(x)$.

Notice that this is almost analogous to Proposition 4.3.6. The only difference (besides the change from the super summit set to the ultra summit set), is that $u, v \in S$ in Proposition 4.3.6, while here we have $u, v \in M$.

The following corollary is analogous to Corollary 3.2.3 and Proposition 4.3.7, giving us a **convexity property** for the ultra summit set.

Corollary 5.1.4 (Corollary 1.19 in [Geb05])**.** *Let* $x \in G$ *and* $y, z \in \mathrm{USS}(x)$. *There exist elements* $y = y_0, y_1, \ldots, y_{t-1}, y_t = z$ *and elements* $s_1, \ldots, s_t \in S$ *such that* $s_i^{-1} y_{i-1} s_i = y_i$ *for* $i = 1, \ldots, t$.

Gebhardt also introduces **minimal simple elements for the ultra summit set**, in a similar way as is done in [FGM03]. Let $x \in G$ and $y \in \mathrm{USS}(x)$. For a simple element $s \in S$,

there is a unique simple element which is minimal w.r.t $\preccurlyeq$, denoted[2] $\mathfrak{c}_s = \mathfrak{c}_s(y)$, such that $s \preccurlyeq \mathfrak{c}_s \preccurlyeq \Delta$ and $\mathfrak{c}_s^{-1}\, y\, \mathfrak{c}_s \in \mathrm{USS}(x)$. Notice the similarity between $\mathfrak{c}_s$ and ρ_a from the previous chapter. ρ_a is the minimal element in $S_{\mathcal{P}_y^{\mathrm{SSS}}} \cap mult(a)$ where a is an atom. If we let $S_{\mathcal{P}_y^{\mathrm{USS}}}$ be the set of simple elements which conjugate $y \in \mathrm{USS}(x)$ to another element of $\mathrm{USS}(x)$, then $\mathfrak{c}_a$ is the minimal element in $S_{\mathcal{P}_y^{\mathrm{USS}}} \cap mult(a)$ if a is an atom of M. The main difference is that $\mathfrak{c}_s$ is defined for any simple element s, while ρ_a is only defined for a an atom. Gebhardt does also define $\rho_s = \rho_s(y)$ for $s \in S$, and his algorithm for minimal simple elements references ρ_s where $s \in S$. However, he makes it clear that we find S_y^{USS} by letting s range over the atoms of M, so at the end of the day, we only need ρ_a and $\mathfrak{c}_a$ where a are atoms. Nonetheless, while all the results of [FGM03] may not necessarily hold for ρ_s with $s \in S$, the algorithms which compute ρ_a in [FGM03] will still work if we input a simple element instead of an atom.

We also have[3] $S_y^{\mathrm{USS}} = \min(S_{\mathcal{P}_y^{\mathrm{USS}}})$, the set of minimal simple elements which conjugate $y \in \mathrm{USS}(x)$ to another element in $\mathrm{USS}(x)$ (we let $\mathcal{P}_y^{\mathrm{USS}}$ be the property "conjugates $y \in \mathrm{USS}(x)$ to another element in $\mathrm{USS}(x)$" for simple elements). As before, the cardinality of S_y^{USS} is at most the number of atoms in M. Notice that Theorem 5.1.3 assures us that the property $\mathcal{P}_y^{\mathrm{USS}}$ is closed under gcd – $S \subseteq M$, and the gcd of two simple elements is also simple.

Finding $\mathfrak{c}_s(y)$ and S_y^{USS} for an element $y \in \mathrm{USS}(x)$ is of high importance: we need to be able to find the ultra summit set without having to compute the entire super summit set; hence, being able to find simple elements which conjugate a representative of $\mathrm{USS}(x)$ to other elements of $\mathrm{USS}(x)$ will allow us to determine the rest of $\mathrm{USS}(x)$, without finding the entire $\mathrm{SSS}(x)$.

Gebhardt also states that if we have some non-empty subset U of $\mathrm{USS}(x)$, we can find the rest of $\mathrm{USS}(x)$ by conjugating the elements of U with minimal simple elements, i.e., $\mathrm{USS}(x)$ can be computed as the closure of any non-empty subset U of $\mathrm{USS}(x)$ under conjugation by minimal simple elements. In particular,

Corollary 5.1.5 (Corollary 1.21 in [Geb05]). *Let $x \in G$ and $\emptyset \neq U \subseteq \mathrm{USS}(x)$. If $\{c^{-1}\, y\, c \mid y \in U, c \in S_y^{\mathrm{USS}}\} \subseteq U$, then $U = \mathrm{USS}(x)$.*

Thus, once we have a representative of $\mathrm{USS}(x)$, we use minimal simple elements for that representative to find other elements of $\mathrm{USS}(x)$, and then use the new elements to find more elements, until no new elements can be found via conjugation with minimal simple elements.

The next result shows that when we compute the ultra summit set $\mathrm{USS}(x)$ of an element x as the closure of a non-empty subset U of $\mathrm{USS}(x)$ under conjugation by minimal simple elements, it is sufficient to test the conjugates of representatives of trajectories; we do not need to consider every element in a trajectory.

Theorem 5.1.6 (Theorem 1.22 in [Geb05]). *Let $x \in G$, $y \in \mathrm{USS}(x)$, and $z \in T_y$. For any $s \in S_z^{\mathrm{USS}}$, there exists $t \in S_y^{\mathrm{USS}}$ such that $s^{-1}\, z\, s \in T_{t^{-1}\, y\, t}$.*

[2] This is denoted by $c_s = c_s(y)$ in the paper, but I find this to be confusing, as we often use a similar symbol in our algorithms to indicate the conjugator between $\tilde{x}$ and another element.

[3] Gebhardt uses different notation for these sets; in particular, the set of simple elements which conjugate $y \in \mathrm{USS}(x)$ to an element in $\mathrm{USS}(x)$, $S_{\mathcal{P}_y^{\mathrm{USS}}}$, is denoted by D_y in the paper (the set of simple elements is denoted by D); the set of minimal elements in $S_{\mathcal{P}_y^{\mathrm{USS}}}$ is denoted by C_y.

So, if any element in T_y conjugates to an element in a "new" trajectory via a minimal simple element, y will have a minimal simple element conjugating it to an element in the same trajectory.

Corollary 5.1.7 (Corollary 1.23 in [Geb05]). *Let $x \in G$, $\emptyset \neq I \subseteq \mathrm{USS}(x)$, and $U = \bigcup_{y \in I} T_y \subseteq \mathrm{USS}(x)$. If $\{c^{-1}\, y\, c \mid c \in S_y^{\mathrm{USS}}\} \subseteq U$ for all $y \in I$, then $U = \mathrm{USS}(x)$.*

Thus, we have the following algorithm to determine the ultra summit set of an element. We still need to discuss how to find S_y^{USS} (Step 4(c)i), but for the reader referring back, we use Algorithm 5.1.18 under a loop over all the atoms of M to find S_y^{USS}.

Algorithm 5.1.8 (Algorithm 1.24 in [Geb05]).
Input: $x \in G$.
Output: $\mathrm{USS}(x)$.

1. Use Algorithm 5.1.2 to find $\tilde{x} \in \mathrm{USS}(x)$.
2. Use Algorithm 5.1.1 to find $T_{\tilde{x}}$.
3. If $\tilde{x} = \Delta^p$ for some p, then return $\{\Delta^p\}$.
4. While $V \neq W$, do:
   (a) Let $y_1, \ldots, y_m \in V$ such that $V = W \cup T_{y_1} \cup \cdots \cup T_{y_m}$.
   (b) Set $W = V$.
   (c) For $y \in \{y_1, \ldots, y_m\}$, do:
      i. Compute S_y^{USS}.
      ii. For $c \in S_y^{\mathrm{USS}}$, do:
         A. Compute $c^{-1}\, y\, c$ and use Algorithm 5.1.1 to compute $T_{c^{-1}\, y\, c}$.
         B. If $T_{c^{-1}\, y\, c} \notin V$, then set $V = V \cup T_{c^{-1}\, y\, c}$.
5. Return V.

We start with $V = T_{\tilde{x}}$ and $W = \emptyset$. Of course, if $\tilde{x} = \Delta^p$, then it has normal length 0 and infimum p, and is the only element such, so $\mathrm{SSS}(x) = \mathrm{USS}(x) = \{\Delta^p\}$ in this case. If this is not the case, we continue.

Clearly $T_{\tilde{x}} \neq \emptyset$ (i.e. $V \neq W$ indeed), so we enter the while-loop. Now, $V = \emptyset \cup T_{\tilde{x}}$, so we will only consider $\tilde{x}$ in the for-loop. Also, we update W to equal V, which will allow us to keep track of whether new elements are found in the for-loop of Step 4(c)ii. Next, we find $S_{\tilde{x}}^{\mathrm{USS}}$ and use this set to find conjugates of $\tilde{x}$ in $\mathrm{USS}(x)$ as well as the trajectories of these conjugates, which are added to V. If no new elements are found (i.e. for every $c \in S_{\tilde{x}}^{\mathrm{USS}}$, $c^{-1}\, \tilde{x}\, c = \mathbf{c}^i(\tilde{x})$ for some i), we will not be able to enter the while-loop again, so $\mathrm{USS}(x) = T_{\tilde{x}}$. However, if new elements are indeed found, we enter the while-loop yet again. At this stage $W = T_{\tilde{x}}$, so

only the elements which were added to V in the previous while-loop and are not in $T_{\tilde{x}}$ will be considered, and we will not look at the entire trajectory of these new elements. If, for example, $S^{\mathrm{USS}}_{\tilde{x}} = \{c_1, c_2\}$ in the previous loop, and $c_1^{-1}\,\tilde{x}\,c_1 \neq \mathbf{c}^i(\tilde{x})$ for any i and similarly for c_2, then we added $T_{c_1^{-1}\,\tilde{x}\,c_1}$ and $T_{c_2^{-1}\,\tilde{x}\,c_2}$ to V. In particular, $V = T_{\tilde{x}} \cup T_{c_1^{-1}\,\tilde{x}\,c_1} \cup T_{c_2^{-1}\,\tilde{x}\,c_2} = W \cup T_{y_1} \cup T_{y_2}$, so we will consider $y_1 = c_1^{-1}\,\tilde{x}\,c_1$ and $y_2 = c_2^{-1}\,\tilde{x}\,c_2$ (and update W to keep track of new elements) and evaluate the for-loop as before. Notice that if indeed $c_1^{-1}\,\tilde{x}\,c_1 = \mathbf{c}^i(\tilde{x})$ for some i, then $T_{c_1^{-1}\,\tilde{x}\,c_1} = T_{\tilde{x}}$; thus, each time we find a trajectory distinct from $T_{\tilde{x}}$, we find a circuit in Γ_x which is disjoint from the circuit formed by $T_{\tilde{x}}$.

Gebhardt also provides a nifty algorithm for the conjugacy search problem if we know that two elements are indeed conjugate. Suppose $x, y \in G$ are conjugate. Recall that for the super summit set, we would have to construct a portion of the $\mathrm{SSS}(x)$ (perhaps the whole set) and identify the representative $\tilde{y} \in \mathrm{SSS}(y)$ of y in $\mathrm{SSS}(x)$ before being able to find a conjugating element $c \in G$ such that $c^{-1}\,x\,c = y$. Finding the entire $\mathrm{SSS}(x)$ is quicker with minimal simple elements, but up to the entire set is still required. The algorithm we see now does not require the calculation of the ultra summit set for either x or y.

We do not yet know how to find the element $\mathfrak{c}_s$ in Step 4b, but for the reader referring back, we use Algorithm 5.1.18 to find $\mathfrak{c}_s$. Recall that c_z for $z \in T_{\tilde{x}}$, denotes the associated conjugator in $C_{\tilde{x}} = \{c_z \mid z \in T_{\tilde{x}} \text{ and } c_z^{-1}\,\tilde{x}\,c_z = z\}$.

Algorithm 5.1.9 (Algorithm 3.1 in [Geb05])**.**
Input: $x, y \in G$*, two conjugate elements.*
Output: $c \in G$ *such that* $c^{-1}\,x\,c = y$.

1. Use Algorithm 5.1.2 to find $\tilde{x} \in \mathrm{USS}(x)$ and d such that $d^{-1}\,x\,d = \tilde{x}$, and $\tilde{y} \in \mathrm{USS}(x)$ and b such that $b^{-1}\,y\,b = \tilde{y}$.
2. Use Algorithm 5.1.1 to find $T_{\tilde{x}}$ and $C_{\tilde{x}}$.
3. Set $z = \tilde{y}$ and $s = b$.
4. Loop:
   (a) If $z \in T_{\tilde{x}}$, then return $d \cdot c_z \cdot s^{-1}$.
   (b) Choose a random atom a of M. Compute $\mathfrak{c}_a = \mathfrak{c}_a(z)$.
   (c) Set $z = \mathfrak{c}_a^{-1}\,z\,\mathfrak{c}_a$ and $s = s \cdot \mathfrak{c}_a$.

It is easy to see that if $z = \tilde{y} \in T_{\tilde{x}}$ in the first iteration of the loop, then the conjugating element from x to y is $d \cdot c_{\tilde{y}} \cdot b^{-1}$ since d conjugates x to $\tilde{x}$, $c_{\tilde{y}}$ conjugates $\tilde{x}$ to $\tilde{y}$, and b conjugates $\tilde{y}$ to y.

If $\tilde{y}$ is not immediately in $T_{\tilde{x}}$, we conjugate it with minimal simple elements until we have an element indeed in $T_{\tilde{x}}$ (the first few minimal simple elements we find may conjugate to an element in the same trajectory as $\tilde{y}$). Gebhardt notes that the expected number of iterations

of the loop in this algorithm is the number of circuits in the graph Γ_x. It is not specified how to choose a random atom, but since the number of atoms will always be finite, I suspect one can simply enumerate the atoms and use a random number generator.

Gebhardt does not provide an algorithm to determine whether two elements are conjugate. We can construct an algorithm analogous to the other algorithms solving the conjugacy search and decision problem (where we construct the invariant set for x and check if a representative of y appears as we construct). However, if we first have to determine whether two elements are conjugate, it is not clear whether we should then solve the search problem in the same algorithm by tracking the conjugators (as in previous algorithms), or if we should adjust the decision algorithm to keep track of as few conjugators as possible, and then use the above algorithm to determine the conjugating element.

Each of the algorithms we have seen has one computation we do not know how to perform yet – computing S_y^{USS} in Algorithm 5.1.8 and computing $\mathfrak{c}_a$ in Algorithm 5.1.9. Thus, the rest of this section is dedicated to these minimal simple elements for the ultra summit set. Once we have the algorithm for finding $\mathfrak{c}_s$, we simply loop the algorithm, setting s to a different atom each time, which will produce a superset of S_y^{USS} which has cardinality at most the number of atoms of M.

Let $x \in G$ be an element of its own super summit set (i.e. $x \in \mathrm{SSS}(x)$) with non-zero normal length, so $x = \Delta^p x_1 \dots x_r$ in left-normal form and $r > 0$. The following proposition shows how the normal form of a conjugate of x relates to the normal form of x:

Proposition 5.1.10 (Proposition 2.1 in [Geb05])**.** *Let x be in left-normal form as above and let $u \in M$ such that $u^{-1} x u \in \mathrm{SSS}(x)$. There are elements $u_0, \dots, u_r \in M$ such that the normal form of $u^{-1} x u$ is $\Delta^p(u_0^{-1} x_1 u_1)(u_1^{-1} x_2 u_2) \dots (u_{r-1}^{-1} x_r u_r)$, where the factors in brackets are the simple elements in the normal form of $u^{-1} x u$. Furthermore, $u_0 = \tau^p(u)$, $u_r = u$, and for $i = 1, \dots, r-1$,*

$$u_i = x_{i+1} \dots x_r u \wedge \Delta\tau(x_i^{-1} u_{i-1}).$$

If we look back to the end of Section 4.2, where we used the right complement ∂ to write a factorisation in left-weighted form, we can see that the same process is applied here. Also, note that $u^{-1} x u = u^{-1} \cdot \Delta^p x_1 \dots x_r \cdot u = \Delta^p \tau^p(u^{-1}) x_1 \dots x_r u$. Since $\tau^p(u^{-1}) = \tau^p(u)^{-1}$, we can see why $u_0 = \tau^p(u)$. The intermediate u_i (for $i = 1, \dots, r-1$) force $\Delta^p \tau^p(u^{-1}) x_1 \dots x_r u$ into left-normal from.

This proposition, and in particular the computation of u_1, will be important in finding $\mathfrak{c}_s$. Towards this goal, we have the following key lemma:

Lemma 5.1.11 (Lemma 2.3 in [Geb05])**.** *Let x be in left-normal form as above and $u \in M$ such that $u^{-1} x u \in \mathrm{SSS}(x)$. Let $u_0, \dots, u_r$ be the positive elements obtained by writing $u^{-1} x u$ in left-normal form as described in Proposition 5.1.10. Let $\phi_x(u) = \tau^{-p}(u_1)$.*

(i) $\phi_x(u) \in M$ *satisfies* $\boldsymbol{c}(u^{-1} x u) = (\phi_x(u))^{-1}\, \boldsymbol{c}(x)\, \phi_x(u)$.

(ii) $\sup \phi_x(u) \leq \sup(u)$. *In particular, if u is simple then $\phi_x(u)$ is simple.*

(iii) The conjugating element along any path in the diagram below only depends on the starting point and the end point on the path. (Horizontal arrows indicate cycling.)

$$\begin{array}{ccc} x & \xrightarrow{\tau^{-p}(x_1)} & \boldsymbol{c}(x) \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle \phi_x(u)} \\ u^{-1}xu & \xrightarrow[\tau^{-p}(u_0^{-1}x_1u_1)]{} & \boldsymbol{c}(u^{-1}xu) \end{array}$$

Thus, if u conjugates x to $u^{-1}\,x\,u \in \mathrm{SSS}(x)$, then $\phi_x(u)$ conjugates $\mathbf{c}(x)$ to $\mathbf{c}(u^{-1}\,x\,u)$. We call $\phi_x(u)$ the **transport** of u along $x \to \mathbf{c}(x)$. Given what we know from Lemma 5.1.11 and Proposition 5.1.10 as well as statement (v) of Lemma 4.2.2, we can write

$$\begin{aligned} \phi_x(u) = \tau^{-p}(u_1) &= \tau^{-p}(x_2 \dots x_r u \wedge \Delta\tau(x_1^{-1}u_0)) \\ &= \tau^{-p}(x_2 \dots x_r u) \wedge \Delta\tau^{1-p}(x_1^{-1})\tau(u). \end{aligned} \tag{5.1}$$

This is due to $\tau^{-p}(\tau(x_1^{-1})) = \tau^{1-p}(x_1^{-1})$ and $\tau^{-p}(\tau(u_0)) = \tau(\tau^{-p}(u_0)) = \tau(u)$, since $u_0 = \tau^p(u)$ implies $\tau^{-p}(u_0) = u$.

If x is obvious from context, we write $u^{(0)} = u$ and the transport of $u^{(0)}$ is $u^{(1)}$. Furthermore, $u^{(i+1)} = (u^{(i)})^{(1)} = \phi_{\mathbf{c}^i(x)}(u^{(i)})$. Thus, we could continue the above diagram to the right and find the conjugating element $u^{(p)}$ between $\mathbf{c}^p(x)$ and $\mathbf{c}^p(u^{-1}\,x\,u)$ for any p; however, we have to know the left-normal form of $\mathbf{c}^{p-1}(x)$ in order to find $u^{(p)}$.

The following result shows that u is in a period under transport:

Lemma 5.1.12 (Lemma 2.6 in [Geb05])**.** *Let x be in left-normal form as above, let $u \in M$ such that $u^{-1}\,x\,u \in \mathrm{SSS}(x)$, and let $\boldsymbol{c}^N(x) = x$ and $\boldsymbol{c}^N(u^{-1}\,x\,u) = u^{-1}xu$ for some integer $N > 0$. There is an integer $m > 0$ such that $u^{(mN)} = u$.*

It is easy to see that m might be 1, but higher values of m are definitely possible.

For an example of computing transports, see Section A.3.

Now, let $x \in G$ be in its own ultra summit set (i.e. $x \in \mathrm{USS}(x)$), with left-normal form as before, and normal length at least 1. Also, let N be the minimal positive integer satisfying $\mathbf{c}^N(x) = x$ (note that this is not the requirement for N in Lemma 5.1.12). Recall the definition of ρ_s. We may be able to find $\mathfrak{c}_s$ by iteratively transporting ρ_s, but it may be impossible to find $\mathfrak{c}_s$ only by transporting ρ_s. In this case, we will derive a suitable element $p_x = p_x(s)$ from ρ_s, and then apply iterated transport to p_x until we reach $\mathfrak{c}_s$. Lemma 5.1.12 tells us $\mathfrak{c}_s$ is in a period under transport, so we will know we have reached $\mathfrak{c}_s$ once this period is reached when we transport p_x. The author points out that since $\mathrm{USS}(x) \subseteq \mathrm{SSS}(x)$, we have $\rho_s \preccurlyeq \mathfrak{c}_s$.

Let $u \in S$ be such that $u^{-1}\,x\,u \in \mathrm{SSS}(x)$ (i.e. $u \in S_{\mathcal{P}_x^{\mathrm{SSS}}}$) and consider the elements $u^{(iN)}$ (these conjugate $\mathbf{c}^{iN}(x)$ to $\mathbf{c}^{iN}(u^{-1}\,x\,u)$). Statement (ii) of Lemma 5.1.11 and S being finite implies there are integers $i_2 > i_1 \geq 0$ such that $u^{(i_1N)} = u^{(i_2N)}$; let i_1 and i_2 be minimal subject to this condition. We define $F_x(u) = \{u^{(iN)} \mid i_1 \leq i < i_2\}$ and $l_x(u) = i_2 - i_1$. $F_x(u)$ is all the multiple-of-N-fold transports of u. Notice that $l_x(u)$ gives us the cardinality of $F_x(u)$.

Gebhardt states that $1 \in F_x(u) \Leftrightarrow F_x(u) = \{1\}$. The $\Leftarrow$ aspect of this statement is trivial, but the $\Rightarrow$ is interesting (and not proven in the paper). If $1 \in F_x(u)$, then $u^{(jN)} = 1$ for

some $i_1 \leq j < i_2$, so $\mathbf{c}^{jN}(x) = \mathbf{c}^{jN}(u^{-1}\,x\,u)$. Since $\mathbf{c}^N(x) = x$, $\mathbf{c}^{jN}(x) = x$ (for any j), and so $x = \mathbf{c}^{jN}(u^{-1}\,x\,u)$ too. If we transport $u^{(jN)} = 1$ to get $u^{(jN+1)}$, we find the conjugator between $\mathbf{c}(x)$ and itself, which will also be 1. $u^{(jN+2)}$ will also be 1, the conjugator between $\mathbf{c}^2(x)$ and itself. Thus, no new transports will be found if we continue. Furthermore, by our assumption $i_1 \leq j < i_2$ and by our discussion $u^{(i_2 N)} = 1$, since $i_2 > j$, which means $u^{(i_1 N)} = 1$ as well, and so $j = i_1$. Since we do not write down repeated elements in a set, $F_x(u) = \{1\}$.

The set $F_x(u)$ will be crucial in our search for $\mathfrak{c}_s$. To start, it allows us to find an element which conjugates x to an element in $\mathrm{USS}(x)$. This element is not necessarily minimal, however.

Lemma 5.1.13 (Lemma 4.2 in [Geb05]). *Let $u \in S$ such that $u^{-1}\,x\,u \in \mathrm{SSS}(x)$, let $v \in F_x(u)$ and let $l = l_x(u)$. Then $v^{(ilN)} = v$ for all integers $i > 0$. Moreover, $v^{-1}\,x\,v \in \mathrm{USS}(x)$.*

Lemma 5.1.14 (Lemma 4.3 in [Geb05]). *Let $s \in S$. If $\mathfrak{c}_s \preccurlyeq \mathfrak{c}_s^{(iN)}$ for some $i > 0$, then $\mathfrak{c}_s^{(iN)} = \mathfrak{c}_s$.*

Lemma 5.1.15 (Lemma 4.4 in [Geb05]). *Let $p, s \in S$ satisfy $p \preccurlyeq \mathfrak{c}_s$ and $p^{-1}\,x\,p \in \mathrm{SSS}(x)$. Consider $F_x(p)$.*

(i) If there exists $v \in F_x(p)$ such that $s \preccurlyeq v$, then $\mathfrak{c}_s = v$.

(ii) If $F_x(p) \neq \{1\}$ and $s \not\preccurlyeq v$ for all $v \in F_x(p)$ then $\mathfrak{c}_s$ is not minimal in $S_{P_x^{\mathrm{USS}}}$.

Note that checking whether $s \preccurlyeq v$ amounts to checking whether $s^{-1}v$ is positive.

We will use Lemma 5.1.15 in the algorithm to find $\mathfrak{c}_s$ by finding ρ_s for whichever $s \in S$ and then using iterated transport on ρ_s to find $F_x(\rho_s)$. Then one of three things can happen: (1) $F_x(\rho_s)$ contains an element which is a multiple of s; (2) $F_x(\rho_s)$ contains no multiples of s and is not the set $\{1\}$; or (3) $F_x(\rho_s) = \{1\}$. In the first case, we have found $\mathfrak{c}_s$. In the second case, we need to start over with a different simple element, say s_1, because the element $\mathfrak{c}_s$ that satisfies $s \preccurlyeq \mathfrak{c}_s \preccurlyeq \Delta$ and $\mathfrak{c}_s^{-1}\,x\,\mathfrak{c}_s \in \mathrm{USS}(x)$ is not a minimal simple element, i.e. $\mathfrak{c}_s \notin S_x^{\mathrm{USS}}$. In the last case, we need to find a different element p_x (related to ρ_s) so that we can find a different set of iterated transports $F_x(p_x)$ such that the existence of $v \in F_x(p_x)$ satisfying $s \preccurlyeq v$ is guaranteed. In Section A.3 we discuss an example provided in the paper where $F_x(\rho_s) = \{1\}$. Our discussion of this same example in Section B.2.2 also explains a correction to a definition and a proposition in the paper. The corrected versions are presented here.

Let $s \in S$ and let $y \in \mathrm{USS}(x)$ (we still assume $x \in \mathrm{USS}(x)$; y can be any element, including x, in $\mathrm{USS}(x)$). There exists a unique minimal element $\pi_y(s)$ satisfying $\pi_y(s)^{-1}\,y\,\pi_y(s) \in \mathrm{SSS}(x)$ and $s \preccurlyeq \phi_y(\pi_y(s))$. $\pi_y(s)$ is called the **pullback** of s along $y \to \mathbf{c}(y)$. So, $\pi_y(s)$ conjugates y to $\pi_y(s)^{-1}\,y\,\pi_y(s) \in \mathrm{SSS}(x)$ and if we transport $\pi_y(s)$ (i.e. find the conjugator from $\mathbf{c}(y) \in \mathrm{USS}(x)$ to $\mathbf{c}(\pi_y(s)^{-1}\,y\,\pi_y(s)) \in \mathrm{SSS}(x)$), we obtain a multiple of s.

If y is obvious from context, we define $s_{(0)} = s$ and $s_{(i+1)} = \pi_{\mathbf{c}^\alpha(y)}(s_{(i)})$ for $i \geq 0$, where $0 \leq \alpha \equiv -(i+1) \mod N$ (where N is the minimal positive integer such that $\mathbf{c}^N(x) = x$). Thus, $s_{(i+1)}$ will satisfy $s_{(i+1)}^{-1}\,\mathbf{c}^\alpha(y)\,s_{(i+1)} \in \mathrm{SSS}(x)$. The following proposition shows us how to find $\pi_y(s)$:

Proposition 5.1.16 (Proposition 4.7 in [Geb05]). *Let $s \in S$ and let $y \in \mathrm{USS}(x)$ have left-normal form $\Delta^p y_1 \dots y_r$. Define*

$$b = \left(1 \vee \tau^{-p}(y_1)\, s\, \Delta^{-1}\right) \vee \left(1 \vee y_r^{-1} \dots y_2^{-1} \tau^p(s)\right).$$

Then $b \in S$ and $\rho_b = \pi_y(s)$.

Note that $1 \vee q$ where $q \in G$ will be the smallest (w.r.t. $\preccurlyeq$) multiple (by positive multiplication) of q which is positive, since $1 \preccurlyeq 1 \vee q$ means $1p = 1 \vee q$ where $p \in M$. Now we can use the pullback to find p_x.

Proposition 5.1.17 (Proposition 4.9 in [Geb05], corrected). *Let $s \in S$ and consider the elements $s_{(iN)}$, for $1 \geq 0$, obtained by applying the pullback definition for $y = \boldsymbol{c}^N(x)$. S is finite, so there are integers $i_2 > i_1 \geq 0$ such that $s_{(i_1 N)} = s_{(i_2 N)}$. Choose minimal values for i_1 and i_2, let $l = i_2 - i_1$, and choose an integer j such that $jl \geq i_1$. Let $p_x = p_x(s) = s_{(jlN)}$. Then $p_x \preccurlyeq \mathfrak{c}_s$ and there exists $v \in F_x(p_x)$ with $s \preccurlyeq v$. In particular, $v = \mathfrak{c}_s$.*

Since $\mathbf{c}^N(x) = x$, p_x will satisfy $p_x^{-1}\, x\, p_x \in \mathrm{SSS}(x)$.

After this proposition, Gebhardt gives an example (which we discuss in Section A.3), after which he concludes that in this case, $p_x \notin F_x(p_x)$, showing that iterated transport of p_x is necessary even after reaching a stable loop under iterated pullback of s.

Finally, we have the algorithm to find $\mathfrak{c}_s$ for $y \in \mathrm{USS}(x)$. Part of the input of this algorithm is a boolean value `f`, which tells the algorithm whether elements which are known not to be minimal in $S_{\mathcal{P}_y^{\mathrm{USS}}}$ should be discarded (when `f` = `true`). If we set `f` = `false`, we will find an element which does indeed conjugate y to an element in $\mathrm{USS}(x)$, but it will not be minimal.

Algorithm 5.1.18 (Algorithm 4.12 in [Geb05]).
Input: `f` *a boolean value, $y \in \mathrm{USS}(x)$ (for some $x \in G$), $s \in S$, $p = \inf y$, $t = \ell(y)$.*
Output: $\mathfrak{c}_s$ *or* `not minimal`.

1. Compute ρ_s using Algorithm 4.3.12 and compute $F_y(\rho_s)$.
2. If $\exists v \in F_y(\rho_s)$ such that $s \preccurlyeq v$, then return v.
3. If `f` and $F_y(\rho_s) \neq \{1\}$, then return `not minimal`.
4. Compute $p_y(s)$ and $F_y(p_y(s))$.
5. Choose $v \in F_y(p_y(s))$ such that $s \preccurlyeq v$.
6. Return v.

We can find a superset of S_y^{USS} using the above algorithm with `f` = `true`, by letting s range over all the atoms of M. The author notes that the cardinality of this superset of S_y^{USS} will be bounded by the number of atoms of M. A further remark indicates that we could skip

Steps 2 and 3 and start at Step 4 (thanks to Proposition 5.1.17), but computing pullbacks is relatively expensive and often unnecessary, so we first try with transports.

The author offers empirical results at the end of this paper, based on the braid group. He does not give a big-$\mathcal{O}$ time estimation for the algorithm, but it is quite clear that it is more efficient than the previous methods we've discussed (in particular, Gebhardt compares computing the ultra summit set to computing the super summit set using the methods of Franco and González-Meneses). As n gets larger than 3, the ultra summit set has substantially fewer elements than the super summit set. For $n = 3$, we encounter $|\mathrm{SSS}(x)| = |\mathrm{USS}(x)|$, but the average time taken to compute $\mathrm{SSS}(x)$ is consistency bigger than the time taken to compute $\mathrm{USS}(x)$, and this feature remains as n grows. See Section 5 of [Geb05] for the detailed results of Gebhardt's computer experiments.

5.2 Partial cycling and partial twisted decycling

Shortly after the publication of the ultra summit set result, three authors came together and published three highly significant papers. These papers form part of their over-arching project of finding a polynomial-time solution to the conjugacy problem in Garside groups. We have already encountered these authors on our journey – Joan Birman, Volker Gebhardt, and Juan González-Meneses. The papers are *Conjugacy in Garside groups I: cyclings, powers and rigidity* [BGGM07a], *Conjugacy in Garside groups II: structure of the ultra summit set* [BGGM08], and *Conjugacy in Garside groups III: periodic braids* [BGGM07b]. The titles make it clear that this is a trilogy of papers. One may even go so far as to call this the Lord of the Rings of the braid conjugacy problem. And hence, dear reader, I cannot even summarise the abundance of results proven over the course of these three papers which totals 139 pages. Thus, we discuss only one aspect of their work. In explaining my choice of discussion topic, I hope to give a taste of the other key results, and urge you, reader, to experience these papers for yourself. They are easy to read and show fascinating results, but we simply cannot discuss all of it here.

Our focus is on the improvement to the ultra summit set solution, which holds for all elements of a Garside group. The other improvements to the conjugacy solution are each for special cases. Below is a brief overview of these special cases.

An element $x \in G$, with left-normal form $x = \Delta^p x_1 \dots x_r$ is **rigid** if $\Delta^p x_1 \dots x_r \tau^{-p}(x_1)$ is in left-normal form as written, which implies $\mathbf{c}(x) = \Delta^p x_2 \dots x_r \tau^{-p}(x_1)$ is also in left-normal form as written. If x is rigid, $\mathbf{c}^i(x)$ and $\mathbf{d}^i(x)$ are rigid for every $i \geq 1$. Thus, if an element is rigid, it takes less work to cycle and decycle it, since there will be no need for converting elements into their left-normal form. Finding a representative of the super summit set or ultra summit set of a rigid element is thus quicker than for elements which are not rigid. In fact, if x is rigid with $\ell(x) > 1$, then every element in $\mathrm{USS}(x)$ is also rigid. The solution we discuss can be refined further for rigid elements, but we do not discuss this refinement.

We can classify braids into three disjoint classes: periodic, reducible, and pseudo-Anosov (or PA). This classification is often called the Thurston-Nielsen trichotomy for braids. Let $X \in B_n$. If X satisfies $X^m = \Delta^k$ for some non-zero integers m and k, then X is **periodic**. If X is such that we can "combine" some number of strands which start and end adjacent to

one another into a "tube", so that the braid we obtain consists of braided tubes, then it is **reducible**. This can be done again and again until we obtain an irreducible braid, which is then periodic or PA. Every conjugate of a reducible braid is also reducible [exa22]. Figure 5.1 shows an illustration of a reducible braid and its corresponding tubular braid from [GM03]. Lastly, if a braid is neither reducible nor periodic, it is **pseudo-Anosov**.

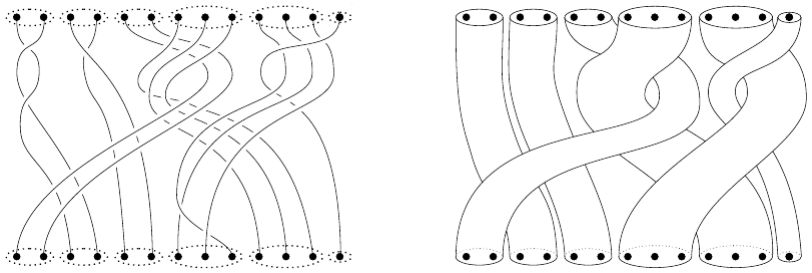


Figure 5.1

In [BGGM07a] the authors show if a braid $X \in B_n$ is pseudo-Anosov, then for some small power m, $\mathrm{USS}(X^m)$ consists of rigid braids. Furthermore, if $Y \in B_n$ is also PA, then it suffices to solve the conjugacy problem for X^m and Y^m – nothing is lost when passing to powers. This is due to the fact (shown in [GM03]) that for arbitrary braids $A, B \in B_n$ and every non-zero integer m, A and B are conjugate if and only if A^m and B^m are conjugate; PA braids also have unique roots, and so if X and Y are PA and $Z \in B_n$ conjugates X^m to Y^m, then Z also conjugates X to Y.

For periodic braids, the authors find a polynomial-time solution to the conjugacy problem, presented in [BGGM07b]. The existence of a polynomial-time algorithm for just a subset of braids is exciting news! As part of their project, the authors also investigate cycling and powers of elements thoroughly in [BGGM07a].

We discuss Section 2 of [BGGM08], in which the authors improve upon the ultra summit set solution for arbitrary elements. To do so, they introduce two new conjugation operations to be performed on elements: partial cycling and partial twisted decycling. Using these operations, they show how the directed graph γ_x can be decomposed into "black" and "grey" subgraphs, which are used to improve the conjugacy solution. The crux of the improved solution is that it is quicker to determine one of these components than the entire USS. In particular, x and y are conjugate if and only if the intersection of the black component of x and the grey component of y is non-empty. There are some cases for which this solution takes just as long as the ultra summit set solution, but in general it is an improvement.

The definition of γ_x is quite different from the graph Γ_x which we saw in the previous section. We use γ_x to denote this new graph, even though the authors reuse Γ_x in their notation. Recall for $x \in G$, the directed graph Γ_x has the set $\mathrm{SSS}(x)$ as its set of vertices, and its directed edges are $\{(y, \mathbf{c}(y)) \mid y \in \mathrm{SSS}(x)\}$. In this paper, the directed graph γ_x describes the entire set $\mathrm{USS}(x)$: each vertex represents an element $y \in \mathrm{USS}(x)$, and for every $y \in \mathrm{USS}(x)$ and every minimal simple element s for y w.r.t. $\mathrm{USS}(x)$ (i.e. for every $s \in S^{\mathrm{USS}}_y$), there is an arrow labelled by s going from y to $s^{-1}\, y\, s$. Thus, in this definition, the edges go between y and $s^{-1}\, y\, s$, instead of y and $\mathbf{c}(y)$. In Theorem 1.13 of this paper, the authors state that this

graph is connected[4]

We will return to γ_x in a moment. We first consider some definitions. For $x \in G$, the **twisted decycling** of x is $\tau(\mathbf{d}(x))$, where $\mathbf{d}(x)$ is the decycling operation as usual. The name comes from the authors referring to τ as the twisting automorphism.

We will encounter a lot of interplay between x and x^{-1}, so recall that if $x = \Delta^p x_1 \dots x_r$ in left-normal form, then the left-normal form of x^{-1} is

$$x^{-1} = \Delta^{-p-r} x'_r \dots x'_1 \text{ where } x'_i = \tau^{-p-i}(\partial(x_i)).$$

Also recall that for $s \in S$, the right complement of s is $\partial(s) = s^{-1}\Delta$ and the left complement of s is $\partial^{-1}(s) = \Delta s^{-1}$. Furthermore, since $\ell(y^{-1}) = \ell(y)$ for arbitrary $y \in G$, we have that

$$SSS(x^{-1}) = \{y^{-1} \mid y \in SSS(x)\}.$$

For $x \in G$ with normal form as above and $\ell(x) > 0$, we define the **initial factor** of x as

$$\iota(x) = \tau^{-p}(x_1).$$

This is the conjugating factor between x and $\mathbf{c}(x)$. The **final factor** of x is

$$\varphi(x) = x_r,$$

which is the inverse of the conjugating factor between x and $\mathbf{d}(x)$. If $\ell(x) = 0$, we define $\iota(x) = \iota(\Delta^p) = 1$ and $\varphi(x) = \varphi(\Delta^p) = \Delta$. Equivalently, we can also write $\iota(x) = x\Delta^{-p} \wedge \Delta$ and $\varphi(x) = (\Delta^{p+r-1} \wedge x)^{-1}x$; these do not require knowing the complete left-normal form of x, only the values of inf and sup.

A result from the first paper in the trilogy is highlighted at the start of the section we are considering. For the proof of this, see Lemma 1.8 in [BGGM07a].

Lemma 5.2.1 (Lemma 2.1 in [BGGM08] or Lemma 1.8 in [BGGM07a])**.** *For $x \in G$,*

$$\iota(x^{-1}) = \partial(\varphi(x)) \text{ and } \varphi(x^{-1}) = \partial^{-1}(\iota(x)).$$

Since $\partial^{-1}(\iota(x)) = \Delta(\iota(x))^{-1}$ and $\partial(\varphi(x)) = \varphi(x)^{-1}\Delta$, this lemma means that

$$\varphi(x)\iota(x^{-1}) = \Delta = \iota(x)\varphi(x^{-1}),$$

as remarked by the authors following the proof of this lemma in [BGGM07a].

With Lemma 5.2.1 in mind, and recalling that τ is conjugation by Δ, we have the following:

[4]When we deal with undirected graphs, we say that a graph is **connected** if there is a path (a sequence of vertices and edges with no repeats) between any pair of vertices. However, there are multiple notions of connectivity when it comes to directed graphs (which γ_x is). It is not made clear which sense of connectivity the authors are referring to here: a directed graph is **weakly connected** if, upon replacing all the directed edges with undirected edges, the resulting (undirected) graph is connected; if there is a directed path from u to v *or* a directed path from v to u for all vertices u, v, then the directed graph is **semiconnected**; a directed graph is **strongly connected** if there is a directed path from u to v *and* a directed path from v to u for all vertices u, v. We will see that that γ_x is strongly connected.

Lemma 5.2.2 (Lemma 2.2 in [BGGM08]). *Let $x \in G$ with $\ell(x) > 0$. Conjugating x by $\iota(x)$ gives $\boldsymbol{c}(x)$. Conjugating x by $\varphi(x)^{-1}$ gives $\boldsymbol{d}(x)$. Conjugating x by $\iota(x^{-1}) = \partial(\varphi(x))$ gives $\tau(\boldsymbol{d}(x))$.*

Note that $(\partial(\varphi(x)))^{-1}\, x\, \partial(\varphi(x)) = \Delta^{-1}\varphi(x)\, x\, \varphi(x)^{-1}\Delta$, which is why the last statement holds.

A **partial cycling** of x is a conjugation of x by a prefix of $\iota(x)$. A **partial twisted decycling** of x is a conjugation of x by a prefix of $\iota(x^{-1}) = \partial(\varphi(x))$. The following results allow us to use partial cycling and partial twisted decycling to decompose γ_x into the black and grey subgraphs mentioned earlier.

Lemma 5.2.3 (Lemma 2.4 in [BGGM08]). *The automorphism τ commutes with both cycling and decycling, i.e. $\tau(\boldsymbol{c}(x)) = \boldsymbol{c}(\tau(x))$ and $\tau(\boldsymbol{d}(x)) = \boldsymbol{d}(\tau(x))$ for all $x \in G$. Moreover, the ultra summit set of an element is closed under τ, $\boldsymbol{c}$, and $\boldsymbol{d}$. In other words, for every $y \in \mathrm{USS}(x)$, we have $\tau(y) \in \mathrm{USS}(x)$, $\boldsymbol{c}(y) \in \mathrm{USS}(x)$, and $\boldsymbol{d}(y) \in \mathrm{USS}(x)$.*

Theorem 5.2.4 (Theorem 2.5 in [BGGM08]). *Let $x \in \mathrm{USS}(x)$ and let $s \in S_x^{\mathrm{USS}}$. Then one and only one of the following conditions hold:*

(i) $\varphi(x)s$ is a simple element.

(ii) $\varphi(x)s$ is left-weighted as written.

Proposition 5.2.5 (Proposition 2.6 in [BGGM08]). *Let $x \in \mathrm{SSS}(x)$ with $\ell(x) > 0$ and let $s \in S_x^{\mathrm{SSS}}$. If $\varphi(x)s$ is left-weighted as written, then $s \preccurlyeq \iota(x)$.*

Corollary 5.2.6 (Corollary 2.7 in [BGGM08]). *Let $x \in \mathrm{USS}(x)$ with $\ell(x) > 0$ and let $s \in S_x^{\mathrm{USS}}$. Then s is a prefix of either $\iota(x)$ or $\iota(x^{-1})$, or both.*

For the sake of understanding these cases better, we consider the proof of this corollary:

Proof. Theorem 5.2.4 tells us that $s \in S_x^{\mathrm{USS}}$ means $\varphi(x)s$ is either left-weighted as written or simple. In the first case, Proposition 5.2.5 makes it clear that $s \preccurlyeq \iota(x)$ (simply note that if $s \in S_x^{\mathrm{USS}}$, then $s \in S_x^{\mathrm{SSS}}$ also by $\mathrm{USS}(x) \subseteq \mathrm{SSS}(x)$, and so the proposition is applicable). On the other hand, if $\varphi(x)s$ is simple, it means $\varphi(x)s \preccurlyeq \Delta$, which means $s \preccurlyeq \varphi(x)^{-1}\Delta$. Noticing that $\varphi(x)^{-1}\Delta = \partial(\varphi(x)) = \iota(x^{-1})$ (by Lemma 5.2.1), we have that $s \preccurlyeq \iota(x^{-1})$ if $\varphi(x)s$ is simple.

Having $s \preccurlyeq \iota(x)$ and $s \preccurlyeq \iota(x^{-1})$ at the same time is only possible in the case where $\varphi(x)s$ is simple. □

The last statement in this proof is not explained in the paper, and the exact conditions under which $s \preccurlyeq \iota(x)$ and $s \preccurlyeq \iota(x^{-1})$ are not discussed in this proof. However, we can tell from the examples in the paper (which we discuss briefly in Section A.4), that $\varphi(x)s$ being simple does not imply $s \preccurlyeq \iota(x)$ and $s \preccurlyeq \iota(x^{-1})$. $\varphi(x)s$ being simple is a necessary but not sufficient condition for $s \preccurlyeq \iota(x)$ and $s \preccurlyeq \iota(x^{-1})$.

Lastly, the **convexity property**:

Corollary 5.2.7 (Corollary 2.10 in [BGGM08]). *Given $x, y \in \mathrm{USS}(x)$, there exists a sequence of partial cycling and partial twisted decyclings joining x to y.*

When we consider γ_x and corollary 5.2.6, there are two kinds of minimal simple elements, and so there are two kinds of arrows in γ_x. For a vertex y, we say that an arrow s, starting at vertex y, is **black** if $s \preccurlyeq \iota(y)$ (i.e. corresponds to a partial cycling of y) and **grey** if $s \preccurlyeq \iota(y^{-1})$ (i.e. corresponds to a partial twisted decycling of y). An arrow can also be bi-coloured, in the case where $s \preccurlyeq \iota(y)$ *and* $s \preccurlyeq \iota(y^{-1})$, but we will not think of this as a third kind of arrow, rather as a black arrow and a grey arrow coinciding.

As mentioned, we briefly discuss the examples shown in the paper in Section A.4. These are all done for braids, and are helpful in terms of visualising a lot of the results we encounter.

The next step towards our algorithm is investigating black and grey arrows and components in more detail.

Let $\gamma_x^{\mathrm{black}}$ (respectively γ_x^{grey}) be the subgraph[5] of γ_x having the same set of vertices as γ_x, but only the black (resp. grey) arrows. These subgraphs are not necessarily connected, i.e. there could be two vertices $u, v \in \gamma_x^{\mathrm{black}}$ (resp. $u, v \in \gamma_x^{\mathrm{grey}}$) with no path from u to v or from v to u. This is clear if we consider Examples A.4.1, A.4.2, and A.4.4 Section A.4. Thus, we cannot generate the whole $\mathrm{USS}(x)$ with only partial cyclings or only partial twisted decyclings; both types of conjugation will be required.

The **black components** of γ_x are the connected components[6] of $\gamma_x^{\mathrm{black}}$, and are denoted by $\mathcal{B}_1, \ldots, \mathcal{B}_s$. Similarly, the **grey components** of γ_x are denoted by $\mathcal{G}_1, \ldots, \mathcal{G}_t$ and are the connected components of γ_x^{grey}. Furthermore, for $y \in \mathrm{USS}(x)$, we denote the black (resp. grey) component of γ_x which contains y as a vertex by $\mathcal{B}_y$ (resp. $\mathcal{G}_y$).

An arrow in γ_x is characterized by its **starting point** $y \in \mathrm{USS}(x)$ and its **label** s, which is a minimal simple element for y. The **endpoint** of the arrow is $s^{-1} y s$. We denote the arrow with label s by s, to simplify notation. We can also consider s^{-1} as an arrow – if y is the starting point of s and z is the endpoint, then z is the starting point of s^{-1} and y the endpoint. We will define a **path** in γ_x as a sequence $(s_1^{e_1}, \ldots, s_k^{e_k})$, possibly empty, where s_i is an arrow in γ_x and $e_i = \pm 1$, such that the endpoint of $s_i^{e_i}$ is equal to the starting point of $s_{i+1}^{e_{i+1}}$ for every $i = 1, \ldots, k-1$. A path is **black** (resp. **grey**) if all of its arrows are black (resp. grey). Lastly, a path $(s_1^{e_1}, \ldots, s_k^{e_k})$ is **oriented** if $e_i = 1$ for all $i = 1, \ldots, k$.

Every path $(s_1^{e_1}, \ldots, s_k^{e_k})$ has an associated element $\alpha = s_1^{e_1} \ldots s_k^{e_k}$. Different paths may have the same associated element. If the path is oriented, then $\alpha \in M$, since the labels of arrows are simple elements. If x and y are the initial and final vertices of the path above, i.e. x is the starting point of the arrow $s_1^{e_1}$ and y is the endpoint of the arrow $s_k^{e_k}$, then $\alpha^{-1} x \alpha = y$.

We need one last definition before we can look at the results regarding paths in γ_x. For an

[5] A subgraph of a graph is another graph formed from a subset of the vertices and edges of the original graph. The vertex subset must include all start- and endpoints of the edge subset, but may also include additional vertices. In the case we are considering, the subgraphs have the same vertex set as the original graph, but only a subset of the edges of the original graph.

[6] A connected component of a graph is a maximal connected subgraph, where "maximal" is understood as follows: say H_1 is a connected subgraph of the graph H; then H_1 is a maximal connected subgraph if any subgraph of H that properly contains H_1 as a subgraph is not connected.

element $y \in G$ whose left-normal form is $\Delta^p y_1 \dots y_r$, we define

$$y^\circ = y\Delta^{-p} = \Delta^p\, y_1 \dots y_r\, \Delta^{-p} = \tau^{-p}(y_1 \dots y_r).$$

Proposition 5.2.8 (Proposition 2.16 in [BGGM08]). *Let $(s_1, \dots, s_k)$ be an oriented path in γ_x starting at vertex y and let $\alpha = s_1 \dots s_k \in M$ be its associated element. Then,*

(*i*) *If $\alpha \preccurlyeq y^\circ$ then $(s_1, \dots, s_k)$ is an oriented black path.*

(*ii*) *If $\alpha \preccurlyeq (y^{-1})^\circ$ then $(s_1, \dots, s_k)$ is an oriented grey path.*

If y has left-normal form as above, then $y^{-1} = \Delta^{-p-r} y'_r \dots y'_1$ in left-normal form where $y'_i = \tau^{-p-i}(\partial(y_i))$. Thus, $(y^{-1})^\circ = y^{-1}\Delta^{-(-p-r)} = \Delta^{-p-r}\, y'_r \dots y'_1\, \Delta^{p+r} = \tau^{p+r}(y'_r \dots y'_1)$.

From the left-normal form of y^{-1}, we know $\iota(y^{-1}) = \tau^{-(-p-r)}(y'_r) = \tau^{p+r}(y'_r)$. Furthermore, $\inf y^\circ = 0$ and $\inf(y^{-1})^\circ = 0$, so $\iota(y^\circ) = \tau^0(\tau^{-p}(y_1)) = \tau^{-p}(y_1) = \iota(y)$ and $\iota((y^{-1})^\circ) = \tau^0(\tau^{p+r}(y'_r)) = \tau^{p+r}(y'_r) = \iota(y^{-1})$. Lastly, for a simple element $s \in S$ we have $s \preccurlyeq y^\circ \Leftrightarrow s \preccurlyeq \iota(y^\circ)$ and $s \preccurlyeq (y^{-1})^\circ \Leftrightarrow s \preccurlyeq \iota((y^{-1})^\circ)$. Putting all of this together with the above proposition, we get the following corollary. This makes clear the connection between black and grey paths and partial cyclings and partial twisted decyclings:

Corollary 5.2.9 (Corollary 2.17 in [BGGM08]). *Let $(s_1, \dots, s_k)$ be an oriented path in γ_x starting at vertex y. If the associated element $s = s_1 \dots s_k \in M$ is simple, then,*

(*i*) *If $s \preccurlyeq \iota(y)$ then $(s_1, \dots, s_k)$ is an oriented black path (so $s^{-1}\, y\, s$ is a partial cycling of y).*

(*ii*) *If $s \preccurlyeq \iota(y^{-1})$ then $(s_1, \dots, s_k)$ is an oriented grey path (so $s^{-1}\, y\, s$ is a partial twisted decycling of y).*

The converse of this statement does not hold; one can have an oriented black (resp. grey) path with simple associated element such that the simple element is *not* a prefix of the initial factor of the starting point (resp. inverse of the starting point) of the path.

The following result is necessary to ensure the black (resp. grey) components of γ_x are connected:

Proposition 5.2.10 (Proposition 2.19 in [BGGM08]). *Given $y \in \mathrm{USS}(x)$ and a black (resp. grey) arrow s in γ_x starting at y, there exists an oriented black (resp. grey) path $(s_1, \dots, s_k)$ in γ_x starting and ending at y, such that $s_1 = s$.*

In other words, if we can "leave" y, we can "return" to y.

Corollary 5.2.11 (Corollary 2.20 in [BGGM08]). *Given two elements y and z in a black component $\mathcal{B}_i$ (resp. grey component $\mathcal{G}_i$) of γ_x, there exists an oriented black (resp. grey) path going from y to z.*

Lastly, we have,

Proposition 5.2.12 (Proposition 2.21 in [BGGM08]). *The set of vertices in a black component of γ_x is a union of orbits under cycling. This is not necessarily true for grey components.*

The authors do not define the term "orbit", but based on their discussion of the examples given in their paper, an **orbit** is a set of elements which form a graph-theoretic cycle under some operation. An orbit under cycling, in particular, can be viewed as a trajectory if the elements are in the ultra summit set, as defined in [Geb05]. Indeed, in the proof of this proposition, the first statement is proven by showing that for every $y \in \mathrm{USS}(x)$, $\mathbf{c}(y)$ is a vertex of $\mathcal{B}_y$. On the other hand, example A.4.2 has four grey components, and these are each orbits under partial twisted decycling, not cycling, illustrating the last statement in Proposition 5.2.12.

Finally, we have the algorithms to compute $\mathcal{B}_x$ and $\mathcal{G}_x$ given $x \in \mathrm{USS}(x)$. These algorithms are similar to previous algorithms we've seen, in that updating the sets V and W and checking whether they are equal ensure that we do not consider the same element twice, and stop when we've constructed the entire set. Note that some additional steps have been inserted for clarity.

Algorithm 5.2.13 (Algorithm 1 in [BGGM08]).
Input: $x \in \mathrm{USS}(x)$, *in left-normal form.*
Output: $\mathcal{B}_x$.

1. List the atoms of M, say $a_1, \ldots, a_\lambda$. Set $V = \{x\}$ and $W = \emptyset$.
2. While $V \neq W$, do:
   (a) Take $y \in V \backslash W$.
   (b) For every atom $a \in \{a_1, \ldots, a_\lambda\}$, do:
      i. If $a \preccurlyeq \iota(y)$, then:
         A. Compute $\mathfrak{c}_a(y)$ using Algorithm 5.1.18. If it returns `not minimal`, then continue to the next atom.
         B. Compute the left-normal form of $z = \mathfrak{c}_a(y)^{-1}\, y\, \mathfrak{c}_a(y)$.
         C. Set $V = V \cup \{z\}$, and store $\mathfrak{c}_a(y)$ as a black arrow going from y to z.
   (c) Set $W = W \cup \{y\}$.
3. Return V together with the information on all black arrows.

Since all elements are in left-normal form, it is easy to find $\iota(y)$ for each y we consider.

In the next algorithm, we avoid computing y^{-1} by taking every atom a such that $\varphi(y)a$ is simple, which the proof of Corollary 5.2.6 tells us implies $a \preccurlyeq \iota(y^{-1})$.

Algorithm 5.2.14 (Algorithm 2 in [BGGM08])**.**
Input: $x \in \mathrm{USS}(x)$*, in left-normal form.*
Output: $\mathcal{G}_x$*.*

1. List the atoms of M, say $a_1, \ldots, a_\lambda$. Set $V = \{x\}$ and $W = \emptyset$.
2. While $V \neq W$, do:
   (a) Take $y \in V \backslash W$.
   (b) For every atom $a \in \{a_1, \ldots, a_\lambda\}$, do:
      i. If $\varphi(y)a \preccurlyeq \Delta$, then:
         A. Compute $\mathfrak{c}_a(y)$ using Algorithm 5.1.18. If it returns `not minimal`, then continue to the next atom.
         B. Compute the left-normal form of $z = \mathfrak{c}_a(y)^{-1}\, y\, \mathfrak{c}_a(y)$.
         C. Set $V = V \cup \{z\}$, and store $\mathfrak{c}_a(y)$ as a grey arrow going from y to z.
   (c) Set $W = W \cup \{y\}$.
3. Return V together with the information on all grey arrows.

Similarly to the previous algorithm, we can find $\varphi(y)$ easily for each y. Also note that $\varphi(y)a \preccurlyeq \Delta$ means $\varphi(y)a$ is simple.

The information on the black and grey arrows is vital to finding a conjugating element when we solve the conjugacy search problem. We have enough knowledge at this point to understand the algorithm which solves the conjugacy search and decision problems, so I will state it here, and then discuss the results that assure it works. As discussed at the start of the section, $x, y \in G$ are conjugate if $\mathcal{B}_x \cap \mathcal{G}_y \neq \emptyset$. We will use this and apply it to representatives of the elements for which we are solving the conjugacy problem.

Algorithm 5.2.15 (Algorithm 3 in [BGGM08])**.**
Input: $x, y \in G$*.*
Output: $c \in G$ *such that* $c^{-1}\, x\, c = y$*, or* `fail` *if* x *and* y *are not conjugate.*

1. Use Algorithm 5.1.2 to compute $\tilde{x} \in \mathrm{USS}(x)$ and a such that $a^{-1}\, x\, a = \tilde{x}$.
2. Use Algorithm 5.1.2 to compute $\tilde{y} \in \mathrm{USS}(y)$ and b such that $b^{-1}\, y\, b = \tilde{y}$.
3. Use Algorithm 5.2.13 to compute $\mathcal{B}_{\tilde{x}}$ and for each vertex v of $\mathcal{B}_{\tilde{x}}$, an element $c_{(\tilde{x},v)}$ conjugating $\tilde{x}$ to v.
4. Use Algorithm 5.2.14 to compute $\mathcal{G}_{\tilde{y}}$ and for each vertex v of $\mathcal{G}_{\tilde{y}}$, an element $c_{(\tilde{y},v)}$ conjugating $\tilde{y}$ to v.

5. If $\mathcal{B}_{\tilde{x}} \cap \mathcal{G}_{\tilde{y}} = \emptyset$, then return `fail`.

6. Choose $v \in \mathcal{B}_{\tilde{x}} \cap \mathcal{G}_{\tilde{y}}$ and return $ac_{(\tilde{x},v)}c_{(\tilde{y},v)}^{-1}b^{-1}$.

Notice that the elements $c_{(\tilde{x},v)}$ and $c_{(\tilde{y},v)}$ found in Steps 3 and 4 (and used in Step 6) will be the arrows stored in Algorithms 5.2.13 and 5.2.14. If a vertex is not one partial cycling or partial twisted decycling away from $\tilde{x}$ or $\tilde{y}$, we need to concatenate arrows to obtain the conjugator to that vertex. It is easy to see that the element returned in Step 6 conjugates x to y, since a conjugates x to $\tilde{x}$, $c_{(\tilde{x},v)}$ conjugates $\tilde{x}$ to v, $c_{(\tilde{y},v)}^{-1}$ conjugates v to $\tilde{y}$, and b^{-1} conjugates $\tilde{y}$ to y.

To show that this algorithm will produce the correct results, the authors show that *every* black component intersects *every* grey component of γ_x, which means that for $x \in \mathrm{USS}(x)$ and $y \in \mathrm{USS}(y)$, x and y are conjugate if and only if $\mathcal{B}_x \cap \mathcal{G}_y \neq \emptyset$. To achieve these results, it is shown that every two elements in $\mathrm{USS}(x)$ can be joined by a grey path followed by a black path and vice versa.

From Proposition 5.2.5 and Corollary 5.2.9, and applying this knowledge to $y^{-1} \in \mathrm{SSS}(x^{-1})$, we get,

Proposition 5.2.16 (Proposition 2.22 in [BGGM08])**.** *Let $(s_1, \ldots, s_k)$ be an oriented path in γ_x starting at a vertex y with $\ell(y) > 0$. If the associated element $s = s_1 \ldots s_k$ is simple, then*

(*i*) *If $\varphi(y)s$ is left-weighted as written (i.e., $s \preccurlyeq \iota(y)$), then $(s_1, \ldots, s_k)$ is an oriented black path.*

(*ii*) *If $\varphi(y^{-1})s$ is left-weighted as written (i.e., $s \preccurlyeq \iota(y^{-1})$), then $(s_1, \ldots, s_k)$ is an oriented grey path.*

For non-simple elements we have,

Proposition 5.2.17 (Proposition 2.23 in [BGGM08])**.** *Let $x \in G$ and $y \in \mathrm{USS}(x)$ with $\ell(y)$. Suppose that $\alpha \in M$ is such that $\inf \alpha = 0$ and $\alpha^{-1}y\alpha \in \mathrm{USS}(x)$.*

(*i*) *If $\varphi(y)\iota(\alpha)$ is left-weighted as written, then α can be decomposed as $\alpha = s_1 \ldots s_k$ where $(s_1, \ldots, s_k)$ is an oriented black path.*

(*ii*) *If $\varphi(y^{-1})\iota(\alpha)$ is left-weighted as written, then α can be decomposed as $\alpha = s_1 \ldots s_k$ where $(s_1, \ldots, s_k)$ is an oriented grey path.*

Now, our two core results:

Theorem 5.2.18 (Theorem 2.24 in [BGGM08])**.** *Let $x \in G$. Given $y, z \in \mathrm{USS}(x)$ with $\ell(y) > 0$, there exists an oriented path $(g_1, \ldots, g_s, b_1, \ldots, b_t)$ in γ_x going from y to z, such that $(g_1, \ldots, g_s)$ is a (possible empty) grey path and $(b_1, \ldots, b_t)$ is a (possible empty) black path. If $z = \alpha^{-1}\, y\, \alpha$ with $\alpha \in M$, then the paths can be chosen such that $g_1 \ldots g_s b_1 \ldots b_t = \alpha$.*

Note that $\ell(y) > 0$ means $\ell(z) > 0$, since y and z are in the same ultra summit set.

Analogously, we also have:

Theorem 5.2.19 (Theorem 2.25 in [BGGM08]). *Let $x \in G$. Given $y, z \in \mathrm{USS}(x)$ with $\ell(y) > 0$, there exists an oriented path $(b_1, \ldots, b_t, g_1, \ldots, g_s)$ in γ_x going from y to z, such that $(b_1, \ldots, b_t)$ is a (possible empty) black path and $(g_1, \ldots, g_s)$ is a (possible empty) grey path. If $z = \alpha^{-1} y \alpha$ with $\alpha \in M$, then the paths can be chosen such that $b_1 \ldots b_t g_1 \ldots g_s = \alpha$.*

Notice that since these two theorems hold for *any* $y, z \in \mathrm{USS}(x)$, we can find an oriented path between y and z, and between z and y. Thus, γ_x is strongly connected in the graph theoretic sense (see footnote 4).

Now it is easy to see that all black components intersect all grey components of γ_x:

Corollary 5.2.20 (Corollary 2.26 in [BGGM08]). *Let $x \in G$. Given $y, z \in \mathrm{USS}(x)$, one has $\mathcal{B}_y \cap \mathcal{G}_z \neq \emptyset$.*

By Theorem 5.2.18, there is an oriented path $(b_1, \ldots, b_t, g_1, \ldots, g_s)$ going from y to z such that $(b_1, \ldots, b_t)$ is a black path and $(g_1, \ldots, g_s)$ is a grey path. If we define $v = (b_1 \ldots b_t)^{-1} y (b_1 \ldots b_t) = (g_1 \ldots g_s) z (g_1 \ldots g_s)^{-1}$, then v is in the same black component as y and in the same grey component as z, and so $v \in \mathcal{B}_y \cap \mathcal{G}_z$.

Finally, by an easy extension of the above result, we have,

Corollary 5.2.21 (Corollary 2.27 in [BGGM08]). *Given $x, y \in G$, let $\tilde{x} \in \mathrm{USS}(x)$ and $\tilde{y} \in \mathrm{USS}(y)$. Then x and y are conjugate if and only if $\mathcal{B}_{\tilde{x}} \cap \mathcal{G}_{\tilde{y}} \neq \emptyset$*

The complexity of Algorithm 5.2.15 is not studied in the paper, but the authors claim that it will be faster than using the entire ultra summit set, since the components of γ_x are usually smaller than $\mathrm{USS}(x)$. I have one concern regarding this algorithm, however. In all the previous algorithms we encountered for solving the conjugacy problem, it was possible to look for $\tilde{y}$ as we construct the invariant subset of the conjugacy class of $\tilde{x}$. Now, however, is is unlikely that $\tilde{y}$ will be in $\mathcal{B}_{\tilde{x}}$, so we cannot rely on that trick. Of course, if the components are small enough compared to the size of $\mathrm{USS}(x)$, this algorithm should be fine. In the case where $|\mathcal{B}_{\tilde{x}}| = |\mathrm{USS}(x)|$ or $|\mathcal{G}_{\tilde{y}}| = |\mathrm{USS}(y)|$, we are constructing the ultra summit set and then some, which is suboptimal. Thus, at the very least, we should look for $\tilde{y}$ as we construct $\mathcal{B}_{\tilde{x}}$, as we may get lucky and find it. We could also construct $\mathcal{B}_{\tilde{x}}$ and $\mathcal{G}_{\tilde{y}}$ in parallel, and check for intersection as we go. However, checking whether two sets have an intersection is not exactly a constant-time operation. We could optimise by sorting the two sets according to the binary representation of the elements, but in the worst case the very last element we find is the one in the intersection, and then we have wasted a lot of time. Thus, as it currently stand, this algorithm is likely the best approach.

More can be said about the structure of γ_x when x is a rigid element (see Section 3 of [BGGM08]), but this is beyond the scope of this dissertation.

Chapter 6

A more natural conjugation: cyclic sliding

In this final chapter of new content, we consider an almost poetic culmination of the work we've seen thusfar. We see a new subset of the conjugacy class, the set of sliding circuits, which is a subset of the ultra summit set. The algorithms for finding the set of sliding circuits share many commonalities with the algorithms we saw for the ultra summit set – we will utilise transports and pullbacks in an analogous way. The key difference is that we work with a new type of conjugation, called cyclic sliding, which one might think of as a "combination" of partial cycling and partial twisted decycling (as seen in Section 5.2). In particular, it replaces cycling and decycling to allow us to find a representative of the super summit set (using only one kind of conjugation!), and iterated cyclic sliding eventually reaches a period, which we use to construct the set of sliding circuits in a similar way as the ultra summit set is constructed via cycling. Furthermore, while cycling and decycling are trivial operations on elements of normal length 1, cyclic sliding is not.

Volker Gebhardt and Juan González-Meneses are responsible for this development. There are two papers to consider: *The cyclic sliding operation in Garside groups* [GGM10a] and *Solving the conjugacy problem in Garside groups by cyclic sliding* [GGM10b]. The first paper introduces cyclic sliding and makes it clear why it is useful with regards to the conjugacy problem, and the second paper lays out the technical details required to understand the algorithms for solving the conjugacy problem. I mainly focus on the second paper, as all the necessary facts from the first paper are reiterated in it, but I also touch on some results which are only stated in the first paper, as these expose us to the mathematical beauty of what is being done.

It should also be noted that cyclic sliding is especially natural when it comes to rigidity. Firstly, if an element $x \in G$ is conjugate to a rigid element, then $\mathrm{SC}(x)$ is the set of rigid conjugates of x. Secondly, if $x \in \mathrm{SSS}(x)$ has rigid conjugates, then iterated cyclic sliding will conjugate x to a rigid element, and the conjugating element we obtain via this process is, in fact, the minimal positive element which conjugates x to a rigid element. We will not, however, discuss these aspects related to rigidity, as it is beyond the scope of this dissertation.

We define for $x \in G$ the **preferred prefix** $\mathfrak{p}(x)$ of x as

$$\mathfrak{p}(x) = \iota(x) \wedge \iota(x^{-1}).$$

The **cyclic sliding** $\mathfrak{s}(x)$ of x is conjugation by the preferred prefix of x, i.e.,

$$\mathfrak{s}(x) = \mathfrak{p}(x)^{-1}\, x\, \mathfrak{p}(x).$$

First, notice that $\mathfrak{p}(x) \preccurlyeq \iota(x)$ and $\mathfrak{p}(x) \preccurlyeq \iota(x^{-1}) = \partial(\varphi(x))$. We cannot actually say that $\mathfrak{s}(x)$ is a partial cycling and a partial twisted decycling (which is why I put "combination" in quotes earlier), but we can clearly see there is a connection.

We can think of $\mathfrak{s}(x)$ in a different way, however. Consider $x \in G$ in left-normal form as $x = \Delta^p x_1 \dots x_r$, which can be rewritten as $x = \tau^{-p}(x_1)\Delta^p x_2 \dots x_r$. Cycling x gives $\mathbf{c}(x) = \Delta^p x_2 \dots x_r \tau^{-p}(x_1)$ and decycling x gives $\mathbf{d}(x) = x_r \tau^{-1}(x_1)\Delta^p x_2 \dots x_{r-1}$. Both cases bring the factors $x_r = \varphi(x)$ and $\tau^{-p}(x_1) = \iota(x)$ together. Now consider decomposing $x_r\tau^{-p}(x_1)$ into a left-weighted factorisation. Recall from Section 4.2 that to do so, we would let $s = \partial(x_r) \wedge \tau^{-p}(x_1)$ and write $(x_r s) \cdot (s^{-1}\tau^{-p}(x_1))$, which is then left-weighted. Thus, in order to simplify the pair formed by the first and last factor of x, we remove the prefix s from $\tau^{-p}(x_1)$ and multiply it to x_r from the right, i.e., we *slide* s from $\tau^{-p}(x_1)$ over to x_r. But this is the same as conjugating x by s! Since $\iota(x) = \tau^{-p}(x_1)$ and $\iota(x^{-1}) = \partial(\varphi(x)) = \partial(x_r)$, we can see that $s = \mathfrak{p}(x)$, the preferred prefix of x. The authors describe the cycling sliding $\mathfrak{s}(x)$ of x as a simplification of the complexity of x. To be more specific, cyclic sliding can reduce the number of factors in the left-normal form (it may also stay the same, but it will definitely not increase).

Lemma 6.1 (Lemma 1 from [GGM10a])**.** *For every $x \in G$, we have $\inf \mathfrak{s}(x) \geq \inf x$, $\sup \mathfrak{s}(x) \leq \sup x$, and $\ell(\mathfrak{s}(x)) \leq \ell(x)$.*

Clearly if $x \in \mathrm{SSS}(x)$, then $\mathfrak{s}(x) \in \mathrm{SSS}(x)$ too.

Since inf (respectively sup) can only increase (resp. decrease) a finite number of times (because $\inf y \leq \sup y$ for all $y \in G$), and since the set of elements with given inf and normal length is finite, we have,

Corollary 6.2 (Corollary 1 from [GGM10a] or[1] Corollary 2.2 from [GGM10b])**.** *For every $x \in G$, iterated application of cyclic sliding eventually reaches a period, i.e., there exist integers $0 \leq i < j$ such that $\mathfrak{s}^i(x) = \mathfrak{s}^j(x)$. In particular, for all $k \geq i$, we have $\mathfrak{s}^{j-i}(\mathfrak{s}^k(x)) = \mathfrak{s}^k(x)$.*

We can relate $\mathfrak{s}(x)$ to $\mathbf{c}(x)$ and $\mathbf{d}(x)$ by the following lemma. Note also that $\mathfrak{p}(\tau(x)) = \tau(\mathfrak{p}(x))$ and so $\mathfrak{s}(\tau(x)) = \tau(\mathfrak{s}(x))$.

Lemma 6.3 (Lemma 4 in [GGM10a])**.** *Let $x \in G$ with normal length $\ell(x) > 1$.*

(i) If $\Delta \preccurlyeq \varphi(x)\iota(x) \preccurlyeq \Delta$, then $\mathfrak{s}(x) = \tau(\boldsymbol{d}(x)) = \boldsymbol{c}(x)$ and $\ell(\mathfrak{s}(x)) < \ell(x)$.

[1] This result is proven in [GGM10a], but restated more clearly in [GGM10b]; we follow the restatement here.

(*ii*) *If* $\Delta \not\preccurlyeq \varphi(x)\iota(x) \preccurlyeq \Delta$, *then* $\mathfrak{s}(x) = \boldsymbol{c}(\boldsymbol{d}(x)) = \boldsymbol{c}(x)$ *and* $\ell(\mathfrak{s}(x)) < \ell(x)$.

(*iii*) *If* $\Delta \preccurlyeq \varphi(x)\iota(x) \not\preccurlyeq \Delta$, *then* $\mathfrak{s}(x) = \tau(\boldsymbol{d}(x)) = \boldsymbol{d}(\boldsymbol{c}(x))$ *and* $\ell(\mathfrak{s}(x)) < \ell(x)$.

(*iv*) *If* $\Delta \not\preccurlyeq \varphi(x)\iota(x) \not\preccurlyeq \Delta$, *then* $\mathfrak{s}(x) = \boldsymbol{c}(\boldsymbol{d}(x)) = \boldsymbol{d}(\boldsymbol{c}(x))$.

Moreover, if $\ell(\boldsymbol{c}(\boldsymbol{d}(x))) = \ell(x)$ *or* $\ell(\boldsymbol{d}(\boldsymbol{c}(x))) = \ell(x)$, *then case (iv) applies, which particularly implies* $\boldsymbol{c}(\boldsymbol{d}(x)) = \boldsymbol{d}(\boldsymbol{c}(x))$.

The following two corollaries make clear the value of $\mathfrak{s}$ with regards to the super summit set:

Corollary 6.4 (Corollary 2 in [GGM10a])**.** *For every* $x \in G$, *if* $\ell(x)$ *is not minimal in the conjugacy class of* x, *then there exists a positive integer* $m < ||\Delta||$ *such that* $\ell(\mathfrak{s}^m(x)) < \ell(x)$.

Recall that $||\cdot||$ indicates the number of atoms used to write the element, and in the braid group B_n, $||\Delta|| = \frac{n(n-1)}{2}$.

Corollary 6.5 (Corollary 3 in [GGM10a])**.** *Let* $x \in G$. *There exists an integer*

$$M \leq (\ell(x) - 1)(||\Delta|| - 1)$$

such that $\mathfrak{s}^m(x) \in \mathrm{SSS}(x)$ *for all* $m \geq M$.

Corollary 6.5 gives us a lower bound for how many times we would have to apply cyclic sliding before reaching an element in $\mathrm{SSS}(x)$, and Corollary 6.4 tells us that it would take at most $||\Delta|| - 1$ consecutive applications of cyclic sliding for the normal length to decrease. Thus if the normal length does not decrease within $||\Delta|| - 1$ consecutive applications, it will never decrease and so we have reached the super summit set.

For $y \in G$, we say y belongs to a **sliding circuit** if $\mathfrak{s}^k(y) = y$ for some $k \geq 1$. Given $x \in G$, we define the **set of sliding circuits** of x, $\mathrm{SC}(x)$, as the set of all conjugates of x which belongs to a sliding circuit. Symbolically, where $C(x)$ denotes the conjugacy class of x,

$$\mathrm{SC}(x) = \{y \in C(x) \mid \mathfrak{s}^m(y) = y \text{ for some } m \geq 1\}.$$

Notice the similarity to the definition of the ultra summit set. If $y \in \mathrm{SC}(x)$, we say that N is the **length of the sliding circuit** of y if N is the smallest integer such that $\mathfrak{s}^N(y) = y$.

$\mathrm{SC}(x)$ satisfies

$$\mathrm{SC}(x) \subseteq \mathrm{USS}(x) \subseteq \mathrm{SSS}(x) \subseteq \mathrm{SS}(x)$$

and depends only on the conjugacy class of x, not on x itself. Thus, as with the other sets in the above inclusion sequence, $x, y \in G$ are conjugate if and only if $\mathrm{SC}(x) = \mathrm{SC}(y)$, or equivalently, if $\mathrm{SC}(x) \cap \mathrm{SC}(y) \neq \emptyset$. Furthermore, to determine whether x and y are conjugate, we will proceed similarly to previous algorithms: find representatives $\tilde{x} \in \mathrm{SC}(x)$ and $\tilde{y} \in \mathrm{SC}(y)$ using cyclic sliding (and keep track of conjugators); start computing the whole $\mathrm{SC}(x)$ until $\tilde{y}$ is found as an element of $\mathrm{SC}(x)$ and return the applicable conjugating element between x and y; or compute the entire $SC(x)$ without encountering $\tilde{y}$, in which case x and y are not conjugate.

The tricky part of the above procedure is being able to find the entire $\mathrm{SC}(x)$. For now, however, we can state the algorithm which finds a representative $\tilde{x}$ of $\mathrm{SC}(x)$.

Algorithm 6.6 (Algorithm 1 in [GGM10b])**.**
Input: $x \in G$.
Output: $\tilde{x} \in \mathrm{SC}(x)$ *and* $c \in G$ *such that* $c^{-1}\, x\, c = \tilde{x}$.

1. Set $\tilde{x} = x$, $c = 1$, and $T = \emptyset$.
2. While $\tilde{x} \notin T$, do:
   (a) Set $T = T \cup \{\tilde{x}\}$.
   (b) Set $c = c \cdot \mathfrak{p}(\tilde{x})$ and $\tilde{x} = \mathfrak{s}(\tilde{x})$.
3. Set $y = \mathfrak{s}(\tilde{x})$ and $d = \mathfrak{p}(\tilde{x})$.
4. While $y \neq \tilde{x}$, do:
   (a) Set $d = d \cdot \mathfrak{p}(y)$, and $y = \mathfrak{s}(y)$.
5. Return $\tilde{x}$ and $c \cdot d^{-1}$.

Thanks to Steps 3 to 4a, this algorithm returns the smallest conjugator from x to $\tilde{x} \in \mathrm{SC}(x)$. Suppose $\mathfrak{s}^i(x) = \mathfrak{s}^j(x)$ for $i < j$. c conjugates x to $\mathfrak{s}^j(x)$, and $d \neq 1$ conjugates $\mathfrak{s}^j(x)$ to $\mathfrak{s}^i(x)$. Hence, by removing d from c, we get $c \cdot d^{-1}$, which conjugates x to $\mathfrak{s}^i(x)$, and is smaller w.r.t. $\preccurlyeq$ than c.

We now know how to find a representative of $\mathrm{SC}(x)$. Once we have such a representative, we need to find the rest of $\mathrm{SC}(x)$. To do so will require finding minimal elements which conjugate an element of $\mathrm{SC}(x)$ to another element of $\mathrm{SC}(x)$. The authors approach this from a graph theoretic perspective.

Given $x \in G$, the **sliding circuits graph** of x, which we will denote[2] by $\mathfrak{g}_x$, is the directed graph with vertex set $\mathrm{SC}(x)$ and directed edges/arrows corresponding to conjugating elements as follows: There is an arrow which starts at $u \in \mathrm{SC}(x)$, ends at $v \in \mathrm{SC}(x)$, and is labelled by $s \in M \backslash \{1\}$ if and only if:

(*i*) $s^{-1}\, u\, s = v$.

(*ii*) s is an indecomposable conjugator, that is, $s \neq 1$ and there is no element t such that $1 \prec t \prec s$ and $t^{-1}\, u\, t \in \mathrm{SC}(x)$.

As mentioned when we first introduced $\preccurlyeq$, $\prec$ is a "stricter" relation, as equality is not an option, so $t \prec s \Rightarrow ta = s$ where $a \in M \backslash \{1\}$. Notice that "indecomposable conjugator" is equivalent to "minimal simple element" if we know that s is simple. In fact, a later corollary confirms:

[2]This graph is denoted by $\mathrm{SCG}(x)$ in the paper, but I feel a variation on the letter g is more consistent with what we've seen so far when denoting graphs.

Corollary 6.7 (Corollary 3.3 in [GGM10b]). *Let $x \in G$ and let $y \in \mathrm{SC}(x)$ be a vertex of $\mathfrak{g}_x$. Then the label of each arrow in $\mathfrak{g}_x$ is a simple element and the number of arrows in $\mathfrak{g}_x$ starting at y is bounded by the number of atoms of G.*

Thus, we can continue with the notation introduced in Section 4.3 and denote by S_y^{SC} the set of minimal simple elements which conjugate $y \in \mathrm{SC}(x)$ to other elements of $\mathrm{SC}(x)$.

The fact that $|S_y^{\mathrm{SC}}| \leq k$ where k is the number of atoms of G is due to the property, "conjugates to an element in $\mathrm{SC}(x)$," being closed under gcd, which is thanks to this result:

Proposition 6.8 (Proposition 7 in [GGM10a]). *Let $x \in G$. If $a^{-1} x a \in \mathrm{SC}(x)$ and $b^{-1} x b \in \mathrm{SC}(x)$ for elements $a, b \in G$, then $(a \wedge b)^{-1} x (a \wedge b) \in \mathrm{SC}(x)$.*

Furthermore, we have the following corollary, which is analogous to the **convexity property** established for the summit set, super summit set, and ultra summit set:

Corollary 6.9. *For any $x \in G$, the graph $\mathfrak{g}_x$ is finite and connected.*

Let's dive into the minimal simple elements. In a similar way to the process for finding minimal simple elements in the ultra summit set, we start by looking for minimal simple elements which conjugate to elements in the super summit set. However, the authors offer a new way to find minimal simple elements for the super summit set. We start with the following definition:

Corollary 6.10 (Corollary 2.8 in [GGM10b]). *Let $x \in G$. There is a unique positive element $\rho(x)$ (possibly trivial) satisfying $\rho(x)^{-1} x \rho(x) \in \mathrm{SSS}(x)$ and $\rho(x) \preccurlyeq b$ for every positive $b \in G$ satisfying $b^{-1} x b \in \mathrm{SSS}(x)$.*

Notice that this does not assume x is in $\mathrm{SSS}(x)$, but it will also hold if $x \in \mathrm{SSS}(x)$ – this is the trivial case. We can find $\rho(x)$ as below. This algorithm assumes we know the summit infimum and summit supremum of x (recall that these are denoted, respectively, by $\inf_* x$ and $\sup_* x$). This information will already be known if we are finding $\rho(x)$ for $x \in \mathrm{SSS}(x)$ (but we know the algorithm will output the identity, so there is little point to running it), but if $x \notin \mathrm{SSS}(x)$, I assume in the worst case we would need to use cyclic sliding to find $\inf_* x$ and $\sup_* x$. The more likely scenario, however, is that we would be looking for $\rho(s^{-1} y s)$ where $y \in \mathrm{SSS}(x)$, in which case we would already have $\inf_* x$ and $\sup_* x$ at hand. We will shortly see why this would be the case.

Algorithm 6.11 (Proposition 3.4 in [GGM10b]).
Input: $x \in G$, $\inf_* x$, *and* $\sup_* x$.
Output: $\rho(x)$.

1. Set $\rho = 1$.
2. While $\inf \rho^{-1} x \rho < \inf_* x$ or $\sup \rho^{-1} x \rho > \sup_* x$, do:
   (a) Set $\rho = \rho \cdot (1 \vee (\rho^{-1} x \rho)^{-1} \Delta^{\inf_* x} \vee \rho^{-1} x \rho \Delta^{-\sup_* x})$.
3. Return $\rho(x) = \rho$.

Notice that the while-loop only ends once $\inf \rho^{-1} x \rho = \inf_* x$ **and** $\sup \rho^{-1} x \rho = \sup_* x$, i.e., once we reach an element in $\mathrm{SSS}(x)$.

We can now define:

Corollary 6.12 (Corollary 3.1 in [GGM10b])**.** *Let $x \in G$. Given $y \in \mathrm{SSS}(x)$ and $u \in G$, define $\rho_u = \rho_u(y) = u \cdot \rho(u^{-1} y u)$. Then ρ_u is the unique minimal element satisfying $u \preccurlyeq \rho_u$ and $\rho_u^{-1} y \rho_u \in \mathrm{SSS}(x)$. Moreover, $\rho_u(y) \preccurlyeq \Delta^{\sup u}$.*

The last statement in this corollary implies that if u is simple, ρ_u is simple too.

This ρ_u is nearly the same as the ρ_a we define in Section 4.3 and update to $\rho_s(y)$ in Section 5.1. The only difference is that $u \in G$, while a referred to an atom of G and s to a simple element. The core idea is the same, and now with Algorithm 6.11 and the above corollary, we can find $\rho_u(y)$ with more ease.

We have similar definitions for elements conjugating x to another element in $\mathrm{SC}(x)$:

Corollary 6.13 (Corollary 2.10 in [GGM10b])**.** *Let $x \in G$. There is a unique positive element $\kappa(x)$ (possibly trivial) satisfying $\kappa(x)^{-1} x \kappa(x) \in \mathrm{SC}(x)$ and $\kappa(x) \preccurlyeq b$ for every positive $b \in G$ satisfying $b^{-1} x b \in \mathrm{SC}(x)$.*

Corollary 6.14 (Corollary 3.2 in [GGM10b])**.** *Let $x \in G$. Given $y \in \mathrm{SC}(x)$ and $u \in G$, define $\kappa_u = \kappa_u(y) = u \cdot \kappa(u^{-1} y u)$. Then κ_u is the unique minimal element satisfying $u \preccurlyeq \kappa_u$ and $\kappa_u^{-1} y \kappa_u \in \mathrm{SC}(x)$. Moreover, $\kappa_u \preccurlyeq \Delta^{\sup u}$.*

$\kappa(x)$ and $\kappa_u(x) = \kappa_u$ are denoted by $c(x)$ and $c_u(x)$ in the paper, but I find this to be confusing, as we often use c_u to indicate a conjugator from $\tilde{x}$ to u in our algorithms.

The method for finding $\kappa_u(x)$ is analogous to Algorithm 5.1.18 (which finds $\mathfrak{c}_s(x)$): we use transports and pullbacks as defined for cyclic sliding. Then, by letting u iterate over the atoms of G, we can find all the arrows starting at x in the graph $\mathfrak{g}_x$.

Given $x, a \in G$, we define the **transport** of a at x under cyclic sliding as

$$a^{(1)} = \mathfrak{p}(x)^{-1} a \, \mathfrak{p}(a^{-1} x a).$$

$a^{(1)}$ conjugates $\mathfrak{s}(x)$ to $\mathfrak{s}(a^{-1} x a)$. For any integer $i > 1$ we define recursively $a^{(i)} = (a^{(i-1)})^{(1)}$, where $(a^{(i-1)})^{(1)}$ indicates the transport of $a^{(i-1)}$ at $\mathfrak{s}^{i-1}(x)$; in particular, $a^{(i)}$ conjugates $\mathfrak{s}^i(x)$ to $\mathfrak{s}^i(a^{-1} x a)$. Furthermore, we define $a^{(0)} = a$.

We can transport iteratively[3], via

$$a^{(i)} = \mathfrak{p}\big(\mathfrak{s}^{i-1}(x)\big)^{-1} a^{(i-1)} \, \mathfrak{p}\big(\mathfrak{s}^{i-1}(a^{-1} x a)\big). \tag{6.1}$$

We can visualise transports using the following commutative diagram, where horizontal arrows correspond to cyclic sliding:

$$\begin{array}{ccccccccc}
x & \xrightarrow{\mathfrak{p}(x)} & \mathfrak{s}(x) & \xrightarrow{\mathfrak{p}(\mathfrak{s}(x))} & \mathfrak{s}^2(x) & \longrightarrow \cdots \longrightarrow & \mathfrak{s}^{i-1}(x) & \xrightarrow{\mathfrak{p}(\mathfrak{s}^{i-1}(x))} & \mathfrak{s}^i(x) \\
\downarrow{\scriptstyle a=a^{(0)}} & & \downarrow{\scriptstyle a^{(1)}} & & \downarrow{\scriptstyle a^{(2)}} & & \downarrow{\scriptstyle a^{(i-1)}} & & \downarrow{\scriptstyle a^{(i)}} \\
a^{-1}xa & \xrightarrow[\mathfrak{p}(a^{-1}xa)]{} & \mathfrak{s}(a^{-1}xa) & \xrightarrow[\mathfrak{p}(\mathfrak{s}(a^{-1}xa))]{} & \mathfrak{s}^2(a^{-1}xa) & \longrightarrow \cdots \longrightarrow & \mathfrak{s}^{i-1}(a^{-1}xa) & \xrightarrow[\mathfrak{p}(\mathfrak{s}^{i-1}(a^{-1}xa))]{} & \mathfrak{s}^i(a^{-1}xa)
\end{array}$$

[3]This expression is not given in the paper, but it is easy to deduce from the information provided

We have the following useful features of the transport:

Proposition 6.15 (Proposition 2.4 in [GGM10b]). *Let $x, a, b \in G$ such that $x, a^{-1}\,x\,a, b^{-1}\,x\,b \in \mathrm{SSS}(x)$ and consider transports at x. Then the following hold:*

(*i*) *If a is positive then $a^{(1)}$ is positive.*

(*ii*) *If a is positive then $\mathfrak{p}(x) \preccurlyeq a\,\mathfrak{p}(a^{-1}\,x\,a)$.*

(*iii*) *If $a = \Delta^k$ for $k \in \mathbb{Z}$ then $a^{(1)} = \Delta^k$.*

(*iv*) *If $a \preccurlyeq b$ then $a^{(1)} \preccurlyeq b^{(1)}$.*

(*v*) *If a is simple then $a^{(1)}$ is simple.*

(*vi*) $(a \wedge b)^{(1)} = a^{(1)} \wedge b^{(1)}$.

Next, we can see how to use transports in the search for κ_u. The results we encounter look very similar to those we saw for the ultra summit set.

Lemma 6.16 (Lemma 2.5 in [GGM10b]). *Let $x \in G$, $y \in \mathrm{SC}(x)$, and $r \in G$ such that $r^{-1}\,y\,r \in \mathrm{SSS}(x)$. Let N be a positive integer such that $\mathfrak{s}^N(y) = y$ and for integers $i \geq 0$ consider the transports $r^{(iN)}$ at y. Then there are integers $i_2 > i_1 \geq 0$ such that $r^{(i_1 N)} = r^{(i_2 N)}$. Furthermore, $r^{-1}\,y\,r \in \mathrm{SC}(x)$ if and only if there is a positive integer k such that $r^{(kN)} = r$.*

Given the situation in the above lemma, we can let i_1 and i_2 be minimal subject to the conditions listed, and define $F(r) = \{r^{(iN)} \mid i_1 \leq i < i_2\}$ and $l(r) = i_2 - i_1$.

Lemma 6.17 (Lemma 3.8 in [GGM10b]). *In the situation of the above definition, the following hold.*

(*i*) *For all $k \geq i_1$, we have $r^{((k+l(r))N)} = r^{(kN)}$ and $l(r)$ is the minimal positive integer satisfying this condition. In particular, for all $t \in F(r)$ and all integers $i \geq 0$ one has $t^{(il(r)N)} = t$, from which we have $t^{-1}\,y\,t \in \mathrm{SC}(x)$.*

(*ii*) *$1 \in F(r)$ if and only if $F(r) = \{1\}$.*

(*iii*) *$r^{-1}\,y\,r \in \mathrm{SC}(x)$ if and only if $r \in F(r)$.*

(*iv*) $F(r) = \{r^{(iN)} \mid (r^{(iN)})^{-1}\,y\,r^{(iN)} \in \mathrm{SC}(x)$ and $i \geq 0$ is an integer$\}$.

The similarities to $F_x(u)$, which we saw in Section 5.1, are very apparent. We use $F(r)$ in our work towards finding κ_u.

Lemma 6.18 (Lemma 3.9 in [GGM10b]). *Let $x \in G$, $y \in \mathrm{SC}(x)$, $u \in G$ and denote $\kappa_u = \kappa_u(y)$. Let N be the length of the sliding circuit of y. If $\kappa_u \preccurlyeq \kappa_u^{(iN)}$ for some $i > 0$ then $\kappa_u^{(iN)} = \kappa_u$.*

Lemma 6.19 (Lemma 3.10 in [GGM10b]). *Let $x \in G$, $y \in \mathrm{SC}(x)$, $u \in M$, and denote $\kappa_u = \kappa_u(y)$. Assume r is a positive element satisfying $r \preccurlyeq \kappa_u$ and $r^{-1}\, y\, r \in \mathrm{SSS}(y)$ and assume further that $F(r) \neq \{1\}$.*

(i) *If there exists $t \in F(r)$ such that $u \preccurlyeq t$, then $\kappa_u = t$.*

(ii) *If $u \not\preccurlyeq t$ for all $t \in F(r)$, then κ_u is not an indecomposable conjugator starting at y (i.e. it is not minimal w.r.t. $\preccurlyeq$).*

Thus, in some cases iterated transport may be enough to find κ_u or to say that it is not minimal (in which case we would want to consider a different u). However, we may encounter $F(r) = \{1\}$, in which case we need to do more work to find κ_u.

The following result give us another way to check whether iterated transport would be enough:

Corollary 6.20 (Corollary 3.12 in [GGM10b]). *Let $x \in G$ and $y \in \mathrm{SC}(x)$. Let a be an atom such that $a \not\preccurlyeq \mathfrak{p}(y)$. Then either $F(\rho_a) \neq \{1\}$ or κ_a is not an indecomposable conjugator starting at y.*

Checking whether $a \preccurlyeq \mathfrak{p}(y)$ is much quicker than finding the set $F(\rho_a)$ only to discover it is $\{1\}$ and hence useless. If indeed $a \preccurlyeq \mathfrak{p}(y)$, then $F(\rho_a) = \{1\}$ and we first need to utilise pullbacks, and then transports, to find κ_a.

Let $x \in G$, $z \in \mathrm{SC}(x)$, $y = \mathfrak{s}(z)$, and let $s \in G$ be positive. There exists a unique minimal positive element $s_{(1)} \in G$ satisfying $s_{(1)}^{-1}\, z\, s_{(1)} \in \mathrm{SSS}(x)$ and $s \preccurlyeq (s_{(1)})^{(1)}$, where $(s_{(1)})^{(1)}$ indicates the transport of $s_{(1)}$ at z. $s_{(1)}$ is called the **pullback** of s at y.

In Proposition 3.19 of [GGM10b], it is shown that, in the situation above, we have

$$s_{(1)} = \left(\mathfrak{p}(z)\, s\, \mathfrak{p}^{\curvearrowleft}(s^{-1}\, y\, s)^{-1}\right) \vee 1.$$

We define $\mathfrak{p}^{\curvearrowleft}$ in a moment. For any integer $k > 1$, we define recursively the k-fold pullback $s_{(k)} = (s_{(k-1)})_{(1)}$ of s at y. $(s_{(k-1)})_{(1)}$ indicates the pullback of $s_{(k-1)}$ at w, where w is the unique element in the sliding circuit of y satisfying $\mathfrak{s}^{k-1}(w) = y$. Furthermore, we define $s_{(0)} = s$. No expression is provided for iterated pullbacks, but we can find one easily, based on the description provided. To do so requires the following result:

Lemma 6.21. *Let $x \in G$, $z \in \mathrm{SC}(x)$, $y = \mathfrak{s}(z)$, and let z have sliding circuit length N, i.e., N is the smallest integer satisfying $\mathfrak{s}^N(z) = z$. Then, y also has sliding circuit length N.*

Proof. Corollary 6.2 assures us that for all $k \geq 0$, $\mathfrak{s}^N(\mathfrak{s}^k(z)) = \mathfrak{s}^k(z)$, and that $N > 0$. Thus, with $k = 1$, we have $\mathfrak{s}^N(y) = y$. We still need to show, however, that N is the smallest integer that satisfies $\mathfrak{s}^N(y) = y$. Suppose, to the contrary, that $\mathfrak{s}^M(y) = y$ where $M < N$. Then $\mathfrak{s}^M(y) = \mathfrak{s}^M(\mathfrak{s}(z)) = y$ which implies $\mathfrak{s}^{M+1}(z) = \mathfrak{s}(z)$. Applying cyclic sliding $N - 1$ times on both sides of this equality gives us $\mathfrak{s}^{N-1}(\mathfrak{s}^{M+1}) = \mathfrak{s}^{N-1}(\mathfrak{s}(z))$, which results in $\mathfrak{s}^{N-1+M+1}(z) = \mathfrak{s}^{N-1+1}(z)$. This, in turn, gives us $\mathfrak{s}^{N+M}(z) = \mathfrak{s}^N(z)$. The left-hand side can be rewritten as $\mathfrak{s}^{N+M}(z) = \mathfrak{s}^{M+N}(z) = \mathfrak{s}^M(\mathfrak{s}^N(z))$, and we know $\mathfrak{s}^N(z) = z$. Thus, we have that $\mathfrak{s}^M(z) = z$. But this contradicts the minimality of N. Therefore, $y = \mathfrak{s}(z)$ has sliding circuit length N. □

Given what we know about the sliding circuit length of z and y in the definition of the pullback, we can see that $w = \mathfrak{s}^\alpha(y)$ where $0 \le \alpha \equiv -(k-1) \mod N$, which means $\alpha = qN - (k-1)$ for some integer q, and so $\mathfrak{s}^{k-1}(w) = \mathfrak{s}^{k-1}(\mathfrak{s}^\alpha(y)) = \mathfrak{s}^{k-1+qN-(k-1)}(y) = \mathfrak{s}^{qN}(y) = y$, as required (since $\mathfrak{s}^{qN}(y) = \underbrace{\mathfrak{s}^N(\mathfrak{s}^N(\mathfrak{s}^N(\dots(\mathfrak{s}^N}_{q \text{ times}}(y)))))$, which will remain y).

Furthermore, if we're considering the pullback at $w = \mathfrak{s}^\alpha(y)$, we will need to know v such that $\mathfrak{s}(v) = w = \mathfrak{s}^\alpha(y)$ (analogous to knowing z such that $\mathfrak{s}(z) = y$ when we find the pullback at y). We could simply let $v = \mathfrak{s}^{\alpha-1}(y)$, but this will cause issues if $\alpha = 0$. Thus, we will let $v = \mathfrak{s}^\beta(y)$ such that $0 \le \beta = -k \mod N$. Then, $\beta = pN - k$ for some integer p, and $\mathfrak{s}(v) = \mathfrak{s}(\mathfrak{s}^\beta(y)) = \mathfrak{s}^{1+pN-k}(y) = \mathfrak{s}^{pN-(k-1)}(y) = w$ (since $pN - (k-1) \equiv -(k-1) \mod N$) as desired.

Putting this all together, we can derive the following expression for finding $s_{(k)}$, the pullback of $s_{(k-1)}$ at $w = \mathfrak{s}^\alpha(y)$:

$$s_{(k)} = (s_{(k-1)})_{(1)} = \left(\mathfrak{p}\big(\mathfrak{s}^\beta(y)\big)\, s_{(k-1)}\, \mathfrak{p}^{\curvearrowleft}\big(s_{(k-1)}^{-1}\, \mathfrak{s}^\alpha(y)\, s_{(k-1)}\big)^{-1}\right) \vee 1, \tag{6.2}$$

where $0 \le \alpha \equiv -(k-1) \mod N$ and $0 \le \beta \equiv -k \mod N$. Note that each $s_{(k)}$ satisfies $s_{(k)}^{-1}\, \mathfrak{s}^\beta(y)\, s_{(k)} \in \mathrm{SSS}(x)$ where β is as defined earlier. We consider an example, to illustrate the use of this expression, in Section A.5.

We now define $\mathfrak{p}^{\curvearrowleft}$. Recall that we use $^{\curvearrowleft}$ to indicate operations with respect to $\succcurlyeq$. Recall the right gcd $\wedge^{\curvearrowleft}$ and right lcm $\vee^{\curvearrowleft}$. If $x \in G$ is in right-normal form as $x = y_r \dots y_1 \Delta^p$, then the **right initial factor** is

$$\iota^{\curvearrowleft}(x) = \Delta^{-\inf x} x \wedge^{\curvearrowleft} \Delta = \tau^p(y_1)$$

and the **right final factor** is

$$\varphi^{\curvearrowleft}(x) = x(\Delta^{\sup x - 1} \wedge^{\curvearrowleft} x)^{-1} = y_r.$$

We also have that $\iota^{\curvearrowleft}(x^{-1}) = \partial^{-1}(\varphi^{\curvearrowleft}(x))$.

The **preferred suffix** $\mathfrak{p}^{\curvearrowleft}(x)$ of x is

$$\mathfrak{p}^{\curvearrowleft}(x) = \iota^{\curvearrowleft}(x) \wedge^{\curvearrowleft} \iota^{\curvearrowleft}(x^{-1}) = \iota^{\curvearrowleft}(x) \wedge^{\curvearrowleft} \partial^{-1}(\varphi^{\curvearrowleft}(x)).$$

The **cyclic right sliding** $\mathfrak{s}^{\curvearrowleft}(x)$ of x is the conjugation of x by the inverse of its preferred suffix:

$$\mathfrak{s}^{\curvearrowleft}(x) = \mathfrak{p}^{\curvearrowleft}(x)\, x\, \mathfrak{p}^{\curvearrowleft}(x)^{-1}.$$

In our commutative diagrams, the direction of the arrow indicates the direction of the conjugation. For example, $x \xrightarrow{\iota(x)} \mathbf{c}(x)$ and $x \xrightarrow{\mathfrak{p}(x)} \mathfrak{s}(x)$. The cyclic right sliding comes "toward" x, however, since it is conjugation by the inverse of the preferred suffix. Thus, in a commutative diagram, we will have $x \xleftarrow{\mathfrak{p}^{\curvearrowleft}(x)} \mathfrak{s}^{\curvearrowleft}(x)$. This would also be the case for decycling, since $\mathbf{d}(x) = \varphi(x) x \varphi(x)^{-1}$; so $x \xleftarrow{\varphi(x)} \mathbf{d}(x)$.

With all of this in mind, we can now consider the pullback in terms of a commutative diagram. For $x \in G$, $z \in \mathrm{SC}(x)$, $y = \mathfrak{s}(z)$ and $s \in G$ a positive element such that $s^{-1}\, y\, s \in \mathrm{SSS}(x)$, the

pullback of s at y is $s_{(1)} = \varepsilon \vee 1$ where $\varepsilon \in G$ is the element that makes the below diagram commute:

$$\begin{array}{ccc} z & \xrightarrow{\mathfrak{p}(z)} & y \\ \downarrow{\scriptstyle \varepsilon} & & \downarrow{\scriptstyle s} \\ \mathfrak{s}^{\curvearrowleft}(s^{-1}ys) & \xrightarrow{\mathfrak{p}^{\curvearrowleft}(s^{-1}ys)} & s^{-1}ys \end{array}$$

Recall that taking the left lcm of ε and the identity 1 means that we are finding the smallest (w.r.t. $\preccurlyeq$) multiple of ε which is positive.

The pullback satisfies some useful properties, listed in the following three lemmas:

Lemma 6.22 (Lemma 3.15 in [GGM10b]). *Let $x \in G$, $z \in \mathrm{SC}(x)$, $y = \mathfrak{s}^k(z)$ for a positive integer k and let $s \in G$ be positive. Then the k-fold pullback $s_{(k)}$ of s at y is the minimal positive element satisfying $s \preccurlyeq (s_{(k)})^{(k)}$ and $s_{(k)}^{-1}\, z\, s_{(k)} \in \mathrm{SSS}(x)$.*

Lemma 6.23 (Lemma 3.16 in [GGM10b]). *Let $x \in G$, $z \in \mathrm{SC}(x)$, $y = \mathfrak{s}^k(z)$ for a positive integer k and let $s, t \in G$ such that $1 \preccurlyeq s \preccurlyeq t$. Then the k-fold pullbacks $s_{(k)}$ of s and $t_{(k)}$ of t at y satisfy $s_{(k)} \preccurlyeq t_{(k)}$.*

Lemma 6.24 (Lemma 3.17 in [GGM10b]). *Let $x \in G$, $z \in \mathrm{SC}(x)$, $y = \mathfrak{s}(z)$ and let $s \in G$ be positive. Then the pullback $s_{(1)}$ of s at y satisfies $s_{(1)} \preccurlyeq \Delta^{\sup s}$. In particular, the pullback of a simple element is simple.*

Finally, we can see how pullbacks are useful in our goal of finding arrows in $\mathfrak{g}_x$:

Proposition 6.25 (Porposition 3.18 in [GGM10b]). *Let $x \in G$, $v \in \mathrm{SC}(x)$ and let N be the length of the sliding circuit of v. Let $s \in M \backslash \{1\}$ such that $s^{-1}\, v\, s \in \mathrm{SSS}(x)$ and for integers $k \geq 0$ consider the iterated pullbacks $s_{(kN)}$ at v. Let $i \geq 0$ be such that $s_{(iN)} = s_{(jN)}$ for some $j > i$. Then κ_s is the only element in $F(s_{(iN)})$ which admits s as a prefix. In particular, $F(s_{(iN)}) \neq \{1\}$.*

In the notation of [Geb05], we would name $s_{(iN)}$ $p_v = p_v(s)$. We can now consider the algorithm to find all the arrows which start at the vertex v in $\mathrm{SC}(x)$, and the algorithm to solve the conjugacy problem using the set of sliding circuits.

Algorithm 6.26 (Algorithm 2 in [GGM10b]).
Input: $v \in \mathrm{SC}(x)$.
Output: The set $\mathcal{A}_v$ of arrows in the graph $\mathfrak{g}_x$.

1. Compute the minimal integer $N > 0$ such that $\mathfrak{s}^N(v) = v$.
2. List the atoms of G, say $a_1, \ldots, a_\lambda$. Set $\mathcal{A}_v = \emptyset$ and $\texttt{Atoms} = \emptyset$.
3. For $t = 1, \ldots, \lambda$, do:
   (a) Set $s = a_t$.

(b) While $\ell(s^{-1}\, v\, s) > \ell(v)$, do:

i. Set $s = s \cdot \big(1 \vee (s^{-1}\, v\, s)^{-1} \Delta^{\inf(v)} \vee s^{-1}\, v\, s \Delta^{-\sup v}\big)$.

(c) If $a_t \preccurlyeq \mathfrak{p}(v)$, then compute the iterated N-pullbacks $s, s_{(N)}, s_{(2N)}, \ldots$ until the first repetition is encountered, say $s_{(rN)}$, and set $s = s_{(rN)}$.

(d) Compute the iterated N-transports $s, s^{(N)}, s^{(2N)}, \ldots$ until the first repetition is encountered, say $s^{(jN)}$. Let $i < j$ be such that $s^{(iN)} = s^{(jN)}$.

(e) If $a_t \preccurlyeq s^{(mN)}$ for some m with $i \leq m < j$, then do:

i. If $a_k \not\preccurlyeq s^{(mN)}$ for all $k = 1, \ldots, \lambda$ such that either $a_k \in \texttt{Atoms}$ or $k > t$, then set

$$\mathcal{A}_v = \mathcal{A}_v \cup \{s^{(mN)}\} \quad \text{and} \quad \texttt{Atoms} = \texttt{Atoms} \cup \{a_t\}.$$

4. Return $\mathcal{A}_v$.

Step 1 is accomplished by counting how many times one must apply cyclic sliding to v before encountering v again. This is guaranteed to happen, since $v \in \mathrm{SC}(x)$ to start with. Step 3(b)i is using Algorithm 6.11 and Corollary 6.12 to find $\rho_{a_t}(v) = a_t \cdot \rho(a_t^{-1}\, v\, a_t)$. Thus, when we are about to evaluate Step 3c, $s = \rho_{a_t}(v)$, and so we know $s^{-1}\, v\, s \in \mathrm{SSS}(x)$. Step 3c starts by checking whether we will need to use pullbacks and transports, or if transports alone will be sufficient (this check is thanks to Corollary 6.20).

If pullbacks will be necessary (i.e., if $a_t \preccurlyeq \mathfrak{p}(v)$), we find the iterated pullbacks of $s = \rho_{a_t}$, paying attention to multiples-of-N-fold pullbacks. When the first repetition is found, say $s_{(rN)} = (\rho_{a_t})_{(rN)}$, we update s and continue to the next step where we will use transports to find κ_{a_t} (thanks to Proposition 6.25). On the other hand, if we have $a_t \not\preccurlyeq \mathfrak{p}(v)$, then we know we will only need transports, so we can skip the rest of Step 3c and go ahead to Step 3d, where we will find the transports of $s = \rho_{a_t}$ in order to find κ_{a_t}.

In Step 3d, we find the iterated transports of $s = \rho_{a_t}$ or $s = (\rho_{a_t})_{(rN)}$, depending on what occurred in the previous step, and keep track of the multiple-of-N-fold transports. When we have our first repetition, $s^{(jN)}$ (i.e., there is an integer $i < j$ such that $s^{(iN)} = s^{(jN)}$), we have implicitly also found $F(s) = \{s^{(pN)} \mid i \leq p < j\}$ (this is $F(\rho_{a_t})$ or $F((\rho_{a_t})_{(rN)})$). Now, Step 3e checks whether a_t is a prefix of any of the elements in $F(s)$. If none of the elements of $F(s)$ admit a_t as a prefix, then Lemma 6.19(ii) tells us that κ_{a_t} is not an indecomposable conjugator, and so we abandon these efforts and consider the next atom in the list.

On the other hand, if there is an element in $F(s)$, say $s^{(mN)}$, which admits a_t as a prefix, we go to Step 3(e)i. The goal from here is to check whether the element $s^{(mN)}$ we have found, is truly an indecomposable conjugator. This is done using the same logic as used in Algorithms 4.3.10 and 4.3.13.

Note that we cannot simply find $s^{(N)}, s^{(2N)}, s^{(3N)}, \ldots$ and $s_{(N)}, s_{(2N)}, s_{(3N)}, \ldots$ from s. We need all the intermediate transports and intermediate pullbacks as they are required when

we use expressions 6.1 and 6.2. Lastly, we can solve the conjugacy problem using the below algorithm. Note that the logic of this algorithm has been adjusted slightly, for consistency with the rest of the algorithms in this dissertation. As before, we let c_q denote the conjugator between $\tilde{x}$ and q.

Algorithm 6.27 (Algorithm 3 in [GGM10b])**.**
Input: $x, y \in G$.
Output: A conjugating element $c \in G$ *such that* $c^{-1}\, x\, c = y$, *or* `fail` *if* x *and* y *are not conjugate.*

1. Use Algorithm 6.6 to compute $\tilde{x} \in \mathrm{SC}(x)$ and a such that $a^{-1}\, x\, a = \tilde{x}$, and $\tilde{y} \in \mathrm{SC}(y)$ and b such that $b^{-1}\, y\, b = \tilde{y}$.
2. Set $V = \{\tilde{x}\}$, $W = \emptyset$, and $c_{\tilde{x}} = 1$.
3. While $V \neq W$, do:
   (a) Take $v \in V \backslash W$.
   (b) Use Algorithm 6.26 to compute $\mathcal{A}_v$.
   (c) For every $s \in \mathcal{A}_v$, do:
      i. If $s^{-1}\, v\, s = \tilde{y}$, then set $c_{\tilde{y}} = c_v \cdot s$. Return $a \cdot c_{\tilde{y}} \cdot b^{-1}$.
      ii. If $s^{-1}\, v\, s \notin V$, then set $c_{s^{-1}\, v\, s} = c_v \cdot s$ and $V = V \cup \{s^{-1}\, v\, s\}$.
   (d) Set $W = W \cup \{v\}$
4. Return `fail`.

This algorithm works in a similar fashion to Algorithm 4.3.15. As we find new vertices, we compute the conjugator between $\tilde{x}$ and the new vertex. Once we find a vertex with an arrow which points to $\tilde{y}$, the conjugator from $\tilde{x}$ to $\tilde{y}$ is computed and stored as $c_{\tilde{y}}$. Then the conjugator between x and y is $a \cdot c_{\tilde{y}} \cdot b^{-1}$. If $\tilde{y}$ is never found, $V = W$, and the algorithm returns `fail`.

Lastly, we briefly discuss the complexity of Algorithm 6.27, which solves the conjugacy problem. Assume that $x \in G$ is given as a product of simple elements or inverses of simple elements, with at most k such factors; since Δ is a simple element, if we have $x = \Delta^p x_1 \dots x_r$ in left-normal form, then $k = |p| + r$. Let λ be the number of atoms for M. Let $||\Delta||$ be the number of atoms used to write Δ. If a is an atom and s is a simple element, then testing whether $a \preccurlyeq s$ (resp. $s \succcurlyeq a$) and, if so, computing $a^{-1}s$ (resp. sa^{-1}) can be performed effectively and we let the cost of this operation be $\mathcal{O}(C)$.

Let $T \in \mathbb{Z}$ such that there exist two integers $0 \leq i < j \leq T$ satisfying $\mathfrak{s}^i(x) = \mathfrak{s}^j(x)$ (i.e, T is an upper bound on the distance to a cyclic sliding repetition). Let $M \in \mathbb{Z}$ be such that for any element $z \in \mathrm{SC}(x)$, there exists a positive integer $N \leq M$ such that $\mathfrak{s}^N(z) = z$ (i.e., M is an upper bound on the length of sliding circuits in $\mathrm{SC}(x)$). Let $R \in \mathbb{Z}$ be such that for any element $z \in \mathrm{SC}(x)$ and any simple element satisfying $s^{-1}\, z\, s \in \mathrm{SSS}(x)$, there exist

two integers $0 \leq i < j \leq R$ satisfying $s^{(iN)} = s^{(jN)}$, where $\mathfrak{s}^N(z) = z$ and $s^{(m)}$ denotes the m-fold transport at z for m a non-negative integer (i.e., R is an upper bound on the distance to repetition of transports).

The authors determine that the complexity of Algorithm 6.27 is

$$\mathcal{O}\big(C\lambda k||\Delta|| \cdot (k + T + |\mathrm{SC}(x)|\lambda(||\Delta|| + RM))\big).$$

Unfortunately the only upper bound known for $|\mathrm{SC}(x)|$ is $|\mathrm{SSS}(x)|$, but the authors do remark that these upper bounds are crude and do not describe the behaviour observed in computer experiments.

Chapter 7

Conclusion

Let $C(x)$ denote the conjugacy class of $x \in G$ (or $C(X)$ for $X \in B_n$), i.e., the set of all elements which are conjugate to x. We have investigated how to solve the conjugacy decision and search problems using the following finite invariant subsets of the conjugacy class of x:

- The **summit set** of x, from [Gar69]:
$$\mathrm{SS}(x) = \{y \in C(x) \mid \inf y \text{ is maximal in } C(x)\}.$$
- The **super summit set** of x, from [ERM94] and improved in [FGM03]:
$$\mathrm{SSS}(x) = \{y \in C(x) \mid \ell(y) \text{ is minimal in } C(x)\}.$$
- The **ultra summit set** of x, from [Geb05] and improved in [BGGM08]:
$$\mathrm{USS}(x) = \{y \in \mathrm{SSS}(x) \mid \mathbf{c}^k(y) = y \text{ for some } k \geq 1\}.$$
- The **set of sliding circuits**, from [GGM10b]:
$$\mathrm{SC}(x) = \{y \in C(x) \mid \mathfrak{s}^k(y) = y \text{ for some } k \geq 1\}.$$

In general, we have that $\mathrm{SC}(x) \subseteq \mathrm{USS}(x) \subseteq \mathrm{SSS}(x) \subseteq \mathrm{SS}(x)$, and for all of these invariant subsets, x and y in G are conjugate if and only if they have the same invariant subset of their conjugacy class. Furthermore, as all of these sets satisfy the so-called convexity property, we can use these invariant subset to find a conjugating element between x and y if they are conjugate.

There is another invariant subset that we did not study in this document, the reduced super summit set from [Lee00]:

$$\mathrm{RSSS}(x) = \{y \in C(x) \mid \mathbf{c}^j(y) = y \text{ and } \mathbf{d}^k(y) = y \text{ for some } j, k \geq 1\}.$$

When the summit length $\ell_*(x)$ of x is greater than 1, $\mathrm{SC}(x) = \mathrm{RSSS}(x)$, and when $\ell_*(x) = 1$, $\mathrm{RSSS}(x) = \mathrm{USS}(x) = \mathrm{SSS}(x)$. Due to this overlap I decided to omit the discussion of $\mathrm{RSSS}(x)$. It would have been interesting to see its development from a historical point of view, though, especially since it was also proposed as part of a PhD thesis, just like $\mathrm{SS}(x)$ and $\mathrm{SSS}(x)$.

Appendix A

Computational examples in B_n

A.1 Cycling and decycling changing the canonical length

This section contains an example of cycling increasing inf and decycling decreasing sup. Unfortunately, I have not been able to construct an example in which both cycling and decycling are necessary to reach the summit length, but we can see that cycling[1] and decycling indeed behave as stated in Section 3.2.

Cycling

Let $X = \sigma_2\sigma_1\sigma_3\sigma_2\sigma_2\sigma_1\sigma_2 \in B_4^+$, which is $\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1 \cdot \sigma_2\sigma_1$ in left-canonical form. Thus, $\inf X = 0$, $\sup X = 2$, and $\ell(X) = 2$.

$\mathbf{c}(X) = \sigma_2\sigma_1 \cdot \sigma_2\sigma_1\sigma_3\sigma_2\sigma_1$, which is $\Delta\sigma_2$ in left-canonical form. Hence, $\inf \mathbf{c}(X) = 1$. In this case, cycling also decreased sup, but this is not necessarily the case in general.

For the sake of interest, $\mathrm{SSS}(X) = \{\Delta\sigma_2\}$, which I found using the software available at [Thi22]

Decycling

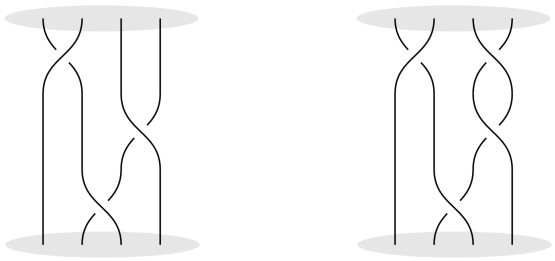
Let $Y = \sigma_1^{-1}\sigma_3\sigma_3\sigma_2 \in B_4$, which is $\Delta^{-1}\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2 \cdot \sigma_3 \cdot \sigma_3\sigma_2$ in left-canonical form. Hence, $\inf Y = -1$, $\sup Y = 2$, and $\ell(Y) = 3$.

$\mathbf{d}(Y) = \sigma_3\sigma_2\Delta^{-1}\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2 \cdot \sigma_3$ which is $\Delta^{-1}\sigma_1\sigma_2\sigma_1 \cdot \sigma_1\sigma_2\sigma_1\sigma_3\sigma_2$ in left-canonical form. Thus $\ell(Y) = 2$ and $\sup Y = \ell(Y) + \inf Y = 1$, with inf remaining the same.

A.2 Computing $\wedge^{\curvearrowleft}$, $\vee^{\curvearrowleft}$, $\wedge$, and $\vee$

Based on the methods discussed in Section 4.1, we find $\wedge$, $\vee$, $\wedge^{\curvearrowleft}$, and $\vee^{\curvearrowleft}$ of two example braids. Let $X_1 = \sigma_1^{-1}\sigma_3\sigma_2$ (see figure A.1a) and $X_2 = \sigma_3\sigma_1^{-1}\sigma_3\sigma_2$ (figure A.1b).

[1] The braid used to illustrate that cycling increases inf is based on an example given in [ERM94].



(a) (b)

Figure A.1

We can make an educated guess regarding the results of some of the computations, based on our understanding of the left- and right gcd and left- and right lcm.

As they are currently written, it is clear that $X_2 = \sigma_3 X_1$. Thus, $X_1 \wedge^{\curvearrowleft} X_2 = X_1$ and $X_1 \vee^{\curvearrowleft} X_2 = X_2$. Even if we did not realise this, we can see that both braids can end with the braid $\sigma_1^{-1}\sigma_3\sigma_2$, where the remaining factor at the start is positive; hence, the right gcd is $\sigma_1^{-1}\sigma_3\sigma_2 = X_1$. On the other hand, we can concatenate a factor of σ_3 at the start of X_1 to create a multiple which can end with both X_1 and X_2, and hence $\sigma_3 X_1 = X_2$ is the right lcm.

The left gcd is slightly less obvious. If we rewrite X_2 as $\sigma_1^{-1}\sigma_3\sigma_3\sigma_2$ (due to the first braid relation), we see that both braids can start with $\sigma_1^{-1}\sigma_3$ such that the factor remaining at the end is positive. Thus, we predict the left gcd as $X_1 \wedge X_2 = \sigma_1^{-1}\sigma_3$. The left lcm is not at all obvious. We will have to concatenate factors on the right of both X_1 and X_2 to create a multiple which could start with either braid.

$\wedge^{\curvearrowleft}$, the right greatest common divisor

Write $(X_1 X_2^{-1})^-$ as $U_1^{-1}U_2$ in mixed normal form:

$$\begin{aligned}(\sigma_1^{-1}\sigma_3\sigma_2 \cdot \sigma_2^{-1}\sigma_3^{-1}\sigma_1\sigma_3^{-1})^- &= \sigma_1\sigma_3^{-1}\sigma_2^{-1}\sigma_2\sigma_3\sigma_1^{-1}\sigma_3 \\ &= \sigma_1\sigma_3^{-1}\sigma_3\sigma_1^{-1}\sigma_3 \\ &= \sigma_1\sigma_1^{-1}\sigma_3 \\ &= \sigma_3 \\ &= 1^{-1}\cdot\sigma_3\end{aligned}$$

This is in mixed normal form. $U_1 = 1$ and $U_2 = \sigma_3$. We get:

$$\begin{array}{rcccc} X_1 \wedge^{\curvearrowleft} X_2 & = & U_1^- X_1 & = & U_2^- X_2 \\ & & \| & & \| \\ & & 1^{-1} \cdot \sigma_1^{-1}\sigma_3\sigma_2 & = & \sigma_3^{-1} \cdot \sigma_3\sigma_1^{-1}\sigma_3\sigma_2 \\ & & \searrow & & \swarrow \\ & & & \sigma_1^{-1}\sigma_3\sigma_2 & \\ & & & \| & \\ & & & X_1, & \end{array}$$

as predicted.

$\vee^{\curvearrowleft}$, the right least common multiple

Write $X_1 X_2^{-1}$ as $U_1^{-1}U_2$ in mixed normal form:

$$\begin{aligned} \sigma_1^{-1}\sigma_3\sigma_2 \cdot (\sigma_3\sigma_1^{-1}\sigma_3\sigma_2)^{-1} &= \sigma_1^{-1}\sigma_3\sigma_2\sigma_2^{-1}\sigma_3^{-1}\sigma_1\sigma_3^{-1} \\ &= \sigma_1^{-1}\sigma_3\sigma_3^{-1}\sigma_1\sigma_3^{-1} \\ &= \sigma_1^{-1}\sigma_1\sigma_3^{-1} \\ &= \sigma_3^{-1} \\ &= \sigma_3^{-1} \cdot 1 \end{aligned}$$

This is in mixed normal form. $U_1 = \sigma_3$ and $U_2 = 1$. We get:

$$\begin{array}{rcccc} X_1 \vee^{\curvearrowleft} X_2 & = & U_1 X_1 & = & U_2 X_2 \\ & & \| & & \| \\ & & \sigma_3 \cdot \sigma_1^{-1}\sigma_3\sigma_2 & = & 1 \cdot \sigma_3\sigma_1^{-1}\sigma_3\sigma_2 \\ & & \searrow & & \swarrow \\ & & & \sigma_3\sigma_1^{-1}\sigma_3\sigma_2 & \\ & & & \| & \\ & & & X_2, & \end{array}$$

as predicted.

$\wedge$, the left greatest common divisor

Write $X_1^{-1}X_2$ as $U_1^{-1}U_2$ in mixed normal form:

$$\begin{aligned} (\sigma_1^{-1}\sigma_3\sigma_2)^{-1} \cdot \sigma_3\sigma_1^{-1}\sigma_3\sigma_2 &= \sigma_2^{-1}\sigma_3^{-1}\sigma_1\sigma_3\sigma_1^{-1}\sigma_3\sigma_2 \\ &= \sigma_2^{-1}\sigma_3^{-1}\sigma_3\sigma_1\sigma_1^{-1}\sigma_3\sigma_2 \\ &= \sigma_2^{-1}\sigma_1\sigma_1^{-1}\sigma_3\sigma_2 \\ &= \sigma_2^{-1}\sigma_3\sigma_2 \\ &= \sigma_2^{-1} \cdot \sigma_3\sigma_2 \end{aligned}$$

This is in mixed normal form. $U_1 = \sigma_2$ and $U_2 = \sigma_3\sigma_2$. We get:

$$\begin{array}{ccccc} X_1 \wedge X_2 & = & X_1U_1^{-1} & = & X_2U_2^{-1} \\ & & \| & & \| \\ & & \sigma_1^{-1}\sigma_3\sigma_2 \cdot \sigma_2^{-1} & = & \sigma_3\sigma_1^{-1}\sigma_3\sigma_2 \cdot \sigma_2^{-1}\sigma_3^{-1} \\ & & \| & & \| \\ & & \sigma_1^{-1}\sigma_3 & & \sigma_3\sigma_1^{-1} \\ & & \diagdown\!\!\diagdown & & \diagup\!\!\diagup \\ & & & \sigma_1^{-1}\sigma_3, & \end{array}$$

as predicted.

$\boxed{\vee, \text{ the left least common multiple}}$

At last, the one we were not able to predict.

Write $(X_1^{-1}X_2)^-$ as $U_1^{-1}U_2$ in mixed normal form:

$$\begin{aligned} ((\sigma_1^{-1}\sigma_3\sigma_2)^{-1} \cdot \sigma_3\sigma_1^{-1}\sigma_3\sigma_2)^- &= (\sigma_2^{-1}\sigma_3^{-1}\sigma_1 \cdot \sigma_3\sigma_1^{-1}\sigma_3\sigma_2)^- \\ &= \sigma_2\sigma_3\sigma_1^{-1} \cdot \sigma_3^{-1}\sigma_1\sigma_3^{-1}\sigma_2^{-1} \\ &= \sigma_2\sigma_1^{-1}\sigma_3 \cdot \sigma_3^{-1}\sigma_1\sigma_3^{-1}\sigma_2^{-1} \\ &= \sigma_2\sigma_1^{-1}\sigma_1\sigma_3^{-1}\sigma_2^{-1} \\ &= \sigma_2\sigma_3^{-1}\sigma_2^{-1} \end{aligned}$$

This is **not** in mixed canonical form yet. Consider $\sigma_2\sigma_3^{-1}\sigma_2^{-1}$ as a geometric braid. We need to move the positive crossing down so that the two negative crossings occur first. This is illustrated in figure A.2.

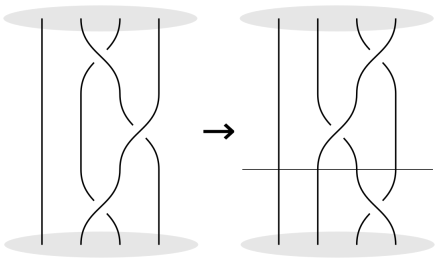


Figure A.2

Thus, we have that $\sigma_2\sigma_3^{-1}\sigma_2^{-1} = \sigma_3^{-1}\sigma_2^{-1} \cdot \sigma_3 = (\sigma_2\sigma_3)^{-1} \cdot \sigma_3$, which is in mixed canonical form as written (no work is necessary to canonicalise the negative or positive braid). Hence, $U_1 = \sigma_2\sigma_3$ and $U_2 = \sigma_3$.

$$\begin{array}{ccccc} X_1 \vee X_2 & = & X_1(U_1^-)^{-1} & = & X_2(U_2^-)^{-1} \\ & & \| & & \| \\ & & \sigma_1^{-1}\sigma_3\sigma_2 \cdot (\sigma_2^{-1}\sigma_3^{-1})^{-1} & = & \sigma_3\sigma_1^{-1}\sigma_3\sigma_2 \cdot (\sigma_3^{-1})^{-1} \\ & & \| & & \| \\ & & \sigma_1^{-1}\sigma_3\sigma_2\sigma_3\sigma_2 & = & \sigma_3\sigma_1^{-1}\sigma_3\sigma_2\sigma_3 \\ & & \diagdown\!\!\diagdown & & \diagup\!\!\diagup \\ & & & \sigma_1^{-1}\sigma_3\sigma_2\sigma_3\sigma_2 & \end{array}$$

By the first braid relation, $\sigma_3\sigma_1^{-1}\sigma_3\sigma_2\sigma_3 = \sigma_1^{-1}\sigma_3\sigma_3\sigma_2\sigma_3$. By the second braid relation, this becomes $\sigma_1^{-1}\sigma_3\sigma_2\sigma_3\sigma_2$, and so we have our final result.

A.3 Finding a minimal simple element in $\mathrm{USS}(x)$ using transports and pullbacks

We discuss Example 4.5 in [Geb05] which continues in Example 4.10 of the paper. Let $X = \sigma_3\sigma_2\sigma_1\sigma_2\sigma_3 \cdot \sigma_3 \in B_4^+$, which is in left-canonical form as written. Let $S = \sigma_1$. Firstly, $\mathbf{c}^3(X) = \mathbf{d}^3(X) = X$, so $X \in \mathrm{USS}(X)$ with $N = 3$. $\mathbf{c}(X) = \sigma_1\sigma_2\sigma_3\sigma_2 \cdot \sigma_2\sigma_1$ and $\mathbf{c}^2(X) = \sigma_2\sigma_1\sigma_3 \cdot \sigma_1\sigma_2\sigma_3$ also have canonical length 2, so we are certain $X \in \mathrm{SSS}(X)$. $S^{-1}\,X\,S = \sigma_2\sigma_3\sigma_2\sigma_1 \cdot \sigma_1\sigma_3$, which also has canonical length 2, so $S^{-1}\,X\,S \in \mathrm{SSS}(X)$, thus $\rho_S = S$. Let's find $S^{(1)} = \phi_X(S^{(0)})$.

We know the canonical form of X, and in particular $\inf X = 0$. Thus, applying equation 5.1 we get

$$\begin{aligned} \phi_X(S) &= \tau^0(\sigma_3 \cdot \sigma_1) \wedge \Delta\tau^{1-0}(\sigma_3^{-1}\sigma_2^{-1}\sigma_1^{-1}\sigma_2^{-1}\sigma_3^{-1})\tau(\sigma_1) \\ &= \sigma_3 \cdot \sigma_1 \wedge \sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1 \cdot \sigma_1^{-1}\sigma_2^{-1}\sigma_3^{-1}\sigma_2^{-1}\sigma_1^{-1} \cdot \sigma_3 \\ &= \sigma_3\sigma_1 \wedge \sigma_2\sigma_3 \end{aligned}$$

where the last step occurs via simplification using the braid relations. We can clearly see these two braids have no prefix in common, and double-checking with the left gcd method from Section 4.1, we conclude that

$$\phi_X(S) = S^{(1)} = 1.$$

We could also check and see that $\mathbf{c}(X) = \mathbf{c}(S^{-1}\,X\,S) = \sigma_1\sigma_2\sigma_3\sigma_2 \cdot \sigma_2\sigma_1$, so the transport of $S = \sigma_1$ is indeed the identity. Thus, $F_X(S) = \{1\}$, and neither requirements of Lemma 5.1.15 are satisfied, so we have to find p_X.

To do so, we start by computing iterated pullbacks of $S = S_{(0)} = \sigma_1$, and look for i_1 and i_2 such that $S_{(i_1 \cdot 3)} = S_{(i_2 \cdot 3)}$ and $i_2 > i_1 \geq 0$ and minimal such. We know $S_{(1)} = \pi_{\mathbf{c}^2(X)}(S_{(0)})$ (since $\alpha \equiv -1 \mod 3 = 2$) and $\mathbf{c}^2(X) = \sigma_2\sigma_1\sigma_3 \cdot \sigma_1\sigma_2\sigma_3$, so we can use Proposition 5.1.16

and get

$$\begin{aligned} b &= \left(1 \vee \tau^0(\sigma_2\sigma_1\sigma_3)\sigma_1\Delta^{-1}\right) \vee \left(1 \vee \sigma_3^{-1}\sigma_2^{-1}\sigma_1^{-1}\tau^0(\sigma_1)\right) \\ &= (1 \vee \sigma_2\sigma_1\sigma_3 \cdot \sigma_1 \cdot \sigma_1^{-1}\sigma_2^{-1}\sigma_1^{-1}\sigma_3^{-1}\sigma_2^{-1}\sigma_1^{-1}) \vee (1 \vee \sigma_3^{-1}\sigma_2^{-1}\sigma_1^{-1} \cdot \sigma_1) \\ &= (1 \vee \sigma_1^{-1}\sigma_3^{-1}\sigma_2^{-1}\sigma_3) \vee (1 \vee \sigma_3^{-1}\sigma_2^{-1}) \\ &= \sigma_2\sigma_1 \vee 1 \\ &= \sigma_2\sigma_1 \end{aligned}$$

where the third line occurs via simplification using the braid relations, and the last two lines are found using the left lcm method from Section 4.1. In this case (and in all the pullback computations in this example), b conjugates the element in question to an element in $\mathrm{SSS}(X)$, and so

$$b = \rho_b = \pi_{\mathbf{c}^2(X)}(S_{(0)}) = S_{(1)} = \sigma_2\sigma_1.$$

By the same method, we get $S_{(2)} = \pi_{\mathbf{c}(X)}(S_{(1)}) = \sigma_3$, $S_{(3)} = \pi_X(S_{(2)}) = \sigma_1\sigma_2$, $S_{(4)} = \pi_{\mathbf{c}^2(X)}(S_{(3)}) = \sigma_2\sigma_1$, $S_{(5)} = \pi_{\mathbf{c}(X)}(S_{(4)}) = \sigma_3$, and $S_{(6)} = \pi_X(S_{(5)}) = \sigma_1\sigma_2$.

We observe that $S_{(3)} = S_{(6)}$, i.e., $S_{(1\cdot 3)} = S_{(2\cdot 3)}$, and both of these conjugate $\mathbf{c}^3(X) = X$ to an element in $\mathrm{SSS}(X)$, so by Proposition 5.1.17, we have $i_1 = 1$, $i_2 = 2$, $l = 2 - 1 = 1$, and we choose $j = 1$ to get $p_X = p_X(S) = S_{(1\cdot 1\cdot 3)} = S_{(3)} = \sigma_1\sigma_2$. To find $\mathfrak{c}_S$, we need to find $F_X(p_X)$ using iterated transport.

We let $p_X = p_X^{(0)} = \sigma_1\sigma_2$. We start by looking for for i_1 and i_2 such that $p_X^{(i_1\cdot 3)} = p_X^{(i_2\cdot 3)}$ with $i_2 > i_1 \geq 0$ and minimal such. Using equation 5.1 we get $p_X^{(1)} = \phi_X(p_X^{(0)}) = \sigma_3$, $p_X^{(2)} = \phi_{\mathbf{c}(X)}(p_X^{(1)}) = \sigma_2\sigma_1$, $p_X^{(3)} = \phi_{\mathbf{c}^2(X)}(p_X^{(2)}) = \sigma_1\sigma_2\sigma_3$, $p_X^{(4)} = \phi_{\mathbf{c}^3(X)}(p_X^{(3)}) = \sigma_3$, $p_X^{(5)} = \phi_{\mathbf{c}^4(X)}(p_X^{(4)}) = \sigma_2\sigma_1$, and $p_X^{(6)} = \phi_{\mathbf{c}^5(X)}(p_X^{(5)}) = \sigma_1\sigma_2\sigma_3$.

Notice that $p_X^{(3)} = p_X^{(6)}$, i.e., $p_X^{(1\cdot 3)} = p_X^{(2\cdot 3)}$, so $i_1 = 1$, $i_2 = 2$, and so $F_X(p_X) = \{p_X^{(3)}\} = \{\sigma_1\sigma_2\sigma_3\}$. Clearly $s = \sigma_1 \preccurlyeq \sigma_1\sigma_2\sigma_3 = p_X^{(3)}$, and so

$$\mathfrak{c}_S = \sigma_1\sigma_2\sigma_3.$$

We can compute $\mathfrak{c}_S^{-1}\, X\, \mathfrak{c}_S$, to obtain $\mathfrak{c}_S^{-1}\, X\, \mathfrak{c}_S = \sigma_2\sigma_1\sigma_3 \cdot \sigma_1\sigma_2\sigma_3 = \mathbf{c}^2(X)$. We know $\mathbf{c}^2(X) \in \mathrm{USS}(X)$, so while this may not conjugate to a distinct trajectory, it does confirm that the method finds an element of S_X^{USS}

Notice that $p_X \notin F_X(p_X)$, making it clear that we need to transport p_X in order to find $\mathfrak{c}_S$, despite reaching a stable loop under iterated pullback.

For the sake of interest, $\mathrm{USS}(X) = \{X, \mathbf{c}(X), \mathbf{c}^2(X), \tau(X), \tau(\mathbf{c}(X)), \tau(\mathbf{c}^2(X))\}$, which I determined using the software available at [Thi22].

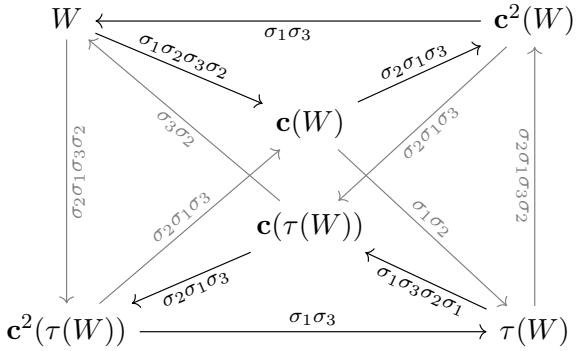
A.4 Partial cycling and partial twisted decycling in γ_x

We consider the graph γ_X for a few different braids X, to get an idea of what this graph may look like, paying attention to black and grey arrows (which correspond to partial cycling and

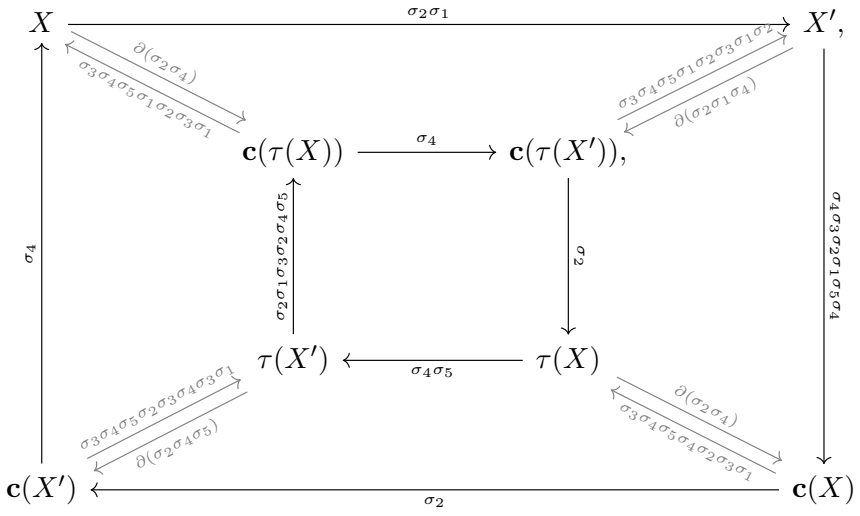
partial twisted decycling, respectively). We do not focus on how the entire γ_X is obtained. We can use Algorithms 5.2.14 and 5.2.15 repeatedly, until we have generated all the vertices and arrows in the graph. We are more interested in the structure of the graphs in these examples, as it adds intuition to many of the results discussed in Section 5.2.

All of the braids X for which we consider γ_X satisfy $X \in \mathrm{USS}(X)$.

Example A.4.1 (Example 2.11 in [BGGM08])**.** In this case, each black arrow is a cycling and each grey arrow is a twisted decycling. Let $W = \sigma_1\sigma_2\sigma_3\sigma_2 \cdot \sigma_2\sigma_1\sigma_3 \cdot \sigma_1\sigma_3 \in B_4^+$. $\mathrm{USS}(W)$ has 2 orbits under cycling, each of length 3, and so has 6 elements in total. In particular, $\mathbf{c}^3(W) = W$ and $\mathbf{c}^3(\tau(W)) = \tau(W)$. γ_W has 2 black components (corresponding to the 2 orbits under cycling) and 1 grey component.

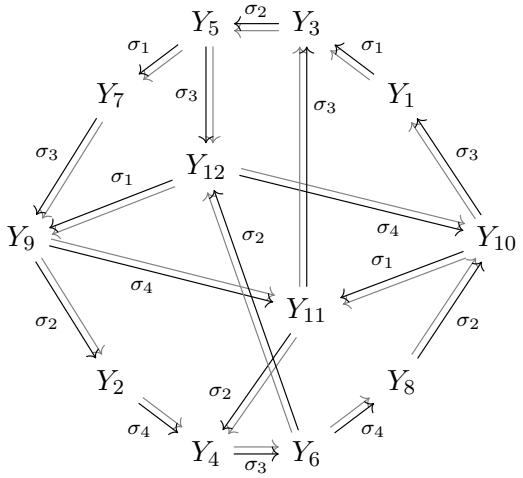


Example A.4.2 (Example 2.12 in [BGGM08])**.** Here each black arrow is a partial cycling and each grey arrow is a twisted decycling. Let $X = \sigma_2\sigma_1\sigma_4\sigma_3\sigma_2\sigma_1\sigma_5\sigma_4 \cdot \sigma_2\sigma_4 \in B_6^+$. $\mathrm{USS}(X)$ has 4 orbits of length 2 under cycling, and so has 8 elements in total. In particular, we have $\mathbf{c}^2(X) = X$, $\mathbf{c}^2(\tau(X)) = \tau(X)$, $\mathbf{c}^2(X') = X'$, and $\mathbf{c}^2(\tau(X')) = \tau(X')$ where $X' = \sigma_1^{-1}\sigma_2^{-1} X \sigma_2\sigma_1$ (obtained via the minimal simple element $\mathfrak{c}_{\sigma_2}(X) = \sigma_2\sigma_1$). In the black components, the concatenation of two consecutive arrows corresponds to a cycling. There are 2 black components and 4 grey components in γ_X.

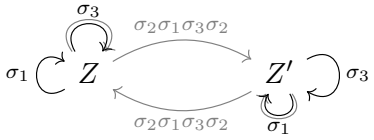


The labels of the grey arrows are not given in the example, but since the authors state that the grey arrows correspond to twisted decyclings, we can find the labels of the grey arrows as $\partial(\varphi(\cdot))$. For space consideration, I kept the right complement notation if the final result is too long to fit in the diagram.

Example A.4.3 (Example 2.13 in [BGGM08])**.** Now every arrow is bi-coloured, since each conjugation corresponds to both a partial cycling and a partial twisted decycling. Let $Y = \sigma_4\sigma_1\sigma_2\sigma_3\sigma_4 \in B_5^+$. This is already in left-canonical form, and so $\inf Y = 0$, $\sup Y = 1$, and $\ell(Y) = 1$. Thus, $\mathbf{c}(Y) = Y$. USS(Y) has 12 elements (each also of canonical length 1) and there is 1 black component and 1 grey component in γ_Y. The 11 other elements of USS(Y) is not given in the paper, but they are easy to compute by following the arrows and conjugating. Let $Y_1 = Y$, then we have $Y_2 = \sigma_4\sigma_3\sigma_2\sigma_1\sigma_2$, $Y_3 = \sigma_4\sigma_2\sigma_3\sigma_4\sigma_1$, $Y_4 = \sigma_3\sigma_2\sigma_1\sigma_2\sigma_4$, $Y_5 = \sigma_4\sigma_3\sigma_4\sigma_1\sigma_2$, $Y_6 = \sigma_2\sigma_1\sigma_2\sigma_4\sigma_3$, $Y_7 = \sigma_4\sigma_3\sigma_4\sigma_2\sigma_1$, $Y_8 = \sigma_2\sigma_1\sigma_2\sigma_3\sigma_4$, $Y_9 = \sigma_4\sigma_3\sigma_2\sigma_1\sigma_3$, $Y_{10} = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_2$, $Y_{11} = \sigma_2\sigma_3\sigma_4\sigma_2\sigma_1$, $Y_{12} = \sigma_4\sigma_3\sigma_1\sigma_2\sigma_3$ and USS$(Y) = \{Y_1, \ldots, Y_{12}\}$.



Example A.4.4 (Example 2.14 in [BGGM08])**.** In this case, the black arrows correspond to partial cyclings, the grey arrows correspond to partial twisted decyclings, and where they occur simultaneously (i.e. the bi-coloured arrows) the conjugation corresponds to both. Let $Z = \sigma_1\sigma_3 \cdot \sigma_1 \in B_4^+$. USS$(Z)$ consists of 2 elements, Z and $Z' = \sigma_1\sigma_3 \cdot \sigma_3$. These satisfy $\mathbf{c}(Z) = Z$ and $\mathbf{c}(Z') = Z'$, and only connect to one another in γ_Z via partial twisted decycling. γ_Z has 2 black components and 1 grey component.



A.5 Finding a minimal simple element in $\mathrm{SC}(x)$ using transports and pullbacks

We consider $Y = X = \Delta\sigma_2\sigma_1\sigma_4\sigma_3\sigma_4 \cdot \sigma_1$ (in left-canonical form) with $S = \sigma_3\sigma_2\sigma_1$, as discussed in examples 3.11 and 3.20 in [GGM10b]. We start by seeing if $Y \in \mathrm{SC}(x)$ and what the length of its sliding circuit is. We get the following preferred prefixes and cyclic slidings:

$$\begin{aligned} Y &= \Delta\sigma_2\sigma_1\sigma_3\sigma_4\sigma_3 \cdot \sigma_1 & \mathfrak{p}(Y) &= \sigma_3\sigma_2\sigma_1\sigma_4 \\ \mathfrak{s}(Y) &= \Delta\sigma_1\sigma_3 \cdot \sigma_3\sigma_2\sigma_1\sigma_4 & \mathfrak{p}(\mathfrak{s}(Y)) &= \sigma_2 \\ \mathfrak{s}^2(Y) &= \Delta\sigma_1\sigma_3\sigma_2\sigma_1\sigma_4 \cdot \sigma_2 & \mathfrak{p}(\mathfrak{s}^2(Y)) &= \sigma_4 \\ \mathfrak{s}^3(Y) &= \Delta\sigma_1\sigma_3\sigma_2\sigma_1\sigma_4 \cdot \sigma_4 & \mathfrak{p}(\mathfrak{s}^3(Y)) &= \sigma_2\sigma_1\sigma_3\sigma_4 \\ \mathfrak{s}^4(Y) &= \Delta\sigma_2\sigma_4 \cdot \sigma_2\sigma_1\sigma_3\sigma_4 & \mathfrak{p}(\mathfrak{s}^4(Y)) &= \sigma_3 \\ \mathfrak{s}^5(Y) &= \Delta\sigma_2\sigma_1\sigma_3\sigma_4\sigma_3 \cdot \sigma_3 & \mathfrak{p}(\mathfrak{s}^5(Y)) &= \sigma_1 \\ \mathfrak{s}^6(Y) &= \Delta\sigma_2\sigma_1\sigma_3\sigma_4\sigma_3 \cdot \sigma_1 = Y \end{aligned}$$

Thus $\mathfrak{s}^6(y) = y$, and so $y \in \mathrm{SC}(x)$. The length of the sliding circuit of Y is $N = 6$.

Next, we find that $S^{-1}\,Y\,S = \Delta\sigma_1\sigma_3 \cdot \sigma_1\sigma_3\sigma_2\sigma_1$, which has canonical length 2, so $S^{-1}\,Y\,S \in \mathrm{SSS}(x)$. This means that $\rho(S^{-1}\,Y\,S) = 1$, and so $\rho_S = S \cdot \rho(S^{-1}\,Y\,S) = S$. We can find the transport of $S = \rho_S$ at Y via the equation $S^{(1)} = \mathfrak{p}(Y)^{-1}\,S\,\mathfrak{p}(S^{-1}\,Y\,S)$. We already know $\mathfrak{p}(Y)$, but we need to find $\mathfrak{p}(S^{-1}\,Y\,S)$. Note that $\iota(S^{-1}\,Y\,S) = \tau^{-1}(\sigma_1\sigma_3) = \sigma_4\sigma_2$ and $\partial(\varphi(x)) = \partial(\sigma_1\sigma_3\sigma_2\sigma_1) = \sigma_3\sigma_2\sigma_4\sigma_3\sigma_2\sigma_1$. Thus, $\mathfrak{p}(S^{-1}\,Y\,S) = \iota(S^{-1}\,Y\,S) \wedge \partial(\varphi(S^{-1}\,Y\,S)) = \sigma_4$. Putting this together, we have

$$S^{(1)} = (\sigma_3\sigma_2\sigma_1\sigma_4)^{-1} \cdot \sigma_3\sigma_2\sigma_1 \cdot \sigma_4 = 1.$$

This means $F(S) = F(\rho_S) = \{1\}$, and so iterated transports is not enough to find κ_S. We need to use pullbacks first, and then we can use transports.

We use equation 6.2 repeatedly to find the iterated pullbacks of S. Notice that we already know $\mathfrak{p}(\mathfrak{s}^\beta(Y))$ and $\mathfrak{s}^\alpha(Y)$ for any $0 \le \alpha, \beta \le 5$, as we had to find these in order to find $\mathfrak{s}^6(Y)$. For each pullback $S_{(k)}$ we will need to find $\mathfrak{p}^{\curvearrowleft}(S_{(k-1)}^{-1}\,\mathfrak{s}^\alpha(Y)\,S_{(k-1)})$ where $0 \le \alpha \equiv -(k-1) \mod 6$, before we can find the actual pullback. All of this can be done by hand, but there is software available at [Thi22] which speeds the process up dramatically. However, there is no $\wedge^{\curvearrowleft}$ option in the software (only $\wedge$ and $\vee$), so when I did the computations for this example, I used statement (iv) of Lemma 4.2.2 to rework $\mathfrak{p}^{\curvearrowleft}$ so that it can be found using $\vee$ instead. In particular, we get that for $U \in B_n$,

$$\mathfrak{p}^{\curvearrowleft}(U) = \left(\iota^{\curvearrowleft}(U)^{-1} \vee \iota^{\curvearrowleft}(U^{-1})^{-1}\right)^{-1} = \left(\iota^{\curvearrowleft}(U)^{-1} \vee (\partial^{-1}(\varphi^{\curvearrowleft}(U)))^{-1}\right)^{-1}.$$

Since the software also computes the left- and right canonical form of an arbitrary braid with ease, $\mathfrak{p}^{\curvearrowleft}$ can be found without pain. Thus, we have the iterated transports of S as follows:

$$
\begin{aligned}
S_{(1)} &= \left(\mathfrak{p}\left(\mathfrak{s}^5(Y)\right)S_{(0)}\mathfrak{p}^{\curvearrowleft}\left(S_{(0)}^{-1}\mathfrak{s}^0(Y)S_{(0)}\right)^{-1}\right)\vee 1 && = (\sigma_1\cdot\sigma_3\sigma_2\sigma_1\cdot(\sigma_3\sigma_2\sigma_1)^{-1})\vee 1 && = \sigma_1\\
S_{(2)} &= \left(\mathfrak{p}\left(\mathfrak{s}^4(Y)\right)S_{(1)}\mathfrak{p}^{\curvearrowleft}\left(S_{(1)}^{-1}\mathfrak{s}^5(Y)S_{(1)}\right)^{-1}\right)\vee 1 && = (\sigma_3\cdot\sigma_1\cdot(\sigma_1\sigma_4)^{-1})\vee 1 && = \sigma_3\\
S_{(3)} &= \left(\mathfrak{p}\left(\mathfrak{s}^3(Y)\right)S_{(2)}\mathfrak{p}^{\curvearrowleft}\left(S_{(2)}^{-1}\mathfrak{s}^4(Y)S_{(2)}\right)^{-1}\right)\vee 1 && = (\sigma_2\sigma_1\sigma_3\sigma_4\cdot\sigma_3\cdot(\sigma_1\sigma_4\sigma_3)^{-1})\vee 1 && = \sigma_2\sigma_3\\
S_{(4)} &= \left(\mathfrak{p}\left(\mathfrak{s}^2(Y)\right)S_{(3)}\mathfrak{p}^{\curvearrowleft}\left(S_{(3)}^{-1}\mathfrak{s}^3(Y)S_{(3)}\right)^{-1}\right)\vee 1 && = (\sigma_4\cdot\sigma_2\sigma_3\cdot(\sigma_2\sigma_3)^{-1})\vee 1 && = \sigma_4\\
S_{(5)} &= \left(\mathfrak{p}\left(\mathfrak{s}^1(Y)\right)S_{(4)}\mathfrak{p}^{\curvearrowleft}\left(S_{(4)}^{-1}\mathfrak{s}^2(Y)S_{(4)}\right)^{-1}\right)\vee 1 && = (\sigma_2\cdot\sigma_4\cdot(\sigma_1\sigma_4)^{-1})\vee 1 && = \sigma_2\\
S_{(6)} &= \left(\mathfrak{p}\left(\mathfrak{s}^0(Y)\right)S_{(5)}\mathfrak{p}^{\curvearrowleft}\left(S_{(5)}^{-1}\mathfrak{s}^1(Y)S_{(5)}\right)^{-1}\right)\vee 1 && = (\sigma_3\sigma_2\sigma_1\sigma_4\cdot\sigma_2\cdot(\sigma_1\sigma_2\sigma_1)^{-1})\vee 1 && = \sigma_3\sigma_4\\
S_{(7)} &= \left(\mathfrak{p}\left(\mathfrak{s}^5(Y)\right)S_{(6)}\mathfrak{p}^{\curvearrowleft}\left(S_{(6)}^{-1}\mathfrak{s}^0(Y)S_{(6)}\right)^{-1}\right)\vee 1 && = (\sigma_1\cdot\sigma_3\sigma_4\cdot(\sigma_3\sigma_4\sigma_3)^{-1})\vee 1 && = \sigma_1\\
S_{(8)} &= \left(\mathfrak{p}\left(\mathfrak{s}^4(Y)\right)S_{(7)}\mathfrak{p}^{\curvearrowleft}\left(S_{(7)}^{-1}\mathfrak{s}^5(Y)S_{(7)}\right)^{-1}\right)\vee 1 && = (\sigma_3\cdot\sigma_1\cdot(\sigma_1\sigma_3\sigma_4)^{-1})\vee 1 && = \sigma_3\\
S_{(9)} &= \left(\mathfrak{p}\left(\mathfrak{s}^3(Y)\right)S_{(8)}\mathfrak{p}^{\curvearrowleft}\left(S_{(8)}^{-1}\mathfrak{s}^4(Y)S_{(8)}\right)^{-1}\right)\vee 1 && = (\sigma_2\sigma_1\sigma_3\sigma_4\cdot\sigma_3\cdot(\sigma_1\sigma_4\sigma_3)^{-1})\vee 1 && = \sigma_2\sigma_3\\
S_{(10)} &= \left(\mathfrak{p}\left(\mathfrak{s}^2(Y)\right)S_{(9)}\mathfrak{p}^{\curvearrowleft}\left(S_{(9)}^{-1}\mathfrak{s}^3(Y)S_{(9)}\right)^{-1}\right)\vee 1 && = (\sigma_4\cdot\sigma_2\sigma_3\cdot(\sigma_2\sigma_3)^{-1})\vee 1 && = \sigma_4\\
S_{(11)} &= \left(\mathfrak{p}\left(\mathfrak{s}^1(Y)\right)S_{(10)}\mathfrak{p}^{\curvearrowleft}\left(S_{(10)}^{-1}\mathfrak{s}^2(Y)S_{(10)}\right)^{-1}\right)\vee 1 && = (\sigma_2\cdot\sigma_4\cdot(\sigma_1\sigma_4)^{-1})\vee 1 && = \sigma_2\\
S_{(12)} &= \left(\mathfrak{p}\left(\mathfrak{s}^0(Y)\right)S_{(11)}\mathfrak{p}^{\curvearrowleft}\left(S_{(11)}^{-1}\mathfrak{s}^1(Y)S_{(11)}\right)^{-1}\right)\vee 1 && = (\sigma_3\sigma_2\sigma_1\sigma_4\cdot\sigma_2\cdot(\sigma_1\sigma_2\sigma_1)^{-1})\vee 1 && = \sigma_3\sigma_4
\end{aligned}
$$

For the sake of interest, we can check that these indeed conjugate the relevant elements to elements of $\mathrm{SSS}(x)$:

$$
\begin{aligned}
S_{(1)}^{-1}\mathfrak{s}^5(Y)S_{(1)} = S_{(7)}^{-1}\mathfrak{s}^5(Y)S_{(7)} &= Y\\
S_{(2)}^{-1}\mathfrak{s}^4(Y)S_{(2)} = S_{(8)}^{-1}\mathfrak{s}^4(Y)S_{(8)} &= \mathfrak{s}^5(Y)\\
S_{(3)}^{-1}\mathfrak{s}^3(Y)S_{(3)} = S_{(9)}^{-1}\mathfrak{s}^3(Y)S_{(9)} &= \Delta\sigma_1\sigma_2\sigma_4\cdot\sigma_2\sigma_4\sigma_3\\
S_{(4)}^{-1}\mathfrak{s}^2(Y)S_{(4)} = S_{(10)}^{-1}\mathfrak{s}^2(Y)S_{(10)} &= \mathfrak{s}^3(Y)\\
S_{(5)}^{-1}\mathfrak{s}^1(Y)S_{(5)} = S_{(11)}^{-1}\mathfrak{s}^1(Y)S_{(11)} &= \mathfrak{s}^2(Y)\\
S_{(6)}^{-1}\mathfrak{s}^0(Y)S_{(6)} = S_{(12)}^{-1}\mathfrak{s}^0(Y)S_{(12)} &= \Delta\sigma_1\sigma_3\sigma_4\sigma_3\cdot\sigma_3\sigma_4
\end{aligned}
$$

We know $\mathfrak{s}^i(Y)\in\mathrm{SSS}(x)$ for all i, and it is easy to see that the two new elements have canonical length 2 and are conjugate to x, and so are also elements of $\mathrm{SSS}(x)$.

Thus, we have found that $S_{(6)} = S_{(12)}$ (or, in the notation of Proposition 6.25, we have found that $i=1$ and $j=2$), so we let $p = S_{(6)} = \sigma_3\sigma_4$. We now consider $F(p) = F(S_{(6)})$ using iterated transports. To do so, we use equation 6.1 repeatedly. We have already found all the $\mathfrak{s}^i(Y)$ and their associated preferred prefixes (note that for $i > 6 = N$, we simply find $0 \le j \equiv i \mod 6$, so that $0 \le j < 5$ and then we have all $\mathfrak{s}^i(Y) = \mathfrak{s}^j(Y)$). To use the equation, however, we need to find all the $\mathfrak{s}^i(p^{-1}Yp)$ and their associated prefixes as well. This turns out to be quite simple in this case:

$p^{-1}\,Y\,p = \Delta\sigma_1\sigma_3\sigma_4\sigma_3\cdot\sigma_3\sigma_4$ and $\mathfrak{p}(p^{-1}\,Y\,p) = \sigma_1\sigma_2\sigma_1$, which leads to $\mathfrak{s}(p^{-1}\,Y\,p) = \Delta\sigma_1\sigma_3\sigma_2\sigma_1\sigma_4\cdot\sigma_2$, which is the same element as $\mathfrak{s}^2(Y)$. Now we are within the sliding circuit of Y again,

and we get that for $i \geq 1$, $\mathfrak{s}^i(p^{-1}\, Y\, p) = \mathfrak{s}^{i+1}(Y)$. Thus we already have all the information to find the iterated transports of $p = \sigma_3\sigma_4 = p^{(0)}$! In particular, we get $p^{(1)} = \sigma_2$, $p^{(2)} = \sigma_4$, $p^{(3)} = \sigma_2\sigma_1\sigma_3\sigma_4$, $p^{(4)} = \sigma_3$, $p^{(5)} = \sigma_1$, $p^{(6)} = \sigma_3\sigma_2\sigma_1\sigma_4$, $p^{(7)} = \sigma_2$, $p^{(8)} = \sigma_4$, $p^{(9)} = \sigma_2\sigma_1\sigma_3\sigma_4$, $p^{(10)} = \sigma_3$, $p^{(11)} = \sigma_1$, $p^{(12)} = \sigma_3\sigma_2\sigma_1\sigma_4$. Thus, $p^{(6)} = p^{(12)}$, and so $F(p) = \{p^{(6)} = \sigma_3\sigma_2\sigma_1\sigma_4\}$.

Notice that $S = \sigma_3\sigma_2\sigma_1 \preccurlyeq \sigma_3\sigma_2\sigma_1\sigma_4 = p^{(6)}$, and so we can conclude that $\kappa_S = \sigma_3\sigma_2\sigma_1\sigma_4$.

For the sake of interest, $\mathrm{SC}(Y) = \{Y, \mathfrak{s}(Y), \mathfrak{s}^2(Y), \mathfrak{s}^3(Y), \mathfrak{s}^4(Y), \mathfrak{s}^5(Y)\}$, which I found using the software available at [Thi22].

Appendix B

Proofs and corrections

B.1 Proofs

Lemma B.1.1. *If $X = \Delta^p X_1 \dots X_r$ is in left-canonical form as written, then $\tau(X) = \Delta^p \tau(X_1) \dots \tau(X_r)$ is in left-canonical form as written too.*

Proof. Firstly, since τ is a group homomorphism and Δ^p is invariant under τ for any integer p, we indeed have $\tau(X)$ as above. Since X is in left-canonical form, $S(X_{i+1}) \subset F(X_i)$ for each $i = 1, \dots, r-1$. Since τ is well-defined, we have that $S(\tau(X_{i+1})) \subset F(\tau(X_i))$ for each $i = 1, \dots, r-1$ as well. Thus, $\tau(X)$ is also in left-canonical form as written. □

Theorem B.1.2 (Derived from Theorem 1.13(d) in [Geb05])**.** *For all $X \in B_n$, $\tau(\boldsymbol{c}(X)) = \boldsymbol{c}(\tau(X))$ and $\tau(\boldsymbol{d}(X)) = \boldsymbol{d}(\tau(X))$.*

Proof. Let $X = \Delta^p X_1 \dots X_r$ in left-canonical form, where $p = \inf X$ and $X_1 \neq \Delta$. Then $\mathbf{c}(X) = \Delta^p X_2 \dots X_r \tau^p(X_1)$, $\mathbf{d}(X) = X_r \Delta^p X_1 \dots X_{r-1}$, and $\tau(X) = \Delta^p \tau(X_1) \dots \tau(X_r)$. By Lemma B.1.1, $\tau(X)$ is in left-canonical form as written.

Firstly,

$$\begin{aligned}
\mathbf{c}(\tau(X)) &= \Delta^p \tau(X_2) \dots \tau(X_r) \tau^p(\tau(X_1)) \\
&= \Delta^p \tau(X_2) \dots \tau(X_r) \tau(\tau^p(X_1)) \\
&= \tau\big(\Delta^p X_2 \dots X_r \tau^p(X_1)\big) \\
&= \tau(\mathbf{c}(X)),
\end{aligned}$$

as desired.

Secondly,

$$\begin{aligned}
\mathbf{d}(\tau(X)) &= \tau(X_r) \Delta^p \tau(X_1) \dots \tau(X_{r-1}) \\
&= \tau\big(X_r \Delta^p X_1 \dots X_{r-1}\big) \\
&= \tau(\mathbf{d}(X)),
\end{aligned}$$

as desired. □

B.2 Corrections

B.2.1 In minimal simple elements for SSS(x) paper

Let $x \in G$. Proposition 4.8 of [FGM03] is as follows:

Proposition. *For every $v \in M$ and every $m \in \mathbb{Z}$, the property $\mathcal{P}_v^{\geq m}$ is closed under gcd.*

Recall that a simple element s satisfies the property $\mathcal{P}_v^{\geq m}$ if $s^{-1} v s \in C^{\geq m}(x)$, where $C^{\geq m}(x) = \{y \in C(x) \mid \inf y \geq m\}$. Also recall that $v \in M$ means v is a positive element.

This proposition is correct, and the proof is valid. However, it is not powerful enough to be consistent with later statements in the paper. In particular, very shortly after this proposition, the authors remark that this proposition, along with Corollary 4.3.2, implies that the cardinality of $S_v^{\geq m}$ is at most the number of atoms in M, for every $v \in C^{\geq m}(x)$. If the property $\mathcal{P}_v^{\geq m}$ were only closed under gcd for $v \in M$, we would not be able to say that about $S_v^{\geq m}$ for *all* $v \in C^{\geq m}(x)$. Since m can be any integer, $C^{\geq m}(x)$ may very well include elements not in M. Furthermore, in the discussion of the algorithms in the paper, v is again taken to be in $C^{\geq m}(x)$ when we compute $S_v^{\geq m}$.

There are a few more inconsistencies in the paper, but all appear in remarks following correct results, so it does not affect our discussion of the paper.

The adjusted statement (as presented in our discussion of [FGM03] in Section 4.3) and proof of Proposition 4.8 follow below. First, a simple lemma for the sake of completeness:

Lemma B.2.1. *If $s_1, s_2 \in S$ and $s_1 \wedge s_2 = s$, then $ws_1 \wedge ws_2 = ws$ for any $w \in M$.*

Proof. $s = s_1 \wedge s_2$ implies $s \preccurlyeq s_1$ and $s \preccurlyeq s_2$, and is the greatest such. Thus, if any other element p satisfies $p \preccurlyeq s_1$ and $p \preccurlyeq s_2$, then $p \preccurlyeq s$.

Since $\preccurlyeq$ is invariant under left-multiplication, we have that $ws \preccurlyeq ws_1$ and $ws \preccurlyeq ws_2$ for any $w \in M$. Suppose there is some element q such that $q \preccurlyeq ws_1$ and $q \preccurlyeq ws_2$. Left-multiplying both expressions by w^{-1}, we get $w^{-1}q \preccurlyeq s_1$ and $w^{-1}q \preccurlyeq s_2$. Thus, $w^{-1}q \preccurlyeq s$, which is equivalent to $q \preccurlyeq ws$. Hence we have shown that ws is maximal (w.r.t. $\preccurlyeq$) among the prefixes of ws_1 and ws_2, and so $ws = ws_1 \wedge ws_2$. □

The above proof was completed with assistance from the examiners of this dissertation [exa22].

Recall from Proposition 4.3.3 that a simple element s satisfies $\mathcal{P}_v^{\geq m}$ if and only if $\tau^m(s) \preccurlyeq ws$ where $w \in M$ satisfies $v = \Delta^m w$.

Proposition. *For every $v \in C^{\geq m}(x)$ and every $m \in \mathbb{Z}$, the property $\mathcal{P}_v^{\geq m}$ is closed under gcd.*

Proof. Write v as $v = \Delta^m w$ where $w \in M$. Suppose s_1 and s_2 are simple elements which satisfy $\mathcal{P}_v^{\geq m}$. Thus we have that $\tau^m(s_1) \preccurlyeq ws_1$ and $\tau^m(s_2) \preccurlyeq ws_2$. Suppose $s_1 \wedge s_2 = s$. This means $s \preccurlyeq s_1$ and $s \preccurlyeq s_2$, and so $\tau^m(s) \preccurlyeq \tau^m(s_1)$ and $\tau^m(s) \preccurlyeq \tau^m(s_2)$, since $\preccurlyeq$ is invariant under τ. Putting this together, we get that $\tau^m(s) \preccurlyeq \tau^m(s_1) \preccurlyeq ws_1$ and $\tau^m(s) \preccurlyeq \tau^m(s_2) \preccurlyeq ws_2$.

Lemma B.2.1 tells us that $ws = ws_1 \wedge ws_2$. Since $\tau^m(s) \preccurlyeq ws_1$ and $\tau^m(s) \preccurlyeq ws_2$, we must have that $\tau^m(s) \preccurlyeq ws$, by the maximality of ws among the prefixes of ws_1 and ws_2. Hence, $s = s_1 \wedge s_2$ satisfies $\mathcal{P}_v^{\geq m}$, and so $\mathcal{P}_v^{\geq m}$ is closed under gcd. □

B.2.2 In USS(x) paper

In the original text of [Geb05], the iterative definition of the pullback $\pi_y(s)$ is given as $s_{(0)} = s$ and $s_{(i+1)} = \pi_{\mathbf{c}^\alpha(y)}(s_{(i)})$ for $i \geq 0$ where $0 \leq \alpha \equiv -i \mod N$. Recall that, with this definition, $s_{(i+1)}$ conjugates $\mathbf{c}^\alpha(y)$ to an element in SSS(x). My correction of this definition is motivated by the example we discussed in Section A.3. The $s_{(i)}$ we found match those given in the text, but when I tried to follow the example without any corrections to the definition, I ran into a lot of trouble. Firstly, I could not manage to get the same results for each $s_{(i)}$, and when I attempted to test the given $s_{(i)}$, none of them conjugated their associated elements into the super summit set. Thus, I conjugated each of the distinct transports with each of x, $\mathbf{c}(x)$, and $\mathbf{c}^2(x)$, to finally conclude

$$s_{(3)}^{-1} x s_{(3)} = s_{(6)}^{-1} x s_{(6)} \in \text{SSS}(x)$$
$$s_{(2)}^{-1} \mathbf{c}(x) s_{(2)} = s_{(5)}^{-1} \mathbf{c}(x) s_{(5)} \in \text{SSS}(x)$$
$$s_{(1)}^{-1} \mathbf{c}^2(x) s_{(1)} = s_{(4)}^{-1} \mathbf{c}^2(x) s_{(4)} \in \text{SSS}(x)$$

All other combinations of conjugation lead to elements not in SSS(x). Hence, I adjusted the definition of α to be $0 \leq \alpha \equiv -(i+1) \mod N$, which is consistent with the example.

The second correction to this text is with regards to Proposition 5.1.17, which originally is as follows:

Proposition B.2.2 (Proposition 4.9 in [Geb05], as given)**.** *Let $s \in S$ and consider the elements $s_{(iN)}$, for $i \geq 0$, obtained by applying the pullback definition for $y = \boldsymbol{c}^{N-1}(x)$. S is finite, so there are integers $i_2 > i_1 \geq 0$ such that $s_{(i_1 N)} = s_{(i_2 N)}$. Choose minimal values for i_1 and i_2, let $l = i_2 - i_1$, and choose an integer j such that $jl \geq i_1$. Let $p_x = p_x(s) = s_{(jlN)}$. Then $p_x \preccurlyeq \mathfrak{c}_s$ and there exists $v \in F_x(p_x)$ with $s \preccurlyeq v$. In particular, $v = \mathfrak{c}_s$.*

If we take this proposition at face value, it implies that $s_{(iN)} = \pi_{\mathbf{c}^\alpha(x)}(s_{(iN-1)})$ conjugates $\mathbf{c}^{N-1}(x)$ to an element in SSS(x). Based on the pullbacks given in the example, and based on the $s_{(iN)}$ chosen in the example (in accordance with the above proposition), we know $s_{(iN)}$ conjugates $x = \mathbf{c}^N(x)$ to an element in SSS(x).

With our corrected definition of α, we would have $\alpha \equiv -(iN) \mod N = 0$, which means $s_{(iN)} = \pi_{\mathbf{c}^0(x)}(s_{(iN-1)}) = \pi_x(s_{(iN-1)})$, and indeed this matches what we know about $s_{(iN)}$ from the example, but does not match the proposition.

If we revert to the incorrect definition of α, the proposition makes even less sense, since in this case $\alpha \equiv -(iN-1) \mod N \equiv 1 \mod N = 1$, and so $s_{(iN)} = \pi_{\mathbf{c}^1(x)}(s_{(iN-1)})$, which would mean $s_{(iN)}$ conjugates $\mathbf{c}(x)$ to an element in SSS(x), which we know is not true and also does not match the proposition.

Thus, the proposition should read "...obtained by applying the pullback definition for $y = \mathbf{c}^N(x)$...", as it is presented in our discussion of the ultra summit set in Section 5.1.

Bibliography

[Art25] Emil Artin. Theorie der Zöpfe. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 4(1):47–72, 1925.

[Art47a] Emil Artin. Braids and Permutations. *Annals of Mathematics*, 48(3):643–649, 1947.

[Art47b] Emil Artin. Theory of Braids. *Annals of Mathematics*, 48(1):101–126, 1947.

[BGGM07a] Joan Birman, Volker Gebhardt, and Juan González-Meneses. Conjugacy in Garside groups I: cyclings, powers and rigidity. *Groups, Geometry, and Dynamics*, pages 221–279, 2007.

[BGGM07b] Joan Birman, Volker Gebhardt, and Juan González-Meneses. Conjugacy in Garside groups III: Periodic braids. *Journal of Algebra*, 316(2):746–776, 2007. Number: 2.

[BGGM08] Joan Birman, Volker Gebhardt, and Juan González-Meneses. Conjugacy in Garside groups II: structure of the ultra summit set. *Groups, Geometry, and Dynamics*, pages 13–61, 2008.

[Bir75] Joan Birman. *Braids, Links, and Mapping Class Groups*, volume 82 of *Annals of Mathematics Studies*. Princeton University Press, Princeton, New Jersey, 1975.

[Bir82] Joan Birman. Errata: On the Conjugacy Problem in the Braid Group. *Canadian Journal of Mathematics*, 34(6):1396–1397, 1982. Number: 6 Publisher: Cambridge University Press.

[BKL01] Joan Birman, Ki Hyoung Ko, and Sang Jin Lee. The Infimum, Supremum, and Geodesic Length of a Braid Conjugacy Class. *Advances in Mathematics*, 164(1):41–56, 2001. Number: 1.

[Boh47] H. Frederic Bohnenblust. The Algebraical Braid Group. *Annals of Mathematics*, 48(1):127–136, 1947.

[Cho48] Wei-Liang Chow. On the Algebraical Braid Group. *Annals of Mathematics*, 49(3):654–658, 1948.

[Dal11] Ester Dalvit. *New proposals for the popularization of braid theory.* phd, University of Trento, 2011.

[DP99] Patrick Dehornoy and Luis Paris. Gaussian Groups and Garside Groups, Two Generalisations of Artin Groups. *Proceedings of the London Mathematical Society*, 79(3):569–604, November 1999. Number: 3.

[Eps92] David Epstein. *Word Processing in Groups*. Jones and Bartlett, Boston, Massachusetts, 1992.

[ER88] El-Sayed El-Rifai. *Positive braids and Lorenz links*. Ph.D., University of Liverpool, 1988. Accepted: 1988.

[ERM94] El-Sayed El-Rifai and Hugh Morton. Algorithms for positive braids. *The Quarterly Journal of Mathematics*, 45(4):479–497, 1994. Number: 4.

[exa22] Communicated by the examiners of this dissertation, 2022.

[FGM03] Nuno Franco and Juan González-Meneses. Conjugacy problem for braid groups and Garside groups. *Journal of Algebra*, 266(1):112–132, 2003. Number: 1.

[Gar65] Frank Garside. *The theory of knots and associated problems*. Ph.D., University of Oxford, 1965. Accepted: 1965.

[Gar69] Frank Garside. The braid group and other groups. *The Quarterly Journal of Mathematics*, 20(1):235–254, 1969. Number: 1.

[Geb05] Volker Gebhardt. A new approach to the conjugacy problem in Garside groups. *Journal of Algebra*, 292(1):282–302, October 2005. Number: 1.

[GGM10a] Volker Gebhardt and Juan González-Meneses. The cyclic sliding operation in Garside groups. *Mathematische Zeitschrift*, 265(1):85–114, May 2010.

[GGM10b] Volker Gebhardt and Juan González-Meneses. Solving the conjugacy problem in Garside groups by cyclic sliding. *Journal of Symbolic Computation*, 45(6):629–656, June 2010.

[GM03] Juan Gonzalez-Meneses. The nth root of a braid is unique up to conjugacy. *Algebraic & Geometric Topology*, 3(2):1103–1118, November 2003. Number: 2 Publisher: Mathematical Sciences Publishers.

[Lee00] Sang-Jin Lee. *Algorithmic solutions to decision problems in the braid groups*. PhD thesis, Korea Advanced Institute of Science and Technology, 2000.

[Pic01] Matthieu Picantin. The conjugacy problem in small Gaussian groups. *Communications in Algebra*, 29(3):1021–1039, 2001. Number: 3.

[Rol03] Dale Rolfsen. New developments in the theory of Artin's braid groups. *Topology and its Applications*, 127(1):77–90, January 2003.

[Thi08] Jean-Luc Thiffeault. Lectures on Braids and Dynamics, 2008. University of Wisconsin - Madison.

[Thi22] Jean-Luc Thiffeault. jeanluct/cbraid, January 2022. original-date: 2014-12-08T14:43:51Z.