Computational Problems in the Braid Group with Applications to Cryptography

Umut ISIK

March 18, 2005

Abstract

After making some basic definitions and results on links and braids, we focus on computational problems concerning the braid group such as the word and conjugacy problems and examine the recent use of the braid group and these problems in cryptography. We finally consider the NP-completeness of the NON-MINIMAL BRAIDS problem. We also briefly present some open problems as well as some basic notions of the theory of computation.

1 Introduction

Artin first introduced braids in 1926 and later published a more rigorous study of braids in 1947 [2]. Since then, braids and computational problems concerning braids have attracted much attention. In this paper, we shall present some of these problems and some of the solutions to those together with some applications.

The concatenation of braids provides us a group operation and leads us to the braid group B_n. B_n can be generated by a finite set of elements called the standard generators. These generators lead us to the word and conjugacy problems, concerning products of those generators, i.e. strings in terms of the generators, and their meaning as elements. These problems lead to applications in cryptography.

We begin with a brief summary of the basic notions of computability theory and complexity before focusing on braids.

2 Computability and Complexity

Since we shall be considering algorithmic problems concerning braids, we introduce several definitions for the sake of completeness.

In order to consider algorithms mathematically, one needs an abstract model of computing and a clear definition of a problem. We shall be considering *decision problems*, that is problems that require an answer yes or

no which can be seen as deciding whether an element belongs to a specific set or not. Several models such as the *lambda calculus, recursive functions, Markov algorithms, Post Systems* and *Turing machines* have been proposed for this purpose and they all have been shown to be of the same power. That is if one can do a computation with any one of these, then one can do the same computation with all of the others. One remark is that most of these models were introduced before computers were actually made. We shall be using Turing machines since it is the model which is the closest to today's computers.

A *Turing machine* is an abstract machine, accompanied with an infinite tape. The machine has a tape using which it reads what is on the tape and can write on the tape. The tape can also be used as memory. Initially, the tape contains the input (consisting of a finite set of symbols, usually `0`'s and `1`'s) and the rest of the tape is blank with the head positioned at the beginning of the input string. The machine has a finite set of *states*. When the machine is at some state, it reads the symbol under its head, and according to the input and its current state, it prints a symbol under the head and moves its head right or left to the next symbol. Two states are *accept* and *reject* states. They correspond to the termination of the computing, *accept* means a positive answer and *reject* means a negative answer. The reason for us to consider decision problems is that if we could make a Turing machine which solves a problem which is not a decision problem and gives the output on the tape, then we could make a Turing machine that tells us whether the nth symbol of the output is `0` or `1` and extract the required information from those. Here is the formal definition of a Turing machine:

Definition 1 *A Turing machine is a 7-tuple* $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$*, with* Q*,* Σ *and* Γ *are finite sets and:*

1. Q *is the set of states.*
2. Σ *is the input alphabet, which does not contain a special blank symbol* `B`
3. Γ *is the tape alphabet where* $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ *is the transition function.*
5. q_0 *is the initial state.*
6. q_{accept} *is the accept state.*
7. q_{reject} *is the reject state and* $q_{reject} \neq q_{accept}$*.*

A Turing machine M makes computation by taking finite string $\{w_n\}$ in the alphabet Σ as input, starting at the initial state, writing and going to the next state according to the state transition function. If the machine

reaches the accept or reject states, then it *halts*. Note that a machine can run forever, never reaching the accept or reject states.

We say that a Turing machine M *decides* a language L (a set of strings over some alphabet Σ) if, when given an input string s, M accepts if and only if $s \in L$; and M halts for every input. This definition coincides nicely with our notion of decision problems because we can encode inputs as strings and define a language which is the set of strings for which the answer is yes. For example, if our decision problem is to determine whether a given integer is prime, we can encode the binary representation of numbers as binary strings and say that L is the language the elements of which represent prime numbers in base 2. An important result in the theory of computation is that there do exist undecidable languages. For example, the language of strings corresponding to pairs of Turing machines and their inputs (one can encode a Turing machine as a binary string) that halt in a finite amount of time is not decidable. The *Church-Turing thesis* states that a problem can be solved with a Turing machine if and only if it can be solved by any *reasonable*[1] means of computation. So if we accept the Church-Turing thesis, it means that one cannot design a program (say, coded in C) which takes another C program and an input string as its input and which decides whether it will ever stop running or not.

There also is another notion of Turing machine called a *non-deterministic Turing machine*, which basically has a state transition function the range of which consist of a finite sets of triples instead of just triples. In other words, it is a Turing machine which makes copies of itself as necessary. A non-deterministic Turing machine accepts or rejects if and only if at least one of the copies accepts or rejects. This model of computing has the same computing power as the ordinary (deterministic) Turing machine in terms of the class of languages it decides.

We can now define the complexity of a Turing machine or a problem.

Definition 2 *The* time complexity *of a Turing machine M is a function $f : \boldsymbol{N} \to \boldsymbol{N}$, where $f(n)$ is the maximum number of steps (maximum number of referral to the state transition function) M takes to accept or reject, given any input of length n.*

This notion is often referred to as *worst case complexity*, for one can also consider *average complexity* which is the average running time of M on inputs of size n.

There is also a notion of *space complexity* but we shall not consider it here and refer to time complexity as complexity. We shall use the notion of complexity for algorithms, which will be the complexity of a Turing ma-

[1] this includes some newer, hopefully applicable abstract systems of computing, including quantum computation or probabilistic computation

chine that simulates that algorithm. Usually, one is mainly interested in the growth of the complexity function.

Definition 3 *Let $f : \boldsymbol{N} \to \boldsymbol{N}$ and $g : \boldsymbol{N} \to \boldsymbol{N}$. We say that $f = O(g)$ if $f(n) \leq cg(n)$ for some constant c and sufficiently large n. We say $f = \Omega(g)$ if $f(n) \geq cg(n)$ for some constant c and sufficiently large n. If $f = O(g)$ and $f = \Omega(g)$ then we say $f = \Theta(g)$*

For example, $5n^2 + 9n + 5 = O(n^2)$. If we find an algorithm that runs in $5n^2+9n+5 = O(n^2)$ time, we say it is an $O(n^2)$ algorithm. So $O(f)$ gives us a general idea about the growth of f. Since we are mainly interested in how much time we shall need as we run a program on larger and larger inputs. But we also have that $n^2 + 9n + 5 = O(n^3)$, but $n^2 + 9n + 5 \neq \Theta(n^3)$.

These notions will give us an idea of the growth of an algorithm's complexity. We shall usually say that an algorithm has complexity $O(f)$ where f is a function of n. Roughly, f will be an upper bound for sufficiently large inputs for the number of additions, subtractions, comparisons and use of memory bits in today's computers. For example, if we consider the algorithm of finding the largest number among n integers through scanning all of them and keeping in memory the largest found so far at all times, it would be an $O(n)$ algorithm. We now introduce complexity classes.

Definition 4 *For a function $f : \boldsymbol{N} \to \boldsymbol{N}$,* TIME($f$) *is the class of languages (problems) that can be decided on a Turing machine with complexity $O(f)$.*

Definition 5 P *is defined to be the class of languages (problems) which can be decided by a Turing machine of polynomial time complexity. That is:*

$$\mathrm{P} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(n^k)$$

For example, the problem of finding the maximum of a finite set of integers is in P with the algorithm we just described. But we do not know whether the clique problem, that is the problem of deciding whether a graph with n vertices has a k-clique (a fully connected subgraph with k vertices) is in P (no such algorithm is known).

Definition 6 *A* verifier *for a language L is a Turing machine (algorithm) V where*

$$L = \{w \, is \, a \, string \mid V \, accepts \, (w, c) \, for \, some \, string \, c\}$$

The measure of the time of a verifier is in terms of w. A polynomial time verifier runs in polynomial time in the length of w. A language is polynomially verifiable if it has a polynomial time verifier.

For example, the clique problem has a polynomial time verifier. The verifier would be the algorithm that takes a graph G and a suggested k-clique and that checks if the suggested k-clique is really a k-clique or not.

Definition 7 *The class* NP *is the complexity class of polynomially verifiable languages.*

The name NP stands for nondeterministic polynomial time and refers in complexity theory to the standard theorem that every polynomially verifiable language can be solved in polynomial time by a non-deterministic Turing machine. One of the greatest open problems in mathematics is to determine whether the classes P and NP are equal. For this matter, we finally introduce the notion of NP-completeness.

Definition 8 *The class of* NP-complete *languages is the class of languages N that are in* NP *such that for every other language L in* NP*, there is a polynomial-time algorithm, that takes a string s and outputs a string $f(s)$ such that $s \in L$ if and only if $f(s) \in N$. That is, every problem in* NP *is polynomial time reducible or polynomially reducible to L.*

Since the addition of two polynomials is a polynomial, solving one NP problem in P would mean that P = NP. The first problem shown to be NP-complete is the satisfiability problem, which is the decision problem of finding whether a given boolean logical expression in terms of some variables belongs to the class of boolean logical expressions that can be made to be true with some value assignment of the variables.

A language is said to be NP-hard if every problem in NP is polynomial time reducible to it. So an NP-complete language is an NP-hard language that is polynomial time verifiable.

Some problems can be very easy to verify but this does not imply that the negation of those problems are easy to verify. By negation, we mean the complement of the corresponding language in the set of strings over the given alphabet. For example, in the case of the CLIQUE problem, the negation NON-CLIQUE would be the problem of answering whether the graph does not have a k-clique, that is the decision problem corresponding to the language of pairs (G, k) such that G does *not* have a k-clique. In general, solving the negation of a problem means solving the problem itself. However, there are differences in verifiability, for example, observe that it is harder to verify that a graph does *not* have a k-clique rather than it *does* have a k-clique.

Definition 9 *A language is said to be* co-NP *(respectively* co-NP-complete*) if it is the complement of a language in* NP *(respectively* NP-complete*).*

It is not known whether the negation of any problem in NP, that is the complement of the language corresponding to that problem is in NP. So, It is not known whether NP=co-NP. The MINIMAL BRAIDS problem we shall consider below is not known to be in NP, but it is NP-hard. But its complement is NP-complete.

For a very intuitive introduction to the theory of computation and more detail see [11].

3 Braids

We begin with the basic definitions concerning braids. There are several ways to define a braid but the effective way is, especially for a computational approach, the following definition.

Definition 10 *A Braid is the union of images of a set of simple piecewise linear curves $\phi_1, \phi_2, ..., \phi_n$ in $\boldsymbol{R}^3$ such that*

- *each ϕ_i has domain $[0,1]$ and satisfies $\phi_i(0) = (i,0,0)$; and $\phi_i(1) = (\sigma(i),0,1)$ where σ is some permutation of $1,2,...,n$.*
- *the z-coordinate of $\phi_i(t)$ increases as t increases in $[0,1]$*
- *the ϕ_i do not intersect each other*

In this definition, we take piecewise linear curves in order not to have braids that have strings that curve in a very complicated manner. The equivalence relation on braids is defined similar to the equivalence relation of knots. In the smooth setting, two braids are equivalent if and only if one can be continuously deformed to the other without making the curves intersect and keeping the points $(i,0,0)$ and $(i,0,1)$ for $i = 1,2,...,n$ fixed. In the piecewise linear setting, one can define this equivalence as the existence of a finite sequence of some basic moves, as for polygonal knots. Throughout this report, we shall refer to the whole equivalence class of a braid when mentioning a braid. We shall say that a braid having n piecewise linear curves is a braid with n strings or an n-braid.

Braids can be represented by diagrams just as links can as the following theorem and its corollary show.

Theorem 11 *Any polygonal link (and hence any link) can be represented by a link diagram.*

To convince ourselves, we can consider a polygonal link L in $\mathbf{R}^3$ consisting of line segments $l_1, l_2, ..., l_m$. Without loss of generality, assume that no two of the line segments are parallel. If we pick a point x far from the link, and project the link to a plane in front of it, we will have a diagram if and

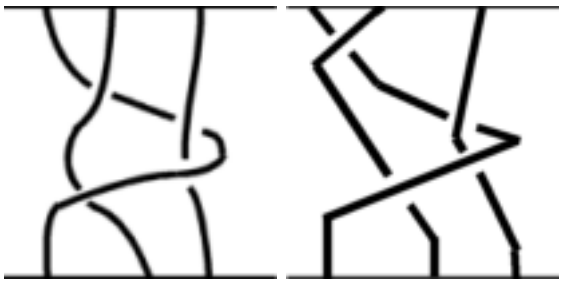

Figure 1: A smooth braid and a polygonal braid

only if there is no line passing through x and going through three of the line segments. Since there are only finitely many line segments, we may assume that x is not on the extension to a line of any one of them. Observe that for any y in 3-space, and two line segments which are not parallel, there cannot be more than two lines passing through y and both of the line segments. Let $L(l_i, l_j, y)$ be the union of those lines. Observe that if we unite $K(i,j,k) = \bigcup_{y \in l_k} L(l_i, l_j, y)$ we get a subset of a two dimensional surface in 3-space. So, $\bigcup_{i,j,k} K(i,j,k)$ will not be covering the 3-space, which means that we can find an x such that no line will pass through x and three of the line segments.

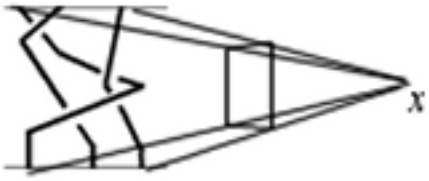

Figure 2: The projection described above

Assume we have an n-braid, by the same argument, there must be such a point x in the box $[0,n] \times [0,+\infty) \times [0,1]$, modifying the beginning and endpoints slightly, we get a diagram for the braid. Hence

Corollary 12 *Any braid has a diagram.*

4 Braiding of a Link

The closure of a braid is the link formed by joining the points $(i,0,0)$ with $(i,0,1)$ for each i as in this figure:

The closure operation gives us an important connection between braids and links. The following theorem of Alexander shows that this connection

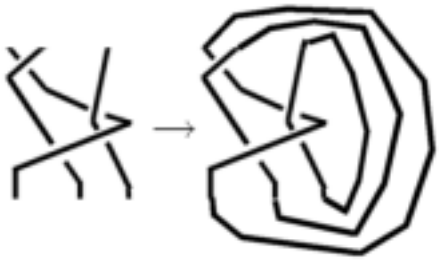

Figure 3: The closure of a braid

is quite strong.

Theorem 13 *(Alexander) Every link is ambient isotopic to the closure of a braid.*

In order to prove the theorem, we start with the following definition.

Definition 14 *An oriented polygonal link diagram is said to wind around a point $x \in \boldsymbol{R}^2$ if every line segment in the link diagram has a positive orientation with respect to the point x (i.e. counterclockwise orientation where the center is taken to be x).*

Figure 4: An example of a link diagram that winds around a point x and how one can see such a link as the closure of a braid

Observe that if a given link winds around any point, then we can form a braid which has the link as the closure. So it suffices to prove the following theorem in order to prove Theorem 13

Theorem 15 *Any link has a diagram which winds around a point.*

Proof: We shall describe an algorithm that terminates and gives an equivalent link that winds. Assume we are given the diagram of a polygonal link:

- 1 pick any point x not on any of the the line segments (or on their extension to lines) in the diagram and give the link an orientation.
- 2 do the following as long as the last obtained link does not wind around x
  - (i) find a line segment l of the diagram that is not positively oriented with respect to x
  - (ii) if l has no crossings, then replace l with two line segments such that they both go over or both go under all the other line segments they intersect, and l and the two newly added line segments form a triangle that contains x. If the line segment contains a single crossing on which l goes over, do the same but with the two new line segments going over all the other line segments they intersect, and similarly with undercrossings if l goes under. If l contains more than one crossing, break l apart into line segments such that each part contains only one crossing and do the above for each of them.

Observe that at each time "(ii)" is used, the number of edges that is not positively oriented with respect to x is reduced by one. So this algorithm terminates, and since it terminates only when there are no non-positively oriented edges, the last link obtained must be winding around x. Thus, every link can be turned into the closure of a braid. Thus, any linked is ambient isotopic to the closure of some braid. □

Observe that this algorithm runs in polynomial time. If we took the link as a set of line n segments with indications of over and under crossings between each pair of line segments, it would take $O(n)$ time to find a negatively oriented line segment, $O(n)$ time to find the segments it intersects, $O(n^2)$ time to break it into a set of segments if necessary and O(n) time to create the two new line segments in the algorithm if necessary. Also since two line segments can have at most one intersection, (so we may not reach large numbers of line segments when breaking some line segments) For a more formal addressing of matters of complexity, see Section 2.

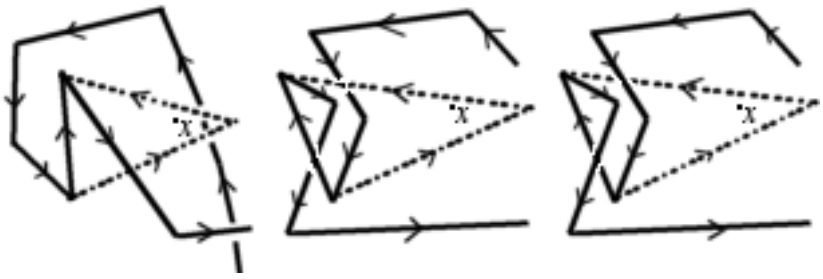

Figure 5: The above algorithm

It is clear that if we define a map $\kappa : Braids \to Links$ taking a braid to its closure; then this map preserves equivalence. But κ is not injective, as we can see in the figure below.

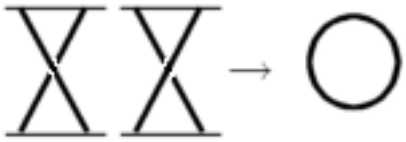

Figure 6: Two non-equivalent braids the closures of which is the unknot

5 The Braid Group B_n

Let B_n be the set of equivalence classes of n-braids. Define a multiplication operation $B_n \times B_n \to B_n$ as the operation taking two braids a and b to the braid formed by connecting the ith ending point of a to the ith starting point of b for $i = 1, 2, ..., n$. It is clear that this operation is well-defined and that it is associative. It has as identity element the braid formed by linear curves. Also, every braid has an inverse with respect to this operation because if we just take the mirror image of the braid with respect to the xy-plane and translate, we get the inverse. So B_n is a group.

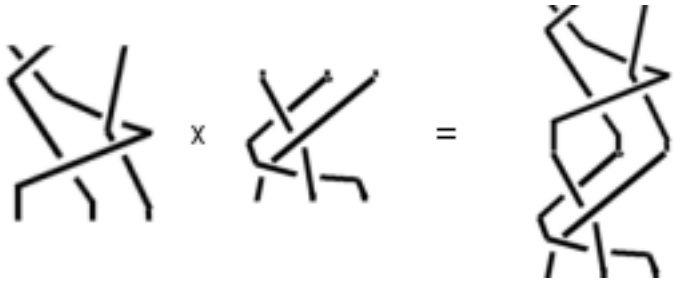

Figure 7: The group operation of B_n

Let σ_i, for $1 \leq i < n$ be the element of B_n which interchanges the ith string with the $(i+1)$th, with the ith string overcrossing and keeps the others fixed.

Theorem 16 *B_n is generated by the elements σ_i.*

The result is intuitively clear but we should say that any n-braid can be represented with a diagram which is vertically divided into a finite number of parts at each of which only two strings are exchanged and the others are fixed.

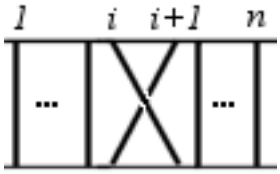

Figure 8: One of the standard generators σ_i of B_n

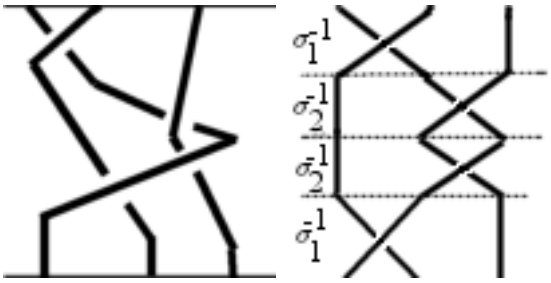

Figure 9: A polygonal knot which is equal to $\sigma_1^{-1}\sigma_2^{-1}\sigma_2^{-1}\sigma_1^{-1}$

Remark One can make a polynomial time algorithm that transforms a given diagram to a multiplication of σ_i's. If we are given a braid diagram, we can just order the crossings from top to bottom, moving some of the crossing slightly upwards as necessary. In general we shall think that our input string consists of a string of the generators σ_i when we would like to devise algorithms about braids.

Observe that braids satisfy:

(i) $\sigma_i\sigma_j = \sigma_j\sigma_i$ for i and j such that $|j-i| > 1$

(ii) $\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}$for $0 \leq i < n-1$

(i) is usually referred to as the far commutativity relation and (ii) is usually referred to as the braid relation. Actually, these relations are the defining relations of B_n, that is, if two braids are the same, they can be transformed to each other using these relations. More precisely, we have the following.

Theorem 17 *(Artin's theorem)* B_n *is isomorphic to the group*

$$\langle \sigma_1, \sigma_2, ..., \sigma_{n-1} \,|\, \sigma_i\sigma_j = \sigma_j\sigma_i \; for \; all \; i \; and \; j \tag{1}$$

$$such \; that \; |j-i| > 1; \; and \tag{2}$$

$$\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} \; for \; 1 \leq i < n-1\rangle \tag{3}$$

From now on, we shall see B_n simply as the group in the last theorem. Let S_n be the symmetric group on n letters. Now consider a map $\pi : B_n \to S_n$ defined as follows. For $b \in B_n$, the corresponding element $\pi(b)$ is given by

$$\pi(b)(i) = b_{(i)} \ for \ i = 1, ..., n$$

where $b_{(i)}$ is the place where ith string ends in the braid b. Observe that $\pi(\sigma_i \sigma j) = \pi(\sigma_i)\pi(\sigma j)$, so, since B_n is generated by the σ_i, π is a group homomorphism.

Definition 18 *The group of pure braids P_n is the kernel of the map π defined above.*

P_i is henceforth the subgroup of B_n consisting of the elements that have each string end at the same place they started. By the first isomorphism theorem, we have that $P_n \lhd B_n$ and $B_n/P_n \cong S_n$.

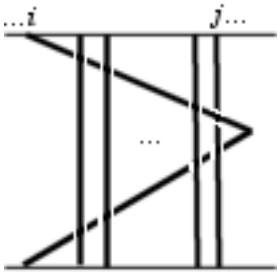

Figure 10: One of the generators $b_{i,j}$ of P_n

An induction on the number of strings of the braid shows

Proposition 19 *The pure braid group is finitely generated by the elements*

$$b_{i,j} = \sigma_i^{-1}\sigma_{i+1}^{-1}...\sigma_{j-1}^{-1}\sigma_j\sigma_j\sigma_{j-1}^{-1}...\sigma_i^{-1} \tag{4}$$

for $0 \leq i < j < n$.

6 The word and conjugacy problems in B_n

Assume we are given a group that is generated by finitely many elements which behave according to a finite set of rules, just as in (1). The word problem is an algorithmic problem to determine whether two strings (words) of elements of a fixed generating subset of the group (which represent their multiplication) represent the same element or not. In general, the problem

was proven to be unsolvable for arbitrary groups. That is there does not exist an algorithm which would take any finite set of generators and relations between them and words and give the answer to the question whether the two given strings represent the same element of the group or not. However, the word problem sometimes has a solution for some fixed groups. And the braid group is one of the groups for which the word problem is solvable. There actually is an efficient $O(n^2)$ algorithm to solve it.

We focus on the braid group versions of the word and conjugacy problems. Here are the formal definitions.

Definition 20 *(WORD)*
Instance: *Two words w_1 and w_2, given in terms of the standard generators of B_n*
Question: *Do w_1 and w_2 represent the same element in B_n?*

Another important problem that we shall discuss is the conjugacy problem.

Definition 21 *(CONJUGACY)*
Instance: *Two words w_1 and w_2, given in terms of the standard generators of B_n such that the elements represented by w_1 and w_2 are known to be in the same conjugacy class.*
Question: *What is $c \in B_n$ such that $cw_1c^{-1} = w_2$?*

Observe that the conjugacy problem is not stated as a decision problem, but one can create a related decision problems asking if the length of a word representing c is of length less than or equal to some k and asking if the ith generator in that word is σ_j. Using these, the same information could still be extracted in polynomial time if we could find a polynomial time algorithm that would be able to compute c as a word and print it.

The solvability of the word problem in B_n was originally proven by Artin through his "combing" algorithm. A polynomial time solution to this problem was first given by Thurston [5].

Observe that the word problem is actually the braid analogue of the KNOT EQUIVALENCE PROBLEM which seems to be a lot harder [7].

Dehornoy's algorithm for the word problem is an easy to implement and efficient algorithm, we start describing it with the following definition.

Definition 22 *A word ω is said to be* reduced *if for any i such that ω is of the form:*

$$\omega_1\sigma_i\omega_0\sigma_i^{-1}\omega_2 \tag{5}$$

or of the form

$$\omega_1\sigma_i^{-1}\omega_0\sigma_i\omega_2 \tag{6}$$

where ω_1, ω_0, ω_2 are substrings, there exists some $j < i$ such that σ_j or σ_j^{-1} is in w_0.

For example, the string $\sigma_4\sigma_3\sigma_5^{-1}\sigma_4^{-1}$ is reduced because there is a σ_3 between the σ_4 and σ_4^{-1}, but $\sigma_3\sigma_4\sigma_5\sigma_3^{-1}$ is not reduced since there is no σ_j with $j < i$ between the σ_3 and σ_3^{-1}.

Definition 23 *Let a word ω be as in (5), with σ_{j-1} as the element with the highest index to appear in w_0. Then the Dehornoy reduction function R takes ω to*

$$\omega_0\sigma_{i+1}^{-1}\sigma_{i+2}^{-1}...\sigma_{j-1}^{-1}f(w_0)\sigma_{j-1}...\sigma_{i+1}w_2$$

where f changes the σ_k to σ_{k+1} and σ_k^{-1} to σ_{k+1}^{-1} for $i < k < j$; and similarly for words as in (6).

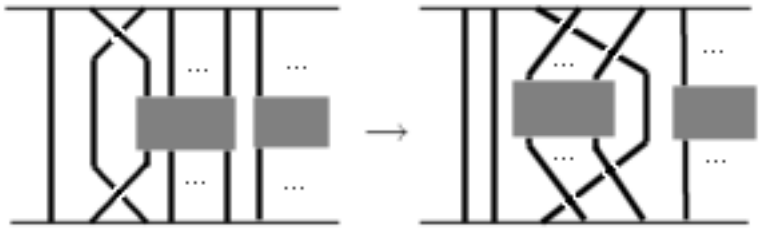

Figure 11: Dehornoy's reduction function $w \mapsto R(w)$

Given two strings in terms of the standard generators of B_n, Dehornoy's algorithm is to apply the R function to both strings until they become reduced. At that point, if the two strings are the same, then the words represents the same braid. In other words, the reduced word of a braid is a complete invariant. We shall not prove the following theorem because it is long and has quite a few technicalities:

Theorem 24 *Dehornoy's algorithm terminates and the reduced word of a braid is a complete invariant of braids.*

Observe that this algorithm takes a braid to an equivalent braid. This algorithm is conjectured to run in quadratic time. Experiments show that it is likely that it does so. But there is no current bound on the number of applications of the R function required to make a word reduced. However, there is a polynomial-time algorithm for the word problem.

There are interesting problems in combinatorial group theory about the braid group. A braid word w is said to be σ-positive (or respectively σ-negative) if the σ_i with minimal index appearing in w appears only as σ_i (respectively as σ_i^{-1}). An open problem of Dehornoy is to prove or disprove, for every fixed n, the existence of a constant c_n such that every $x \in B_n$ is equivalent to a σ-positive or $\sigma - negative$ word of length at most c_n.

For example, let us reduce the word $\sigma_1\sigma_2\sigma_3\sigma_1^{-1}\sigma_3^{-1}\sigma_2^{-1}$ using this algorithm (the substrings in parentheses represent the substrings $\sigma_i w_0 \sigma_i^{-1}$ as in (5) or (6)).

$$\begin{aligned} w &= (\sigma_1\sigma_2\sigma_3\sigma_1^{-1})\sigma_3^{-1}\sigma_2^{-1} \\ R(w) &= \sigma_2^{-1}\sigma_3^{-1}\sigma_1\sigma_3(\sigma_2\sigma_3^{-1}\sigma_2^{-1}) \\ R^2(w) &= \sigma_2^{-1}\sigma_3^{-1}\sigma_1\sigma_3\sigma_3^{-1}\sigma_2^{-1}\sigma_3 \end{aligned}$$

On the other hand, the conjugacy problem for braid groups has been shown by Garside [6] to be solvable. But no polynomial time solution is known.

7 Braid groups and cryptography

Cryptography is the study of methods for exchanging information such that it will not be understood by third parties. A *cipher* is an algorithm for encryption and decryption. The way a cipher works is usually controlled by a *key*.

Symmetric key ciphers use the the same key for encryption and decryption. But it is unsafe to exchange a key through an insecure channel. *Public key ciphers* use different keys for encryption and decryption. In public key cryptography, a person, say Alice, has a public and a private key. Alice publishes the public key or sends it to another person, say Bob, she wants to receive a message from. Then Bob encrypts his message using an encryption algorithm that has the property that it is very hard to decipher the encrypted message without knowing Alice's private key. So it is very hard for a third party to decipher the encrypted message because only Alice knows her private key. An important point in this method would be that it should be vary hard to understand what the private key is knowing the public key. The most popular method of this kind is the RSA[2] which relies on the difficulty of factoring integers. Public key cryptography, and especially the RSA algorithm is used in many applications such as e-commerce and military/diplomatic correspondence.

Usually, public key cryptography is used to create a shared secret key that will be used with a symmetric cryptosystem because known public key cryptosystems are a lot slower than symmetric cryptosystems (such systems are called *hybrid* cryptosystems). For this purpose, systems that provide just a shared secret key are also referred to as public key cryptosystems.

Here, we describe methods of public key cryptography that use the difficulty of the conjugacy problem for braid groups. Note that it is not known whether the *difficult* problems above are really difficult or not but they are

[2]named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman

thought to be so. The following protocol was proposed by Anshel, Anshel and Goldfield [1].

Let Alice and Bob be the two parties that would like to exchange information. Let them agree on some fixed integer n. Alice and Bob each pick a subgroup of B_n, Γ_A and Γ_B, say

$$\begin{aligned} \Gamma_A = & < s_1, s_2, ..., s_{r_1} > \\ \Gamma_B = & < t_1, t_2, ..., t_{r_2} > \end{aligned} \tag{7}$$

which will be their public keys. Now, Alice picks an element $a = s_{i_1} s_{i_2} ... s_{i_j} \in \Gamma_A$ and Bob picks and element $b = t_{i_1} t_{i_2} ... t_{i_k} \in \Gamma_B$, (these elements are among the generators). They send each other the sets of pairs:

$$\{(t_1, at_1a^{-1})...(t_{r_2}, at_{r_2}a^{-1})\} \tag{8}$$

$$\{(s_1, bs_1b^{-1})...(s_{r_1}, bs_{r_1}b^{-1})\} \tag{9}$$

Using this information, Alice can compute

$$(bs_{i_1}b^{-1})(bs_{i_2}b^{-1})...(bs_{i_j}b^{-1}) = bab^{-1} \tag{10}$$

and Bob can compute:

$$(at_{i_1}a^{-1})(at_{i_2}a^{-1})...(at_{i_k}a^{-1}) = aba^{-1} \tag{11}$$

Using these, Alice and Bob can both compute the commutator $bab^{-1}a^{-1}$ which is their shared secret. Using this shared secret, they can produce a shared key and communicate safely.

The second protocol we describe was proposed by Ko et al. [8] We first need the following definition.

Definition 25 *In B_{2n}, the subgroups*

$$LB_{2n} = < \sigma_1, ..., \sigma_{n-1} > \tag{12}$$

$$UB_{2n} = < \sigma_{n+1}, ..., \sigma_{2-1} > \tag{13}$$

are called the lower and upper braid groups.

Observe that elements of these two subgroups commute with each other and that $UB_{2n} \cong LB_{2n} \cong B_n$.

Alice and Bob agree on some fixed integer n and some $x \in B_{2n}$, these are the public keys. Then Alice chooses an element $a \in LB_{2n}$ and sends $y_a = axa^{-1}$ to Bob; and Bob chooses an element $b \in B_{2n}$ and sends $y_b = bxb^{-1}$ to Alice. Using what they received, Alice and Bob can compute

$$ay_ba^{-1} = abxb^{-1}a^{-1} = baxa^{-1}b^{-1} = by_ab^{-1} \tag{14}$$

which will be their shared secret key. Observe that a solution to the word problem must be used in both of the above methods because the elements Alice and Bob find in the end are equal as braids, but they may not be equal as words, so Alice and Bob must use a reduction algorithm (for example, Dehornoy's algortihm) in order to make sure they generate the same secret key.

These methods can be generalized for many finitely generated groups for which there is an efficient solution to the word problem but the conjugacy problem is hard to solve. But after the publication of these protocols, it has become apparent that the conjugacy problem is not as hard as anticipated for the braid group and that several successful methods to attack such cryptosystems can be made. For a survey of these attacks see [9]. But research on algebraic methods for public key cryptography is still active and it is not seen to be impossible that a better protocol can be discovered.

8 MINIMAL BRAIDS

The problem MINIMAL BRAIDS is the problem of deciding whether, for a given word in the braid group, there exists a shorter braid representing the same braid or not. But we are more interested in the negation of this problem, which is:

Definition 26 *(NON-MINIMAL BRAIDS)*
Instance: *A word in terms of the standard generators of* B_n
Question: *Is there a shorter word representing the same braid?*

Lemma 27 NON-MINIMAL BRAIDS *is in* NP

Proof: We know that the word problem is in P. A verifier for this problem would take two words, and check that the second word is indeed shorter than the first and that these words represent the same braid using the polynomial-time algorithm for the word problem.

In fact we have the following.

Theorem 28 *NON-MINIMAL BRAIDS is NP-complete.*

Which means that MINIMAL BRAIDS is co-NP. The proof of this fact is due to Paterson and Razborov [10], who polynomially reduce the problem a problem about strings over an alphabet $\Sigma = 1, 2, ..., r$ and permutations, which is NP-complete; to MINIMAL BRAIDS by associating each string with a particular braid. A complete proof is unnecessary for the purposes of this paper.

9 Conclusion

The study of algorithmic problems on braids is far from complete. It is not known if there can or cannot be a polynomial-time solution to the conjugacy problem. Or even if the conjugacy problem is in NP. There is no known upper bound on the number of reductions one has to apply to reduce a given word to its reduced form using Dehornoy's algorithm. We have achieved even less satisfying results with computational problems concerning knots and links. Through Alexander's theorem and Markov's theorem, some computational results with braids can lead to solutions to harder problems concerning knots and links. Also, with recent applications of knot and braid theory to different fields, the study of braids (and especially the study the braid group for computational purposes) has attracted a lot of interest.

References

[1] Iris Anshel, Michael Anshel, and Dorian Goldfeld. An algebraic method for public-key cryptography. *Math. Res. Lett.*, 6(3-4):287–291, 1999.

[2] E. Artin. Theory of braids. *Ann. of Math. (2)*, 48:101–126, 1947.

[3] Patrick Dehornoy. A fast method for comparing braids. *Adv. Math.*, 125(2):200–235, 1997.

[4] Patrick Dehornoy. Braid-based cryptography. In *Group theory, statistics, and cryptography*, volume 360 of *Contemp. Math.*, pages 5–33. Amer. Math. Soc., Providence, RI, 2004.

[5] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, Boston, MA, 1992.

[6] F. A. Garside. The braid group and other groups. *Quart. J. Math. Oxford Ser. (2)*, 20:235–254, 1969.

[7] Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *J. ACM*, 46(2):185–211, 1999.

[8] Ki Hyoung Ko, Sang Jin Lee, Jung Hee Cheon, Jae Woo Han, Ju-sung Kang, and Choonsik Park. New public-key cryptosystem using braid groups. In *Advances in cryptology—CRYPTO 2000 (Santa Barbara, CA)*, volume 1880 of *Lecture Notes in Comput. Sci.*, pages 166–183. Springer, Berlin, 2000.

[9] Karl Mahlburg. An overview of braid group cryptography. *www.math.wisc.edu/˜boston/mahlburg.pdf*, 2004.

[10] M. S. Paterson and A. A. Razborov. The set of minimal braids is co-NP-complete. *J. Algorithms*, 12(3):393–408, 1991.

[11] Michael Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, 1996.

[12] A.B. Sossinsky V.V. Prasolov. *Knots, Links, Braids and 3-Manifolds, Translations of Mathematical Monographs Vol. 154.* Amer. Math. Soc. Providence, RI, 1997.

[13] D. J. A. Welsh. Knots and braids: some algorithmic questions. In *Graph structure theory (Seattle, WA, 1991)*, volume 147 of *Contemp. Math.*, pages 109–123. Amer. Math. Soc., Providence, RI, 1993.