J. Symbolic Computation (2002) **34**, 91–103
doi:10.1006/jsco.2002.0521
Available online at http://www.idealibrary.com on IDEAL®

Identifying Half-Twists Using Randomized Algorithm Methods

S. KAPLAN[†] AND M. TEICHER[‡]

Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, 52900 Israel

Since the braid group was discovered by Artin (1947), the question of its conjugacy problem has been solved by Garside (1969) and Birman *et al.* (1998). However, the solutions given thus far are difficult to compute with a computer, since the number of operations needed is extremely large. Meanwhile, random algorithms used to solve difficult problems such as the primality of a number were developed, and the random practical methods have become an important tool. We give a random algorithm, along with a conjecture of how to improve its convergence speed, in order to identify elements in the braid group, which are conjugated to its generators (say σ_1^k) for a given power k. These elements of the braid group, the half-twists, are important in themselves, as they are the key players in some geometrical and algebraical methods, the building blocks of quasipositive braids and they construct endless sets of generators for the group.

1. Introduction

Let B_n be the braid group on n strings. The conjugacy problem in B_n is difficult and was addressed in several cases in the past. The computation of a solution is still not accessible. Although a solution was proposed (Garside, 1969; Elrifai and Morton, 1994) the running time of a computerized program based on these algorithm is extremely long.

In what follows we will describe an algorithm that solves a partial problem of the conjugacy problem. Our algorithm will make it possible to identify whether for a given braid w there exists an integer k such that w is conjugated to σ_1^k. Therefore, we actually identify some special conjugacy classes of the braid group.

Our algorithm is based on a random technique, and has the property that in any case it returns *true* (meaning that the input element w of the braid group is conjugated to a generator of the group in some power), it will also return to which generator σ_i the element is conjugated, and what is q such that $q^{-1}wq = \sigma_i^k$. Although we do not have any estimations for the probability of an erroneous return value, we performed a large number of experiments that gave us information as to how well the algorithm converges.

Elements in the conjugacy class of one of Artin generators are called half-twists. If one uses braid monodromy in order to classify geometrical hypersurfaces up to deformation (Moishezon and Teicher, 1988; Kulikov and Teicher, 2000), this will result in half-twists. Therefore, by using this algorithm it is possible to verify braid monodromy computations. Moreover, identifying the elements of this conjugacy class has implications in the research of quasipositive braids.

[†] E-mail: kaplansh@macs.biu.ac.il
[‡] E-mail: teicher@macs.biu.ac.il

0747–7171/02/020091 + 13 $35.00/0

We start, in Section 2, by giving some braid group preliminaries. In Section 3 we give a complete description of the random method for identifying half-twists along with full proofs of its correctness and complexity. Section 4 is dedicated to the presentation of the experiments and benchmarks that were performed in order to understand the capabilities of the random algorithm. Finally, Section 5 is dedicated to closing arguments.

2. Braid Group Preliminaries

2.1. E. Artin's definition of the braid group

Definition 2.1. *Artin's braid group B_n* is the group generated by $\{\sigma_1, \dots, \sigma_{n-1}\}$ subject to the relations

1. $\sigma_i \sigma_j = \sigma_j \sigma_i$ where $|i-j| \geq 2$
2. $\sigma_i \sigma_{i+1} \sigma_i = \sigma_{i+1} \sigma_i \sigma_{i+1}$ for all $i = 1, \dots, n-2$.

This algebraic definition can be seen from a geometrical point of view, by associating with every generator of the braid group σ_i, a tie between n strings going monotonically from top to bottom, such that we switch by a positive rotation between the two adjacent pair of strings i and $i+1$. This means that σ_i corresponds to the geometrical element described in Figure 1.

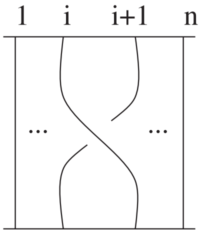

Figure 1. The geometrical braid associated with σ_i.

The operation for the geometrical group is the concatenation of two geometrical sets of strings.

Example 2.2. The geometrical braid that corresponds to $\sigma_1 \sigma_2^{-1} \sigma_1 \sigma_3$ is presented in Figure 2.

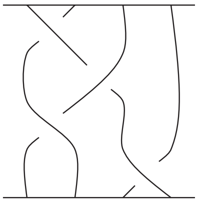

Figure 2. The geometrical braid $\sigma_1 \sigma_2^{-1} \sigma_1 \sigma_3$.

2.2. half-twists in the braid group

We are now going to describe what half-twists are, and give their geometrical interpretation.

Definition 2.3. Let H be the conjugacy class of σ_1, (i.e. $H = \{q^{-1}\sigma_1 q : q \in B_n\}$). We call H *the set of half-twists in* B_n, and we call an element $\beta \in H$ a *half-twist.*

Recall that half-twists have a geometrical interpretation. One can look at the braid group as the mapping class group of an n-punctured disk. The half-twists then correspond to geometric half-twists around an embedded arc that connects two punctures, and does not intersect itself or any other puncture. Using this way to look at the half-twists it is easy to see that they occupy a full conjugacy class of the braid group, and that they all conjugate to Artin's generators of the group.

Our main goal now is to describe the algorithm, but before we can do that we need some definitions.

Definition 2.4. Let $w \in B_n$ be a braid. Then it is clear that $w = \sigma_{i_1}^{e_1} \cdot \ldots \cdot \sigma_{i_l}^{e_l}$ for some sequence of generators, where $i_1, \ldots, i_l \in \{1, \ldots, n-1\}$ and $e_1, \ldots, e_l \in \{1, -1\}$. We will call such a presentation of w a *braid word*, and $\sigma_{i_k}^{e_k}$ will be called the kth *letter of the word* w. We will call l the *length* of the braid word.

We will distinguish between two relations on braid words.

Definition 2.5. Let w_1 and w_2 be two braid words. We will say that $w_1 = w_2$ if they represent the same element of the braid group.

Definition 2.6. Let w_1 and w_2 be two braid words. We will say that $w_1 \equiv w_2$ if w_1 and w_2 are identical letter by letter.

Definition 2.7. A *positive braid* is an element of B_n which can be written as a word in positive powers of the generators $\{\sigma_i\}$, without the use of the inverse elements σ_i^{-1}. We denote this subsemigroup B_n^+.

Next, we are going to recall some of the algorithms that were developed by Jacquemard (1990). These algorithms perform manipulation over braid words, and we will use a combination of them in our random method for determining if an element of the braid group is a half-twist or not.

The *LetterExtractLeft* algorithm enables us to determine if it is possible to write a certain positive braid word in such a way that a given generator is its first letter.

Lemma 2.8. (Jacquemard, 1990) *If l is the length of the word w, then the LetterExtractLeft procedure complexity is $O(l^2)$.*

Remark 2.9. The *LetterExtractLeft* procedure can be altered easily to become the *LetterExtractRight* procedure.

The $WordExtractLeft$ algorithm tries to drive a given positive braid word to the left of a positive braid word. It determines if it is possible to write the positive braid word w in such a way that the left part of w will be a given positive braid word.

We make use of this algorithm as a function in the following way: $WordExtractLeft(w, w')$, its inputs are a positive braid word w from which we are going to extract the positive braid word w' to the left. If it is not possible to write $w = w'w''$ for some positive braid word w'', the function will return *false*. If it is possible to write $w = w'w''$ for some positive braid word w'', the return value will be $w'w''$ (i.e. the word w written with w' as its prefix).

Lemma 2.10. (Jacquemard, 1990) *If l_1 is the length of w and l_2 is the length of w', then the complexity of the algorithm is $O(l_2 \cdot l_1^2)$.*

Remark 2.11. The $WordExtractLeft$ procedure can be altered easily to become the $WordExtractRight$ procedure.

The *Normalize* algorithm transforms a given braid word w into a normal form $w = \Delta_n^{-r}$ w', where Δ_n is the fundamental braid word, r is minimal and w' is a positive braid word given in its lowest lexicographical order possible. (This is Jacquemard solution to the braid word problem and it is cubic in the length of the braid word.)

3. The Randomized Algorithm

In this section we will describe our random algorithm for identifying half-twists in any power. First we will describe what a random algorithm is and how it works.

3.1. randomized algorithm

Let D be the set of all possible inputs to an algorithm $\mathcal{A}$, which is supposed to compute a function A. Consider the situation where D is divided into equivalence classes by the relation $\sim$. Suppose that $A : D/\sim \longrightarrow \{0, 1\}$ is well defined mathematically. If $\mathcal{A}$ is an algorithm that will return the wrong answer on some inputs given from D, but on others will return the right answer, and if we have a method to make sure that the answer is correct or not in some cases, this makes a solid foundation for using it as a random algorithm.

In our case, D is the set of braid words w represented by the generators in both positive and negative powers. The equivalence relation $\sim$ is given by $w_1 \sim w_2$ if they are conjugated. This actually means that the half-twists occupy a full equivalence class of $\sim$, and that two half-twists in the same power share the same equivalence class of $\sim$.

The function A returns *true* if w is conjugated to a half-twist in some positive power and *false* otherwise. We will present an algorithm for solving the problem $\mathcal{A}$, but unfortunately we will only be able to be convinced of its result if its answer is *true*, or in some cases a *false* answer, which we will call a *genuine false*. In some cases, although its input w is a half-twist in some positive power, our algorithm will return *false*.

As we will state in the following sections the probability of such an error is low; therefore it will be possible to iterate the algorithm on different elements of the conjugacy class of the input, resulting in a substantial reduction in the probability of the error. It is possible to reduce the probability of an error as much as possible, simply by increasing the number of iterations.

One very important and known example for a random algorithm is the algorithm for checking whether a natural number is prime or not. A randomized method developed by Miller (1976) and Rabin (1980), using iteration of checking "witnesses" to the primality of the given number, results in a probability of an error which can be reduced as much as one wants. Calculation of large prime numbers, especially with a connection to encryption, is done today using this method.

3.2. CHECK FOR CONJUGACY TO σ_i

In this section we will describe the algorithm for checking whether or not a given braid word w is a half-twist in some power. The algorithm is based on two functions that will be described in detail in the subsections below.

The idea of the algorithm is based on the fact that if w is a half-twist, then there exists a braid word q such that $q^{-1}wq = \sigma_i$, for any σ_i generator of the braid group. First we find the power of the alleged half-twist by summing up the powers of the generators in w; this power will be denoted by $k = \deg(w)$. Then we will try to manipulate the braid word w to be written as $w = q^{-1}\sigma_i^k q$; this will be done using the algorithms for braid word manipulation given above.

If the result of the word manipulation is *true* (meaning that we achieved the form $w = q^{-1}\sigma_i^k q$), then we return *true*, since the braid word is obviously a half-twist. Moreover, we can return q and σ_i which are the way the word w is conjugated to a generator of the braid group in the kth power.

In contrast, if the result of the algorithm is *false*, it could still be possible that the input braid word w is conjugated to a generator in the kth power.

However, this problematic situation mentioned above is quite rare, allowing us the possibility to use the random iteration method. In this case we choose a random braid word in the length of w and conjugate w by it, resulting in a new braid word w' which is conjugated to σ_i^k if and only if w is conjugated to σ_i^k. Then we try the algorithm again on w'.

We conjecture, by looking at the data from the experiments that we performed, that the probability of an error in the result does not change significantly when we build the new element w' in the conjugacy class of the given word w. Therefore, although we do not know how to prove that rigorously, we believe that if the probability of an error in the result is smaller than $p < 1$ then, by iterating n times, one can get to a certainty of nearly $1 - p^n$ that the given braid word w is not a half-twist.

In the performance and benchmarks section we will give our estimations of the error rate.

3.3. THE IS HALF TWIST PROCEDURE

First we give the algorithm that tries to manipulate the input braid word w into $w = q^{-1}\sigma_i^k q$, where k is the sum of powers of the braid word w.

But, before we can do that we need to introduce a notation.

NOTATION 3.1. Let w be a braid word. We denote by $w_{i,j}$ the part of w that starts at the ith letter and ends at the jth letter of w. If $j < 1$ then $w_{i,j}$ denotes the empty word.

Algorithm 3.2. *IsHalfTwist*
Input: *w—non empty braid word in its normal form. k—the power of the half-twist.*

Output: *true—if the element is a half-twist to the kth power, false if the element is not a half-twist to the kth power. c—the generator for which we found the conjugacy, q—a positive braid word which conjugates w to c (i.e. $q^{-1}wq = c^k$).*

```
IsHalfTwist$(w, k)$
  $iPos \leftarrow$ the position of the first positive letter in $w$
  $l \leftarrow$ the length of $w$
  for $i \leftarrow 1$ to (the number of strings in $B_n$) $- 1 = n - 1$ do
      for $t \leftarrow 0$ to $k$ do
          $pLeft \leftarrow \sigma_i^t$
          $pRight \leftarrow \sigma_i^{k-t}$
          for $p \leftarrow iPos^{-1}$ to $l$ do
              if $WordExtractRight(w_{iPos,p}, pLeft) = false$ then
                  continue
              if $WordExtractLeft(w_{p+1,l}, pRight) = false$ then
                  continue
              $Test \leftarrow w_{0,p-t}w_{p+k-t+1,l}$
              $q \leftarrow (w_{p+k-l+1,l})^{-1}$
              if $Test = Id$ then
                  return $true, q, \sigma_i$
  return false
```

Proposition 3.3. *Given a braid word w if the algorithm returns true, then the braid word w is a half-twist in the kth power.*

Proof. The algorithm tries to write w as $q\sigma_i^k q'$ for each position in the positive part of w and each generator possible for σ_i^k ($i = 1, \ldots, n-1$). Then, it checks if $q = q'^{-1}$. If this happens, the algorithm will return *true*. But this is exactly the case when w can be written as a conjugacy to a half-twist in the kth power, meaning that w is a half-twist in the kth power. Moreover, if w is not conjugated to σ_i^k, then there is no way to write w as $q\sigma_i^k q'$ where $q = q'^{-1}$, and since the only possible return of *true* is after such writing occur, the algorithm will not return *true* if w is not a half-twist to the kth power. □

We give two examples to illustrate why the algorithm can return *false* although its result should be *true*.

Example 3.4. Look at the braid word $\sigma_1^{-1}\sigma_2\sigma_1\sigma_2\sigma_1^{-1}$. There is no possible letter manipulation, using the two braid relations, without eliminating $\sigma_i\sigma_i^{-1}$ that will change it into $q^{-1}\sigma_i q$ for some braid word q, although $\sigma_1^{-1}\sigma_2\sigma_1\sigma_2\sigma_1^{-1} = \sigma_2$.

Actually, since the algorithm works only on words in Garside normal form, this braid word does not cause any incorrect *false*. The next example gives us a braid word which does cause the algorithm to return an incorrect *false*.

Example 3.5. The following braid word in B_5 is a half-twist (conjugated to σ_1) but the algorithm will return *false* while processing on it:

$\sigma_4^{-1}\sigma_1\sigma_3^{-1}\sigma_1^{-1}\sigma_3^{-1}\sigma_4\sigma_4\sigma_1^{-1}\sigma_1\sigma_2^{-1}\sigma_1\sigma_2^{-1}\sigma_1\sigma_2\sigma_1^{-1}\sigma_2\sigma_1^{-1}\sigma_1\sigma_4^{-1}\sigma_4^{-1}\sigma_3$
$\sigma_1\sigma_3\sigma_1^{-1}\sigma_4$.

We now give an explanation why the algorithm might return incorrect *false* results. Let $w = q^{-1}\sigma_i^k q$ be a half-twist. Therefore, there is a finite sequence of braid relation actions on w that will transform it into $w' \equiv q^{-1}\sigma_i^k q$. We keep track of the position in the word of each of the letters σ_i that belongs to the original σ_i^k of w'. Suppose that we only use the first braid relation ($\sigma_i\sigma_j = \sigma_j\sigma_i$ where $|i-j| > 1$). It is clear that if we eliminate the letters from the original σ_i^k in any stage, this results in a word that equals $Id \in B_n$. Therefore, activation of the first braid relation will never cause a braid word to result in an incorrect result. This leaves only the second rule as a possible cause, and so we have the following lemma:

Lemma 3.6. *Let $w = q^{-1}\sigma_1^k q$ be a braid word such that the IsHalfTwist(w,k) function returns false. Then, in the process of transforming the braid word from $q^{-1}wq$ into w we must use the second braid relation $\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}$.*

Proposition 3.7. *The complexity of the IsHalfTwist algorithm is bounded by $O(n^2 \cdot \log n \cdot k^2 \cdot l^3)$, where n is the number of strings, $k = deg(w)$ and $l = len(w)$.*

Proof. The proof involves the following facts: (a) we go over all possible generators σ_i $1 \le i \le n-1$. (b) We go over all possibilities to divide the σ_i^k into two subwords. (c) We go over all positions p in the positive braid part of the braid word w. The total gives us $O(n \cdot k \cdot l)$ times which we call the WordExtractLeft and WordExtractRight procedures. The complexity of the WordExtractRight procedure in our case is given by $O(k' \cdot l'^2)$, where k' is the power of the left part of σ_i^k and $l' = p - iPos$ at each loop. The complexity of the WordExtractLeft procedure in our case is given by $O(k'' \cdot l''^2)$ where k'' is the power of the right part of σ_i^k and $l'' = l - p$. Together we have that $k' + k'' = k$ and $l' + l'' = l - iPos$, and, therefore, the number of operations we make in the two procedures is bounded by $O(k \cdot l^2)$. Combining that result with the above, we get a total of $O(n \cdot k^2 \cdot l^3)$. To finish the proof, we need to observe that the check whether $Test = Id$ takes $O(n \cdot log(n) \cdot l^2)$ (see, for example; Birman *et al.* (1998); Dehornoy (1995); Dehornoy (1997); for a fast solution for the braid word problem). This yields the total of $O(n^2 \cdot \log n \cdot k^2 \cdot l^3)$ as the total complexity bound for the IsHalfTwist algorithm. □

3.4. some ways of improving the running time

There are some ways to make the algorithm run faster. The first two shortcuts involve keeping track of exactly what happens with the strings of the braid. So, we will enumerate the strings by $1, \dots, n$.

Definition 3.8. We call the number of times that two strings $i \ne j$ cross one another (counted with the positivity induced by the sign of the generator denoting the switch) their *crossing number*, and we denote it by $cr(i,j)$.

Lemma 3.9. *If w is a half-twist to the kth power, and k is even, there must be only one pair of strings with crossing number $cr(i,j) \ne 0$, and then $cr(i,j) = k$.*

Proof. It follows from the fact that using braid relations do not change the crossing numbers of pairs of strings in w, and from the geometrical interpretation of the half-twist elements. □

Therefore, if k is even we can begin our algorithm by counting the crossing numbers of each pair of strings. If $cr(i,j) \neq 0$ for more than one pair, or $cr(i,j) = 0$ for all $i \neq j$, we can return $false$ which is genuine (i.e. the probability of this $false$ result to be correct is 1).

Moreover, if w is a half-twist and we eliminate from w the two strings whose crossing number is not 0, resulting with $w' \in B_{n-2}$, then $w' = Id \in B_{n-2}$. Therefore, if $w' \neq Id \in B_{n-2}$, we can return a *genuine false.*

Definition 3.10. Assign to each letter in w a pair of numbers which represents the numbers of the strings that switch because of this generator; we call this pair the *switching numbers* of the letter.

By keeping track of all the switching numbers of the letters of w from the beginning to the end (no need to defer between positive or negative power of the letter in w), we result in an easy way to compute the permutation of the strings at each part of the word w. After looking at all the letters, we get the permutation of the strings resulting from w.

Lemma 3.11. *Let w be a braid word representing a half-twist to the kth power. Then the permutation induced by w should be Id if k is even or a transposition of exactly two strings if k is odd.*

Proof. The proof immediately follows from the fact that a half-twist always changes the position of two strings along a path between the strings (see the geometrical interpretation of the half-twists). □

Now, we can check if the permutation is consistent with k which is the sum of the powers of the letters in w. If there is inconsistency, we can return a *genuine false.* Moreover, if k is odd, w is a half-twist and we eliminate from w the two strings that permute, we result with $w' t B_{n-2}$ where $w' = Id \leftarrow B_{n-2}$. Therefore, if $w' \neq Id t B_{n-2}$ we can return a exit genuine false.

After performing these two steps, we know what two strings are switching position. Therefore, checking conjugacy to another pair of switching strings is irrelevant, and can be skipped. This means that after we have written our braid word as $w = q\sigma_i^k q'$, we can check if σ_i^k induces the switch of the two strings we found at the beginning. If not, there is no need to check if $q = q'^{-1}$.

It seems that we need to compute what are the switching pair of strings at the position σ_i^k each time we manage to write our braid as $w = q\sigma_i^k q'$, but this is not true. One can easily create the switching pairs for all the letters by one pass over the given braid word w, and then easily maintain the switching pair data by following the next rules:

Lemma 3.12. *In the braid manipulation process, one of the following must occur:*

1. *If we use the first braid rule $\sigma_i\sigma_j = \sigma_j\sigma_i$, then we must switch the two appropriate switching pairs.*

2. If we use the second braid rule $\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}$, then we must switch the two switching pairs assigned to the two ends of the sequence.

Proof. To begin with, it is clear that no other relation is used in the braid manipulation process; therefore we need only to proof that these are the changes in the switching pairs that are induced by the two relations.

For the first rule, since $\sigma_i\sigma_j = \sigma_j\sigma_i$ we know that $|i - j| > 1$. This means that the two switching pairs are distinct. Therefore, the switching pair associated with σ_i and σ_j continue with their letters, and they do not collide.

For the second rule, we know that the switching pairs must be of the form $(a, b), (a, c)$, (b, c). Therefore, the change in the letters induces the change of the switching pairs to $(b, c), (a, c), (a, b)$ (see the diagram in Figure 3.). □

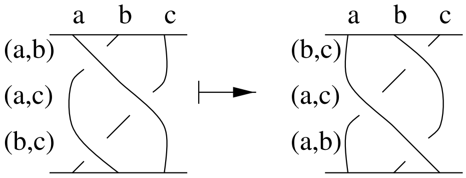

Figure 3. The second rule of switching pairs.

The last improvement in the running time of the algorithm we are going to present is a consequence of a *false* return value of the *LetterExtractLeft* or the *LetterExtractRight* functions. Suppose that we are trying to extract to the right the letter σ_i at position p, and we do not succeed. This means that there is no chance of extracting the letter σ_i to the right until we encounter the next σ_i in the braid word. Therefore, any attempt to extract to the right the letter σ_i for positions from p to the position of the next σ_i letter in w will fail. Therefore, we can skip these tries without affecting the result of the algorithm. Note, that the same is true for the LetterExtractLeft, but this time we should check what is the left-most position that we could extract the letter σ_i to the left and move p immediately to this position. Note that due to the way that the *LetterExtractLeft* function works, this left-most position is given instantly after the first unsuccessful LetterExtractLeft call.

This enables us to skip a large amount of tries to extract the letters into positions that we already know we cannot extract to. By keeping track of what is the left-most or right-most position we can extract each letter, we can make the algorithm run faster.

We want to clarify that although in some cases it is possible to write the word with the letter that was extracted to the left (or to the right) at a position which is left to its position as returned by the LetterExtractLeft algorithm, since the IsHalfTwist procedure does not consider these situations, skipping extraction to these positions does not change the results.

3.5. the test random halftwist algorithm

In this section, we will give the random core of the algorithm, the TestRandomHalfTwist procedure. This procedure gets the braid word and tries successively to check if it is a half-twist. The procedure will terminate in one of three conditions. The first is when

IsHalfTwist returns *true*, this happens when no question about the correctness of the answer exists, therefore one can return the computed result. The second is when we encounter a *genuine false*; here again there is no doubt of the correctness of the result. The third is when we have exceeded some predefined maximum value of tries. In this situation we say that the given braid word w is not conjugated to a half-twist, with the restriction that there could be an error which its probability is less than p (where p will be discussed later).

Algorithm 3.13. *Test Random Half-Twist procedure*
Input: *w—non empty braid word in its normal form.*
Output: *true—if the element is a half-twist to the kth power. False—if the element is not a half-twist to the kth power, or a genuine false if possible. c—the generator for which we found the conjugacy. q—the braid word which conjugate w to c (i.e. $q^{-1}wq = c^k$).*

```
TestRandomHalfTwist(w)
  $r \leftarrow Id$
  $r1 \leftarrow w$
  $i \leftarrow 0$
  $K \leftarrow \deg(w)$
  do
      $i \leftarrow i + 1$
      if $IsHalfTwist(r1, k) = true$ then
          $q \leftarrow r \cdot q$
          $c \leftarrow$ the letter where $IsHalfTwist$ found the conjugacy
          return true
      else
          if the return value was a genuine false then
              return a genuine false.
          else
              $r \leftarrow$ a random braid word of size length$(w)$
              $r1 \leftarrow Normalize(r^{-1} \cdot w \cdot r)$
  while $i <$ maximum number of tries
  return false (not genuine)
```

Proposition 3.14. *Given a braid word w when the algorithm returns true, the result q is the conjugacy word for which $q^{-1}wq = c^k$. Moreover, if the result of the algorithm is false, then it is correct in the probability of $1 - p' \cdot p^{i-1}$ where p is the probability of an error for the result of IsHalfTwist on conjugated words to w, p' is the probability of an error for the result of IsHalfTwist on w itself, and i is the number of tries.*

Proof. The first part is obvious, since the only reason that this procedure returns *true* is when IsHalfTwist returns *true*.

Now, we have to show that the returned conjugacy word q satisfies that $q^{-1}wq = c^k$. We have two cases. The first case is when the algorithm returns *true* in the first pass of the "do" loop. In this case, the braid word that was entered to the IsHalfTwist procedure is w, and the returned q is exactly as the q given by IsHalfTwist; therefore, this is the correct q. In the second case, we have iterated more than once over the "do" loop. At each iteration we have conjugated by r the braid word w resulting with $r1 = r^{-1}wr$.

Therefore, the q returned by the IsHalfTwist procedure satisfies that $q^{-1} \cdot r1 \cdot q = c_i^k$. But $r1 = r^{-1}wr$; therefore, $q^{-1}r^{-1}wrq = c_i^k$, as the algorithm returns.

For the second part of the proposition suppose that the probability to receive an error by activating IsHalfTwist on w is p', and that the probability to receive an error by activating IsHalfTwist on the random conjugates of w is p. Consider the fact that if the algorithm has returned *false* this means that the "do" was executed i times. Therefore, the probability of an erroneous result is $1 - p' \cdot p^{i-1}$. □

Proposition 3.15. *The complexity of the TestRandomHalfTwist procedure is* $O(n^2 \cdot \log n \cdot k^2 \cdot l^3)$.

Proof. Since the number of times we activate the procedure IsHalfTwist is constant, and since all the other procedures we perform in the "do" loop take constant time or less than the time for the IsHalfTwist procedure, this results in a constant times the complexity of the IsHalfTwist procedure which is $O(n^2 \cdot k^2 \cdot l^3)$. □

4. Performance

In this section we will give the probability estimations for the success rate for the IsHalfTwist procedure. We will also state that one can check w^2 instead of w, reducing dramatically the probability for an incorrect *false*.

4.1. probability estimations

Since the combinatoric computations of the braid group that are needed in order to give a result on the probability of an incorrect *false* return value are yet unknown, we have made numerous experiments using a computer program. The results are encouraging since it looks as if the probability to get an incorrect *false* return value from the IsHalfTwist procedure is low.

In our experiments we shuffled random half-twists and transformed them into normal form. Then, we activated the IsHalfTwist once, counting for each length of word the number of times we tried the algorithm and the number of times that the algorithm resulted with *true*. We did this in the groups B_5 and B_8 for powers of the half-twists between 1 and 5 of the half-twists. The results are summarized in the tables below. Each line represents the tests on words of length in an interval of 100 letters. The two numbers in each Power column represent the number of words shuffled and the probability of success of the algorithm.

Table 1. Success probability of the algorithm on B_5.

Length	Power 1	Power 2	Power 3	Power 4	Power 5
100	535, 0.985	673, 1	1149, 1	1139, 1	1196, 1
200	550, 0.829	684, 1	1126, 1	1208, 1	1131, 1
300	608, 0.781	671, 0.997	1087, 1	1046, 1	1118, 1
400	553, 0.772	555, 0.994	941, 1	905, 1	881, 1
500	492, 0.770	311, 0.993	558, 1	542, 1	532, 1
600	452, 0.803	92, 1	127, 1	139, 1	116, 1
700	297, 0.771	14, 1	12, 1	21, 1	26, 1
800	109, 0.779	—	—	—	—
900	44, 0.701	—	—	—	—
1000	—	—	—	—	—

Table 2. Success probability of the algorithm on B_8.

Length	Power 1	Power 2	Power 3	Power 4	Power 5
100	147,1	135,1	141, 1	762, 1	488, 1
200	125,0.944	142,1	148, 1	685, 1	476, 1
300	135,0.755	124,1	138, 1	634, 1	427, 1
400	136,0.669	135,1	130, 1	678, 1	443, 1
500	64,0.625	58,1	59, 1	297, 1	195, 1
600	64,0.577	118,0.991	100, 1	556, 1	410, 1
700	107,0.598	100,1	107, 1	511, 1	417, 1
800	91,0.466	89,0.977	77, 1	411, 1	425, 1
900	55,0.527	58,0.931	54, 1	142, 1	397, 1
1000	15,0.733	22,1	17, 1	189, 1	188, 1
1100	17,0.470	11,1	18, 1	79, 1	374, 1
1200	—	—	6, 1	44, 1	311, 1
1300	—	—	4, 1	9, 1	190, 1
1400	—	—	—	1, 1	73, 1
1500	—	—	—	2, 1	103, 1
1600	—	—	1, 1	—	45, 1
1700	—	—	—	—	24, 1
1800	—	—	—	—	10, 1
1900	—	—	—	—	1, 1
2000	—	—	—	—	3, 1

One may suggest that instead of using conjugation by a random braid word, it may be sufficient to simply increase the number of strings, in order to get a correct result in case of a mistaken *false*. Unfortunately, experiments show that this is not true. If a given braid word w causes an incorrect *false* return value in the IsHalfTwist procedure, then the result is independent of the number of the strings in B_n, Therefore, trying to embed B_n in B_m where $n < m$ will not solve the incorrect *false* problem.

4.2. Square method

During the check of the different benchmarks, we found that the IsHalfTwist algorithm almost never returns an incorrect *false* result on powers greater than 1. In addition to the results summarized in the tables above, we have tested 13 half-twists to the fifth power in B_8 with size in normal form of more than 4500 letters, resulting in the return value of *true* each time. This led us to check the next situation using a computer and we reached the following proposition, which proved in a simple manner in Ben-Itzhak *et al.* (preprint).

Proposition 4.1. *A braid word w is conjugated to the generator σ_i if and only if w^2 is conjugated to σ_i^2.*

One direction of the proposition is obviously true, since if w is conjugated to the generator σ_i then it is obvious that w^2 is conjugated to σ_i^2. The surprising part is the other direction.

The meaning of this is that it is possible to reduce the IsHalfTwist problem for general k into larger numbers. Given a braid word w with $\deg(w) = 1$. Activate IsHalfTwist on w^2 and return its result.

Since the probability of an erroneous result in large powers is much lower, we get a better solution.

Note, that a *false* result returned by checking powers of w implies that w is not a half-twist in any power.

5. Conclusions

We would like to point out that the algorithms given for the LetterExtractLeft and WordExtractLeft, as well as for the Normalize, are based on the natural set of generators for the braid group. It is already known (see, for example, Kang *et al.*, 1997; Birman *et al.*, 1998) that if one changes the set of generators it is possible to improve even more the algorithms complexity. This improvement may reflect immediately on the complexity of the random method given here, if we will find a way to extend the results of Jacquemard (1990) to work on the new sets of generators for B_n.

The above algorithm enables us, in a fast way, to decide whether a given braid word is conjugated to any generator of the braid group in any power. This makes it possible to identify these unique and important elements of the braid group, which are the factors in a factorization yielded from a braid monodromy. For example, if we use this method, it is possible to check whether or not a given factorization of Δ^2 can result from a braid monodromy.

Another implication of this algorithm is the identification of elements that construct quasipositive braids. Therefore, we believe that this algorithm may help to solve the quasipositivity problem in the braid group.

Acknowledgements

This work was partially supported by the Emmy Noether Research Institute for Mathematics, (center of the Minerva Foundation, Germany), and by the Excellency Center "Group Theoretic Methods in the Study of Algebraic Varieties" of the Israel Science Foundation and EAGER (EU network, HPRN-CT-2009-00099). This paper is part of the first author's PhD thesis.

References

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

Ben-Itzhak, T., Kaplan, S., Teicher, M. *Identifying Half-Twists Powers and Computing its Root, Preprint.*

Birman, J. S., Ko, K. H., Lee, S. J. (1998). A new approach to the word and conjugacy problems in the braid groups. *Adv. Math.*, **139**, 322–353.

Dehornoy, P. (1995). From large cardinals to braids via distributive algebra. *J. Knot Theory Ramifications*, **4**, 33–79.

Dehornoy, P. (1997). A fast method for comparing braids. *Adv. Math.*, **125**, 200–235.

Elrifai, E. A., Morton, H. R. (1994). Algorithms for positive braids. *Q. J. Math. Oxford Ser.*, **45**, 479–497.

Garside, F. A. (1969). The braid group and other groups. *Q. J. Math. Oxford Ser.*, **78**, 235–254.

Jacquemard, A. (1990). About the effective classification of conjugacy classes of braids. *J. Pure. Appl. Algebra*, **63**, 161–169.

Kang, E. S., Ko, K. H., Lee, S. J. (1997). Band-generator presentation for the 4-braid group. *Top. Appl.*, **78**, 39–60.

Kulikov, V. S., Teicher, M. (2000). Braid monodromy factorization and diffeomorphism types. *Izv. Math.*, **64:2**, 89–120.

Miller, G. L. (1976). Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, **13**, 300–317.

Moishezon, B., Teicher, M. (1988). Braid group techniques in complex geometry I, line arrangements in $\mathbf{CP}^2$. *Contemp. Math.*, **78**, 425–555.

Rabin, M. O. (1980). Probabilistic algorithm for testing primality. *J. Number Theory*, **12**, 128–138.

Received 16 July 2001
Accepted 20 January 2002