Project Report ——
Algorithms for Braids and Braid Words

Student: Xi Zhang 200906945

Supervisor: Igor Potapov

CONTENT

1 Abstract

Emil Artin firstly posed that the group braids has been interested by the mathematicians since the 1920's. From that on, many people was working on the braids and the braid algorithms. The Braids and Braid Words program is aiming to be built to support the braid ordering algorithms, and many other related algorithms which include braid rewrite, the Garside Normal form, the σ-positive braid and some other algorithms.

The Braids and Braid Words program is designed and actually implemented in parts that divided by the braid data structure and the different algorithms. For the Braids and Braid Words program, when the user input a braid, it could show the braid picture, the rewrite form and the braid picture of the Garside Normal form and the σ-positive form, the result whether the braid is an σ-positive braid, and the user could also compare two braids with both the braid pictures by input the two braids into the program. Besides these functions, the Braids and Braid Words program also support a methods for the experiment that could ordering a list of braid with the input strands number and the maximum braid length.

Experiments about the braid ordering has been done and it is found that the braids could hardly ordered by just looking at the braid words, it is so hard to found a common rule to compare the braids easy and efficient.

2 Introduction

The Algorithms for the Braids and Braid Words project is specified by the supervisor Igor Potapov as the Final Year Project of semester 2013 - 2014. The project is worked on the braids and the braid words, and the rewrite and the comparative of the braid words. And the comparative of the braids is based on the positive form transform of the given braids. [1]

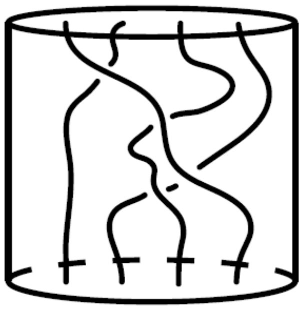


Figure 2.1 The picture of a real braid with 4 strands.

Braids are classical topological objects, and they as well as the topological knots and links has great connection to polymer chemistry, molecular biology, cryptography and robotics. Geometrically, n-braid or a braid on n strands is defined as a cylinder with in *n* continuous paths or stands. And all the braid could easily defined by Greek letter σ (sigma).

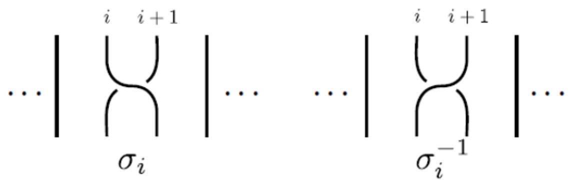


Figure 2.2 The simple example of braid word σ_i and σ_i^{-1}.

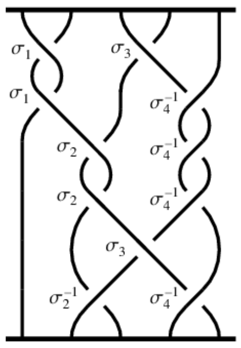


Figure 2.3 The braid picture of braid $\sigma_1\sigma_3\sigma_1\sigma_4{}^{-1}\sigma_2\sigma_4{}^{-1}\sigma_2\sigma_4{}^{-1}\sigma_3\sigma_2{}^{-1}\sigma_4{}^{-1}$

Besides the braid word σ there is another Greek letter Δ could be defended for the half twist braid.

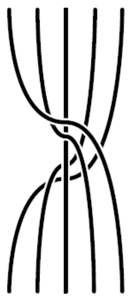


Figure 2.4 The braid picture of Δ_5

There three very important rewrite rule for all braids. The first rule is:

$$\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} (i = 1,2,\cdots,n-2)$$

And it mean that the strand above or below other strand could be move but not influence the equivalence of the whole braid. The second rule is:

$$\sigma_i\sigma_j = \sigma_j\sigma_i (|i-j| > 1)$$

And it means that two nearby and not related braid could exchange the location but not influence the equivalence of the whole braid. The third rule is:

$$\sigma_i{\sigma_i}^{-1} = e$$

And it could be easily seen that when connect on positive case braid and one inverse case braid could cancel them together.

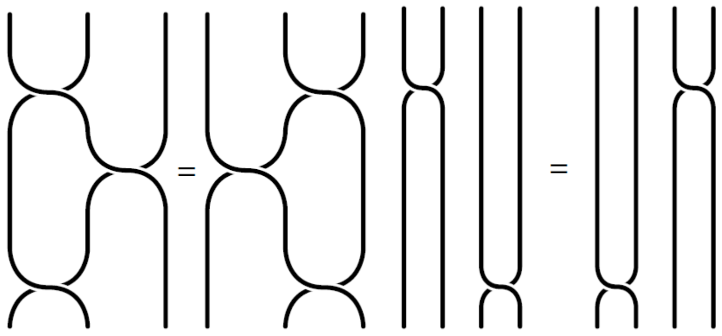


Figure 2.5 The example of the first and second rewrite rule

- **Aims**

  The main goal of the project is to build a Java based program which enable to do complex computation of braids, which might include the rewrite of the braids, the transformation from a braid to the σ-positive form or the Garside normal form, and also include the comparative between two different braids.

- **Challenges and Solution**

When working on the project, it has been met several challenges. It has two main challenge of the project, the first is the understanding of the braid and the braid algorithms, and the other is the structure building design of the braid words.

For the understanding of the braid and braid algorithms, several books has been read, and discussion with the supervisor. For the design of the braid structure, the main difficulty is that the java tree method didn't support the 1 to n tree node and, it is did not support some method for recursion, so a class of TreeNode is written in the program used as the structure of the rewrite tree of the braid.

Besides that, the visualization of the braid is another challenge in the project. To resolve this problem, several pictures is used to show the single braid word or the strand. Let the braid initialize as several strands, and filter the braid then instead σ_i or ${\sigma_i}^{-1}$ one by one on the corresponding position

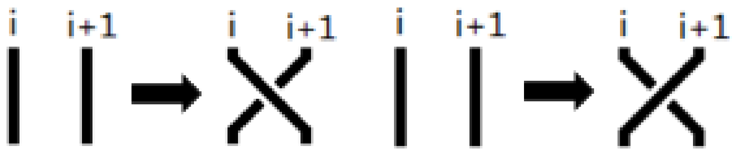


Figure 2.6 The instead example of σ_i and ${\sigma_i}^{-1}$ according to the visualization

The final program could efficiently solve and get the result of the σ-positive braid, the rewrite of Garside Normal form, braid rewrite and braid ordering. But, because of the σ-positive braid rewrite is not proved or the large amount calculation in the java machine, when calculate the σ-positive braid with large index number or huge length, the program sometimes crash. The other methods works regularly in the program.

3 Background

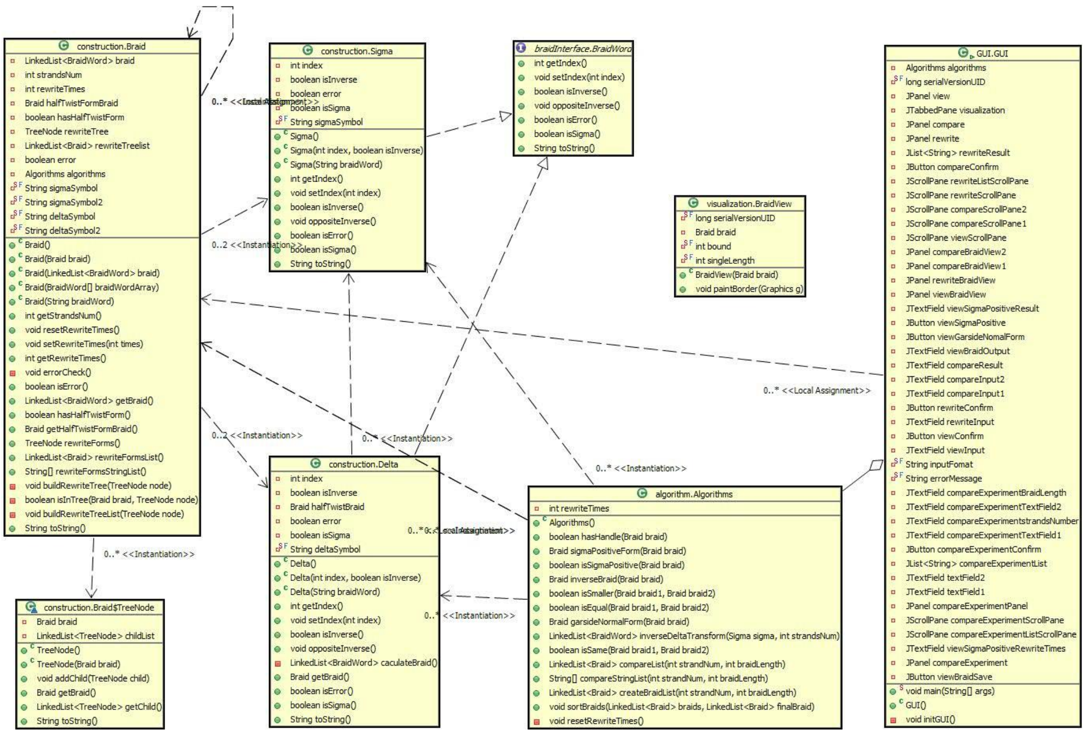
There are not less progress on the braid algorithms has been done in the past years. Jonathan Gomez Boiser has had a research on the computational problems in the braid group which introduce the braid group and some algorithms to cryptography and other fields. He has pointed out that the braid group has been interested by the mathematicians since the 1920's. In his research, the Garside Normal form has been pointed out. And that in Jonathan Gomez Boiser's research, the Garside Normal form is a braid rewrite form that rewrite the braid with inverse cases braid words into new braid that only have positive braid words on the right and the inverse case half twist braid words.

And Patrick Dehornoy had also done large amount research on braids and braids. In his research [5], the σ-positive braid and the braid ordering have been introduced carefully. The research shows how to check whether the braid is an σ-positive braid, and it also said that when braid ordering, the program need to connect the inverse case with the second braid. And when checking the σ-positive braid, the program will remove the handle first. The program remove the handle in two ways, the first way is to remove the handle to the end of the braid and the second is just remove to the nearest place on the right of the braid. The first case is easy but not comfier that for every braid could remove to the end of the braids. The second case is just rewrite it one braid to rewrite to the nearest right braid. Although the method is efficient than the first one, but this method has been proofed.

There is some researches about the braids or the braid algorithms, but there is less software that working support the braids. The Braids and Braid Words program is such a program developed according to and obey the methods above.

4 Design

4.1 Data Structure Design



construction.Braid
LinkedList<BraidWord> braid
int strandsNum
int rewriteTimes
Braid halfTwistFormBraid
boolean hasHalfTwistForm
TreeNode rewriteTree
LinkedList<Braid> rewriteTreelist
boolean error
Algorithms algorithms
String sigmaSymbol
String sigmaSymbol2
String deltaSymbol
String deltaSymbol2
Braid()
Braid(Braid braid)
Braid(LinkedList<BraidWord> braid)
Braid(BraidWord[] braidWordArray)
Braid(String braidWord)
int getStrandsNum()
void resetRewriteTimes()
void setRewriteTimes(int times)
int getRewriteTimes()
void errorCheck()
boolean isError()
LinkedList<BraidWord> getBraid()
boolean hasHalfTwistForm()
Braid getHalfTwistFormBraid()
TreeNode rewriteForms()
LinkedList<Braid> rewriteFormsList()
String[] rewriteFormsStringList()
void buildRewriteTree(TreeNode node)
boolean isInTree(Braid braid, TreeNode node)
void buildRewriteTreeList(TreeNode node)
String toString()
construction.Sigma
int index
boolean isInverse
boolean error
boolean isSigma
String sigmaSymbol
Sigma()
Sigma(int index, boolean isInverse)
Sigma(String braidWord)
int getIndex()
void setIndex(int index)
boolean isInverse()
void oppositeInverse()
boolean isError()
boolean isSigma()
String toString()
braidInterface.BraidWord
int getIndex()
void setIndex(int index)
boolean isInverse()
void oppositeInverse()
boolean isError()
boolean isSigma()
String toString()
GUI.GUI
Algorithms algorithms
long serialVersionUID
JPanel view
JTabbedPane visualization
JPanel compare
JPanel rewrite
JList<String> rewriteResult
JButton compareConfirm
JScrollPane rewriteListScrollPane
JScrollPane rewriteScrollPane
JScrollPane compareScrollPane2
JScrollPane compareScrollPane1
JScrollPane viewScrollPane
JPanel compareBraidView2
JPanel compareBraidView1
JPanel rewriteBraidView
JPanel viewBraidView
JTextField viewSigmaPositiveResult
JButton viewSigmaPositive
JButton viewGarsideNomalForm
JTextField viewBraidOutput
JTextField compareResult
JTextField compareInput2
JTextField compareInput1
JButton rewriteConfirm
JTextField rewriteInput
JButton viewConfirm
JTextField viewInput
String inputFomat
String errorMessage
JTextField compareExperimentBraidLength
JTextField compareExperimentTextField2
JTextField compareExperimentstrandsNumber
JTextField compareExperimentTextField1
JButton compareExperimentConfirm
JList<String> compareExperimentList
JTextField textField2
JTextField textField1
JPanel compareExperimentPanel
JScrollPane compareExperimentScrollPane
JScrollPane compareExperimentListScrollPane
JTextField viewSigmaPositiveRewriteTimes
JPanel compareExperiment
JButton viewBraidSave
void main(String[] args)
GUI()
void initGUI()
visualization.BraidView
long serialVersionUID
Braid braid
int bound
int singleLength
BraidView(Braid braid)
void paintBorder(Graphics g)
construction.Braid$TreeNode
Braid braid
LinkedList<TreeNode> childList
TreeNode()
TreeNode(Braid braid)
void addChild(TreeNode child)
Braid getBraid()
LinkedList<TreeNode> getChild()
String toString()
construction.Delta
int index
boolean isInverse
Braid halfTwistBraid
boolean error
boolean isSigma
String deltaSymbol
Delta()
Delta(int index, boolean isInverse)
Delta(String braidWord)
int getIndex()
void setIndex(int index)
boolean isInverse()
void oppositeInverse()
LinkedList<BraidWord> caculateBraid()
Braid getBraid()
boolean isError()
boolean isSigma()
String toString()
algorithm.Algorithms
int rewriteTimes
Algorithms()
boolean hasHandle(Braid braid)
Braid sigmaPositiveForm(Braid braid)
boolean isSigmaPositive(Braid braid)
Braid inverseBraid(Braid braid)
boolean isSmaller(Braid braid1, Braid braid2)
boolean isEqual(Braid braid1, Braid braid2)
Braid garsideNormalForm(Braid braid)
LinkedList<BraidWord> inverseDeltaTransform(Sigma sigma, int strandsNum)
boolean isSame(Braid braid1, Braid braid2)
LinkedList<Braid> compareList(int strandNum, int braidLength)
String[] compareStringList(int strandNum, int braidLength)
LinkedList<Braid> createBraidList(int strandNum, int braidLength)
void sortBraids(LinkedList<Braid> braids, LinkedList<Braid> finalBraid)
void resetRewriteTimes()
0..* <<Local Assignment>>
0..2 <<Instantiation>>
0..* <<Local Assignment>>
0..2 <<Instantiation>>
0..* <<Instantiation>>
0..* <<Instantiation>>
0..* <<Local Assignment>>
0..* <<Instantiation>>
0..* <<Instantiation>>

Figure 4.1 The Class Diagram of the Braids and Braid Words Software

4.1.1 Braid Words

- Braid Word (interface)

The braid word is created as an interface class that is implemented by σ and Δ. The braid word interface contain the classes support the algorithms of σ and Δ, which included the component of the all braid words, and some Boolean judgment for the algorithms.

On the other hand, the braid word interface is also used as a data type of braid word. Although σ and Δ are both braid words, but they are actually different, so the class is used as the conclusion of their data type.

- σ —— Sigma

For every single braid word sigma (σ_i or σ_i^{-1}), it is and it shows the simplest strand cross of the given two strands.

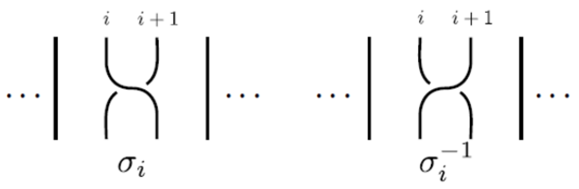


Figure 4.2 The Example of braid σ_i and σ_i^{-1}

According to the example from Figure 4.2, it is easy to see that all braid word sigma have three main part, the first is the Greek letter "σ", and other two parts are its index number and the inverse mark of the braid word. For a given braid σ_i^j, than

i — the index of the braid

j — the inverse mark of the braid

For every single braid word sigma, an object implements BraidWord is used to store. And it should include the index of the braid, a variable to store whether the braid word stand for an inverse braid and a list of variables and methods supports the algorithms with σ.

- Δ —— Delta

Δ is the fundamental braid word, and sometimes called half-twist according to [1]. And every fundamental braid word could be write into a list of braid words. For a given fundamental braid word delta, it could be defined as:

If $\Delta_n \in B_n$ than

$$\Delta_n := (\sigma_1 \cdots \sigma_{n-1})(\sigma_1 \cdots \sigma_{n-2}) \cdots (\sigma_1\sigma_2)\sigma_1$$

If $\Delta_n^{-1} \in B_n$ than

$$\Delta_n^{-1} := (\sigma_1^{-1} \cdots \sigma_{n-1}^{-1})(\sigma_1^{-1} \cdots \sigma_{n-2}^{-1}) \cdots (\sigma_1^{-1}\sigma_2^{-1})\sigma_1^{-1}$$

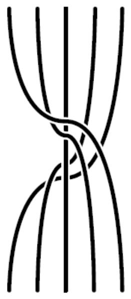


Figure 4.3 The braid picture of Δ_5

Similarly to braid word σ the fundamental braid word Δ could also divided in to three main parts, the Greek letter "Δ", ts index number and the inverse mark of the fundamental braid word. For a given fundamental braid Δ_i^j, than

i — the index of the fundamental braid

j — the inverse mark of the fundamental braid

For every single fundamental braid word delta, another object implements BraidWord is used to store. And it should include the index of the fundamental braid, a variable to store whether the fundamental braid word stand for an inverse braid and a list of variables and methods supports the algorithms with Δ. Besides that, the object also stores what the rewrite form of Δ is

4.1.2 Braid

- b_n

In [1], b_n is a finitely presented group of a set of n-braids, it is mean that the highest index number of σ in b_n is smaller equal then n-1 (The index number of Δ in b_n is n), and b_n is one braid in B_n. Then b_n could be defined as below:

$$b_n = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_1} \cdots \sigma_{i_m}{}^{j_m}\ (i_1, i_2, \cdots, i_m \le n-1, b_n \in B_n)$$

For each braid, they are stored in objects. For the braids in the object, a Java linked list with a type Braid (LinkedList<BraidWord>) is used to store the all the braids words object (σ or Δ) one by one. And the Braid class also store the transform types of the braid and support the algorithms' calculation.

- Rewrite Tree

According to the three rewrite rules of the braids, the braids could be rewrite into difference cases, these rewrite forms are saved in the braid object with a tree mode that written in the inner class in the braid class. The TreeNode structure is a 1 to n tree applied by use java linked list with the data type TreeNode (LinkedList<TreeNode>) as the child in every single node.

A modification compare to the Design document here is that, the design document is using the java JTree to apply the rewrite tree.

4.2 Algorithm Design

4.2.1 σ-Positive

In [5], it is defined that, if a braid is an σ-positive braid, all the braid words with the smallest index must be positive after all the smallest handles has been removed.

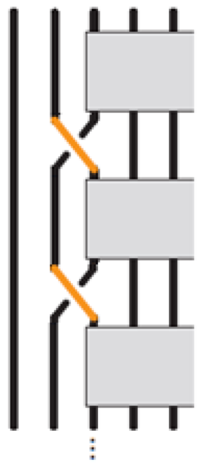


Figure 4.4 The example of the σ-Positive braid

And the handle is if the positive case and the inverse case of σ with smallest index appears in the braid at the same time. And a handle is the structure with the nearest two such model.

$$b_n = \cdots \sigma_m \cdots {\sigma_m}^{-1} \cdots \text{(m is the smallest)}$$

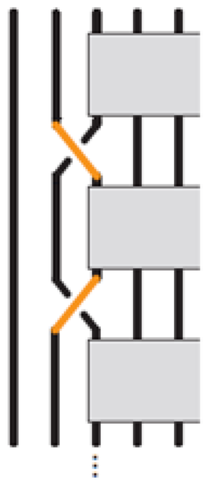


Figure 4.5 The example of a handle

Handle Remove

For Braid $B_n = w_1\sigma_i^x w_2 \sigma_i^{-x} w_3$

(i is the smallest index of all braid word)

w_2' is w_2 after the shift

Then the Braid after handle remove B_n' is

$w_1\sigma_{n-i}^{-x} \dots \sigma_{n-1}^{-x}\sigma_n^{-x} w_2' \sigma_n^x \sigma_{n-1}^x \dots \sigma_{n-i}^x w_3$

Braid Shift

For Braid $B_n =$

$\sigma_{i_1}^{x_1}\sigma_{i_2}^{x_2}\dots\sigma_{i_n}^{x_n}$

B_n' is $B_n$after the shift

$B_n' = \sigma_{i_1-1}^{x_1}\sigma_{i_2-1}^{x_2}\dots\sigma_{i_n-1}^{x_n}$

Here is an example, let us give

$$b_4 = \sigma_1{}^{-1}\sigma_2\sigma_3\sigma_1\sigma_2{}^{-1}(b_4 \in B_4)$$

1 is the smallest index in b$_4$, and it appears a handle (σ_1 and σ_1^{-1}), so the handle is moved to the most right of the braid, and b_4 is transform to b_4'

$$b_4' := \sigma_2\sigma_3\ \sigma_1\sigma_2\sigma_3{}^{-1}\sigma_2{}^{-1}\sigma_2{}^{-1}$$

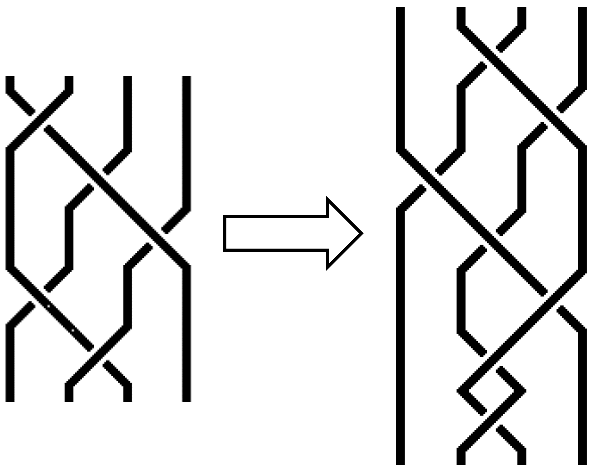


Figure 4.6 $b_4 := \sigma_1{}^{-1}\sigma_2\sigma_3\sigma_1\sigma_2{}^{-1}$ transform to $b_4' := \sigma_2\sigma_3\ \sigma_1\sigma_2\sigma_3{}^{-1}\sigma_2{}^{-1}\sigma_2{}^{-1}$

For b_4', 1 is the smallest index, and there is no handle appears. Than no σ_1^{-1} appears in the braid, so b_4 is an σ-positive braid.

In general case, an algorithms is designed to check whether one braid is σ-positive, to get the result, an iteration is used to remove all the handles in the braid, and it consists two parts

1) find the handle

   A traversal of b$_n$ is done with a temp value to store the found smallest braid word, once a smaller index is found the temp value should be updated, and if another braid with the same index but the negative case, the case will also be tempted and returned if there is no other smaller index in the braid.

2) remove the handle

Once a handle is found, it should be removed to the most right of the braid, if $b_n = w_1\sigma_i w_2 \sigma_i^{-1} w_3$ (i is the smallest index, and $\sigma_i \cdots \sigma_i^{-1}$ is the first pair), than

$$b_n' = w_1(\sigma_{i+1}^{-1}\sigma_{i+2}^{-1}\cdots\sigma_{n-1}^{-1})w_2'(\sigma_{n-1}\cdots\sigma_{i+2}\sigma_{i+1})w_3$$

Else if $b_n = w_1\sigma_i^{-1}w_2\sigma_i w_3$ (i is the smallest index, and $\sigma_i \cdots \sigma_i^{-1}$ is the first pair), than

$$b_n' := w_1(\sigma_{i+1}\sigma_{i+2}\cdots\sigma_{n-1})w_2'(\sigma_{n-1}^{-1}\cdots\sigma_{i+2}^{-1}\sigma_{i+1}^{-1})w_3$$

w_2' it is w_2 shift to left for one index.

$$w_2 = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_2}\cdots\sigma_{i_m}{}^{j_m}$$
$$w_2' = \sigma_{i_1-1}{}^{j_1}\sigma_{i_2-1}{}^{j_2}\cdots\sigma_{i_m-1}{}^{j_m}$$

After the rewritten of the braid a new iteration is started with b_n'.

For the final step to check whether the braid is σ-positive is, when get the non-handle braid, one more check of the smallest index should be done, and if all the braid with the smallest index is positive, than the braid is σ-positive, if once appear an inverse case, it should be not σ-positive.

Pseudo code of the algorithm to get the σ-positive form of the given braid

```
isSigmaPositive(w)
    boolean isHandle := false
    boolean tmpInverse := null
    while !isHandle do
        int tmpIndex := infinite
        tmpInverse := null
        Braid tmpBraid := null
        for $\sigma_i{}^j$ as each braid word in w do
            if i < tmpIndex do
                tmpIndex := i
                tmpInverse := j
                tmpBraid := null
                continue
            else if i = tmpIndex do
                if tmpInverse = j do
                    continue
                else if tmpInverse != j do
                    Braid w1 := the braids before $\sigma_{tmpIndex}{}^{tmpInverse}$
                    Braid w2 := the braids between $\sigma_{tmpIndex}{}^{tmpInverse}$ and $\sigma_i{}^j$
                    Braid w2 := the braids after $\sigma_i{}^j$
                    tmpBraid := $w_1\sigma_i{}^{-j}w_2\sigma_i{}^j w_3$
                endIf
            endIf
        endFor
        if tmpBraid != null do
            for $\sigma_i{}^j$ as each braid word in w2 do
                $\sigma_i{}^j := \sigma_{i-1}{}^j$
            endFor
            w := $w_1(\sigma_{i+1}{}^j\sigma_{i+2}{}^j\cdots\sigma_{n-1}{}^j)w_2(\sigma_{n-1}{}^{-j}\cdots\sigma_{i+2}{}^{-j}\sigma_{i+1}{}^{-j})w_3$
        else if tmpBraid = null do
            isHandle := true
        endIf
    endWhile
    if !tmpInverse do
        return true
    else if tmpInverse do
        return false
    endIf

Braid $b_n$ := the input braid
return isSigmaPositive($b_n$)
```

4.2.2 Garside Normal Form

The Garside normal form if achievement of Frank Garside who use it to solve the conjugacy problem in the braid groups. The Garside normal form change a normal braid group with inverse braid into a braid group with only positive braids on the right and the inverse fundamental braid word on the left.

$$b_n = \Delta^{-m} b'^{+}_{n}$$

The transformation is based on two equations. The first is that, all the braid word $w \in B_n$ could be written uniquely

$$w = \Delta^m p \ (m \in Z, p \in B_n^+)$$

or

$$w = p_1 \Delta^m p_2 \ (m \in Z, p_1 \in B_n^+, p_2 \in B_n^+)$$

The second is the fundamental braid word could be moved in B_n according to

$$\sigma_i \Delta = \Delta \sigma_{n-i}$$

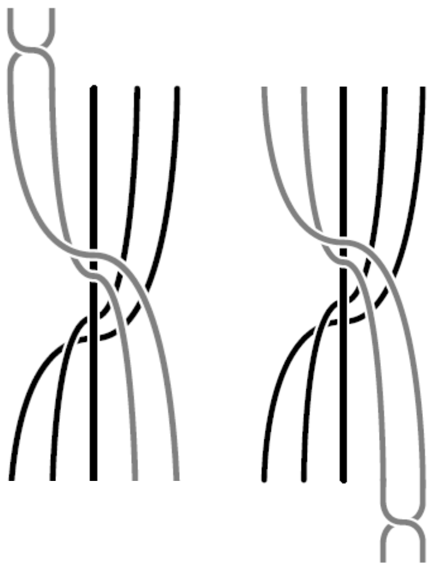


Figure 4.7 In B5, $\sigma_1\Delta = \Delta\sigma_4$

Here is an example, let us give

$$w = \sigma_3\sigma_2{}^{-1}\sigma_3{}^{-1}\sigma_1\sigma_2\sigma_3{}^{-1}$$

All the inverse case braids are picked out and get the equations

$$\sigma_2 = \sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\Delta^{-1}$$

$$\sigma_3 = \sigma_2\sigma_1\Delta^{-1}\sigma_1\sigma_2\sigma_1 = \sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\Delta^{-1}$$

Than

$$w = \sigma_3\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\Delta^{-1}\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\Delta^{-1}\sigma_1\sigma_2\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\Delta^{-1}$$

Than move all the Δ^{-1} to the left to get the final Garside normal form

$$w = \Delta^{-3}\sigma_1\sigma_3\sigma_2\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\sigma_3\sigma_2\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1$$

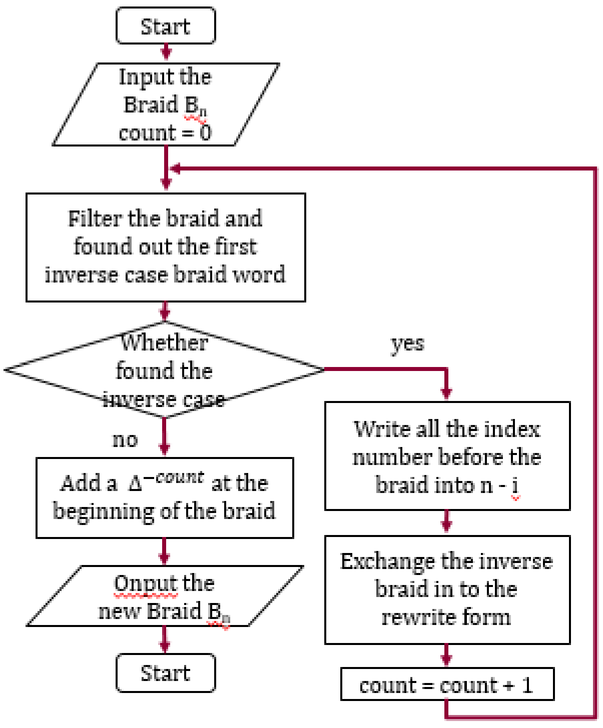

Start
Input the Braid B_n count = 0
Filter the braid and found out the first inverse case braid word
Whether found the inverse case
yes
no
Write all the index number before the braid into n - i
Add a Δ^{-count} at the beginning of the braid
Exchange the inverse braid in to the rewrite form
Onput the new Braid B_n
count = count + 1
Start

Figure 4.8 The flow diagram of the Garside Nomal Form

In general case, an algorithms is designed to rewrite the given braid to the Garside normal form. It is consider that

$$\mathrm{w} = w_1\sigma_{i_1}^{-1}w_2\sigma_{i_2}^{-2}\cdots w_s\sigma_{i_s}^{-1}w_{s+1}(\mathrm{w} \in\ B_n)$$

All inverse case braids are picked out and get the equations according to

$$\mathrm{w} = \Delta^m p\ (m \in Z, p \in B_n^+)$$

So it is easy to get

$$\sigma_{i_s}^{-1} = \mathrm{x}_s\Delta^{-1}$$

Than

$$\mathrm{w} = w_1\mathrm{x}_1\Delta^{-1}w_2\mathrm{x}_2\Delta^{-1}\cdots w_s\mathrm{x}_s\Delta^{-1}w_{s+1}$$

Than move all the Δ^{-1} to the left ($\Delta_s{}^{-1}w'_{n-s}\mathrm{x}'_{n-s} = w_s\mathrm{x}_s\Delta_s{}^{-1}$)

If *s* is odd

$$\mathrm{w} = \Delta^{-s}w'_{n-1}\mathrm{x}'_{n-1}w_2\mathrm{x}_2\cdots w'_{n-s}\mathrm{x}'_{n-s}w_{s+1}$$

If *s* is even

$$\mathrm{w} = \Delta^{-s}w_1\mathrm{x}_1w'_{n-2}\mathrm{x}'_{n-2}\cdots w'_{n-s}\mathrm{x}'_{n-s}w_{s+1}$$

Pseudo code of the algorithm to get the Garside normal form of the given braid

```
Braid $b_n$ := the input braid
int count := 0
for $\sigma_i{}^j$ as each braid word in $b_n$ do
    if j is inverse do
        count++
        Braid w1 := the Braid before $\sigma_i{}^j$
        Braid w2 := the Braid after $\sigma_i{}^j$
        Braid w3 := $x_{i_1}$ (where $\sigma_i{}^{-1} = x_{i_1}\Delta^{-1}x_{i_2}$)
        Braid w4 := $x_{i_2}$ (where $\sigma_i{}^{-1} = x_{i_1}\Delta^{-1}x_{i_2}$)
        for $\sigma_m$ as each braid word in w1 do
            $\sigma_m = \sigma_{n-m}$
        endFor
        for $\sigma_m$ as each braid word in w3 do
            $\sigma_m = \sigma_{n-m}$
        endFor
        $b_n$ := w1w3w4w2
    endIf
endFor
$b_n$ := $\Delta^{-\text{count}}b_n$
return $b_n$
```

4.2.3 Braid Rewrite

All the braids has three rewrite rules. If the braid group is B_n, the first rule is:

$$\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}(i = 1,2,\cdots,n-2)$$

The second rule is:

$$\sigma_i\sigma_j = \sigma_j\sigma_i(|i-j| > 1)$$

The third rule is:

$$\sigma_i\sigma_i{}^{-1} = e$$

But this rule is not consider in this program, this program is only consider the all positive or all inverse cases.

Here is an example, let us give

$$b = \sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1$$

Than we can get

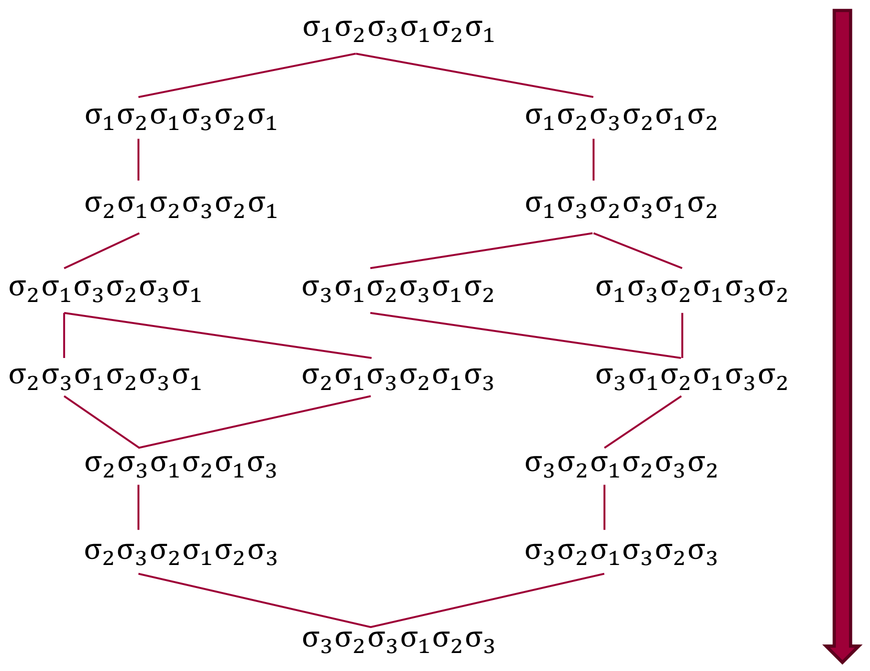

$\sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1$
$\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1$
$\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\sigma_2$
$\sigma_2\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1$
$\sigma_1\sigma_3\sigma_2\sigma_3\sigma_1\sigma_2$
$\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\sigma_1$
$\sigma_3\sigma_1\sigma_2\sigma_3\sigma_1\sigma_2$
$\sigma_1\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2$
$\sigma_2\sigma_3\sigma_1\sigma_2\sigma_3\sigma_1$
$\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1\sigma_3$
$\sigma_3\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2$
$\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1\sigma_3$
$\sigma_3\sigma_2\sigma_1\sigma_2\sigma_3\sigma_2$
$\sigma_2\sigma_3\sigma_2\sigma_1\sigma_2\sigma_3$
$\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3$
$\sigma_3\sigma_2\sigma_3\sigma_1\sigma_2\sigma_3$

Figure 4.9 The rewrite tree of braid $\sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1$

So all the rewrite form of b are $\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1$, $\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\sigma_2$, $\sigma_2\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1$, $\sigma_1\sigma_3\sigma_2\sigma_3\sigma_1\sigma_2$, $\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\sigma_1$, $\sigma_3\sigma_1\sigma_2\sigma_3\sigma_1\sigma_2$, $\sigma_1\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2$, $\sigma_2\sigma_3\sigma_1\sigma_2\sigma_3\sigma_1$, $\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1\sigma_3$, $\sigma_3\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2$, $\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1\sigma_3$, $\sigma_3\sigma_2\sigma_1\sigma_2\sigma_3\sigma_2$, $\sigma_2\sigma_3\sigma_2\sigma_1\sigma_2\sigma_3$, $\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3$, $\sigma_3\sigma_2\sigma_3\sigma_1\sigma_2\sigma_3$. And the depth of it is 7.

In general case, an algorithms is designed to get all the rewrite forms of the given braid. To achieve that, recursion is used. Let

$$b = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_1}\cdots\sigma_{i_m}{}^{j_m}\ (i_1, i_2, \cdots, i_m \le n-1, b \in B_n\ )$$

For each time of recursion there is mainly two steps

1) The first step is to found out the all the cases which could be rewrite once in b_n according for the cases
$$\sigma_i{}^j\sigma_{i+1}{}^j\sigma_i{}^j(i = 1,2,\cdots,n-2)$$
$$\sigma_i{}^m\sigma_j{}^n(|i-j| > 1)$$
2) Rewrite the positive cases according to
$$\sigma_i{}^j\sigma_{i+1}{}^j\sigma_i{}^j = \sigma_{i+1}{}^j\sigma_i{}^j\sigma_{i+1}{}^j(i = 1,2,\cdots,n-2)$$
$$\sigma_i{}^m\sigma_j{}^n = \sigma_j{}^m\sigma_i{}^n(|i-j| > 1)$$
And start the recursion using the rewritten once braid, if the rewritten braid has never been appeared before.

The program apply the algorithms and create the whole braid rewrite tree by using recursion. In each iterator, the program filter the braid, once it found a possible rewrite form of the braid, it check through the tree for whether it is exist, if not, create a new iteration with the found braid as the input.

Pseudo code of the algorithm to get the rewirte tree of the given braid

```
Braid $b_n$ := the input braid
root firstRoot := $b_n$
Tree rewriteTree := where first root is firstRoot

BuildTree(root r)
  Braid tmpB := Braid in r
  for $\sigma_x\sigma_y$ as each $\sigma_i\sigma_j(|i-j|>1)$ case in tmpB do
    Braid tmpB' := tmpB change $\sigma_x\sigma_y$ to $\sigma_y\sigma_x$
    If tmB' is in the not in rewriteTree do
      root r': = tmB'
      set r' as one of the leave of r
      BuildTree(r')
    endIf
  endFor
  for $\sigma_x\sigma_{x+1}\sigma_x$ as each $\sigma_i\sigma_{i+1}\sigma_i(i=1,2,\cdots,n-2)$ case in tmpB do
    Braid tmpB' := tmpB change $\sigma_x\sigma_{x+1}\sigma_x$ to $\sigma_{x+1}\sigma_x\sigma_{x+1}$
    If tmB' is in the not in rewriteTree do
      root r': = tmB'
      set r' as one of the leave of r
      BuildTree(r')
    endIf
  endFor

BuildTree(firstRoot)
return rewriteTree
```

4.2.4 Braid Ordering

In [2], it is said that braids could also be ordering between each other. For

$$\mathrm{b}, \mathrm{b}' \in B_n$$

It is proved $\mathrm{b} < \mathrm{b}'$, if the braid $\mathrm{b}^{-1}\mathrm{b}'$ is an σ-positive braid.

Before the braid ordering, the inverse case of the braid should be calculated first. If there is a braid b_n

$$b_n = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_1}\cdots\sigma_{i_m}{}^{j_m}\ (i_1, i_2, \cdots, i_m \le n-1)$$

Than the inverse case of the braid should be ${b_n}^{-1}$

$$b_n^{-1} = \sigma_{i_m}^{-j_m}\sigma_{i_{m-1}}^{-j_{m-1}}\cdots\sigma_{i_1}^{-j_1}\ (i_1, i_2, \cdots, i_m \le n-1)$$

Here is an example of the braid ordering, let us give

$$\mathrm{b} = \sigma_1\sigma_3^{-1}\sigma_2$$

$$\mathrm{b}' = \sigma_2\sigma_1$$

Than

$$\mathrm{b}^{-1}\mathrm{b}' = \sigma_2^{-1}\sigma_3\sigma_1^{-1}\sigma_2\sigma_1 = \sigma_3^{-1}\sigma_2\sigma_2\sigma_3\sigma_1\sigma_3^{-1}\sigma_2^{-1}$$

And it is an σ-positive braid, so $\mathrm{b} < \mathrm{b}'$

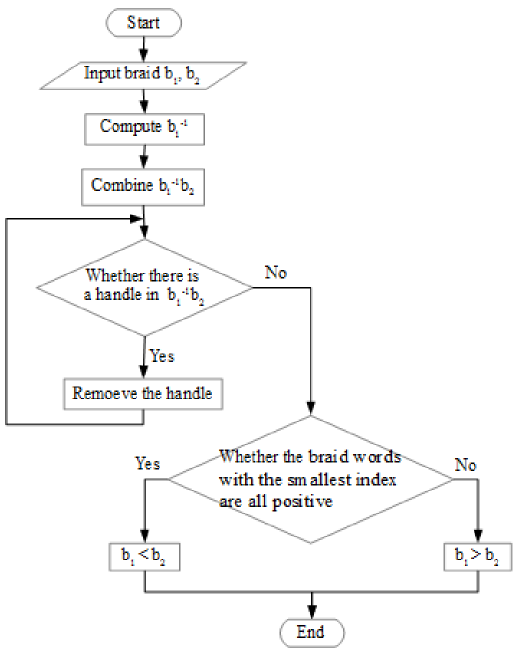

Start
Input braid b₁, b₂
Compute b₁⁻¹
Combine b₁⁻¹b₂
Whether there is a handle in b₁⁻¹b₂
No
Yes
Remoeve the handle
Whether the braid words with the smallest index are all positive
Yes
No
b₁<b₂
b₁>b₂
End

Figure 4.10 The flow diagram of the Braid Ordering

In general case, an algorithms is designed to check the order between two braids. Let

$$\mathrm{b} = \sigma_{i_1}^{j_1}\sigma_{i_2}^{j_1}\cdots\sigma_{i_m}^{j_m}\ (i_1, i_2, \cdots, i_m \le n-1)$$

$$\mathrm{b}' = \sigma_{i'_1}^{j'_1}\sigma_{i'_2}^{j'_1}\cdots\sigma_{i'_{m'}}^{j'_{m'}}\ (i'_1, i'_2, \cdots, i'_{m'} \le n-1)$$

Than $\mathrm{b}^{-1}\mathrm{b}'$ could be get

$$\mathrm{b}^{-1}\mathrm{b}' = \sigma_{i_{m'}}^{-j_{m'}}\sigma_{i_{m'-1}}^{-j_{m'-1}}\cdots\sigma_{i_1}^{-j_1}\sigma_{i'_1}^{j'_1}\sigma_{i'_2}^{j'_1}\cdots\sigma_{i'_m}^{j'_m}$$

If $\mathrm{b}^{-1}\mathrm{b}'$ is σ-positive braid, than $\mathrm{b} < \mathrm{b}'$

If $\mathrm{b}^{-1}\mathrm{b}'$ is not σ-positive braid, than $\mathrm{b} > \mathrm{b}'$

Pseudo code of the algorithm to check the order between two braids

```
Braid b, b' := the input two braids
Braid tmpB := null
for $\sigma_i{}^j$ as each braid word in b do
    tmpB := $\sigma_i{}^{-j}$tmpB
endFor
tmpB := tmpBb'
boolean isSmaller := isSigmaPositive(tmpB)
if isSmaller do
    return $b < b'$
else if !isSmaller do
    return $b > b'$
endIf
```

4.3 GUI Design

The user interface is one of the most important part of the program, it enables user to input the braids conveniently, and it gave a particular way to visualize the braids and to show the rewrite forms. The user interface could also improve the quality of research.

The user interface is also based on Java, and it should be achieved by Java graphic user interface. For the specific user interface, it should include the following points:

1) The visualization part of the braid group
2) The input text field of the braid group, enable to use the braid words σ and Δ
3) The area shows the equivalent braid groups of the input braid group, and the σ-positive form (the rewrite case which remove all the handles) and the Garside normal form is also include in it.
4) Show weather the braid is σ-positive
5) When click the braid in 3) the corresponding braid graphic should be print on 1)

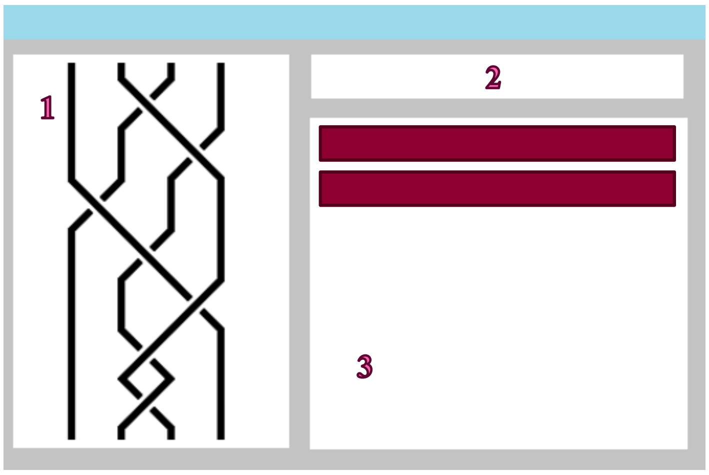


Figure 4.11 The diagram graphic user interface example

5 Realization

5.1 Data Structure Realization

5.1.1 Braid Words Realization

In the program, the two kinds of braid words (σ and Δ) implements an interface called BraidWord, the BraidWord interface is used to create the normal classes of the braid words that should be used in the algorithms. The BraidWord interface support methods about the index of the braid word, the inverse mark the implemented braid word, and some Boolean judgment about the braid word.

BraidWord
getIndex() : int setIndex(int index) : void isInverse() : boolean oppositeInverse() : void isError() : boolean isSigma() : boolean toString() : String

The braid word σ is created and stored in a class, the class implements the interface BraidWord. Besides that, the class Sigma could be created in two forms. For the first type creation, the constructor using a String parameter, the String parameter is in the format of a single of braid word input. The regular expression of the input should be:

[σ][1-9]{1}[0-9]+([\-][1])?

The second type creation is created by an integer parameter as the index of the braid word sigma, and a Boolean parameter as the inverse mark.

Sigma
index : Int isInverse : boolean error : boolean isSigma : boolean sigmaSymbol : String
Sigma() Sigma(int, boolean) Sigma(String) getIndex() : Int setIndex(Int) : void isInverse() : boolean oppositeInverse() : void isError() : boolean isSigma() : boolean toString() : String

The half twist braid word Δ is created and stored in a class, the class implements the interface BraidWord. Besides that, similarity to the Sigma class, the class Delta could be also created in two forms. For the first type creation, the constructor using a String parameter, the String parameter is in the format of a single of half twist braid word input. The regular expression of the input should be:

[Δ][1-9]{1}[0-9]+([\-][1])?

The second type creation is created by an integer parameter as the strand number of the braid which the half twist braid word delta is in, and a Boolean parameter as the inverse mark.

Different form the Sigma class, the HalfTwist class also include the variables and method store, calculate and returns the acutely braid of what the half twist braid is mean.

HalfTwist
index : Int isInverse : boolean halfTwistBraid : Braid

error : boolean isSigma : boolean delteSymbol : String
Delta() Delta(int, boolean) Delta(String) getIndex() : Int setIndex(Int) : void isInverse() : boolean oppositeInverse() : void caculateBraid() : void getBraid() : Braid isError() : boolean isSigma() : boolean toString() : String

5.1.2 Braid Realization

The braid in the program is saved in a class called Braid. The Braid class stored the braid as a java LinkedList with the data type as BraidWord.

The Braid class could be created by several types. Firstly, the braid could be created by input a Braid class, it is used as initialize an input braid as a different braid which enable to change but not influence the previous one. Secondly, the braid could be created by input a linked list of braid words. Thirdly, the created parameter could a braid word array include the braid words needed of the braid. At last, the braid could also be created by a list of actually string braid words. For the input string braid words, the symbol could be “σ” as Sigma and “Δ” as Delta, besides that, the program also enable to input “s” as Sigma and “d” as Delta to make the input cases simpler. The regular expression of the input of the braid should be:

```
([s|σ|d|Δ][1-9]{1}[0-9]*([\-][1])?)+
```

The braid not only store the braid, it could also return the strand number, the half twist exchange rewrite form of the braid. Besides these methods, the braid method supports a very important method, which is to calculated and return the braid rewrite forms in both TreeNode format and LinkedList format.

Braid
braid : LinkedList<BraidWord> strandsNum : Int rewriteTimes : Int

halfTwistFormBraid : Braid hasHalfTwistForm : boolean rewriteTree : TreeNode rewriteTreeList : LinkedList<BraidWord> error : boolean algorithms : Algorithms sigmaSymbol : String deltaSymbol2 : String sigmaSymbol : String deltaSymbol2 : String
Braid() Braid(braid) Braid(LinkedList<BraidWord>) Braid(BraidWord[]) Braid(String) getStrandNum() : Int resetRewriteTiimes() : void setRewriteTimes(Int) : void getRewriteTimes() : Int errorCheck() : void isError() : boolean getBraid() : LinkedList<BraidWord> hasHalfTwistForm() : boolean getHalfTwistFormBraid() : LinkedList<BraidWord> rewriteForms() : TreeNode rewriteFormsList() : LinkedList<BraidWord> rewriteFormsStringList() : String[] buildRewriteTree(TreeNode) : void isInTree(Braid, TreeNode) : boolean buildRewriteTreeList(TreeNode) : void toString : String

It is said that, the Braid class has methods that enables the calculating of the rewrite tree node. The tree node is planned to use the Java JTree, but the JTree did not support the 1 to n tree and the JTree could did not have the methods supports the algorithms. So, an inner class in the Braid class is created to be the structure of the rewrite braids. The inner class TreeNode, has braid as its information and the LinkedList with the data type TreeNode as the child.

TreeNode
braid : Braid childList : LinkedList<TreeNode>

```
TreeNode()
TreeNode(Braid braid)
addChild(TreeNode child) : void
getBraid() : Braid
getChild() : LinkedList<TreeNode>
```

5.2 Algorithms Realization

5.2.1 σ-Positive Realization

The σ-positive braid one of the most important algorithms of all. The program implement σ-positive braid in a while loop, in each iterator the program filter the braid, check whether there is a handle structure in the braid which means found two nearest braid with the smallest index number of the braid which one is in inverse case and another is not. Because of the program needs to get the smallest index, it needs to filter all the elements of the braid in each iterator. The program remembers the location of the braid which is the two side of the handle.

Code of filter the handle in σ-positive Algorithm

```
int countB1 = 0;
int countB2 = -1;
for (int i = 1; i < tmpBraid.getBraid().size(); i++) {
        BraidWord tmpB2 = tmpBraid.getBraid().get(i);
        if (tmpB2.getIndex() < tmpB1.getIndex()) {
                tmpB1 = tmpB2;
                countB1 = i;
                countB2 = -1;
                hasSigmaPositive = false;
        } else if (tmpB2.getIndex() == tmpB1.getIndex()) {
                if ((tmpB1.isInverse() == true && tmpB2.isInverse() == false)
                        || (tmpB1.isInverse() == false && tmpB2.isInverse() == true)) {
                countB2 = i;
                hasSigmaPositive = true;
                break;
                } else {
                        countB1 = i;
                }
        }
}
```

After each time of filter, if the algorithms found a handle structure, the program while remove the handle by exchange the remembered braid as the calculated rewrite and shift the braid words between the two handle braid words (The underline code in the below example of the handle remove is the code of braid shifting).

Code of the handle remove in σ-positive Algorithm

```
int tmpIndexcount = tmpBraid.getStrandsNum();
int tmpIndex = tmpBraid.getStrandsNum() - 1;
for (int j = countB1 + 1; j < countB2; j++) {
      tmpBraid.getBraid().get(j)
            .setIndex(tmpBraid.getBraid().get(j).getIndex() - 1);
}
LinkedList<BraidWord> tmpList1 = new LinkedList<BraidWord>();
LinkedList<BraidWord> tmpList2 = new LinkedList<BraidWord>();
for (int k = tmpB1.getIndex() + 1; k < tmpIndexcount; k++) {
      tmpList1.add(new Sigma(k, !tmpB1.isInverse()));
      tmpList2.add(new Sigma(tmpIndex, tmpB1.isInverse()));
      tmpIndex--;
}
tmpBraid.getBraid().remove(countB2);
tmpBraid.getBraid().addAll(countB2, tmpList2);
tmpBraid.getBraid().remove(countB1);
tmpBraid.getBraid().addAll(countB1, tmpList1);
```

The last step of the σ-positive braid checking after the while loop which means all the handles has been removed from the braid is easily to check whether all the braid with the smallest index is positive.

For the problem that it is not proved that all the braid could be removed the handles by the method and when removing handles from large index or strand number braid might make the program crash, a counter of the rewrite times is used, when the rewrite times is more than 1000 times, the method will automatically stop the loop and return to the program. The times counter could also be used to count and remember the final count of the braid rewrite times.

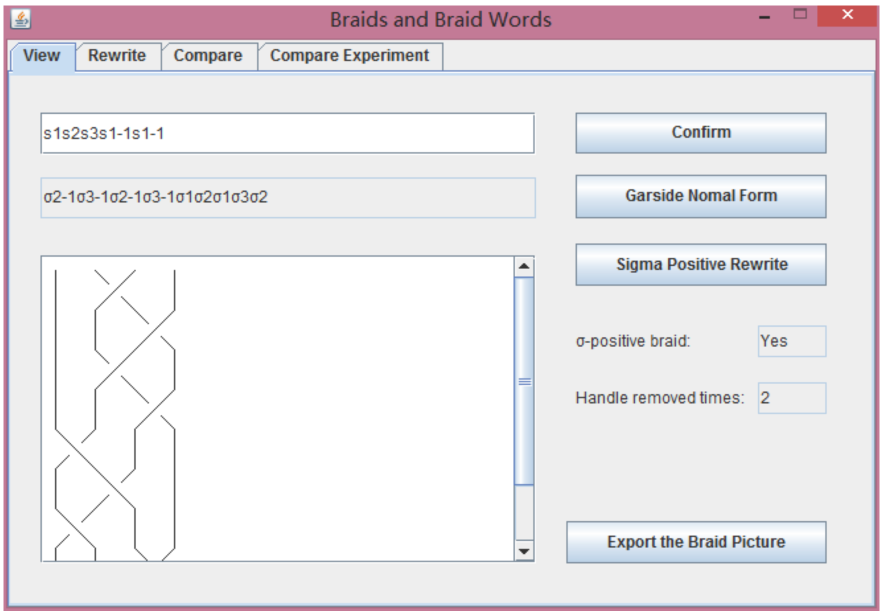
- Testing

  The first example is the braid

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

  The braid should be an σ-positive braid, and it takes twice handle remove, after that the rewrite form should be

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



s1s2s3s1-1s1-1

σ2-1σ3-1σ2-1σ3-1σ1σ2σ1σ3σ2

σ-positive braid: Yes

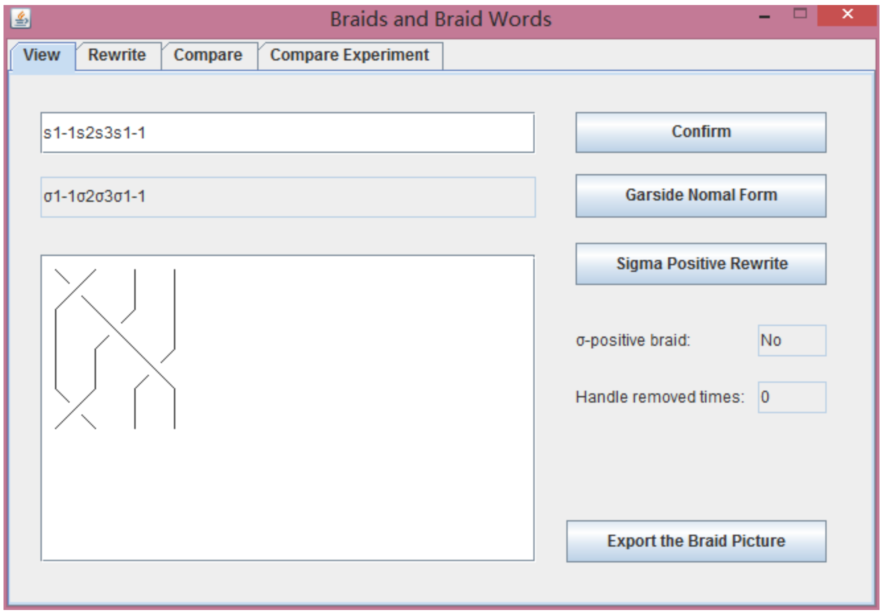
Handle removed times: 2

Figure 5.1 The result of the first example of the σ-positive braid

The second example is the braid

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

The braid should not be an σ-positive braid, did not have handle structure, so it do not need to rewrite.



s1-1s2s3s1-1

σ1-1σ2σ3σ1-1

σ-positive braid: No

Handle removed times: 0

Figure 5.2 The result of the second example of the σ-positive braid

5.2.2 Garside Normal Form Realization

The rewrite of a Garside Normal form braid is another important algorithms of the Braids and Braid Words project. Different from the σ-positive braid algorithms, the Garside Normal form did not need to filter the braid many times, because of the Garside Normal form just need to find out the inverse cases of the braid, it just filter the target braid once. In the design the Garside Normal form filters the braid from the beginning, but when implementing the algorithms, it filters the braid from the end to the beginning. It is because the Garside Normal form method adds braid words into the braid, it is hard to implement for the method.

During the filter, when the program found am inverse case braid word, the first step the program do is to build the rewrite form braid according to the method in the design part. An important variable should be introduced is a Boolean variable called “transForm”. The variable enables the index number of the rewrite form braid rewrite into the strand number minus the previous index number twice the program found the inverse case braid.

Code of building rewrite braid in Garside Mormal Form Algorithm

```
LinkedList<BraidWord> tmpList = inverseDeltaTransform(
        (Sigma) tmpBraid.getBraid().get(i),braid.getStrandsNum());
if (transForm) {
        for (BraidWord b : tmpList) {
                b.setIndex(braid.getStrandsNum() - b.getIndex());
        }
}
```

```
public LinkedList<BraidWord> inverseDeltaTransform(Sigma sigma,
                        int strandsNum) {
        LinkedList<BraidWord> list = new LinkedList<BraidWord>();
        if (!sigma.isInverse()) {
                list.add(sigma);
                return list;
        } else if (sigma.getIndex() >= strandsNum) {
                return null;
        }
        for (int i = 1; i < strandsNum - 1; i++) {
                for (int j = i; j >= 1; j--) {
                        list.add(new Sigma(j, false));
                }
        }
        for (int k = strandsNum - 1; k > sigma.getIndex(); k--) {
                list.add(new Sigma(k, false));
        }
        for (int l = 1; l < sigma.getIndex(); l++) {
                list.add(0, new Sigma(strandsNum - l, false));
        }
        return list;
}
```

After the calculation of the rewrite from, the program exchange the inverse braid word into the rewrite form. Then the program rewrite all the index number of the braid before and marked inverse braid into the total strand number minus the braid original index number.

Code of braid index exchange in Garside Mormal Form Algorithm

```
for (int j = 0; j < i; j++) {
        if (!tmpBraid.getBraid().get(j).isInverse()) {
                tmpBraid.getBraid().get(j).setIndex(braid.getStrandsNum()
                        - tmpBraid.getBraid().get(j).getIndex());
        }
}
```

The final step of the Garside normal form is to add the deltas at the beginning of the braid with the number of the braid word inverse cases that has been found in the braid.

Code of add deltas in the beginning in Garside Mormal Form Algorithm

```
for (int k = 0; k < count; k++) {
        tmpBraid.getBraid().add(0, new Delta(braid.getStrandsNum(), true));
}
```

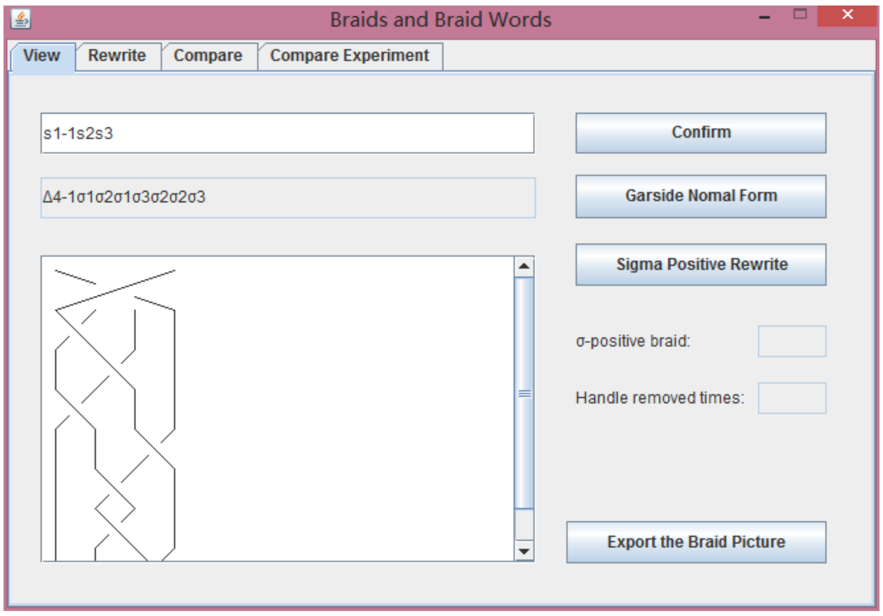
- Testing

The first example is the braid

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

The rewrite form should be

$$\Delta^{-1}\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2\sigma_2\sigma_3$$



s1-1s2s3

Δ4-1σ1σ2σ1σ3σ2σ2σ3

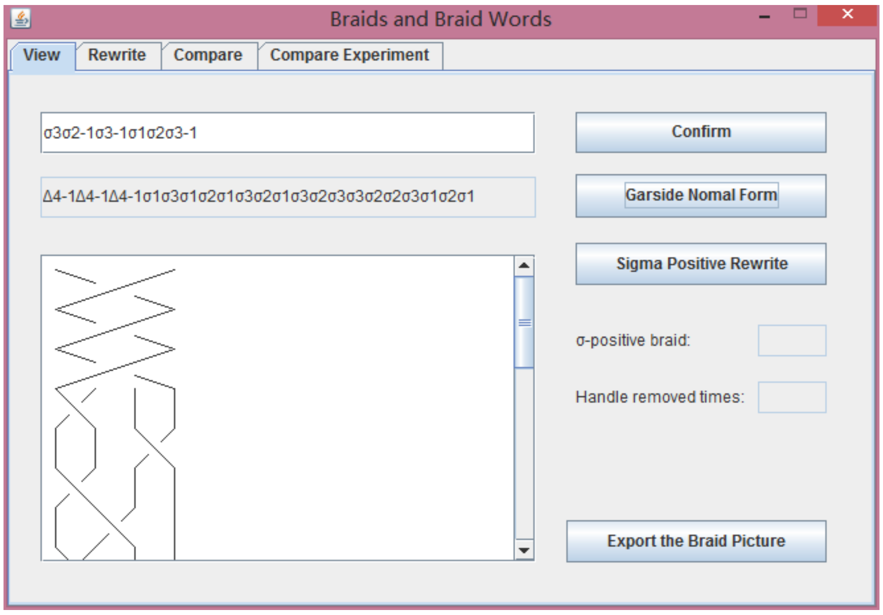
Figure 5.3 The result of the first example of the Garisde Normal form

The second example is the braid

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

The rewrite form should be

$$\Delta^{-3}\sigma_1\sigma_3\sigma_2\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\sigma_3\sigma_2\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1$$



σ3σ2-1σ3-1σ1σ2σ3-1

Δ4-1Δ4-1Δ4-1σ1σ3σ1σ2σ1σ3σ2σ1σ3σ2σ3σ3σ2σ2σ3σ1σ2σ1

Figure 5.4 The result of the second example of the Garisde Normal form

5.2.3 Braid Rewrite Realization

For the braid rewrite algorithm, the program solve that by reclusion, in each iteration of the program, the first step is to filter the braid.

Once the program found the rewrite rule:

$$\sigma_i\sigma_j = \sigma_j\sigma_i(|i-j| > 1)$$

Two braid word with the difference between two index number is greater or equal to 2, it will created the new braid which exchange the two braid place.

Code of rewrite case 1 in Braid Rewrite Algorithm

```
if (Math.abs(b.getBraid().get(i).getIndex()
        - b.getBraid().get(i - 1).getIndex()) >= 2) {
        BraidWord tmp = b.getBraid().get(i);
        b.getBraid().remove(i);
        b.getBraid().add(i - 1, tmp);
        if (!isInTree(b, rewriteTree)) {
                node.addChild(new TreeNode(new Braid(b)));
        }
        b = new Braid(node.getBraid());
}
```

The second rewrite form:

$$\sigma_i{\sigma_i}^{-1} = e$$

It will be founded if the braid has two nearby braid word with the same index number and the difference inverse mark, which means that one of the braid is in inverse case and the other is in positive case. The rewrite case of the braid is simply remove from the braid which means that connect two that kind of braid could be canceled.

Code of rewrite case 2 in Braid Rewrite Algorithm

```
if (((b.getBraid().get(i).getIndex() == b.getBraid().get(i - 1)
        .getIndex()) && (b.getBraid().size() != 2))
                && (b.getBraid().get(i).isInverse() != b.getBraid()
                        .get(i - 1).isInverse())) {
        b.getBraid().remove(i);
        b.getBraid().remove(i - 1);
        if (!isInTree(b, rewriteTree)) {
                node.addChild(new TreeNode(new Braid(b)));
        }
        b = new Braid(node.getBraid());
}
```

The last rewriting case:

$$\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}(i = 1,2,\cdots,n-2)$$

The program could only check three braid words then judgment whether the braid obey the rewrite rule. If the program found the three nearby braid word index number cases i, i + 1, i, or i + i, i, i + 1. The program create the new braid with exchange $\sigma_i\sigma_{i+1}\sigma_i$ to $\sigma_{i+1}\sigma_i\sigma_{i+1}$ or from $\sigma_{i+1}\sigma_i\sigma_{i+1}$ to $\sigma_i\sigma_{i+1}\sigma_i$.

Code of rewrite case 3 in Braid Rewrite Algorithm

```
if ((b.getBraid().get(i).isInverse() == b.getBraid().get(i - 1).isInverse())
        && (b.getBraid().get(i - 1).isInverse() == b.getBraid()
                .get(i - 2).isInverse())) {
        if (b.getBraid().get(i - 2).getIndex() == b.getBraid()
                        .get(i).getIndex()) {
                if (Math.abs(b.getBraid().get(i).getIndex()
                        - b.getBraid().get(i - 1).getIndex()) == 1) {
                        BraidWord tmp = b.getBraid().get(i - 1);
                        b.getBraid().remove(i - 2);
                        b.getBraid().add(i, tmp);
                        if (!isInTree(b, rewriteTree)) {
                                node.addChild(new TreeNode(new Braid(b)));
                        }
                        b = new Braid(node.getBraid());
                }
        }
}
```

Every time found a braid, the program will check whether the rewrite form braid is already in the TreeNode of the rewrite braids. Only if the rewrite form is not in the TreeNode of the braid, then the program will start the new reclusion with the braid as the input.

Code of exist in TreeNode check in Braid Rewrite Algorithm

```
private boolean isInTree(Braid braid, TreeNode node) {
        if (algorithms.isSame(braid, node.getBraid())) {
                return true;
        } else if (node.getChild().size() == 0) {
                return false;
        }
        for (TreeNode t : node.getChild()) {
                if (isInTree(braid, t)) {
                        return true;
                }
        }
        return false;
}
```

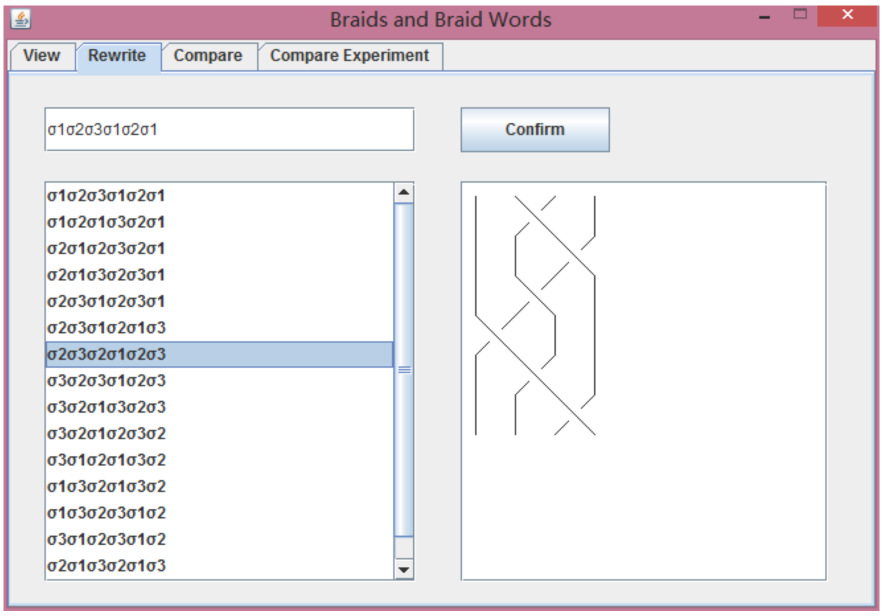
◆ Testing

The example is the braid

$$\sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1$$

The rewrite forms should be

$$\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1,\ \sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\sigma_2,\ \sigma_2\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1,\ \sigma_1\sigma_3\sigma_2\sigma_3\sigma_1\sigma_2,$$
$$\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3\sigma_1,\ \sigma_3\sigma_1\sigma_2\sigma_3\sigma_1\sigma_2,\ \sigma_1\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2,\ \sigma_2\sigma_3\sigma_1\sigma_2\sigma_3\sigma_1,$$
$$\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1\sigma_3,\ \sigma_3\sigma_1\sigma_2\sigma_1\sigma_3\sigma_2,\ \sigma_2\sigma_3\sigma_1\sigma_2\sigma_1\sigma_3,\ \sigma_3\sigma_2\sigma_1\sigma_2\sigma_3\sigma_2,$$
$$\sigma_2\sigma_3\sigma_2\sigma_1\sigma_2\sigma_3,\ \sigma_3\sigma_2\sigma_1\sigma_3\sigma_2\sigma_3,\ \sigma_3\sigma_2\sigma_3\sigma_1\sigma_2\sigma_3$$



```
σ1σ2σ3σ1σ2σ1
```

```
σ1σ2σ3σ1σ2σ1
σ1σ2σ1σ3σ2σ1
σ2σ1σ2σ3σ2σ1
σ2σ1σ3σ2σ3σ1
σ2σ3σ1σ2σ3σ1
σ2σ3σ1σ2σ1σ3
σ2σ3σ2σ1σ2σ3
σ3σ2σ3σ1σ2σ3
σ3σ2σ1σ3σ2σ3
σ3σ2σ1σ2σ3σ2
σ3σ1σ2σ1σ3σ2
σ1σ3σ2σ1σ3σ2
σ1σ3σ2σ3σ1σ2
σ3σ1σ2σ3σ1σ2
σ2σ1σ3σ2σ1σ3
```

Figure 5.5 The result of the example of the rewrite algorithms

5.2.4 Braid Ordering Realization

The braid ordering algorithm is the most important algorithm in the project. To apply the algorithm the program also used the methods of the σ-positive algorithm.

For the first step of the braid ordering, the program calculate the inverse case braid of the first braid need to compare. In the method, he program get the braid words from the end of the braid, and put it into a new braid from the beginning with its inverse case.

Code of create inverse braid in Braid Ordering Algorithm

```
public Braid inverseBraid(Braid braid) {
    LinkedList<BraidWord> list = new LinkedList<BraidWord>();
    for (int i = 0; i < braid.getBraid().size(); i++) {
        if (braid.getBraid().get(i).isSigma()) {
            list.add(0, new Sigma(braid.getBraid().get(i).getIndex(),
                    !braid.getBraid().get(i).isInverse()));
        } else {
            list.add(0, new Delta(braid.getBraid().get(i).getIndex(),
                    !braid.getBraid().get(i).isInverse()));
        }
    }
    Braid tmpBraid = new Braid(list);
    return tmpBraid;
}
```

When the program get the inverse case braid, it connect the braid with the second braid, and check whether the new braid is an σ-positive braid. If the answer is true, then the first is smaller.

Code of main program in Braid Ordering Algorithm

```
public boolean isSmaller(Braid braid1, Braid braid2) {
        LinkedList<BraidWord> list = new LinkedList<BraidWord>();
        list.addAll(new Braid(inverseBraid(braid1)).getBraid());
        list.addAll(braid2.getBraid());
        Braid tmpBraid = new Braid(list);
        if (isSigmaPositive(tmpBraid)) {
            return true;
        } else {
            return false;
        }
}
```

The java program could not return none string value that shoes all "smaller", "greater" and "same" at the same time. And it is have cases that the two comparing braid is the same braid after rewrite, which mean the two braids are equal or the connected new braid in the previous steps is an empty braid after handle removing. Because of this reason, the program do the double check with the opposite order of the braid to check whether the two braids are equal or not.

Code of equal check of the two braids in Braid Ordering Algorithm

```
public boolean isEqual(Braid braid1, Braid braid2) {
        if (!isSmaller(braid1, braid2) && !isSmaller(braid2, braid1)) {
            return true;
        } else {
            return false;
        }
}
```

- Testing

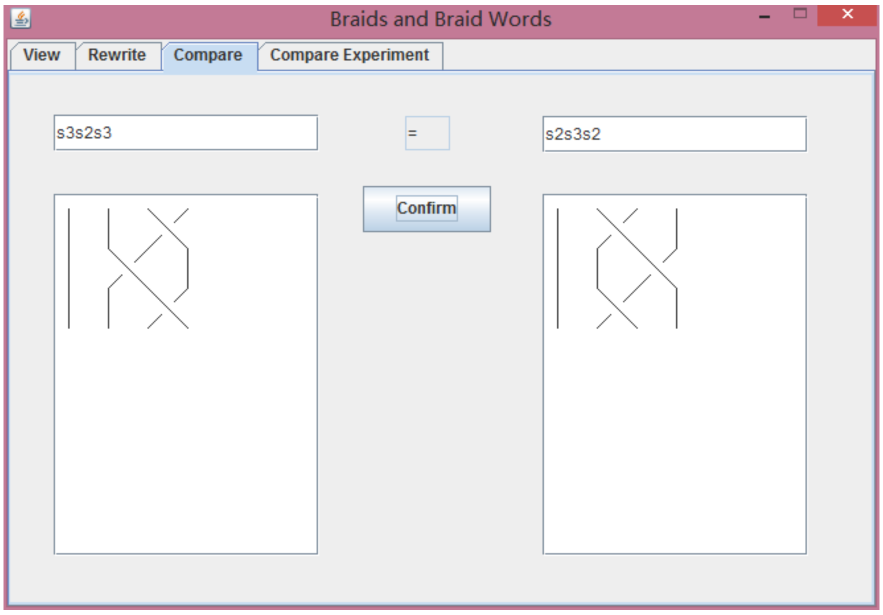
The first example is to compare braid

$$\sigma_3\sigma_2\sigma_3$$

And the braid

$$\sigma_2\sigma_3\sigma_2$$

The result should be the two braid are equal



s3s2s3

=

s2s3s2

Figure 5.6 The result of the first example of the braid comparing

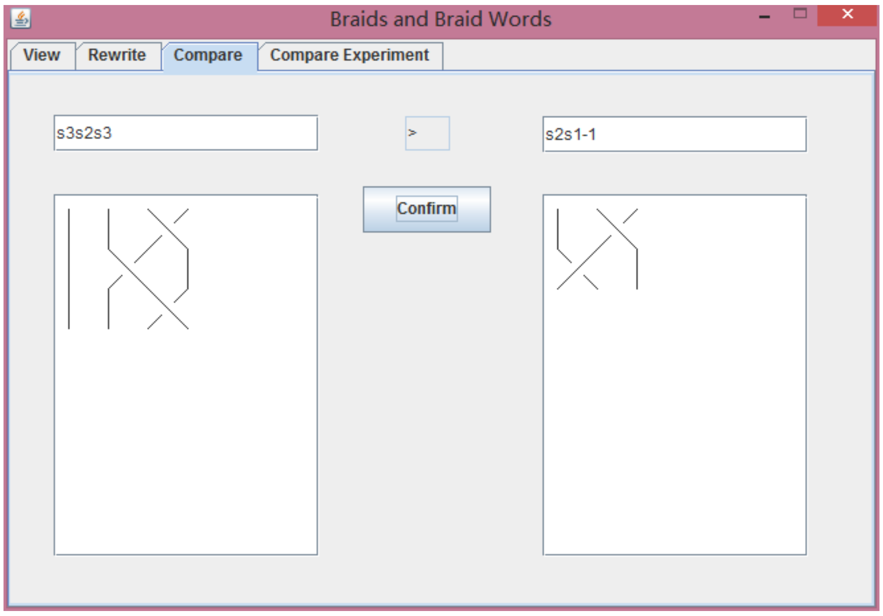
The second example is to compare braid

$$\sigma_3\sigma_2\sigma_3$$

And the braid

$$\sigma_2{\sigma_1}^{-1}$$

The result should be the braid $\sigma_3\sigma_2\sigma_3$ is larger the braid $\sigma_2{\sigma_1}^{-1}$.



s3s2s3

>

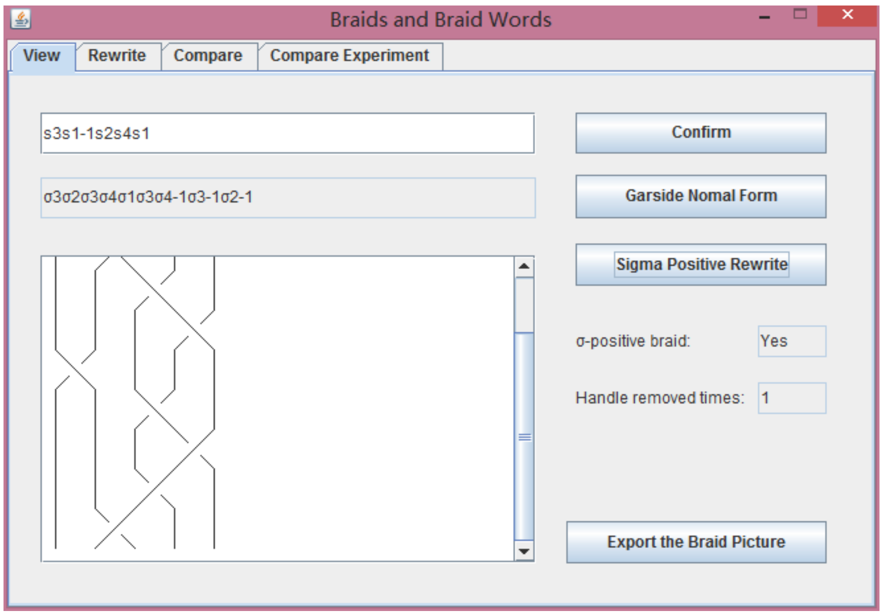
s2s1-1

Figure 5.7 The result of the second example of the braid comparing

5.3 GUI Realization

The design is only a part that simply show what the graphic user interface will include. Here is what the GUI is showing in the program. The graphic user interface is divided into four main parts.

Firstly, the program allows user to input a braid, the program will show the braid picture of the braid, besides that, this part of the interface support the interface that the user could get the picture and rewrite form of the σ-positive braid and the Garside normal form of the input braid. For the σ-positive braid input, the program also shows whether the braid is an σ-positive braid and the handle remove times of the braid when doing the rewrite. The view part of the graphic user interface also could output the braid as image file (.jpg) into the root folder.



s3s1-1s2s4s1

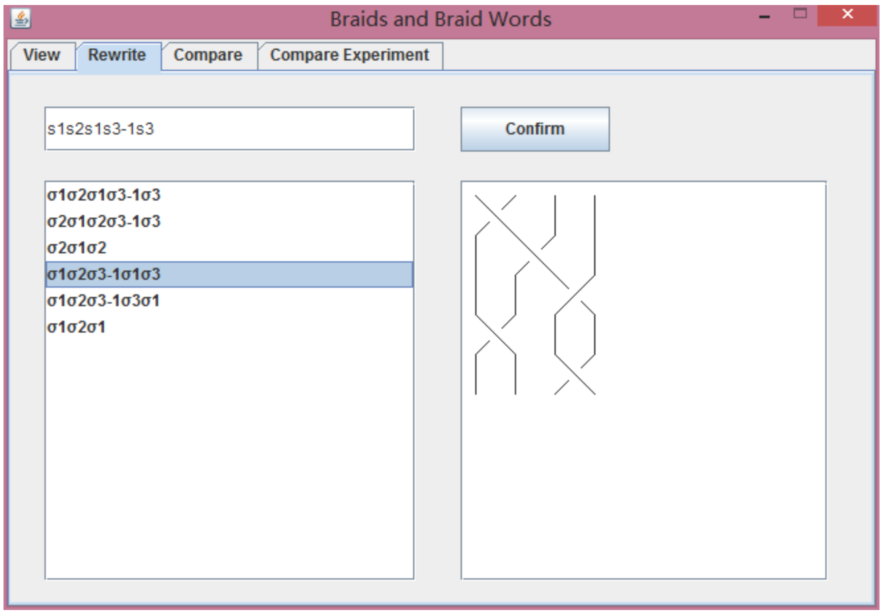
σ3σ2σ3σ4σ1σ3σ4-1σ3-1σ2-1

σ-positive braid: Yes

Handle removed times: 1

Figure 5.8 The screen shot of the braid view part

Secondly, the program support the interface for the braid rewrite algorithm. The interface allows the user to import the braid in the text field, then the program listed all the rewrite forms below, it also allows user to choose the rewrite form braid and show the braid picture on the right.



s1s2s1s3-1s3

σ1σ2σ1σ3-1σ3
σ2σ1σ2σ3-1σ3
σ2σ1σ2
σ1σ2σ3-1σ1σ3
σ1σ2σ3-1σ3σ1
σ1σ2σ1

Figure 5.9 The screen shot of the braid rewrite part

Thirdly, the program applies the braid comparing. The user could input two braids wish to compare on both sides of the braid, and both of the braid picture is shown on the below, the program gives the comparing result to the two braids in the middle as well.

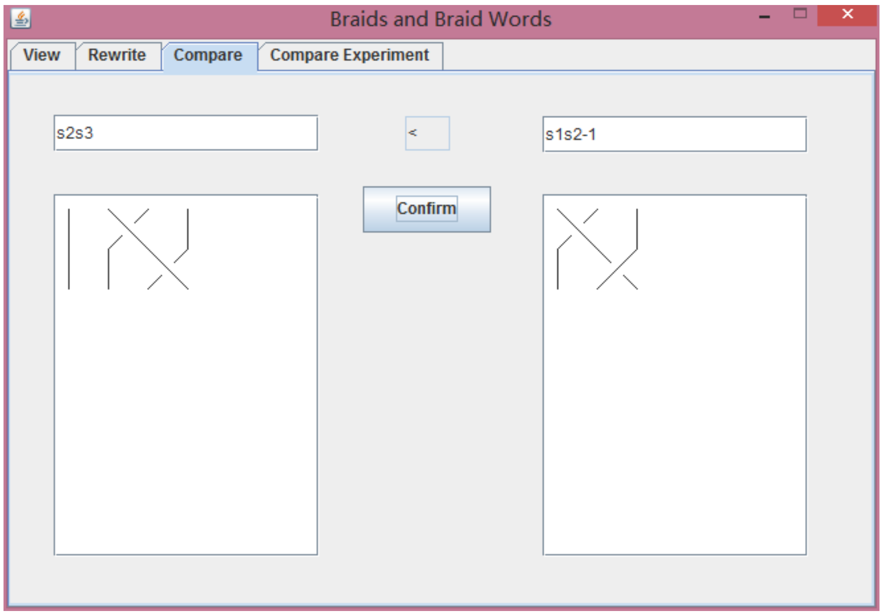
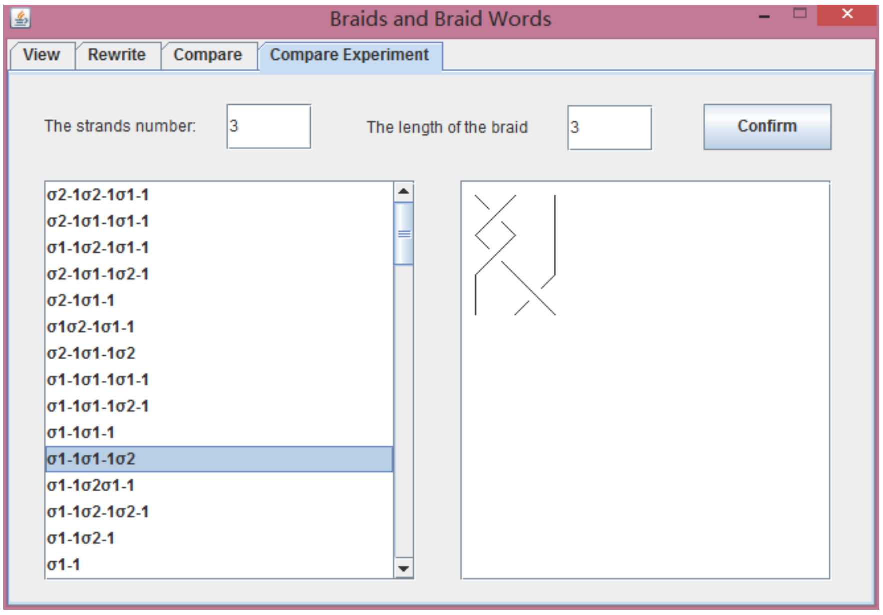


Figure 5.10 The screen shot of the braid rewrite part

Fourthly, the program supports an interface to support the experiment of braid ordering.



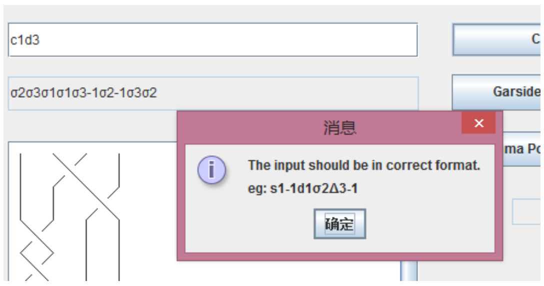
σ2-1σ2-1σ1-1
σ2-1σ1-1σ1-1
σ1-1σ2-1σ1-1
σ2-1σ1-1σ2-1
σ2-1σ1-1
σ1σ2-1σ1-1
σ2-1σ1-1σ2
σ1-1σ1-1σ1-1
σ1-1σ1-1σ2-1
σ1-1σ1-1
σ1-1σ1-1σ2
σ1-1σ2σ1-1
σ1-1σ2-1σ2-1
σ1-1σ2-1
σ1-1

Figure 5.11 The screen shot of the braid ordering example graphic user inter face

The program has input control of all braids need to input. The regular expression of the braid input is:

([s|σ|d|Δ][1-9]{1}[0-9]*([\-][1])?)+

The braid word "s" means "σ", "d" means "Δ", this enables the braid coul be input easily. And if the input is not the correct form input control regular expression, the program will gives a message that informs the user that the input braid is not in correct format..


c1d3

σ2σ3σ1σ1σ3-1σ2-1σ3σ2

消息

The input should be in correct format.
eg: s1-1d1σ2Δ3-1

Figure 5.11 Example of input control

The graphic user interface (GUI) is created by the Eclipse java plugin Jigloo. The graphic user interface components are created by Jigloo, and then the actions and the braid views are created after that.

6 Experiments

6.1 Braid Ordering Experiment

The project is aimed to build a program support the braid ordering, so an experiment of the braid ordering is positive to the program. The aim of the experiment is to find out some rules of the equivalence among the braid and whether the strands number of the braid or the braid index of the braid influence the braid ordering.

To support the braid ordering experiment, the program creates a new method that ordered all the input braids in a list. The method ordered the methods by quick sort, which is an efficient sort method. In each iterator of the quick sort method, the program put all the smaller braids on the left of the target braid, and the greater braids on the right, then start new braid sorting in both sides.

Code of the quick sort of the experiment

```
vpublic void sortBraids(LinkedList<Braid> braids,
            LinkedList<Braid> finalBraid) {
      if (braids.size() == 0) {
            return;
      } else if (braids.size() == 1) {
            finalBraid.add(braids.getFirst());
      } else {
            LinkedList<Braid> left = new LinkedList<Braid>();
            LinkedList<Braid> mid = new LinkedList<Braid>();
            LinkedList<Braid> right = new LinkedList<Braid>();
            Braid thisBraid = braids.getFirst();
            for (Braid b : braids) {
                if (isSmaller(b, thisBraid)) {
                    left.add(b);
                } else if (isSmaller(thisBraid, b)) {
                    right.add(b);
                } else if (isEqual(b, thisBraid)) {
                    mid.add(b);
                }
            }
            sortBraids(left, finalBraid);
            finalBraid.addAll(mid);
            sortBraids(right, finalBraid);
        }
}
```

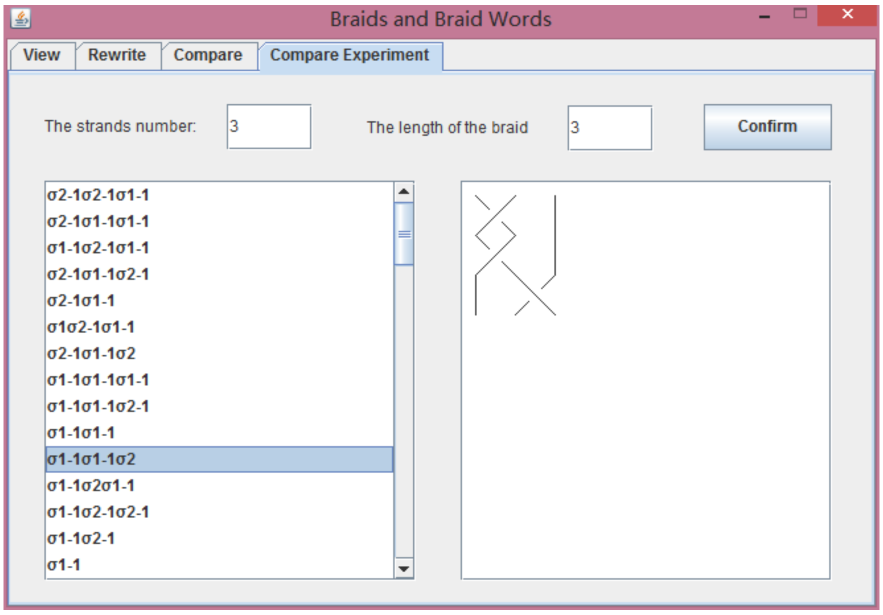
The ordering of the braid orders all the possible braids of the given strand number and the braid length. A method is also used to create such kind braid list.

Code of the create the braid by strand number and braid length in the experiment

```
public LinkedList<Braid> createBraidList(int strandNum, int braidLength) {
    LinkedList<Braid> tmpList = new LinkedList<Braid>();
    String[][] tmpString = new String[braidLength][];
    tmpString[0] = new String[(strandNum - 1) * 2];
    for (int i = 1; i <= (strandNum - 1); i++) {
        tmpString[0][i * 2 - 2] = "s" + i;
        tmpString[0][i * 2 - 1] = "s" + i + "-1";
    }
    for (int i = 2; i <= braidLength; i++) {
        tmpString[i - 1] = new String[tmpString[i - 2].length
                * (strandNum - 1) * 2];
        for (int j = 0; j < tmpString[i - 2].length; j++) {
            for (int k = 1; k <= (strandNum - 1); k++) {
                tmpString[i - 1][k * 2 - 2 + (strandNum - 1) * 2 * j]
                    = tmpString[i - 2][j] + "s" + k;
                tmpString[i - 1][k * 2 - 1 + (strandNum - 1) * 2 * j]
                    = tmpString[i - 2][j] + "s" + k + "-1";
            }
        }
    }
    for (String[] s : tmpString) {
        for (String ss : s) {
```

```
                        tmpList.add(new Braid(ss));
                }
        }
        return tmpList;
}
```

And in the graphic user interface, a part of the interface is added in to the program to support the experiment.



The strands number: 3

The length of the braid 3

σ2-1σ2-1σ1-1
σ2-1σ1-1σ1-1
σ1-1σ2-1σ1-1
σ2-1σ1-1σ2-1
σ2-1σ1-1
σ1σ2-1σ1-1
σ2-1σ1-1σ2
σ1-1σ1-1σ1-1
σ1-1σ1-1σ2-1
σ1-1σ1-1
σ1-1σ1-1σ2
σ1-1σ2σ1-1
σ1-1σ2-1σ2-1
σ1-1σ2-1
σ1-1

Figure 6.1 The screen shot of the braid ordering example graphic user inter face

The first testing braid list is with the braid strand number of 2 and maximum braid length of 2. It is easy to get that,

$$\sigma_1{}^{-1}\sigma_1{}^{-1} < \sigma_1{}^{-1} < \sigma_1\sigma_1{}^{-1} = \sigma_1{}^{-1}\sigma_1 < \sigma_1 < \sigma_1\sigma_1$$

The second testing braid list is with the braid strand number of 2 and maximum braid length of 3. It is could get that,

$$\sigma_1{}^{-1}\sigma_1{}^{-1}\sigma_1{}^{-1} < \sigma_1{}^{-1}\sigma_1{}^{-1} < \sigma_1{}^{-1} = \sigma_1\sigma_1{}^{-1}\sigma_1{}^{-1} = \sigma_1{}^{-1}\sigma_1\sigma_1{}^{-1} = \sigma_1{}^{-1}\sigma_1{}^{-1}\sigma_1 < \sigma_1\sigma_1{}^{-1} = \sigma_1{}^{-1}\sigma_1 < \sigma_1{}^{-1}\sigma_1\sigma_1 = \sigma_1\sigma_1{}^{-1}\sigma_1 = \sigma_1\sigma_1\sigma_1{}^{-1} = \sigma_1 < \sigma_1\sigma_1 < \sigma_1\sigma_1\sigma_1$$

For the two examples above when remove the equivalent braids, then it could get that,

$$\sigma_1{}^{-1}\sigma_1{}^{-1}\sigma_1{}^{-1} < \sigma_1{}^{-1}\sigma_1{}^{-1} < \sigma_1{}^{-1} < e < \sigma_1 < \sigma_1\sigma_1 < \sigma_1\sigma_1\sigma_1$$

And it is easy to see that, the all positive braids are greater than all positive braids, and if the index number of the braid is same, the positive braid with the more power should be greater and the inverse braid with more power should be smaller.

$$(\sigma_i{}^{-1})^n < \ldots < \sigma_i{}^{-1}\sigma_i{}^{-1} < \sigma_i{}^{-1} < e < \sigma_i < \sigma_i\sigma_i < \ldots < (\sigma_i)^n$$

The third testing braid list is with the braid strand number of 3 and maximum braid length of 2. It is could get that,

$$\sigma_2{}^{-1}\sigma_1{}^{-1} < \sigma_1{}^{-1}\sigma_1{}^{-1} < \sigma_1{}^{-1}\sigma_2{}^{-1} < \sigma_1{}^{-1} < \sigma_1{}^{-1}\sigma_2 < \sigma_2\sigma_1{}^{-1} < \sigma_2{}^{-1}\sigma_2{}^{-1} < \sigma_2{}^{-1} < \sigma_1\sigma_1{}^{-1} = \sigma_1{}^{-1}\sigma_1 = \sigma_2\sigma_2{}^{-1} = \sigma_2{}^{-1}\sigma_2 < \sigma_2 < \sigma_2\sigma_2 < \sigma_2{}^{-1}\sigma_1 < \sigma_1\sigma_2{}^{-1} < \sigma_1 < \sigma_1\sigma_2 < \sigma_1\sigma_1 < \sigma_2\sigma_1$$

There is not a very clear rule among the equivalence of the braids, it could still conclude that,

$$\ldots < (\sigma_i{}^{-1})^n < \ldots < \sigma_i{}^{-1}\sigma_i{}^{-1} < \sigma_i{}^{-1} < (\sigma_{i+1}{}^{-1})^n < \ldots < \sigma_{i+1}{}^{-1}\sigma_{i+1}{}^{-1} < \sigma_{i+1}{}^{-1} < e < \sigma_{i+1} < \sigma_{i+1}\sigma_{i+1} < \ldots < (\sigma_{i+1})^n < \sigma_i < \sigma_i\sigma_i < \ldots < (\sigma_i)^n < \ldots$$

The experiment found some rules when ordering braids with the same braid word index and are all inverse or all positive. If the index number is different, the all inverse case braid with the less index number will be smaller, and the all positive case braid with the less index number will be greater. And if the index number is the same, as it is said before the positive braid with the more power should be greater and the inverse braid with more power should be smaller.

For the other format of braids, there is not a clear rule that could easily compare each other just according to the braid words.

6.2 σ-Positive Rewrite Times Experiment

The σ-positive rewrite experiment is based on the braid ordering experiment. The σ-positive rewrite experiment is analyzing the braid handle remove times when the braids compares each other.

For example to compare braid $\sigma_2{}^{-1}\sigma_1{}^{-1}$ and $\sigma_1{}^{-1}$

These two braids will rewrite and connect together and become a new braid

$$\sigma_1\sigma_2\sigma_1{}^{-1}$$

The next step of is the handle removing and check whether the braid is σ-positive. The experiment here is to get the handle remove times. For the braid $\sigma_2{}^{-1}\sigma_1{}^{-1}$ and $\sigma_1{}^{-1}$ the handle remove time should be 1.

The testing braids arewith the braid strand number of 3 and maximum braid length of 2. It is could get that,

	$\sigma_2^{-1}\sigma_1^{-1}$	$\sigma_1^{-1}\sigma_1^{-1}$	$\sigma_1^{-1}\sigma_2^{-1}$	σ_1^{-1}	$\sigma_1^{-1}\sigma_2$	$\sigma_2\sigma_1^{-1}$	$\sigma_2^{-1}\sigma_2^{-1}$	σ_2^{-1}	$\sigma_1\sigma_1^{-1}$	$\sigma_1^{-1}\sigma_1$	$\sigma_2\sigma_2^{-1}$	$\sigma_2^{-1}\sigma_2$	σ_2	$\sigma_2\sigma_2$	$\sigma_2^{-1}\sigma_1$	$\sigma_1\sigma_2^{-1}$	σ_1	$\sigma_1\sigma_2$	$\sigma_1\sigma_1$	$\sigma_2\sigma_1$
$\sigma_2^{-1}\sigma_1^{-1}$	3	2	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
$\sigma_1^{-1}\sigma_1^{-1}$	2	2	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
$\sigma_1^{-1}\sigma_2^{-1}$	1	1	3	1	2	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
σ_1^{-1}	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
$\sigma_1^{-1}\sigma_2$	1	1	2	1	3	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
$\sigma_2\sigma_1^{-1}$	1	1	1	1	1	3	0	0	1	1	0	0	0	0	0	0	0	0	0	0
$\sigma_2^{-1}\sigma_2^{-1}$	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
σ_2^{-1}	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
$\sigma_1\sigma_1^{-1}$	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1
$\sigma_1^{-1}\sigma_1$	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1	1	1	1
$\sigma_2\sigma_2^{-1}$	0	0	0	0	0	0	1	1	2	2	2	2	1	1	0	0	0	0	0	0
$\sigma_2^{-1}\sigma_2$	0	0	0	0	0	0	1	1	2	2	2	2	1	1	0	0	0	0	0	0
σ_2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
$\sigma_2\sigma_2$	0	0	0	0	0	0	0	0	1	1	1	1	1	2	0	0	0	0	0	0
$\sigma_2^{-1}\sigma_1$	0	0	0	0	0	0	0	0	1	1	0	0	0	0	3	1	1	1	1	1
$\sigma_1\sigma_2^{-1}$	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	2	1	1	1	1
σ_1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1
$\sigma_1\sigma_2$	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	2	1	1
$\sigma_1\sigma_1$	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	2	2
$\sigma_2\sigma_1$	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	2	3

Figure 6.2 The table result of the σ-positive rewrite experiment

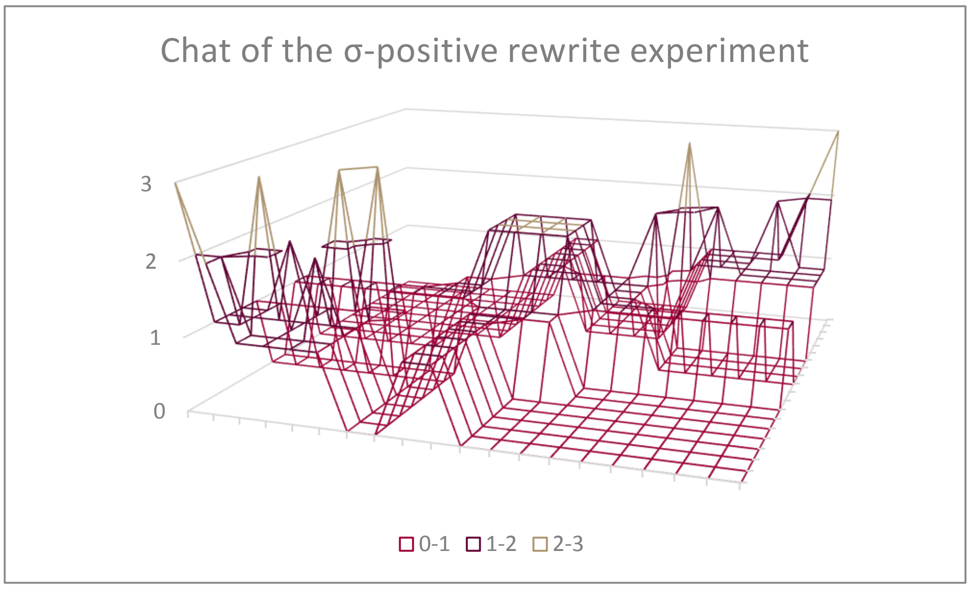


Figure 6.3 The chat result 1 of the σ-positive rewrite experiment

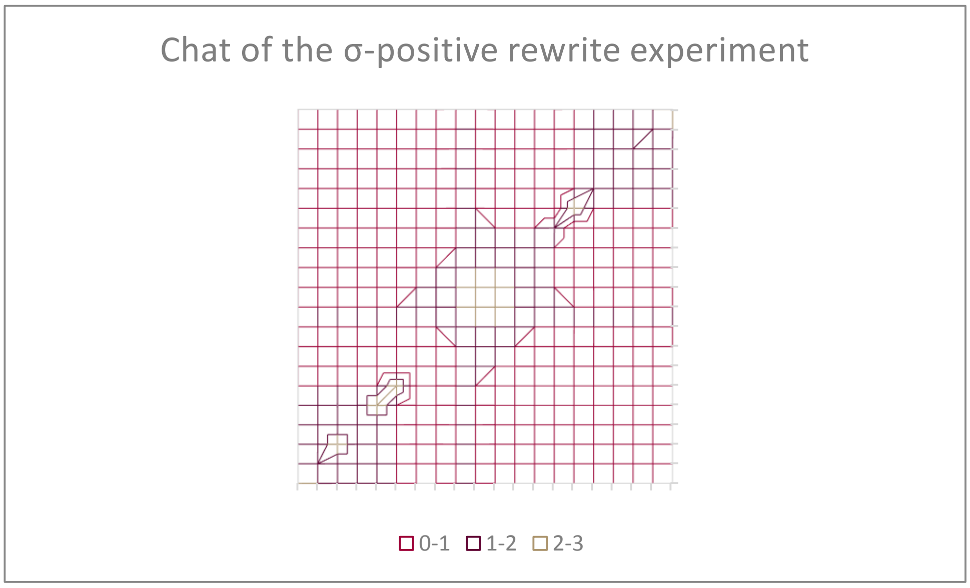


Figure 6.3 The chat result 2 of the σ-positive rewrite experiment

From the result of the experiment, it could find that, the handle remove did not take too much, most of the cases in the braid ordering did not need to do the handle remove, only few braids need to do a large number of handle remove.

7 Evaluation

7.1 Evaluation of Program

- Evaluation of Data Structure

For Searching among a braid b_n, the big-O is $O(n)$

To compare the equivalence between b and $b'(b, b' \in B_n)$, the big-O is $O(n)$

To add or remove some braid word into or from the braid, the big-O is $O(1)$

- Evaluation of σ-positive Algorithm

Let the input braid

$$b_n = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_1} \cdots \sigma_{i_m}{}^{j_m}\ (i_1, i_2, \cdots, i_m \leq n-1, b_n \in B_n\ )$$

Than for each iteration, to find a handle the time complexity of calculate times should be *m*, and the length of b_n is increased *2(n-1)* once.

So, for *k* iterations, the time complexity of calculate times should be

$$\sum_{i=0}^{k} m + 2(n-1)i = \mathrm{km} + (\mathrm{n}-1)\mathrm{k}(\mathrm{k}+1)$$

➢ Evaluation of Garside Normal Form Algorithm

Let the input braid

$$b_n = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_1}\cdots\sigma_{i_m}{}^{j_m}\ (i_1, i_2, \cdots, i_m \leq n-1, b_n \in B_n\ )$$

For there is one inverse case $\sigma_{i_x}{}^{j_x}$ in b_n, the time complexity of calculate times should be

$$\mathrm{m} + x$$

For there is the worth case the time complexity of calculate times should be

$$\mathrm{m} + \frac{m(1+m)}{2}$$

➢ Evaluation of Rewrite Algorithm

Let the input braid

$$b_n = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_1}\cdots\sigma_{i_m}{}^{j_m}\ (i_1, i_2, \cdots, i_m \leq n-1, b_n \in B_n\ )$$

For found out all the rewrite possibilities of every single braid the time complexity of calculate times is *m* just the size of the braid. And for rewrite, it is just reconnect the link list. So, if b_n has totally *x* rewriting possibilities, than the time complexity of calculate times is *xm*

➢ Evaluation of Braid Ordering Algorithm

Let the input braid

$$\mathrm{b} = \sigma_{i_1}{}^{j_1}\sigma_{i_2}{}^{j_1}\cdots\sigma_{i_m}{}^{j_m}\ (i_1, i_2, \cdots, i_m \leq n-1, \mathrm{b} \in B_n\ )$$

$$\mathrm{b}' = \sigma_{i'_1}{}^{j'_1}\sigma_{i'_2}{}^{j'_1}\cdots\sigma_{i'_{m'}}{}^{j'_{m'}}\ (i'_1, i'_2, \cdots, i'_{m'} \leq n-1, \mathrm{b}' \in B_n)$$

The time complexity of calculate times of changing b to b^{-1} is *m*

It doesn’t take any time complexity when linking b and b'

And the checking of whether $\mathrm{b}^{-1}\mathrm{b}'$ is σ-positive, is based on the σ-positive algorithm, so the time complexity of calculate times should be

$$\mathrm{k}(\mathrm{m}+\mathrm{m}') + (\mathrm{n}-1)\mathrm{k}(\mathrm{k}+1)$$

So the time complexity of calculate times of the whole braid ordering algorithm is

$$\mathrm{m} + \mathrm{k}(\mathrm{m}+\mathrm{m}') + (\mathrm{n}-1)\mathrm{k}(\mathrm{k}+1)$$

7.2 Project strength & weakness

➢ Strength

The program supports all the algorithms of the braids and braid words project. And the program has further functions that enables and convenient the user to use the software.

➢ Weakness

Some methods of the program, such as the Garside Normal form and σ-positive algorithm, might crash the whole program, because the large size of the braid input or the unproved algorithms of handle removing in the σ-positive algorithm. The other weakness is that, the method is not so efficient that the program will be a little slow then enter a larger size braid.

7.3 Further Work

There are several possible further work might be implement

1) To undo the third rewrite rule

$$\sigma_i {\sigma_i}^{-1} = e$$

According the third rewrite rule, the when connect a braid word and its inverse case together, it could be canceled. Because of that, it could be get such connected braid words could be add into any location of the braid word, and it will cause the rewrite forms much more changeable.

2) To implement a more efficient handle remove method in σ-positive algorithm

For the σ-positive Algorithm in the design is to move the handle to the most right of the braid, it is too complex. A better method is to move one strand to the right of the previous location, it is proved to be the lest complex handle remove method

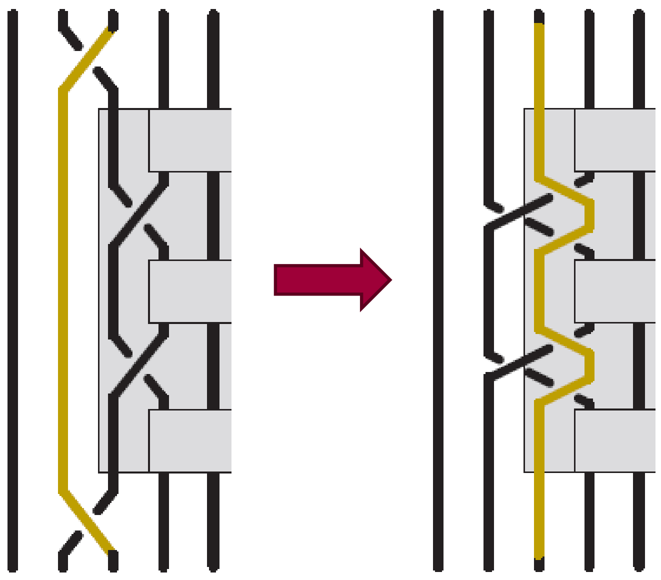


Figure 7.1 The transfrom of reducing a handle

8 Learning Points

The building of the Braids and Braid Words program enables many knowledge. And many things are learned from the program.

Firstly, for the simplest, the knowledge of the braids could be learned from the Braids and Braid Words program. By this program, the information and knowledge of σ-positive braid, Garside Normal form of the braid, braid ordering and braid rewrite are all what have been learned from the Braids and Braid Words program. Besides the actual method in the Braids and Braid Words program, the program helps to learn the way of thinking about the methods, an helps to improve how to understanding the braids.

Secondly, the Braids and Braid Words program improves the skill of programming. The Braids and Braid Words program is a single program, the programming from the design to debugging is all done by oneself. It trainings how to develop a program along. And when programming is under coding, many coding skills have been improved or leant. From the Braids and Braid Words program, the coding of the java graphic user interface, which include the knowledge of coding buttons, add mouse actions, draw pictures on in java draw panel, add mouse event on output list of braids and so on. On the other hand, working on the Braids and Braid Words program improves the coding style and the coding quality. These are important skills that could only learned when doing the project but could not learn from the book or classes.

Thirdly, the Braids and Braid Words program improves the writing skills of the report. The Braids and Braid Words program is include several reports, from the specification to the design document, to the project report. The writing skills are learned or improved by writing these reports.

Finally, the Braids and Braid Words program also improves the presentation skill. The two presentations, which are the design prostration and the demonstration, could help somehow improve the

9 Professional Issues

The British Computer Society (BCS) set out the professional standards for individuals, organizations and society. For this Braids and Braid Words program, Code of Conduct was applied and complied follows:

1. The Braids and Braid Words program is carefully worked and studied with the requirements.
2. The Braids and Braid Words program has no influence of the public health, safety and environment.
3. The Braids and Braid Words program verified all the methods and information to avoid violation of the legitimate rights of third parties.
4. The Braids and Braid Words program is built with learning about the relevant legislation, regulations and standards of the professional field.
5. The Braids and Braid Words program could be easily used by against clients or colleagues.
6. The Braids and Braid Words program rejected any offer of bribery or inducement.
7. The Braids and Braid Words program would not meet conflict of interest with relevant authority.
8. The Braids and Braid Words program will not disclose or authorize to be disclosed, or use for personal gain or to benefit a third party without the permission.
9. The Braids and Braid Words program is easy to all user to use.
10. The Braids and Braid Words program keep communicate with other profession in general and seek to improve professional standards through participation in their development.

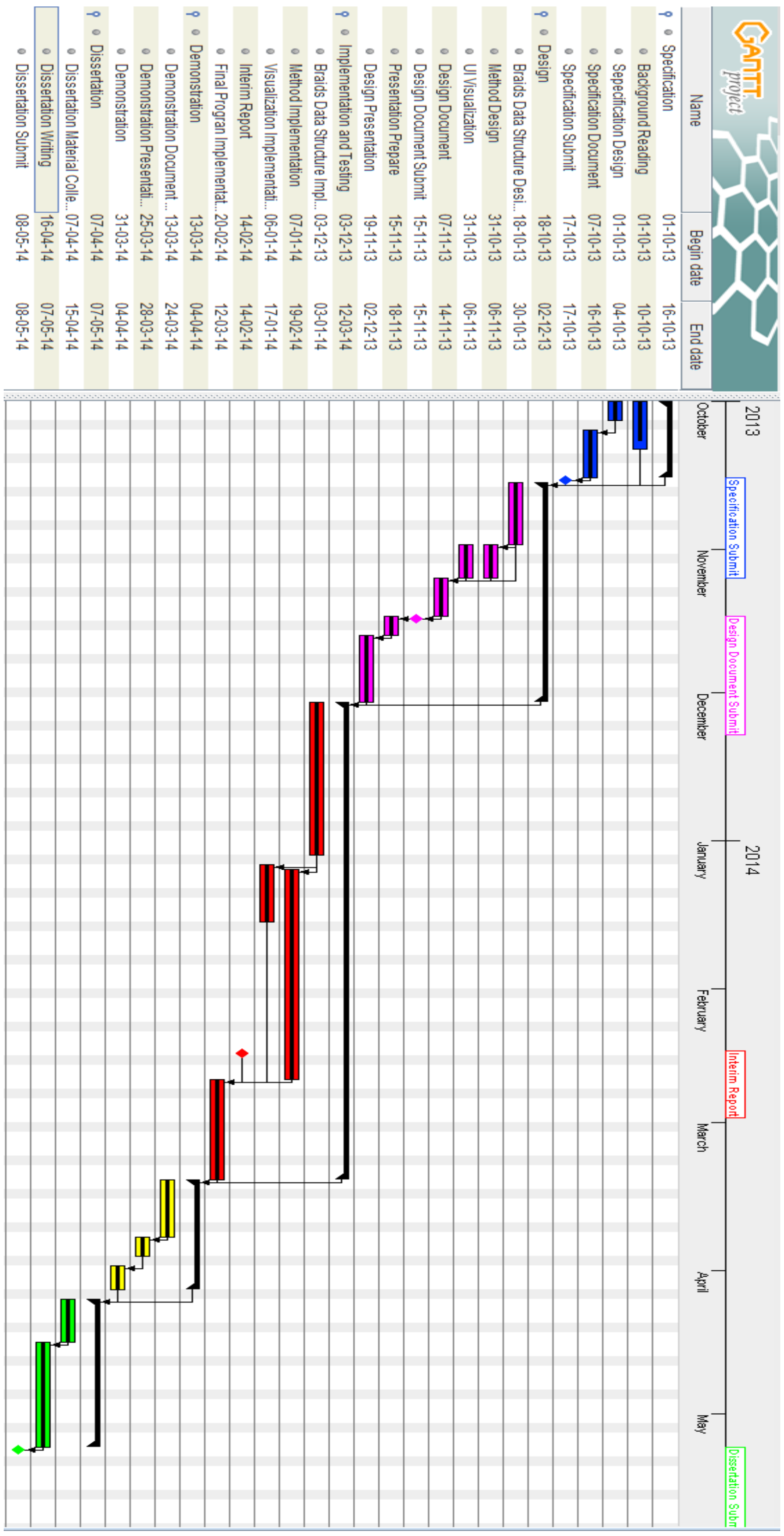
11. The Braids and Braid Words program is developed by own and did not have problems with integrity in the relationships with all other members.
12. The Braids and Braid Words program would not make any public statement about the system without authorized to do so.
13. The Braids and Braid Words program should notify the society if convicted of a criminal offence or upon becoming bankrupt or disqualified as the director.
14. The author were learning the professional knowledge and skill to seek to upgrade the Braids and Braid Words program.

10 Bibliography

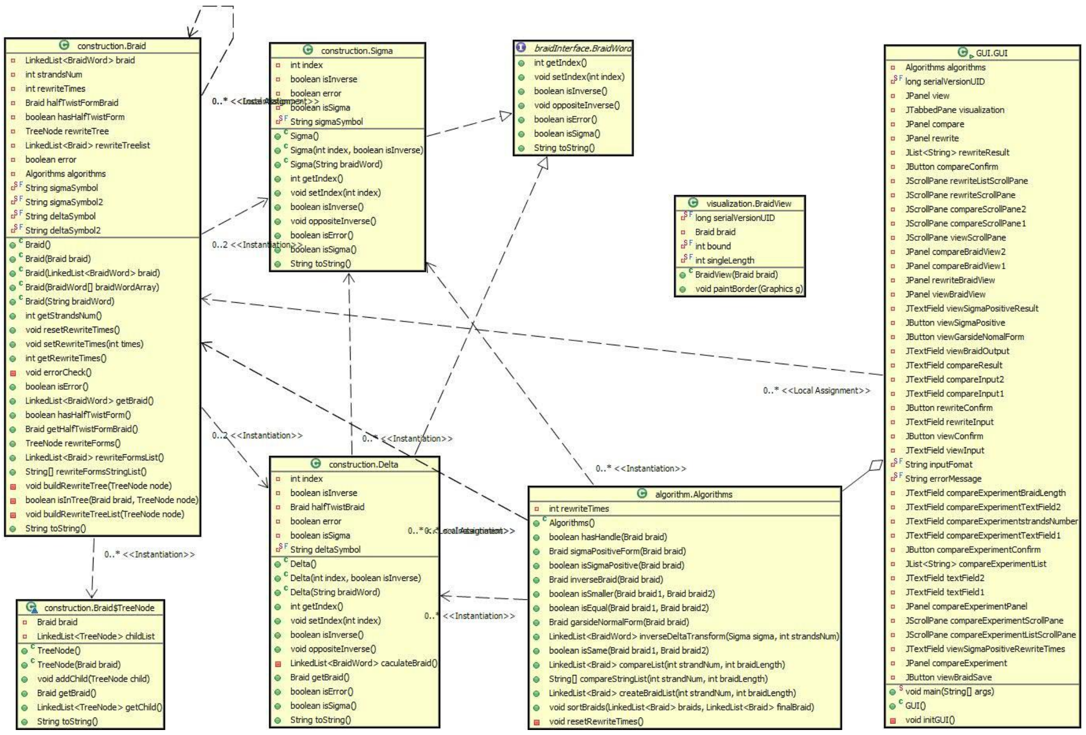
1. Jonathan Gomez Boiser, *Computational Problems in the Braid Group*, Fall 2009
2. Patrick Dehornoy, *The braid order: history and connection with knots,* ICTP, Trieste, Italy, May 2008
3. Patrick Dehornoy, *Foundations of Garside Theory*, Universit ́e de Caen, 14032 Caen, France, 2013
4. Patrick Dehornoy, *Ordering Braids*, Universit ́e de Caen, 14032 Caen, France, 2008
5. Patrick Dehornoy, *Why Are Braids Orderable?*, Universit ́e de Caen, 14032 Caen, France, 2002

11 Appendices

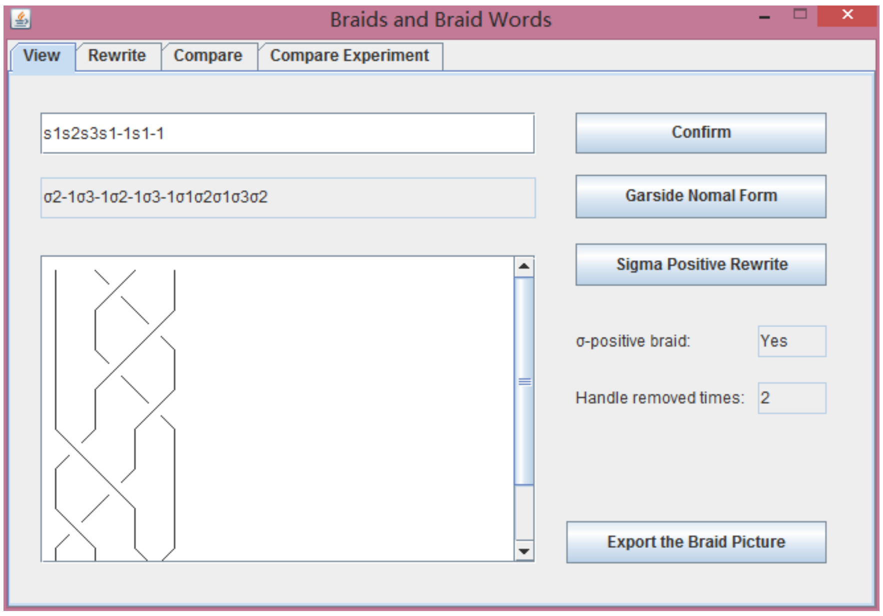
● Design Diagrams



Name	Begin date	End date
Specification	01-10-13	16-10-13
Background Reading	01-10-13	10-10-13
Sepecification Design	01-10-13	04-10-13
Specification Document	07-10-13	16-10-13
Specification Submit	17-10-13	17-10-13
Design	18-10-13	02-12-13
Braids Data Structure Desi...	18-10-13	30-10-13
Method Design	31-10-13	06-11-13
UI Visualization	31-10-13	06-11-13
Design Document	07-11-13	14-11-13
Design Document Submit	15-11-13	15-11-13
Presentation Prepare	15-11-13	18-11-13
Design Presentation	19-11-13	02-12-13
Implementation and Testing	03-12-13	12-03-14
Braids Data Structure Impl...	03-12-13	03-01-14
Method Implementation	07-01-14	19-02-14
Visualization Implementati...	06-01-14	17-01-14
Interim Report	14-02-14	14-02-14
Final Progran Implementat...	20-02-14	12-03-14
Demonstration	13-03-14	04-04-14
Demonstration Document...	13-03-14	24-03-14
Demonstration Presentati...	25-03-14	28-03-14
Demonstration	31-03-14	04-04-14
Dissertation	07-04-14	07-05-14
Dissertation Material Colle...	07-04-14	15-04-14
Dissertation Writing	16-04-14	07-05-14
Dissertation Submit	08-05-14	08-05-14


construction.Braid
LinkedList<BraidWord> braid
int strandsNum
int rewriteTimes
Braid halfTwistFormBraid
boolean hasHalfTwistForm
TreeNode rewriteTree
LinkedList<Braid> rewriteTreelist
boolean error
Algorithms algorithms
String sigmaSymbol
String sigmaSymbol2
String deltaSymbol
String deltaSymbol2
Braid()
Braid(Braid braid)
Braid(LinkedList<BraidWord> braid)
Braid(BraidWord[] braidWordArray)
Braid(String braidWord)
int getStrandsNum()
void resetRewriteTimes()
void setRewriteTimes(int times)
int getRewriteTimes()
void errorCheck()
boolean isError()
LinkedList<BraidWord> getBraid()
boolean hasHalfTwistForm()
Braid getHalfTwistFormBraid()
TreeNode rewriteForms()
LinkedList<Braid> rewriteFormsList()
String[] rewriteFormsStringList()
void buildRewriteTree(TreeNode node)
boolean isInTree(Braid braid, TreeNode node)
void buildRewriteTreeList(TreeNode node)
String toString()
construction.Sigma
int index
boolean isInverse
boolean error
boolean isSigma
String sigmaSymbol
Sigma()
Sigma(int index, boolean isInverse)
Sigma(String braidWord)
int getIndex()
void setIndex(int index)
boolean isInverse()
void oppositeInverse()
boolean isError()
boolean isSigma()
String toString()
braidInterface.BraidWord
int getIndex()
void setIndex(int index)
boolean isInverse()
void oppositeInverse()
boolean isError()
boolean isSigma()
String toString()
visualization.BraidView
long serialVersionUID
Braid braid
int bound
int singleLength
BraidView(Braid braid)
void paintBorder(Graphics g)
GUI.GUI
Algorithms algorithms
long serialVersionUID
JPanel view
JTabbedPane visualization
JPanel compare
JPanel rewrite
JList<String> rewriteResult
JButton compareConfirm
JScrollPane rewriteListScrollPane
JScrollPane rewriteScrollPane
JScrollPane compareScrollPane2
JScrollPane compareScrollPane1
JScrollPane viewScrollPane
JPanel compareBraidView2
JPanel compareBraidView1
JPanel rewriteBraidView
JPanel viewBraidView
JTextField viewSigmaPositiveResult
JButton viewSigmaPositive
JButton viewGarsideNomalForm
JTextField viewBraidOutput
JTextField compareResult
JTextField compareInput2
JTextField compareInput1
JButton rewriteConfirm
JTextField rewriteInput
JButton viewConfirm
JTextField viewInput
String inputFomat
String errorMessage
JTextField compareExperimentBraidLength
JTextField compareExperimentTextField2
JTextField compareExperimentstrandsNumber
JTextField compareExperimentTextField1
JButton compareExperimentConfirm
JList<String> compareExperimentList
JTextField textField2
JTextField textField1
JPanel compareExperimentPanel
JScrollPane compareExperimentScrollPane
JScrollPane compareExperimentListScrollPane
JTextField viewSigmaPositiveRewriteTimes
JPanel compareExperiment
JButton viewBraidSave
void main(String[] args)
GUI()
void initGUI()
construction.Braid$TreeNode
Braid braid
LinkedList<TreeNode> childList
TreeNode()
TreeNode(Braid braid)
void addChild(TreeNode child)
Braid getBraid()
LinkedList<TreeNode> getChild()
String toString()
construction.Delta
int index
boolean isInverse
Braid halfTwistBraid
boolean error
boolean isSigma
String deltaSymbol
Delta()
Delta(int index, boolean isInverse)
Delta(String braidWord)
int getIndex()
void setIndex(int index)
boolean isInverse()
void oppositeInverse()
LinkedList<BraidWord> caculateBraid()
Braid getBraid()
boolean isError()
boolean isSigma()
String toString()
algorithm.Algorithms
int rewriteTimes
Algorithms()
boolean hasHandle(Braid braid)
Braid sigmaPositiveForm(Braid braid)
boolean isSigmaPositive(Braid braid)
Braid inverseBraid(Braid braid)
boolean isSmaller(Braid braid1, Braid braid2)
boolean isEqual(Braid braid1, Braid braid2)
Braid garsideNormalForm(Braid braid)
LinkedList<BraidWord> inverseDeltaTransform(Sigma sigma, int strandsNum)
boolean isSame(Braid braid1, Braid braid2)
LinkedList<Braid> compareList(int strandNum, int braidLength)
String[] compareStringList(int strandNum, int braidLength)
LinkedList<Braid> createBraidList(int strandNum, int braidLength)
void sortBraids(LinkedList<Braid> braids, LinkedList<Braid> finalBraid)
void resetRewriteTimes()
0..* <<Local Assignment>>
0..2 <<Instantiation>>
0..* <<Instantiation>>
0..* <<Local Assignment>>
0..* <<Instantiation>>
0..2 <<Instantiation>>
0..* <<Instantiation>>
0..* <<Local Assignment>>
0..* <<Instantiation>>

● Screen Shot of Sample Runs



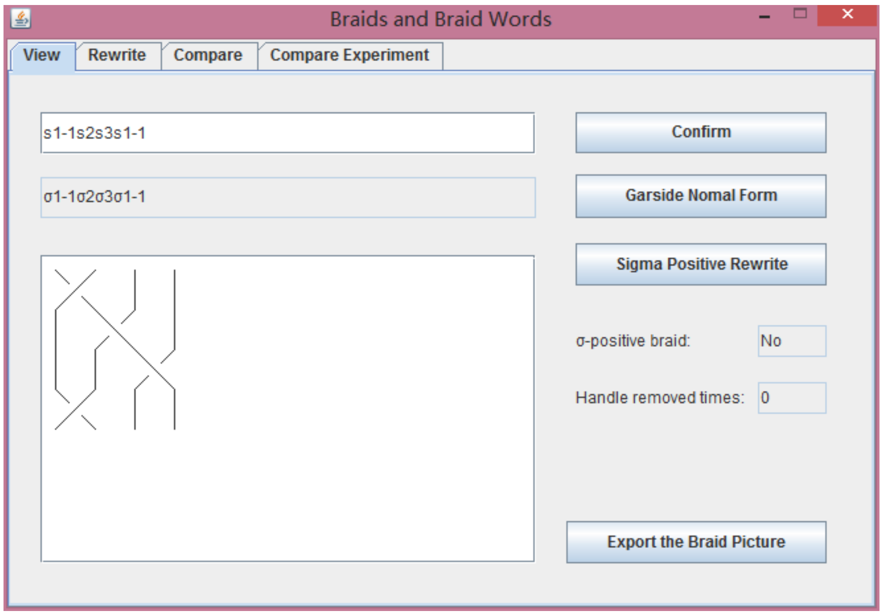
s1s2s3s1-1s1-1

σ2-1σ3-1σ2-1σ3-1σ1σ2σ1σ3σ2

σ-positive braid: Yes

Handle removed times: 2

Figure 5.1 The result of the first example of the σ-positive braid



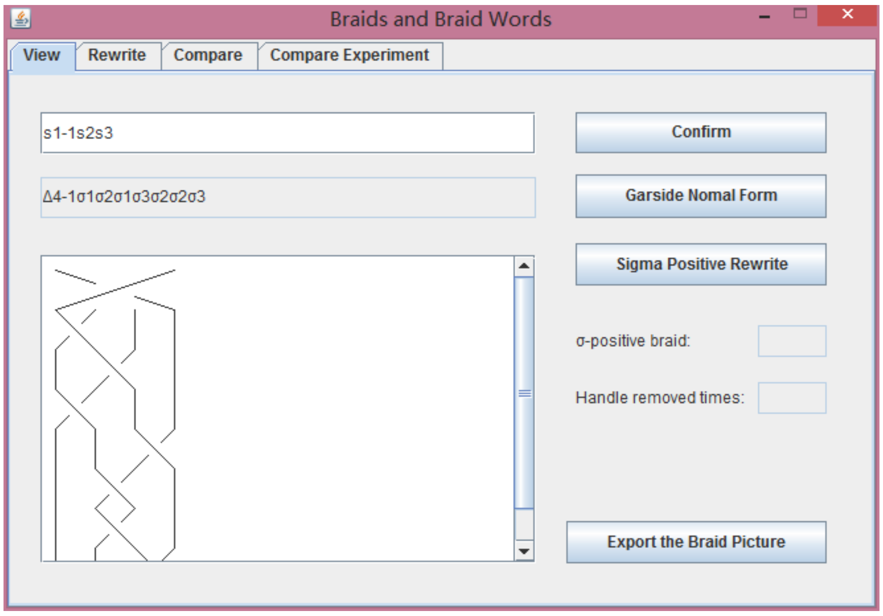
s1-1s2s3s1-1

σ1-1σ2σ3σ1-1

σ-positive braid: No

Handle removed times: 0

Figure 5.2 The result of the second example of the σ-positive braid



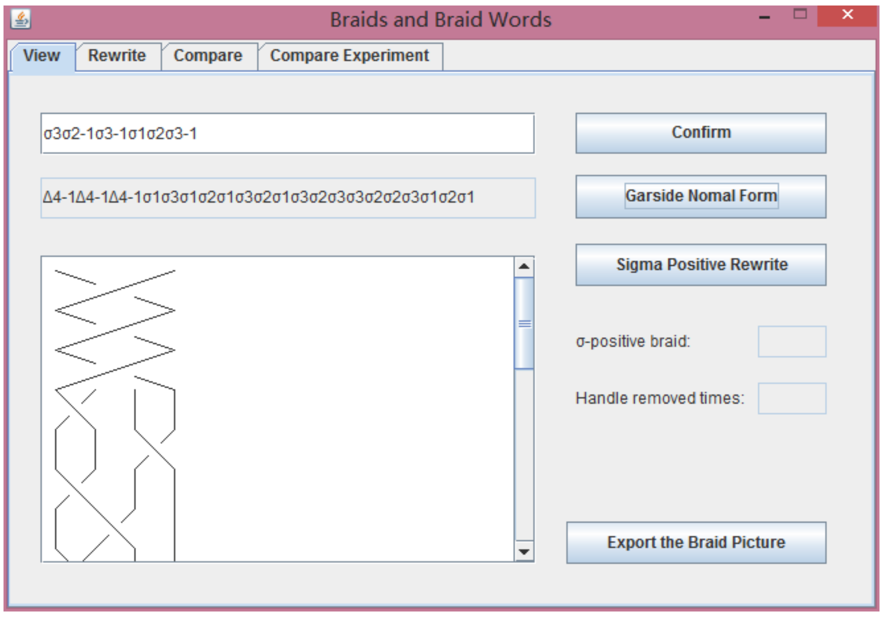
s1-1s2s3

Δ4-1σ1σ2σ1σ3σ2σ2σ3

σ-positive braid:

Handle removed times:

Figure 5.3 The result of the first example of the Garisde Normal form



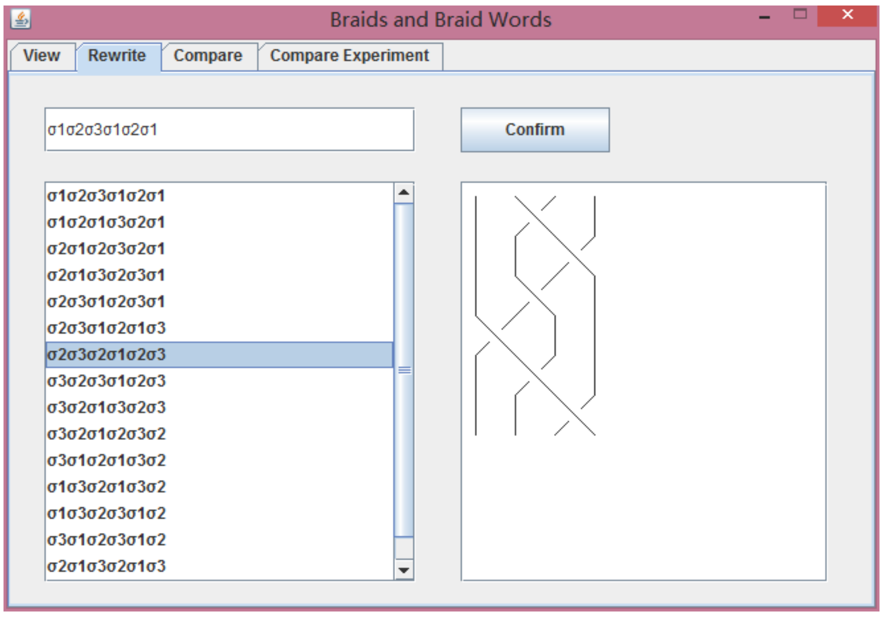
σ3σ2-1σ3-1σ1σ2σ3-1

Δ4-1Δ4-1Δ4-1σ1σ3σ1σ2σ1σ3σ2σ1σ3σ2σ3σ3σ2σ2σ3σ1σ2σ1

σ-positive braid:

Handle removed times:

Figure 5.4 The result of the second example of the Garisde Normal form



σ1σ2σ3σ1σ2σ1

```
σ1σ2σ3σ1σ2σ1
σ1σ2σ1σ3σ2σ1
σ2σ1σ2σ3σ2σ1
σ2σ1σ3σ2σ3σ1
σ2σ3σ1σ2σ3σ1
σ2σ3σ1σ2σ1σ3
σ2σ3σ2σ1σ2σ3
σ3σ2σ3σ1σ2σ3
σ3σ2σ1σ3σ2σ3
σ3σ2σ1σ2σ3σ2
σ3σ1σ2σ1σ3σ2
σ1σ3σ2σ1σ3σ2
σ1σ3σ2σ3σ1σ2
σ3σ1σ2σ3σ1σ2
σ2σ1σ3σ2σ1σ3
```

Figure 5.5 The result of the example of the rewrite algorithms

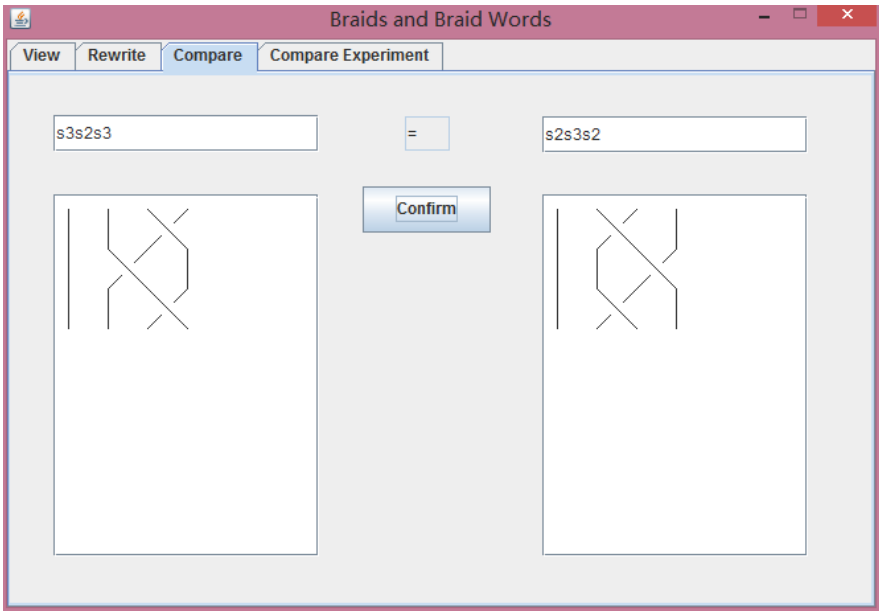


Figure 5.6 The result of the first example of the braid comparing

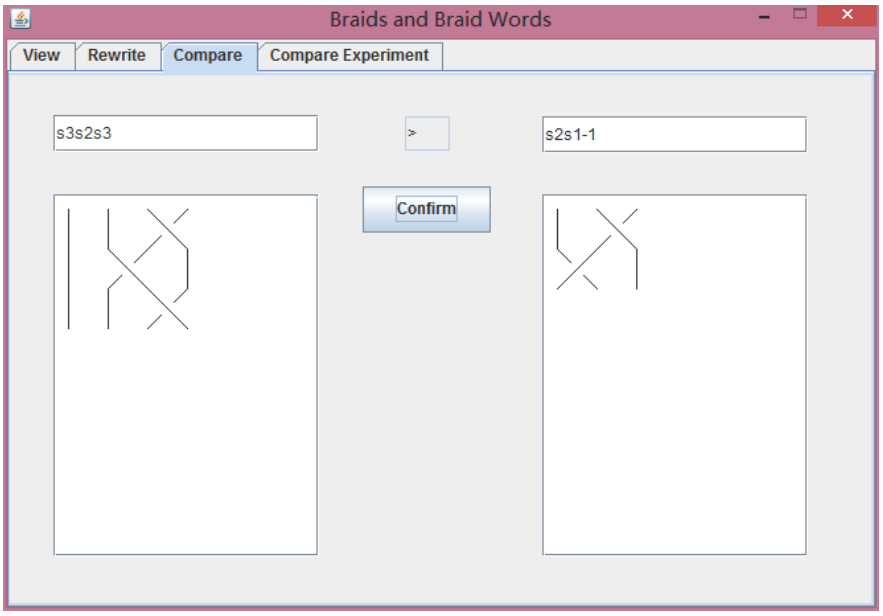
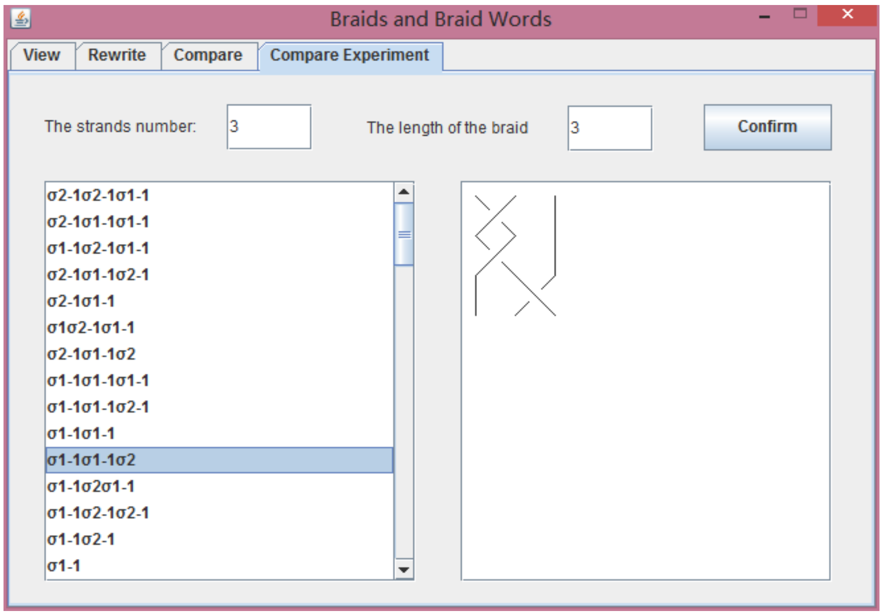


Figure 5.7 The result of the second example of the braid comparing



The strands number: 3

The length of the braid 3

σ2-1σ2-1σ1-1
σ2-1σ1-1σ1-1
σ1-1σ2-1σ1-1
σ2-1σ1-1σ2-1
σ2-1σ1-1
σ1σ2-1σ1-1
σ2-1σ1-1σ2
σ1-1σ1-1σ1-1
σ1-1σ1-1σ2-1
σ1-1σ1-1
σ1-1σ1-1σ2
σ1-1σ2σ1-1
σ1-1σ2-1σ2-1
σ1-1σ2-1
σ1-1

Figure 6.1 The screen shot of the braid ordering example graphic user inter face

- ## User Guide

The program is well designed and simple to use. For the users to use the program, it is just need to input the braid in the text field, and click confirm to show the braid wish to see or to rewrite.

● Code Listing

Algorithms.java

```
package algorithm;

import java.util.LinkedList;

import braidInterface.BraidWord;
import construction.Braid;
import construction.Delta;
import construction.Sigma;

/**
 * The Algorithms of the Braids and Braid Words Program, and it stores most of
 * the algorithms of the braid.
 *
 * @author Zhang Xi
 * @author 200906945
 * @version 06-05-2014
 */
public class Algorithms {

	/** The variable stores the time of the rewrite times. */
	private int rewriteTimes;

	/**
	 * Create a algorithm object.
	 *
	 */
	public Algorithms() {
		rewriteTimes = 0;
	}

	/**
	 * Check whether there is handle in the braid.
	 *
	 * @param braid
	 *            the braid need to check
	 * @return whether there is a handle
	 */
	public boolean hasHandle(Braid braid) {
		Braid tmpBraid;
		if (braid.hasHalfTwistForm()) {
			tmpBraid = braid.getHalfTwistFormBraid();
		} else {
			tmpBraid = braid;
		}
		boolean tmpBoolean = false;
		BraidWord tmpB1 = tmpBraid.getBraid().get(0);
		for (int i = 1; i < tmpBraid.getBraid().size(); i++) {
			BraidWord tmpB2 = tmpBraid.getBraid().get(i);
			if (tmpB2.getIndex() < tmpB1.getIndex()) {
				tmpB1 = tmpB2;
				tmpBoolean = false;
			} else if (tmpB2.getIndex() == tmpB1.getIndex()) {
				if ((tmpB1.isInverse() == true && tmpB2.isInverse() == false)
						|| (tmpB1.isInverse() == false && tmpB2.isInverse() == true))
{
					tmpBoolean = true;
				}
			}
		}
		return tmpBoolean;
	}

	/**
	 * Get the sigma positive rewrite form of a braid.
	 *
	 * @param braid
	 *            the braid need to rewrite
	 * @return the rewrite braid
	 */
	public Braid sigmaPositiveForm(Braid braid) {
		Braid tmpBraid;
		if (braid.hasHalfTwistForm()) {
			tmpBraid = new Braid(braid.getHalfTwistFormBraid());
		} else {
			tmpBraid = new Braid(braid);
		}
		while (rewriteTimes <= 1000) {
			boolean hasSigmaPositive = false;
			if (tmpBraid.getBraid().size() == 0) {
				Braid newBraid = new Braid();
				newBraid.setRewriteTimes(rewriteTimes);
```

```
                resetRewriteTimes();
                return newBraid;
            }
            BraidWord tmpB1 = tmpBraid.getBraid().get(0);
            int countB1 = 0;
            int countB2 = -1;
            for (int i = 1; i < tmpBraid.getBraid().size(); i++) {
                BraidWord tmpB2 = tmpBraid.getBraid().get(i);
                if (tmpB2.getIndex() < tmpB1.getIndex()) {
                    tmpB1 = tmpB2;
                    countB1 = i;
                    countB2 = -1;
                    hasSigmaPositive = false;
                } else if (tmpB2.getIndex() == tmpB1.getIndex()) {
                    if ((tmpB1.isInverse() == true && tmpB2.isInverse() == false)
                            || (tmpB1.isInverse() == false && tmpB2.isInverse() ==
true)) {
                        countB2 = i;
                        hasSigmaPositive = true;
                        break;
                    } else {
                        countB1 = i;
                    }
                }
            }
            if (!hasSigmaPositive) {
                tmpBraid.setRewriteTimes(rewriteTimes);
                resetRewriteTimes();
                return tmpBraid;
            } else {
                int tmpIndexcount = tmpBraid.getStrandsNum();
                int tmpIndex = tmpBraid.getStrandsNum() - 1;
                for (int j = countB1 + 1; j < countB2; j++) {
                    tmpBraid.getBraid()
                            .get(j)
                            .setIndex(tmpBraid.getBraid().get(j).getIndex() - 1);
                }
                LinkedList<BraidWord> tmpList1 = new LinkedList<BraidWord>();
                LinkedList<BraidWord> tmpList2 = new LinkedList<BraidWord>();
                for (int k = tmpB1.getIndex() + 1; k < tmpIndexcount; k++) {
                    tmpList1.add(new Sigma(k, !tmpB1.isInverse()));
                    tmpList2.add(new Sigma(tmpIndex, tmpB1.isInverse()));
                    tmpIndex--;
                }
                tmpBraid.getBraid().remove(countB2);
                tmpBraid.getBraid().addAll(countB2, tmpList2);
                tmpBraid.getBraid().remove(countB1);
                tmpBraid.getBraid().addAll(countB1, tmpList1);
            }
            rewriteTimes++;
        }
        Braid newBraid = new Braid();
        newBraid.setRewriteTimes(-1);
        resetRewriteTimes();
        return newBraid;
    }

    /**
     * Check whether the braid is a sigma positive braid.
     *
     * @param braid
     *            the braid need to be checked
     * @return whether the braid is a sigma positive braid
     */
    public boolean isSigmaPositive(Braid braid) {
        Boolean isSigmaPositive = false;
        Braid tmpBraid = new Braid(sigmaPositiveForm(braid));
        int tmpIndex = tmpBraid.getStrandsNum();
        for (BraidWord b : tmpBraid.getBraid()) {
            if (b.getIndex() < tmpIndex) {
                tmpIndex = b.getIndex();
                if (b.isInverse()) {
                    isSigmaPositive = false;
                } else {
                    isSigmaPositive = true;
                }
            } else if (b.getIndex() == tmpIndex) {
                if (b.isInverse()) {
                    isSigmaPositive = false;
                }
            }
        }
        return isSigmaPositive;
    }
```

```
/**
 * Get the inverse case of a braid.
 *
 * @param braid
 *            the braid need to rewrite
 * @return the inverse rewrite braid
 */
public Braid inverseBraid(Braid braid) {
    LinkedList<BraidWord> list = new LinkedList<BraidWord>();
    for (int i = 0; i < braid.getBraid().size(); i++) {
        if (braid.getBraid().get(i).isSigma()) {
            list.add(0, new Sigma(braid.getBraid().get(i).getIndex(),
                    !braid.getBraid().get(i).isInverse()));
        } else {
            list.add(0, new Delta(braid.getBraid().get(i).getIndex(),
                    !braid.getBraid().get(i).isInverse()));
        }
    }
    Braid tmpBraid = new Braid(list);
    return tmpBraid;
}

/**
 * Check which braid is smaller.
 *
 * @param braid1
 *            the check braid
 * @param braid2
 *            the check braid
 * @return whether braid1 is smaller then braid 2
 */
public boolean isSmaller(Braid braid1, Braid braid2) {
    LinkedList<BraidWord> list = new LinkedList<BraidWord>();
    list.addAll(new Braid(inverseBraid(braid1)).getBraid());
    list.addAll(braid2.getBraid());
    Braid tmpBraid = new Braid(list);
    if (isSigmaPositive(tmpBraid)) {
        return true;
    } else {
        return false;
    }
	}

	/**
	 * Check whether two braids are equal.
	 *
	 * @param braid1
	 *            the check braid
	 * @param braid2
	 *            the check braid
	 * @return whether braid1 and braid2 are equal
	 */
	public boolean isEqual(Braid braid1, Braid braid2) {
		if (!isSmaller(braid1, braid2) && !isSmaller(braid2, braid1)) {
			return true;
		} else {
			return false;
		}
	}

	/**
	 * Rewrite the braid into Garside normal form
	 *
	 * @param braid
	 *            the braid to rewrite
	 * @return the Gariside normal form of the braid
	 */
	public Braid garsideNormalForm(Braid braid) {
		Braid tmpBraid;
		if (braid.hasHalfTwistForm()) {
			tmpBraid = new Braid(braid.getHalfTwistFormBraid());
		} else {
			tmpBraid = new Braid(braid);
		}
		boolean transForm = false;
		int count = 0;
		int i = tmpBraid.getBraid().size() - 1;
		for (; i >= 0; i--) {
			if (tmpBraid.getBraid().get(i).isInverse()) {
				LinkedList<BraidWord> tmpList = inverseDeltaTransform(
						(Sigma) tmpBraid.getBraid().get(i),
						braid.getStrandsNum());
				if (transForm) {
```

```
                        for (BraidWord b : tmpList) {
                            b.setIndex(braid.getStrandsNum() - b.getIndex());
                        }
                    }
                    tmpBraid.getBraid().remove(i);
                    tmpBraid.getBraid().addAll(i, tmpList);
                    for (int j = 0; j < i; j++) {
                        if (!tmpBraid.getBraid().get(j).isInverse()) {
                            tmpBraid.getBraid()
                                    .get(j)
                                    .setIndex(
                                            braid.getStrandsNum()
                                                    - tmpBraid.getBraid().get(j)
                                                            .getIndex());
                        }
                    }
                    transForm = !transForm;
                    count++;
                }
            }
            for (int k = 0; k < count; k++) {
                tmpBraid.getBraid().add(0, new Delta(braid.getStrandsNum(), true));
            }
            return tmpBraid;
        }

        /**
         * Transform the braid in to inverse delta case.
         *
         * @param sigma
         *            the sigma braid
         * @param strandsNum
         *            the total strand number
         * @return the final braid
         */
        public LinkedList<BraidWord> inverseDeltaTransform(Sigma sigma,
                int strandsNum) {
            LinkedList<BraidWord> list = new LinkedList<BraidWord>();
            if (!sigma.isInverse()) {
                list.add(sigma);
                return list;
            } else if (sigma.getIndex() >= strandsNum) {
                return null;
            }
            for (int i = 1; i < strandsNum - 1; i++) {
                for (int j = i; j >= 1; j--) {
                    list.add(new Sigma(j, false));
                }
            }
            for (int k = strandsNum - 1; k > sigma.getIndex(); k--) {
                list.add(new Sigma(k, false));
            }
            for (int l = 1; l < sigma.getIndex(); l++) {
                list.add(0, new Sigma(strandsNum - l, false));
            }
            return list;
        }

        /**
         * Check whether two braids are same.
         *
         * @param braid1
         *             the check braid
         * @param braid2
         *             the check braid
         * @return whether two braids are same
         */
        public boolean isSame(Braid braid1, Braid braid2) {
            if (braid1.getBraid().size() == braid2.getBraid().size()) {
                for (int i = 0; i < braid1.getBraid().size(); i++) {
                    if (braid1.getBraid().get(i).getIndex() != braid2.getBraid()
                            .get(i).getIndex()) {
                        return false;
                    }
                    if (braid1.getBraid().get(i).isInverse() != braid2.getBraid()
                            .get(i).isInverse()) {
                        return false;
                    }
                }
                return true;
            }
            return false;
        }
```

```
/**
 * The main method support the compare experiment.
 *
 * @param strandNum
 *            the strand number of the braid
 * @param braidLength
 *            the total braid length
 * @return the final compare list
 */
public LinkedList<Braid> compareList(int strandNum, int braidLength) {
    LinkedList<Braid> tmpBraid = new LinkedList<Braid>();
    sortBraids(createBraidList(strandNum, braidLength), tmpBraid);
    return tmpBraid;
}

/**
 * Support the compare experiment and get the String result.
 *
 * @param strandNum
 *            the strand number of the braid
 * @param braidLength
 *            the total braid length
 * @return the final compare list
 */
public String[] compareStringList(int strandNum, int braidLength) {
    LinkedList<Braid> tmpBraid = compareList(strandNum, braidLength);
    String[] tmpStringBraid = new String[tmpBraid.size()];
    for (int i = 0; i < tmpBraid.size(); i++) {
        tmpStringBraid[i] = tmpBraid.get(i).toString();
    }
    return tmpStringBraid;
}

/**
 * Create the whole rewrite list.
 *
 * @param strandNum
 *            the strand number of the braid
 * @param braidLength
 *            the total braid length
 * @return the final result
 */
	public LinkedList<Braid> createBraidList(int strandNum, int braidLength) {
		LinkedList<Braid> tmpList = new LinkedList<Braid>();
		String[][] tmpString = new String[braidLength][];
		tmpString[0] = new String[(strandNum - 1) * 2];
		for (int i = 1; i <= (strandNum - 1); i++) {
			tmpString[0][i * 2 - 2] = "s" + i;
			tmpString[0][i * 2 - 1] = "s" + i + "-1";
		}
		for (int i = 2; i <= braidLength; i++) {
			tmpString[i - 1] = new String[tmpString[i - 2].length
					* (strandNum - 1) * 2];
			for (int j = 0; j < tmpString[i - 2].length; j++) {
				for (int k = 1; k <= (strandNum - 1); k++) {
					tmpString[i - 1][k * 2 - 2 + (strandNum - 1) * 2 * j] = tmpString[i -
2][j]
							+ "s" + k;
					tmpString[i - 1][k * 2 - 1 + (strandNum - 1) * 2 * j] = tmpString[i -
2][j]
							+ "s" + k + "-1";
				}
			}
		}
		for (String[] s : tmpString) {
			for (String ss : s) {
				tmpList.add(new Braid(ss));
			}
		}
		return tmpList;
	}

	/**
	 * Sorts the braids
	 * 
	 * @param braids
	 *            list of braids
	 * @param finalBraid
	 *            the braid list the the result stored in
	 */
	public void sortBraids(LinkedList<Braid> braids,
			LinkedList<Braid> finalBraid) {
		if (braids.size() == 0) {
			return;
		} else if (braids.size() == 1) {
			finalBraid.add(braids.getFirst());
		} else {
			LinkedList<Braid> left = new LinkedList<Braid>();
			LinkedList<Braid> mid = new LinkedList<Braid>();
			LinkedList<Braid> right = new LinkedList<Braid>();
			Braid thisBraid = braids.getFirst();
			for (Braid b : braids) {
				if (isSmaller(b, thisBraid)) {
					left.add(b);
				} else if (isSmaller(thisBraid, b)) {
					right.add(b);
				} else if (isEqual(b, thisBraid)) {
					mid.add(b);
				}
			}
			sortBraids(left, finalBraid);
			finalBraid.addAll(mid);
			sortBraids(right, finalBraid);
		}
	}

	/**
	 * Reset the rewrite times in to 0;
	 *
	 */
	private void resetRewriteTimes() {
		rewriteTimes = 0;
	}

}
```

BraidWord.java

```
package braidInterface;

/**
 * The interface that is the data type of the sigma and delta.
 *
 * @author Zhang Xi
 * @author 200906945
```

```
  * @version 06-05-2014
  */
public interface BraidWord {

	/**
	 * Get the index number.
	 *
	 * @return the index number
	 */
	public int getIndex();

	/**
	 * Set the index number of the braid.
	 *
	 * @param index the index number
	 */
	public void setIndex(int index);

	/**
	 * Whether the braid is in inverse case.
	 *
	 * @return whether is inverse
	 */
	public boolean isInverse();

	/**
	 * Change the braid into the opposite inverse mark
	 *
	 */
	public void oppositeInverse();

	/**
	 * Whether is the braid is error.
	 *
	 * @return whether the braid is error
	 */
	public boolean isError();

	/**
	 * Whether the braid word is sigma.
	 *
	 * @return whether the braid word is sigma
	 */
	public boolean isSigma();

	/**
	 * Get the string case of the braid.
	 *
	 * @return the string braid word.
	 */
	public String toString();
}
```

Braid.java

```
package construction;

import java.util.LinkedList;

import algorithm.Algorithms;
import braidInterface.BraidWord;

/**
 * The object saved the braid with the braid words
 *
 * @author Zhang Xi
 * @author 200906945
 * @version 06-05-2014
 */
public class Braid {

	/**
	 *
	 */
	private LinkedList<BraidWord> braid;
	private int strandsNum;
	private int rewriteTimes;
	private Braid halfTwistFormBraid;
	private boolean hasHalfTwistForm;
	private TreeNode rewriteTree;
	private LinkedList<Braid> rewriteTreelist;
	private boolean error;
	private Algorithms algorithms = new Algorithms();
private static final String sigmaSymbol = "σ";
private static final String sigmaSymbol2 = "s";
private static final String deltaSymbol = "Δ";
private static final String deltaSymbol2 = "d";

public Braid() {
        braid = new LinkedList<BraidWord>();
        strandsNum = -1;
        rewriteTimes = -2;
        hasHalfTwistForm = false;
        halfTwistFormBraid = null;
        rewriteTree = null;
        rewriteTreelist = null;
        error = true;
}

public Braid(Braid braid) {
        this.braid = new LinkedList<BraidWord>();
        for (BraidWord b : braid.getBraid()) {
                if (b.isSigma()) {
                        this.braid.add(new Sigma(b.getIndex(), b.isInverse()));
                } else {
                        this.braid.add(new Delta(b.getIndex(), b.isInverse()));
                }
        }
        strandsNum = -1;
        rewriteTimes = -2;
        hasHalfTwistForm = braid.hasHalfTwistForm();
        halfTwistFormBraid = null;
        rewriteTree = null;
        rewriteTreelist = null;
        error = braid.isError();
}

public Braid(LinkedList<BraidWord> braid) {
        this.braid = new LinkedList<BraidWord>();
        for (BraidWord b : braid) {
                if (b.isSigma()) {
                        this.braid.add(new Sigma(b.getIndex(), b.isInverse()));
                } else {
                        this.braid.add(new Delta(b.getIndex(), b.isInverse()));
```

```
            }
        }
        strandsNum = -1;
        rewriteTimes = -2;
        hasHalfTwistForm = false;
        halfTwistFormBraid = null;
        rewriteTree = null;
        rewriteTreelist = null;
        error = false;
    }

    public Braid(BraidWord[] braidWordArray) {
        braid = new LinkedList<BraidWord>();
        strandsNum = -1;
        rewriteTimes = -2;
        hasHalfTwistForm = false;
        for (BraidWord o : braidWordArray) {
            braid.add(o);
            if (!o.isSigma()) {
                hasHalfTwistForm = true;
            }
        }
        halfTwistFormBraid = null;
        rewriteTree = null;
        rewriteTreelist = null;
        errorCheck();
    }

    public Braid(String braidWord) {
        braid = new LinkedList<BraidWord>();
        strandsNum = -1;
        rewriteTimes = -2;
        halfTwistFormBraid = null;
        rewriteTree = null;
        rewriteTreelist = null;
        String tmp = braidWord;
        while (true) {
            boolean isDelta = false;
            if ((tmp.startsWith(deltaSymbol) || tmp.startsWith(deltaSymbol2))
                        || (tmp.startsWith(sigmaSymbol) || tmp
                                .startsWith(sigmaSymbol2))) {
                if ((tmp.startsWith(deltaSymbol) || tmp
                        .startsWith(deltaSymbol2))) {
                    isDelta = true;
                }
                tmp = tmp.substring(1);
                int sigmaLocation = tmp.indexOf(sigmaSymbol);
                if (tmp.indexOf(sigmaSymbol2) != -1) {
                    if ((tmp.indexOf(sigmaSymbol) == -1)
                            || (tmp.indexOf(sigmaSymbol2) < tmp
                                    .indexOf(sigmaSymbol))) {
                        sigmaLocation = tmp.indexOf(sigmaSymbol2);
                    }
                }
                int deltaLocation = tmp.indexOf(deltaSymbol);
                if (tmp.indexOf(deltaSymbol2) != -1) {
                    if ((tmp.indexOf(deltaSymbol) == -1)
                            || (tmp.indexOf(deltaSymbol2) < tmp
                                    .indexOf(deltaSymbol))) {
                        deltaLocation = tmp.indexOf(deltaSymbol2);
                    }
                }
                String braidInfo = "";
                if (sigmaLocation == -1 && deltaLocation == -1) {
                    braidInfo = tmp;
                    if (!isDelta) {
                        braid.add(new Sigma(braidInfo));
                    } else {
                        braid.add(new Delta(braidInfo));
                        hasHalfTwistForm = true;
                    }
                    errorCheck();
                    return;
                } else if (sigmaLocation == -1 || deltaLocation == -1) {
                    braidInfo = tmp.substring(0,
                            Math.max(sigmaLocation, deltaLocation));
                    tmp = tmp.substring(Math.max(sigmaLocation, deltaLocation));
                } else {
                    braidInfo = tmp.substring(0,
                            Math.min(sigmaLocation, deltaLocation));
                    tmp = tmp.substring(Math.min(sigmaLocation, deltaLocation));
                }
                if (!isDelta) {
                    braid.add(new Sigma(braidInfo));
                } else {
                    braid.add(new Delta(braidInfo));
                    hasHalfTwistForm = true;
                }
            } else {
                break;
            }
        }
        error = true;
    }

    public int getStrandsNum() {
        if (braid.size() == 0) {
            return 0;
        }
        for (BraidWord o : braid) {
            if (o.isSigma()) {
                if (strandsNum < o.getIndex() + 1) {
                    strandsNum = o.getIndex() + 1;
                }
            } else {
                if (strandsNum < o.getIndex()) {
                    strandsNum = o.getIndex();
                }
            }
        }
        return strandsNum;
    }

    public void resetRewriteTimes() {
        rewriteTimes = 0;
    }

    public void setRewriteTimes(int times) {
        rewriteTimes = times;
    }

    public int getRewriteTimes() {
        return rewriteTimes;
    }

    private void errorCheck() {
```

```
        for (int i = 0; i < braid.size(); i++) {
            if (braid.get(i).isError()) {
                error = true;
                return;
            }
        }
        error = false;
    }

    public boolean isError() {
        errorCheck();
        return error;
    }

    public LinkedList<BraidWord> getBraid() {
        return braid;
    }

    public boolean hasHalfTwistForm() {
        return hasHalfTwistForm;
    }

    public Braid getHalfTwistFormBraid() {
        if (halfTwistFormBraid == null) {
            LinkedList<BraidWord> halfTwistForm = new LinkedList<BraidWord>();
            for (BraidWord o1 : braid) {
                if (o1.isSigma()) {
                    halfTwistForm.add(o1);
                } else {
                    Delta tmpD = (Delta) o1;
                    for (BraidWord o2 : tmpD.getBraid().getBraid()) {
                        halfTwistForm.add(o2);
                    }
                }
            }
            halfTwistFormBraid = new Braid(halfTwistForm);
        }
        return halfTwistFormBraid;
    }

    public TreeNode rewriteForms() {
        if (rewriteTree == null) {
            rewriteTree = new TreeNode(this);
            buildRewriteTree(rewriteTree);
        }
        return rewriteTree;
    }

    public LinkedList<Braid> rewriteFormsList() {
        if (rewriteTree == null) {
            rewriteTree = new TreeNode(this);
            buildRewriteTree(rewriteTree);
        }
        if (rewriteTreelist == null) {
            rewriteTreelist = new LinkedList<Braid>();
            buildRewriteTreeList(rewriteTree);
        }
        return rewriteTreelist;
    }

    public String[] rewriteFormsStringList() {
        String[] tmpList = new String[rewriteFormsList().size()];
        for (int i = 0; i < rewriteFormsList().size(); i++) {
            tmpList[i] = rewriteFormsList().get(i).toString();
        }
        return tmpList;
    }

    private void buildRewriteTree(TreeNode node) {
        Braid b = new Braid(node.getBraid());
        for (int i = 0; i < b.getBraid().size(); i++) {
            if (i >= 1) {
                if (Math.abs(b.getBraid().get(i).getIndex()
                        - b.getBraid().get(i - 1).getIndex()) >= 2) {
                    BraidWord tmp = b.getBraid().get(i);
                    b.getBraid().remove(i);
                    b.getBraid().add(i - 1, tmp);
                    if (!isInTree(b, rewriteTree)) {
                        node.addChild(new TreeNode(new Braid(b)));
                    }
                    b = new Braid(node.getBraid());
                }
                if (((b.getBraid().get(i).getIndex() == b.getBraid().get(i - 1)
                        .getIndex()) && (b.getBraid().size() != 2))
                        && (b.getBraid().get(i).isInverse() != b.getBraid()
                                .get(i - 1).isInverse())) {
                    b.getBraid().remove(i);
                    b.getBraid().remove(i - 1);
                    if (!isInTree(b, rewriteTree)) {
                        node.addChild(new TreeNode(new Braid(b)));
                    }
                    b = new Braid(node.getBraid());
                }
            }
            if (i >= 2) {
                if ((b.getBraid().get(i).isInverse() == b.getBraid().get(i - 1)
                        .isInverse())
                        && (b.getBraid().get(i - 1).isInverse() == b.getBraid()
                                .get(i - 2).isInverse())) {
                    if (b.getBraid().get(i - 2).getIndex() == b.getBraid()
                            .get(i).getIndex()) {
                        if (Math.abs(b.getBraid().get(i).getIndex()
                                - b.getBraid().get(i - 1).getIndex()) == 1) {
                            BraidWord tmp = b.getBraid().get(i - 1);
                            b.getBraid().remove(i - 2);
                            b.getBraid().add(i, tmp);
                            if (!isInTree(b, rewriteTree)) {
                                node.addChild(new TreeNode(new Braid(b)));
                            }
                            b = new Braid(node.getBraid());
                        }
                    }
                }
            }
        }
        for (TreeNode t : node.getChild()) {
            buildRewriteTree(t);
        }
    }

    private boolean isInTree(Braid braid, TreeNode node) {
        if (algorithms.isSame(braid, node.getBraid())) {
            return true;
        } else if (node.getChild().size() == 0) {
            return false;
        }
        for (TreeNode t : node.getChild()) {
            if (isInTree(braid, t)) {
                return true;
            }
        }
        return false;
    }

    private void buildRewriteTreeList(TreeNode node) {
        rewriteTreelist.add(node.getBraid());
        for (TreeNode t : node.getChild()) {
            buildRewriteTreeList(t);
        }
    }

    public String toString() {
        String tmp = "";
        for (int i = 0; i < braid.size(); i++) {
            tmp += braid.get(i).toString();
        }
        return tmp;
    }

    class TreeNode {

        private Braid braid;
        private LinkedList<TreeNode> childList;

        public TreeNode() {
            braid = null;
            childList = new LinkedList<TreeNode>();
        }

        public TreeNode(Braid braid) {
            this.braid = braid;
            childList = new LinkedList<TreeNode>();
        }

        public void addChild(TreeNode child) {
            childList.add(child);
        }
```

```
		public Braid getBraid() {
			return braid;
		}

		public LinkedList<TreeNode> getChild() {
			return childList;
		}

		public String toString() {
			return braid.toString() + childList.toString();
		}

	}

}
```

Delta.java

```
package construction;

import java.util.LinkedList;

import braidInterface.BraidWord;

/**
 * The braid word delta.
 *
 * @author Zhang Xi
 * @author 200906945
 * @version 06-05-2014
 */
public class Delta implements BraidWord {

	/** The index number of the braid word. */
	private int index;
	/** The inverse mark of the braid word. */
	private boolean isInverse;
	/** The half twist braid. */
	private Braid halfTwistBraid;
	/** Whether the braid has error. */
	private boolean error;
	/** The braid word is not sigma. */
	private boolean isSigma = false;

	/** The string symbol of delta. */
	private static final String deltaSymbol = "Δ";

	/**
	 * Create the delta.
	 *
	 */
	public Delta() {
		index = 0;
		isInverse = false;
		error = true;
		halfTwistBraid = null;
	}

	/**
	 * Create the delta.
	 *
	 * @param index
	 *            the index number
	 * @param isInverse
	 *            the inverse mark
	 */
	public Delta(int index, boolean isInverse) {
		this.index = index;
		this.isInverse = isInverse;
		error = false;
		halfTwistBraid = null;
	}

	/**
	 * Create the delta.
	 *
	 * @param braidWord
	 *            the input braid words
	 */
	public Delta(String braidWord) {
		error = false;
		halfTwistBraid = null;
		String[] bArray = braidWord.split("-");
		if (bArray.length <= 2) {
			try {
				index = Integer.parseInt(bArray[0]);
				if (bArray.length == 2 && bArray[1].equals("1")) {
					isInverse = true;
				} else if (bArray.length == 1) {
					isInverse = false;
				} else {
					error = true;
				}
			} catch (Exception e) {
				error = true;
			}
			return;
		}
	}

	/**
	 * Get the index number.
	 *
	 * @return the index number
	 */
	public int getIndex() {
		return index;
	}

	/**
	 * Set the index number of the braid.
	 *
	 * @param index
	 *            the index number
	 */
	public void setIndex(int index) {
		this.index = index;
	}

	/**
	 * Whether the braid is in inverse case.
	 *
	 * @return whether is inverse
	 */
	public boolean isInverse() {
        return isInverse;
    }

    /**
     * Change the braid into the opposite inverse mark
     *
     */
    public void oppositeInverse() {
        isInverse = !isInverse;
    }

    /**
     * Calculate the half twist rewrite braid.
     *
     * @return the half twist rewrite braid
     */
    private LinkedList<BraidWord> caculateBraid() {
        LinkedList<BraidWord> tmp = new LinkedList<BraidWord>();
        for (int i = getIndex() - 1; i >= 1; i--) {
            for (int j = 1; j <= i; j++) {
                tmp.add(new Sigma(j, isInverse()));
            }
        }
        halfTwistBraid = new Braid(tmp);
        return tmp;
    }

    /**
     * Return the half twist rewrite.
     *
     * @return the rewrite form braid
     */
    public Braid getBraid() {
        if (halfTwistBraid == null) {
            caculateBraid();
        }
        return halfTwistBraid;
    }

    /**
     * Whether is the braid is error.
     *
```

```
	 * @return whether the braid is error
	 */
	public boolean isError() {
		return error;
	}

	/**
	 * Whether the braid word is sigma.
	 *
	 * @return whether the braid word is sigma
	 */
	public boolean isSigma() {
		return isSigma;
	}

	/**
	 * Get the string case of the braid.
	 *
	 * @return the string braid word.
	 */
	public String toString() {
		String inverse = "";
		if (isInverse()) {
			inverse = "-1";
		}
		return deltaSymbol + index + inverse;
	}

}
```

Sigma.java

```
package construction;

import braidInterface.BraidWord;

/**
 * The braid word sigma.
 *
 * @author Zhang Xi
 * @author 200906945
 * @version 06-05-2014
 */
public class Sigma implements BraidWord {

	/** The index number of the braid word. */
	private int index;
	/** The inverse mark of the braid word. */
	private boolean isInverse;
	/** Whether the braid has error. */
	private boolean error = true;
	/** The braid word is not sigma. */
	private boolean isSigma = true;

	/** The string symbol of sigma. */
	private static final String sigmaSymbol = "σ";

	/**
	 * Create the sigma.
	 *
	 */
	public Sigma() {
		index = 0;
		isInverse = false;
		error = true;
	}

	/**
	 * Create the sigma.
	 *
	 * @param index
	 *            the index number
	 * @param isInverse
	 *            the inverse mark
	 */
	public Sigma(int index, boolean isInverse) {
		this.index = index;
		this.isInverse = isInverse;
		error = false;
	}

	/**
	 * Create the sigma.
	 *
	 * @param braidWord
	 *            the input braid words
	 */
	public Sigma(String braidWord) {
		error = false;
		String[] bArray = braidWord.split("-");
		if (bArray.length <= 2) {
			try {
				index = Integer.parseInt(bArray[0]);
				if (bArray.length == 2 && bArray[1].equals("1")) {
					isInverse = true;
				} else if (bArray.length == 1) {
					isInverse = false;
				} else {
					error = true;
				}
			} catch (Exception e) {
				error = true;
			}
			return;
		}
	}

	/**
	 * Get the index number.
	 * 
	 * @return the index number
	 */
	public int getIndex() {
		return index;
	}

	/**
	 * Set the index number of the braid.
	 * 
	 * @param index
	 *            the index number
	 */
	public void setIndex(int index) {
		this.index = index;
	}
	/**
	 * Whether the braid is in inverse case.
	 *
	 * @return whether is inverse
	 */
	public boolean isInverse() {
		return isInverse;
	}

	/**
	 * Change the braid into the opposite inverse mark
	 *
	 */
	public void oppositeInverse() {
		isInverse = !isInverse;
	}

	/**
	 * Whether is the braid is error.
	 *
	 * @return whether the braid is error
	 */
	public boolean isError() {
		return error;
	}

	/**
	 * Whether the braid word is sigma.
	 *
	 * @return whether the braid word is sigma
	 */
	public boolean isSigma() {
		return isSigma;
	}

	/**
	 * Get the string case of the braid.
	 *
	 * @return the string braid word.
	 */
	public String toString() {
		String inverse = "";
```

```
            if (isInverse()) {
                  inverse = "-1";
            }
            return sigmaSymbol + index + inverse;
      }

}
```

GUI.java

```
package GUI;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics2D;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.ListModel;
import javax.swing.ListSelectionModel;
import javax.swing.WindowConstants;
import javax.swing.SwingUtilities;

import algorithm.Algorithms;
import visualization.BraidView;
import construction.Braid;
```

```
/**
  * This code was edited or generated using CloudGarden's Jigloo SWT/Swing GUI
  * Builder, which is free for non-commercial use. If Jigloo is being used
  * commercially (ie, by a corporation, company or business for any purpose
  * whatever) then you should purchase a license for each developer using Jigloo.
  * Please visit www.cloudgarden.com for details. Use of Jigloo implies
  * acceptance of these licensing terms. A COMMERCIAL LICENSE HAS NOT BEEN
  * PURCHASED FOR THIS MACHINE, SO JIGLOO OR THIS CODE CANNOT BE
USED LEGALLY FOR
  * ANY CORPORATE OR COMMERCIAL PURPOSE.
  */
public class GUI extends JFrame {
	/**
	 *
	 */
	private Algorithms algorithms;
	private static final long serialVersionUID = 1L;
	private JPanel view;
	private JTabbedPane visualization;
	private JPanel compare;
	private JPanel rewrite;
	private JList<String> rewriteResult;
	private JButton compareConfirm;
	private JScrollPane rewriteListScrollPane;
	private JScrollPane rewriteScrollPane;
	private JScrollPane compareScrollPane2;
	private JScrollPane compareScrollPane1;
	private JScrollPane viewScrollPane;
	private JPanel compareBraidView2;
	private JPanel compareBraidView1;
	private JPanel rewriteBraidView;
	private JPanel viewBraidView;
	private JTextField viewSigmaPositiveResult;
	private JButton viewSigmaPositive;
	private JButton viewGarsideNomalForm;
	private JTextField viewBraidOutput;
	private JTextField compareResult;
	private JTextField compareInput2;
	private JTextField compareInput1;
	private JButton rewriteConfirm;
	private JTextField rewriteInput;
	private JButton viewConfirm;
	private JTextField viewInput;

	private static final String inputFomat = "([s|σ|d|Δ][1-9]{1}[0-9]*([\\-][1])?)+";//
([s|σ|d|Δ][1-9]{1}[0-9]+([\-][1])?)+
	private static final String errorMessage = "The input should be in correct format.
\r\neg: s1-1d1σ2Δ3-1";
	private JTextField compareExperimentBraidLength;
	private JTextField compareExperimentTextField2;
	private JTextField compareExperimentstrandsNumber;
	private JTextField compareExperimentTextField1;
	private JButton compareExperimentConfirm;
	private JList<String> compareExperimentList;
	private JTextField textField2;
	private JTextField textField1;
	private JPanel compareExperimentPanel;
	private JScrollPane compareExperimentScrollPane;
	private JScrollPane compareExperimentListScrollPane;
	private JTextField viewSigmaPositiveRewriteTimes;
	private JPanel compareExperiment;
	private JButton viewBraidSave;

	/**
	 * Auto-generated main method to display this JFrame
	 */
	public static void main(String[] args) {
		SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				GUI inst = new GUI();
				inst.setLocationRelativeTo(null);
				inst.setVisible(true);
				inst.setPreferredSize(new java.awt.Dimension(663, 458));
				inst.setSize(663, 458);
			}
		});
	}

	public GUI() {
		super();
		initGUI();
		algorithms = new Algorithms();
	}
```

```
private void initGUI() {
	try {
		setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
		this.setTitle("Braids and Braid Words");
		this.setPreferredSize(new java.awt.Dimension(662, 458));
		this.setResizable(false);
		{
			visualization = new JTabbedPane();
			getContentPane().add(visualization, BorderLayout.CENTER);
			visualization
					.setPreferredSize(new java.awt.Dimension(660, 429));
			{
				view = new JPanel();
				visualization.addTab("View", null, view, null);
				view.setLayout(null);
				{
					viewInput = new JTextField();
					view.add(viewInput);
					viewInput.setBounds(23, 28, 375, 32);
				}
				{
					viewConfirm = new JButton();
					view.add(viewConfirm);
					viewConfirm.setText("Confirm");
					viewConfirm.setBounds(428, 28, 190, 31);
					viewConfirm.addMouseListener(new MouseAdapter() {
						public void mouseClicked(MouseEvent arg0) {
							String braidText = viewInput.getText();
							if (!braidText.matches(inputFomat)
									|| braidText == "") {
								JOptionPane.showMessageDialog(null,
										errorMessage);
								return;
							}
							view.remove(viewBraidOutput);
							viewBraidOutput = new JTextField();
							view.add(viewBraidOutput);
							viewBraidOutput.setBounds(23, 77, 375, 31);
							viewBraidOutput.setEditable(false);
							Braid braid = new Braid(braidText);
							viewScrollPane.remove(viewBraidView);
```

```
                                viewBraidView = new BraidView(braid);
                                viewScrollPane.setViewportView(viewBraidView);
                                viewSigmaPositiveResult = new JTextField();
                                view.add(viewSigmaPositiveResult);
                                viewSigmaPositiveResult.setBounds(566, 189, 52,
                                        24);
                                viewSigmaPositiveResult.setEditable(false);
                                viewSigmaPositiveRewriteTimes = new
JTextField();
                                view.add(viewSigmaPositiveRewriteTimes);
                                viewSigmaPositiveRewriteTimes
                                        .setEditable(false);
                                viewSigmaPositiveRewriteTimes.setBounds(566,
                                        232, 52, 24);
                            }
                        });
                    }
                    {
                        viewBraidOutput = new JTextField();
                        view.add(viewBraidOutput);
                        viewBraidOutput.setBounds(23, 77, 375, 31);
                        viewBraidOutput.setEditable(false);
                    }
                    {
                        viewGarsideNomalForm = new JButton();
                        view.add(viewGarsideNomalForm);
                        viewGarsideNomalForm.setText("Garside Nomal Form");
                        viewGarsideNomalForm.setBounds(428, 75, 190, 33);
                        viewGarsideNomalForm
                                .addMouseListener(new MouseAdapter() {
                                    public void mouseClicked(MouseEvent arg0) {
                                        String braidText = viewInput.getText();
                                        if (!braidText.matches(inputFomat)
                                                || braidText == "") {

    JOptionPane.showMessageDialog(null,
                                                        errorMessage);
                                            return;
                                        }
                                        Braid tmpBraid = new Braid(braidText);
                                        Braid braid = algorithms
                                                .garsideNormalForm(tmpBraid);
```

```
                                                                            view.remove(viewBraidOutput);
                                                                            viewBraidOutput = new JTextField();
                                                                            view.add(viewBraidOutput);
                                                                            if (!tmpBraid.hasHalfTwistForm()) {
                                                                                    viewBraidOutput.setText(braid
                                                                                                    .toString());
                                                                                    viewScrollPane
                                                                                                    .remove(viewBraidView);
                                                                                    viewBraidView = new
BraidView(braid);
                                                                                    viewScrollPane

        .setViewportView(viewBraidView);
                                                                            } else {
                                                                                    viewBraidOutput
                                                                                                    .setText("The braid should
not have the halftwist form braid word(Δ)!");
                                                                            }
                                                                            viewBraidOutput.setBounds(23, 77, 375,
                                                                                            31);
                                                                            viewBraidOutput.setEditable(false);
                                                                            viewSigmaPositiveResult = new
JTextField();
                                                                            view.add(viewSigmaPositiveResult);
                                                                            viewSigmaPositiveResult.setBounds(566,
                                                                                            189, 52, 24);
                                                                            viewSigmaPositiveResult
                                                                                            .setEditable(false);
                                                                            viewSigmaPositiveRewriteTimes = new
JTextField();

        view.add(viewSigmaPositiveRewriteTimes);
                                                                            viewSigmaPositiveRewriteTimes
                                                                                            .setEditable(false);
                                                                            viewSigmaPositiveRewriteTimes
                                                                                            .setBounds(566, 232, 52, 24);
                                                                    }
                                                            });
                                    }
                                    {
                                            viewSigmaPositive = new JButton();
                                            view.add(viewSigmaPositive);
```

```
                              viewSigmaPositive.setText("Sigma Positive Rewrite");
                              viewSigmaPositive.setBounds(428, 127, 190, 32);
                              viewSigmaPositive.addMouseListener(new MouseAdapter() {
                                   public void mouseClicked(MouseEvent arg0) {
                                        String braidText = viewInput.getText();
                                        if (!braidText.matches(inputFomat)
                                                  || braidText == "") {
                                             JOptionPane.showMessageDialog(null,
                                                       errorMessage);
                                             return;
                                        }
                                        Braid tmpBraid = new Braid(braidText);
                                        Braid braid = algorithms
                                                  .sigmaPositiveForm(tmpBraid);
                                        view.remove(viewBraidOutput);
                                        viewBraidOutput = new JTextField();
                                        view.add(viewBraidOutput);
                                        viewBraidOutput.setText(braid.toString());
                                        viewScrollPane.remove(viewBraidView);
                                        viewBraidView = new BraidView(braid);
                                        viewScrollPane.setViewportView(viewBraidView);
                                        viewSigmaPositiveResult = new JTextField();
                                        view.add(viewSigmaPositiveResult);
                                        viewSigmaPositiveResult.setBounds(566, 189, 52,
                                                  24);
                                        if (algorithms.isSigmaPositive(tmpBraid)) {
                                             viewSigmaPositiveResult.setText("Yes");
                                        } else {
                                             viewSigmaPositiveResult.setText("No");
                                        }
                                        viewSigmaPositiveResult.setEditable(false);
                                        viewSigmaPositiveRewriteTimes = new
JTextField();
                                        view.add(viewSigmaPositiveRewriteTimes);
                                        viewSigmaPositiveRewriteTimes.setBounds(566,
                                                  232, 52, 24);
                                        viewSigmaPositiveRewriteTimes.setText(braid
                                                  .getRewriteTimes() + "");
                                        viewSigmaPositiveRewriteTimes
                                                  .setEditable(false);
                                        viewBraidOutput.setBounds(23, 77, 375, 31);
                                        viewBraidOutput.setEditable(false);
```

```
                }
            });
        }
        {
            viewSigmaPositiveResult = new JTextField();
            view.add(viewSigmaPositiveResult);
            viewSigmaPositiveResult.setBounds(566, 189, 52, 24);
            viewSigmaPositiveResult.setEditable(false);
        }
        {
            viewScrollPane = new JScrollPane();
            view.add(viewScrollPane);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            viewScrollPane.setBounds(23, 136, 375, 233);
            {
                viewBraidView = new JPanel();
                viewScrollPane.setViewportView(viewBraidView);
                viewBraidView.setBackground(new java.awt.Color(255,
                        255, 255));
                viewBraidView.setPreferredSize(new Dimension(0, 0));
            }
        }
        {
            viewBraidSave = new JButton();
            view.add(viewBraidSave);
            viewBraidSave.setText("Export the Braid Picture");
            viewBraidSave.setBounds(421, 337, 197, 32);
            viewBraidSave.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent arg0) {
                    Dimension imageSize = viewBraidView.getSize();
                    BufferedImage image = new BufferedImage(
                            imageSize.width, imageSize.height,
                            BufferedImage.TYPE_INT_ARGB);
                    File file = new File("braid.png");
                    Graphics2D g = image.createGraphics();
                    viewBraidView.paint(g);
                    g.dispose();
                    try {
                        if (!file.exists()) {
                            try {
                                file.delete();
                                file.createNewFile();
```

```
                                                } catch (IOException e) {
                                                        // TODO Auto-generated catch block
                                                        e.printStackTrace();
                                                }
                                        }
                                        ImageIO.write(image, "png", file);
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                });
        }
        {
                viewSigmaPositiveRewriteTimes = new JTextField();
                view.add(viewSigmaPositiveRewriteTimes);
                viewSigmaPositiveRewriteTimes.setEditable(false);
                viewSigmaPositiveRewriteTimes.setBounds(566, 232, 52,
                                24);
        }
        {
                textField1 = new JTextField();
                view.add(textField1);
                textField1.setText("\u03c3-positive braid:");
                textField1.setBounds(428, 189, 105, 24);
                textField1.setEditable(false);
                textField1.setBorder(BorderFactory.createEmptyBorder(0,
                                0, 0, 0));
        }
        {
                textField2 = new JTextField();
                view.add(textField2);
                textField2.setText("Handle removed times:");
                textField2.setBounds(428, 232, 138, 24);
                textField2.setEditable(false);
                textField2.setBorder(BorderFactory.createEmptyBorder(0,
                                0, 0, 0));
        }
}
{
        rewrite = new JPanel();
        visualization.addTab("Rewrite", null, rewrite, null);
```

```
                    rewrite.setPreferredSize(new java.awt.Dimension(564, 367));
                    rewrite.setLayout(null);
                    {
                        rewriteInput = new JTextField();
                        rewrite.add(rewriteInput);
                        rewriteInput.setBounds(26, 24, 281, 34);
                    }
                    {
                        rewriteConfirm = new JButton();
                        rewrite.add(rewriteConfirm);
                        rewriteConfirm.setText("Confirm");
                        rewriteConfirm.setBounds(341, 24, 113, 34);
                        rewriteConfirm.addMouseListener(new MouseAdapter() {
                            public void mouseClicked(MouseEvent arg0) {
                                String braidText = rewriteInput.getText();
                                if (!braidText.matches(inputFomat)
                                        || braidText == "") {
                                    JOptionPane.showMessageDialog(null,
                                            errorMessage);
                                    return;
                                }
                                Braid braid = new Braid(braidText);
                                rewriteScrollPane.remove(rewriteBraidView);
                                rewriteBraidView = new BraidView(braid);
                                rewriteScrollPane
                                        .setViewportView(rewriteBraidView);
                                rewriteListScrollPane.remove(rewriteResult);
                                ListModel<String> rewriteResultModel = new
DefaultComboBoxModel<String>(
                                        braid.rewriteFormsStringList());
                                rewriteResult = new JList<String>();
                                rewriteListScrollPane
                                        .setViewportView(rewriteResult);
                                rewriteResult.setModel(rewriteResultModel);
                                rewriteResult.setBounds(26, 80, 281, 303);
                                rewriteResult

    .setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                                rewriteResult
                                        .addMouseListener(new MouseAdapter() {
                                            public void mouseClicked(
                                                    MouseEvent e) {
                                                            String braidText = rewriteResult
                                                                    .getSelectedValue();
                                                            Braid braid = new Braid(
                                                                    braidText);
                                                            rewriteScrollPane

    .remove(rewriteBraidView);
                                                            rewriteBraidView = new
BraidView(
                                                                    braid);
                                                            rewriteScrollPane

    .setViewportView(rewriteBraidView);
                                                        }
                                                    });
                                        }
                                    });
                                }
                                {
                                    rewriteScrollPane = new JScrollPane();
                                    rewrite.add(rewriteScrollPane);
                                    rewriteScrollPane.setBounds(341, 80, 278, 303);
                                    {
                                        rewriteBraidView = new JPanel();
                                        rewriteScrollPane.setViewportView(rewriteBraidView);
                                        rewriteBraidView.setBounds(105, 69, 278, 303);
                                        rewriteBraidView.setBackground(new java.awt.Color(
                                                255, 255, 255));
                                        rewriteBraidView
                                                .setPreferredSize(new java.awt.Dimension(
                                                        30, 30));
                                    }
                                }
                                {
                                    rewriteListScrollPane = new JScrollPane();
                                    rewrite.add(rewriteListScrollPane);
                                    rewriteListScrollPane.setBounds(26, 80, 281, 303);
                                    {
                                        ListModel<String> rewriteResultModel = new
DefaultComboBoxModel<String>();
                                        rewriteResult = new JList<String>();
                                        rewriteListScrollPane
                                                .setViewportView(rewriteResult);
                                        rewriteResult.setModel(rewriteResultModel);
                                        rewriteResult.setBounds(141, 19, 232, 196);
                                        rewriteResult

    .setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                                    }
                                }
                            }
                            {
                                compare = new JPanel();
                                visualization.addTab("Compare", null, compare, null);
                                compare.setLayout(null);
                                {
                                    compareInput1 = new JTextField();
                                    compare.add(compareInput1);
                                    compareInput1.setBounds(33, 30, 201, 28);
                                }
                                {
                                    compareInput2 = new JTextField();
                                    compare.add(compareInput2);
                                    compareInput2.setBounds(403, 31, 201, 28);
                                }
                                {
                                    compareResult = new JTextField();
                                    compare.add(compareResult);
                                    compareResult.setBounds(299, 31, 34, 26);
                                    compareResult.setEditable(false);
                                }
                                {
                                    compareConfirm = new JButton();
                                    compare.add(compareConfirm);
                                    compareConfirm.setText("Confirm");
                                    compareConfirm.setBounds(267, 84, 97, 35);
                                    compareConfirm.addMouseListener(new MouseAdapter() {
                                        public void mouseClicked(MouseEvent arg0) {
                                            String braidText1 = compareInput1.getText();
                                            String braidText2 = compareInput2.getText();
                                            if (!braidText1.matches(inputFomat)
                                                    || braidText1 == "") {
                                                JOptionPane.showMessageDialog(null,
                                                        errorMessage);
                            return;
                        }
                        if (!braidText2.matches(inputFomat)
                                || braidText2 == "") {
                            JOptionPane.showMessageDialog(null,
                                    errorMessage);
                            return;
                        }
                        Braid braid1 = new Braid(braidText1);
                        Braid braid2 = new Braid(braidText2);
                        compareScrollPane1.remove(compareBraidView1);
                        compareBraidView1 = new BraidView(braid1);
                        compareScrollPane1
                                .setViewportView(compareBraidView1);
                        compareScrollPane2.remove(compareBraidView2);
                        compareBraidView2 = new BraidView(braid2);
                        compareScrollPane2
                                .setViewportView(compareBraidView2);
                        compare.remove(compareResult);
                        compareResult = new JTextField();
                        compare.add(compareResult);
                        compareResult.setBounds(299, 31, 34, 26);
                        compareResult.setEditable(false);
                        if (algorithms.isEqual(braid1, braid2)) {
                            compareResult.setText("=");
                        } else if (algorithms.isSmaller(braid1, braid2)) {
                            compareResult.setText("<");
                        } else {
                            compareResult.setText(">");
                        }
                    }
                });
            }
            {
                compareScrollPane1 = new JScrollPane();
                compare.add(compareScrollPane1);
                compareScrollPane1.setBounds(33, 90, 201, 274);
                {
                    compareBraidView1 = new JPanel();
                    compareScrollPane1
                            .setViewportView(compareBraidView1);
                    compareBraidView1.setBounds(262, 90, 201, 274);
```

```
                                    compareBraidView1.setBackground(new java.awt.Color(
                                            255, 255, 255));
                                }
                            }
                            {
                                compareScrollPane2 = new JScrollPane();
                                compare.add(compareScrollPane2);
                                compareScrollPane2.setBounds(403, 90, 201, 274);
                                {
                                    compareBraidView2 = new JPanel();
                                    compareScrollPane2
                                            .setViewportView(compareBraidView2);
                                    compareBraidView2.setBounds(173, 90, 201, 274);
                                    compareBraidView2.setBackground(new java.awt.Color(
                                            255, 255, 255));
                                    compareBraidView2
                                            .setPreferredSize(new java.awt.Dimension(
                                                    20, 27));
                                }
                            }
                        }
                        {
                            compareExperiment = new JPanel();
                            visualization.addTab("Compare Experiment", null,
                                    compareExperiment, null);
                            compareExperiment.setLayout(null);
                            {
                                compareExperimentListScrollPane = new JScrollPane();
                                compareExperiment.add(compareExperimentListScrollPane);
                                compareExperimentListScrollPane.setBounds(26, 80, 281,
                                        303);
                                {
                                    ListModel<String> compareExperimentListModel = new
DefaultComboBoxModel<String>();
                                    compareExperimentList = new JList<String>();
                                    compareExperimentListScrollPane
                                            .setViewportView(compareExperimentList);
                                    compareExperimentList
                                            .setModel(compareExperimentListModel);
                                }
                            }
                            {
                                compareExperimentScrollPane = new JScrollPane();
                                compareExperiment.add(compareExperimentScrollPane);
                                compareExperimentScrollPane
                                        .setBounds(341, 80, 281, 303);
                                {
                                    compareExperimentPanel = new JPanel();
                                    FlowLayout compareExperimentPanelLayout = new
FlowLayout();
                                    compareExperimentScrollPane
                                            .setViewportView(compareExperimentPanel);
                                    compareExperimentPanel
                                            .setLayout(compareExperimentPanelLayout);
                                    compareExperimentPanel
                                            .setPreferredSize(new java.awt.Dimension(
                                                    278, 300));
                                    compareExperimentPanel
                                            .setBackground(new java.awt.Color(255, 255,
                                                    255));
                                }
                            }
                            {
                                compareExperimentConfirm = new JButton();
                                compareExperiment.add(compareExperimentConfirm);
                                compareExperimentConfirm.setText("Confirm");
                                compareExperimentConfirm.setBounds(525, 22, 97, 35);
                                compareExperimentConfirm
                                        .addMouseListener(new MouseAdapter() {
                                            public void mouseClicked(MouseEvent arg0) {
                                                String strandNumber =
compareExperimentstrandsNumber
                                                        .getText();
                                                String braidLength =
compareExperimentBraidLength
                                                        .getText();
                                                if (!strandNumber.matches("^[1-9]\\d*$")
                                                        || strandNumber == "") {

    JOptionPane.showMessageDialog(null,
                                                            "Input a number.");
                                                    return;
                                                }
                                                if (!braidLength.matches("^[1-9]\\d*$")
                                                            || braidLength == "") {

    JOptionPane.showMessageDialog(null,
                                                                "Input a number.");
                                                      return;
                                                  }
                                                  String[] braidList =
algorithms.compareStringList(
                                                            Integer.parseInt(strandNumber),
                                                            Integer.parseInt(braidLength));
                                                  compareExperimentScrollPane

    .remove(compareExperimentPanel);
                                                  compareExperimentScrollPane

    .setViewportView(compareExperimentPanel);
                                                  compareExperimentListScrollPane

    .remove(compareExperimentList);
                                                  ListModel<String>
compareExperimentListModel = new DefaultComboBoxModel<String>(
                                                            braidList);
                                                  compareExperimentList = new
JList<String>();
                                                  compareExperimentListScrollPane

    .setViewportView(compareExperimentList);
                                                  compareExperimentList

    .setModel(compareExperimentListModel);
                                                  compareExperimentList

    .setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                                                  compareExperimentList
                                                            .addMouseListener(new
MouseAdapter() {
                                                                public void mouseClicked(
                                                                          MouseEvent e) {
                                                                     String braidText =
compareExperimentList

    .getSelectedValue();
```

```
                                                                    Braid braid = new
Braid(
                                                                            braidText);

    compareExperimentScrollPane

    .remove(compareExperimentPanel);

    compareExperimentPanel = new BraidView(
                                                                            braid);

    compareExperimentScrollPane

    .setViewportView(compareExperimentPanel);
                                                                }
                                                            });
                                                }
                                            });
                                }
                                {
                                    compareExperimentTextField1 = new JTextField();
                                    compareExperiment.add(compareExperimentTextField1);
                                    compareExperimentTextField1
                                            .setText("The strands number:");
                                    compareExperimentTextField1.setBounds(26, 22, 126, 35);
                                    compareExperimentTextField1.setEditable(false);
                                    compareExperimentTextField1.setBorder(BorderFactory
                                            .createEmptyBorder(0, 0, 0, 0));
                                }
                                {
                                    compareExperimentstrandsNumber = new JTextField();
                                    compareExperiment.add(compareExperimentstrandsNumber);
                                    compareExperimentstrandsNumber.setBounds(164, 22, 65,
                                            35);
                                }
                                {
                                    compareExperimentTextField2 = new JTextField();
                                    compareExperiment.add(compareExperimentTextField2);
                                    compareExperimentTextField2
                                            .setText("The length of the braid");
                                    compareExperimentTextField2.setBounds(270, 23, 140, 35);
                                    compareExperimentTextField2.setEditable(false);
```

```
                                        compareExperimentTextField2.setBorder(BorderFactory
                                                    .createEmptyBorder(0, 0, 0, 0));
                                    }
                                    {
                                        compareExperimentBraidLength = new JTextField();
                                        compareExperiment.add(compareExperimentBraidLength);
                                        compareExperimentBraidLength.setBounds(422, 23, 65, 35);
                                    }
                              }
                        }
                        pack();
                        this.setSize(662, 458);
                  } catch (Exception e) {
                        // add your error handling code here
                        e.printStackTrace();
                  }
      }
}
```

BraidView.java

```
package visualization;

import java.awt.Dimension;
import java.awt.Graphics;

import javax.swing.JPanel;

import braidInterface.BraidWord;
import construction.Braid;

/**
  * The class support the braid visualization.
  *
  * @author Zhang Xi
  * @author 200906945
  * @version 06-05-2014
  */
public class BraidView extends JPanel {

      private static final long serialVersionUID = 1L;
      private Braid braid;
```

```
    private final static int bound = 10;
    private final static int singleLength = 30;

    public BraidView( Braid braid) {
        setPreferredSize(new Dimension((braid.getStrandsNum()) * singleLength,
                braid.getBraid().size() * singleLength + bound * 2));
        setBackground(new java.awt.Color(255, 255, 255));
        this.braid = braid;
    }

    public void paintBorder(Graphics g) {
        for (int i = 0; i < braid.getBraid().size(); i++) {
            BraidWord tmpBraidWord = braid.getBraid().get(i);
            if (tmpBraidWord.isSigma()) {
                for (int j = 1; j <= braid.getStrandsNum(); j++) {
                    if (tmpBraidWord.getIndex() == j) {
                        if (!tmpBraidWord.isInverse()) {
                            g.drawLine(bound + singleLength * (j - 1), bound
                                    + singleLength * i, bound + singleLength
                                    * j, bound + singleLength * (i + 1));
                        } else {
                            g.drawLine(bound + singleLength * j, bound
                                    + singleLength * i, bound + singleLength
                                    * (j - 1), bound + singleLength * (i + 1));
                        }
                    } else if (tmpBraidWord.getIndex() == j - 1) {
                        if (!tmpBraidWord.isInverse()) {
                            g.drawLine(bound + singleLength * (j - 1), bound
                                    + singleLength * i, bound + singleLength
                                    * (j - 1) - singleLength / 3, bound
                                    + singleLength * i + singleLength / 3);
                            g.drawLine(bound + singleLength * (j - 1)
                                    - singleLength / 3 * 2, bound
                                    + singleLength * i + singleLength / 3 * 2,
                                    bound + singleLength * ((j - 1) - 1), bound
                                            + singleLength * (i + 1));
                        } else {
                            g.drawLine(bound + singleLength * ((j - 1) - 1),
                                    bound + singleLength * i, bound
                                            + singleLength * ((j - 1) - 1)
                                            + singleLength / 3, bound
                                            + singleLength * i + singleLength
```

```
                                                                          / 3);
                                            g.drawLine(bound + singleLength * ((j - 1) - 1)
                                                      + singleLength / 3 * 2, bound
                                                      + singleLength * i + singleLength / 3 * 2,
                                                      bound + singleLength * (j - 1), bound
                                                                + singleLength * (i + 1));
                                   }
                              } else {
                                   g.drawLine(bound + singleLength * (j - 1), bound
                                             + singleLength * i, bound + singleLength
                                             * (j - 1), bound + singleLength * (i + 1));
                              }
                         }
                    } else {
                         if (!tmpBraidWord.isInverse()) {
                              g.drawLine(bound, bound + singleLength * i, bound
                                        + singleLength * (braid.getStrandsNum() - 1), bound
                                        + singleLength * (i + 1));
                              g.drawLine(bound + singleLength
                                        * (braid.getStrandsNum() - 1), bound + singleLength
                                        * i, bound + singleLength
                                        * (braid.getStrandsNum() - 1) - singleLength
                                        * (braid.getStrandsNum() - 1) / 3, bound
                                        + singleLength * i + singleLength / 3);
                              g.drawLine(bound + singleLength
                                        * (braid.getStrandsNum() - 1) - singleLength
                                        * (braid.getStrandsNum() - 1) / 3 * 2, bound
                                        + singleLength * i + singleLength / 3 * 2, bound,
                                        bound + singleLength * (i + 1));
                         } else {
                              g.drawLine(bound + singleLength
                                        * (braid.getStrandsNum() - 1), bound + singleLength
                                        * i, bound, bound + singleLength * (i + 1));
                              g.drawLine(bound, bound + singleLength * i, bound
                                        + singleLength * (braid.getStrandsNum() - 1)
                                        - singleLength * (braid.getStrandsNum() - 1) / 3
                                        * 2, bound + singleLength * i + singleLength / 3);
                              g.drawLine(bound + singleLength
                                        * (braid.getStrandsNum() - 1) - singleLength
                                        * (braid.getStrandsNum() - 1) / 3, bound
                                        + singleLength * i + singleLength / 3 * 2, bound
                                        + singleLength * (braid.getStrandsNum() - 1), bound
```

```
                                                + singleLength * (i + 1));
                            }
                        }
                    }
                }

}
```