TRABAJO FIN DE GRADO

Doble Grado en Ingeniería Informática y Matemáticas

Entendiendo la teoría de nudos mediante la simulación y la informática gráfica.

Autora
Cristina Zuheros Montes.

Tutores
Alejandro J. León Salas
Antonio Martínez López

Granada, Diciembre de 2016

Entendiendo la teoría de nudos mediante la simulación y la informática gráfica.

Cristina Zuheros Montes.

Palabras clave: nudo, trenza, handle...PONER LAS PALABRAS CLAVE.

Resumen

//HACER RESUMEN AQUI!

PONER TITULO EN INGLES!!!

Cristina Zuheros Montes.

Keywords: knot, braid, handle....PONER AQUI KEYWORDS!!!!!

Abstract

//PONER RESUMEN DE MILLON DE PALABRAS INGLES.!!!!!!

Índice general

Índice de figuras

Capítulo 1

Introducción

HACER INTRO AQUIII!!!

1.1 Objetivos

En la propuesta inicial se propusieron los siguientes objetivos:

1. Estudiar...//PONER LOS OBJETIVOS INICIALES

   //PONER DONDE SE CUBREN LOS OBJETIVOS

Capítulo 2

Teoría de nudos.

Capítulo 3

Teoría de trenzas.

3.1 El problema de las palabras.

En esta sección vamos a tratar de ver si dos trenzas dadas son equivalentes entre sí sin hacer uso de invariantes. Vamos a apoyarnos en la idea que vimos en la sección **??**:

Consideramos el grupo B_n con la representación que vimos en el teorema **??**. El problema de ver si dos trenzas dadas son equivalentes se puede ver como el problema de las palabras del grupo B_n.

Problema de las palabras del grupo de las trenzas:
Dadas dos palabras de trenzas $\beta 1, \beta 2 \in B_n$ tratamos de encontrar algún método que nos permita confirmar si son o no equivalentes.

Ver si dos palabras (de trenzas) dadas $\beta 1, \beta 2 \in B_n$ son equivalentes se pude ver cómo el problema de ver si la palabra $\beta 1 \beta 2^{-1}$ es equivalente a la palabra vacía (trenza trivial). Por tanto el problema de las palabras se puede reducir a distinguir si una palabra es equivalente o no a la palabra vacía.

En este proyecto vamos a ver el método de Patrick Dehornoy que nos permite reducir las palabras de trenzas dadas hasta una forma específica que nos permitirá ver si la palabra es igual a la cadena vacía. Para verlo con más detalle necesitamos ver algunas ideas previas.

Definición:
Diremos que una palabra es **libremente reducida** si no contiene secuencias de la forma $\sigma_i^{-1}\sigma_i$ ni $\sigma_i\sigma_i^{-1}$.

Por ejemplo, la palabra $\sigma 2\sigma 1^{-1}\sigma 1\sigma 3^{-1}$ que representa a la trenza de la figura 3.1 no es libremente reducida pues contiene la palabra $\sigma 1^{-1}\sigma 1$.

Definición:
Sea la palabra $\beta \in B_n$. Llamaremos **generador principal** de β al generador de β de menor índice.

Figura 3.1: Trenza no libremente reducida.

Por ejemplo, la palabra $\sigma 2\sigma 1^{-1}\sigma 1\sigma 3^{-1}$ tiene como generador principal el 1, mientras que la palabra $\sigma 2\sigma 4^{-1}$ tiene como generador principal el número 2.

Definición: Diremos que una palabra $\beta \in B_n$ es **reducida** si se cumple alguna de estas condiciones:

1. β es la palabra cadena vacía.
2. El generador principal de β se presenta sólo positiva o negativamente.

Por ejemplo, la palabra $\sigma 2\sigma 1^{-1}\sigma 3^{-1}\sigma 1$ no es reducida pues el generador principal (1) se encuentra en un cruce negativo y en un cruce positivo. La palabra $\sigma 2\sigma 1\sigma 3^{-1}\sigma 1$ si sería reducida pues el generador principal (1) se presenta sólo como cruces positivos.

Si tenemos una palabra no reducida con generador principal σ_i, podemos considerar ocurrencias de la palabra de la forma $\sigma_i^{\pm}...\sigma_i^{\mp}$ que no contengan cruces σ_i, es decir, entre los cruces de signo opuesto del generador σ_i sólo encontramos generadores σ_k, $k > i$. Podemos visualizar la idea en la figura 3.2. En este caso diremos que la cadena i+1 forma a **handle**.

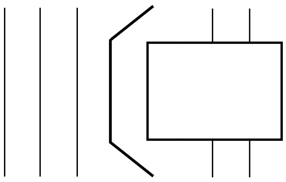

Figura 3.2: Main handle.

Proposición 3.1.1. *Una palabra reducida no vacía no puede ser equivalente a la cadena vacía.*

Por tanto, si encontramos un método para obtener la palabra reducida de una palabra, podremos resolver el problema de las palabras:
Si la palabra $\beta 1$ reducida es equivalente a la palabra $\beta 2$, entonces $\beta 2$ es equivalente a la cadena vacía si y sólo si $\beta 1$ es la cadena vacía.

Proposición 3.1.2. *Cualquier palabra admite una palabra reducida.*

Para ver una demostración de esta proposición, vamos ir viendo el método que realizaremos para transformar una palabra cualquiera en su palabra reducida.
Con este método tratamos de eliminar los handles de la trenza, pero para no entrar en bucles infinitos en el algoritmo, vamos a tener que considerar handles generados por cualquier generador y no sólo por el generador principal.

Definición:
Un σ_j**-handle** de una palabra es una sub-palabra de la forma $\sigma_j^{\pm} v \sigma_j^{\mp}$ donde la sub-palabra v sólo contiene generadores $\sigma_k^{(-1)}$ con k < j-1 o k >j. Si el generador σ_j es el generador principal de la palabra, a dicho σ_j-handle se conocerá como **main handle**.

Podemos ver un σ_j-handle en la figura 3.3.

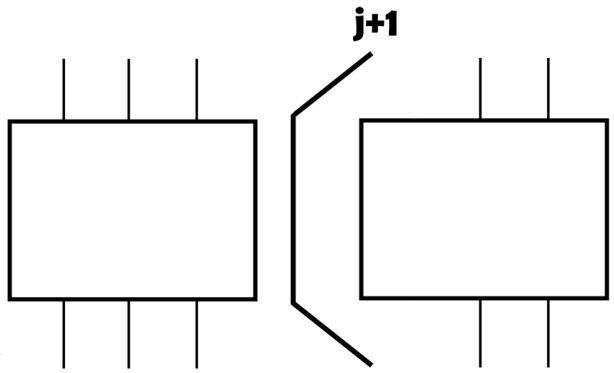

Figura 3.3: A handle.

Supongamos que tenemos la palabra $\sigma 3\sigma 1^{-1}\sigma 2\sigma 4^{-1}\sigma 2^{-1}\sigma 4^{-1}\sigma 1$.
La sub-palabra $\sigma 1^{-1}\sigma 2\sigma 4^{-1}\sigma 2^{-1}\sigma 4^{-1}\sigma 1$ es un main handle mientras que $\sigma 2\sigma 4^{-1}\sigma 2^{-1}$ es un $\sigma 2$-handle.

Definición:
Sea $\sigma_j^e\ v\ \sigma_j^{-e}$ un σ_j-handle de una palabra β, donde $e \in \{-1, 1\}$. En particular, podemos denotar dicho σ_j-handle como la siguiente sub-palabra:

$$\sigma_j^e\ v_0\ \sigma_{j+1}^{d_1}\ v_1 ... v_{m-1}\ \sigma_{j+1}^{d_m}\ v_m\ \sigma_j^{-e}$$

donde $v_0, ..., v_m$ no contienen generadores $\sigma_k^{(-1)}$ con $j-1 \leq k \leq j+1$, $d_i \in \{-1, 1\}$
Podemos aplicarle una **reducción local** quedando la sub-palabra equivalente:

$$v_0\ \sigma_{j+1}^{-e}\sigma_j^{d_1}\sigma_{j+1}^{e}\ v_1 ... v_{m-1}\ \sigma_{j+1}^{-e}\sigma_j^{d_m}\sigma_{j+1}^{e}\ v_m$$

En definitiva hemos aplicado el homomorfismo $\phi_{j,e}$ definido como:

$$\sigma_j^{\pm 1} \to \epsilon$$
$$\sigma_{j+1}^{\pm 1} \to \sigma_{j+1}^{-e}\sigma_j^{\pm 1}\sigma_{j+1}^{e}$$
$$\sigma_k^{\pm 1} \to \sigma_k^{\pm 1}, \text{ siendo k}\neq\text{j,j+1.}$$

Podemos ver esta transformación en la figura 3.4.

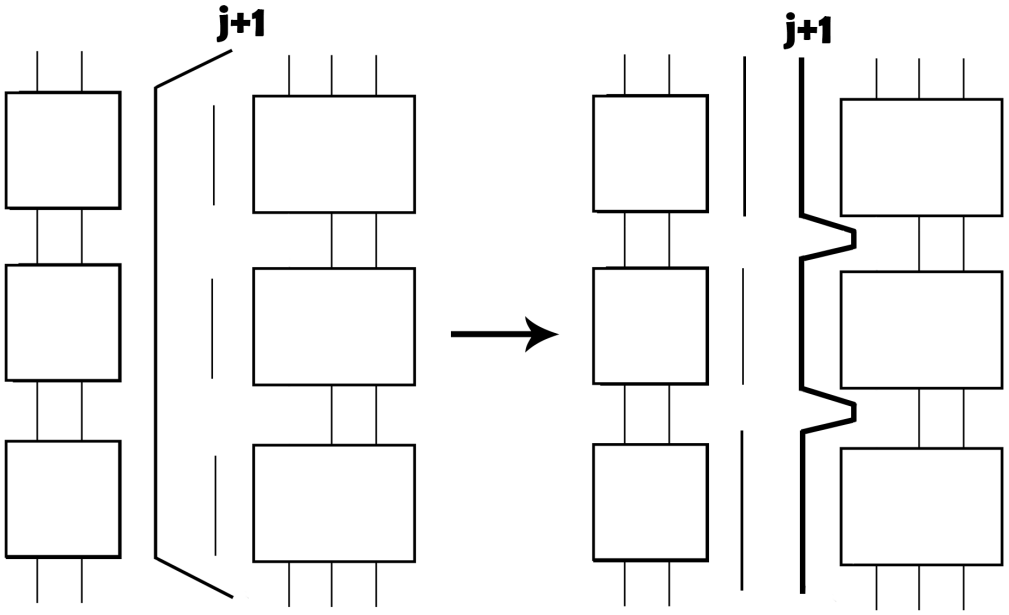

Figura 3.4: Reducción local.

Es importante destacar que antes de realizar una reducción local a una palabra, tendremos que asegurarnos de que dicha palabra sea libremente reducida para evitar una mayor complejidad.

Sea la palabra $\sigma 2\sigma 1^{-1}\sigma 2\sigma 3\sigma 1$. Podemos aplicar una reducción local al main handle $\sigma 1^{-1}\sigma 2\sigma 3\sigma 1$ quedando la palabra $\sigma 2\sigma 1\sigma 2^{-1}\sigma 3$. Esta sería su palabra reducida.

Pero antes de hacer una reducción local a un σ_j-handle, tendremos que asegurarnos que no tenemos ningún σ_{j+1}-handle en el σ_j-handle. En el caso de tenerlo, tendríamos que aplicar la reducción local a este σ_{j+1}-handle y posteriormente aplicar la reducción local al σ_j-handle. De este modo podremos evitar ciclos infinitos. Veámoslo con un ejemplo:

Supongamos que queremos obtener la palabra reducida de la palabra $\sigma 1\sigma 2\sigma 3\sigma 2^{-1}\sigma 1^{-1}$. El generador principal es 1 y tenemos un main handle (que, en este caso, es toda la palabra en sí misma). Aplicamos una reducción local a este main handle quedando:
$\sigma 2^{-1}\sigma 1\sigma 2\sigma 3\sigma 2^{-1}\sigma 1^{-1}\sigma 2$. Vemos que volvemos a tener un main handle y entraremos en bucle infinito. Veamos cómo aplicaríamos la solución que hemos comentado:

Supongamos de nuevo que queremos obtener la palabra reducida de la palabra $\sigma 1\sigma 2\sigma 3\sigma 2^{-1}\sigma 1^{-1}$. Tenemos un main handle pero dicho main handle consta de una sub-palabra $\sigma 2$-handle. Aplicamos una reducción local a este $\sigma 2$-handle quedando:
$\sigma 1\sigma 3^{-1}\sigma 2\sigma 3\sigma 1^{-1}$. Ahora sí podemos aplicar una reducción local al main handle quedando la palabra reducida:
$\sigma 3^{-1}\sigma 2^{-1}\sigma 1\sigma 2\sigma 3$.

Definición:
Diremos que un σ_j-handle está permitido si no contiene ningún σ_{j+1}-handle.

Por ejemplo el main handle $\sigma 1\sigma 2\sigma 3\sigma 2^{-1}\sigma 1^{-1}$ no está permitido porque incluye al $\sigma 2$-handle $\sigma 2\sigma 3\sigma 2^{-1}$. Este 2-handle sí estaría permitido porque no incluye ningún $\sigma 3$-handle.

Si un σ_j-handle no está permitido, le aplicaremos reducciones locales, como hemos hecho en el ejemplo, hasta que pase a estar permitido.

Definición:
Diremos que una palabra β' es deducida de una palabra β usando una **reducción de handle** si β' se obtiene a partir de una reducción local de un σ_j-handle permitido de β.

Teorema 3.1.1. *Cualquier palabra puede ser reducida por una secuencia finita de reducciones de handle hasta llegar a su palabra reducida.*

Para poner en uso todas estas ideas vamos a ver un ejemplo en el que demostraremos que dos palabras dadas son equivalentes:
Supongamos que tenemos las palabras $\beta 1 = \sigma 2\sigma 3^{-1}\sigma 1\sigma 2^{-1}\sigma 3$ y $\beta 2 = \sigma 1\sigma 2\sigma 3^{-1}\sigma 1\sigma 2^{-1}$ que representan a las trenzas de la figura 3.5.

(a) $\beta 1$ (b) $\beta 2$

Figura 3.5: Trenzas equivalentes.

A simple vista es difícil saber si las palabras son o no equivalentes. Para ver que las palabras son equivalentes ($\beta 1 \sim \beta 2$), tenemos que ver que se verifica $\beta 1\beta 2^{-1} \sim \epsilon$, donde ϵ representa a la cadena vacía.

En primer lugar vamos a considerar la palabra con la que vamos a trabajar:
$\beta 1 \beta 2^{-1} = \sigma 2 \sigma 3^{-1} \sigma 1 \sigma 2^{-1} \sigma 3 \sigma 2 \sigma 1^{-1} \sigma 3 \sigma 2^{-1} \sigma 1^{-1}$.

Nos encontramos el main handle $\sigma 1 \sigma 2^{-1} \sigma 3 \sigma 2 \sigma 1^{-1}$. Vamos a aplicarle una reducción local, pero vemos que contiene al $\sigma 2$-handle $\sigma 2^{-1} \sigma 3 \sigma 2$. Por tanto aplicamos una reducción local a este $\sigma 2$-handle:
$\sigma 2^{-1} \sigma 3 \sigma 2 \rightarrow \sigma 3 \sigma 2 \sigma 3^{-1}$, obteniendo la palabra:
$\beta 1 \beta 2^{-1} = \sigma 2 \sigma 3^{-1} \sigma 1 \sigma 3 \sigma 2 \sigma 3^{-1} \sigma 1^{-1} \sigma 3 \sigma 2^{-1} \sigma 1^{-1}$.

Nos encontramos el main handle permitido $\sigma 1 \sigma 3 \sigma 2 \sigma 3^{-1} \sigma 1^{-1}$. Le aplicamos una reducción local:
$\sigma 1 \sigma 3 \sigma 2 \sigma 3^{-1} \sigma 1^{-1} \rightarrow \sigma 3 \sigma 2^{-1} \sigma 1 \sigma 2 \sigma 3^{-1}$, obteniendo la palabra:
$\beta 1 \beta 2^{-1} = \sigma 2 \sigma 3^{-1} \sigma 3 \sigma 2^{-1} \sigma 1 \sigma 2 \sigma 3^{-1} \sigma 3 \sigma 2^{-1} \sigma 1^{-1}$.

Esta palabra no es libremente reducida porque nos encontramos un par de veces la sub-palabra $\sigma 3^{-1} \sigma 3$. Las eliminamos y obtenemos la palabra:
$\beta 1 \beta 2^{-1} = \sigma 2 \sigma 2^{-1} \sigma 1 \sigma 2 \sigma 2^{-1} \sigma 1^{-1}$.

De nuevo nos encontramos con una palabra que no es libremente reducida porque nos encontramos un par de veces la sub-palabra $\sigma 2 \sigma 2^{-1}$. Las eliminamos y obtenemos la palabra:
$\beta 1 \beta 2^{-1} = \sigma 1 \sigma 1^{-1}$.

Otra vez nos encontramos con una palabra no libremente reducida. Eliminamos la sub-palabra $\sigma 1 \sigma 1^{-1}$ quedando la cadena vacía, es decir:
$\beta 1 \beta 2^{-1} = \epsilon$.

Concluimos que ambas palabras son equivalentes, luego las trenzas a las que representan son equivalentes. Sus trenzas cerradas serán equivalentes y por tanto los nudos a los que se está representando son también equivalentes.

Capítulo 4

Desarrollo informático.

4.1 Pseudoalgoritmos.

En esta sección vamos a ver algunos pseudoalgoritmos referentes a los algoritmos que hemos implementado. Nos vamos a centrar en aquellos que tienen mayor complejidad e interés, sin entrar en gran detalle.

En concreto, vamos a ver los pseudoalgoritmos para el algoritmo de Dehornoy (que se descompone en varios algoritmos auxiliares), el algoritmo que nos permite comprobar si dos trenzas dadas (o sus cierres) son o no equivalentes entre sí y el algoritmo que nos permite comprobar si una trenza dada (o su cierre) es equivalente a la trenza trivial.

Algoritmo de Dehornoy para trenzas.

Vamos a ir viendo los algoritmos auxiliares y concluiremos con el propio algoritmo de Dehornoy. Recordemos que este algoritmo se explica con detalle en la sección 3.1.

Mediante el algoritmo Simplifica eliminamos ocurrencias de tipo $\sigma_i^e \sigma_i^{-e}, e \in$ {-1,1} de una palabra que representa a cierta trenza.

Algoritmo 4.1.1. ***Algoritmo Simplifica****(indices_braid)*
ENTRADA: indices_braid (cadena de enteros que representa los cruces de una trenza)
SALIDA: braid_aux (cadena auxiliar para mejor representacion visual)
nueva_braid (cadena de enteros que representa los cruces de la trenza tras eliminar dos cruces opuestos)
encontrado (bool para indicar si se produce simplificación)

```
1    Recorro los cruces de indices_braid
2    Si hay dos cruces seguidos con signos opuestos, creo una copia de
        indices_braid y elimino dichos cruces.
```

Haciendo uso del siguiente algoritmo podremos encontrar las posiciones que delimitan un σ_{minimo}-handle.

Algoritmo 4.1.2. ***Algoritmo encuentra_handle****(indices_braid, minimo)*
ENTRADA: indices_braid (cadena de enteros que representa los cruces de una trenza)
minimo (generador de handle a encontrar)
SALIDA: pos1 (posición inicial del handle encontrado)
pos2 (posición final del handle encontrado)

```
1   Recorro los cruces de indices_braid
2   Si hay un cruce generado por el elemento minimo, me quedo con esa
      posicion como pos1 y su signo. Sino fin.
3   Si encuentro un cruce generado por el elemento minimo con signo
      opuesto, me quedo con esa posicion como pos2.
```

Haciendo uso del algoritmo reduccion_base aplicamos una reducción local a la trenza representada por indices_braid sobre el σ_{minimo}-handle que está situado entre las posiciones pos1 y pos2. Es importante destacar el hecho de que este σ_{minimo}-handle no contiene $\sigma_{minimo+1}$-handle.

Algoritmo 4.1.3. ***Algoritmo reduccion_base****(indices_braid, minimo, pos1, pos2)*
ENTRADA: indices_braid (cadena de enteros que representa los cruces de una trenza)
minimo (generador de handle)
pos1 (posición inicial del handle)
pos2 (posición final del handle)
SALIDA: braid_aux2 (cadena auxiliar para mejor representación visual)
nuevo (cadena de enteros que representa los cruces de la trenza tras aplicar la reducción local al σ_{minimo}-handle entre pos1 y pos2)
simplificado2 (bool auxiliar para mejor representación visual)

```
1   Creo vector_auxiliar
2   Recorro los cruces de indices_braid desde pos1 hasta pos2
3   Si hay un cruce generado por el elemento (minimo+1) aniado al
      vector_auxiliar los 3 cruces correspondientes. Sino, aniado el
      mismo cruce.
4   Creo vector_nuevo con los elementos desde inicio de indices_braid
      hasta pos1, vector_auxiliar, y los elementos desde pos2 hasta
      final de indices_braid.
5   Si indices_braid y vector_auxiliar tienen distinto tamanio, asigno
      a braid_aux2 una cadena con ciertos ceros para mejor
      visualizacion.
```

Estos algoritmos auxiliares no se proporcionan para uso directo al usuario ya que el realmente interesante es el algoritmo de Dehornoy, pero sí podrían ser usados. Veamos entonces el algoritmo de Dehornoy.

Algoritmo 4.1.4. ***Algoritmo dehornoy****(br, N_cortes, Radio, representar)*
ENTRADA: br (trenza)
N_cortes (numero de cortes de las cadenas de la trenza)
Radio (radio de las cadenas de la trenza)
representar (bool para representar las equivalencias de la trenza)
SALIDA: es_trivial (bool que nos indica si la trenza dada es o no trivial)
trenza_final (cadena de enteros que representa a la trenza reducida equivalente a br)

```
1   Si numero_argumentos=1 -> N_cortes=20, Radio=0.5, representar=1.
2   indices_braid = cadena de enteros que representa a la trenza
3   Si br tiene cadenas a la derecha triviales, las eliminamos
       visualmente.
4   Mientras queden handles en la trenza dada...
5           Obtenemos la palabra libremente reducida de indices_braid.
6           Si no se produce reduccion...
7                   minimo = generador principal de indices_braid.
8                   [pos1,pos2]=encuentra_handle(indices_braid,minimo)
9                   Si pos1 y pos2 son posiciones validas....
10                          Busco primer subhandle a realizar en el
                               handle.
11                          Actualizo pos1 y pos2
12                          Aplico reduccion_base a dicho subhandle.
13          Creo matriz con secuencia de palabras generadas en el
               proceso.
14  Si representar, muestro las trenzas usando dicha matriz.
```

Finalmente cabe comentar que el algoritmo de Dehornoy se podrá aplicar sobre trenzas cerradas. El proceso que se realizará internamente será exactamente el mismo que para trenzas, ya que el cerrar la trenza no aporta información nueva para el algoritmo de Dehornoy.

Algoritmo de equivalencia para trenzas.