UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Grup de Trenes i el Teorema d'Alexander

Autora: Anna Perea Navarro

Director: Dr. Javier José Gutiérrez Marín
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 19 de gener de 2020

Abstract

Braid theory is a an important tool in low dimensional topology. In this work we study the braid group and see how it relates to knot theory. The main objective is to formulate and prove Alexander's theorem stating that any knot, or more generally any link, can be obtained as the closure of a braid. We give two constructive proofs, one based on Alexander's original proof and the other one following the Yamada–Vogel's algorithm. Moreover, we provide the code of of an implementation of the latter algorithm, written in Python .

Resum

La teoria de trenes és una eina important de la topologia de baixa dimensió. En aquest treball estudiem el grup de trenes i veiem la relació d'aquest amb la teoria de nusos. L'objectiu principal és enunciar i provar el Teorema d'Alexander que afirma que qualsevol nus (o enllaç) pot ser representat com la clausura d'una trena. Donem dues demostracions constructives, una basada en la original d'Alexander i una altra que segueix l'algorisme de Yamada–Vogel. A més, proporcionem el codi d'un programa escrit en Python el qual descriu la implementació d'aquest últim algorisme esmentat.

2010 Mathematics Subject Classification: 20F36, 57M25

Agraïments

Vull agrair en primer lloc a tots els professors que m'han acompanyat durant aquests anys, gràcies per infondre'm l'amor per les matemàtiques.

Al Dr.Javier Gutiérrez per recolzar-me al llarg del treball i al Seve i al Fritz per animar-me a realitzar els meus objectius i donar-me les eines necessàries.

A l'Ana per iniciar-me en aquest camí. Als meus pares, germana i amics per poder comptar amb ells sempre que ho he necessitat. I en especial al Ferran per donar-me suport, confiança i estima tots i cadascun dels dies.

Sense vosaltres no estaria fent això en aquests moments.

Sumari

1 Introducció

1.1 El projecte

Al contrari del que succeeix amb la teoria de trenes, en les matemàtiques no és usual poder identificar quin és el moment exacte en que una teoria va començar a ser desenvolupada per primer cop. Aquest camp de la topologia que és la teoria de trenes es va començar a estudiar com a disciplina matemàtica l'any 1925 amb la *Theorie der Zöpfe* [7] d'Emil Artin, que va ser corregida i publicada l'any 1947 com *Theory of braids* [8]. Artin és oficialment l'únic autor d'aquest document, però com Mortiz Epples nota, va treballar amb la col·laboració de Otto Schreier. Tot i que la recerca en l'àmbit de les trenes matemàtiques ja va ser iniciada l'any 1890 per Adolf Hurwitz, aquesta va ser de manera implícita a través de les superfícies de Riemann. Artin va ser el primer en intentar estudiar les trenes com un objecte matemàtic independent i intentar donar una formalització algebraica completa del conjunt de trenes. Un punt crucial d'aquest estudi va ser en el que Artin va demostrar que el conjunt de trenes té una estructura de grup i va donar una representació explícita per a aquest grup tal i com veurem en el Teorema d'Artin.

Al llarg d'aquest treball podrem notar que les trenes i els nusos estan estretament relacionats i és necessari fer una ullada també a la teoria de nusos per apreciar plenament les trenes. Tot i que al segle XVIII la teoria de nusos fa la seva aparició en estudis de Vandermonde, Gauss i Klein, va ser al voltant de l'any 1860 quan es va iniciar un estudi sistemàtic de la teoria de nusos degut a un interès físic. El matemàtic escocès Peter Guthrie Tait va treballar durament per a fer una llista de nusos topològicament diferents en resposta a una demanda de l'investigador, també escocès, Lord Kelvin. Aquest últim havia suggerit que podia ser que els àtoms fossin nusos de èter i que aleshores diferents nusos representessin diferents elements, i per tant es creia que classificant tots els nusos s'obtindria una taula de tots els elements. Quan l'èter no va ser detectat en l'experiment de Michelson i Morley, aquesta teoria va ser rebutjada i va perdre gran part del seu interès pels físics. No obstant això, els matemàtics van seguir el seu estudi i, amb el desenvolupament de la topologia, topòlegs com Alexander, Reidemeister i Dehn van investigar els nusos. A partir de al segona part del segle XX es van produir els desenvolupaments més importants d'aquesta teoria, de la mà d'estudis de Conway, Jones i Kauffman entre d'altres. Gràcies a totes aquestes contribucions ara la teoria de nusos té aplicacions en l'estudi de la replicació i recombinació de l'ADN juntament amb altres temes d'estudi en la química i física.

Cal destacar que en aquest treball ens centrarem sobretot en la relació entre nusos i trenes que ens ve donada pel Teorema d'Alexander. Incloent un algorisme programat en Python en el que queda reflectit un dels procediments utilitzats per a demostrar el Teorema d'Alexander.

1.2 Estructura de la memòria

Iniciarem la memòria en la Secció 2 introduint el concepte de trena i provant que el conjunt de les trenes forma un grup. A continuació veurem la descripció que va fer Artin del grup de trenes a partir d'uns certs generadors que definirem, aquesta presentació queda definida i demostrada pel Teorema d'Artin.

Un cop descrita la base de la teoria de trenes passarem a enunciar el Teorema d'Alexander en la Secció 3 juntament amb les definicions de la teoria de nusos necessàries

per comprendre'l. Donarem dues demostracions diferents per a provar el Teorema d'Alexander: Una que utilitza l'algorisme de Seifert per a demostrar-ho a través d'un altre algorisme, anomenat de Yamada-Vogel; i una altra que està basada en la demostració original feta per Alexander a l'any 1923. Aquestes dues demostracions les acompanyarem gràficament amb uns exemples.

Per últim, en la Secció 4 donarem les indicacions seguides per a programar l'algorisme basat en el de Yamada-Vogel, utilitzat en la primera demostració nomenada del Teorema d'Alexander. Explicarem en detall, pas per pas, el procés dut a terme per a elaborar el codi en Python que trobem a l'Annex A, amb el que conclou el treball.

2 Grup de trenes

Intuïtivament, una trena és un conjunt de n cordes un extrem de les quals està adherit a una barra horitzontal superior i l'altre està adherit a una barra horitzontal inferior. De manera que cada corda interseca exactament un cop amb qualsevol pla horitzontal que es trobi entre les dues barres, és a dir, cada corda ha de dirigir-se en direcció estrictament descendent.

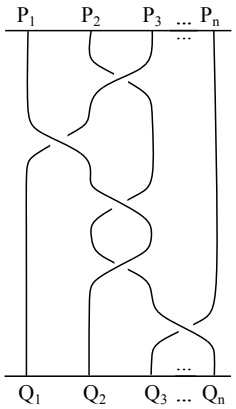


Figura 1: Representació d'una n-trena

Una manera més formal de descriure-ho seria:

Definició 2.1. Sigui $\mathbb{R}^3$ l'espai euclidià de dimensió 3. Siguin ϵ_0^2 i ϵ_1^2 dos plans paral·lels amb la coordenada z igual a 0 i a 1, respectivament. Siguin P_i i Q_i (amb $1 \leq i \leq n$) els punts de coordenades $(i, 0, 1)$ i $(i, 0, 0)$, respectivament, de manera que $P_1, \ldots, P_n$ es troben a la recta $y = 0$ del pla superior i $Q_1, \ldots, Q_n$ es troben a la recta $y = 0$ del pla inferior. Una trena β de n cordes (també anomenada *n-trena*) està formada per un sistema de n arcs $a_1, \ldots, a_n$ tal que a_i connecta el punt P_i del pla superior amb el punt Q_j del pla inferior complint que:

(a) Cada arc a_i interseca al pla $z = t$ un i només un cop per qualsevol $t \in [0, 1]$.

(b) Els arcs $a_1, \ldots, a_n$ intersequen el pla $z = t$ en n punts diferents per tot $t \in [0, 1]$

A la permutació π que assigna a cada punt P_i el seu corresponent Q_j li diem la permutació de la trena. Si aquesta permutació és la trivial diem que la trena és *pura*.

Considerarem dues trenes equivalents ($\beta \sim \beta'$) si podem aconseguir que es veguin iguals reordenant les seves respectives cordes sense que aquestes passin a través d'una altra corda o d'elles mateixes. Per tant, l'equivalència de trenes només tindrà sentit si β i β' tenen la mateixa permutació. Per arribar d'una trena a l'altre no es podrà tallar i tornar a enganxar cap corda, igual que tampoc està permès treure les cordes per sobre de la barra superior o per sota de la inferior.

Definició 2.2. Dues trenes β i β' amb la mateixa permutació π es diuen *equivalents* si existeix una homotopia de β a β' mitjançant trenes β_t totes amb permutació π i amb $t \in [0, 1]$.

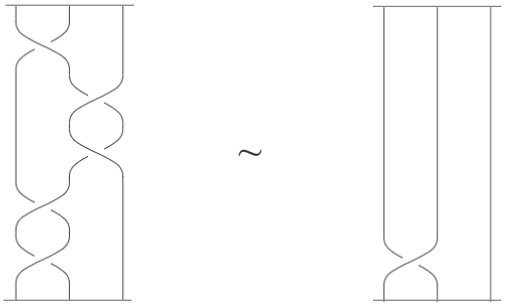


Figura 2: Exemple de dues trenes equivalents

Una altra manera de definir l'equivalència de trenes és mitjançant processos Δ aplicats a trenes poligonals.

Definició 2.3. Una *trena poligonal* és una trena les cordes de la qual estan representades com segments lineals. Tota trena la podem representar com una trena poligonal i a l'inversa.

Definició 2.4. Sigui u un segment d'una trena poligonal β a $\mathbb{R}^3$ i sigui D un triangle a $\mathbb{R}^3$ amb cares u, v i w. Si $D \cap \beta = u$, aleshores $\beta' = (\beta - u) \cup v \cup w$ defineix una altra trena poligonal. En aquest cas direm que β' s'obté de β mitjançant un *procés* Δ (o moviment elemental). E procés invers de Δ es denota com Δ^{-1}

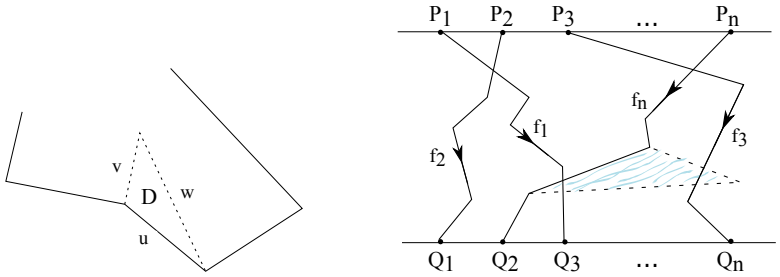


Figura 3: Representació d'un procés Δ en una trena poligonal

Aleshores, si podem transformar una trena en l'altre a través d'un seqüència finita de processos $\Delta^{\pm 1}$ direm que aquestes dues trenes són equivalents.

$$\beta = \beta_0 \xrightarrow{\Delta^{\pm 1}} \beta_1 \xrightarrow{\Delta^{\pm 1}} \cdots \xrightarrow{\Delta^{\pm 1}} \beta_n = \beta'$$

Lema 2.5. *L'equivalència de trenes* ($\sim$) *defineix una relació d'equivalència sobre el conjunt de les n-trenes.*

Donades dues trenes β, β' podem definir una operació composició (o producte) que defineix una estructura de grup no abelià al conjunt de les n-trenes mòdul la relació d'equivalència de trenes, el qual anomenarem B_n.

Definició 2.6. Donades dues trenes β i β' l'operació *composició* consisteix en identificar els punts inferiors Q_i de β amb els punts superiors P_i' de β', de manera que combinant les dues n-trenes n'obtenim una tercera $\beta\beta'$.

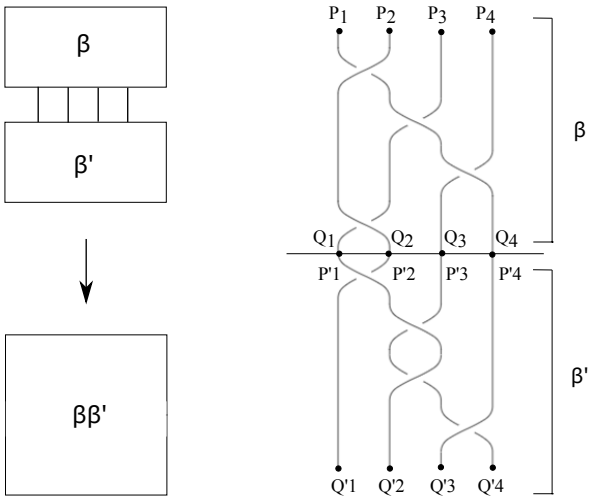


Figura 4: Operació composició en el grup B_n

Definició 2.7. El subgrup de B_n format per les trenes amb permutació trivial s'anomena *grup pur de trenes* i es denota per PB_n.

Proposició 2.8. *La composició de trenes respecta l'equivalència de trenes. És a dir, si* $\beta \sim \beta'$ *i* $\bar{\beta} \sim \bar{\beta}'$ *aleshores,* $\beta\bar{\beta} \sim \beta'\bar{\beta}'$

Demostració. Com $\beta \sim \beta'$ i $\bar{\beta} \sim \bar{\beta}'$, per la definició d'equivalència de trenes tenim les cadenes següents:

$$\beta = \beta_0 \xrightarrow{\Delta^{\pm 1}} \beta_1 \xrightarrow{\Delta^{\pm 1}} \cdots \xrightarrow{\Delta^{\pm 1}} \beta_n = \beta'$$

$$\bar{\beta} = \bar{\beta}_0 \xrightarrow{\Delta^{\pm 1}} \bar{\beta}_1 \xrightarrow{\Delta^{\pm 1}} \cdots \xrightarrow{\Delta^{\pm 1}} \bar{\beta}_n = \bar{\beta}'$$

de les quals es dedueix:

$$\beta\bar{\beta} = \beta_0\bar{\beta} \xrightarrow{\Delta^{\pm 1}} \beta_1\bar{\beta} \xrightarrow{\Delta^{\pm 1}} \cdots \xrightarrow{\Delta^{\pm 1}} \beta_n\bar{\beta} = \beta'\bar{\beta}$$

$$\beta'\bar{\beta} = \beta'\bar{\beta}_0 \xrightarrow{\Delta^{\pm 1}} \beta'\bar{\beta}_1 \xrightarrow{\Delta^{\pm 1}} \cdots \xrightarrow{\Delta^{\pm 1}} \beta'\bar{\beta}_n = \beta'\bar{\beta}'$$

Aleshores, com a conseqüència de la transitivitat de la relació d'equivalència ($\sim$) tenim que $\beta\bar{\beta} = \beta'\bar{\beta}'$ □

Aquesta proposició és important perquè demostra que la tria del representant d'una classe és indiferent respecte a la composició de trenes.

Ara que hem vist que la operació està ben definida sobre els elements de B_n, passem a demostrar que efectivament B_n és un grup no abelià. És a dir, veurem que la composició és associativa, que existeix la identitat o element neutre per tot element i també que cada element de B_n té un element invers. A més de comprovar que efectivament la composició és una operació no commutativa.

Proposició 2.9. *La composició és una operació associativa.*

Demostració. Tal i com hem definit la composició de trenes és evident que la trena $\beta_0\beta_1\beta_2$ resultant de la composició de β_0, β_1 i β_2 serà la mateixa tant si calculem $(\beta_0\beta_1)\beta_2$ com si calculem $\beta_0(\beta_1\beta_2)$. □

L'element identitat (o element neutre) de B_n és la trena formada per n cordes paral·leles sense cap encreuament. L'anomenarem *n-trena trivial*, i la denotarem per Id_n.

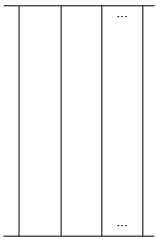


Figura 5: n-trena trivial

És immediat comprovar que per tota n-trena $\beta \in B_n$ es verifica que $\beta Id_n \sim Id_n\beta \sim \beta$, ja que la composició de qualsevol trena amb la trena trivial s'obté només l'allargament de les cordes d'aquesta trena, pel que el trenat resta igual.

Proposició 2.10. *Per tot element $\beta \in B_n$ existeix un element invers $\beta^{-1} \in B_n$ el qual està format pel reflex de β en el pla horitzontal i verifica que $\beta\beta^{-1} = \beta^{-1}\beta = Id_n$.*

Demostració. Per construcció de β^{-1}, aquesta desfà el trenat de β. Aleshores podem fer un número finit de moviments Δ a $\beta\beta^{-1}$ així obtenint la trena trivial. I per demostrar que $\beta^{-1}\beta = Id_n$ només cal considerar a β com el reflexe de β^{-1}, la seva inversa. □

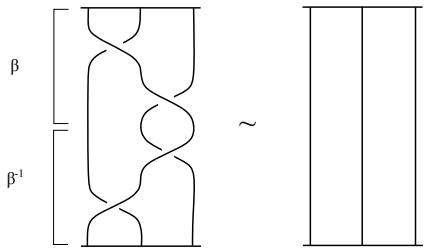


Figura 6: La trena β^{-1} és l'invers de β

A més hem dit que el grup B_n no és abelià, és a dir, que en general la operació composició depèn de l'ordre dels seus factors. Veguem-ho amb un exemple:

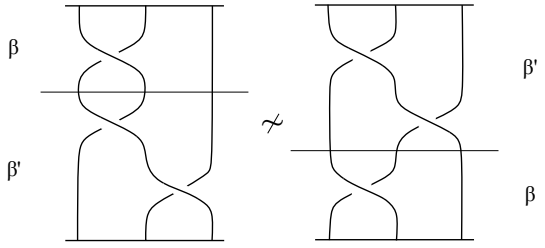


Figura 7: La composició de les trenes β i β' no és commutativa

És fàcil comprovar que en la Figura 7 les trenes $\beta\beta'$ i $\beta'\beta$ no són equivalents, ja que no tenen la mateixa permutació.

2.1 Presentació de Artin

L'any 1947 Emil Artin va descriure el grup de trenes B_n mitjançant una presentació explícita a partir de n-trenes elementals. En aquesta secció introduïrem el concepte de n-trena elemental i veurem aquesta presentació de Artin.

2.1.1 Les trenes elementals

Definició 2.11. La *n-trena elemental* σ_i es defineix com la n-trena formada per l'encreuament de la corda i-èsima sobre la $(i+1)$-èsima. Anàlogament, en la n-trena elemental σ_i^{-1} la i-èsima corda creua per sota la $(i+1)$-èsima.

Figura 8: n-trenes elementals

Com veurem a continuació, qualsevol n-trena es pot escriure com a producte d'un nombre finit de n-trenes elementals $\sigma_1, \ldots, \sigma_n$.

Proposició 2.12. *Tota n-trena és composició de n-trenes elementals.*

Demostració. Sigui β una trena qualsevol en B_n. Representem β en un diagrama i partim aquest en rectangles de manera que a cada rectangle hi hagi exactament un encreuament de β. En el cas de que coincideixi de que hi ha més d'un encreuament a la mateixa alçada, és obvi que podem agafar una trena equivalent a β sense aquest problema ja que si tenim dos encreuaments a la mateixa alçada podem pujar o baixar un d'ells.

Aleshores, per construcció a cada rectangle hi haurà només un encreuament i això és el mateix que tenir una trena elemental a cada rectangle. Per tant, la juxtaposició dels rectangles amb la que obtenim la n-trena β no serà una altra cosa que la composició de trenes elementals. □

Podem observar mitjançant les representacions gràfiques que les n-trenes elementals compleixen les relacions següents:

(i) $\sigma_i\sigma_j = \sigma_j\sigma_i$ si $|i-j| \geq 2$.

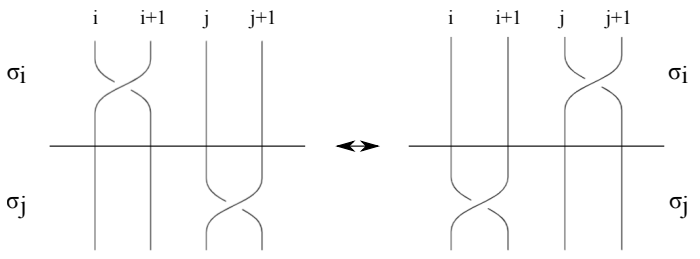


Figura 9: Representació gràfica de (i)

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

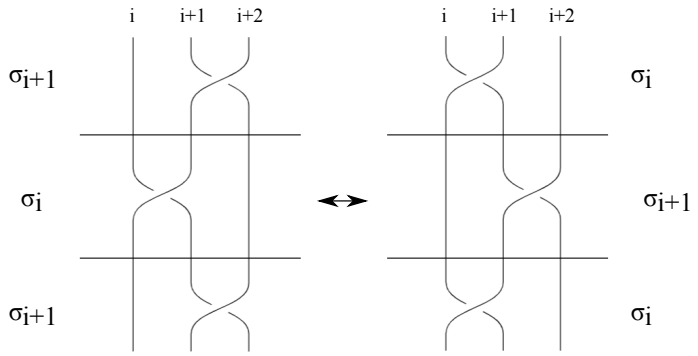


Figura 10: Representació gràfica de (ii)

2.1.2 El Teorema d'Artin

La presentació que va fer Artin del grup de trenes B_n utiliza les relacions vistes en les Figures 9 i 10. Aquesta, queda definida en el seu teorema com veurem a continuació.

Teorema 2.13 (Teorema d'Artin)**.** *El grup de trenes B_n és isomorf a un grup G amb n generadors $x_1, \ldots, x_n$ complint les relacions: $x_i x_j = x_j x_i$ si $|i-j| \geq 2$ i $x_i x_{i+1} x_i = x_{i+1} x_i x_{i+1}$ per $i = 1, \ldots, n-2$. Aquest isomorfisme entre G i B_n envia els elements x_i de G a les n-trenes elementals σ_i pertanyents a B_n.*

Demostració. Per a veure que els grups G i B_n són isomorfs considerem una aplicació $\varphi : G \to B_n$ de manera que a cada paraula $w = x_{i_1}^{e_1} \ldots x_{i_k}^{e_k} \in G$ hi associï una imatge $\varphi(w) = \varphi(x_{i_1}^{e_1} \ldots x_{i_k}^{e_k}) = \sigma_{i_1}^{e_1} \ldots \sigma_{i_k}^{e_k} \in B_n$. On $e_j \in \{\pm 1\}$ per a tot $j = 1, \ldots, k$ i x_i^{-1} serà l'element invers de x_i que s'enviarà a la n-trena elemental σ_i^{-1}.

Per a veure que aquesta aplicació és un isomorfisme de grups entre G i B_n en primer lloc veguem que φ és homomorfisme de grups. Degut a la manera en que hem definit φ serà suficient amb demostrar que φ porta les relacions de G en el neutre de B_n: la n-trena trivial Id_n.

Llavors, a partir de la definició de φ i de les relacions vistes en les Figures 9 i 10 sabem que es compleix que $\varphi(x_i x_j x_i^{-1} x_j^{-1}) = \sigma_i \sigma_j \sigma_i^{-1} \sigma_j^{-1} = Id_n$ per a tot $i, j = 1, \ldots, n-1$ satisfent $|i-j| \geq 2$, i $\varphi(x_i x_{i+1} x_i x_{i+1}^{-1} x_i^{-1} x_{i+1}^{-1}) = \sigma_i \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} \sigma_i^{-1} \sigma_{i+1}^{-1} = Id_n$, per a tot $i = 1, \ldots, n-2$. Per tant, podem afirmar que φ és homomorfisme de grups.

Per a seguir amb la demostració de que φ és un isomorfisme, veguem que és exhaustiva. Recordem que ho serà si per a tot $\beta \in B_n$ existeix $w \in G$ tal que $\varphi(w) = \beta$. Donat $\beta \in B_n$, podem expressar la n-trena β com a composició de n-trenes elementals $\beta = \sigma_{i_1}^{e_1} \ldots \sigma_{i_k}^{e_r}$ amb $i_j \in \{1, \ldots, n-1\}$ i $e_j \in \{\pm 1\}$, $\forall j = 1, \ldots, r$. I aleshores és suficient prendre com a w el $w = x_{i_1}^{e_1} \ldots x_{i_k}^{e_k} \in G$ que té a β com a imatge per a demostrar que φ és exhaustiva.

Finalment, ens caldrà veure que φ és injectiva, i un cop demostrat això ja haurem acabat. És a dir, ens cal provar que si $\varphi(g) = \varphi(g')$ aleshores $g = g'$, amb $\varphi(g), \varphi(g') \in B_n$ i $g, g' \in G$.

Considerem les trenes $\beta = \varphi(g)$ i $\beta' = \varphi(g')$ amb $\varphi(g) = \varphi(g')$. Aquestes dues trenes

són equivalents, i per tant podem construir una cadena finita de n-trenes intermediàries

$$\beta = \beta_1 \xrightarrow{\Omega} \beta_2 \xrightarrow{\Omega} \cdots \xrightarrow{\Omega} \beta_s = \beta'$$

que verifiqui que cada parell de trenes es distingeixi només en un moviment elemental. Com que φ és exhaustiva, per a tot β_i existeix algun $w_i \in G$ tal que la seva imatge per φ és β_i. Per tant, temin el diagrama següent:

$$\begin{array}{ccccccccc} \beta = \beta_1 & \xrightarrow{\Omega} & \beta_2 & \xrightarrow{\Omega} & \beta_3 & \xrightarrow{\Omega} & \cdots & \xrightarrow{\Omega} & \beta_s = \beta' \\ \uparrow{\scriptstyle\varphi} & & \uparrow{\scriptstyle\varphi} & & \uparrow{\scriptstyle\varphi} & & & & \uparrow{\scriptstyle\varphi} \\ g = w_1 & & w_2 & & w_3 & & & & w_s = g' \end{array}$$

Ara si demostrem que $w_i = w_{i+1}$ $\forall i = 1, \ldots, n-1$ en G tindrem que $g = g'$ i haurem acabat.

Considerem el fragment del diagrama:

$$\begin{array}{ccc} \beta_i & \xrightarrow{\Omega} & \beta_{i+1} \\ \uparrow{\scriptstyle\varphi} & & \uparrow{\scriptstyle\varphi} \\ w_i & & w_{i+1} \end{array}$$

Notem que cada β_{i+1} s'obté de β_i mitjançant un sol moviment elemental. Prenem diagrames D_i i D_{i+1} de β_i i β_{i+1} respectivament, i considerem les seves particions respectives en rectangles que verifiquin que tan sols es produeixi un únic encreuament en cada regió rectangular.

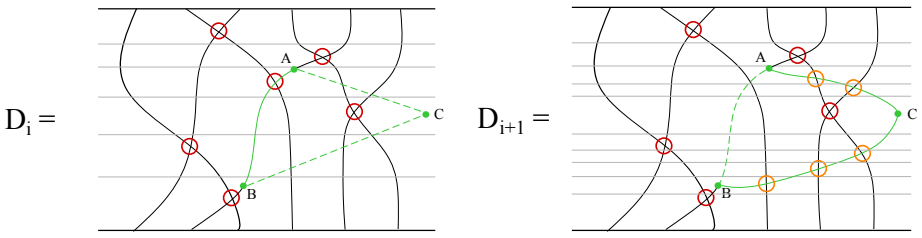
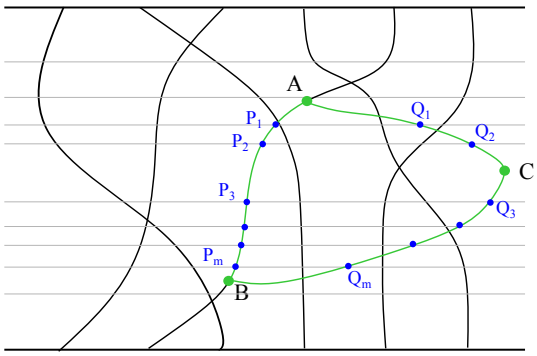


Figura 11: Diagrames D_i i D_{i+1} de β_i i β_{i+1} respectivament

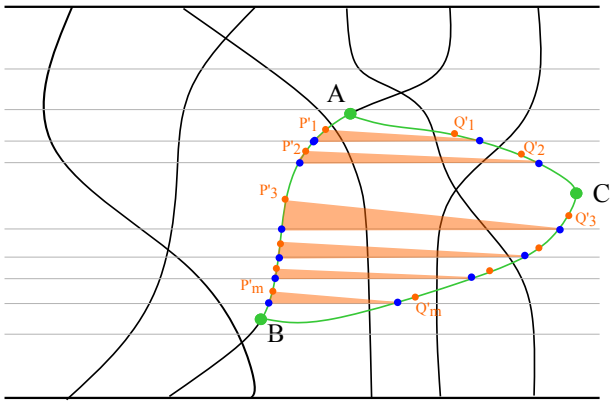
Aquest moviment elemental introdueix uns arcs AC i CB on abans hi havia un arc AB tal i com podem veure en la Figura 11. Com a conseqüència d'aquest moviment elemental desapareixeran els encreuaments en els que l'arc AB formava part i se'n crearan de nous, els quals apareixen de color taronja a la Figura 11.

Els arcs AB i $AC \cup CB$ tallen els segments que divideixen el diagrama (els rectangles de la partició) en els punts $P_1, \ldots, P_m$ i $Q_1, \ldots, Q_m$ respectivament. Tal i com s'il·lustra a continuació:



Siguin $P'_1, \ldots, P'_m$ els punts sobre l'arc AB que verifiquen:

- Cada P'_j es troba per sobre del seu respectiu P_j.
- Cap triangle definit pels punts $P'_j Q_j P_j$ conté encreuaments de β_i ni de β_{i+1}.



Ara substituïm l'arc AQ_1 per $AP'_1 \cup P'_1 Q_1$ i a continuació, substituïm $P'_1 Q_1 \cup Q_1 Q_2$ per $P'_1 P'_2 \cup P'_2 Q_2$. A aquesta operació li direm *moviment quasi-elemental* i consisteix en un parell de moviments elementals successius.

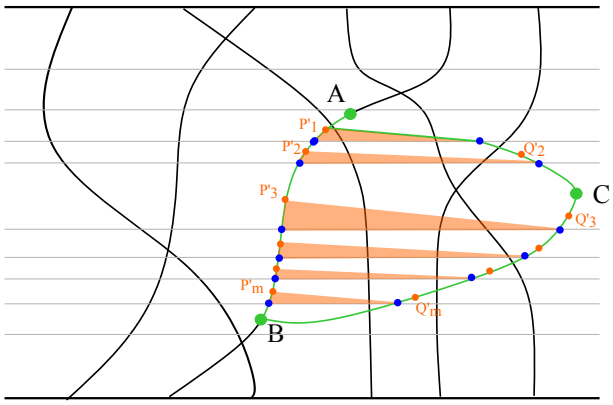


Figura 12: Substituïm AQ_1 per $AP'_1 \cup P'_1 Q_1$

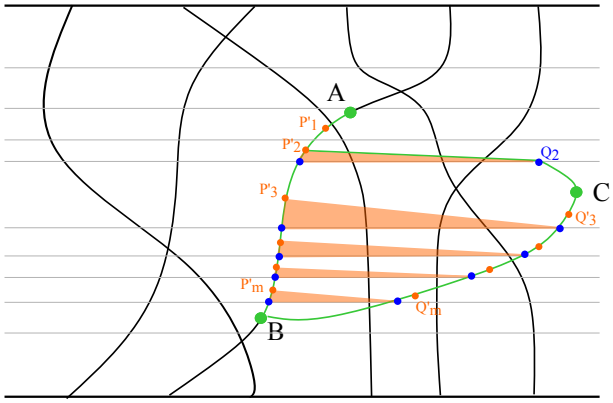


Figura 13: Substituïm $P_1'Q_1 \cup Q_1Q_2$ per $P_1'P_2' \cup P_2'Q_2$

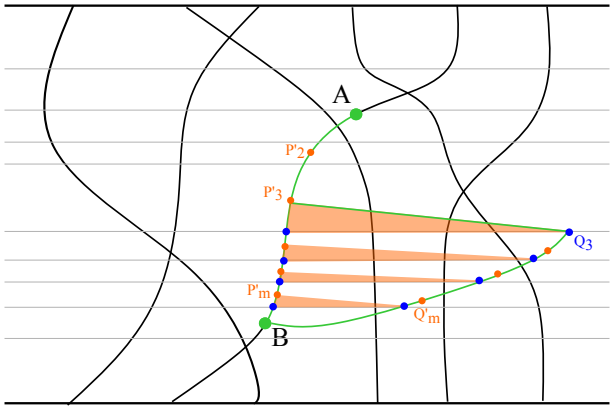


Figura 14: Substituïm $P_2'Q_2 \cup Q_2Q_3$ per $P_2'P_3' \cup P_3'Q_3$

Repetim aquest procediment fins a arribar a $P_m'Q_m \cup Q_mB$ i substituïm aquest últim arc per $P_m'B$.

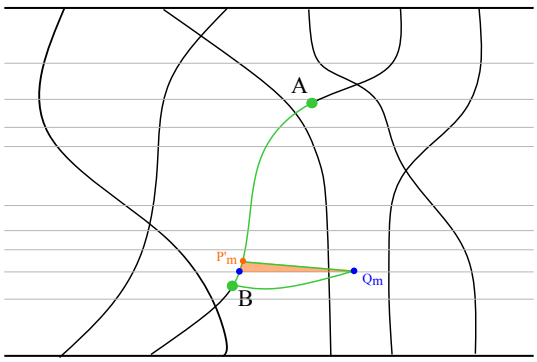


Figura 15: Arribem a $P_m'Q_m \cup Q_mB$

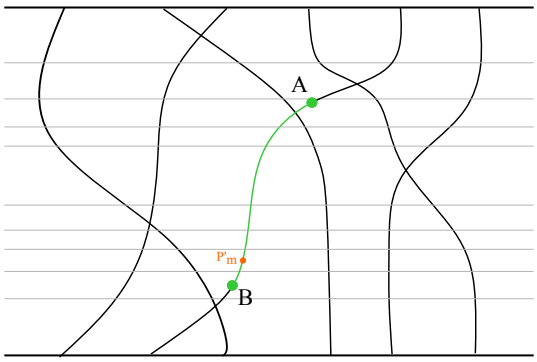


Figura 16: Substituïm $P'_mQ_m \cup Q_mB$ per P'_mB

Ara doncs, tenim la cadena intermitja entre β_{i+1} i β_i següent:

$$\beta_{i+1} = \gamma_0 \xrightarrow{\Omega} \gamma_1 \xrightarrow{q-e} \gamma_2 \xrightarrow{q-e} \cdots \xrightarrow{q-e} \gamma_m \xrightarrow{\Omega} \gamma_{m+1} = \beta_i$$

On γ_0, γ_1 i γ_m, γ_{m+1} estan relacionades només per un moviment elemental i cada parell γ_j, γ_{j+1} per el moviment quasi-elemental que consisteix en la substitució de $P'_jQ_j \cup Q_jQ_{j+1}$ per $P'_jP'_{j+1} \cup P'_{j+1}Q_{j+1}$ per $j = 1, \ldots, m$.

Utilitzant una altra vegada l'exhaustivitat de φ podem trobar elements $X_1, \ldots, X_{m+1}$ de G tals que $\varphi(X_j) = \gamma_j$ amb $j = 1, \ldots, m+1$. El nostre objectiu és expressar cada γ_j en funció dels generadors σ_i, per tal d'obtenir explícitament els X_j.

Notem que cada corda que entri al quadrilàter amb vèrtexs $P'_jQ_jQ_{j+1}P'_{j+1}$ per sobre (sota) d'un arc, haurà de sortir per sobre (sota) del mateix arc com veiem en la Figura 17. Així desfent l'encreuament que crea, ja que per construcció no hi ha encreuaments a cap triangle $P'_jQ_jP_j$ contingut a $P'_jQ_jQ_{j+1}P'_{j+1}$.

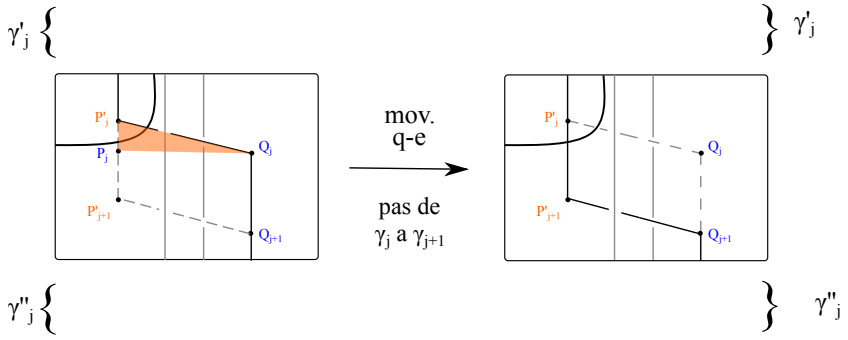


Figura 17: Substituïm $P'_jQ_j \cup Q_jQ_{j+1}$ per $P'_jP'_{j+1} \cup P'_jQ_{j+1}$

Ara si expressem el trenat de γ_j i γ_{j+1} en funció dels generadors, obtenim que $\gamma_j = \gamma'_j\sigma_j^{e_j}\sigma_{j+1}^{e_{j+1}} \ldots \sigma_l^{e_l}\gamma''_j = \gamma_{j+1}$; $e_k = \pm 1$ $k = j, \ldots, l$. On les paraules γ'_j i γ''_j denoten la part que tenen en comú les trenes γ_j i γ_{j+1}, és a dir, els encreuments anteriors i posteriors al quadrilàter $P'_jQ_jQ_{j+1}P'_{j+1}$.

Per tant, $\gamma_j = \gamma_{j+1}$ i podem expressar $X_j = X_{j+1} = X'_jx_j^{e_j}x_{j+1}^{e_{j+1}} \ldots x_l^{e_l}X''_j$ on X'_j i X''_j són les preimatges de γ'_j i γ''_j. Aleshores acabem de veure que $X_j = X_{j+1}$ i per l'arbitrarietat del subíndex $j \in \{0, \ldots, m\}$ tenim que $X_0 = X_{m+1}$ $(X_0 = X_1 = \ldots = X_m = X_{m+1})$

i com $\gamma_0 = \beta_{i+1}$ i $\gamma_{m+1} = \beta_i$ tenim que:

$$\begin{array}{ccccccccc} \beta' = \beta_s & \longrightarrow & \cdots & \longrightarrow & \beta_{i+1} = \gamma_0 & \longrightarrow & \gamma_1 & \longrightarrow \cdots \longrightarrow & \gamma_{m+1} = \beta_i \\ \uparrow & & & & \uparrow & & \uparrow & & \uparrow \\ g' = w_s & & & & w_{i+1} = X_0 & & X_1 & & X_{m+1} = w_i \end{array}$$

$$\begin{array}{ccccccccccccc} \beta' = \beta_s & \longrightarrow & \cdots & \longrightarrow & \beta_{i+1} = \gamma_0 & \longrightarrow & \gamma_1 & \longrightarrow & \cdots & \longrightarrow & \gamma_{m+1} = \beta_i & \longrightarrow \cdots \longrightarrow & \beta_1 = \beta \\ \uparrow & & & & \uparrow & & \uparrow & & & & \uparrow & & \uparrow \\ g' = w_s & = \cdots = & & & w_{i+1} = X_0 & = X_1 & & = \cdots = & & & X_{m+1} = w_i & = \cdots = & w_1 = g \end{array}$$

A més, com $w_i = X_{m+1} = X_0 = w_{i+1}$, per l'arbitrarietat de i arribem a que $w_1 = w_s$ i això implica que $g = g'$. Aquest procediment es pot aplicar per a tot $g, g' \in G$ i per tant, φ és injectiva. Això implica que φ és un isomorfisme. □

3 Teorema d'Alexander

El Teorema d'Alexander s'anomena així perquè va ser provat per primer cop l'any 1923 per James Wadell Alexander. Posteriorment es va veure que aquest teorema pot ser provat de diverses formes, encara que no totes les proves dónen un mètode computable el qual donat un nus arbitrari trobi la corresponent clausura d'un trena. L'any 1990, Vogel va descobrir un algorisme que la troba i que resol el Teorema d'Alexander. En aquesta secció veurem una demostració del Teorema d'Alexander basada en la prova original de de 1923 i una altra feta a partir de l'algorisme de Vogel amb optimitzacions de Yamada, a aquest algorisme l'anomenarem *algorisme de Yamada-Vogel*.

Teorema 3.1 (Teorema d'Alexander)**.** *Tot nus, i tot enllaç, es pot representar com la clausura d'una trena*

3.1 Demostració mitjançant l'algorisme de Yamada–Vogel

Per a poder dur a terme aquesta demostració de manera satisfactòria, primer hem de introduir una sèrie de conceptes nous.

Definició 3.2. La *clausura d'un trena* (o trena tancada) es defineix a partir de la identificació de cadascun dels punts P_i del pla ϵ_1^2 amb el seu corresponent punt Q_i a ϵ_0^2. És equivalent a connectar els punts P_i i Q_i mitjançant sèries d'arcs concèntrics com es mostra en la Figura 18.

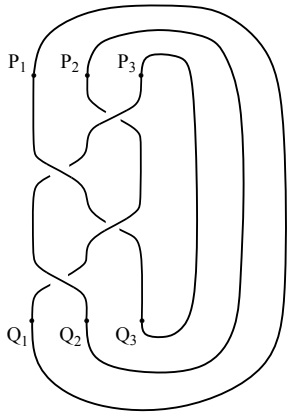


Figura 18: Clausura de la trena $\sigma_2^{-1}\sigma_1\sigma_2^{-1}\sigma_1$

Definició 3.3. Un *nus* és un subespai $K \in \mathbb{R}^3$ homeomorf a la circumferència S^1.

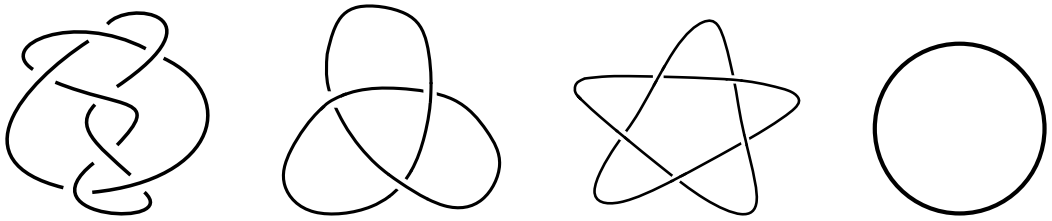


Figura 19: Exemples de nusos de 7, 3 i 5 encreuaments i el nus trivial K_0

Intuïtivament, podem pensar en els nusos de les cordes dels cordons però en matemàtiques els dos extrems de la corda estan connectats i la corda no té gruix. La majoria dels cops considerem la projecció del nus a $\mathbb{R}^2$.

Definició 3.4. L'*índex-trena* d'un nus és el mínim nombre de cordes que pot tenir una trena, la clausura de la qual sigui aquest nus.

Definició 3.5. Un *enllaç* és una col·lecció de nusos lligats entre si i amb intersecció buida.

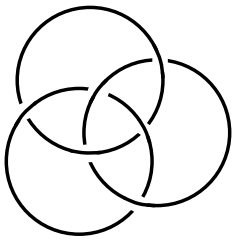
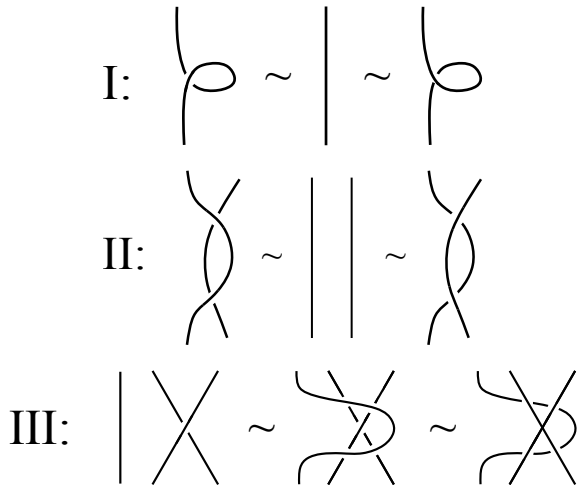


Figura 20: Exemple d'un enllaç

Diem que dos nusos, i dos enllaços, són equivalents si un pot ser deformat fins esdevenir l'altre sense que cap corda travessi una altra del propi nus o d'un altre dels que forma l'enllaç. Hi ha tres tipus de moviments fonamentals mitjançant els quals es pot transformar la projecció d'un nus, o d'un enllaç, en qualsevol altre projecció d'aquest mateix nus o enllaç. Aquests moviments s'anomenen *moviments de Reidemeister* i poden ser de tres tipus:



De fet, aquests tres moviments permeten caracteritzar l'equivalència de nusos.

Teorema 3.6 (Teorema de Reidemeister)**.** *Dues projeccions de nusos o d'enllaços són* equivalents *si i només si una es pot transformar en l'altre mitjançant moviments de Reidemeister.*

Al considerar la clausura d'una trena en comptes de cordes disconnexes tenim un nus o bé un enllaç, per tant aquesta trena tancada pot deformar-se en altres projeccions d'ella mateixa com qualsevol altre nus.

3.1.1 L'algorisme de Seifert

Un cop introduïts certs conceptes de la teoria de nusos, passem a parlar d'alguns altres que ens resultaran especialment útils per a l'algorisme de Yamada–Vogel. Començarem amb l'*algorisme de Seifert*, ja que el diagrama que n'obtindrem d'ell serà el que utilitzarem per a iniciar l'algorisme de Yamada–Vogel.

El *diagrama de Seifert* d'un nus està format per un cert nombre de cercles anomenats *cercles de Seifert* connectats per línies. Aquest es construeix eliminant tots els encreuaments del nus. Eliminarem els encreuaments desconnectant els extrems de cadascun d'ells i movent les cordes que el formen de manera que quedin paral·leles, així desfem l'encreuament i les tornem a connectar per on haviem tallat. Tot i que ara la corda del encreuament que connectava amb l'extrem superior, es reconnectarà amb l'inferior i la que connectava amb l'extrem inferior quedarà connectada al superior. Com queda il·lustrat en la següent figura:

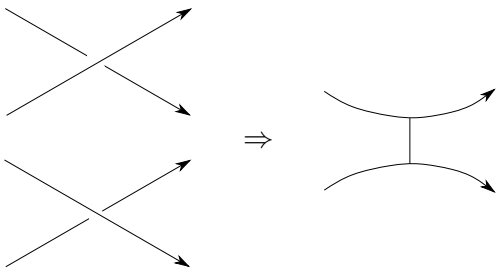


Figura 21: Manera de reconnectar encreuaments a l'algorisme de Seifert

Els cercles de Seifert resultants, restaran connectats amb línies on abans hi havia encreuaments, i com sempre reconnectarem les cordes seguint les seves orientacions aquestes només poden reconnectar-se d'una sola forma. Un cop eliminats tots els encreuaments seguint aquest procediment, que anomenarem *algorisme de Seifert*, obtindrem un diagrama de Seifert.

Observació 3.7. Podem extendre els cercles de Seifert per tal de veure amb claredat les regions que es troben entre ells, cosa que ens serà útil més endavant.

Definició 3.8. Direm que un parell de cercles de Seifert són *coherents* si:

(a) Es troben un dins l'altre i tenen la mateixa orientació, o bé,

(b) Un no està a l'interior de l'altre i estan orientats en direccions oposades.

Si els cercles no satisfan cap de les dues condicions direm que són *incoherents*.

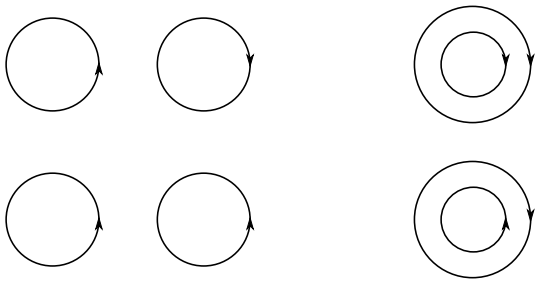


Figura 22: Els cercles de la primera fila són coherents i els de la segona incoherents

Observació 3.9. Qualsevol parell de cercles de Seifert connectats per un encreuament, és coherent. Ja que les noves parts disconnexes sempre quedaran orientades en direccions oposades.

Definició 3.10. L'*alçada* del diagrama D d'un nus, denotada per $h(D)$, es defineix com el nombre de parells de cercles de Seifert incoherents que trobem en el seu corresponent diagrama de Seifert.

A continuació descrivim l'algorisme de Yamada–Vogel. El seu objectiu serà obtenir, a partir d'un nus inicial, una projecció seva que tingui com a diagrama de Seifert un conjunt coherent de cercles que es trobin un a l'interior de l'altre. Aquest algorisme és interessant per a demostrar el Teorema d'Alexander perquè la projecció que s'obté aplicant-lo és la representació de la clausura d'una trena, com veiem en la següent figura:

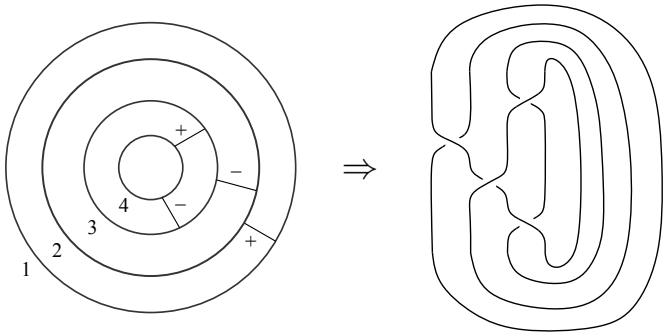


Figura 23: Un diagrama de Seifert de cercles coherents concèntrics representa la clausura d'una trena

Els signes positius i negatius del diagrama que s'obté ens indiquen si la corda de la clausura de la trena va per sobre o per sota, tal i com veurem amb detall quan descriguem l'algorisme de Yamada–Vogel.

3.1.2 L'algorisme de Yamada–Vogel

Abans de passar a descriure l'algorisme detalladament definim també quins són els moviments que utilitzarem. Un és el moviment II de Reidemeister i l'altre s'anomena *canvi a l'infinit*. Aquest últim pot ser utilitzat en qualsevol vora que delimiti la regió exterior del nus, i consisteix intuïtivament en donar la volta a una vora per a que quedi a l'altra banda del nus. I el moviment II de Reidemeister el realitzarem en parelles de vores de cercles incoherents de la projecció del nus, lliscant el cercle incoherent C_i sobre (o sota) l'altre cercle incoherent C_j. Això ens portarà a substituir dos cercles de Seifert disconnexos i incoherents en el diagrama de Seifert, per un parell de cercles de Seifert que es trobin un dins l'altre i coherents, també afegint dos nous encreuaments. Anomenarem *maniobra Q* a aquest procediment.

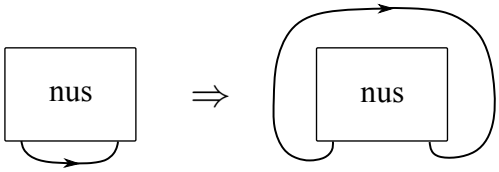


Figura 24: Canvi a l'infinit a un nus qualsevol

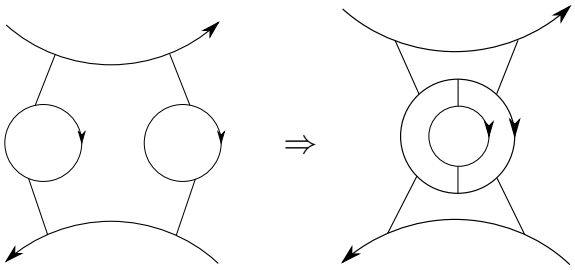


Figura 25: Maniobra Q realitzada a dos cercles incoherents

L'*algorisme de Yamada–Vogel* s'inicia amb l'aplicació de l'algorisme de Seifert a un nus, o un enllaç, orientat K i té l'objectiu d'anar rebaixant l'alçada del diagrama de Seifert obtingut fins a que aquesta esdevingui 0. Aleshores, tots els cercles del diagrama seran coherents i efectuant canvis a l'infinit obtindrem un diagrama on tots els cercles siguin coherents i concèntrics. Arribar a un diagrama d'aquesta forma és interessant perquè tal i com hem vist en la Figura 23, podrem obtenir la deformació del nus K corresponent a la clausura d'una trena. Un cop explicat a grans trets en que consisteix aquest algorisme, passem a veure els passos que segueix:

1. Sigui D el diagrama d'un nus o d'un enllaç orientat K. Eliminem tots els encreuaments mitjançant l'algorisme de Seifert, així obtenint n cercles de Seifert $C_1, \ldots, C_n$. Designem un signe positiu o negatiu en cadascuna de les línies que s'han format al eliminar un encreuament, aquests signes ens seran útils per a saber quina corda va per sobre i quina va per sota al arribar a la representació de la clausura d'una trena al acabar l'algorisme. Hi escriurem un $(+)$ per als encreuaments positius (també anomenats *encreuaments dretans*) i escriurem un $(-)$ pels negatius (o *encreuaments esquerrans*), descrits en la Figura 26. Ara un cop tenim el diagrama de Seifert corresponent al nostre diagrama D, fixem-nos que qualssevol dos cercles connectats per un arc amb signe positiu o negatiu, són necessàriament coherents com havíem comentat anteriorment.

Figura 26: Encreuament negatiu (o esquerrà) i encreuament positiu (o dretà)

2. Si $h(D) = 0$, el nus o enllaç no té cap parella de cercles incoherents i podem passar directament al pas 4. En el cas que $h(D) > 0$ podem trobar almenys una *regió defectuosa*, tal i com veurem en el Lema 3.15. Aquesta es defineix com una regió del diagrama de Seifert la frontera de la qual conté almenys un parell de vores pertanyents a cercles de Seifert incoherents i diferents. Per tant, són les úniques regions en les que podrem aplicar la maniobra Q. Realitzem la maniobra Q a les vores tot just esmentades, tenint en compte que si llisquem C_i per sobre de C_j un dels nous encreuaments té signe positiu i l'altre té signe negatiu, però que si llisquem C_i per sota de C_j els signes dels nous encreuaments ara es trobaran al revés designats.

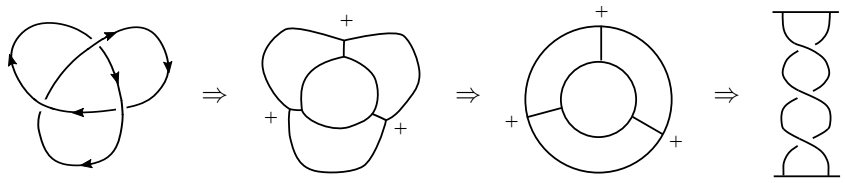
3. Continuem realitzant maniobres Q a parells de vores de cercles incoherents fins que el diagrama tingui altura 0.

4. Un cop $h(D) = 0$, és a dir, quan no quedi cap zona defectuosa realitzarem canvis a l'infinit fins que tots els cercles de Seifert quedin un dins l'altre –en el Lema 3.17 veurem que no caldran més de $n/2$ moviments.

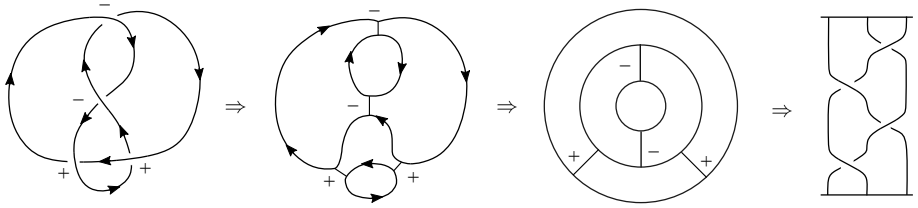
Un cop finalitzat l'algorisme, tenim un diagrama format per cercles concèntrics que sabem que representarà la clausura d'una trena com hem vist a la Figura 23. Per llegir quina trena serà aquesta enumerem els cercles del diagrama de l'exterior a l'interior, sent 1 el cercle més exterior i n el més interior. Cal notar que la trena resultant tindrà tantes cordes com cercles té el diagrama. Ara ens col·locarem a la posició que correspondria a les "12 en punt" i recorrem els cercles en sentit antihorari. Cada cop que trobem una connexió entre dos cercles apuntarem la n-trena elemental a la que correspon. Aquesta serà σ_i si la connexió té signe positiu i serà σ_i^{-1} si el signe és negatiu, on i fa referència al nombre del cercle més exterior dels connectats. Un cop apuntades totes les connexions de la forma esmentada, ens quedarà una paraula de la forma $\sigma_{i_1}^{e_1} \ldots \sigma_{i_n}^{e_n}$ amb $i_1, \ldots, i_n \in 1, \ldots, n$ i $e_j \in \{\pm 1\}$ per a tot $j = 1, \ldots, k$ que ens indicarà la trena que buscàvem escrita com a composició de trenes elementals.

Passem a veure alguns exemples de l'algorisme ara descrit en la projecció d'un nus.

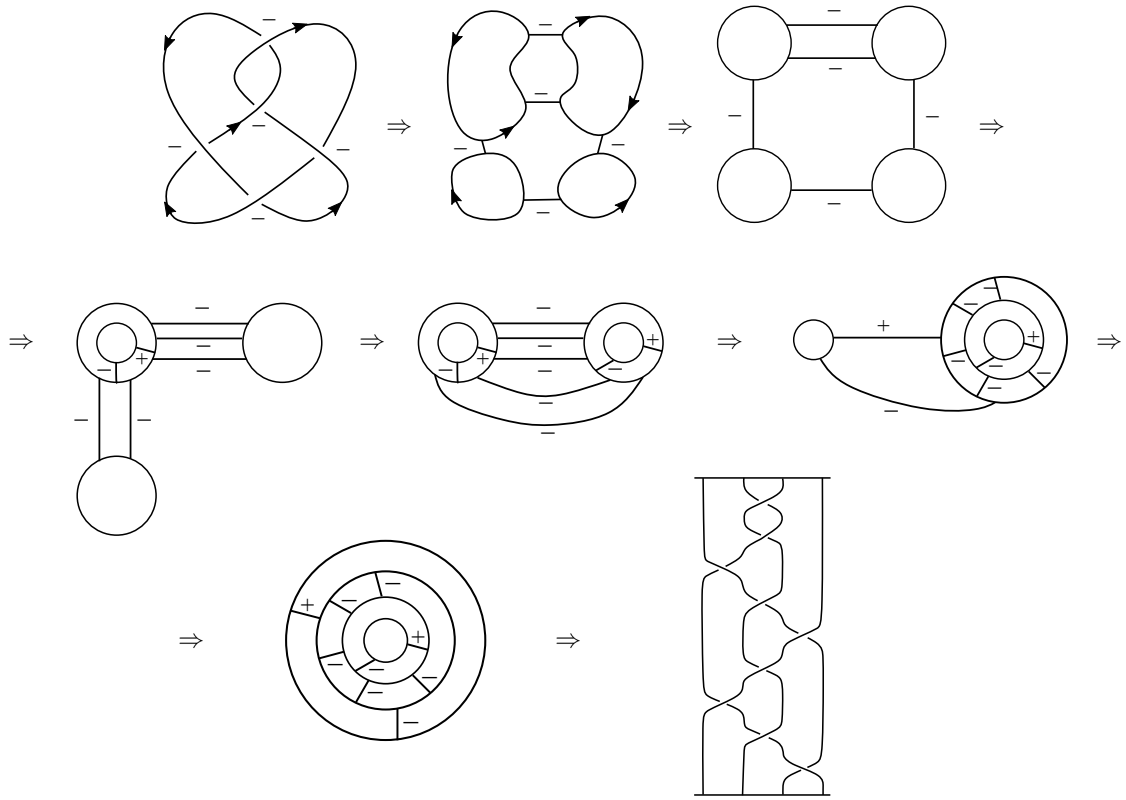
Exemple 3.11. Si apliquem l'algorisme de Yamada–Vogel al nus de 3 encreuaments, obtenim la trena $\sigma_1\sigma_1\sigma_1$.



Exemple 3.12. Si apliquem l'algorisme de Yamada–Vogel al nus de 4 encreuaments, obtenim la trena $\sigma_2^{-1}\sigma_1\sigma_2^{-1}\sigma_1$.



Exemple 3.13. Si apliquem l'algorisme de Yamada–Vogel al següent nus de 5 encreuaments, obtenim la trena $\sigma_2^{-1}\sigma_2^{-1}\sigma_1\sigma_2^{-1}\sigma_3^{-1}\sigma_2^{-1}\sigma_1^{-1}\sigma_2^{-1}\sigma_3$. En la figura es pot veure que utilitzem la maniobra Q dos cops i finalitzem amb dos canvis a l'infinit.



3.1.3 Demostració del Teorema d'Alexander

Un cop descrit l'algorisme de Yamada–Vogel, veguem un teorema i un seguit de lemes que necessitarem per a justificar que mitjançant aquest algorisme es demostra el Teorema d'Alexander.

Teorema 3.14. *Una regió amb més de dos cercles de Seifert intersecant l'àrea que delimita ha de tenir (almenys) un parell de cercles de Seifert incoherents.*

Demostració. Considerem una regió R que està delimitada per només dos cercles de Seifert coherents, cap dins l'altre. Llavors tenim el cercle C_i orientat en sentit horari i C_j orientat en sentit antihorari. Al afegir un tercer cercle de Seifert a la regió, aquest serà incoherent amb C_i si està orientat en sentit horari, o bé serà incoherent amb C_j si està orientat en sentit antihorari.

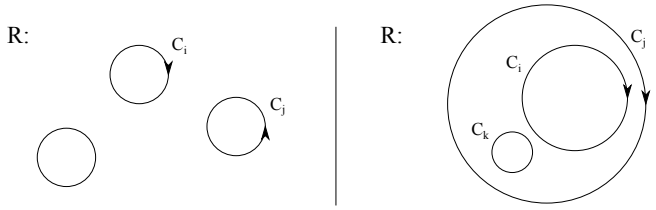


Figura 27: Afegim un tercer cercle a C_i, C_j coherents cap a l'interior de l'altre (esquerra) i a C_i, C_j coherents un dins l'altre (dreta)

De manera similar veiem que si tenim dos cercles coherents, amb C_i dins de C_j, aleshores han de tenir la mateixa orientació que suposarem horària. Ara si col·loquem un tercer cercle C_k dins de C_j, si aquest té orientació horària serà incoherent amb C_i perquè no està dins seu i tenen orientacions iguals, però si la seva orientació fos antihorària seria incoherent amb C_j doncs es trobaria dins seu amb una orientació oposada. Anàlogament obtenim el mateix resultat per C_i, C_j amb orientació antihorària. □

Lema 3.15. *Si un diagrama de Seifert té un parell de cercles incoherents, llavors té una regió defectuosa.*

Demostració. Denotarem la orientació del cercle de Seifert C_i per $O(C_i)$ i aquesta serà o bé en sentit horari o en antihorari. I denotarem que el cercle de Seifert C_a es troba dins del cercle de Seifert C_b com $C_a(n)C_b$.

Suposem que no hi ha regions defectuoses i arribem a contradicció. Si $C_a(n)C_b$, aleshores $O(C_a) = O(C_b)$, ja que sinó serien incoherents i la regió que es troba entre els cercles seria defectuosa. Tampoc no hi pot haver cap cercle C_c a la regió delimitada per C_a i C_b, ja que en la regió hi hauria més de dos cercles i, com hem vist en el Teorema 3.14, tindríem un parell que seria incoherent. Aleshores no poden existir cercles C_a i C_c tal que $C_a(n)C_b$ i $C_c(n)C_b$.

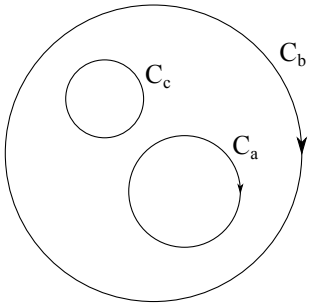


Figura 28: Afegim C_c a l'interior de C_b, amb $C_a(n)C_b$ coherents

Considerem un cercle C_1 que delimiti la regió exterior. Existeixen cercles de Seifert $C_i(n)C_{i-1}(n)\dots(n)C_1$ els quals són els únics cercles possibles dins de C_1, ja que no pot haver més d'un cercle a l'interior immediat de cadascun, com hem vist al paràgraf anterior. Per cada $k = 1, \dots, i-1$ es compleix la igualtat $O(C_k) = O(C_{k+1})$ ja que aquests delimiten una regió i hem suposat que no n'hi havia cap de defectuosa, per tant, si estan un dins l'altre han de tenir la mateixa orientació. Aleshores, es compleix $O(C_i) = \dots = O(C_1)$.

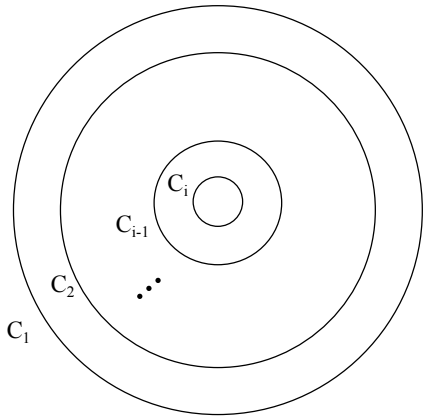


Figura 29: Cercles de Seifert $C_i(n)C_{i-1}(n)\dots(n)C_1$

Si l'exterior està només delimitat per un únic cercle, C_1, hem acabat perquè tots els cercles que es troben un dins l'altre tenen la mateixa orientació, cosa que els fa tots coherents. Ara considerem que l'exterior està delimitat per dos cercles, C_1 i C_1'. Anàlogament al procés dut a terme per C_1 obtenim que si existeixen cercles $C_i'(n)C_{i-1}'(n)\dots(n)C_1'$ aleshores $O(C_i') = \dots = O(C_1')$. Però degut a que C_1 no es troba dins de C_1', ni a la inversa, tenim que $O(C_1) \neq O(C_1')$. Ja que sinó serien incoherents i la regió exterior seria defectuosa. Això significa que per a qualsevol i, j tenim que $O(C_i) \neq O(C_j')$. Per tant, concloem amb que dos cercles qualssevol estaran un dins l'altre si tenen la mateixa orientació, i no ho estaran si tenen orientació oposada, sent aleshores sempre coherents.

Si exigim que no hi hagi regions defectuoses, tots els cercles de Seifert seran coherents. Per tant, si tenim un parell de cercles de Seifert incoherents, per força hi ha d'haver una regió defectuosa. □

Lema 3.16. *Realitzar la maniobra Q en el diagrama d'un nus o d'un enllaç redueix en una unitat el nombre de cercles de Seifert incoherents.*

Demostració. Els dos cercles de Seifert als que realitzem la maniobra Q han de tenir la mateixa orientació, ja que han de ser incoherents i no estar un dins l'altre per a realitzar el moviment. Per tant, la resta de cercles de Seifert del diagrama hauran de ser o bé coherents o bé incoherents respecte aquests dos. Quan fem la maniobra Q en els dos cercles esmentats inicialment, obtindrem dos nous cercles orientats en la mateixa direcció que els anteriors però un dins l'altre. Això significa que tots els altres cercles de Seifert seguiran sent o bé coherents o incoherents amb aquests dos, no obstant aquests ara seran coherents l'un amb l'altre. Degut a que aquest és l'únic canvi en la coherència dels cercles de Seifert, el nombre de cercles incoherents disminueix en 1. □

Lema 3.17. *Qualsevol projecció d'un nus o d'un enllaç en la que tots els cercles de Seifert són coherents pot transformar-se en la representació de la clausura d'una trena amb no més de $n/2$ canvis a l'infinit, on n és el nombre de cercles.*

Demostració. Tenim dos conjunts de cercles de Seifert, cadascun dels quals conté un seguit de cercles de Seifert col·locats un a l'interior de l'altre. Com aquests conjunts han de tenir direccions oposades, degut a que són coherents, podem realitzar un canvi a l'infinit a la vora que delimita la regió exterior d'un dels conjunts, fent que ara el cercle al que hem aplicat el canvi el infinit esdevingui el cercle exterior de l'altre conjunt. Si n era el nombre total de cercles de Seifert, ara el conjunt més petit d'entre aquests dos haurà de tenir menys de $n/2$ cercles de Seifert (com a màxim en tindrà $n/2 - 1$). Per tant, el nombre total de cercles a traslladar d'aquest conjunt a l'altre per tal de que tots els cercles quedin un a l'interior de l'altre serà més petit o igual que $n/2$. És a dir, es faran com a màxim $n/2$ canvis a l'infinit.

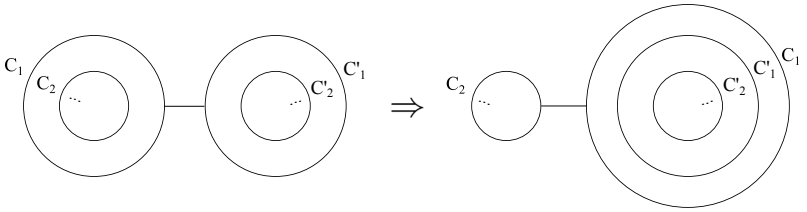


Figura 30: Canvi a l'infinit del conjunt dels C_i als C'_i

□

Demostració (Teorema d'Alexander). Sigui K un nus o un enllaç qualsevol amb diagrama D, apliquem-hi l'algorisme de Yamada–Vogel. L'alçada $h(D)$ serà 0 només en el cas que el diagrama de Seifert estigui compost únicament de dos cercles de Seifert coherents, ja que pel Teorema 3.14 sabem que tota regió amb més de dos cercles de Seifert intersecant l'àrea que delimita aquesta regió, ha de tenir un parell de cercles de Seifert incoherent i per tant tindríem $h(D) > 0$. Pel Lema 3.15 aleshores sabem que hi haurà regions defectuoses, i això significa que podem realitzar la maniobra Q com indica l'algorisme. Aquesta maniobra Q sabem pel Lema 3.16 que redueix en una unitat el nombre de cercles de Seifert incoherents cada cop que es realitza. Sempre que resti algun parell de cercles incoherents podem aplicar la maniobra Q a l'àrea defectuosa que delimiten, així arribant a la projecció en que tots els cercles de Seifert són coherents. Ara un cop estem en l'últim pas de l'algorisme, gràcies al Lema 3.17 sabem que utilitzant només $n/2$ canvis a l'infinit arribem a un diagrama on tots els cercles de Seifert es troben un dins l'altre. I com sabem que tota projecció d'un nus o enllaç pot arribar a tenir aquesta forma i que tot

nus o enllaç amb un diagrama com aquest és la clausura d'una trena, haurem demostrat el Teorema d'Alexander mitjançant l'algorisme de Yamada–Vogel. □

Corołari 3.18. *L'índex-trena d'un nus o enllaç és més petit o igual que el nombre mínim de cercles de Seifert en qualsevol projecció del nus.*

Demostració. Cada cercle de Seifert representa una corda de la trena al final de l'algorisme. I cap moviment de l'algorisme no treu ni afegeix cercles de Seifert, només els reordena. Així que el nombre més petit possible de cordes no pot ser més gran que el nombre de cercles de Seifert amb el que comencem. □

3.2 Demostració basada en la original d'Alexander

Un cop vista la demostració del Teorema d'Alexander mitjançant l'algorisme de Yamada–Vogel, passem a veure la demostració alternativa basada en la original del 1923 feta per Alexander.

Demostració (Teorema d'Alexander): Sigui K un nus o un enllaç i D un diagrama de K, considerem un rectangle que englobi tot el diagrama i apliquem el següent procediment:

(a) Prenem un punt p que es trobi a D i que no sigui un encreuament i retallem el diagrama per aquest punt. Obtindrem dos extrems de la "corda" que tot just hem tallat.

(b) Portem l'extrem superior a la base superior del rectangle i l'extrem inferior a la base inferior. Denotem per p_1 i p_1' els punts d'intersecció dels extrems amb aquestes bases.

Notem que la unió de p_1 i p_1' mitjançant un arc exterior al rectangle ens dóna un diagrama equivalent a D. Com la base inferior i superior del rectangle després seran les barres inferior i superior de la trena que en resultarà, els arcs exteriors que uniran p_i i p_i' seran els de la clausura de la trena resultant, la qual tindrà aleshores el mateix diagrama que el nus. Il·lustrem aquest procediment:

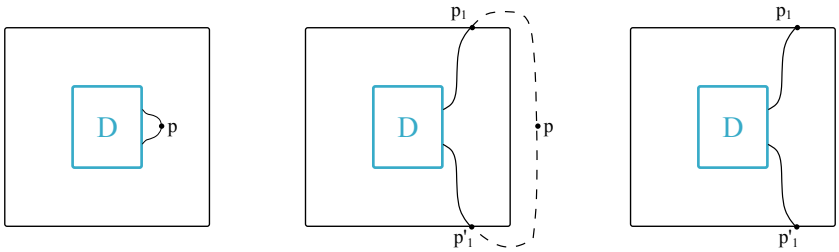


Figura 31: Agafant un punt p de D que no sigui un encreuament obtenim aquesta equivalència de diagrames

Degut a que la corba del diagrama que uneix p_1 amb p_1' comença a la part superior del rectangle i acaba a la inferior, aquesta corba tindrà el mateix nombre de mínims locals que de màxims locals. Sigui $m \in \mathbb{N}$ el nombre de parelles de màxims i mínims, podem diferenciar dos casos:

- En el cas que no hi hagi cap mínim ni cap màxim ($m = 0$) ens trobem davant el diagrama de la trena trivial d'un sola corda i aleshores K seria el nus trivial K_o.

- En el cas que almenys hi hagi un màxim i un mínim locals ($m > 0$), prenem dos punts a i b de la corba abans esmentada, de manera que a i b siguin un mínim i un màxim respectivament i que en l'arc ab no hi hagi cap altre màxim ni mínim. Notem que ab pot creuar-se amb altres arcs en el diagrama. Aleshores sigui $n \in \mathbb{N}$ el nombre d'encreuaments que trobem en ab, escollim $n+1$ punts de l'arc ab, que denotarem com: $a = a_0, a_1, \ldots, a_n = b$. Aquests punts els haurem d'agafar de manera que $a_i a_{i+1}$ creui només un únic cop amb un altre arc per a tot $i = 0, \ldots, n-1$.

  Ara fixem punts p_2 i p_2' a la dreta de p_1 i p_1' respectivament i substituïm $a_0 a_1$ per $a_0 p_2' p_2 a_1$, on l'arc $p_2' p_2$ connecta els punts p_2' i p_2 per fora del rectangle com mostra la següent figura:

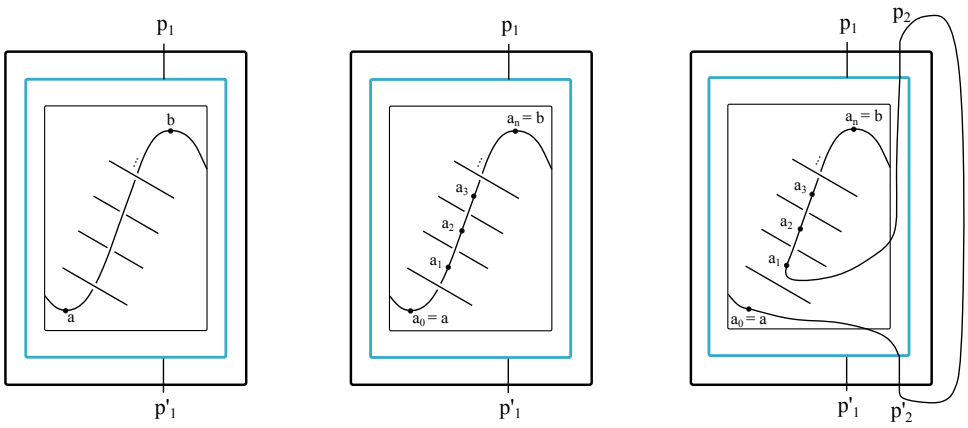


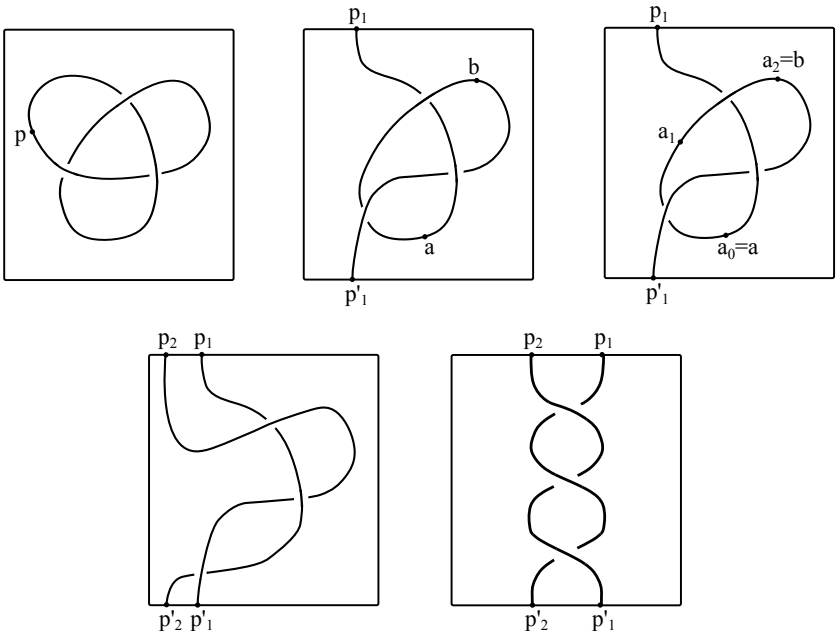
Figura 32: Procés mitjançant el qual hem eliminat un mínim local del diagrama d'un nus o enllaç

D'aquesta manera hem eliminat el mínim local $a_0 = a$, i ara a_1 és un mínim del diagrama equivalent que hem obtingut.

Repetint n cops més aquest procediment, obtindrem un diagrama equivalent a D en el qual ni a serà un mínim ni b serà un màxim. Per tant, haurem disminuït en una unitat el nombre de màxims i mínims que presentava el diagrama original. Ara bé, si apliquem aquest algorisme als $m-1$ parells restants de màxims i mínims locals de la nova corba obtinguda, arribarem a tenir una trena en el rectangle, la clausura de la qual és per construcció el nus o l'enllaç de partida. □

Repetim l'Exemple 3.11 ara a partir de l'algorisme que ens dóna aquesta demostració i vegem que arribem a la mateixa trena.

Exemple 3.19. La següent figura ens mostra l'algorisme utilitzat en la demostració d'Alexander basada en la original en el nus de 3 encreuaments.



4 Algorisme per obtenir la trena que té com a clausura un cert nus

En la secció anterior hem demostrat el Teorema d'Alexander de dues maneres diferents, les quals tenen en comú que segueixen un algorisme partint d'un nus fins arribar a la forma de clausura d'una trena a partir de la qual obtenim la trena concreta que té com a clausura el nus inicial. En aquesta secció veurem un programa que també utilitza un algorisme, el qual va ser ideat per Choi i està basat en el de Yamada–Vogel, per a arribar a la trena que té com a clausura un nus, o un enllaç, donat.

4.1 Idea de l'estructura de l'algorisme

L'algorisme s'inicia introduint un codi al qual anomenarem *labelled peer code*, a partir d'aquest podrem obtenir tota la informació referent al nus del qual volem trobar la trena que el té com a clausura. Mitjançant aquesta informació obtinguda trobarem una serie de cicles: uns ens permetran obtenir el diagrama de Seifert del nus i uns altres ens permetran saber si podem fer una maniobra Q per tal de reduir en una unitat el nombre de parelles de cercles incoherents en el diagrama de Seifert. Això és important ja que el nostre objectiu, tal i com en l'algorisme de Yamada–Vogel, serà arribar a un diagrama de Seifert on no hi hagi cap parella de cercles incoherents. Realitzant un canvi a l'infinit adequat en aquest diagrama arribem a un conjunt de cercles concèntrics i orientats tots en la mateixa direcció a partir del qual llegirem la paraula que designa la trena que volíem trobar.

Cal destacar que aquest algorisme l'he programat personalment amb llenguatge Python i es pot trobar a l'Annex A.

4.2 Definicions i conceptes preliminars

4.2.1 Peer code i labelled peer code

Per a elaborar el codi que hem d'introduir a l'algorisme sigui D un diagrama orientat d'un nus o d'un enllaç amb k components, sempre considerarem la seva immersió en el pla amb n encreuaments (als quals els hi direm vèrtexs) i $2n$ arestes. Respecte la orientació de D, cada vèrtex té dues arestes que surten del vèrtex i dues que n'arriben. A aquestes últimes les anomenarem *peers*.

Donat el diagrama D escollim una aresta d'una certa component a partir de la qual començarem a numerar des del 0 totes les arestes d'aquesta mateixa component seguint la orientació del diagrama. Un cop les haguem numerat totes continuarem pel nombre en el que havíem deixat la numeració, ara en una aresta d'una altra component la qual arribi a un vèrtex que ja tingui numerada l'altra aresta d'arribada. Continuarem la numeració seguint la orientació de D fins a tenir totes les arestes del diagrama numerades.

Tal i com estan numerades les arestes a cada vèrtex arribarà una aresta amb un nombre parell, la qual anomenarem *naming edge* del vèrtex al que arriba. Ja que sigui aquesta aresta $2i$, i serà el nombre que tindrà designat el vèrtex al que aquesta aresta arriba.

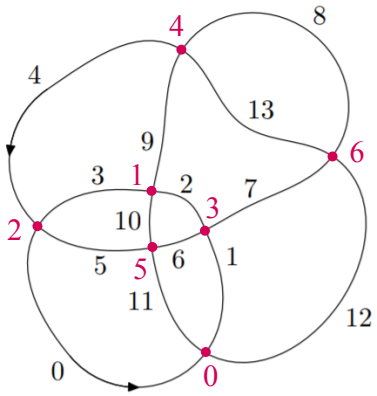
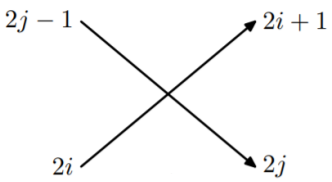


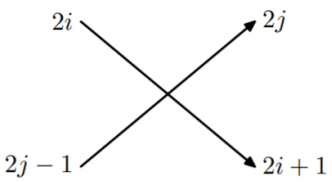
Figura 33: Nus amb les arestes numerades i en vermell els vèrtexs segons el seu *naming edge*

És important detallar com quedarà definida la separació en components, ja que hi haurà vèrtexs que pertanyin a dues components alhora. Associarem cada vèrtex amb la component que contingui el *naming edge* d'aquest vèrtex.

Com ja hem dit, sempre hi haurà dues arestes d'arribada i dues de sortida per a cada vèrtex. Però depenent de si els nombres assignats a aquestes arestes són parells o senars ens trobem davant de dos tipus diferents d'encreuaments: els de *tipus I* i els de *tipus II*.



Tipus I



Tipus II

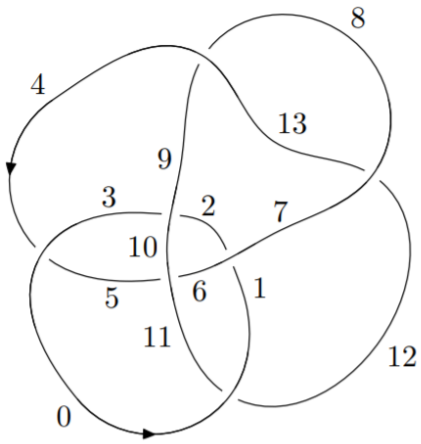
Definició 4.1. Ens referirem com a *peer code* a la llista de *peers* senars ordenats segons els vèrtexs als quals arriben, separats per components amb una coma també segons la component a la qual pertany el vèrtex al que arriben i amb un signe negatiu al davant dels *peers* que vagin a vèrtexs de *tipus I*.

Per exemple, el *peer code* de la Figura 33 és -11 9, -3 1 -13 5 -7.

Un cop tenim el *peer code*, per a elaborar el *labelled peer code* que introduirem al algorisme assignarem un signe + al vèrtex i si la seva *naming edge* $2i$ passa per sobre una altra corda del nus i li assignarem un signe − si aquesta passa per sota.

Definició 4.2. Un *labelled peer code* d'un diagrama D consisteix en un *peer code* de D seguit d'una llista de signes + i/o − assignats als vèrtexs de D i ordenats segons aquests vèrtexs.

Continuant amb l'exemple anterior, el *labelled peer code* del nus seria:

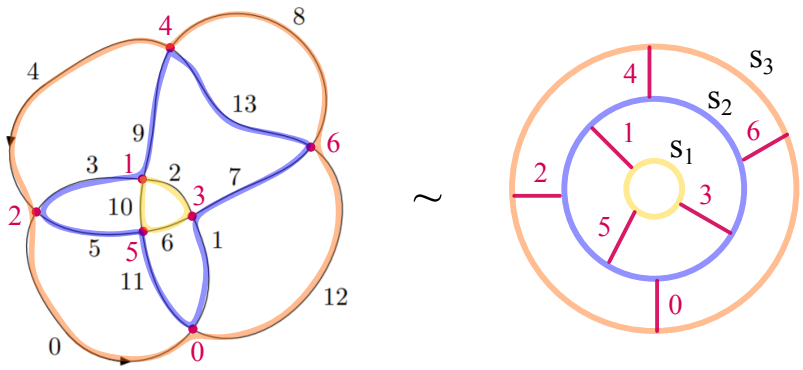


-11 9, -3 1 -13 5 -7 / + − − + − + −

4.2.2 Grafs de Seifert

A partir dels diagrames de Seifert definits a la Secció 3.1.1 definirem els *grafs de Seifert*.

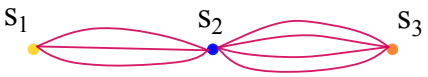
Notem que ara per obtenir un diagrama de Seifert a partir del diagrama d'un nus, els cercles de Seifert vindran donats per cicles d'arestes del diagrama del nus d'una mateixa paritat i les línies que connecten els cercles de Seifert seran els vèrtexs que comparteixen aquests cicles. Il·lustrem-ho seguint el nostre exemple:



Definició 4.3. El *graf de Seifert* Γ d'un diagrama de Seifert D és un graf en el que els vèrtexs de Γ estan en correspondència u a u amb els cercles de Seifert de D i les arestes de Γ estan en correspondència u a u amb el conjunt de línies que connecten els cercles de Seifert a D. Una aresta a Γ connecta dos vèrtexs si i només si la línia corresponent a aquesta aresta connecta els dos cercles de Seifert corresponents a aquests vèrtexs a D.

Podem observar que si dos cercles de Seifert estan connectats per varies línies en un diagrama de Seifert, aleshores els vèrtexs corresponents al graf de Seifert estaran connectats per varies arestes.

Per tant, el graf de Seifert del diagrama de Seifert anterior seria:



Un graf que ens resulta de molta utilitat per a treballar de manera més senzilla en l'algorisme que descriurem és el *graf reduït de Seifert*, derivat del graf de Seifert.

Definició 4.4. El *graf reduït de Seifert* Γ' d'un diagrama de Seifert D és un graf obtingut a partir del graf de Seifert Γ amb les següents propietats:

(i) Γ' té els mateixos vèrtexs que Γ.

(ii) Dos vèrtexs estan connectats per una aresta a Γ' si i només si ho estan a Γ.

(iii) Com a màxim una aresta connecta dos vèrtexs a Γ'.

4.2.3 Propietats heretades del *labelled peer code*

Fem les següents observacions sobre els cercles de Seifert associats a un diagrama descrit mitjançant un *labelled peer code*:

(i) Les línies que connecten dos cercles de Seifert estaran associades o bé totes a vèrtexs de tipus I o bé totes a vèrtexs de tipus II

(ii) Sigui C_1 la component delimitada per l'interior d'un cercle s i C_2 i la component que engloba tot el que es trobi a l'exterior de s. Aleshores totes les línies que connectin el cercle s amb qualsevol altre cercle a C_i estaran associades a vèrtexs d'un mateix tipus. Totes les de C_1 seran d'un tipus i les de C_2 de l'altre.

(iii) Els cercles de Seifert estan composats completament o bé d'arestes parells o bé d'arestes senars.

Notem que la primera observació ens permet assignar un tipus a cada aresta de Γ' depenent de si els cercles de Seifert corresponents als vèrtexs de Γ' estan connectats amb línies associades a un vèrtex de tipus I o II.

Ara introduïm un nou concepte que ens serà d'utilitat a l'algorisme per a identificar amb quines arestes concretes del nus podem fer una maniobra Q. Aquest és el de cicles dretans i esquerrans:

Definició 4.5. Per cada aresta e en el diagrama d'un nus hi ha una seqüència d'arestes $e_0, \ldots, e_k$ amb $e = e_0 = e_k$ anomenada *cicle esquerrà* que s'obté girant a l'esquerra a cada vèrtex que trobem seguint la orientació del diagrama si l'aresta en la que ens trobem es parell, i al contrari de la orientació del diagrama si és senar. Definim els *cicles dretans* de manera anàloga als esquerrans, però girant a dreta en aquest cas.

És clar que cada aresta en un cicle esquerrà (dretà) determina el mateix cicle esquerrà (dretà).

4.2.4 Maniobra Q

La maniobra Q utilitzada a l'algorisme és la ja introduïda a la Secció 3.1.2, seguint els mateixos conceptes de coherència i incoherència de cercles de Seifert i el mateix objectiu d'utilitzar-la per a reduir el nombre de cercles incoherents fins que no en resti cap. No obstant això, en l'algorisme que ara presentem l'ús d'aquest moviment depèn del següent lema:

Lema 4.6. *Sigui v un vèrtex del graf reduït de Seifert Γ' d'un diagrama i $star(v)$ les arestes adjacents a aquest. Si dues de les arestes de $star(v)$ tenen el mateix tipus assignat aleshores dos dels cercles de Seifert corresponents als vèrtexs adjacents a $star(v)$, que no siguin v, són incoherents i admeten una maniobra Q.*

Demostració. Sigui s un cercle de Seifert en un diagrama de Seifert D corresponent al vèrtex v a Γ'. Si $e_1, e_2 \in star(v)$ tenen el mateix tipus assignat aleshores hi ha almenys dos cercles de Seifert connectats amb s que es troben el la mateixa component de s, és a dir que hi ha almenys dos que es troben a l'interior de s o a l'exterior de s. Aquests seran incoherents entre ells, ja que al estar els dos connectats a s els dos seran coherents amb s. Per tant, com son incoherents i es troben en una mateixa component, admeten una maniobra Q. □

Corołari 4.7. *Si un vèrtex v pertanyent al graf reduït Γ' té ordre major que 2, és a dir si té més de dues arestes adjacents, aleshores dos dels cercles de Seifert corresponents als vèrtexs adjacents a $star(v)$ i diferents de v admeten una maniobra Q.*

Després d'un nombre finit de maniobres Q tots els vèrtexs de Γ' tindran ordre menor o igual que 2, i cada vèrtex d'ordre 2 serà adjacent a una aresta de tipus I i una altra de tipus II. En aquest punt el diagrama no podrà contenir cap parell de cercles de Seifert incoherent, i per tant serà de la forma de la clausura d'una trena.

4.3 Descripció de l'algorisme

L'algorisme té com a entrada un *labelled peer code* d'un nus obtingut tal i com hem descrit anteriorment, a partir d'aquest es seguiran els passos següents fins a arribar a tenir com a output la paraula que descriu la trena que buscàvem, la qual anomenarem *braid word*.

1. Crear una taula a partir del *labelled peer code* que descrigui quines són les arestes parells i senars que entren a cada vèrtex (*peers*), quines són les arestes parells i senars que surten de cada vèrtex, el tipus de cada vèrtex (*tipus I* o *II*) i els signes (+ o −) assignats a cada vèrtex.

2. Determinar el conjunt de diferents cicles dretans i cicles esquerrans a partir de la taula elaborada en el pas 1.

3. Determinar els cercles de Seifert com llistes cícliques d'arestes i registrar els vèrtexs pels que passen aquestes arestes, ja que seran les línies que uneixen els cercles de Seifert corresponents en el seu diagrama de Seifert.

4. Construir el graf reduït de Seifert a partir dels cercles de Seifert del pas anterior

5. A partir del graf reduït de Seifert, determinar si hem de fer una maniobra Q per a reduir el nombre de parelles de cercles incoherents o si ja no hi ha cap. Si no hi ha cap, anar directament al pas número 8.

6. Elaborar el *labelled peer code* que tindrà el nou nus després d'haver fet una maniobra Q.

7. Anar un altre cop al pas 1, ara aplicant-hi el nou *labelled peer code* obtingut al pas anterior.

8. Numerar els cercles de Seifert com cordes d'una trena i obtenir la paraula que designa la trena que té com a clausura aquest nus, a partir d'aquests cercles i els signes (+/−) assignats a cada vèrtex.

En les pròximes subseccions descriurem cadascun d'aquests passos amb detall, utilitzant el diagrama d'un nus que tenim a continuació com a exemple:

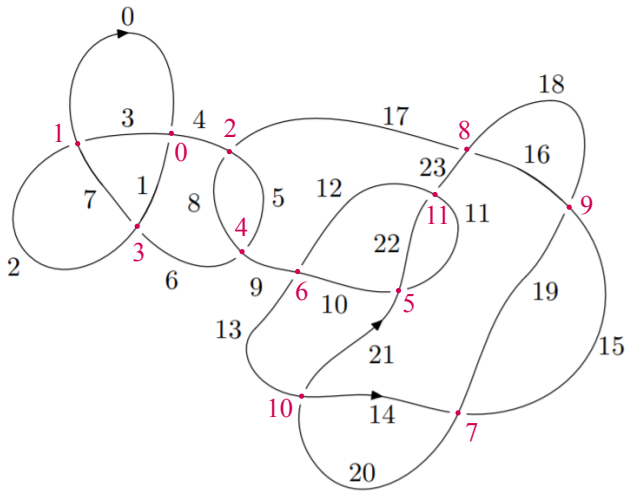


Figura 34: Diagrama d'un nus de tres components

4.3.1 Pas número 1. Crear la taula

Agafarem el *labelled peer code* de la Figura 34 com a exemple amb el que treballar. Aquest és:

$$3\ 7\ \text{-}17\ 1,\ \text{-}5\ 21\ 9\ \text{-}23,\ \text{-}15\ \text{-}13\ 11\ /\ --+-+-------$$

Creem la taula de manera que emmagatzemi en les seves files diferents tipus d'informació obtinguda a través del *labelled peer code*. Per cada vèrtex i emmagatzemem la aresta d'arribada parell $2i$ i la senar $2i+1$, el tipus del vèrtex (*tipus I* o *II*) i el signe associat a vèrtex. A més de registrar per cada vèrtex a quina component pertany el *naming edge* $2i$, començant la numeració de les components des del 0. I a partir d'aquesta informació som capaços d'obtenir les arestes de sortida parells i senars, que també emmagatzemarem.

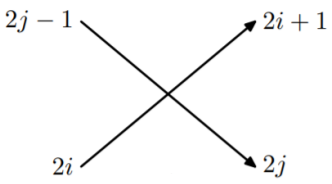
Així, a partir del *labelled peer code* del exemple ens queda la següent taula:

Vèrtex	0	1	2	3	4	5	6	7	8	9	10	11
Tipus associat al vèrtex	*II*	*II*	*I*	*II*	*I*	*II*	*II*	*I*	*I*	*I*	*I*	*II*
Signe associat al vèrtex	−	−	+	−	+	−	−	−	−	−	−	−
Component	0	0	0	0	1	1	1	1	1	2	2	2
Arestes parells d'arribada	0	2	4	6	8	10	12	14	16	18	20	22
Arestes senars d'arribada	3	7	17	1	5	21	9	19	23	15	13	11
Arestes parells de sortida	4	0	8	2	6	22	10	20	18	16	14	12
Arestes senars de sortida	1	3	5	7	9	11	13	15	17	19	21	23

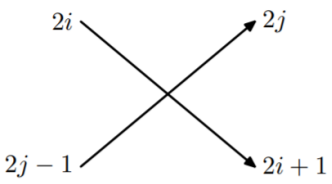
Cal notar que tot i que aquí hem descrit la informació en una taula per a fer-ho més visual, a l'hora de programar l'algorisme emmagatzemem la informació en llistes independents per cada fila, així facilitem el ús de la informació en els passos posteriors. Tindrem les arestes d'arribada parelles a la llista *eventer*, les arestes d'arribada senars a la llista *oddter*, les arestes de sortida parells a la llista *evenori*, les arestes de sortida senars a la llista *oddori* i la fila que descriu la component estarà en una llista anomenada *component*. També el tipus del vèrtex l'emmagatzemarem a una llista anomenada *tipus*, en la que la coordenada i tindrà valor *False* si el vèrtex i és de *tipus I* i tindrà valor *True* si el vèrtex i és de *tipus II*. Així com els signes + i − també estaran assignats a una llista anomenada *label*, en la que la coordenada i tindrà valor *False* si el vèrtex i té assignat − i tindrà valor *True* si té assignat +.

4.3.2 Pas número 2. Determinar els cicles dretans i esquerrans

Descrivim que hem de fer per a determinar els cicles esquerrans ja que trobar els dretans es farà de manera anàloga. Serà important recordar els dos tipus d'encreuaments que teníem:



Tipus I



Tipus II

Per a determinar el conjunt de cicles esquerrans comencem amb l'aresta 0, la recorrem seguint la orientació del diagrama perquè considerem el 0 com a un número parell. En canvi, les arestes amb números senars seran recorregudes en sentit contrari a la orientació del diagrama. Al recórrer l'aresta 0 arribem al vèrtex 0, i com en el nostre exemple el vèrtex 0 és de *tipus II* i venim d'una aresta parell, girar a l'esquerra significarà trobar-nos amb la aresta parell de sortida: 4. Continuem seguint la orientació del diagrama i arribem al vèrtex 2, el qual és *tipus I*, aleshores girar a l'esquerra significarà trobar-nos amb l'aresta d'entrada senar, 17. Ara com 17 és un nombre parell, seguim l'aresta al contrari de la orientació del diagrama i arribem al vèrtex $(17-1)/2 = 8$. Aquest vèrtex també és *tipus I*, aleshores arribant per la aresta senar de sortida al girar a l'esquerra trobarem la aresta parell de sortida, 18. Continuant amb aquest procediment fins que tornem a trobar l'aresta 0 podem determinar el cicle esquerrà $(0, 4, 17, 18, 15, 20, 13, 9, 6, 2)$.

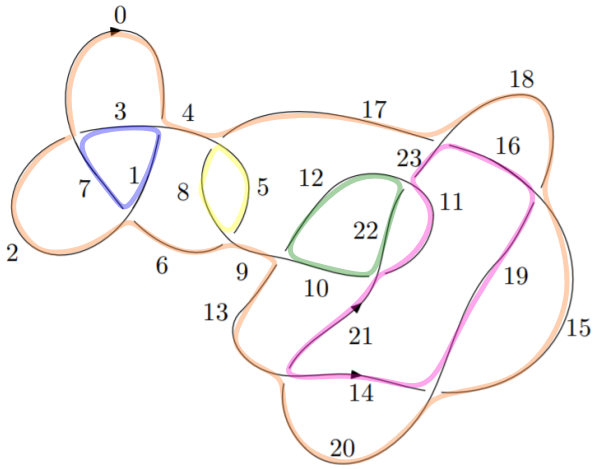


Figura 35: Cicles esquerrans del nus de l'exemple

Per a determinar la resta de cicles esquerrans comencem el mateix procediment amb una aresta que no hi sigui a cap des cicles esquerrans determinats anteriorment, fins que totes les arestes apareguin en algun cicle.

Els cicles esquerrans i dretans del nostre exemple són:

Cicles esquerrans	Cicles dretans
$(13, 9, 6, 2, 0, 4, 17, 18, 15, 20)$	$(16, 18)$
$(19, 16, 23, 11, 21, 14)$	$(20, 14)$
$(5, 8)$	$(1, 4, 8, 6)$
$(22, 12, 10)$	$(3, 0)$
$(7, 1, 3)$	$(7, 2)$
	$(11, 22)$
	$(19, 15)$
	$(21, 13, 10)$
	$(23, 12, 9, 5, 17)$

En el codi del programa de l'algorisme he seguit el raonament següent: Si l'aresta amb la que ens iniciavem era parell la agafem de la llista *eventer* i si era senar l'agafem de la llista *oddori*. Suposem que l'aresta que agafem en primer lloc és parell, aleshores és de *eventer* i per tant si el vèrtex al que arriba és de tipus I girar a l'esquerra serà anar al element d'*oddter* d'aquell mateix vèrtex i si és tipus II girar a l'esquerra serà anar al element d'*evenori*. Si hem anat a parar a l'aresta *evenori* del vèrtex, això significa que aquesta mateixa serà d'*eventer* respecte un altre vèrtex i hauriem de tornar a començar el procediment. En canvi, si al girar hem anat a parar a l'aresta d'*oddter*, aquesta mateixa serà d'*oddori* per un altre vèrtex. Si el vèrtex per el qual és d'*oddori* és tipus I girar a l'esquerra significarà anar a l'aresta d'*evenori* d'aquest vèrtex la qual serà d'*eventer* pel següent vèrtex i tornarem al inici de procés. I si el vèrtex és de tipus II, girar a l'esquerra significarà anar a parar a una aresta d'*oddter*, que pel següent vèrtex serà *oddori* i tornem a començar aquesta segona part del procés. La resta de casos es fa de manera anàloga.

Trobar els cicles dretans i esquerrans ens serà útil per a saber amb quines arestes hem de fer la maniobra Q més endavant.

4.3.3 Pas número 3. Determinar els cercles de Seifert

De manera similar a com hem trobat els cicles dretans i esquerrans, la taula ens permet trobar els cercles de Seifert. Tal i com podem veure en la Figura 34 si arribem a un encreuament mitjançant una aresta senar (parell) i seguim el que seria el cercle de Seifert, sortim del vèrtex per una aresta també senar (parell).

Per obtenir tots els cercles de Seifert busquem tots els cicles començant per una aresta parell, anant d'aresta parell en aresta parell fins arribar a la inicial, i també busquem tots els cicles començant per una aresta senar i seguint el procediment anàleg. Fixant-nos que cada aresta aparegui només un cop en el total del conjunt de cicles d'arestes que defineixen els cercles de Seifert, guardats en el diccionari anomenat *seifert_edges* . Alhora també anem enregistrant els vèrtexs pels que passem al elaborar els cicles d'arestes i els guardem en un diccionari anomenat *seifert_vertex* de manera que els vèrtexs en la posició i d'aquest diccionari corresponguin als que passem al seguir el cicle que es troba en la posició i de *seifert_edges*.

Aleshores, els cercles de Seifert determinats pel nostre exemple són:

Vèrtexs	Arestes
$cs_1 : (0, 2, 4, 3, 1)$	$s_1 : (0, 4, 8, 6, 2)$
$cs_2 : (8, 9)$	$s_2 : (16, 18)$
$cs_3 : (10, 7)$	$s_3 : (20, 14)$
$cs_4 : (11, 6, 5)$	$s_4 : (22, 12, 10)$
$cs_5 : (3, 0, 1)$	$s_5 : (7, 1, 3)$
$cs_6 : (9, 7)$	$s_6 : (19, 15)$
$cs_7 : (10, 6, 4, 2, 8, 11, 5)$	$s_7 : (21, 13, 9, 5, 17, 23, 11)$

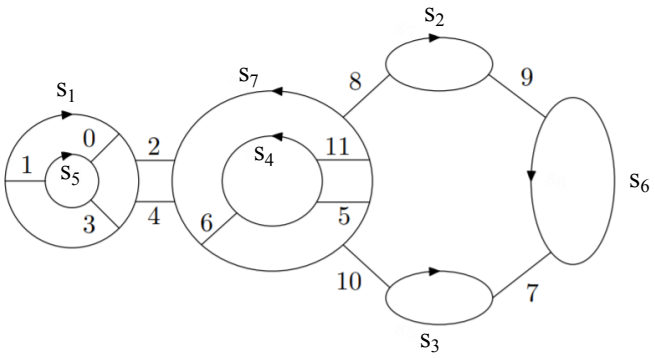


Figura 36: Diagrama de Seifert del nus de l'exemple

El conjunt de cicles de vèrtexs l'utilitzarem per construir el graf reduit de Seifert i a l'últim pas a l'hora de crear la paraula que designa la trena que buscàvem. El conjunt de cicles d'arestes l'utilitzarem per a fer una maniobra Q.

4.3.4 Pas número 4. Construir el graf reduït de Seifert

A partir del conjunt de cercles de Seifert podem determinar el graf reduït de Seifert Γ'. Notem que cadascun dels vèrtexs d'un cercle de Seifert, especificats en cs_i, apareixen exactament un cop en un altre cercle de Seifert i corresponen a una línia connectant dos cercles en el diagrama de Seifert. Si hi ha varies línies connectant dos cercles de Seifert, voldrà dir que hi ha varis arestes unint els seus corresponents vèrtexs al graf Seifert. Per tant, per a construir el graf reduït de Seifert Γ' només en considerem una d'elles, tal i com hem vist en la definició de la Secció 4.2.2.

Enregistrem Γ' com una llista de vèrtexs (els $s_1, \ldots, s_n$, sent n el nombre de cercles de Seifert) i associat a cada un, el conjunt d'arestes incidents amb aquest vèrtex. Recordem que aquestes arestes seran les línies que uneixen els cercles de Seifert que representen els vèrtexs.

A la pràctica el que farem és que com per exemple cs_1 : (0, 2, 4, 3, 1) comparteix vèrtexs amb cs_5 : (3, 0, 1) i cs_7 : (10, 6, 4, 2, 8, 11, 5), aleshores tindrem que s_1 està relacionat amb s_5 i s_7. I ho escriurem com $s_1 \sim s_5, s_7$. A més de saber quan podrem fer una maniobra Q també ens importa el tipus dels vèrtexs que comparteixen. Si els vèrtexs que comparteixen cs_i i cs_j són de tipus I, els corresponents cercles s_i i s_j tindran un signe $-$ al davant. De manera que seguint l'exemple tindríem $s_1 \sim s_5, -s_7$, ja que s_1 i s_7 comparteixen els vèrtexs 2 i 4 els quals són de tipus I com podem comprovar en la taula de la secció 4.3.1. Aplicant aquest procediment a totes les llistes de vèrtexs obtenim que:

$s_1 \sim s_5, -s_7$

$s_2 \sim -s_6, -s_7$

$s_3 \sim -s_6, -s_7$

$s_4 \sim s_7$

$s_5 \sim s_1$

$s_6 \sim -s_2, -s_3$

$s_7 \sim -s_1, -s_2, -s_3, s_4$

4.3.5 Pas número 5. Determinar si cal realitzar una maniobra Q

Una forma eficient de determinar si és necessari realitzar una maniobra Q ens ve donada pel Lema 4.6 i el Corol·lari 4.7. Aquesta serà comprovar si hi ha algun vèrtex a Γ' d'ordre més gran que dos o d'ordre dos amb ambdós arestes connectades al vèrtex del mateix tipus. O el que és el mateix, comprovar si hi ha algun s_i de la llista anterior relacionat amb més de dos cercles, o amb dos cercles que tinguin el mateix signe a davant.

En el cas de que hi hagi un s_i complint això considerem dos cercles relacionats amb ell, s_j, s_k, que tinguin el mateix signe i busquem un cicle dretà o esquerrà que contingui una aresta de cadascun dels cercles. Aquestes arestes contingudes en un mateix cicle seran amb les que realitzarem la maniobra Q i el tipus associat a elles serà el *edge type* del moviment. Aquest *edge type* ve determinat per el tipus dels vèrtexs de cs_j i cs_k que compartien amb cs_i i queda reflectit en les relacions del pas anterior.

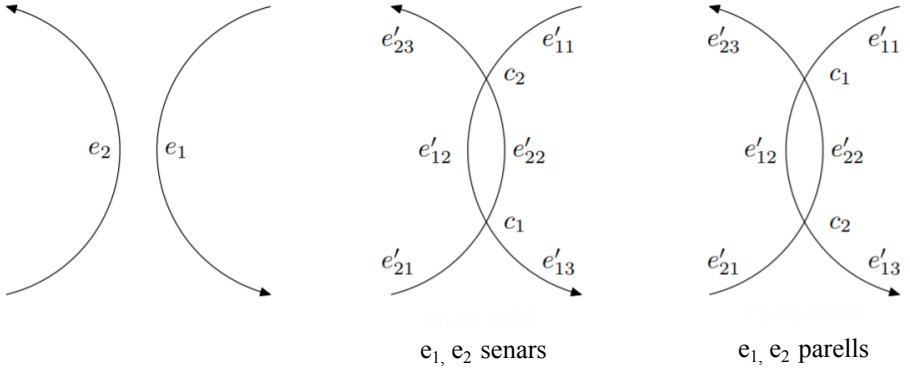
Veguem-ho en el nostre exemple a partir de les relacions especificades en el pas anterior: Com els cercles relacionats amb s_1 només són dos i tenen signe diferent no podríem aplicar una maniobra Q amb ells. En canvi, com els dos cercles relacionats amb s_2 tenen el mateix signe ens indiquen que és necessària una maniobra Q i la realitzarem amb ells. No obstant això, també podem observar que per exemple s_7 està relacionat amb més de dos cercles, per tant podríem fer la maniobra Q amb dos d'aquests que tinguessin el mateix signe com per exemple s_1 i s_2. Nosaltres, com dèiem, ho farem amb s_6 i s_7 els quals són $s_6 : (19, 15)$ i $s_7 : (21, 13, 9, 5, 17, 23, 11)$ i com 19 i 21 apareixen al cicle esquerrà $(19, 16, 23, 11, 21, 14)$ farem la maniobra Q amb aquestes dues arestes. Com podem observar a la Figura 36, hem identificat correctament uns cercles que admeten una maniobra Q.

Notem, que les arestes amb les que portem a terme la maniobra Q sempre seran les dues parells o senars. Això es deu a que els cercles de Seifert sempre es composen de o bé tot de arestes parells o bé tot d'arestes senars i els cercles de Seifert connectats són sempre de paritat oposada.

4.3.6 Pas número 6. Crear un nou *labelled peer code*

La maniobra Q que hem determinat en el pas anterior la realitzarem actualitzant la taula, que és la que conté tota la informació referent al nus amb el que estem treballant. I mitjançant aquests canvis, també obtindrem el *labelled peer code* del diagrama del nus un cop feta la maniobra Q. En primer lloc suposem que la maniobra Q s'ha realitzat amb les arestes e_1 i e_2, sent $e_1 < e_2$, i que el nus abans de realitzar-la tenia n encreuaments, o vèrtexs.

Començarem la numeració de les arestes del nou diagrama pel mateix lloc que vam començar, per tant, en el cas que $e_1 > 0$ les primeres $e_1 - 1$ arestes tindran el mateix número que abans. Les dues arestes numerades per e_1 i e_2 es convertiran cadascuna en tres noves arestes, numerades per $e'_{11}, e'_{12}, e'_{13}$ i $e'_{21}, e'_{22}, e'_{23}$ respectivament, tal i com veiem en la següent figura:



e_1, e_2 senars e_1, e_2 parells

Tindrem $e'_{11} = e_1$, $e'_{12} = e_1+1$, $e'_{13} = e_1+2$ i $e'_{21} = e_2+2$, $e'_{22} = e_2+3$, $e'_{23} = e_2+4$ com a resultat de la maniobra Q. Si $e_2 < 2n$ aleshores les arestes e_i que tinguessin assignats nombres majors que e_2 en el diagrama previ, ara tindran associat el nombre $e'_i = e_i + 4$ amb $e_i = e_2+1, \ldots, 2n$. Notem que les arestes senars continuaran sent senars i les parells continuaran sent parells.

Ara utilitzant la nova numeració de les arestes determinarem els nombres dels vèrtexs del nou diagrama de la manera següent. Si e_1 i e_2 són ambdós senars tindrem que $c_1 = e'_{12}/2$ i $c_2 = e'_{22}/2$, o el que és el mateix, $c_1 = (e_1 + 1)/2$ i $c_2 = (e_2 + 3)/2$. D'altra banda, si e_1 i e_2 són ambdós parells els dos nous vèrtexs seran $c_1 = e'_{11}/2$ i $c_2 = e'_{21}/2$, és a dir, $c_1 = e_1/2$ i $c_2 = (e_2 + 2)/2$ i en qualsevol cas sempre compliran $c_1 < c_2$. Si $c_1 > 0$, aleshores els primers $c_1 - 1$ vèrtexs mantindran el mateix nombre que tenien. Entre c_1 i c_2 el nombre assignat a cada vèrtex s'augmentarà en una unitat i els que es trobin després de c_2 augmentaran el seu nombre en dues unitats.

A més de numerar les arestes i encreuaments del diagrama també serà important saber el tipus i el signe associat als nous vèrtexs, la component en la que es troben les noves arestes així com el *peer* senar de cada vèrtex per a completar la informació del nou *labelled peer code*.

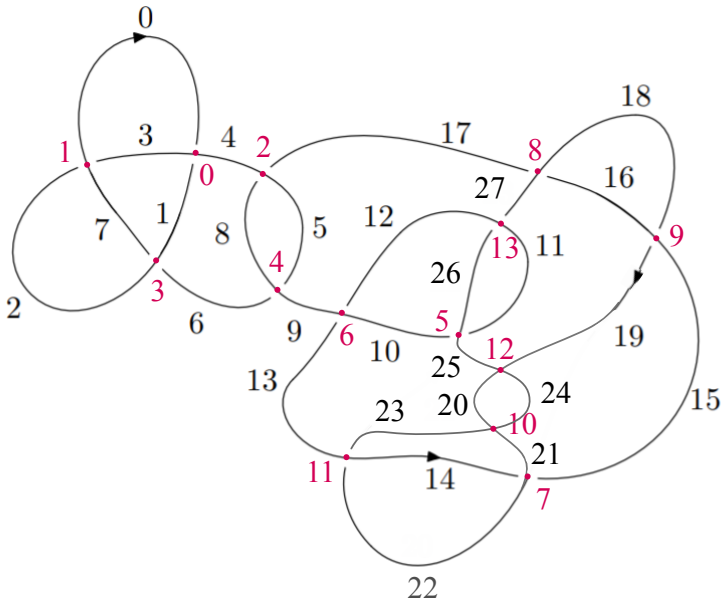
Per començar, el tipus associat als dos nous vèrtexs serà l'oposat al *edge type* que hem definit a l'apartat anterior. Les arestes $e'_{11}, e'_{12}, e'_{13}$ pertanyeran a la mateixa component que e_1 i $e'_{21}, e'_{22}, e'_{23}$ pertanyeran a la mateixa que e_2. Els signes associats als nous encreuaments seran un $+$ i un altre $-$ assignats de manera indiferent un a cadascun. I per últim destacar que si e_1, e_2 són senars, el nou *peer* senar de c_1 serà $e'_{21} = e_2 + 2$ i el de c_2 serà $e'_{11} = e_1$. I si e_1, e_2 són parells, el nou *peer* senar de c_1 serà $e'_{22} = e_2 + 3$ i el de c_2 serà $e'_{12} = e_1 + 1$.

Amb tota aquesta informació ara ja som capaços d'elaborar el nou *labelled peer code* del diagrama un cop la maniobra Q s'ha dut a terme, com podrem veure seguint amb el nostre exemple: Tenim que $e_1 = 19$ i $e_2 = 21$, aleshores com 19 i 21 són senars $c_1 = (e_1 + 1)/2 = 20/2 = 10$ i $c_2 = (e_2 + 3)/2 = 24/2 = 12$ i els nous *peers* senars de c_1 i c_2 seran $e_2 + 2 = 23$ i $e_1 = 19$ respectivament. Com el *edge type* de 19 i 21 era tipus I, ja que teníem $s_2 \sim -s_6, -s_7$ aleshores el tipus dels dos nous *peers* senars, o el que és el mateix, el tipus associat als dos nous vèrtexs, serà tipus II. Les components de $e'_{11}, e'_{12}, e'_{13} = 19, 20, 21$ seran la mateixa que la de 19 i les de $e'_{21}, e'_{22}, e'_{23} = 23, 24, 25$ seran la mateixa que la de 21. El signe associat a $c_1 = 10$ i $c_2 = 12$ seran un $+$ i un $-$ indiferentment. Tenint en compte que les arestes amb nombres inferiors a 19 conservaran el mateix número associat, que les que es trobin entre 19 i 21 hauran d'augmentar el seu nombre en dues unitats i que les posteriors a 21 l'hauran d'augmentar en 4, a més de tota

la informació anterior podem escriure el nou *labelled peer code*. En aquest cas serà:

$$3\ 7\ {-17}\ 1,\ {-5}\ 25\ 9\ {-21}\ {-27},\ {-15}\ 23\ {-13}\ 19\ 11\ /\ --+-+-----+--$$

I com podem comprovar amb el dibuix del diagrama del nus un cop feta la maniobra Q, aquest *labelled peer code* és l'adequat:



4.3.7 Pas número 7. Iterar el procés

Un cop obtingut el nou *labelled peer code* hem de repetir el procés identificant els cicles dretans i esquerrans, els cercles de Seifert i el graf reduït de Seifert. Ja que l'efecte de la maniobra Q és reduir el nombre de cercles de Seifert incoherents i el procés acaba després d'un nombre finit de passos quan tots els cercles estan coherentment orientats, com succeïa a l'algorisme de Yamada–Vogel.

En aquest cas, podrem identificar quan el procés acaba a partir del ordre dels vèrtexs del graf reduït de Seifert Γ', o el que és el mateix, a partir de les relacions dels cercles de Seifert. Si tots els cercles de Seifert estan relacionats exactament amb altres dos, excepte dos cercles de Seifert que només estan relacionats amb un altre tindrem un conjunt concèntric de cercles amb la mateixa orientació, és a dir tots coherents, on els cercles només relacionats amb un altre serien el més interior i el més exterior. O bé tindrem dos conjunts de cercles concèntrics on cada cercle del conjunt té la mateixa orientació, però la orientació dels dos conjunts és oposada. Per tant també tots els cercles són coherents, i en aquest cas els dos cercles que només estan relacionats amb un altre serien el més interior de cadascun dels conjunts.

En el nostre exemple, el procés acaba amb les següents relacions:

$$s_1 \sim s_4, -s_6$$

$$s_2 \sim s_6, -s_7$$

$s_3 \sim -s_5, s_7$

$s_4 \sim s_1$

$s_5 \sim -s_3$

$s_6 \sim -s_1, s_2$

$s_7 \sim -s_2, s_3$

Tinguem en compte, que els s_i no són els mateixos que abans, aquests han estat modificats després de la realització de varies maniobres Q.

4.3.8 Pas número 8. Llegir la paraula que designa la trena

Com ja vam veure a la Seccions 3.1.1 i 3.1.2, sempre podem llegir la paraula que designa la trena de la qual cert nus n'és clausura a partir d'un conjunt de cercles de Seifert coherents concèntrics. Per tant, en el cas que haguem arribat a dos conjunts de cercles de Seifert concèntrics coherents amb orientacions oposades, realitzant canvis a l'infinit adequats arribarem a un sol conjunt de cercles amb les condicions que ens interessaven. A la pràctica, no serà necessari fer aquests canvis a l'infinit explícitament, perquè al obtenir la paraula que designa la trena ja els realitzarem de manera indirecta.

En primer lloc, designarem com a cercle número 1 al més intern del conjunt, o si tenim dos conjunts, qualsevol dels dos cercles més interns. Aquests, en particular són els que estan relacionats amb només un altre cercle, i el que escollim serà el que representarà la primera corda de la trena. Les cordes successives vindran representades per els cercles relacionats successivament. És a dir, el cercle més interior correspondrà a la primera corda, el únic que està relacionat amb ell serà el cercle número 2 i correspondrà a la segona, el que està relacionat amb aquest i no es el primer a la tercera i així successivament. En el nostre exemple, si recordem la llista de relacions final vista a la secció anterior, podríem triar com a cercle número 1 tant a s_4 com a s_5. Suposem que triem s_4, aleshores el cercle número 2 seria s_1, el número 3 s_6, el 4 s_2, el 5 s_7, el 6 s_3 i el 7 s_5, observem que és clar que l'ultim sempre serà l'altre cercle que només està relacionat amb un cercle de Seifert i que no és el cercle número 1.

Un cop numerats els cercles, per a construir la paraula que designa la trena a partir d'aquests substituirem les línies que uneixen cercles de Seifert per trenes elementals, recordem que aquestes línies són els vèrtexs en el graf reduït de Seifert. Si una línia uneix els cercles i i $i+1$ serà representada per la trena elemental $\sigma_i^{\pm 1}$ i seran els signes associats als vèrtexs que designen aquestes línies, juntament amb els seu respectius tipus, els que ens diguin si l'exponent haurà de ser $+1$ o -1. Si el vèrtex de nombre el de la línia que uneix i i $i+1$ té associat signe $+$ i és de tipus I la trena elemental serà σ_i^{+1}, i si és de tipus II la trena elemental serà σ_i^{-1}. De forma anàloga, si té associat el signe $-$ i és de tipus I la trena elemental serà σ_i^{-1}, i si és de tipus II la trena elemental serà σ_i^{+1}.

Al numerar els cercles ja hem fet de manera indirecta els canvis a l'infinit que necessitàvem. Ara podem considerar que tenim un conjunt de cercles concèntrics orientats en la mateixa direcció sent 1 el cercle més intern i n (on n és el nombre total de cercles de Seifert) el més extern, sent ambdós els únics cercles de Seifert connectats amb només un altre cercle. Començarem a construir la paraula que designa la trena començant al centre del conjunt i movent-nos cap a fora. Parlarem de cercle de Seifert *child* i del seu *parent*

quan considerem dos cercles de Seifert amb vèrtexs adjacents a Γ', és a dir, dos cercles de Seifert relacionats un dins l'altre. El *parent* serà el cercle de Seifert que haguem numerat amb un número més gran, és a dir, el que es trobi més exterior.

Considerarem els cercles de Seifert *parent* i *child* com p i s respectivament, i els escriurem com la llista de vèrtexs pels que passen. Per tant si es corresponen als cercles de Seifert s_i i s_j, els designarem per els cicles de vèrtexs cs_i i cs_j de la Secció 4.3.3.

En el cas inicial s serà un cercle relacionat amb només el seu respectiu *parent* p. Suposem que c_1, c_2, c_3 són vèrtexs consecutius de p i que c_2 també apareix a s. Aleshores substituirem c_2 a p per la seva trena elemental equivalent, que al ser s el cercle número 1, tindrà forma $\sigma_1^{\pm 1}$ i li direm de manera genèrica $w(c_2, p)$. Per tant al fer aquesta substitució p esdevindrà:

$$p = (\ldots, c_1, w(c_2, p), c_3, \ldots).$$

Fer aquest procés amb tots els vèrtexs que comparteixen s i p se'n diu introduir s en el seu *parent*.

Pot ser que un cop introduït s en el seu *parent* ens trobem que hi ha vèrtexs consecutius a p que estaven a s, quedant-nos p de la següent forma:

$$p = (\ldots, c_k, w(c_{k+1}, p), \ldots, w(c_{k+n}, p), c_{k+n+1}, \ldots),$$

on c_k i c_{k+n+1} no apareixen a s.

Si ens trobem en aquest cas, concatenem $w(c_{k+1}, p), \ldots, w(c_{k+n}, p)$ i obtenim la nova paraula $w*(c_k, p)$ formada per la concatenació de trenes elementals. Així doncs p quedaria com:

$$p = (\ldots, c_k, w * (c_k, p), c_{k+n+1}, \ldots).$$

Ara suposem que p té un *parent* p', i suposem que el c_1 d'abans és també un vèrtex de p', de manera que $p' = (\ldots, c_a, c_1, c_b, \ldots)$. Degut a que p té el seu propi *child*, s, substituïm c_1 a p' amb la seva trena elemental associada $w(c_1, p')$ seguit de $w * (c_1, p)$. Fent aquest procés amb tots els vèrtex que comparteixen p i p', és a dir, introduint p a p' i concatenant totes les $w(c_i, p')$ i $w * (c_j, p)$ que es trobin en posicions successives tindrem enregistrat a p' les trenes elementals corresponents a les cordes s, p i p'.

Per a obtenir la paraula completa que designa la trena, continuem fent aquest procés de manera inductiva i introduint els cercles de Seifert en els seus respectius *parents*. Quan arribem a l'altre cercle de Seifert que només està relacionat amb un i que no és el cercle pel que hem començat haurem acabat.

5 Conclusions

Hem descrit l'estructura i analitzat algunes de les propietats d'un grup que té menys d'un segle d'estudi al darrere, el grup de trenes. I complementant aquesta informació amb algunes nocions bàsiques de teoria de nusos hem pogut demostrar de diverses formes un teorema que relaciona aquests dos camps, el Teorema d'Alexander. Tot això acompanyat de nombroses figures fetes personalment, amb excepció d'algunes de l'última secció, que faciliten la comprensió dels conceptes.

A nivell personal una de les coses més sorprenents i satisfactòries ha estat el partir de conceptes teòrics topològics i poder relacionar-ho amb la computació algorísmica. Al començar aquest treball tenia una intenció completament teòrica i ha estat gratificant comprovar com les eines tecnològiques i el món de la programació ens pot ajudar a simplificar troballes inclòs en l'àmbit de la topologia.

El repte assolit en aquest treball ha estat aprendre el llenguatge Python de zero per a poder utilitzar-lo en la programació d'un algorisme, també de zero, a partir d'indicacions sense saber si els resultats correspondrien amb els esperats. Ha estat molt gratificant arribar a obtenir el resultat buscat, però com a continuació del treball proposaria continuar l'algorisme de manera que sigui més eficient i que com a resultat s'obtingués la trena més simplificada possible. Seria interessant perquè és molt complicat esbrinar si dues trenes són les mateixes, tot i que els moviments de Markov donen les pautes per a arribar a veure si dues trenes tenen com a clausura el mateix nus o enllaç encara ens trobem davant d'un problema a l'hora de portar-ho a la pràctica.

A Codi en Python de l'algorisme de Choi

En aquest annex trobarem el codi escrit en Python de l'algorisme que hem explicat a la Secció 4. En ell podem distingir els mateixos passos anteriorment descrits, els quals són executats a la part final. A continuació del codi trobem inclòs un exemple de l'output que rebríem al introduir com a *labelled peer code* inicial el que utilitzem per explicar els passos de l'algorisme a la secció nomenada juntament amb les explicacions de com interpretar-lo.

Cal destacar que tot i que aquest algorisme ens proporciona una trena que té un cert nus com a clausura, aquesta no és la mínima. És a dir, pot existir una trena amb menys cordes i menys encreuaments que tingui com a clausura el mateix nus i aquesta serà equivalent a la trena trobada. Com arribar a veure si dues trenes són realment equivalents és un problema complicat, anomenat *the word problem*, en el que s'usen les relacions d'Artin introduïdes en la Secció 2.1.1 per a arribar d'una paraula a l'altra. A més, si volem que aquestes trenes tinguin com a clausura el mateix nus o enllaç, com és el cas, també podem utilitzar uns moviments no explicats en aquest treball, anomenats *moviments de Markov*.

També notar, que en el codi de l'algorisme s'usa molt la paraula *Vogel* i això es deu a que la maniobra Q definida en la Secció 3.1.2 també pot ser anomenada *Vogel move*.

```
# -*- coding: utf-8 -*-
"""Algorisme TFG.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/13M54p-
    CKI4PC1jHzv1EqjdRI28ZR36YC
"""

#Pas número 1. Crear la taula

def taula(x):
  n = len(x)
  oddter=[]      #oddter = arestes senars que arriben
  oddori=[]      #oddori = arestes senars que surten
  eventer=[]     #eventer = arestes parells que arriben
  evenori=[]     #evenori = arestes parells que surten
  tipus=[]
  component=[]

  for i in range (0,n-1):
    oddter = oddter + x[i]

  for e in oddter:
    if e<0:
      tipus.append(False)       #True=tipus2, False=tipus1
    else:
      tipus.append(True)

  k=0
  for i in range (0,n-1):       #el nombre de components serà n-1=len(x)-1
    for j in range (0,len(x[i])):
      component.append(k)
    k+=1

  #afegim 0 si es de la primera component, 1 si es de la segona etc i
```

```
    sabem de quina component es per la llargaria dels x[i] doncs cada x[i]
    és una component
37
38
39  for i in range(len(oddter)):
40    oddter[i] = abs(oddter[i])       #agafem el oddter que teniem i li fem
     valor absolut pq els negatius només indicaven el tipus
41
42  label=x[n-1]                       #la llista label és exactament la ú
    ltima component del peer code (+=True=Per sobre, -=False=Per sota)
43
44  evenori=oddter[:]
45
46  k=0
47  for i in range(n-1):
48    for j in range(2*len(x[i])-1):
49      if(j%2==1):                     # treballem només amb els senars
50        indice=evenori.index(k+j)
51        evenori[indice]+=1            #ó =j+1
52    indice=evenori.index(2*len(x[i])+k-1)
53    evenori[indice]=k
54    k+=j+2
55
56  k=0
57  for i in range(len(oddter)):
58    eventer.append(k)
59    k+=2                              #la llista de les arestes que
    arriben parells es fa anant de 2 en 2
60
61  oddori=[]
62  for i in range(len(oddter)):
63    oddori.append(eventer[i]+1)
64
65  return component, evenori, eventer, oddori, oddter, tipus, label
66
67
68
69
70 #Pas número 2. Determinar els cicles dretans i esquerrans
71
72 def left_cycles(evenori, eventer, oddori, oddter, tipus):
73   lefteven=[]
74   leftodd = []
75   leftcycles=[]
76
77   for k in eventer:
78    if k not in lefteven:
79     i = eventer.index(k)
80     lefteven = []
81     lefteven.append(eventer[i])
82     j=0
83
84     #primer iterat
85     if( (lefteven[j]%2==0 and tipus[i] == True) or (lefteven[j]%2 != 0 and
      tipus[i]==False) ):
86         lefteven.append(evenori[i])
87         i = eventer.index(evenori[i])
88         j+=1
89     else:
90         lefteven.append(oddter[i])
91         i = oddori.index(oddter[i])
```

```
92          j+=1
93
94      #bucle fins que es repeteix
95      while(lefteven[j] != lefteven[0]):
96        if( (lefteven[j]%2==0 and tipus[i] == True) or (lefteven[j]%2 != 0
    and tipus[i]==False) ):
97          lefteven.append(evenori[i])
98          i = eventer.index(evenori[i])
99          j+=1
100       else:
101         lefteven.append(oddter[i])
102         i = oddori.index(oddter[i])
103         j+=1
104
105     lefteven.pop(-1) #treiem l'últim que es repeteix
106     leftcycles.append(lefteven)
107
108     new = []   #creem una llista nova per eliminar els cicles que es
    repeteixen
109     for i in range(len(leftcycles)):
110      new.append(sorted(leftcycles[i]))
111
112     for i in range(int(len(new))):
113       for j in range(len(new)):
114         if new[i] == new[j] and j>i:
115           leftcycles.pop(i);
116
117
118 for k in oddori:
119  if k not in leftodd:
120   i = oddori.index(k)
121   leftodd = []
122   leftodd.append(oddori[i])
123   j=0
124
125   #primer iterat
126   if( (leftodd[j]%2==0 and tipus[i] == True) or (leftodd[j]%2 != 0 and
    tipus[i]==False) ):
127     leftodd.append(evenori[i])
128     i = eventer.index(evenori[i])
129     j+=1
130   else:
131     leftodd.append(oddter[i])
132     i = oddori.index(oddter[i])
133     j+=1
134
135   #bucle fins que es repeteix
136   while(leftodd[j] != leftodd[0]):
137     if( (leftodd[j]%2==0 and tipus[i] == True) or (leftodd[j]%2 != 0 and
     tipus[i]==False) ):
138         leftodd.append(evenori[i])
139         i = eventer.index(evenori[i])
140         j+=1
141       else:
142         leftodd.append(oddter[i])
143         i = oddori.index(oddter[i])
144         j+=1
145
146   leftodd.pop(-1) #treiem l'últim que es repeteix
147   leftcycles.append(leftodd)
148
```

```
    new = []   #creem una llista nova per eliminar els cicles que es repeteixen
    for i in range(len(leftcycles)):
     new.append(sorted(leftcycles[i]))

    for i in range(int(len(new))):
      for j in range(len(new)):
        if new[i] == new[j] and j>i:
          leftcycles.pop(i);

  print(leftcycles)
  return leftcycles




#Pas número 3. Determinar els cercles de Seifert

def seifert_circles (evenori, eventer, oddori, oddter, tipus):

  vertex = []
  edges =[]
  seifert_vertex = []
  seifert_edges = []
  j=0

  for k in eventer:
    if k not in edges:
      i=eventer.index(k)
      edges = []
      vertex = []
      edges.append(k)
      vertex.append(i)
      edges.append(evenori[i])
      j=1

      while(edges[j] != edges[0]):
        i=eventer.index(edges[j])
        edges.append(evenori[i])
        vertex.append(i)
        j+=1
      edges.pop(-1) #treiem l'últim que es repeteix

      seifert_vertex.append(vertex)
      seifert_edges.append(edges)


      new = []   #creem una llista nova per eliminar els cicles que es repeteixen
      for i in range(len(seifert_vertex)):
        new.append(sorted(seifert_vertex[i]))

      for i in range(int(len(new))):
        for j in range(len(new)):
          if new[i] == new[j] and j>i:
            seifert_vertex.pop(i)
            seifert_edges.pop(i)



  vertex = []
```

```
  edges =[]
  j=0

  for k in oddori:
    if k not in edges:
      i=oddori.index(k)
      edges = []
      vertex = []
      edges.append(k)
      vertex.append(i)
      edges.append(oddter[i])
      j=1

      while(edges[j] != edges[0]):
        i=oddori.index(edges[j])
        edges.append(oddter[i])
        vertex.append(i)
        j+=1
      edges.pop(-1) #treiem l'últim que es repeteix

      seifert_vertex.append(vertex)
      seifert_edges.append(edges)

      new = []   #creem una llista nova per eliminar els cicles que es
    repeteixen
      for i in range(len(seifert_edges)):
        new.append(sorted(seifert_edges[i]))

      for i in range(int(len(new))):
        for j in range(len(new)):
          if new[i] == new[j] and j>i:
            seifert_vertex.pop(i)
            seifert_edges.pop(i)

  print("Vèrtexs:")
  for i in range(len(seifert_vertex)):
    print("cs", i+1, " = ", seifert_vertex[i])

  print("\n")

  print("Arestes:")
  for i in range(len(seifert_edges)):
    print("s", i+1, " = ", seifert_edges[i])

  print("\n")

  return seifert_vertex, seifert_edges




#Pas número 4. Construir el graf reduit de Seifert

def reduced_graph (seifert_vertex, seifert_edges, tipus):

  related_dic = {}        #a related_dic, com a "key" tindrem la i del s_i
     al que ens referim que esta relacionat amb els cicles d'arestes del
    seu corresponent "value"
  orientacions_dic = {}   # a orientacions_dic, per cada cicle d'arestes
    del "value" de related_dic tindrem un [0] si els vèrtexs que
    comparteixen s_i i el cicle són de tipus 1 i un [1] si són de tipus 2
```

```
264
265   j=1
266   for i in seifert_edges:
267     related_dic.update({j:[]})
268     j+=1
269
270   j=1
271   for i in seifert_edges:
272     orientacions_dic.update({j:[]})
273     j+=1
274
275
276   #mirem quins seifert_edges comparteixen seifert_vertex, pq aleshores
        voldrà dir que estan relacionats:
277   for k in range(len(seifert_vertex)):
278     for i in range(len(seifert_vertex)):
279       for j in range(len(seifert_vertex[i])):
280         if (seifert_vertex[i][j] in seifert_vertex[k]) and (k!=i) and (
        seifert_edges[i] not in related_dic[k+1]):
281
282           if (tipus[seifert_vertex[i][j]]==False) and (seifert_edges[i]
        not in related_dic[k+1]):
283             orientacions_dic[k+1].append([0])   #mirem el tipus del
        vertex seifert_vertex[i][j] mirant la posició seifert_vertex[i][j] del
        vector tipus, si es tipus1 (False) afegim un 0, que seria un -
284
285           else:                #sino, és tipus2 i fiquem un 1 que seria
        com un +
286             if seifert_edges[i] not in related_dic[k+1]:
287               orientacions_dic[k+1].append([1])
288
289           related_dic[k+1].append(seifert_edges[i])
290
291   print("Relacions Cercles Seifert:")
292
293   for i in range(len(related_dic)):
294     print("s", i+1, "~ ", related_dic[i+1], "amb signes", orientacions_dic
        [i+1])
295
296   print("On 0 indica - i 1 indica +\n")
297   return related_dic, orientacions_dic
298
299
300
301
302 #Pas número 5. Determinar si cal realitzar una maniobra Q
303
304 def vogel (related_dic, orientacions_dic, leftcycles, rightcycles):
305
306   orien = []
307   orivogel = 0
308   relacions = []
309   vogel = []
310   vogelcycle = []
311   triar = []
312   newtriar = []
313   e1 =0
314   e2=0
315   stop = "NO"    #ho utilitzarem per saber quan parar quan iterem tot el
        procés complet
316
```

```
317
318   #transformem els diccionaris en llistes de llistes per a poder treballar
        millor amb ells:
319   for i in range(1,len(related_dic)+1):
320     orien.append(orientacions_dic[i])
321     relacions.append(related_dic[i])
322
323   for i in range(len(relacions)):
324     if len(relacions[i])>2:
325       for j in range(len(relacions[i])):
326         for k in range(len(relacions[i])):
327           if (orien[i][j] == orien[i][k]) and (relacions[i][j] !=
      relacions[i][k]):
328             print("Podem fer una maniobra Q amb", relacions[i][j],
      relacions[i][k])
329             vogel = []
330             vogel.append(relacions[i][j])
331             vogel.append(relacions[i][k])
332             orivogel = orien[i][j]    #en aquesta variable guardarem el
      tipus associat a les arestes amb les que fem la maniobra Q, necessari
      pel següent pas
333             break
334
335     elif (len(relacions[i])==2) and (orien[i][0]==orien[i][1]):
336       print("Podem fer una maniobra Q amb", relacions[i])
337       vogel = []
338       vogel = list(relacions[i])
339       orivogel = orien[i][0]
340       break
341
342   if vogel != []:
343     print("Però finalment, farem una maniobra Q amb:", vogel)
344   else:
345     print("No cal fer cap maniobra Q")
346
347
348   for i in range(len(relacions)):
349     triar.append(len(relacions[i]))         #en aquesta llista "triar"
      guardem el grau dels vèrtexs del reduced graf de seifert per a saber
      quan parar, ja que pararem d'iterar quan exactament dos vèrtexs tinguin
       ordre 1 i els altres ordre 2. Perquè voldrà dir que no es necessita
      cap maniobra Q més
350
351   for i in range(len(triar)):
352     if(triar[i] != 2):
353       newtriar.append(triar[i])             #afegim a "newtriar" tots es
      ordres diferents de 2, ja que si estem al pas final en el que hem de
      parar, només hi haurà dos ordres diferents de 2, els d'ordre 1
354
355   if(len(newtriar) == 2) and (newtriar[0] == newtriar [1] == 1):
356     stop = "SI"
357     return e1, e2, orivogel, stop, triar    #si arribem a les condicions
      en que no es necessita cap altre maniobra Q, parem i retornem el
      necessari. Sino seguim el procés trobant els "e1", "e2", etc.
358
359
360   for i in range(len(vogel[0])):
361       for j in range(len(vogel[1])):
362         for k in range(len(leftcycles)):
363           if (vogel[0][i] in leftcycles[k]) and (vogel[1][j] in leftcycles
      [k]):
```

```
364             print("Com e1 = ", vogel[0][i], "i e2 = ", vogel[1][j], "
    apareixen al cicle esquerrà", leftcycles[k],"i pertanyen a", vogel ,"
    respectivament, farem la maniobra Q amb ells")
365             vogelcycle = leftcycles[k]
366             if vogel[0][i] < vogel[1][j]:
367               e1 = vogel[0][i]
368               e2 = vogel[1][j]
369             else:
370               e2 = vogel[0][i]
371               e1 = vogel[1][j]
372           if(e1 != 0) and (e2!=0):
373             break
374
375   for i in range(len(vogel[0])):
376     for j in range(len(vogel[1])):
377       for k in range(len(rightcycles)):
378         if (vogel[0][i] in rightcycles[k]) and (vogel[1][j] in rightcycles
    [k]):
379           print("Com", vogel[0][i], "i", vogel[1][j], "apareixen al cicle
    dretà", rightcycles[k], "i pertanyen a", vogel, " respectivament, farem
     la maniobra Q amb ells")
380           vogelcycle = rightcycles[k]
381           if vogel[0][i] < vogel[1][j]:
382             e1 = vogel[0][i]
383             e2 = vogel[1][j]
384           else:
385             e2 = vogel[0][i]
386             e1 = vogel[1][j]
387         if(e1 != 0) and (e2!=0):
388             break
389
390   return e1, e2, orivogel, stop, triar
391
392
393 #Pas número 6. Crear un nou labelled peer code
394
395 #el nostre OBJECTIU és donar un nou PEER CODE que representi el nus que
    teniem amb la maniobra Q duta a terme
396
397 def new_code (e1, e2, oddter, tipus, label ,component, orivogel, oddori,
    eventer):
398
399   #només cal mirar un dels dos e_i, pq sempre seran o bé els dos parells o
     bé els dos senars. c1, c2 nous vèrtexs del nus
400
401   if(e1%2 == 0):
402     c1 = e1/2
403     c2 = (e2 + 2)/2
404   else:
405     c1 = (e1 +1)/2
406     c2 = (e2 + 3)/2
407
408   print("Els nous vèrtexs produits per la maniobra Q seran c1 = ", c1, "c2
     = ", c2)
409
410   newpeer = []
411   newlabel = []
412   newcomponent = []
413
414   for i in range(len(oddter)+2):
415     newpeer.append(0)
```

```
    newlabel.append(0)
    newcomponent.append(0)    #omplim les llistes de tants 0s com la seva mida serà, per a després poder omplir una posició i concreta

  if e1%2 == 0:               #omplim on van els nous encreuaments c1 i c2
    newpeer[int(c1)] = e2+3
  else:
    newpeer[int(c1)] = e2+2

  if e1%2 == 0:
    newpeer[int(c2)] = e1+1
  else:
    newpeer[int(c2)] = e1

  #omplim la resta de llocs del peer amb els elements del peer anterior:
  for i in range(len(oddter)):
    if i < c1:
      newpeer[i] = oddter[i]
    elif i >= c1 and i < c2-1:
      newpeer[i+1] = oddter[i]
    elif i >= c2-1:
      newpeer[i+2] = oddter[i]

  for i in range(len(newpeer)):    #fem la modificació que calgui en la numeració dels peers senars degut a la incorporació de c1 i c2 al nus
    if i !=c1 and i!=c2:
      if newpeer[i] >= e1 and newpeer[i] < e2:
        newpeer[i] += 2
      elif newpeer[i] >= e2:
        newpeer[i] += 4

#explicitem el tipus i la label amb el vector newlabel, recordem que True = + = Per sobre, False = - = per sota, també omplim newcomponent amb les components del peer code original
  for i in range(len(tipus)):
    if i<c1:
      newlabel[i] = label[i]
      newcomponent[i] = component[i]
      if(tipus[i] == False):
        newpeer[i] = -newpeer[i]  #si era tipus 1 ho indiquem amb un negatiu al peer code

    elif i>=c1 and i<c2-1:
      newlabel[i+1] = label[i]
      newcomponent[i+1] = component[i]
      if(tipus[i] == False):
        newpeer[i+1] = -newpeer[i+1]

    elif i>=c2-1:
      newlabel[i+2]=label[i]
      newcomponent[i+2] = component[i]
      if(tipus[i] == False):
        newpeer[i+2] = -newpeer[i+2]

  #el tipus dels nous crossings c1 i c2 seran el contrari del tipus de e1 i e2, si eren de tipus 2 ara seran de tipus 1, i ho indiquem amb un negatiu
  if(orivogel == [1]):
    newpeer[int(c1)] = -newpeer[int(c1)]
    newpeer[int(c2)] = -newpeer[int(c2)]

```

```
  #pels nous crossings una label ha de ser positiva i l'altre negativa, però l'ordre és indiferent
  newlabel[int(c1)] = True
  newlabel[int(c2)] = False

  #la component associada als new crossings serà la de e1 per c1 i la de e2 per c2

  if e1%2==0:
    newcomponent[int(c1)] = component[eventer.index(e1)]
    newcomponent[int(c2)] = component[eventer.index(e2)]

  else:
    newcomponent[int(c1)] = component[oddori.index(e1)]
    newcomponent[int(c2)] = component[oddori.index(e2)]

  #per últim queda separar el newpeer en components i afegir newlabel per a tenir el nou peer code acabat

  x1 = []
  maxcomp = newcomponent[-1]  #així trobem el nombre de components -1 que hi ha, ja que comencem a numerar les components amb 0

  for i in range(maxcomp+1):
    x1.append([])             #afegim tantes llistes com components tindrà el peer

  for i in range(maxcomp+1):
    for j in range(len(newpeer)):
      if newcomponent[j] == i:
        x1[i].append(newpeer[j])    #a cada nova llista afegim els elements que tinguessin designada la mateixa component a a llista newcomponent

  x1.append(newlabel)

  print("Nou peer code:", x1, "\n")

  return x1




#Pas número 7. Iterar el procés

def iteratiu (x):
  component, evenori, eventer, oddori, oddter, tipus, label = taula(x)

  print("Cicles esquerrans:")
  leftcycles = left_cycles(evenori, eventer, oddori, oddter, tipus)
  tipus2 = []
  x1 = []

  for i in tipus:
    if i == True:
      tipus2.append(False)
    elif i == False:
      tipus2.append(True)

  print("Cicles dretans:")
  rightcycles = left_cycles(evenori, eventer, oddori, oddter, tipus2)
```

```
524   print("\n")
525
526   seifert_vertex, seifert_edges = seifert_circles (evenori, eventer,
        oddori, oddter, tipus)
527
528   related_dic, orientacions_dic = reduced_graph(seifert_vertex,
        seifert_edges, tipus)
529
530   e1, e2, orivogel, stop, triar = vogel(related_dic, orientacions_dic,
        leftcycles, rightcycles)
531
532   if(stop == "SI"):    #si es retorna el "SI" que ens indica que no calen
        més maniobres Q, aturem el procés d'iteració
533     return x1, stop, triar, seifert_vertex, eventer, tipus, oddori, label
534
535   x1 = new_code (e1, e2, oddter, tipus, label, component, orivogel, oddori
        , eventer)
536
537   return x1, stop, triar, seifert_vertex, eventer, tipus, oddori, label
538
539
540 #Pas número 8. Llegir la paraula que designa la trena
541
542 #hem de començar amb un cercle que només estigui relacionat amb un altre,
      no amb 2 (mirem la posició de qualsevol qe tingui un 1 a la llista "
      triar") i després per anar triant els cercles successius sera el que
      comparteixi vèrtexs amb l'actual
543 def braid_word(triar, seifert_vertex, eventer, tipus, oddori, label):
544
545   s=[]  #cercle inicial
546   s1=[] #cercle final
547   p=[]
548   p_=[]
549   braid = []
550   k=1
551
552   for i in range(len(triar)):
553     if triar[i] == 1:
554       s = seifert_vertex[i]
555       break
556   print("El cercle de Seifert més interior (inicial) té vèrtexs:", s)  #s
        serà el inicial que només està relacionat amb un cercle, i s1 serà el
        cercle final que també només està relacionat amb un
557
558   for i in range(len(triar)):
559     if triar[i] == 1:
560       if seifert_vertex[i] != s:
561         s1 = list(seifert_vertex[i])
562         break
563   print("El cercle de Seifert més exterior (final) té vèrtexs:", s1, "\n")
564
565
566   #p_ en el primer cas esta buit, i després guardem en ell en cada iterat
        el cercle que tenim com a "parent"
567
568   while p_ != s1:   #ho farem fins que arribem al cercle final s1
569
570     for i in range(len(seifert_vertex)):
571       for j in range(len(s)):
572         if (s[j] in seifert_vertex[i]) and (s != seifert_vertex[i]):   #si
        un element de s està en un altre conjunt de vèrtexs que no sigui ell
```

```
    mateix, aquest conjunt de vèrtexs serà el qe forma el cercle de seifert
     relacionat amb s. Els vèrtexs que tenen en comú son els que estan
    representats com les línies que els uneixen
573         p = list(seifert_vertex[i])
574         break                       # agafem com a p el "parent" de s
575 
576     seifert_vertex.remove(p)
577 
578     p_ = p.copy()
579     print("parent p:", p_)
580 
581     for i in range(len(s)):
582       for j in range(len(p)):
583         if s[i] == p[j]:
584           p[j] = ("s", k)            #substituïm el vèrtex que està en s i
     en p per la seva paraula de trena equivalent on ('s', k) = \sigma_k i
    afegim el signe corresponent, ('-','s', k) = \sigma_k^{-1} i ('+','s',
    k) = \sigma_k^{+1}
585 
586         if (s[i] in eventer):
587           if (label[eventer.index(s[i])] == False) and (tipus[s[i]] ==
    False):
588             p[j] = ("-", "s", k)
589           elif (label[eventer.index(s[i])] == False) and (tipus[s[i]] ==
     True):
590             p[j] = ("+", "s", k)
591           elif (label[eventer.index(s[i])] == True) and (tipus[s[i]] ==
    False):
592             p[j] = ("+", "s", k)
593           elif (label[eventer.index(s[i])] == True) and (tipus[s[i]] ==
    True):
594             p[j] = ("-", "s", k)
595 
596         if (s[i] in oddori):
597           if (label[oddori.index(s[i])] == False) and (tipus[s[i]] ==
    False):
598             p[j] = ("-", "s", k)
599           elif (label[oddori.index(s[i])] == False) and (tipus[s[i]] ==
    True):
600             p[j] = ("+", "s", k)
601           elif (label[oddori.index(s[i])] == True) and (tipus[s[i]] ==
    False):
602             p[j] = ("+", "s", k)
603           elif (label[oddori.index(s[i])] == True) and (tipus[s[i]] ==
    True):
604             p[j] = ("-", "s", k)
605 
606         if i<len(s)-1:
607           if isinstance(s[i+1], tuple):
608             p[j] += s[i+1]           #també hem de tenir en compte que les
     "paraules trena" que tenia el cercle s
609 
610     print("p després de introduir-hi s =", p)
611     k+=1
612 
613     if p_ == s1:
614       for i in range(len(p)):
615         braid += p[i]
616       return braid          #p_ ara es una copia de p (el parent) sense
    les modificacions afegides. Per tant diu que si hem arribat al ultim
    cercle, i lhem modificat, tornem ja el cercle modificat que sera la
```

```
    trena
617
618
619     while(isinstance(p[0], tuple)):
620       p.append(p[0])
621       p.remove(p[0])        #si el parent modificat té com a inicial una
    tupla, la fiquem al final, sabem que la trena seguirà sent la mateixa
    perquè aquest és un dels moviments de Markov
622
623     print("p ficant paraula trena inicial al final =", p)    #ens anirà
    millor a l'hora d'agafar les paraules trena de l'iterat anterior per
    afegir-les a l'actual
624
625
626     for i in range(len(p)-1):
627       if isinstance(p[i], tuple) and isinstance(p[i+1], tuple):
628         p[i+1] = p[i]+p[i+1]        #concatenem les paraules de trenes que
     es troben seguides
629
630     print("p després de concatenar =", p)
631
632
633     for i in range(len(p)-1,0,-1):
634       if isinstance(p[i], tuple) and isinstance(p[i-1], tuple):
635         p.pop(i-1)                  #eliminem les paraules trena que ja
    hem concatenat
636
637     print("p amb les tuples concatenades eliminades =", p, "\n")
638
639     s = list(p)
640
641
642   return braid
643
644
645 #-------------------------------------------------------------------------
646 #-------------------------------------------------------------------------
647
648 #Donem nom x al labelled peer code del nus que volem utilitzar
649
650 x = [[3,7,-17,1], [-5, 21, 9, -19, -23], [-15, -13, 11], [False, False,
    True, False, True, False, False, False, False, False, False, False]]
651
652 print("Labelled peer code inicial:", x)
653
654 x1, stop, triar, seifert_vertex, eventer, tipus, oddori, label = iteratiu(
    x)
655
656 while stop != "SI":
657   x, stop, triar, seifert_vertex, eventer, tipus, oddori, label = iteratiu
      (x1)
658   if x != []:
659     x1 = x
660
661 if(x1 != []):
662   print("Labelled peer code final:", x1)
663
664 braid = braid_word(triar, seifert_vertex, eventer, tipus, oddori, label)
665
666 print("Trena:")
667 print(braid)
```

Comentaris sobre l'output

En l'output queda reflectit cadascun dels iterats portats a terme, tots ells amb l'objectiu de reduir el nombre de cercles incoherents, mitjançant maniobres Q, fins que no en quedi cap.

Un cop tots els cercles de Seifert són coherents, comença el procés amb el que obtindrem la paraula de la trena que té com a clausura el nus que té x com a *labelled peer code*. Aquest procés també es pot seguir en l'output veient els passos que es duen a terme fins arribar a la paraula buscada. La interpretarem de la següent manera: $\pm$*'s',i* $= \sigma_i^{\pm 1}$, on el signe + o − a l'inici indica si la corda va per sobre o per sota, aleshores *'+','s',i* $= \sigma_i$ i *'-','s',i* $= \sigma_i^{-1}$. Per tant, si ens desplacem al final de l'output que tenim a continuació, veurem que una de les trenes que té com a clausura el nus de la Figura 34 és

$$\sigma_6\sigma_5^{-1}\sigma_5\sigma_5^{-1}\sigma_4\sigma_3\sigma_2\sigma_3^{-1}\sigma_3\sigma_3^{-1}\sigma_2^{-1}\sigma_1\sigma_4^{-1}\sigma_4\sigma_3\sigma_3\sigma_2\sigma_1\sigma_6\sigma_5^{-1}\sigma_4^{-1}\sigma_5\sigma_4\sigma_4\sigma_3\sigma_2$$

Output

Labelled peer code inicial: [[3, 7, -17, 1], [-5, 21, 9, -19, -23], [-15, -13, 11], [False, False, True, False, True, False, False, False, False, False, False, False]]

Cicles esquerrans: [[22, 12, 10], [5, 8], [7, 1, 3], [13, 9, 6, 2, 0, 4, 17, 18, 15, 20], [19, 16, 23, 11, 21, 14]]

Cicles dretans: [[16, 18], [20, 14], [1, 4, 8, 6], [3, 0], [7, 2], [11, 22], [19, 15], [21, 13, 10], [23, 12, 9, 5, 17]]

Vèrtexs:

cs 1 = [0, 2, 4, 3, 1]

cs 2 = [8, 9]

cs 3 = [10, 7]

cs 4 = [11, 6, 5]

cs 5 = [3, 0, 1]

cs 6 = [9, 7]

cs 7 = [10, 6, 4, 2, 8, 11, 5]

Arestes:

s 1 = [0, 4, 8, 6, 2]

s 2 = [16, 18]

s 3 = [20, 14]

s 4 = [22, 12, 10]

s 5 = [7, 1, 3]

s 6 = [19, 15]

s 7 = [21, 13, 9, 5, 17, 23, 11]

Relacions Cercles Seifert:

s 1 $\sim$ [[7, 1, 3], [21, 13, 9, 5, 17, 23, 11]] amb signes [[1], [0]]

s 2 $\sim$ [[19, 15], [21, 13, 9, 5, 17, 23, 11]] amb signes [[0], [0]]

s 3 $\sim$ [[19, 15], [21, 13, 9, 5, 17, 23, 11]] amb signes [[0], [0]]

s 4 $\sim$ [[21, 13, 9, 5, 17, 23, 11]] amb signes [[1]]

s 5 $\sim$ [[0, 4, 8, 6, 2]] amb signes [[1]]

s 6 $\sim$ [[16, 18], [20, 14]] amb signes [[0], [0]]

s 7 $\sim$ [[0, 4, 8, 6, 2], [16, 18], [20, 14], [22, 12, 10]] amb signes [[0], [0], [0], [1]]

On 0 indica - i 1 indica +

Podem fer una maniobra Q amb [[19, 15], [21, 13, 9, 5, 17, 23, 11]]

Però finalment, farem una maniobra Q amb: [[19, 15], [21, 13, 9, 5, 17, 23, 11]]

Com e1 = 19 i e2 = 21 apareixen al cicle esquerrà [19, 16, 23, 11, 21, 14] i pertanyen a [[19, 15], [21, 13, 9, 5, 17, 23, 11]] respectivament, farem la maniobra Q amb ells

Els nous vèrtexs produits per la maniobra Q seran c1 = 10.0 c2 = 12.0

Nou peer code: [[3, 7, -17, 1], [-5, 25, 9, -21, -27], [-15, 23, -13, 19, 11], [False, False, True, False, True, False, False, False, False, False, True, False, False, False]]

Cicles esquerrans: [[24, 20], [26, 12, 10], [5, 8], [7, 1, 3], [13, 9, 6, 2, 0, 4, 17, 18, 15, 22], [21, 23, 14], [25, 19, 16, 27, 11]]

Cicles dretans: [[16, 18], [22, 14], [1, 4, 8, 6], [3, 0], [7, 2], [11, 26], [19, 15, 21, 24], [23, 13, 10, 25, 20], [27, 12, 9, 5, 17]]

Vèrtexs:

cs 1 = [0, 2, 4, 3, 1]

cs 2 = [8, 9]

cs 3 = [11, 7]

cs 4 = [12, 10]

cs 5 = [13, 6, 5]

cs 6 = [3, 0, 1]

cs 7 = [4, 2, 8, 13, 5, 12, 9, 7, 10, 11, 6]

Arestes:

s 1 = [0, 4, 8, 6, 2]

s 2 = [16, 18]

s 3 = [22, 14]

s 4 = [24, 20]

s 5 = [26, 12, 10]

s 6 = [7, 1, 3]

s 7 = [9, 5, 17, 27, 11, 25, 19, 15, 21, 23, 13]

Relacions Cercles Seifert:

s 1 ∼ [[7, 1, 3], [9, 5, 17, 27, 11, 25, 19, 15, 21, 23, 13]] amb signes [[1], [0]]

s 2 ∼ [[9, 5, 17, 27, 11, 25, 19, 15, 21, 23, 13]] amb signes [[0]]

s 3 ∼ [[9, 5, 17, 27, 11, 25, 19, 15, 21, 23, 13]] amb signes [[0]]

s 4 ∼ [[9, 5, 17, 27, 11, 25, 19, 15, 21, 23, 13]] amb signes [[1]]

s 5 ∼ [[9, 5, 17, 27, 11, 25, 19, 15, 21, 23, 13]] amb signes [[1]]

s 6 ∼ [[0, 4, 8, 6, 2]] amb signes [[1]]

s 7 ∼ [[0, 4, 8, 6, 2], [16, 18], [22, 14], [24, 20], [26, 12, 10]] amb signes [[0], [0], [0], [1], [1]]

On 0 indica - i 1 indica +

Podem fer una maniobra Q amb [0, 4, 8, 6, 2] [16, 18]

Podem fer una maniobra Q amb [16, 18] [0, 4, 8, 6, 2]

Podem fer una maniobra Q amb [22, 14] [0, 4, 8, 6, 2]

Podem fer una maniobra Q amb [24, 20] [26, 12, 10]

Podem fer una maniobra Q amb [26, 12, 10] [24, 20]

Però finalment, farem una maniobra Q amb: [[26, 12, 10], [24, 20]]

Com 10 i 20 apareixen al cicle dretà [23, 13, 10, 25, 20] i pertanyen a [[26, 12, 10], [24, 20]] respectivament, farem la maniobra Q amb ells

Els nous vèrtexs produits per la maniobra Q seran c1 = 5.0 c2 = 11.0

Nou peer code: [[3, 7, -19, 1], [-5, -23, 29, 9, -25, -31], [-17, -11, 27, -15, 21, 13], [False, False, True, False, True, True, False, False, False, False, False, False, True, False, False, False]]

Cicles esquerrans: [[5, 8], [7, 1, 3], [11, 24, 28, 22], [15, 9, 6, 2, 0, 4, 19, 20, 17, 26], [23, 12, 30, 14, 10], [25, 27, 16], [29, 21, 18, 31, 13]]

Cicles dretans: [[18, 20], [26, 16], [1, 4, 8, 6], [3, 0], [7, 2], [13, 30], [23, 11], [25, 28, 21, 17], [27, 15, 10, 24], [29, 22, 12], [31, 14, 9, 5, 19]]

Vèrtexs:

cs 1 = [0, 2, 4, 3, 1]

cs 2 = [9, 10]

cs 3 = [13, 8]

cs 4 = [14, 11, 6, 15, 7, 5, 12]

cs 5 = [3, 0, 1]

cs 6 = [11, 5]

cs 7 = [12, 13, 7, 4, 2, 9, 15, 6, 14, 10, 8]

Arestes:

s 1 = [0, 4, 8, 6, 2]

s 2 = [18, 20]

s 3 = [26, 16]

s 4 = [28, 22, 12, 30, 14, 10, 24]

s 5 = [7, 1, 3]

s 6 = [23, 11]

s 7 = [25, 27, 15, 9, 5, 19, 31, 13, 29, 21, 17]

Relacions Cercles Seifert:

s 1 $\sim$ [[7, 1, 3], [25, 27, 15, 9, 5, 19, 31, 13, 29, 21, 17]] amb signes [[1], [0]]

s 2 $\sim$ [[25, 27, 15, 9, 5, 19, 31, 13, 29, 21, 17]] amb signes [[0]]

s 3 $\sim$ [[25, 27, 15, 9, 5, 19, 31, 13, 29, 21, 17]] amb signes [[0]]

s 4 $\sim$ [[23, 11], [25, 27, 15, 9, 5, 19, 31, 13, 29, 21, 17]] amb signes [[0], [1]]

s 5 $\sim$ [[0, 4, 8, 6, 2]] amb signes [[1]]

s 6 $\sim$ [[28, 22, 12, 30, 14, 10, 24]] amb signes [[0]]

s 7 $\sim$ [[0, 4, 8, 6, 2], [18, 20], [26, 16], [28, 22, 12, 30, 14, 10, 24]] amb signes [[0], [0], [0], [1]]

On 0 indica - i 1 indica +

Podem fer una maniobra Q amb [0, 4, 8, 6, 2] [18, 20]

Podem fer una maniobra Q amb [18, 20] [0, 4, 8, 6, 2]

Podem fer una maniobra Q amb [26, 16] [0, 4, 8, 6, 2]

Però finalment, farem una maniobra Q amb: [[26, 16], [0, 4, 8, 6, 2]]

Com e1 = 26 i e2 = 0 apareixen al cicle esquerrà [15, 9, 6, 2, 0, 4, 19, 20, 17, 26] i pertanyen a [[26, 16], [0, 4, 8, 6, 2]] respectivament, farem la maniobra Q amb ells

Com e1 = 26 i e2 = 4 apareixen al cicle esquerrà [15, 9, 6, 2, 0, 4, 19, 20, 17, 26] i pertanyen a [[26, 16], [0, 4, 8, 6, 2]] respectivament, farem la maniobra Q amb ells

Els nous vèrtexs produits per la maniobra Q seran c1 = 2.0 c2 = 14.0

Nou peer code: [[3, 9, 29, -21, 1], [-7, -25, 33, 11, -27, -35], [-19, -13, 31, 5, -17, 23, 15], [False, False, True, True, False, True, True, False, False, False, False, False, False, True, False, False, False, False]]

Cicles esquerrans: [[7, 10], [9, 1, 3], [13, 26, 32, 24], [17, 11, 8, 2, 0, 4, 30], [19, 28, 6, 21, 22], [25, 14, 34, 16, 12], [29, 5], [31, 18, 27], [33, 23, 20, 35, 15]]

Cicles dretans: [[20, 22], [3, 0], [5, 30, 18, 28], [9, 2], [15, 34], [25, 13], [27, 32, 23, 19], [29, 6, 10, 8, 1, 4], [31, 17, 12, 26], [33, 24, 14], [35, 16, 11, 7, 21]]

Vèrtexs:

cs 1 = [10, 11]

cs 2 = [14, 3, 5, 4, 1, 0, 2, 15, 9]

cs 3 = [16, 12, 7, 17, 8, 6, 13]

cs 4 = [4, 0, 1]

cs 5 = [12, 6]

cs 6 = [14, 2]

cs 7 = [15, 8, 5, 3, 10, 17, 7, 16, 11, 9, 13]

Arestes:

s 1 = [20, 22]

s 2 = [28, 6, 10, 8, 2, 0, 4, 30, 18]

s 3 = [32, 24, 14, 34, 16, 12, 26]

s 4 = [9, 1, 3]

s 5 = [25, 13]

s 6 = [29, 5]

s 7 = [31, 17, 11, 7, 21, 35, 15, 33, 23, 19, 27]

Relacions Cercles Seifert:

s 1 $\sim$ [[31, 17, 11, 7, 21, 35, 15, 33, 23, 19, 27]] amb signes [[0]]

s 2 $\sim$ [[9, 1, 3], [29, 5], [31, 17, 11, 7, 21, 35, 15, 33, 23, 19, 27]] amb signes [[1], [1], [0]]

s 3 $\sim$ [[25, 13], [31, 17, 11, 7, 21, 35, 15, 33, 23, 19, 27]] amb signes [[0], [1]]

s 4 $\sim$ [[28, 6, 10, 8, 2, 0, 4, 30, 18]] amb signes [[1]]

s 5 $\sim$ [[32, 24, 14, 34, 16, 12, 26]] amb signes [[0]]

s 6 ∼ [[28, 6, 10, 8, 2, 0, 4, 30, 18]] amb signes [[1]]

s 7 ∼ [[20, 22], [28, 6, 10, 8, 2, 0, 4, 30, 18], [32, 24, 14, 34, 16, 12, 26]] amb signes [[0], [0], [1]]

On 0 indica - i 1 indica +

Podem fer una maniobra Q amb [9, 1, 3] [29, 5]

Podem fer una maniobra Q amb [29, 5] [9, 1, 3]

Podem fer una maniobra Q amb [20, 22] [28, 6, 10, 8, 2, 0, 4, 30, 18]

Podem fer una maniobra Q amb [28, 6, 10, 8, 2, 0, 4, 30, 18] [20, 22]

Però finalment, farem una maniobra Q amb: [[28, 6, 10, 8, 2, 0, 4, 30, 18], [20, 22]]

Com e1 = 28 i e2 = 22 apareixen al cicle esquerrà [19, 28, 6, 21, 22] i pertanyen a [[28, 6, 10, 8, 2, 0, 4, 30, 18], [20, 22]] respectivament, farem la maniobra Q amb ells

Els nous vèrtexs produits per la maniobra Q seran c1 = 11.0 c2 = 15.0

Nou peer code: [[3, 9, 33, -21, 1], [-7, -27, 37, 11, -29, -39], [31, -19, -13, 35, 23, 5, -17, 25, 15], [False, False, True, True, False, True, True, False, False, False, False, True, False, False, True, False, False, False, False, False]]

Cicles esquerrans: [[7, 10], [9, 1, 3], [13, 28, 36, 26], [17, 11, 8, 2, 0, 4, 34], [19, 30, 24], [21, 22, 32, 6], [27, 14, 38, 16, 12], [31, 23], [33, 5], [35, 18, 29], [37, 25, 20, 39, 15]]

Cicles dretans: [[3, 0], [9, 2], [15, 38], [23, 32, 5, 34, 18, 30], [27, 13], [29, 36, 25, 19], [31, 24, 20, 22], [33, 6, 10, 8, 1, 4], [35, 17, 12, 28], [37, 26, 14], [39, 16, 11, 7, 21]]

Vèrtexs:

cs 1 = [15, 12, 10, 11, 16, 3, 5, 4, 1, 0, 2, 17, 9]

cs 2 = [18, 13, 7, 19, 8, 6, 14]

cs 3 = [4, 0, 1]

cs 4 = [13, 6]

cs 5 = [15, 11]

cs 6 = [16, 2]

cs 7 = [17, 8, 5, 3, 10, 19, 7, 18, 12, 9, 14]

Arestes:

s 1 = [30, 24, 20, 22, 32, 6, 10, 8, 2, 0, 4, 34, 18]

s 2 = [36, 26, 14, 38, 16, 12, 28]

s 3 = [9, 1, 3]

s 4 = [27, 13]

s 5 = [31, 23]

s 6 = [33, 5]

s 7 = [35, 17, 11, 7, 21, 39, 15, 37, 25, 19, 29]

Relacions Cercles Seifert:

s 1 ∼ [[9, 1, 3], [31, 23], [33, 5], [35, 17, 11, 7, 21, 39, 15, 37, 25, 19, 29]] amb signes [[1], [1], [1], [0]]

s 2 ∼ [[27, 13], [35, 17, 11, 7, 21, 39, 15, 37, 25, 19, 29]] amb signes [[0], [1]]

s 3 ∼ [[30, 24, 20, 22, 32, 6, 10, 8, 2, 0, 4, 34, 18]] amb signes [[1]]

s 4 ∼ [[36, 26, 14, 38, 16, 12, 28]] amb signes [[0]]

s 5 ∼ [[30, 24, 20, 22, 32, 6, 10, 8, 2, 0, 4, 34, 18]] amb signes [[1]]

s 6 ∼ [[30, 24, 20, 22, 32, 6, 10, 8, 2, 0, 4, 34, 18]] amb signes [[1]]

s 7 ∼ [[30, 24, 20, 22, 32, 6, 10, 8, 2, 0, 4, 34, 18], [36, 26, 14, 38, 16, 12, 28]] amb signes [[0], [1]]

On 0 indica - i 1 indica +

Podem fer una maniobra Q amb [9, 1, 3] [31, 23]

Podem fer una maniobra Q amb [31, 23] [9, 1, 3]

Podem fer una maniobra Q amb [33, 5] [9, 1, 3]

Però finalment, farem una maniobra Q amb: [[33, 5], [9, 1, 3]]

Com 33 i 1 apareixen al cicle dretà [33, 6, 10, 8, 1, 4] i pertanyen a [[33, 5], [9, 1, 3]] respectivament, farem la maniobra Q amb ells

Els nous vèrtexs produits per la maniobra Q seran c1 = 1.0 c2 = 18.0

Nou peer code: [[5, -35, 11, 37, -23, 3], [-9, -29, 41, 13, -31, -43], [33, -21, -15, 39, 25, 7, -1, -19, 27, 17], [False, True, False, True, True, False, True, True, False, False, False, False, True, False, False, True, False, False, False, False, False, False]]

Cicles esquerrans: [[9, 12], [11, 3, 36, 1, 5], [15, 30, 40, 28], [19, 13, 10, 4, 0, 6, 38], [21, 32, 26], [23, 24, 34, 8], [29, 16, 42, 18, 14], [33, 25], [35, 7, 37, 2], [39, 20, 31], [41, 27, 22, 43, 17]]

Cicles dretans: [[36, 2], [5, 0], [11, 4], [17, 42], [25, 34, 7, 38, 20, 32], [29, 15], [31, 40, 27, 21], [33, 26, 22, 24], [35, 8, 12, 10, 3], [37, 1, 6], [39, 19, 14, 30], [41, 28, 16], [43, 18, 13, 9, 23]]

Vèrtexs:

cs 1 = [18, 1]

cs 2 = [19, 10, 16, 13, 11, 12, 17, 4, 6, 5, 2, 0, 3]

cs 3 = [20, 14, 8, 21, 9, 7, 15]

cs 4 = [14, 7]

cs 5 = [16, 12]

cs 6 = [17, 3, 18, 0, 2, 5, 1]

cs 7 = [19, 9, 6, 4, 11, 21, 8, 20, 13, 10, 15]

Arestes:

s 1 = [36, 2]

s 2 = [38, 20, 32, 26, 22, 24, 34, 8, 12, 10, 4, 0, 6]

s 3 = [40, 28, 16, 42, 18, 14, 30]

s 4 = [29, 15]

s 5 = [33, 25]

s 6 = [35, 7, 37, 1, 5, 11, 3]

s 7 = [39, 19, 13, 9, 23, 43, 17, 41, 27, 21, 31]

Relacions Cercles Seifert:

s 1 ∼ [[35, 7, 37, 1, 5, 11, 3]] amb signes [[0]]

s 2 ∼ [[33, 25], [35, 7, 37, 1, 5, 11, 3], [39, 19, 13, 9, 23, 43, 17, 41, 27, 21, 31]] amb signes [[1], [1], [0]]

s 3 ∼ [[29, 15], [39, 19, 13, 9, 23, 43, 17, 41, 27, 21, 31]] amb signes [[0], [1]]

s 4 ∼ [[40, 28, 16, 42, 18, 14, 30]] amb signes [[0]]

s 5 ∼ [[38, 20, 32, 26, 22, 24, 34, 8, 12, 10, 4, 0, 6]] amb signes [[1]]

s 6 ∼ [[36, 2], [38, 20, 32, 26, 22, 24, 34, 8, 12, 10, 4, 0, 6]] amb signes [[0], [1]]

s 7 ∼ [[38, 20, 32, 26, 22, 24, 34, 8, 12, 10, 4, 0, 6], [40, 28, 16, 42, 18, 14, 30]] amb signes [[0], [1]]

On 0 indica - i 1 indica +

Podem fer una maniobra Q amb [33, 25] [35, 7, 37, 1, 5, 11, 3]

Podem fer una maniobra Q amb [35, 7, 37, 1, 5, 11, 3] [33, 25]

Però finalment, farem una maniobra Q amb: [[35, 7, 37, 1, 5, 11, 3], [33, 25]]

Com 7 i 25 apareixen al cicle dretà [25, 34, 7, 38, 20, 32] i pertanyen a [[35, 7, 37, 1, 5, 11, 3], [33, 25]] respectivament, farem la maniobra Q amb ells

Els nous vèrtexs produits per la maniobra Q seran c1 = 4.0 c2 = 14.0

Nou peer code: [[5, -39, 13, 41, -27, -25, 3], [-11, -33, 45, 15, -35, -47], [37, -7, -23, -17, 43, 29, 9, -1, -21, 31, 19], [False, True, False, True, True, True, False, True, True, False, False, False, False, True, False, False, False, True, False, False, False, False, False, False]]

Cicles esquerrans: [[11, 14], [13, 3, 40, 1, 5], [17, 34, 44, 32], [21, 15, 12, 4, 0, 6, 42], [23, 36, 30], [25, 26, 38, 10], [33, 18, 46, 20, 16], [37, 29, 8, 27], [39, 9, 28, 7, 41, 2], [43, 22, 35], [45, 31, 24, 47, 19]]

Cicles dretans: [[28, 8], [40, 2], [5, 0], [13, 4], [19, 46], [27, 38, 9], [29, 7, 42, 22, 36], [33, 17], [35, 44, 31, 23], [37, 30, 24, 26], [39, 10, 14, 12, 3], [41, 1, 6], [43, 21, 16, 34], [45, 32, 18], [47, 20, 15, 11, 25]]

Vèrtexs:

cs 1 = [14, 4]

cs 2 = [20, 1]

cs 3 = [21, 11, 18, 15, 12, 13, 19, 5, 7, 6, 2, 0, 3]

cs 4 = [22, 16, 9, 23, 10, 8, 17]

cs 5 = [16, 8]

cs 6 = [18, 14, 3, 20, 0, 2, 6, 1, 19, 4, 13]

cs 7 = [21, 10, 7, 5, 12, 23, 9, 22, 15, 11, 17]

Arestes:

s 1 = [28, 8]

s 2 = [40, 2]

s 3 = [42, 22, 36, 30, 24, 26, 38, 10, 14, 12, 4, 0, 6]

s 4 = [44, 32, 18, 46, 20, 16, 34]

s 5 = [33, 17]

s 6 = [37, 29, 7, 41, 1, 5, 13, 3, 39, 9, 27]

s 7 = [43, 21, 15, 11, 25, 47, 19, 45, 31, 23, 35]

Relacions Cercles Seifert:

s 1 $\sim$ [[37, 29, 7, 41, 1, 5, 13, 3, 39, 9, 27]] amb signes [[0]]

s 2 $\sim$ [[37, 29, 7, 41, 1, 5, 13, 3, 39, 9, 27]] amb signes [[0]]

s 3 $\sim$ [[37, 29, 7, 41, 1, 5, 13, 3, 39, 9, 27], [43, 21, 15, 11, 25, 47, 19, 45, 31, 23, 35]] amb signes [[1], [0]]

s 4 $\sim$ [[33, 17], [43, 21, 15, 11, 25, 47, 19, 45, 31, 23, 35]] amb signes [[0], [1]]

s 5 $\sim$ [[44, 32, 18, 46, 20, 16, 34]] amb signes [[0]]

s 6 $\sim$ [[28, 8], [40, 2], [42, 22, 36, 30, 24, 26, 38, 10, 14, 12, 4, 0, 6]] amb signes [[0], [0], [1]]

s 7 ∼ [[42, 22, 36, 30, 24, 26, 38, 10, 14, 12, 4, 0, 6], [44, 32, 18, 46, 20, 16, 34]] amb signes [[0], [1]]

On 0 indica - i 1 indica +

Podem fer una maniobra Q amb [28, 8] [40, 2]

Podem fer una maniobra Q amb [40, 2] [28, 8]

Però finalment, farem una maniobra Q amb: [[40, 2], [28, 8]]

Com e1 = 2 i e2 = 28 apareixen al cicle esquerrà [39, 9, 28, 7, 41, 2] i pertanyen a [[40, 2], [28, 8]] respectivament, farem la maniobra Q amb ells

Els nous vèrtexs produits per la maniobra Q seran c1 = 1.0 c2 = 15.0

Nou peer code: [[7, 31, -43, 15, 45, -29, -27, 5], [-13, -37, 49, 17, -39, -51], [41, 3, -9, -25, -19, 47, 33, 11, -1, -23, 35, 21], [False, True, True, False, True, True, True, False, True, True, False, False, False, False, True, False, False, False, False, True, False, False, False, False, False, False]]

Cicles esquerrans: [[13, 16], [15, 5, 44, 1, 7], [19, 38, 48, 36], [23, 17, 14, 6, 0, 8, 46], [25, 40, 34], [27, 28, 42, 12], [31, 3], [37, 20, 50, 22, 18], [41, 33, 10, 29], [43, 11, 30, 4], [45, 2, 32, 9], [47, 24, 39], [49, 35, 26, 51, 21]]

Cicles dretans: [[3, 32, 10, 30], [7, 0], [15, 6], [21, 50], [29, 42, 11], [31, 4, 44, 2], [33, 9, 46, 24, 40], [37, 19], [39, 48, 35, 25], [41, 34, 26, 28], [43, 12, 16, 14, 5], [45, 1, 8], [47, 23, 18, 38], [49, 36, 20], [51, 22, 17, 13, 27]]

Vèrtexs:

cs 1 = [22, 1, 16, 5, 15, 2]

cs 2 = [23, 12, 20, 17, 13, 14, 21, 6, 8, 7, 3, 0, 4]

cs 3 = [24, 18, 10, 25, 11, 9, 19]

cs 4 = [15, 1]

cs 5 = [18, 9]

cs 6 = [20, 16, 4, 22, 0, 3, 7, 2, 21, 5, 14]

cs 7 = [23, 11, 8, 6, 13, 25, 10, 24, 17, 12, 19]

Arestes:

s 1 = [44, 2, 32, 10, 30, 4]

s 2 = [46, 24, 40, 34, 26, 28, 42, 12, 16, 14, 6, 0, 8]

s 3 = [48, 36, 20, 50, 22, 18, 38]

s 4 = [31, 3]

s 5 = [37, 19]

s 6 = [41, 33, 9, 45, 1, 7, 15, 5, 43, 11, 29]

s 7 = [47, 23, 17, 13, 27, 51, 21, 49, 35, 25, 39]

Relacions Cercles Seifert:

s 1 ∼ [[31, 3], [41, 33, 9, 45, 1, 7, 15, 5, 43, 11, 29]] amb signes [[1], [0]]

s 2 ∼ [[41, 33, 9, 45, 1, 7, 15, 5, 43, 11, 29], [47, 23, 17, 13, 27, 51, 21, 49, 35, 25, 39]] amb signes [[1], [0]]

s 3 ∼ [[37, 19], [47, 23, 17, 13, 27, 51, 21, 49, 35, 25, 39]] amb signes [[0], [1]]

s 4 ∼ [[44, 2, 32, 10, 30, 4]] amb signes [[1]]

s 5 ∼ [[48, 36, 20, 50, 22, 18, 38]] amb signes [[0]]

s 6 ∼ [[44, 2, 32, 10, 30, 4], [46, 24, 40, 34, 26, 28, 42, 12, 16, 14, 6, 0, 8]] amb signes [[0], [1]]

s 7 ∼ [[46, 24, 40, 34, 26, 28, 42, 12, 16, 14, 6, 0, 8], [48, 36, 20, 50, 22, 18, 38]] amb signes [[0], [1]]

On 0 indica - i 1 indica +

No cal fer cap maniobra Q

Labelled peer code final: [[7, 31, -43, 15, 45, -29, -27, 5], [-13, -37, 49, 17, -39, -51], [41, 3, -9, -25, -19, 47, 33, 11, -1, -23, 35, 21], [False, True, True, False, True, True, True, False, True, True, False, False, False, False, True, False, False, False, False, True, False, False, False, False, False]]

El cercle de Seifert més interior (inicial) té vèrtexs: [15, 1]

El cercle de Seifert més exterior (final) té vèrtexs: [18, 9]

parent p: [22, 1, 16, 5, 15, 2]

p després de introduir-hi s = [22, ('+', 's', 1), 16, 5, ('+', 's', 1), 2]

p ficant si hi ha braid word inicial al final = [22, ('+', 's', 1), 16, 5, ('+', 's', 1), 2]

p després de concatenar = [22, ('+', 's', 1), 16, 5, ('+', 's', 1), 2]

p amb les tuples concatenades eliminades = [22, ('+', 's', 1), 16, 5, ('+', 's', 1), 2]

parent p: [20, 16, 4, 22, 0, 3, 7, 2, 21, 5, 14]

p després de introduir-hi s = [20, ('+', 's', 2), 4, ('-', 's', 2, '+', 's', 1), 0, 3, 7, ('+', 's', 2), 21, ('+', 's', 2, '+', 's', 1), 14]

p ficant si hi ha braid word inicial al final = [20, ('+', 's', 2), 4, ('-', 's', 2, '+', 's', 1), 0, 3, 7, ('+', 's', 2), 21, ('+', 's', 2, '+', 's', 1), 14]

p després de concatenar = [20, ('+', 's', 2), 4, ('-', 's', 2, '+', 's', 1), 0, 3, 7, ('+', 's', 2), 21, ('+', 's', 2, '+', 's', 1), 14]

p amb les tuples concatenades eliminades = [20, ('+', 's', 2), 4, ('-', 's', 2, '+', 's', 1), 0, 3, 7, ('+', 's', 2), 21, ('+', 's', 2, '+', 's', 1), 14]

parent p: [23, 12, 20, 17, 13, 14, 21, 6, 8, 7, 3, 0, 4]

p després de introduir-hi s = [23, 12, ('+', 's', 3, '+', 's', 2), 17, 13, ('+', 's', 3), ('+', 's', 3, '+', 's', 2, '+', 's', 1), 6, 8, ('+', 's', 3, '+', 's', 2), ('-', 's', 3), ('+', 's', 3), ('-', 's', 3, '-', 's', 2, '+', 's', 1)]

p ficant si hi ha braid word inicial al final = [23, 12, ('+', 's', 3, '+', 's', 2), 17, 13, ('+', 's', 3), ('+', 's', 3, '+', 's', 2, '+', 's', 1), 6, 8, ('+', 's', 3, '+', 's', 2), ('-', 's', 3), ('+', 's', 3), ('-', 's', 3, '-', 's', 2, '+', 's', 1)]

p després de concatenar = [23, 12, ('+', 's', 3, '+', 's', 2), 17, 13, ('+', 's', 3), ('+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 6, 8, ('+', 's', 3, '+', 's', 2), ('+', 's', 3, '+', 's', 2, '-', 's', 3), ('+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3), ('+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1)]

p amb les tuples concatenades eliminades = [23, 12, ('+', 's', 3, '+', 's', 2), 17, 13, ('+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 6, 8, ('+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1)]

parent p: [23, 11, 8, 6, 13, 25, 10, 24, 17, 12, 19]

p després de introduir-hi s = [('-', 's', 4), 11, ('+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1), ('-', 's', 4), ('+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 25, 10, 24, ('+', 's', 4), ('+', 's', 4, '+', 's', 3, '+', 's', 2), 19]

p ficant si hi ha braid word inicial al final = [11, ('+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1), ('-', 's', 4), ('+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 25, 10, 24, ('+', 's', 4), ('+', 's', 4, '+', 's', 3, '+', 's', 2), 19, ('-', 's', 4)]

p després de concatenar = [11, ('+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1), ('+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4), ('+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 25, 10, 24, ('+', 's', 4), ('+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2), 19, ('-', 's', 4)]

p amb les tuples concatenades eliminades = [11, ('+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 25, 10, 24, ('+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2), 19, ('-', 's', 4)]

parent p: [24, 18, 10, 25, 11, 9, 19]

p després de introduir-hi s = [('+', 's', 5, '+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2), 18, ('-', 's', 5), ('+', 's', 5), ('-', 's', 5, '+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 9, ('-', 's', 5, '-', 's', 4)]

p ficant si hi ha braid word inicial al final = [18, ('-', 's', 5), ('+', 's', 5), ('-', 's', 5, '+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 9, ('-', 's', 5, '-', 's', 4), ('+', 's', 5, '+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2)]

p després de concatenar = [18, ('-', 's', 5), ('-', 's', 5, '+', 's', 5), ('-', 's', 5, '+', 's', 5, '-', 's', 5, '+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 9, ('-', 's', 5, '-', 's', 4), ('-', 's', 5, '-', 's', 4, '+', 's', 5, '+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2)]

p amb les tuples concatenades eliminades = [18, ('-', 's', 5, '+', 's', 5, '-', 's', 5, '+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), 9, ('-', 's', 5, '-', 's', 4, '+', 's', 5, '+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2)]

parent p: [18, 9]

p després de introduir-hi s = [('+', 's', 6, '-', 's', 5, '+', 's', 5, '-', 's', 5, '+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1), ('+', 's', 6, '-', 's', 5, '-', 's', 4, '+', 's', 5, '+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2)]

Trena: ['+', 's', 6, '-', 's', 5, '+', 's', 5, '-', 's', 5, '+', 's', 4, '+', 's', 3, '+', 's', 2, '-', 's', 3, '+', 's', 3, '-', 's', 3, '-', 's', 2, '+', 's', 1, '-', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 3, '+', 's', 2, '+', 's', 1, '+', 's', 6, '-', 's', 5, '-', 's', 4, '+', 's', 5, '+', 's', 4, '+', 's', 4, '+', 's', 3, '+', 's', 2]

Referències

[1] Cromwell, Peter R. *Knots and links*. Cambridge University Press, Cambridge, 2004.

[2] Birman, Joan S. *On the conjugacy problem in the braid group*. Canadian J. Math, 1982.

[3] Adams, Colin C. *The knot book*. American Mathematical Society, Providence, RI, 2004.

[4] Burde, Gerhard and Zieschang, Heiner *Knots*. Walter de Gruyter & Co., Berlin, 2003.

[5] Birman, Joan S. i Brendle, Tara E. *Knots*. Walter de Gruyter & Co., Berlin, 2003.

[6] Yamada, Shuji *The minimal number of Seifert circles equals the braid index of a link*. Springer-Verlag, Inventiones mathematicae, 1987.

[7] Artin, Emil *Theorie der zöpfe*. Springer Berlin/Heidelberg, 1925.

[8] Artin, Emil *Theory of braids*. Ann. of Math, 1947.

[9] Bartholomew, Andrew *An application of Vogel's algorithm*. 2011.
`http://www.layer8.co.uk/maths/download/vogel.pdf`

[10] Navarro Pérez, Miguel Ángel *Trenzas y nudos*. 2016.

[11] Jackson, Nicholas *Notes on braid groups*. 2004.
`https://homepages.warwick.ac.uk/ maseay/doc/braids.pdf`

[12] Wilson, Jenny *The Geometry and Topology of Braid Groups*. 2018.
`https://pdfs.semanticscholar.org/f9ef/7cc21b3eecdf7b2aa275f0959ecc8e173c50.pdf?`$_g a =$ 2.178603751.2122786072.1579264737 − 1879762178.1579264737