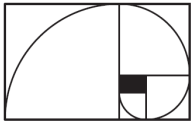
SOCIEDADE BRASILEIRA DE MATEMÁTICA

ENSAIOS MATEMÁTICOS
2016, Volume **30**, 246–262
https://doi.org/10.21711/217504322016/em304



An algorithm for computing the Seifert matrix of a link from a braid representation

Julia Collins

Abstract. A Seifert surface of a knot or link in S^3 is an oriented surface in S^3 whose boundary coincides with that of the link. A corresponding Seifert matrix has as its entries the linking numbers of a set of homology generators of the surface. Thus a Seifert matrix encodes essential information about the structure of a link and, unsurprisingly, can be used to define powerful invariants, such as the Alexander polynomial. The program SeifertView has been designed to visualise Seifert surfaces given by braid representations, but it does not give the user any technical information about the knot or link. This article describes an algorithm which could work alongside SeifertView and compute a Seifert matrix from the same braids and surfaces. It also calculates the genus of the surface, the Alexander polynomial of the knot and the signature of the knot.

2010 Mathematics Subject Classification: 57M25, 20F36.

Contents

1 Introduction

For every link $L \subset S^3$ there exists a compact orientable surface $\Sigma \subset S^3$ with L as its boundary. This result is due to Frankl and Pontrjagin, who proved it in 1930 ([4]), but the most well-known proof is due to Herbert Seifert in 1935 ([9]). He constructed an explicit algorithm for finding such a surface from a knot diagram, and subsequently any such surface became known as a *Seifert surface*.

Given a link L and a connected Seifert surface Σ, we can find a set of generators for the first homology group of Σ and work out the pairwise linking numbers for these. This defines an integer matrix which is called a *Seifert matrix*. Being in such a convenient algebraic format makes it easy to define powerful invariants from it, such as the Alexander polynomial, and it is also easily generalisable to higher dimensional knots.

The program SeifertView, created by Jarke J. van Wijk and Arjeh M. Cohen ([10]), was designed to help people visualise Seifert surfaces of knots and links. Every link has a representation as a closed braid, and the braid notation is a very convenient format in which to put the link as an input to the computer. SeifertView takes a braid as an input and applies Seifert's algorithm to it to obtain a Seifert surface for each knot and link. But although this is an excellent program for visualising these 3D surfaces, the user is not provided with any technical information about the surface they are looking at.

This paper describes an algorithm called *Seifert* for finding the Seifert matrix of the Seifert surface and homology generators determined algorithmically from a braid representation of a knot or link. From this we can find simple invariants such as the determinant of the knot or the genus of the surface, as well as more complicated invariants like the Alexander polynomial. As with SeifertView, an extended program which can cope with pretzel notation is developed, as this is a more concise way of encoding a link in a braid-like representation and produces surfaces of lower genus than that produced by the original program. This notation allows for Seifert matrices which contain any integer value, whereas the original *Seifert* program produces matrices containing only ± 1s and zeros.

Section 2 provides the necessary background on Seifert matrices and braids. Section 3 describes the main idea behind the program and the details of how to find a Seifert matrix for a braid. Finally, in Section 4 we discuss extensions which could be made to the program and directions for further research.

A website accompanying this algorithm can be found at `http://www.maths.ed.ac.uk/~jcollins/SeifertMatrix`.

Program summary

This algorithm has been implemented in Matlab, C++ and Javascript.
Input: The user has two equivalent choices of how to input a braid representation. (See Section 2.2 to find out what the notation means.)

1. As a numerical vector, e.g., [1 1 1] for the trefoil and [1 -2 1 -2] for the figure eight knot.
2. As an alphabetical string, e.g. 'AAA' for the trefoil and 'AbAb' for the figure eight knot.

Output: The *Seifert* program outputs the following information:

1. A Seifert matrix M for the canonical Seifert surface associated with the braid.
2. The genus of the associated surface.
3. The number of disconnected surfaces produced by Seifert's algorithm on the braid.
4. A declaration of whether the braid is a knot, and if not, how many components it has.
5. The coefficients of the Alexander polynomial of the braid.

2 Background information

We start off by reminding the reader of the relevant knot theory definitions. A *knot* is an embedding of S^1 into S^3, and a *link* is an embedding of a disjoint union of circles into S^3.

2.1 Seifert surfaces and matrices

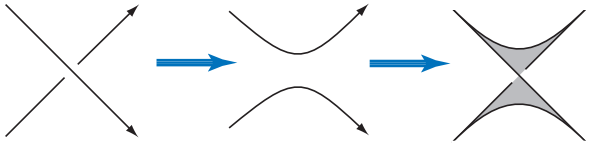
Definition 2.1. A *Seifert surface* of an oriented link $L \subset S^3$ is a compact orientable surface whose boundary coincides with that of the link.

Remark 2.2. We will not assume that the Seifert surface must be connected, although this is sometimes part of the definition in other texts (e.g. [6] and [8]).

Theorem 2.3. Every oriented link has a Seifert surface.

Proof. (Seifert [9]) (The following method is known as *Seifert's algorithm.*) Fix an oriented projection of the link. At each crossing of the projection there are two incoming strands and two outgoing strands. Eliminate the crossings by swapping which incoming strand is connected to which

outgoing strand (see diagram below). The result is a set of non-intersecting oriented topological circles called *Seifert circles*, which, if they are nested, we imagine being at different heights perpendicular to the plane with the z-coordinate changing linearly with the nesting. Fill in these circles, giving discs, and connect the discs together by attaching twisted bands where the crossings used to be. The direction of the twist corresponds to the direction of the crossing in the link.



This procedure forms a surface which has the link as its boundary, and it is not hard to see that it is orientable. If we colour the Seifert circles according to their orientation, e.g. the upward face blue for clockwise and the upward face red for anticlockwise, then the twists will consistently continue the colouring to the whole surface. ■

Of course, the Seifert surface of a link is not unique in any way, and even Seifert's algorithm applied to different link projections will result in different surfaces. Since the surface itself cannot be an invariant of the link, we need to look for other information given in the Seifert surface. The first of these is the genus.

Definition 2.4. The *genus* of a connected orientable surface with boundary is the genus of the corresponding surface without boundary obtained by capping off each boundary component with a disc. The genus of a disconnected surface Σ made up of k connected components $\Sigma = \Sigma_1 \sqcup \cdots \sqcup \cdots \sqcup \Sigma_k$ is the sum $g(\Sigma_1) + \cdots + g(\Sigma_k)$ of the individual genera.

Definition 2.5. The *genus* of a link is the minimal genus of all the Seifert surfaces of the link.

In practice this is a very difficult invariant to calculate, especially since, in some cases, there is no projection of the knot for which Seifert's algorithm produces a surface of minimal genus ([7]). However, it has some useful properties, the main one being that it is additive, i.e. $g(K_1 + K_2) = g(K_1) + g(K_2)$. (See [6], pg 17, for a proof.) Also, for alternating knots (i.e. a knot whose oriented diagram alternates between over and under crossings as you follow it around), Seifert's algorithm does indeed provide a minimal genus surface ([5]).

The genus of a connected Seifert surface produced using Seifert's algorithm is easily calculable using the following formula.

Lemma 2.6. If a connected Seifert surface is produced from the projection of a link with s Seifert circles, c crossings and n components then it will have genus

$$g = 1 - \frac{s - c + n}{2}. \tag{1}$$

Proof. Suppose we have a connected Seifert surface with s and c given as above. We can contract the surface down to a graph which has s vertices (each Seifert circle contracts to a point) and c edges. The Euler characteristic χ_S of this surface is

$$\chi_S = \#\text{vertices} - \#\text{edges} + \#\text{faces} = s - c + 0.$$

The Euler characteristic χ_C of the corresponding closed surface (without boundary) is $s-c+n$. Finally, the genus is calculated from $g = \frac{1}{2}(2-\chi_C)$, which gives us the result. Putting $n = 1$ gives us the result in the case of a knot. ∎

Corollary 2.7. A Seifert surface $\Sigma = \Sigma_1 \sqcup \Sigma_2 \sqcup \cdots \sqcup \Sigma_k$ with k connected components, produced using Seifert's algorithm, has genus

$$g = k - \frac{s - c + n}{2}. \tag{2}$$

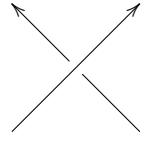
with s, c and n as in Lemma 2.6.

Remark 2.8. If Seifert's algorithm gives us a disconnected surface which is a union of connected surfaces $\Sigma_1, \ldots, \Sigma_k$ then the genus of this is the same as the genus of the connected sum:

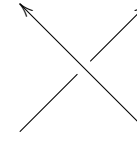
$$g(\Sigma_1 \sqcup \cdots \sqcup \Sigma_k) = g(\Sigma_1) + \cdots + g(\Sigma_k) = g(\Sigma_1 \# \ldots \# \Sigma_k).$$

Although a Seifert surface is not an invariant of a link, it can still provide us with a lot of information about the structure of the link. One object which encapsulates this information is called a *Seifert matrix*.

Definition 2.9. Suppose that D is a regular oriented projection of a two-component link with components J and K (so that whenever two strands meet in the projection they do so transversally). Assign each crossing a sign:



+1 (right-handed)



−1 (left-handed)

The *linking number* $\mathrm{lk}(J,K) \in \mathbb{Z}$ of J and K is half the sum of the signs of the crossings at which one strand is from J and the other is from K. Equivalently we may take the sum of the signs of the crossings in the diagram at which K crosses J, as the linking number is symmetric.

Definition 2.10. Let L be an oriented link of n components and Σ a Seifert surface for L. Take a basis $\{[f_i]\}$ for $H_1(\Sigma;\mathbb{Z})$. The orientation of Σ determines a normal direction, which we will think of as being the 'top' of the surface. Now, given any simple oriented curve f on Σ, we can form the *positive push off* of f, denoted f^+, which runs parallel to f and lies just above Σ. The *Seifert matrix* of L is the integer matrix M with $(i,j)^{th}$ entry

$$m_{ij} = \mathrm{lk}(f_i, f_j^+).$$

The Seifert matrix of a knot or link is not an invariant, as it is not unique. We may get different matrices by choosing different basis curves on our surface, or by using a different surface altogether. However, the effect of these changes on the Seifert matrix are well-known ([6], pg 81) and, as with the Seifert surface, we can therefore construct very good invariants from it.

Definition 2.11. The *Alexander polynomial* of a link L with Seifert matrix M coming from a connected Seifert surface is given by the formula

$$\Delta_L(t) = \det(M - tM^T)$$

where M^T denotes the transpose of M. For a link which has a disconnected Seifert surface we define the Alexander polynomial to be zero.

Remark 2.12. Because of the non-uniqueness of the Seifert matrix, the Alexander polynomial is only well-defined up to multiplication by $\pm t^{\pm n}$. However, up to this ambiguity the Alexander polynomial *is* an invariant of the knot ([6], pg 82 - this proof is easily adapted from the Conway polynomial to the Alexander polynomial).

Definition 2.13. The *determinant* of a knot K is $|\Delta_K(-1)|$.

Definition 2.14. For a unit modulus complex number $\omega \neq 1$, we define the *ω-signature* $\sigma_\omega(L)$ of a link L to be the signature of the Hermitian matrix $(1-\omega)M + (1-\overline{\omega})M^T$. The signature of a matrix is the number of positive eigenvalues minus the number of negative eigenvalues. If $\omega = -1$ we call $\sigma_{-1}(L)$ *the* signature of L.

2.2 Braid representations

Definition 2.15. A *braid* consists of m strings travelling directly from left to right, from one vertical bar to another, with the proviso that they

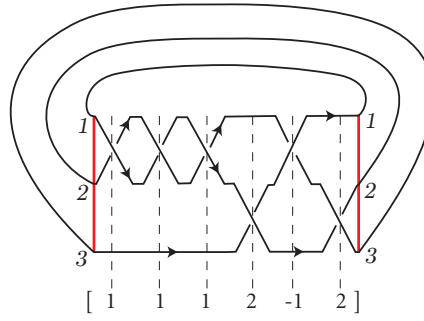


Figure 1: The braid representation of the knot 5_2.

must not loop back on themselves during the journey. If the strings on the right-hand bar are then joined back to the strings on the left-hand bar without making any further crossings, then the resulting object is called a *closed braid*. A braid is described by enumerating the crossings: we write j if strand j crosses over $j+1$, and write $-j$ if strand j goes under $j+1$.

Remark 2.16. 1. The convention of whether j stands for a positive or negative crossing varies exceedingly, so when looking at tables of braid representations it is imperative that the user knows which notation is being used. Sometimes [1 1 1] will be a left-handed trefoil, and sometimes it will be taken to mean a right-handed trefoil. We have chosen our notation because it is consistent with the definition of the linking number, which uses +1 as a positive crossing.

2. An alternative notation is to use letters of the alphabet to denote crossings instead of numbers. Uppercase letters denote positive crossings and lowercase letters denote negative crossings. For example, [1 –2 1 –2] ≡ 'AbAb', [1 1 2 –1 –3 2 –3] ≡ 'AABacBc'.

Definition 2.17. The *braid index* m is the number of strands in the braid. The *braid length* l is the number of crossings in the braid representation; equivalently, the length of the vector (or word) used to describe it.

Remark 2.18. A braid also defines a permutation $\sigma \in S_m$ as follows: $\sigma(i)$ is the position of the i^{th} string when it reaches the right-hand vertical bar. The number of cycles in σ give the number of components n of the link that is defined by the closed braid.

Example 2.19. The knot 5_2 has the braid representation [1 1 1 2 -1 2], as shown in Figure 1. In this case we have $m = 3$, $l = 6$ and $\sigma = (123)$.

The following theorem gives us a justification for only concentrating on braids.

Theorem 2.20 (Alexander, 1923). Every knot (and link) has a closed braid representation.

Proof. A proof will not be given here, but the reader is invited to look at the original paper of Alexander ([1]) or the book of Burde & Zieschang ([3], page 24). ■

The braid representation of a link is by no means unique, but we can try to find representations which minimise the braid index or the braid length. Alexander's original method of producing braids was far from optimal in this respect but a lot of work has been done since then, most notably by Vogel ([11]) who described an algorithm for finding braid representations from knot diagrams which was more efficient than before and also easily programmed into computers. (An implementation may be found in the Braid Programme by Andrew Bartholomew [2].)

A braid has a very naturally-induced Seifert surface. If we apply Seifert's algorithm to the braid diagram we have drawn, we see that each strand turns into a Seifert circle. Thus we can calculate the genus of the Seifert surface induced by the braid representation using Equation (2), which translates as:

$$\text{induced genus} = k - \frac{m - l + n}{2} \tag{3}$$

where k is the number of connected components of the surface, l is the braid length, m is the braid index and n is the number of components of the braid.

However, this induced braid genus is sometimes higher than the actual genus of the knot, even when the braid representation is as 'small' as is possible. For which knots does this happen and why?

Since a braid has a canonical Seifert surface associated with it and this surface has canonical homology generators, it is a natural next step to calculate the Seifert matrix. A method for this is described in the next section.

3 Calculating the Seifert matrix from a braid representation

Our program will be called *Seifert*. The user will input a braid representation into the program, in the form of a numerical vector $x = [x_1\, x_2, \ldots, x_l]$. They may also input an alphabetical version of the braid representation (see Remark 2.16) which the program will translate into the appropriate numerical notation.

To find the Seifert matrix of a braid we first we need to tell the computer how to find a set of homology generators of the corresponding surface (found from Seifert's algorithm). Then we need to find all the ways

in which the generators could interact with each other, and what the corresponding linking numbers are for these situations.

3.1 Finding the homology generators

To be able to find a Seifert matrix for our surface we need to pick a set of homology generators with which to work. There is a fairly obvious choice to make.

Lemma 3.1. Given a braid representation $[x_1, x_2, \ldots, x_l]$, there is a homology generator for the corresponding Seifert surface Σ between each two adjacent crossings on the same strands, i.e. between each x_i and x_j where $|x_i| = |x_j|$ and $|x_k| \neq |x_i|$ for all $i < k < j$. These constitute a basis for $H_1(\Sigma; \mathbb{Z})$.

Proof. Suppose first that Σ is connected. To find a basis for the first homology group of Σ, we must first know the rank of $H_1(\Sigma; \mathbb{Z})$. We know that $\operatorname{rk} H_1(\Sigma_C; \mathbb{Z}) = 2g$, where Σ_C is the corresponding closed surface obtained by capping off the boundary components of Σ with discs, and g the genus of this surface. We also know that $\operatorname{rk} H_1(\Sigma; \mathbb{Z}) = \operatorname{rk} H_1(\Sigma_C; \mathbb{Z}) + n - 1$, where $n \geq 1$ is the number of boundary components of Σ. Combining these pieces of information with Formula (3) (using $k = 1$) gives us

$$\operatorname{rk} H_1(\Sigma; \mathbb{Z}) = 2g + n - 1 = (2 - m + l - n) + n - 1 = 1 - m + l.$$

When $\Sigma = \Sigma_1 \sqcup \cdots \sqcup \Sigma_k$ with the Σ_i connected, then we have

$$\operatorname{rk} H_1(\Sigma; \mathbb{Z}) = \sum_{i=1}^{k} \operatorname{rk} H_1(\Sigma_i; \mathbb{Z}) = k - m + l. \tag{4}$$

It is clear that the homology generators given in the lemma are all linearly independent, so it remains to show that they are a maximal such set.

Suppose that between strands i and $i+1$ there are l_i crossings. Lemma 3.1 then gives us $l_i - 1$ homology generators between those strands. Notice that $l_i = 0$ means that a new connected component of Σ is starting, so there are $k - 1$ instances of $l_i = 0$. In total, therefore, there are

$$\begin{aligned}
\sum_{i=1, l_i \neq 0}^{m-1} (l_i - 1) &= (k - 1) + \sum_{i=1}^{m-1} (l_i - 1) \\
&= (k - 1) + (l_1 + \cdots + l_{m-1}) - (m - 1) \\
&= k + l - m \\
&= \operatorname{rk} H_1(\Sigma; \mathbb{Z})
\end{aligned}$$

generators. These generators are therefore a basis for $\operatorname{rk} H_1(\Sigma; \mathbb{Z})$. ∎

Our convention will be to number the generators in ascending order according to the order in which the first crossing of the generator appears in the braid representation.

Example 3.2. In Figure 2 we can see the homology generators of the braid representative of the knot 5_2. On the left they are shown on the braid diagram of 5_2, and on the right they are drawn onto SeifertView's image of the surface of 5_2.

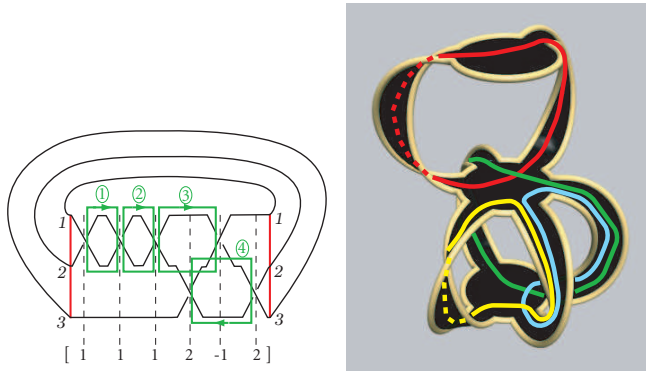


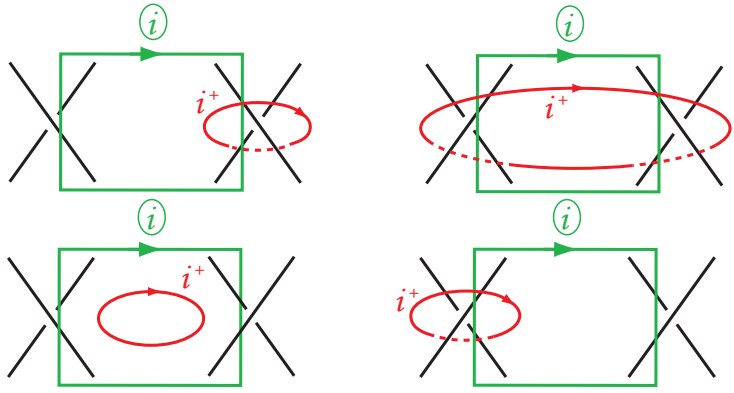
Figure 2: A set of homology generators for the knot 5_2.

The first part of the *Seifert* program implements this idea and puts the information in a vector h. For each $i = 1, \ldots l-1$, the computer checks for the next vector entry with the same absolute value as $|x_i|$, say x_j, and then assigns $h(i) = j$. If there is no such entry to be found then $h(i) = 0$.

The vector h captures all the information about the homology generators we need to be able to calculate their linking numbers.

3.2 Diagonal entries

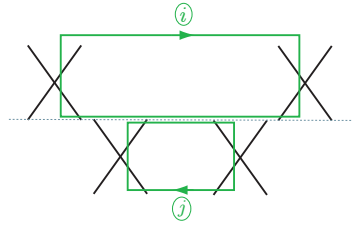
The first task in the Seifert matrix to calculate is how the homology generators interact with themselves. Each generator goes through two crossings. If these crossings are of opposite sign, then the linking number m_{ii} is zero. If the crossings are both right handed (i.e. $x(i)$ and $h(x(i))$ are both positive) then the $m_{ii} = -1$. Similarly, if the crossings are both left handed (so $x(i)$ and $h(x(i))$ are both negative) then $m_{ii} = 1$.



3.3 Non-diagonal entries

The most important part of *Seifert* is a long 'if' loop which tells the computer how to work out the non-diagonal entries of the matrix. There are five cases to consider. Let x_i and x_j be two elements of x, with $i < j$ and $h(i) \neq 0$ (so that there is a homology generator at i).

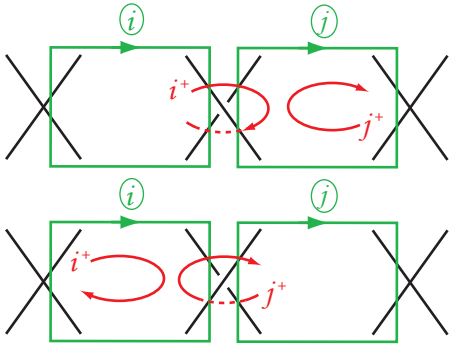
1. If $h(i) > h(j)$ then $m_{ij} = 0 = m_{ji}$. This corresponds to the following situation:



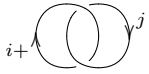
   Note that this also will include the case where $h(j) = 0$ and j has no homology generator, so there is no reason to treat that case specially.

2. If $h(i) < j$ then clearly $m_{ij} = 0 = m_{ji}$ as the two generators cannot interact.

3. If $h(i) = j$ then there are two cases to consider. The first (top picture) is when x_j is a right handed crossing (so $x_j > 0$) and the second (bottom picture) is when x_j is a left handed crossing (so $x_j < 0$).



To calculate m_{ij} we must push j slightly off the surface and see how it interacts with i. In the first case we get two disjoint circles, so $m_{ij} = 0$, and in the second case we have $m_{ij} = -1$ from the diagram 

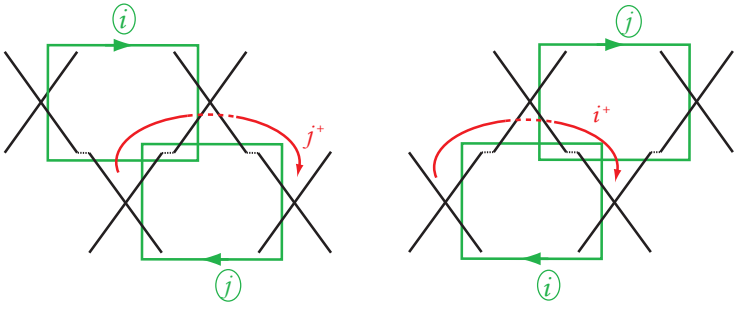
. If instead we push i off the first surface we get $m_{ji} = 1$ from 

and $m_{ji} = 0$ in the second surface.

After these three cases, we are in the situation where $i < j < h(i) < h(j)$.

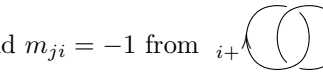
4. If the crossings are separated by more than one strand, then there will clearly be no linking (e.g. if one crossing is between strands 1 and 2, and another is between strands 3 and 4). For the algorithm, we write that if $\big||x(i)| - |x(j)|\big| > 1$ then $m_{ij} = 0 = m_{ji}$.

5. The last situation is where $\big||x(i)| - |x(j)|\big| = 1$, which again splits into two cases:



In the first case (left-hand picture) we have $|x(i)| - |x(j)| = -1$. We find that $m_{ij} = 1$, from the diagram 

, and $m_{ji} = 0$.

In the second case (right-hand picture) we have $|x(i)| - |x(j)| = 1$. We find that $m_{ij} = 0$, and $m_{ji} = -1$ from 

.

These are all the possible cases which could occur in a braid representation of a link, so the algorithm is essentially complete. If $h(i)$ should equal zero, we tell the program to put a row and column of zeros at position i.

3.4 Tidying up

After the computer has executed this part of the program we will have a matrix of dimension one less than the braid length. This is clearly not what we want: we need to delete the rows and columns which have occurred as a result of $h(i)$ being zero.

4 Other things we can make the program do

There are some immediate things we can calculate from the information we have gathered. First of all we can find the genus of the surface from which the Seifert matrix is calculated, using formula (3). It is interesting to see how much higher this is than the actual genus of the knot (which is the minimum genus over all possible Seifert surfaces).

Secondly, we can calculate the Alexander polynomial of the knot, as described in Definition 2.11. This is a good way of checking that the program is working correctly - it is difficult to check whether the matrix is correct as there are many Seifert matrices of the same knot, but the Alexander polynomial is a true invariant and can be looked up in any book. (*Seifert* computes the correct Alexander polynomials for all prime knots up to and including those with 12 crossings.)

We can also calculate the signature of the knot, as described in Definition 2.14. This is an invariant which is able to detect the chirality of the knot (i.e. whether it is its own reflection), a property which the previous two invariants do not have. It is also useful for studying *slice knots*.

5 Questions old and new

A question that was asked in the first section was "Why does the braid representation of a knot sometimes provide us with a non-minimal genus surface?". A tentative answer may be raised to this. It is clear that this algorithm of finding a Seifert matrix from a braid representation will result in a matrix containing only 0's, 1's and -1's. However, if we look at a

surface of minimal genus associated with such a 'degenerate' knot, then its Seifert matrix will contain higher numbers on the leading diagonal. For example, the knot 6_1 is of genus 1 and has Seifert matrix

$$\begin{pmatrix} 1 & 0 \\ 1 & -2 \end{pmatrix}$$

whereas the matrix produced by the program from the braid representation [1 1 2 -1 -3 2 -3] is

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which corresponds to a surface of genus 2. Notice that the sum of the elements on the leading diagonal is the same in both cases - a 4×4 matrix is the smallest possible (even-dimensional) matrix consisting of only $\pm 1's$ and $0's$ with this property.

However, this vein of reasoning does not fully answer the question, as the case of 7_2 shows. Its Seifert matrix has genus 1 and is

$$\begin{pmatrix} -1 & 1 \\ 0 & -3 \end{pmatrix}$$

so we may conjecture that our algorithm will produce a matrix of dimension 4 with all $-1's$ on the leading diagonal. However, applying the program gives us

$$\begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

which has dimension 6. Why is this? If we could identify a pattern in the matrices of 'degenerate' knots then we might be able to apply our knowledge to knots of higher crossing numbers for which the genus and Seifert matrix are unknown, and give a good upper bound (or even exact bound!) on the genus.

A second question would be to devise a program that could tell when two Seifert matrices corresponded to the same knot. We would need to know the effect of Markov moves on the matrix, or to be able to identify when matrices are S-equivalent.

A third question to ask is whether the Seifert matrices generated by *Seifert* uniquely determine the knot, and if not, whether there is a class of links for which this is true.

Appendix: A review of other programs

- *SeifertView* by Jarke van Wijk and Arjeh Cohen (`http://www.win.tue.nl/~vanwijk/seifertview`) is a program designed to visualise Seifert surfaces in 3 dimensions. It is an excellent way of seeing how Seifert's algorithm creates a Seifert surface from a braid, but gives no mathematical information about the surface or the braid.

- The *Braid Programme* by Andrew Bartholomew (`http://www.layer8.co.uk/maths/braids`) is a C++ program designed to calculate invariants, including the Alexander polynomial, for braids and virtual braids. This is a very powerful tool but it does not compute Seifert matrices of braids.

- *seifert* by Morwen Thistlethwaite (`http://www.math.utk.edu/~morwen/seifert`) is a C++ program designed to calculate Seifert matrices of knots, where the knot must be given in its Dowker notation. A fully functional program which makes use of Seifert's algorithm, but has not yet been extended to deal with links. The code is more complicated than the one given in this paper because of the limitations of the Dowker notation, so that the program has to always keep track of what is happening at each crossing of the knot.

- *Seifert's Algorithm, Châtelet Bases and the Alexander Ideals of Classical Knots* by Killian O'Brien (`http://www.maths.ed.ac.uk/~jcollins/killian.pdf`) describes an algorithm to compute Seifert matrices of knots from the Dowker notation. This was implemented in Maple, but once again the code is quite complicated due to the limitations of the notation.

Bibliography

[1] Alexander, J. W. A lemma on systems of knotted curves. *Proceedings of the National Academy of Sciences 9*, 3 (1923), 93–95.

[2] Bartholomew, A. Braid Programme. `http://www.layer8.co.uk/maths/braids/`.

[3] Burde, G., and Zieschang, H. *Knots*. De Gruyter Studies in Mathematics 5. W. de Gruyter, 2003.

[4] Frankl, F., and Pontrjagin, L. Ein Knotensatz mit Anwendung auf die Dimensionstheorie. *Mathematische Annalen 102*, 1 (1930), 785–789.

[5] Gabai, D. Genera of the alternating links. *Duke Mathematical Journal 53*, 3 (1986), 677–681.

[6] Lickorish, W. B. R. *An Introduction to Knot Theory*, vol. 175 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1997.

[7] Moriah, Y. On the free genus of knots. *Proceedings of the American Mathematical Society* (1987), 373–379. http://www.math.technion.ac.il/~ymoriah/papers/Free_genus_Pub.pdf.

[8] Rolfsen, D. Knots and links. *Mathematics Lecture Series,* Publish or Perish Inc. (1976).

[9] Seifert, H. Über das Geschlecht von Knoten. *Mathematische Annalen 110*, 1 (1935), 571–592.

[10] Van Wijk, J. J., and Cohen, A. M. Visualization of Seifert surfaces. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (2006), 485–496.

[11] Vogel, P. Representation of links by braids: A new algorithm. *Commentarii Mathematici Helvetici 65*, 1 (1990), 104–113.

Julia Collins
University of Edinburgh, School of Mathematics
Edinburgh EH9 3FD, UK
Julia.Collins@ed.ac.uk
http://www.maths.ed.ac.uk/~jcollins/