READ ME FOR *HYPERBOLIC LINK STRUCTURES* PACKAGE

DALE KOENIG, ALEX LOWEN, ANASTASIIA TSVIETKOVA

1. Overview

The purpose of this package is to compute the complete hyperbolic structure of a link complement in 3-sphere using a link diagram instead of a triangulation of the complement. The link is given by the user, and can be arbitrary. The program implements the method described in [11,13] by Thislethwaite and Tsvietkova. The initial software was written by Koenig, and has been modified later with his permission.

The method produces the complete hyperbolic structure if a *suitable* link diagram is given. Such diagrams are called *taut*. A diagram is taut if two checkerboard surfaces contain no accidental parabolics, as was noted by [11]. (There is also a condition of incompressibility and boundary incompressibility of these surfaces in [11], but it was noted by Nathan Dunfield later that this condition follows from the lack of accidental parabolics.)

All reduced alternating diagrams of hyperbolic alternating links are known to be a priori suitable, i.e. taut, as was first proved by Thistlethwaite and Tsvietkova [11]. For non-alternating links, it is not known which diagrams are a priori suitable beyond certain families of links. For example, all fully augmented diagrams are known to be suitable by the work of Flint [3].

The suitability of a link diagram can be compared with the suitability of a triangulation in the classical method for finding hyperbolic structure, described by Thurston [12], and implemented in SnapPea [1,16]. There, a suitable triangulation is needed to compute the complete hyperbolic structure, and not every triangulation is suitable a priori. The triangulations that are a priori suitable are only known for some families of links: for example, for two-bridge links due to the work of Sakuma and Weeks, and Futer and Gueritaud [4,10]. The question of a suitable triangulation for a hyperbolic alternating link for the classical method is still open, unlike for the diagrammatic method implemented here. One of the strengths of SnapPea though is that it empirically finds a suitable triangulation rather quickly, modifying the existing triangulation.

The generalizations of the method by Tsvietkova and Thistlethwaite to 3-manifolds beyond link complements in 3-sphere are discussed in the work of Neumann and Tsvietkova, and the work of Kwon, Park, and Tham [6,8]. They are not implemented in this package. The generalization that allows to compute equations for the canonical component of the character variety of a knot complement in 3-sphere is developed in the upcoming preprint by Petersen and Tsvietkova [9]. It similarly does not use any triangulation, and is based on a link diagram.

Date: April 29, 2024.

Note that if one simply needs a tool to compute the complete hyperbolic structure, SnapPy is currently a better developed software package, with wide functionality, beautiful interface, and speedy running times. It is also extensively tested in terms of reliability. Yet the diagrammatic method implemented here, as well as the equations and the complex values produced by it, may be useful in research. For example, using this method, one can often produce general formulae for equations for the complete hyperbolic structure of an infinite family of links with similar diagrams, and use them to explore lengths of arcs, commensurability, exact versions of certain invariants (e.g. hyperbolic volume), or associated number fields (e.g. invariant trace field). This was done by Thislethwaite and Tsvietkova in [11], by Tsvietkova in [13–15], by Neumann and Tsvietkova in [8], by Flint in [3], by Haider in his upcoming PhD thesis [5]. The fact that triangulating the 3-manifold is not necessary is also at times convenient, and at times allows to gather different kind of data than SnapPy about the intrinsic geometry of hyperbolic links. Therefore, it is useful to have a software package that allows some hands-on testing with the diagrammatic method.

Below, we explain how the method was implemented, describe input and output, give an example, and list things to do in the future as the package is still very much a work in progress. Testing and debugging was limited so far to the class of hyperbolic alternating links and their prime alternating diagrams.

We neither prioritized nor carefully investigated computational complexity or practical running times of the package, and often just chose to use functions, modules, and classes that are readily available. For the examples that we tried, the computation was fast. The number of steps required to write down a system of equations for a given diagram is polynomial in the number of crossings. The complexity of the Newton-Raphson method that we use to solve the system is known to be $O(\log n)$ or $O(n)$, depending on the initial value. Empirically, usually about a hundred of iterations for Newton-Raphson method was enough to obtain the correct solution with the needed precision (several decimal places).

2. Implementation

The code for the package is written in Python and uses the Spherogram module from the SnapPy Plink editor [1].

The program takes link diagrams using Dowker-Thistlethwaite (DT) notation [2] or planar diagram (PD) code. From it, it creates a link, by using Link class from SnapPy [1]. In the process, it assigns every region of the link diagram either black and white color, so that the diagram is checkerboard colored. Note that in theory, not every link diagram has such coloring, but, for example, all reduced alternating diagrams do. Then, using the method of Thistlethwaite and Tsvietkova [11], it creates a system of polynomial equations from the given link diagram. It then applies Newton-Raphson method to find a solution to the equations. On every iteration of Newton-Raphson method, the QR decomposition (from Nympy package) is used to make the Jacobian matrix for the system a square matrix, since initially the number of unknowns and the number of equations are not equal. The initial values for the Newton-Raphson method are set so that the hyperbolic polygons that correspond to regions of the link diagram are close to being regular hyperbolic polygons. Empirically, if the diagram is suitable, the Newton-Raphson method then converges to a complex solution, and the solution gives the complete

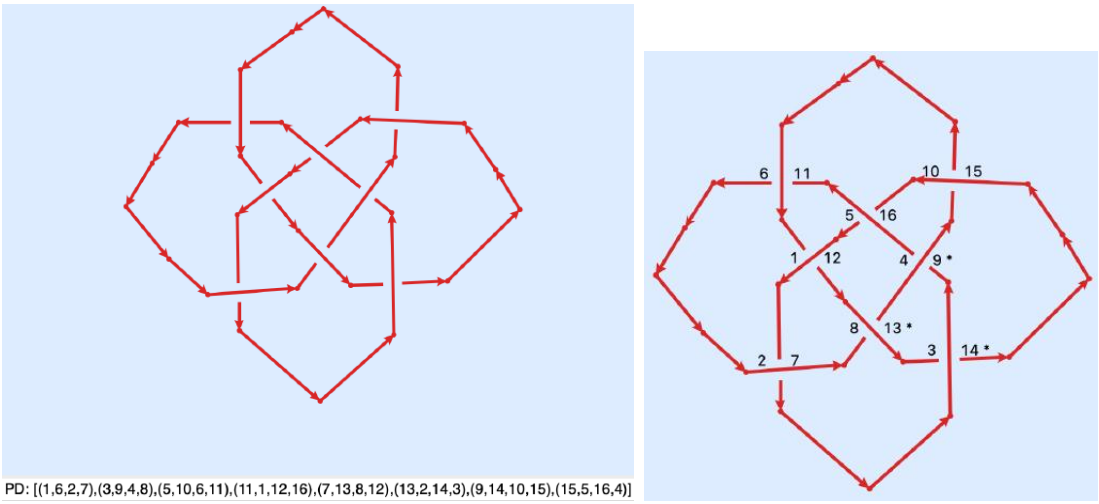

FIGURE 1. Plink Editor from [1] was used to create this figure. Its DT code as given by the editor is (12, 14, 16, 2, 4, 6, 8, 10)

hyperbolic structure on the link complement in 3-sphere. Recall that such structure is unique due to Mostow-Prasad rigidity. The solution consists of complex values called *edge and crossing labels* that give lengths of certain arcs, and values of angles between the arcs in $\mathbb{H}^3$ as explained in [11].

3. Example

Consider Turk's Head knot with a reduced alternating diagram, as in Figure 3.

3.1. **Input and labels.** Recall from [11] that *an edge* is a piece of the strand of a link diagram from one crossing to the next crossing. Each crossing and edge of a given link diagram is assigned a label. The value of the label is a complex number that will be computed by the package. In [11], edge labels are denoted by u_i's, v_i's, crossing labels are denoted by w_i's. The hyp[erbolic structure is determined by the values and the placement of the labels (i.e. which label is assigned to which crossing or edge of the link diagram). As explained in [11], one can then easily compute presentation of Wirtinger generators into $PSL(2, \mathbb{C})$, cusp shape, lengths of various arcs (e.g. meridian) from the labels that correspond to the complete hyperbolic structure. This is not implemented in the package though. To understand the placement of the labels, we need to describe the number labeling that the program gives to the crossings and edges in the code. This depends on the input, i.e. on PD or DT code.

3.2. **Planar Diagram code.** For inputting alternating links, one can use the Planar Diagram (PD) code, as in, for example, [7]. It is only recommended for alternating links since Spherogram module that we use may not accept signs in the input, and traditionally in PD code negative signs correspond to certain crossing in non-alternating link diagrams. Therefore for non-alternating links, go to DT code subsection below.

The Planar Diagram (PD) code for our example is in the bottom of Figure 1, right. There is a total of 8 ordered tuples of 4 integer numbers. The tuples correspond to crossings of the knot diagram. For instance, the first chosen crossing, labeled

0 internally by the code, corresponds to the ordered tuple (1,6,2,7). Overall, the program internally labels the crossings by 0, 1, ..., 7 corresponding to the ordered tuples from left to right.

One can then reconstruct the internal labelling of the edges of the link diagram. In our example, start by labelling an edge that connects crossings 0 and 3 by 1, since label 1 can be found in the first and fourth ordered tuples (corresponding to 0th and 3rd crossings). Similarly, there is an edge labelled 2 from crossing 0 to crossing 5, etc. We can see that crossing 0 will end up surrounded by edges labelled 1, 6, 2, and 7, precisely the first tuple given. The order of the tuples corresponds to traveling around a crossing counterclockwise.

The syntax for the PD code input for the Turk's head is:
$[(1,6,2,7),(3,9,4,8),(5,10,6,11),(11,1,12,16),$
$(7,13,8,12),(13,2,14,3),(9,14,10,15),(15,5,16,4)]$

3.3. **Dowker-Thislethwaite code.** It is possible to use Dowker-Thistlethwaite (DT) code to input any knot or link into the program. Since conventions differ across the literature, below we recall how DT code is obtained. One can also use Plink Editor from [1] to obtain it.

Pick an arbitrary crossing on the link diagram. In Figure 3, the crossing labelled 1 and 12 was picked. Then number the over- and underpasses at every crossing consecutively as we follow the link component. Repeat for other link components, starting with crossing that are already numbered, if there are any.

Now from the pair of labels for every crossing, say (1, 12), consider only the second (largest) label, and record only even such labels. If the even label belongs to an overpass, replace the label with its negative (12 changed to -12, for example). In this example, we start with (1,12), so we record 12. The crossing 12 corresponds to an underpass, so we leave it positive. We then skip (2,7) and look at (3,14). We record the 14 and notice again that the crossing 14 corresponds to an underpass. Then the input for the Turk's Head knot into our program is:

DT: [(12, 14, 16, 2, 4, 6, 8, 10)]

We now describe the internal labeling produced by the program code from DT code input. Starting with the DT code (12, 14, 16, 2, 4, 6, 8, 10), we start with crossing internally labelled 0, which is crossing (1,12). Then move along the link to crossing internally labelled 1, which is crossing (2,7) as seen in the diagram. Continue moving along the link. If a crossing that we encounter already has an internal label, skip it and move to the next one. The last crossing encountered will be the (10,15) crossing, and thus it is the 7th crossing internally.

With this, we can label faces of the link diagram, by listing labels of the crossings that belong to these faces (in arbitrary order). For example, a face internally labelled by [7,3,4] refers to the face with (10,15) DT code at the crossing (the 7th crossing internally), the (4,9) DT code at the crossing (the 3rd crossing internally), and the (5,16) DT code at the crossing (the 4th crossing internally). Similarly the face labelled [7,4,5] internally is the top region in Figure 3, and the face labelled [1,2,5,7] internally is the outer unbounded region of the link diagram.

3.4. **Output: equations.** The output starts by listing out the region and edge equations, defined in [11]. For Turks Head knot, this looks as follows:

$u_1 - v_1 - 1 = 0$
$u_2 - v_2 - 1 = 0$
$u_3 - v_3 - 1 = 0$
$u_1 * u_3 + w_4 = 0$
$u_1 * u_2 - w_7 = 0$
$u_2 * u_3 + w_3 = 0$
$v_{15} * v_5 + w_5 = 0$
$v_1 * v_5 + w_7 = 0$
$v_1 * v_{15} - w_4 = 0$
$u_4 - v_4 - 1 = 0$
$u_5 - v_5 - 1 = 0$
$u_6 - v_6 - 1 = 0$
$u_7 - v_7 - 1 = 0$
$v_{13} * v_2 - w_3 = 0$
$v_{13} * v_4 + w_2 = 0$
$u_4 * u_5 * u_7 - u_5 * w_2 - u_7 * w_7 = 0$
$u_4 * u_5 * u_6 - u_4 * w_5 - u_6 * w_7 = 0$
$u_5 * u_6 * u_7 - u_5 * w_1 - u_7 * w_5 = 0$
$v_2 * v_4 + w_7 = 0$
$u_8 - v_8 - 1 = 0$
$u_9 - v_9 - 1 = 0$
$u_{10} - v_{10} - 1 = 0$
$u_{10} * u_8 + w_0 = 0$
$u_8 * u_9 + w_6 = 0$
$u_{10} * u_9 - w_1 = 0$
$u_{11} - v_{11} - 1 = 0$
$u_{12} - v_{12} - 1 = 0$
$u_{13} - v_{13} - 1 = 0$
$u_{11} * u_{13} - w_2 = 0$
$u_{11} * u_{12} + w_6 = 0$
$u_{12} * u_{13} + w_3 = 0$
$v_7 * v_9 + w_1 = 0$
$v_{11} * v_9 - w_6 = 0$
$v_{11} * v_7 + w_2 = 0$
$v_{12} * v_3 * v_8 - v_3 * w_6 - v_8 * w_3 = 0$
$v_{12} * v_{16} * v_8 - v_{12} * w_0 - v_{16} * w_6 = 0$
$v_{16} * v_3 * v_8 - v_3 * w_0 - v_8 * w_4 = 0$
$u_{14} - v_{14} - 1 = 0$
$u_{15} - v_{15} - 1 = 0$
$u_{16} - v_{16} - 1 = 0$
$u_{14} * u_{16} + w_0 = 0$
$u_{14} * u_{15} - w_5 = 0$
$u_{15} * u_{16} + w_4 = 0$
$v_{10} * v_6 + w_1 = 0$
$v_{14} * v_6 + w_5 = 0$
$v_{10} * v_{14} - w_0 = 0$

There are more equations that are displayed by the program. Note that the program does not use symmetry of the link diagram (which can be used, as shown in [11]) or any other shortcuts, and instead assigns a new label to every crossing, and to every side of an edge of a given diagram. That is why the number of labels and equations it displays is high.

For the Newton-Raphson method that is used to find the geometric solution, the number of iterations is fixed (but easy to change) in the source code.

3.5. **Output: edge and crossing labels.** The output identifies a face that is black. This allows the user to reconstruct the checkerboard coloring of all faces of the link diagram. A face is identified through the internal labels of the face crossings (internal labels are described above). In our example, when the input is given by the DT code, the face with labels $[7, 4, 5]$ for its crossings is black. The program then gives the solution to the equations that it found using Newton-Raphson method. For our example, it gives the following complex values for edge and crossing labels, where $i = \sqrt{-1}$:

$w_0 = (-0.5 - 0.866025i)$
$w_1 = (-0.5 + 0.866025i)$
$w_2 = (-0.5 + 0.866025i)$
$w_3 = (-0.5 - 0.866025i)$
$w_4 = (-0.5 - 0.866025i)$
$w_5 = (-0.5 + 0.866025i)$
$w_6 = (-0.5 - 0.866025i)$
$w_7 = (-0.5 + 0.866025i)$

After this, faces are enumerated and the edges are given numerical labels. As explained in the above subsections, each edge is labeled based on which crossings it is attached to. Each edge also has two labels corresponding to the adjacent faces. In what follows, we describe two faces from the Turk's Head knot diagram. The edge labels u_i correspond to the black sides of the edges, and the edge labels v_i to the white sides.

Face Number: 0
crossing 4 to crossing 7
crossing 7 to crossing 3
crossing 3 to 4
Face Color: Black

Edge from crossing 4 to crossing 7:
u1 = (0.5+0.866025j)
v1 = (-0.5+0.866025j)
Edge from crossing 7 to crossing 3:
u2 = (0.5+0.866025j)
v2 = (-0.5+0.866025j)
Edge from crossing 3 to crossing 4:
u3 = (1+0j)
v3 = 0j

Face Number: 1
crossing 5 to crossing 7
crossing 7 to crossing 4
crossing 4 to crossing 5
Face Color: White

etc.

Things to do

(1) The above description of how edge and crossing labels are placed is not straightforward. Graphic interface would be a plus, with a link diagram with labels on the diagram given as an output.
(2) The software was tested expensively for reduced alternating diagrams of hyperbolic alternating links, but not for other links. Only a few non-alternating examples were run, with mixed results. While not every non-alternating diagram is suitable, some are known to be: e.g. fully augmented diagrams. To start, the program needs to be tested and debugged for fully augmented links. We saw some examples of such links where the complete hyperbolic structure was not produced: most likely, the program does not correctly deal with over and undercrossing and two sides of the same edge of a link diagram in such cases. As noted in [11], either $u_i = v_i \pm 1$ or $u_i = v_i$ for the two labels u_i, v_i corresponding to two sides of the same edge. The link orientation and the information about subsecutive over/undercrossings needs to be tracked carefully here.
(3) The next step would be to implement the algorithm for computing the canonical component of character variety by Petersen and Tsvietkova [9].
(4) The readability of the source code was pushed off in priority. Comments, breakpoints, and functions might need to be updated.
(5) The number of iterations can be changed in the source code. It would be better to build in an error value or an iteration number set by a user.

(6) May be it would be good to change the example described above from a knot to a link. The package works for knots and links.

References

[1] M. Culler, N. M. Dunfield, M. Goerner, and J. R. Weeks, SnapPy, a computer program for studying the geometry and topology of 3-manifolds, http://snappy.computop.org

[2] C. H. Dowker, M. B. Thistlethwaite, Dowker, *Classification of knot projections*, Topology Appl.16 (1983), no.1, 19–31

[3] R. Flint, *Intercusp Geodesics and Cusp Shapes of Fully Augmented Links*, preprint, https://arxiv.org/abs/1811.07397

[4] F. Gueritaud, *On canonical triangulations of once-punctured torus bundles and two-bridge link complements*, with an appendix by D. Futer, Geom. Topol. 10 (2006), 1239–1284.

[5] T. Haider, PhD thesis, Rutgers University - Newark, to be submitted in Fall 2024.

[6] A. Kwon, B. Park, Y. H. Tham, *Generalization of the Thistlethwaite–Tsvietkova method*, preprint, https://arxiv.org/abs/2309.01282

[7] M. Mastin, *Links and Planar Diagram Codes*, J. Knot Theory Ramifications 2 4(2015), no.3, 18 pp.

[8] W. Neumann, A.Tsvietkova, *Intercusp parameters and the invariant trace field*, Proc. of the American Math. Soc. 14 (2016), No. 2, 887–896

[9] K. Petersen, A. Tsvietkova, $PSL(2, C)$*-representations of knot groups from knot diagrams*, preprint

[10] M. Sakuma and J. R. Weeks, *Examples of canonical decomposition of hyperbolic link complements*, Japan. J. Math. (N. S.) 21 (1995), No. 2, 393–439

[11] M. Thistethwaite, A. Tsvietkova, *An alternative approach to hyperbolic structures on link complements*, under review in Algebr. Geom. Topol., ArXiv: math.GT/1108.0510v1.

[12] W. P. Thurston, *The geometry and Topology of Three-Manifolds*, Electronic Version 1.1 (March 2002), http://www.msri.org/publications/books/gt3m/

[13] A. Tsvietkova, *Hyperbolic Structures from Link Diagrams*, Ph.D. thesis, University of Tennessee (2012)

[14] —, *Exact volume of hyperbolic 2-bridge links*, Comm. in Analysis and Geom. 22 (2014), No. 5, 881–896

[15] —, *Determining isotopy classes of crossing arcs in alternating links*, Asian Journal of Mathematics Vol. 22, No. 6 (2018), 1005–1024

[16] J. R. Weeks, *SnapPea*: a computer program for creating and studying hyperbolic 3–manifolds, freely available from http://thames.northnet.org/weeks/index/SnapPea.html