Towards human readability of automated unknottedness proofs

Andrew Fish[1], **Alexei Lisitsa**[2], **Alexei Vernitski**[3],
[1] Centre for Secure, Intelligent and Usable Systems, School of Computing, Engineering & Mathematics, University of Brighton, Brighton UK
[2] Department of Computer Science, University of Liverpool, Liverpool, UK
[3] Department of Mathematical Sciences, University of Essex, Essex, UK
andrew.fish@brighton.ac.uk, a.lisitsa@liverpool.ac.uk, asvern@essex.ac.uk

Abstract

When is a knot actually unknotted? How does one convince a human reader of the correctness of an answer to this question for a given knot diagram? For knots with a small number of crossings, humans can be efficient in spotting a sequence of untangling moves. However, for knot diagrams with hundreds of crossings, computer assistance is necessary. There have been recent developments in algorithms for both (indirectly) (i) detecting unknottedness and (directly) (ii) producing such sequences of untangling moves.

Automated reasoning can be applied to (i) and, to some extent, (ii), but the computer output is not necessarily human-readable.

We report on work in progress towards bridging the gap between the computer output and human readability, via generating human-readable visual proofs of unknottedness.

1 Introduction

We study knot diagrams because this research links algebra, topology and automated reasoning in certain novel ways. The problem of untangling, which is discussed in this paper, is an excellent model problem on which to try to represent computer-generated proofs as human-readable visual proofs.

A knot is a closed curve in the 3-dimensional space; one can conveniently represent knots by their diagrams in 2 dimensions, such as the one in Figure 1. A human can try to decide if a knot is unknotted (that is, is the *trivial knot*, also known as the *unknot*) by taking a rope and checking if it can be untangled without cutting the rope. One can present such an untangling process on paper as a sequence of knot diagrams, see Figure 2, taken from [Fish *et al.*, 2018]. When considering more complicated unknot diagrams, with many crossings, such as the diagram with 141 crossings known as Haken's Gordian knot [Lackenby, 2015], the task of deciding unknottedness, and if so, of finding an untangling sequence, requires computer assistance. Our paper presents progress in presenting computer-generated proofs on unknottedness in a readable form.

For an arbitrary knot diagrams, deciding whether it is the unknot is a decidable problem, as was shown in the early 1960s [Haken, 1961]. It remains unknown whether a polynomial time algorithm for this decision problem exists, but it has been shown that the problem belongs to the complexity class NP ∩ co-NP. Other algorithms have subsequently been proposed and some have been implemented; see, for example, [Dynnikov, 2003; Burton and Ozlen, 2012; Lewin *et al.*, 1996]. Whilst they have limitations, algorithms based on monotone simplifications [Dynnikov, 2003] have been shown to be efficient in detecting the unknot, while algorithms based on normal surface theory [Burton and Ozlen, 2012] have been shown to be efficient in detecting non-trivial knots.

As to proving that a given knot is not trivial (for example, the knot in Figure 1), there are existing methods; these rely on manipulating algebraic constructions which cannot be easily visualized; we do not consider this problem in this paper.

The way a human would manipulate a rope is formalised via three elementary diagram-changing moves, called Reidemeister moves [Reidemeister, 1927], see Figure 3. Whereas it is possible to emulate the human untangling process in the computer[1], in practice the number of Reidemeister moves may be extremely large (see Theorem 1.1 of [Coward and Lackenby, 2014]), and in general the intermediate diagrams can be large and hard to inspect. In the sequence of diagrams in the simple example in Figure 2 each subsequent diagram is less complicated than the previous one; however, note that this sequence of diagrams skips some Reidemeister moves; if we included all diagrams produced by each application of Reidemeister moves, the sequence would be much longer, and some intermediate diagrams would be more complicated than the original diagram.

Recently, generic automated reasoning methods have also been applied and can be seen to be competitive for the unknot detection problem. Instead of applying a specialized algorithm, the problem of unknot detection is reformulated in terms of properties of certain algebraic structures associated with knot diagrams, so that the establishment of these properties can be delegated to automated reasoning tools or computer algebra systems. Thus, automated first-order theorem proving have been used to show triviality of knots [Fish and Lisitsa, 2014], while automated dis-

[1] For example, one convenient algebraic implementation of Reidemeister moves is a construction called a *quandle* [Joyce, 1982].

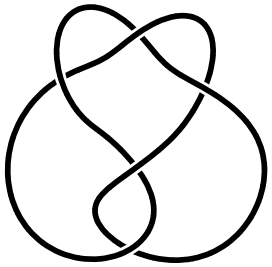


Figure 1: An example of a knot (known as 5_2).

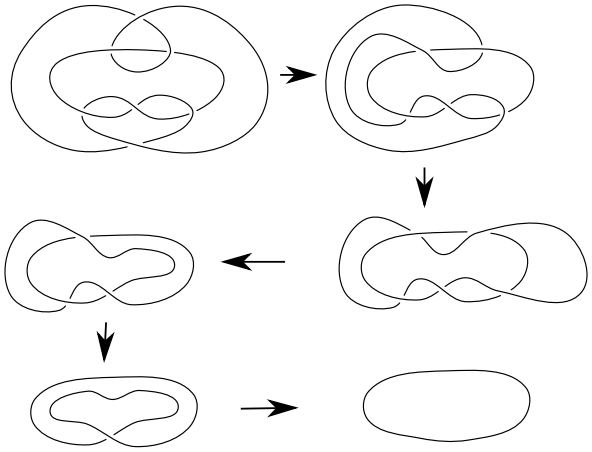


Figure 2: Untangling an unknot.

proving by countermodel building [Fish and Lisitsa, 2014; Lisitsa and Vernitski, 2017] and SAT solving [Fish *et al.*, 2015] have be used to show non-triviality of knots efficiently.

In [Fish *et al.*, 2018] we proposed a notion of "visual algebraic proofs" with the intent of combining visual diagrammatic reasoning with proof automation. We essentially provide a human readable visual proof representation of algebraic deductions that prove unknottedness. Each step of these proofs can be easily understood and checked for correctness by humans.

In this paper we present our progress in implementing the ideas presented in [Fish *et al.*, 2018]. We also reflect upon the difference between the use of a generic automated first-order theorem proving as an automation vehicle, as demonstrated in [Fish *et al.*, 2018], with a newly developed specialized prover implementing the method of [Fish *et al.*, 2018], and with a computer algebra system.

2 Background: Visual algebraic proofs

We give background definitions, intuition and an explanation of the use of labelled tangles; more details can be found in [Fish *et al.*, 2018]. For basic concepts of knot theory (including technical conditions on knots, omitted for the sake of simplifying the narrative), see any of the textbooks [Burde *et al.*, 2013; Livingston, 1993; Kawauchi, 1996; Lickorish, 2012; Manturov, 2004; Murasugi, 2007].

A *tangle* diagram T is like a knot diagram, except that its arcs may have free ends. Our tangle diagrams are *labeled*,

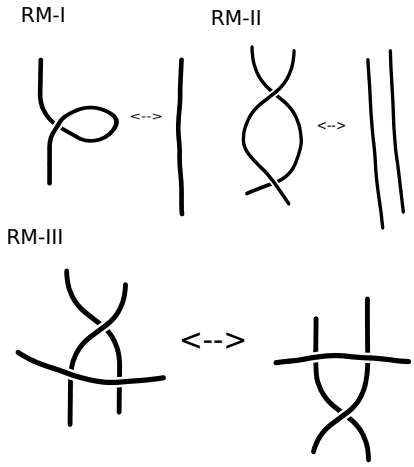


Figure 3: Knot equivalence is captured by the Reidemeister Moves.

that is, each of their arcs is labeled by a letter; from the mathematical point of view, this letter is treated as an element of an algebraic group known as the π-orbifold group of the knot (one could also interpret labels as elements of some other algebraic system, but we chose this interpretation of labels because elements of the π-orbifold group are especially easy to manipulate and to visualize [Lisitsa and Vernitski, 2017; Fish *et al.*, 2018]). A mathematically minded reader might wish to see the definition of the group. For a given knot diagram D, the π-orbifold group OD of the knot is a group generated by the arc letters with the following relations. For each arc x of the diagram D, introduce a relation $x^2 = 1$. At every crossing where x and z are the two arcs terminating at the crossing and y is the arc passing over the crossing, introduce a defining relation $xy = yz$. For more discussion, see [Fish *et al.*, 2018; Lisitsa and Vernitski, 2017].

It is important to note that although a tangle diagram with two free ends x and y is a visual representation of a proof that $x = y$ in OD, we do not need to think of equalities or groups when we inspect the diagram. To ascertain to the proof is correct, we only need to check that labeling of each crossing in the tangle diagram matches exactly with one of the labeled crossings of the original knot diagram.

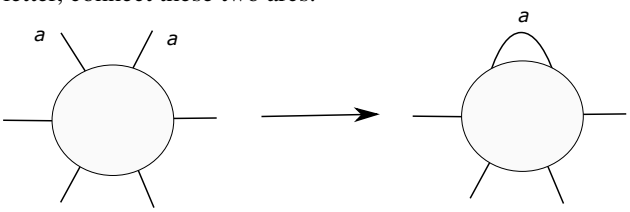
Theorem 1 ([Fish *et al.*, 2018]) *A knot diagram D (with unique labels) represents the unknot if and only if for each pair of its labels a, b there is a labeled tangle diagram T which has exactly 2 free-end arcs labelled a and b, with the property that each crossing in T is labelled in the same way as some crossing in D.*

These labelled tangle diagrams, taken together, constitute a visual proof of the fact that the knot diagram is the unknot. If one wants to look for untangling moves (like in Figure 2), it is important to stress that the labelled tangle diagrams from Theorem 1 are not designed to suggest untangling moves, although they may do so (see an example in Section 5).

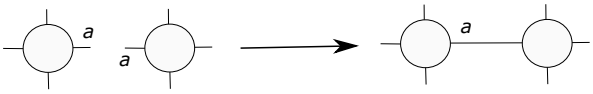
One way of searching for the labelled tangle diagrams needed in Theorem 1 is by starting from tangles corresponding to the original knot diagram's crossings and then combining copies of these diagrams by the application of the follow-

ing two rules:

1. Given a labelled tangle diagram which has, amongst its end arcs, two adjacent end arcs labelled with the same letter, connect these two arcs.



2. Given two labelled tangle diagrams T and U such that both T and U have an end arc labelled with the same letter, connect these two arcs.



3 Automated Theorem proving using generic first-order provers

Whereas automated theorem prover software is not designed specifically for building tangle diagrams, it is possible to use a prover to emulate the process of searching for a tangle diagram with given properties. Then a description of tangle diagrams can be extracted from the output of the prover. The implementation below is a slight modification of the tangle-building process described above.

To a given knot diagram D we can associate a first-order theory T_D in a vocabulary which consists of unary predicate symbol T (the predicate T means that a certain word can be read around a labeled tangle diagram whose crossings are labeled as some crossings in the original knot diagram), binary functional symbol $*$ (the operation $*$ denotes the multiplication in the π-orbifold group of the knot) and constants e and $a_1, \ldots, a_k$ for all of the labels of D (these letters are elements of π-orbifold group of the knot). The axioms of T_D include:

I. Axioms of monoid for $(*, e)$:

- $(x * y) * z = x * (y * z)$ (associativity of multiplication)
- $x * e = e * x = x$ (e is a unit of the monoid)

II.

- $a_i^2 = e$ for all labels a_i (this is a special property of the π-orbifold group of the knot, and it supersedes the definition of inverse elements in an algebraic group)

III. Initial state axioms:

- $T(a_i * a_j * a_k * a_j)$ for all crossings in D, where the over-crossing arc is labelled by a_j and the under-crossing arcs are labelled by a_i and a_k. (That is, each crossing of the original knot diagram is added to the list of labeled tangle diagrams which we use as building blocks for larger tangle diagrams.)

IV. Transition axioms:

- $T(a_i * x) \to T(x * a_i)$ for all arc labels a_i (That is, the list of letters around a tangle diagram can be read starting from any point outside a tangle.)

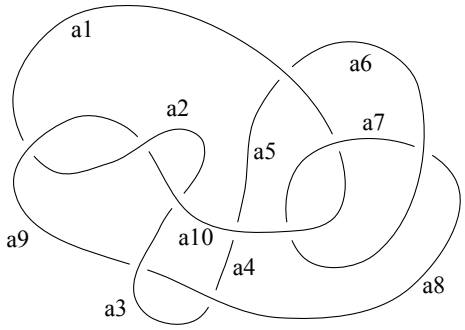


Figure 4: The Culprit unknot.

- $T(x)\ \&\ T(y) \to T(x * y)$. (That is, we can place two tangle diagrams next to each other and treat them as one diagram.)

The terms inside the T predicate are built from constants by the monoid operation, and are meant to represent words read on the ends of arcs around the tangle diagrams. The intended meaning of $T(w)$ for a word w is that a labeled tangle diagram with w read on its free ends which can be built following the rules in Section 2. The initial state axioms declare that the original crossings present us with the initial building blocks of the tangle diagrams. The transition axioms describe operations for building new tangles. From the point of view of the automated theorem proving, Theorem 1 turns into the following result.

Proposition 1 *A knot diagram D with arcs labelled by $a_1, \ldots, a_k$ is a diagram of the unknot if and only if $T_D \vdash \bigwedge_{1 \le i \le k-1} T(a_i * a_{i+1})$, where $\vdash$ denotes first-order logic derivability.*

4 Examples of knot diagrams

The Culprit in Figure 4 is a frequently used example of a diagram of the unknot [Kauffman and Lambropoulou, 2012]. The arcs (that is, the continuous fragments of the curve) of the Culprit diagram in Figure 4 are labelled; introducing a labelling enables algebraic methods and automatic reasoning to work. The interesting property of the Culprit is that if one transforms it by Reidemeister moves in order to untangle it, initially one has to make it more complicated (that, one has to increase the number of crossings in the diagram).

Together with the Culprit, we consider two other examples of unknot diagrams which have the same property. One is the smaller knot diagram in Figure 2. The other is a larger diagram shown in Figure 5, which was used as a central example in [Morton, 1983], and which we call Morton's diagram.

5 Results: Finding untangling diagrams

In our experiments we used three methods of proving unknottedness of the three diagrams. Firstly, we used an automated prover Prover9 as described in Section 3. Secondly, we wrote our own code which explicitly produced all labeled tangle diagrams one after another, using exhaustive search, until the needed tangle diagram is found. Thirdly, we used the GAP

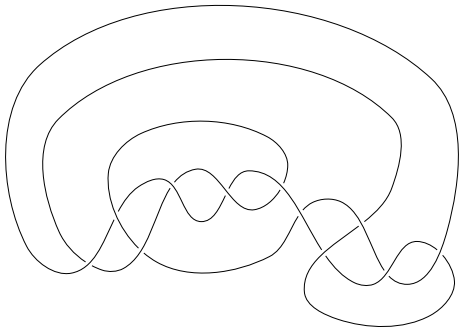


Figure 5: Morton's unknot.

computer algebra software to prove that the knot is the unknot. We compared their output with what we, as humans, would do to untangle the diagrams.

5.1 The Culprit

To prove unknotedness of Culprit by the method of Section 3 we specify the corresponding theory T_D as follows (Prover9 syntax):

```
(x * y) * z = x * (y *z).
x * e = x.
e * x  = x.

a1 * a1 = e.
a2 * a2 = e.
a3 * a3 = e.
a4 * a4 = e.
a5 * a5 = e.
a6 * a6 = e.
a7 * a7 = e.
a8 * a8 = e.
a9 * a9 = e.
a10 * a10 = e.

T(((a6 * a8) * a6) * a7).
T(((a6 * a10) * a7) * a10).
T(((a10 * a7) * a1) * a7).
T(((a1 * a5) * a1) * a6).
T(((a8 * a3) * a8) * a4).
T(((a8 * a3) * a9) * a3).
T(((a10 * a3) * a10) * a2).
T(((a10 * a4) * a10) * a5).
T(((a10 * a2) * a9) * a2).
T(((a2 * a9) * a1) * a9).

T(a1 * x) -> T(x * a1).
T(a2 * x) -> T(x * a2).
T(a3 * x) -> T(x * a3).
T(a4 * x) -> T(x * a4).
T(a5 * x) -> T(x * a5).
T(a6 * x) -> T(x * a6).
T(a7 * x) -> T(x * a7).
T(a8 * x) -> T(x * a8).
T(a9 * x) -> T(x * a9).
T(a10 * x) -> T(x * a10).
T(x) & T(y) -> T(x * y).
```

The goal to prove is

```
T(a1 * a2) & T(a2 * a3) & T(a3 * a4) &
T(a4 * a5) & T(a5 * a6) & T(a6 * a7) &
T(a7 * a8) & T(a8 * a9) & T(a9 * a10).
```

Starting from T_D Prover9 successfully proved the goal and demonstrated that the Culprit is the unknot. The output of Prover9 is difficult to read; as explained in Section 3, Prover9 produces not explicit descriptions of tangle diagrams, but a series of abstract algebraic proofs. We have been able to extract a description of the labeled tangle diagrams manually from Prover9's output for the Culprit.

Our exhaustive-search code was successful at building labelled tangle diagrams proving that the Culprit is the unknot.

Exactly the same first tangle diagram was produced implicitly by our Prover9 code and explicitly by our exhaustive-search code for proving that the Culprit is the unknot. Recall that we expect the computer to find several tangle diagrams: roughly speaking, one for each pair of letters a_i, a_{i+1}; therefore, it was interesting to see that such different approaches as heuristic Prover9 and exhaustive search of tangle diagrams suggested to start in the same way. The tangle diagram claims that the arc $a3$ at the middle of the bottom of the diagram is equal (in a certain algebraic sense, that is, in the π-orbifold group of the knot) to the arc $a6$ in the right-top part of the diagram. Interestingly, this equality corresponds to what a human would do if asked to untangle the Culprit: what seems to be the optimal first simplifying move is concentrating on the vertical strand passing vertically behind the middle of the Culprit all the way from the top to the bottom to the diagram (that is, $a3 - a4 - a5 - a6$) and passing it behind the diagram to one side of it.

To go into more detail, we want to present, as an illustration, a short fragment of the output of Prover9. The first lemma proved by it is presented as follows:

170 T(a10 * (a2 * (a1 * a9))). [hyper(26, a, 44, a, b, 46, a), rewrite([13(15), 13(14), 13(13), 59(12), 66(9)])].

That is, this is step 170 in the proof, and the fact proved is (in algebraic terms) that the product $a_{10}a_2a_1a_9$ is the identity element in the π-orbifold group of the Culprit diagram, or (in terms of diagrams) that it is possible to assemble copies of crossings of the Culprit diagram into a tangle diagram whose free ends are $a10$, $a2$, $a1$, $a9$ (read consecutively around the diagram). Prover9 refers to steps 26, 44 and 46 of the proof. Inspecting them, we see

26 -T(x) | -T(y) | T(x * y).
44 T(a10 * (a2 * (a9 * a2))).
46 T(a2 * (a9 * (a1 * a9))).

That is, line 26 suggests putting two tangle diagrams next to each other, and the two diagrams it refers to are, obviously, the ones named in lines 44 and 46, which are two of the crossings of the Culprit ($a9$ passing under $a2$ becomes $a10$, and $a1$ passing under $a9$ becomes $a2$; you can spot these two crossings in the left-hand side of the labelled diagram in Figure 4).

In this way, reinterpreting Prover9's output as needed one lemma after another, we can construct descriptions of tangle diagrams needed to apply Theorem 1.

In comparison, our exhaustive-search code produces explicit descriptions of labelled tangle diagrams. At the time of writing, the kind of data its output contains can be illustrated by the following example, which is the description of the first untangling diagram for the Culprit (which, as we explained earlier in this section, has two free ends; in the output below, the vertices corresponding to them are denoted by $u0$ and $v0$) and all other vertices are crossings. A drawing of the diagram is presented in Figure 6. The output was:

edge between u0 and u1 labelled a3; edge between u1 and u2 labelled a9; edge between u2 and u3 labelled a2; edge between u2 and u3 labelled a9; edge between u1 and u4 labelled a3; edge between u3 and u4 labelled a10; edge between u3 and u4 labelled a2; edge between u1 and v1 labelled a8; edge between u4 and v2 labelled a10; edge between u2 and v3 labelled a1; edge between v0 and v1 labelled a6; edge between v1 and v2 labelled a6; edge between v2 and v3 labelled a7; edge between v2 and v3 labelled a10; edge between v1 and v3 labelled a7.

5.2 Other diagrams and GAP

As to Morton's diagram, which is somewhat larger than the Culprit, our Prover9 code successfully proved that it is the unknot. However, Prover9's output was long and complicated (the first equality of two arcs is proved in step 23032, compared with step 11566 in Prover9's solution for the Culprit), and we were not able to build labeled tangle diagrams from Prover9's output. Our exhaustive-search code produced labelled tangle diagrams proving that Morton's diagram is the unknot; the tangle diagrams are relatively large, containing up to 40 crossings. As to human reasoning, we find it difficult to suggest a good first untangling move for Morton's diagram. For comparison, in Morton's paper [Morton, 1983] the unknottedness of Morton's diagram is established as an algebraic result using a completely different algebraic group (a braid group), not the same as in our Theorem 1. Morton's untangling can be transformed into an explicit sequence of untangling moves, like in Figure 2, but the sequence is longer (approximately 14 diagrams) and each intermediate knot diagram is larger (10–11 crossings).

As to the knot diagram in Figure 2, which is smaller than the Culprit, both our Prover9 code and our exhaustive-search code easily proved that it is the unknot. The first untangling move suggested by them coincides with what a human would probably do, that is, pass the arc from the right of the diagram to the left of the diagram, as indicated by the first move in Figure 2.

For comparison, the specialized group-theory software GAP [GAP, 2018] (`https://www.gap-system.org/`) proves successfully and very quickly that both the Culprit and Morton's diagram are the unknot (to be precise, GAP proves that the π-orbifold group of the knot is the two-element cyclic group, which is equivalent to the knot being the unknot [Fish *et al.*, 2018; Lisitsa and Vernitski, 2017]). However, GAP's output is just a statement of the fact; it does not include any kind of a proof.

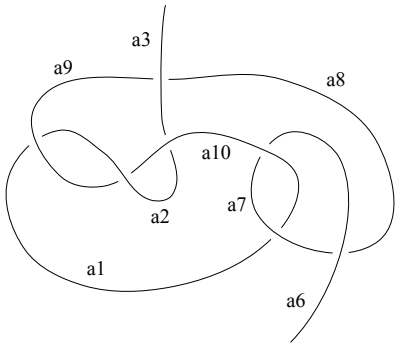


Figure 6: A proof that $a_3 = a_6$ in the π-orbifold group of the Culprit.

6 Results: Visualisation

Theorem 1 shows that the proof of two arcs being equal can always be presented visually as a tangle diagram. However, transforming a computer-generated proof of the equality of two arcs (as in the two examples of computer output in Subsection 5.1) is not straightforward; off-the-shelf tools cannot be immediately applied to draw tangles with labelled arcs.

Let us look in more detail at the tangle diagram whose description was implicitly built in Subsection 5.1. As we have explained, both our Prover9 code and our exhaustive-search code found the same tangle diagram when they started proving that the Culprit is the unknot.

Figure 6 presents this diagram drawn by hand (using the Inkscape software) from the diagram description produced by our exhaustive-search code (in the end of Subsection 5.1). In the future, we would like to see the computer producing such diagrams automatically. For comparision, Figure 7 presents the best approximation to Figure 6 which we could produce by an existing automated drawing tool. The diagram in Figure 7 is produced by KnotPlot software (`http://knotplot.com/`) using a tangle code reconstructed from the diagram in Figure 6; the tangle code used as an input for KnotPlot is "tangle 2r2r@@#.".

As you can see, the available tools for drawing knot-like diagrams automatically have several shortcomings; when it comes to drawing labelled tangle diagrams, they seem to struggle with drawing the free ends of tangle diagrams, and they do not label the arcs. Also, the input code describing the tangle diagrams for drawing them by KnotPlot is not a standard code used in knot theory for describing knot-like diagrams.

7 Conclusion

In our experiments with the variants of automated reasoning applied to unknot detection problem we have observed a clear trade-off between transparency of the proofs (human understandability) and their efficiency. For the following methods we have their transparency increasing, while their efficiency decreasing when we proceed down the list:

- First-order theorem proving for involutory quandles (keis) using generic automated prover [Fish and Lisitsa, 2014];

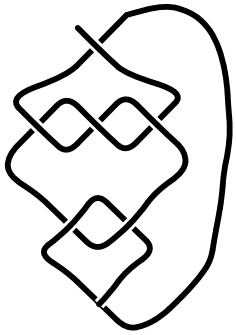


Figure 7: An attempt to draw the diagram automatically.

- First-order theorem proving for π-orbifold groups using generic automated prover [Lisitsa and Vernitski, 2017];
- Visual algebraic proving (tangles building) using generic automated prover [Fish *et al.*, 2018];
- Visual algebraic proving (tangles building) using specialised prover (this paper).

As an alternative approach computer algebra system GAP can be used to solve the group-theoretical problem described in our previous paper; that is, it solves the word problem (for each two-letter word) in the π-orbifold group of the knot diagram. Of course, the word problem for groups is unsolvable in general. As to finding untangling moves, GAP does not provide any output of the sort. As to GAP, it would be very interesting to make large scale comparisons of its efficiency for detecting unknots with an automated theorem proving approach.

Two aspects of the future work are: Automatically producing descriptions (for example, as Gauss codes) of labelled tangle diagrams; Automatically producing drawings of labelled tangle diagrams from their descriptions. For each of these two aspects, we (and other people) have made some progress, but more work is needed. There is plenty of scope for exploration of human produced untangling moves versus the visual algebraic proofs constructed.

References

[Burde *et al.*, 2013] Gerhard Burde, Michael Heusener, and Heiner Zieschang. *Knots*. De Gruyter, 2013.

[Burton and Ozlen, 2012] Benjamin A. Burton and Melih Ozlen. A fast branching algorithm for unknot recognition with experimental polynomial-time behaviour. *CoRR*, abs/1211.1079, 2012.

[Coward and Lackenby, 2014] Alexander Coward and Marc Lackenby. An upper bound on reidemeister moves. *American Journal of Mathematics*, 136(4):1023–1066, 2014.

[Dynnikov, 2003] I.A. Dynnikov. Recognition algorithms in knot theory. *Russian Mathematical Surveys*, 58(6):1093–1139, 2003. cited By 4.

[Fish and Lisitsa, 2014] Andrew Fish and Alexei Lisitsa. Detecting unknots via equational reasoning, i: Exploration. In *International Conference on Intelligent Computer Mathematics*, pages 76–91. Springer, 2014.

[Fish *et al.*, 2015] Andrew Fish, Alexei Lisitsa, and David Stanovský. A combinatorial approach to knot recognition. In Ross Horne, editor, *Embracing Global Computing in Emerging Economies: First Workshop, EGC 2015, Almaty, Kazakhstan, February 26-28, 2015. Proceedings*, pages 64–78. Springer International Publishing, 2015.

[Fish *et al.*, 2018] Andrew Fish, Alexei Lisitsa, and Alexei Vernitski. Visual algebraic proofs for unknot detection. In *Proceedings of the Diagrams Conference*. Springer, 2018.

[GAP, 2018] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.8.10*, 2018.

[Haken, 1961] Wolfgang Haken. Theorie der normalflchen: Ein isotopiekriterium fr den kreisknoten. *Acta Math.*, 105(3-4):245–375, 1961.

[Joyce, 1982] David Joyce. A classifying invariant of knots, the knot quandle. *Journal of Pure and Applied Algebra*, 23(1):37 – 65, 1982.

[Kauffman and Lambropoulou, 2012] Louis H Kauffman and Sofia Lambropoulou. Hard unknots and collapsing tangles. In *Introductory Lectures On Knot Theory: Selected Lectures Presented at the Advanced School and Conference on Knot Theory and Its Applications to Physics and Biology*, pages 187–247. World Scientific, 2012.

[Kawauchi, 1996] Akio Kawauchi. *A survey of knot theory*. Birkhäuser, 1996.

[Lackenby, 2015] Marc Lackenby. Upper bound on reidemeister moves. *Annals of Mathematics*, 182:1–74, 2015.

[Lewin *et al.*, 1996] D. Lewin, O. Gan, and Bruckstein A. M. Trivial or knot: A software tools and algorithms for knot simplification, cis report no 9605. , Technion, IIT, Haifa, 1996.

[Lickorish, 2012] WB Raymond Lickorish. *An introduction to knot theory*, volume 175. Springer Science & Business Media, 2012.

[Lisitsa and Vernitski, 2017] Alexei Lisitsa and Alexei Vernitski. Automated reasoning for knot semigroups and π-orbifold groups of knots. In *International Conference on Mathematical Aspects of Computer and Information Sciences*, pages 3–18. Springer, 2017.

[Livingston, 1993] Charles Livingston. *Knot theory*, volume 24. Cambridge University Press, 1993.

[Manturov, 2004] Vassily Manturov. *Knot theory*. CRC press, 2004.

[Morton, 1983] Hugh R Morton. An irreducible 4-string braid with unknotted closure. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 93, pages 259–261. Cambridge University Press, 1983.

[Murasugi, 2007] Kunio Murasugi. *Knot theory and its applications*. Springer Science & Business Media, 2007.

[Reidemeister, 1927] Kurt Reidemeister. Elementare Begründung der Knotentheorie. *Abh. Math. Sem. Univ. Hamburg*, 5(1):24–32, 1927.