THE FIBONACCI SEQUENCE AND THE TIME COMPLEXITY OF GENERATING THE CONWAY POLYNOMIAL AND RELATED TOPOLOGICAL INVARIANTS

Phillip G. Bradford
University of Kansas, Lawrence, KS 66045
(Submitted July 1988)

1. Introduction

In this discussion, only tame embeddings of S^1 in S^3 that are oriented will be considered. All knots will be represented as regular projections and any projection is assumed to be regular. The reader is expected to know "big O" notation; see [2] for example. Some knowledge of NP-Completeness is also useful.

The algorithm analysis of the complexities is relative to the number of crossings of a given knot projection. Also, the analytical creation of the Conway polynomial is in the Class P. This is shown by the presentation of a well-known algorithm used for computing the Alexander polynomial which can be easily suited to generate the Conway polynomial in better than $O(n^3)$ time.

The proof of the Conway algorithm having exponential worst case time complexity is based on showing the existence of n crossing knot projections. Given these particular projections, the Conway algorithm may perform $O(((1 + \sqrt{5})/2)^n)$ operations on the various knot projections which the algorithm derives in order to calculate the Conway polynomial.

Definition 1.1: The *crossing number* of a knot K is the minimum number of crossings for any regular projection of the knot K.

Definition 1.2: A *split link* (see [10], [11]) $L \subseteq S^3$ is a link $L = L_1 \cup L_2$ where L_1 and L_2 are nonempty sublinks and there exist two open balls, B_1 and B_2 in S^3 such that $B_1 \cap B_2 = \emptyset$; and $L_1 \subseteq B_1$ and $L_2 \subseteq B_2$.

Definition 1.3: A *tangle* (see [4]) is a portion of a knot diagram from which there emerge only four arcs from the four "directions" NE, NW, SE, and SW, and possibly some crossings inside.

Examples:

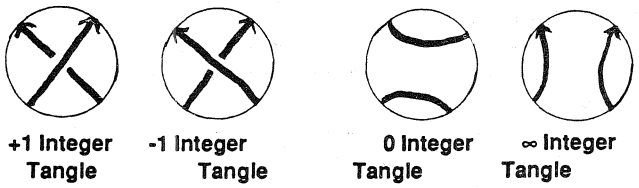

FIGURE 1

A tangle cannot have any knot arcs passing under or over it. The integer tangles pictured above, denoted by the integers +1 and -1 describe all crossings of any knot or link. The following three operations are the elementary knot crossing operations.

Definitions 1.4:

1. *Smoothing*—This operation takes an oriented +1 or -1 integer tangle and replaces it with an oriented ∞ integer tangle.

2. *Changing*—This operation takes an oriented +1 or -1 integer tangle and replaces it with an oriented -1 or +1 integer tangle, respectively.
3. *Deleting*—This operation takes a +1 or -1 integer tangle and replaces it with a 0 tangle.

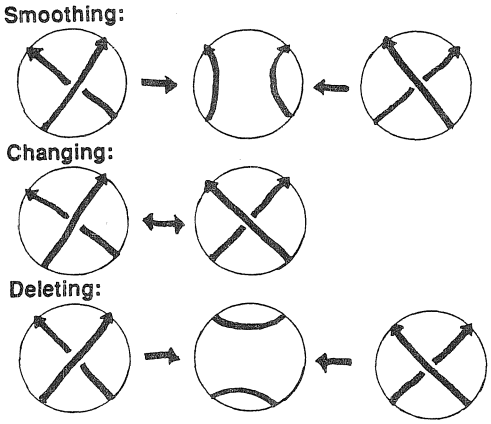

FIGURE 2

Fundamentally, the Conway algorithm is based on the first two knot operations. The operations of changing and smoothing crossings will be focused on, although smoothing and deleting [14] and changing and deleting will also be discussed.

Conceptually, there can be a *tree of projections* built during the application of elementary crossing operations. This tree of projections will become the basis of the complexity analysis.

A knot's tree of projections relative to changing and smoothing is built as follows:

If the knot K is the unknot, then the tree of projections is the trivial tree with an unknot projection as the root with no children.

Given a knot K that is not the unknot, the tree of projections is the binary tree with a projection of K as the root. Choose a crossing, call it X, change the knot projection K at the crossing X to produce the knot projection L. Take the knot projection K, and the crossing X. Smooth it to produce the link projection R. The right child of the root projection K is the smoothed knot projection R and the left child of the projection is the changed projection L. This process of smoothing and changing of knot projections and nonsplit link projections is recursively continued always placing changed knot or link projections as left subtrees and smoothed knot or link projections become right subtrees. When a subtree becomes the unknot or a split link it is a leaf having no more children. It may be observed that by changing a crossing one can reach either the unknot or a split link. A link's tree of projections is constructed similarly.

Examples:

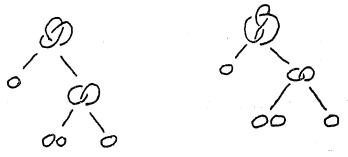

FIGURE 3

Notice that the trefoil and the figure eight knots have identically structured trees of projections. Any combinations of the three elementary operations consistently applied forms a tree of projections in a similar fashion. A tree of projections is assumed to be relative to changing and smoothing unless otherwise noted.

To quote Louis Kauffmann [10], a convenient way to get an unknot from a knot is to perform the following operations on the knot diagram [10, p. 79]:

> Choose a point p on the diagram and draw knot so that you first draw an over-crossing line at the first encounter with a crossing, under cross at the second encounter, and continue until you return to p.

Performing this to a knot projection and then applying Reidemeister moves can produce the unknot in familiar form.

Definition 1.5: An unknot projection developed in this fashion is designated a *descending* knot projection [6], [7]. A descending knot projection's mirror image is, of course, an *ascending* knot projection.

Just as there is a rather straightforward algorithm for creating a descending knot projection there is also a straightforward algorithm for detecting an ascending knot projection.

Definitions 1.6:

1. The *unknotting number* of a knot is the minimum number of changes that must be performed to produce the unknot starting from any knot projection. The unknotting number of a knot K is denoted by $u(K)$.
2. The *unsmoothing number* of a knot is the minimum number of smooths that must be performed to produce the unknot or split link from any knot projection. The unsmoothing number of a knot K is denoted by $us(K)$.
3. The *deleting number* of a knot is the minimum number of deletes that must be performed to the crossings to produce a split link or an unknot from any knot projection. The deleting number of a knot K is denoted $del(K)$.

In 1970, J. H. Conway defined a polynomial, $\nabla_K(z)$, with integer coefficients for oriented knots and links. The polynomial can be recursively calculated from a regular projection of a knot or link K by consistently applying the knot crossing operations of changing and smoothing.

Theorem 1.7: Every knot K has a well-defined Conway polynomial.

This is well known and a proof can be found in [11].

Surprisingly, the Conway polynomial is well defined independent of the particular sequence of smoothings and changings used to calculate it. One of the most important facts about Conway's algorithm is that it can be applied to any regular projection of a knot K and it will produce the same polynomial.

2. The Algorithm

Algorithm 2.1: (Conway's algorithm, see [4], [10], and [11])

Given: a projection of a knot K
Returning: the Conway polynomial of the knot K

1. Choose an orientation of the knot projection K.
2. If (K is the unknot), then:

   Conway polynomial of the unknot is 1: $\nabla_0 = 1$

3. If (K is a split link), then:

   Conway polynomial of the split link is 0: $\nabla_K = 0$

4. Otherwise, when K is not the unknot and not a split link, the tree of projections is built according to the following formula:

$$\nabla_K - \nabla_L = z\nabla_M$$

for K, L, and M knot and/or link projections identical in all respects except that they differ at one crossing in the following manner:

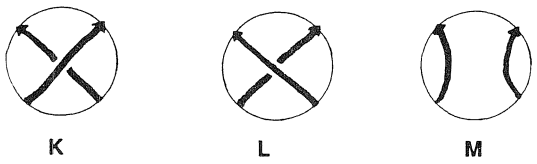

FIGURE 4

One must be sure to preserve the orientation and keep track of the multiple of the indeterminant on the right smoothing edge and the constant multiples on the left changing edges.

5. Recursively repeat steps 2 through 5 with the appropriate smoothed and changed knot and/or link projections until the entire tree of projections is built.
6. Return polynomial ∇_K.
7. End. □

Applying Algorithm 2.1 to a knot projection methodically generates its tree of projections by performing changes and smoothings to the knot projection until there are only projections of unknots or split links left. These changes and smoothings must be performed by adhering to the following formula,

$$\nabla_K - \nabla_L = z\nabla_M$$

where K, L, and M are identical knot projections in every respect except for the following significant differences at only one crossing:

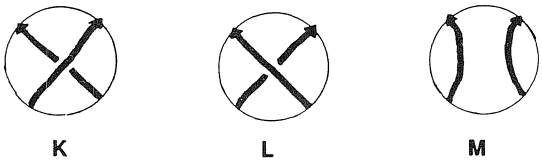

FIGURE 5

Using this relation recursively starting with any knot projection K, the projection eventually becomes resolved into either unknots or split links and, therefore, terminates. Notice that during step 4 of Algorithm 2.1 two new knots are created by changing and smoothing. These new knots may have many possible different projections, but any regular projection will suffice. The Conway polynomial of the unknot is defined to be one ($\nabla_0 = 1$) and any split link is defined to be zero ($\nabla_{\text{split link}} = 0$). The Conway polynomial of both the Trefoil and Figure Eight knots is $z + 1$.

The Conway algorithm terminates when all subtrees have become either unknots or split links.

The unknotting number is a lower bound of the number of crossing changes we must perform in order to construct the unknot from a given knot projection.

Similarly, the unsmoothing number is a lower bound. The next logical question might be: "Given any particular projection of a knot K, how fast can Conway's algorithm resolve the projection of the knot K into unknots and split links?"

Lemma 2.2: At most, $\lfloor (n/2) \rfloor$ is the unknotting number $u(K)$ of any n crossing knot or link K.

Proof: First, taking the case of an alternating and oriented knot or link projection K of n crossings, orient the knot or link and then prepare to traverse it. If the traversal begins at an overcrossing, then set out to build a descending knot or link projection; otherwise, construct an ascending knot or link projection.

Without loss of generality, assume that the descending variety will be constructed. Starting traversal at an overcrossing of the knot or link projection every time a crossing is encountered, if it is the first encounter with the crossing, ensure that it is an overcrossing. If it is the first encounter with a crossing, being passed under, change it to an overcrossing. Otherwise, if it is the second encounter with this crossing, then proceed under the crossing. This will construct the descending knot or link in at most

$$\lceil (n/2) \rceil - 1 \leq \lfloor (n/2) \rfloor$$

changes because it was assumed that the traversal started on an overcrossing.

Now we must contend with the nonalternating knot or link projection. Given a knot or link K with n crossings, if K has more overcrossings than undercrossings, then construct the descending knot projection in less than or equal to $\lfloor (n/2) \rfloor$ crossing changes; otherwise, construct the ascending knot projection similarly. □

Lemma 2.3: At most, $\lfloor (n/2) \rfloor$ is the deleting number $del(K)$ of any n crossing knot or link K.

The proof is similar to the proof of Lemma 2.2, and is therefore omitted.

Lemma 2.4: Given an n crossing knot or link L, at most $n - 1$ is the unsmoothing number $us(K)$.

Proof: Given a projection of the knot K with n crossings, if all n crossings are smoothed, then there will be only unknots and unlinks remaining. After $n - 1$ smooths, the knot K will have, at most, only one crossing left, however many associated unlinks are left. The knot with the one remaining crossing must be the unknot. □

Theorem 2.5: At most, given an n crossing knot, the tree of projections relative to changing and smoothing, and smoothing and deleting can have

$$O\{((1 + \sqrt{5})/2)^n\} \text{ projection nodes.}$$

[Note that $(1 + \sqrt{5})/2 = \phi$, the "golden" ratio.]

Proof: The tree of projections relative to changing and smoothing follows directly from Lemma 2.2 and Lemma 2.4. Given a knot projection K, build a tree of projections that can be described by the recurrence relation

$$f(n) = f(n - 1) + f(n - 2) + 1$$

where $f(p)$ is the number of nodes in any tree of projections of a given knot projection with n crossings and, of course, $f(0) = 1$, $f(1) = 1$, and $f(2) = 3$.

The tree of projections relative to smoothing and deleting follows identically from Lemmas 2.2 and 2.3. □

Theorem 2.5 supplies an upper bound on Conway's algorithm (Algorithm 2.1). This is due to the relation

(1) $\nabla_K = \nabla_L + z\nabla_M$

upon which Conway based his algorithm. Notice the similarity between (1) and the recurrence relation that defines the Fibonacci sequence.

An implementation of Conway's algorithm may not be bounded strictly by the upper bound established by Theorem 2.5, but any implementation of Conway's algorithm can be made to adhere to this upper bound.

A tree of projections relative to changing and deleting has an upper bound of $O(2^{(n/2)})$ given an n crossing knot. Additionally, a tree of projections relative to smoothing and smoothing, or changing and changing, or deleting and deleting all have exponential upper bounds. In summary, we have the following theorem.

Theorem 2.6: At most, the tree of projections relative to any consistently applied elementary knot operation has as an upper bound an exponential number of nodes.

In Theorems 2.5 and 2.6 we have not yet established the existence of any classes of knots which actually adhere to these bounds, i.e., How tight are these bounds?

3. The Complexity

A particular class of knots with specific projections illustrates that there exist knot projections whose trees of projection relative to smoothing and changing, and smoothing and deleting actually satisfy the upper bound of

$$O\{((1 + \sqrt{5})/2)^n\}.$$

The class of knots is the $(2, n)$ torus knots and links and the Fibonacci knots [14] F_n = 1111...1 (n 1's in Conway's notation, see [4]). The $(2, n)$ knots and links will all be assumed to have the projections and orientations described below.

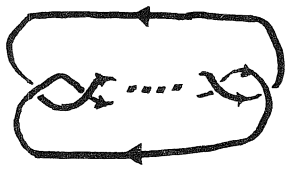

FIGURE 6

Standard projection of the $(2, n)$ knots and links

Throughout the rest of this discussion, the $(2, n)$ torus knots or links will denote the standard projections of the $(2, n)$ knot or link. Also, given any standard projection of a $(2, n)$ knot or link, smoothing and changing will produce standard projection knots or links. This is being done because standard projections can be maintained throughout the execution of Conway's algorithm. The Fibonacci Knot Class is defined by Turner in [14], which we follow. Just as in [14] a knot from the Fibonacci Knot Class will be denoted as F_n. In Conway's notation [4], F_n = 111...1 (n ones). This turns out to be useful in the worst case analysis. Several lemmas are now presented without proof, since they can easily be deduced via examination.

Lemma 3.1: Smoothing the specified projection of a torus knot $(2, n)$, for $m \geq 1$ and $n = 2m + 1$, can produce the link projection $(2, n - 1)$. Additionally, smoothing the link projection $(2, n)$, for $n = 2m$ and $m \geq 1$, can produce a torus knot projection $(2, n - 1)$. We also point out that changing a torus knot

projection $(2, n)$, where $n = 2m + 1$ and $m \geq 1$, can produce the torus knot projection $(2, n - 2)$. And changing the link projection $(2, n)$, for $n = 2$ and $m \geq 1$, can produce the link projection $(2, n - 2)$.

Lemma 3.1 can be verified by examination of parts of the actual torus knot projections.

Lemma 3.2: (Turner [14]) Given a Fibonacci knot F_n deleting and smoothing a crossing can create the following projections F_{n-1} and F_{n-2}.

An n-trefoil is n trefoils connected in the following fashion:

FIGURE 7

Lemma 3.3: Given an n-trefoil, any sequence of elementary knot operations forms an exponential tree of projections.

Lemma 3.3 can be proved by using induction.

Definition 3.4: An *AVL tree* [1], [2] is a binary tree which has the property that from any given node in the tree the depths of the right and left subtrees differ by at most one.

Examples:

FIGURE 8

For brevity, from here the discussion is focused on the elementary operations of changing and smoothing.

The following induction establishes that applying Algorithm 2.1 to a $(2, n)$ knot projection can produce an AVL tree. The Conway algorithm, given a standard projection of a $(2, n)$ knot or link may maintain standard projections throughout smoothing and changing. It is easily proven that any AVL tree has exponential number of nodes, but an exact result will be found.

Theorem 3.5: A torus knot standard projection $(2, n)$, for $n = 2m + 1$ and $m \geq 1$, will produce a tree of projection with $\Theta\{((1 + \sqrt{5})/2)^n\}$ projection nodes provided standard projections are maintained throughout the computation of the Conway algorithm. The same is true for a $(2, n)$ link projection.

Proof: By Lemma 3.1, standard projections can be maintained throughout the construction of a tree of projections of a $(2, n)$ knot or link projection. Standard projections will be maintained throughout this proof.

Claim: When Conway's algorithm is applied to a $(2, n)$ knot or link in standard projection, it produces an AVL tree with the number of nodes described by the recurrence relation

$$f(n) = f(n - 1) + f(n - 2) + 1.$$

Proof by induction:

Basis. The unknot $(2, 1)$ produces a trivial AVL tree. The link $(2, 2)$ produces an AVL tree with 3 knot projections or nodes. The trefoil $(2, 3)$ produces a tree with 5 nodes.

Inductive hypothesis. Assume that for some number q and some function f the standard knot projection $(2, m)$ has an AVL tree of projections with $f(m)$ nodes as long as $m < q$. Additionally, the standard link projection $(2, m - 1)$ has an AVL tree of projections with $f(m - 1)$ nodes.

Inductive step. By the inductive hypothesis, the link or knot projection $(2, q - 1)$ has an AVL tree of projections of size $f(q - 1)$. The knot or link $(2, q - 2)$ has a tree of projections with $f(q - 2)$ nodes in its AVL tree. Now, by adding the +1 integer tangle appropriately to the link or knot projection $(2, q - 1)$, the knot or link projection $(2, q)$ may be produced. Performing a change to the standard projection of the knot or link $(2, q)$ can produce the knot or link projection $(2, q - 2)$. Smoothing a crossing of the knot or link $(2, q)$ may produce the link or knot $(2, q - 1)$ projection. Making the projection of the knot or link $(2, q)$ the root of the tree of projections with the subtrees created by the knot projections $(2, q - 1)$ and $(2, q - 2)$ can make a tree with a total of $f(q - 1) + f(q - 2) + 1$ projection nodes. This is true because the tree of projections of the link or knot $(2, q - 1)$ is an AVL tree, and the tree of projections of the knot or link $(2, q - 2)$ is also an AVL tree by the inductive hypothesis. The link or knot projection $(2, q - 1)$ has smoothing and changing subtrees consisting of the trees of projection $(2, q - 2)$ and $(2, q - 3)$, respectively. Since $\text{depth}((2, q - 2)) - \text{depth}((2, q - 3)) \leq 1$ where $\text{depth}(K)$ is the depth of K's tree of projections, then $\text{depth}((2, q - 1)) - \text{depth}((2, q - 2)) \leq 1$. Therefore, the knot or link projection $(2, q)$ can have an AVL tree of projections with number of nodes described by the recurrence relation

$$f(q) = f(q - 1) + f(q - 2) + 1.$$

This means that applying Conway's algorithm to the standard projection of $(2, n)$ and preserving standard projections throughout the calculation of the Conway polynomial will produce an AVL tree of projections with the number of nodes described by the recurrence relation

$$f(n) = f(n - 1) + f(n - 2) + 1.$$

End of induction.

Next, the exact number of nodes, in closed form, in this tree of projections can easily be derived by solving the nonhomogeneous, constant coefficient difference relation of second order:

$$f(n) - f(n - 1) - f(n - 2) = 1.$$

Given $f(n) - f(n - 1) - f(n - 2) = 1$, where $n \geq 2$, with the boundary conditions $f(0) = 1$, $f(1) = 1$, and $f(2) = 3$.

The closed form solution is easily derived using the method of variation of constants:

$$f(n) = \frac{1 + \sqrt{5}}{2\sqrt{5}}\left(\frac{1 + \sqrt{5}}{2}\right)^n + \frac{-1 + \sqrt{5}}{2\sqrt{5}}\left(\frac{1 - \sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1 - \sqrt{5}}{2}\right)^n \sum_{i=0}^{n-1}\left(\frac{2}{1 - \sqrt{5}}\right)^{i+1}$$

$$+ \frac{1}{\sqrt{5}}\left(\frac{1 + \sqrt{5}}{2}\right)^n \sum_{i=0}^{n-1}\left(\frac{2}{1 + \sqrt{5}}\right)^{i+1}$$

Taking the identity

$$(*) \qquad \sum_{i=1}^{n-2}\left(\frac{2}{1 + \sqrt{5}}\right)^{i+1} = \frac{1 - \sqrt{5}}{1 + \sqrt{5}}\left[\left(\frac{2}{1 + \sqrt{5}}\right)^{n-1} - \frac{2}{1 + \sqrt{5}}\right]$$

which converges to 0 as n approaches ∞, noting $(1 - \sqrt{5})/2 < 1$, $2/(1 + \sqrt{5}) < 1$, and, for some integer k and some constant A when $n \geq k$,

$$f(n) \le A\left(\frac{1+\sqrt{5}}{2}\right)^n,$$

we obtain

$$f(n) = O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right).$$

Now, to show $((1+\sqrt{5})/2) = O(f(n))$: Given equation (*) and the fact that $(1-\sqrt{5})/2 < 1$, then choosing a large enough constant A and some number k when $n \ge k$,

$$((1+\sqrt{5})/2)^n \le A((1+\sqrt{5})/2)^{n+1} \text{ as } n \to \infty.$$

Hence,

$$f(n) = \Theta\{((1+\sqrt{5})/2)^n\}.$$

This completes the proof. □

Since the tree of projections for a $(2, n)$ knot projection whose children remain as standard projections has depth of at least $\lfloor (n/2) \rfloor$, it is quite easy to show $f(n) = O(2^{\lfloor (n/2) \rfloor})$, alternatively.

Noting Theorem 2.5 and Theorem 3.5 and given any n crossing knot K, call its tree of projections relative to smoothing and changing PK. Then the standard projection of the $(2, n)$ knot can have a tree of projections $P(2, n)$ that is larger than or equal to PK.

A similar argument, along with Theorem 2.6, illustrates that the Fibonacci Knot Class forms an upper bound on the size of a tree of projections relative to smoothing and deleting. So we have proved

Theorem 3.6: The $(2, n)$ knots and the Fibonacci knots F_n have trees of projections relative to changing and smoothing, and smoothing and deleting, respectively, which can be the largest possible given a knot with n crossings.

It is left to the reader to show that the n-trefoil will produce a knot which can have the largest possible tree of projections relative to changing and deleting within a constant.

It has been established that there are knot projections (and link projections, for that matter) whose trees of projections can have an exponential number of nodes relative to the number of crossings in the projections. The tree of projections may be built as needed and taken apart immediately afterward.

Any operation on the nodes of the tree of projections of a knot can be considered the fundamental operation. The operation of checking for an unknot or split link seems to fit the bill best (see Algorithm 2.1). This is because the appearance of the unknot or split link indicates a leaf node in the tree of projections with no children; hence, the algorithm may stop smoothing and changing down that particular branch of the tree.

And now a nice application: Denoting the Conway algorithm by C.

Theorem 3.7: $C_{\text{worst}}(n) = O((1+\sqrt{5})/2)^n)$; or the Conway algorithm has exponential worst cast time complexity.

The Conway algorithm, given any knot projection K is invariant since the Conway polynomial of the knot K is well defined given any knot projection of K (Theorem 1.7). Let n denote the number of crossings of a particular knot. By Theorem 3.5, using C in constructing the Conway polynomial of the standard projection of the torus knot $(2, n)$, it is possible to produce an exponential tree of projections. So, $C_{\text{worst}}(n)$ has exponential time complexity dominated by $O(((1+\sqrt{5})/2)^{\ })$ unknotting checks. □

Corollary 3.8: Any polynomial invariant constructed by any consistent combination of knot operations (changing, smoothing, and deleting) on a knot

projection is of exponential worst case cost. Provided the operations are performed to the given knot projection until all of the derived knot or link projections are resolved into unknots and split links at least.

This is true, since these operations are irrespective of the unknown's and their coefficients.

For example, take the Homfly polynomial [6] of a knot K, $\text{Homfly}_K(x, y, z)$, which can be calculated in a similar fashion to Algorithm 2.1. The Conway polynomial of the same knot $\nabla_K(z)$ can then be created by setting $x = 1$ and $y = 1$, resulting in $\text{Homfly}(1, 1, z) = \nabla_K(z)$.

It might be noted that Conway's algorithm has a constant best case time complexity. It seems to be a very hard problem to decide the average case time complexity of Conway's algorithm.

Theorem 3.9: The act of creating the Conway polynomial is in the class P.

This will be proved by the presentation of an algorithm which can determine the Conway polynomial in better than $O(n^3)$ time.

Algorithm 3.10: (A version of Alexander's Algorithm, [3]; see also [13])

Given: a projection of a knot K.
Returning: the Conway polynomial of the knot K.

1. Choose an orientation of the knot projection K. Label the crossings from X_1 to X_n for a knot with n crossings.
2. Create an n by n matrix, calling it mat, filling all entries of mat with zeros.
3. Fill the entries of the matrix as follows:
   Each crossing is associated with a column of the matrix.
   for col : = 1 to n do
   Say crossing col is:

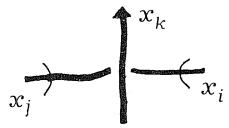

FIGURE 9

Then let

$$\text{mat}[k, \text{col}] : = 1 - z$$
$$\text{mat}[i, \text{col}] : = z$$
$$\text{mat}[j, \text{col}] : = -1$$

endfor

4. Disregard any row and column of the matrix mat producing a $(n-1) \times (n-1)$ matrix.
5. Calculate the determinant of the matrix produced in step 4.
6. The polynomial created by computing the determinant is the Alexander polynomial.
7. The Alexander polynomial is converted to the Conway polynomial by noting $\Delta_K(z^2) = \nabla_K(z - z^{-1})$ where equality is up to multiples of $\pm z^n$.
8. Normalize $\Delta_K(z^2)$ into $\nabla_K(z)$. Return $\nabla_K(z)$.
9. End.

This algorithm clearly terminates, and the computationally time consuming part of Algorithm 3.10 is step 5, calculating the determinant. Step 5 can be done by a straightforward algorithm in $O(n^3)$ time. For example, performing

Gaussian elimination followed by multiplication along the diagonal computes a determinant in $O(n^3)$ time. There are other algorithms which can compute this in slightly better time. The fact that the matrix is sparse, only having three nonzero elements in each column, can also be taken into account.

This $O(n^3)$ complexity is the same given any n crossing knot projection. Therefore, denoting Alexander's algorithm as Alex, it must be that

$$\text{Alex}_{\text{best}}(n) = \text{Alex}_{\text{worst}}(n) = \text{Alex}_{\text{average}}(n) = O(n^3),$$

assuming Gaussian elimination followed by multiplication along the diagonal is used to calculate the determinant. Hence, the act of creating the Conway polynomial is in the class P.

4. Conclusion

In this paper it is established that the consistent application of elementary knot operations may lead to an exponential number of derived knot projections. This illustrates that Conway's algorithm has exponential worst case time complexity. Moreover, the nonvacuous upper bound on the worst case complexity is based on the golden ratio. It was then illustrated that a determinant based algorithm given by Alexander is of polynomial time complexity; hence, calculating the Conway polynomial is in the class P. It is interesting to note that Jaeger [8] has shown the calculation of the Homfly polynomial to be in the class NP-Hard.

Corollary 3.8 shows that this complexity analysis can be applied to the calculations of the Jones polynomial [9], the Homfly polynomial [6], the Kauffman polynomial [12, Appendix], and many other polynomial invariants of knots and links.

Hopefully, algorithms for the calculation of knot polynomials will receive more attention in the future. Many interesting questions remain open. It is presently unknown whether any knot polynomial can detect knottedness. Yet Fellows & Langston [5] have recently nonconstructively shown that detecting knotlessness is in P. So the quest is on to find some invariant, knot polynomial or otherwise, that can recognize the unknot in polynomial time.

Acknowledgments

I owe a great deal to Jim Hoste for introducing me to knot theory and for his suggestions regarding this paper. I must also give my gratitude to Yair Minsky for his helpful criticisms. The referee's comments were also helpful.

References

1. G. M. Adel'son-Vel'skii & Y. M. Landis. "An Algorithm for the Organization of Information." *Soviet math. Dokl. 3* (1962):1259-62.
2. A. V. Aho, J. E. Hopcroft, & J. D. Ullman. *The Design and Analysis of Algorithms*. New York: Addison Wesley, 1974.
3. M. A. Armstrong. *Basic Topology*. New York: Springer Verlag, 1983.
4. J. H. Conway. "An Enumeration of Knots and Links, and Some of Their Algebraic Properties." In *Computational Problems in Abstract Algebra*. Ed. John Leech. New York: Pergamon Press, 1970, pp. 329-58.
5. M. R. Fellows & M. A. Langston. "Nonconstructive Tools for Proving Polynomial Decidability." *Journal of the Assoc. for Comp. Mach. 35.3* (July 1988):727-739.
6. P. Freyd, D. Yetter, J. Hoste, W. B. R. Lickorish, K. Millett, & A. Ocneau. "A New Polynomial Invarient of Knots and Links." *Bull. Amer. Math. Soc. 12.2* (April 1985):239-46.

7. J. Hoste. Personal Note, 1986.
8. F. Jaeger. "On Tutte Polynomials and Link Polynomials, Laboratoire Styructurs Discretes et Didactique." Rapport de Recherche Imag France, Mai 1987.
9. V. F. R. Jones. "A Polynomial Invarient for Knots via von Neumann Algebras. *Bull. Amer. Math. Soc. 12* (1985):103-12.
10. L. H. Kauffman. *Formal Knot Theory*. Mathematical Notes 30. Princeton, NJ: Princeton University Press, 1983.
11. L. H. Kauffman. "The Conway Polynomial." *Topology 20* (1980):101-08.
12. L. H. Kauffman. *On Knots*. Mathematical Notes: Annals Study 115. Princeton, NJ: Princeton University Press, 1987.
13. H. F. Trotter. "Computations in Knot Theory." In *Computational Problems in Abstract Algebra*. Ed. John Leech. New York: Pergamon Press, 1970, pp. 359-64.
14. J. C. Turner. "On a Class of Knots with Fibonacci Invarient Numbers." *Fibonacci Quarterly 24.1* (1986):61-66.

*****