REFERENCES


1. J. W. Alexander and G. B. Briggs: On types of knotted curves. *Ann. of Math.* **28** (1926–27), 562.
2. A. L. Anger: Machine calculation of knot polynomials. Princeton Senior Thesis, 1959.
3. R. H. Fox: A quick trip through knot theory. *Topology of 3-manifolds and related topics* (Prentice-Hall, Englewood Cliffs, N.J., 1962).
4. C. N. Little: On knots, with a census to order 10. *Trans. Conn. Acad. Sci.* **18** (1885), 374–378.
5. C. N. Little: Alternate ± knots of order 11. *Trans. Roy. Soc. Edin.* **36** (1890), 253–255.
6. C. N. Little: Non-alternate ± knots. *Trans. Roy. Soc. Edin.* **39** (1900), 771–778.
7. K. Murasugi: On a certain numerical invariant of link types. *Trans. Amer. Math. Soc.* **117** (1965), 387–422.
8. K. Reidemeister: *Knotentheorie* (reprint) (Chelsea, New York, 1948).
9. P. G. Tait: On knots, I.
10. P. G. Tait: On knots, II.
11. P. G. Tait: On knots, III.

   } First published separately, but available together in Tait's *Scientific Papers*, Vol. I, pp. 273–347 (C.U.P., London, 1898).


# *Computations in knot theory*

H. F. Trotter

**1. Computer representation of knots.** The commonest way of presenting a specific knot to the human eye is by a diagram of the type shown in Fig. 1, which is to be interpreted as the projection of a curve in 3-dimensional space.

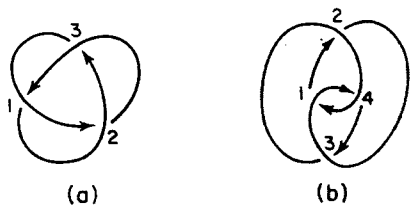


Fig. 1

There are obviously many ways of coding the information in such a diagram for a computer. Conway's notation **[2]** (which I learned of for the first time at the conference) seems to me much the best both for handwork and (perhaps with some modification) for computer representation. In some work done at Kiel **[3, 4, 11]** under the direction of Prof. G. Weise, one notation used is based on noting the cyclic order of vertices around the knot, and another is related to Artin's notation for braids. The simple notation described below is what I have actually used for computer input. It has proved reasonably satisfactory for experimental purposes.

To each vertex of the diagram there correspond two points on the knot, which we refer to as the upper and lower nodes. Each node has a successor arrived at by moving along the knot in the direction indicated by the arrows. Each vertex has one of two possible orientations, as indicated in Fig. 2. If the vertices are then numbered in an arbitrary order, a complete descrip-

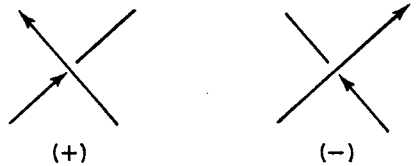


Fig. 2

tion of the diagram is obtained by listing, for each vertex, its orientation and the successors of its upper and lower nodes. The descriptions of the diagrams in Fig. 1 are then

(a)
1+L2 U2
2+L3 U3
3+L1 U1

(b)
1−L4 L2
2−U4 U3
3−U1 U2
4−L1 L3

where $L$, $U$ stand for "lower" and "upper".

This description is highly redundant, but the redundancy is at least in part a virtue, since it increases the likelihood that a coded description is correct if it is self-consistent. In practice, the notation is fairly easy to use, although errors in recording the orientations of vertices do crop up.

The coding just described is intended to be convenient to write down, and does not correspond directly to a useful internal representation for a knot. Some sort of list organization appears to be most appropriate, and the list-processing language *L6* [7] was chosen because it was available and seemed to be well-suited to the problem. In addition to efficient mechanisms for storage allocation and subroutine organization, *L6* has several distinctive features. It deals with blocks of computer words as single objects, and provides for declaration of fields (denoted by single letters) within blocks. A field may contain either a pointer to another block, or numerical or coded data. Indirect referencing (up to five levels deep) is denoted simply by concatenation. For example, *XAT* refers to field *T* of the block pointed to by field *A* of the block pointed to by pseudo-register *X*.

A program has been written which reads a knot description, making some elementary checks for consistency, and produces a linked list of blocks in which there is one block for each node. Each block contains fields which point to the preceding and following nodes and to the other node at the same vertex. Other fields contain the vertex number and indicators for the orientation of the vertex and the type of the node (upper or lower). The program will also handle links (i.e. knots with more than one component) and another field contains the number of the component to which the node belongs. In addition, each block contains two fields used to link it temporarily into various lists for housekeeping purposes during computation. In this representation it turned out to be quite easy to write a program to perform trivial simplifications of diagrams by application of the Reidemeister moves $\Omega.1$, $\Omega.2$ ([12], p. 7) (see Fig. 3).

**2. Computation of algebraic invariants of knots.** The most straightforward application of computers to knot theory is in the computation of known algebraic invariants. The first work of this kind that I know of was done in 1959 at Princeton [1] on an IBM 650 computer, and consisted of the calculation of Alexander polynomials for the alternating 10-crossing knots in Tait's tables [15]. Similar calculations were later made for the non-



alternating 10-crossing knots and for the alternating 11-crossing knots in the tables of Tait and Little [8, 9, 10, 15]. (Conway independently did these calculations by hand for his own knot tables [2].) Programs for computing the Alexander polynomial and several other invariants are described in [3].

The most generally useful algebraic invariants of knots are connected with the homology of cyclic coverings of the complement of the knot. Seifert [14] showed that these invariants can be computed from an integer matrix constructed as follows. The first step is to find a non-self-intersecting orientable surface with the knot as boundary. (It is not altogether obvious that such a surface always exists, but Seifert actually indicates a method of constructing one for any knot diagram.) Unless the knot is trivial, the surface has a genus $h$ greater than 0, and its first homology group is free abelian on $2h$ generators. A system of $2h$ closed curves which represent an integral basis for this homology group can be found on the surface. A Seifert matrix for the knot is then obtained by taking as $ij$th entry the linking number (in 3-space) of the $i$th basis curve with a curve obtained by lifting the $j$th basis curve slightly above the surface. (The side of the surface which is "above" is of course determined by the orientation of the surface.) Any knot has infinitely many different Seifert matrices belonging to it (and any Seifert matrix belongs to infinitely many distinct knots).

Programming an algorithm to find a Seifert matrix from a knot description is an interesting problem. An especially noteworthy program in [3] actually finds a Seifert surface and transforms it into the canonical form of a disk with attached bands, from which a Seifert matrix is then easily obtained. I have written an *L6* program which finds a set of basis curves and computes the matrix without first altering the surface.

Let us say that two Seifert matrices are $s$-equivalent if they are the first and last members of some finite sequence of matrices such that for each consecutive pair in the sequence there is some knot to which both matrices of the pair belong.

No purely algebraic characterization of $s$-equivalence is known, but it is closely related to the property of congruence of matrices. We say that matrices $A$ and $B$ are congruent over a ring $R$ if $A = PBP'$ where $P'$ is the transpose of $P$, and both $P$ and its inverse have elements in $R$. It is quite obvious that two Seifert matrices which are congruent over the integers are $s$-equivalent, since any such change in the matrix can be obtained simply by choosing a different basis for the first homology group of the Seifert surface. The converse is known to be false, but a modified converse is true [16].

Before stating this converse we must remark that any Seifert matrix is $s$-equivalent to a non-singular one (unless it is $s$-equivalent to one belonging to the trivial knot), and that if $V$ and $W$ are $s$-equivalent non-singular matrices then $\det(V) = \det(W)$. (In fact $\det(V - tV') = \det(W - tW') = A(t)$, the Alexander polynomial of the knot.) The statement then is that two

$s$-equivalent non-singular Seifert matrices are congruent over the subring of the rationals generated by the reciprocal of their common determinant, and are *a fortiori* congruent over the rationals. Note that for the special case of the determinant equal to $\pm 1$, integral congruence and $s$-equivalence coincide.

Even the question of whether two Seifert matrices are congruent over the rationals presents some difficulty. Seifert matrices are not symmetric (indeed $V - V'$ is always non-singular, with determinant 1), so that ordinary quadratic form theory does not apply. This problem has been studied (see [17] and its bibliography) and it appears to reduce to questions about quadratic forms in algebraic number fields for which algorithmic solutions are (at least in principle) known.

Questions of congruence over the integers or some other subring of the rationals are of course much more complicated. One becomes involved with determining whether matrices are similar over the ring as an initial step. This problem is connected with that of determining whether two ideals in an algebraic number field are in the same class, and is in general even more difficult.

The prospect for completely general algorithms for determining the congruence classes of Seifert matrices does not therefore seem very hopeful. Something less than complete generality, however, can still be useful. A system of Fortran programs for manipulating integer matrices has been written, and has been used so far for computing Alexander polynomials from the Seifert matrices. Some calculations on individual knots have also been carried out, and in particular the knots labelled 9-28 and 9-29 in Reidemeister's table [7] have been shown to have integrally congruent Seifert matrices. This experience indicates that programs which will automatically decide the $s$-equivalence of Seifert matrices in a great many cases may be quite feasible. Such programs must combine calculation of nivariants which potentially may distinguish the matrices with a search (guided by the theory) for a demonstration of equivalence if the invariants turn out not to distinguish the matrices.

**3. Manipulation of knot diagrams.** The major project in knot theory in which it appears most reasonable to use a computer is to check and extend the tables of knots and their various computable invariants. Tables which are generally believed to be complete and accurate (except for a few errata noted by Seifert [14]) are given in Reidemeister's book [12] for all prime knots of less than ten crossings. These tables include, besides diagrams of the knots themselves, the Alexander polynomials, the torsion numbers of degrees 2 and 3, and the Minkowski invariants of the quadratic forms of these knots. Tables of knot diagrams for knots of up to 10 crossings, and of alternating knots of 11 crossings, published by Tait, Kirkman, and Little [5, 6, 8, 9, 10, 15] during the 1880's, have already been referred to. Conway has made more extensive tables, and found a few errors in the earlier ones.



So far as I know, no other attempts at tables of 10- or 11-crossing knots have been made.

While it seems feasible, although perhaps not easy, to get a program to create all possible knot diagrams of a given complexity, a real difficulty (which becomes more and more serious as the complexity of the diagrams increases) is that a given knot may appear in a number of different forms. Within the limits of the tables that have been made so far, it is humanly possible to recognize the equivalence of these forms, but it is not clear how to make a computer program to do this with any reasonable degree of efficiency.

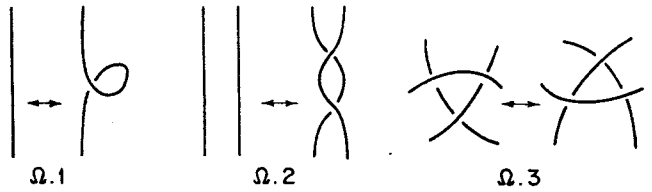


FIG. 3.

If two diagrams are suspected of representing the same knot, then one may attempt to transform one into the other by a sequence of Reidemeister moves, which are indicated schematically in Fig. 3. It is known that if the knots are in fact equivalent, then some sequence of moves which will convert one into the other does exist. It is not hard to make a program to carry out individual moves, but it appears difficult to program anything more efficient than an exhaustive search of all possibilities. Some interesting programs have been written at Kiel [4, 11] which seem to work quite well and have been used successfully to tabulate all knots of up to 8 crossings. The empirical evidence furnished by the handwork of Tait, Little, and Conway, however, shows that the number of distinct diagrams of a given knot rises rapidly as the number of crossings increases, and it is not clear that these programs would be usable for classifying knots of more than 9 crossings.

Another possible technique is that of trying to reduce given projections of knots to a form with a minimum number of bridges or overpasses. (A bridge is a maximal sequence of consecutive upper nodes. In Fig. 1(a) there are 3 bridges. Figure 1(b), in spite of having more vertices, has only 2 bridges.) Practically all the knots of less than 12 crossings can be put in a form with not more than 4 bridges, and the great majority can be reduced to 2 or 3.

The advantage of diagrams with only a few bridges is that even though they may contain many vertices, they can be characterized by a comparatively small number of integers. A knot with only 2 bridges, for example, can be characterized by a pair of integers $(a, b)$, with $a$ odd and $b$ relatively prime

to $a$. Schubert [13] showed that knots $(a, b)$ and $(c, d)$ are equivalent if and only if $a = c$ and either $b \equiv d \pmod{a}$ or $bd \equiv 1 \pmod{a}$. The situation with 3 and 4 bridges is certainly a good deal more complicated. It appears to be easy enough to reduce diagrams to forms that have a good chance of having a minimal number of bridges. The crucial (as yet unanswered) question is whether there are criteria for the equivalence of the corresponding knots which are powerful enough to be helpful and simple enough to be usable. (Simple necessary and sufficient conditions are probably too much to hope for.)

*Note added in proof.* As his senior undergraduate thesis at Princeton in 1968, D. Lombardero wrote a Fortran program which accepts as input the description of a knot or link in Conway's notation and calculates a Seifert matrix for it. In part, the program uses an algorithm due to A. Tristram which was communicated to me by Conway at the conference. The computation of matrices and Alexander polynomials for all the knots listed by Conway required less than five minutes on an IBM 7094.

## REFERENCES


1. A. L. Anger: Machine calculation of knot polynomials. Princeton Senior Thesis, 1959.
2. J. H. Conway: An enumeration of knots and links, and some of their algebraic properties. These Proceedings, pp. 329–358.
3. P. Grosse: Die Berechnung spezieller Knoteninvarianten mit Hilfe elektronischer Rechenanlagen, Diplomarbeit, Kiel, 1962.
4. J. Kielmann: Koordinatenunabhängige Normierung und Erzeugung von Knoten, Diplomarbeit, Kiel, 1965.
5. T. P. Kirkman: The enumeration, description and construction of knots of fewer than ten crossings. *Trans. Roy. Soc. Edinburgh* **32** (1885), 281–309.
6. T. P. Kirkman: The 364 unifilar knots of ten crossings enumerated and defined. *Trans. Roy. Soc. Edinburgh* **32** (1885), 483–506.
7. K. C. Knowlton: A programmer's description of *L*6. *Comm. Assoc. Comp. Mach.* **9** (1966), 616–625.
8. C. N. Little: On knots, with a census for order ten. *Connecticut Trans.* **7** (1885), 27–43.
9. C. N. Little: Non-alternate ± knots, of orders eight and nine. *Trans. Roy. Soc. Edinburgh* **35** (1889), 663–665.
10. C. N. Little; Alternate ± knots of order 11. *Trans. Roy. Soc. Edinburgh* **36** (1890), 253-255.
11. V. Malerczyck: Ein Verfahrung zur Aufzählung von Knoten mit Hilfe elektronischer Rechenanlagen, Diplomarbeit, Kiel, 1966.
12. K. Reidemeister: *Knotentheorie*, Erg. d. Math. 1, No. 1, 1932 (reprint Chelsea, New York, 1948).
13. H. Schubert: Knoten mit zwei Brücken. *Math. Z.* **65** (1956), 133–170.
14. H. Seifert: Über das Geschlecht von Knoten. *Math. Ann.* **110** (1934), 571–592.
15. P. G. Tait: On knots, I, II, III (1877, 1884, 1885). *Scientific Papers* **I**, 273–347.
16. H. F. Trotter: Homology of group systems with applications to knot theory. *Ann. Math.* **76** (1962), 464–498.
17. G. E. Wall: On the conjugacy classes in the unitary, symplectic and orthogonal groups. *J. Australian Math. Soc.* **3** (1963), 1–62.


# *Computer experiments on sequences which form integral bases*

Shen Lin

Let $S \equiv \{s_1, s_2, \ldots, s_k, \ldots\}$ be a sequence of positive integers and consider the set $P(S)$ consisting of all numbers which are representable as a sum of a finite number of distinct terms from $S$. We say $S$ is complete if all sufficiently large integers belong to $P(S)$. For a complete sequence, we call the largest integer not in $P(S)$ the threshold of completeness and denote it by $\theta(S)$. Necessary and sufficient conditions for various sequences to be complete have been studied by many authors [1, 2, 3, 4]. In particular, R. L. Graham [4] showed by elementary methods that any sequence generated by an integral valued polynomial $f(x)$ is complete if $f(x)$ satisfies the following (obviously necessary) conditions:

(1) The polynomial $f(x)$ has positive leading coefficient, and
(2) For any prime $p$, there exists an integer $m$ such that $p$ does not divide $f(m)$.

The method he used in the proof is constructive in nature, and with it he also determined the threshold of completeness for the sequence of squares $S \equiv \{1, 4, 9, \ldots\}$ as 128 and for the sequence of cubes $S \equiv \{1, 8, 27, \ldots\}$ as 12758. A closer look at his method reveals that it is easily adaptable for machine implementation. Indeed, it is possible to prove that some sequences are complete and find their thresholds of completeness by a computer in spite of the fact that the solution to this problem appears to require the verification of an infinite number of cases, namely, that all numbers larger than the threshold are indeed representable as a sum of distinct terms from $S$. To be able to do this, we require that the sequence $S$ satisfy the following property which we shall call condition $A$.

*Condition A.* There exists an integer $j_S$ such that for all $k \geqslant j_S$ we have $2s_k \geqslant s_{k+1}$.

All sequences which we shall consider satisfy this condition trivially. In addition, we may without loss of generality assume that the sequence $S \equiv \{s_1, s_2, \ldots, s_k, \ldots\}$ is arranged in a nondecreasing manner, i.e. $s_i \leqslant s_j$ if $i < j$.

In the following, we shall show how one may use the computer to prove some sequences complete and find their thresholds of completeness. The