Trivial or Knot: A Software Tool and Algorithms for Knot Simplification

Daniel Lewin[*] Orli Gan[*] Alfred Bruckstein[**]

[*] Electrical Engineering Department
Technion, 3200 Haifa, Israel
danl@tx.technion.ac.il
orli@tx.technion.ac.il

[**] Computer Science Department
Technion, 3200 Haifa, Israel
(on sabbatical at AT&T Bell Laboratories,
Murray Hill, NJ 07974, USA)

Abstract

A special type of representation for knots and for local knot manipulations is described and used in a software tool called TOK to implement a number of algorithms on knots. Two algorithms for knot simplification are described: simulated annealing applied to the knot representation, and a "divide-simplify-join" algorithm. Both of these algorithms make use of the compact knot representation and of the basic mechanism TOK provides for carrying out a predefined knot manipulation on the knot representation. The simplification algorithms implemented with the TOK system exploit local knot manipulations and have proven themselves effective for simplifying even very complicated knots in reasonable time.

Introduction - What is Knot Theory?

Knots are very complicated mathematical objects that have intuitive, real-world counterparts. This makes them very interesting to study. A tangle in a (frictionless***)*** rope is a knot if when the ends of the rope are pulled in opposite directions, the tangle is not unraveled. Given a pile of rope with two ends sticking out, it is difficult, or even impossible to say by inspection whether or not the rope is truly knotted. An even more difficult problem is to decide if two piles of tangled rope are ***equivalent***; meaning that one pile may be stretched and deformed to look like the other pile without tearing the rope. Figure 1 illustrates that equivalence is sometimes not obvious even for simple knots.

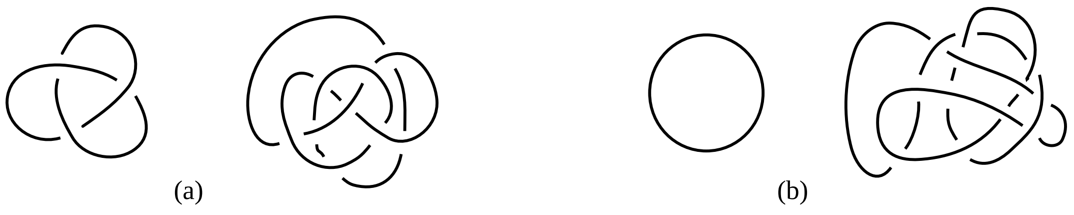


Figure 1. (a) Two trefoil knots (b) Two trivial knots

Knot theory studies an abstraction of the intuitive "knot on a rope" notion. The theory deals with questions such as proving knottedness, and classifying types of knottedness. In a more abstract sense we may say that knot theory studies the placement problem: "Given spaces *X* and *Y*, classify how *X* may be placed in *Y*". Here a placement is usually an embedding, and classification often means up to some form of movement. In these terms classical knot theory studies embeddings of a circle in Euclidean three space. (Hence we consider the two ends of the rope tied together)

There are two main schools in knot theory research. The first is called combinatorial or pictorial knot theory. Here the main idea is to associate with the mathematical object a drawing that represents the knot, and to study various combinatorical properties of this drawing. The second school considers the abstract notion of a knot as an embedding and studies the topology of the so called complementary space of the image of the embedding, by applying to this space the tools of Algebraic Topology. This paper dwells in the first realm - pictorial knot theory.

Following is a brief description of the basic theory that is needed to understand the TOK knot manipulation tool. For a more comprehensive overview see [1][2][3].

What is a Mathematical Knot?

Mathematical knots are obtained by joining together the ends of the rope without introducing new tangling, as depicted in Figure 2.

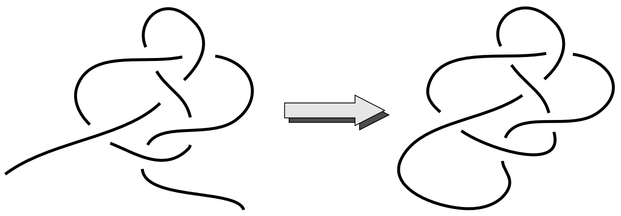


Figure 2. Mathematical knot

We may then readily define a knot as an embedding of a circle in Euclidean three space. This definition is useful when dealing with topological aspects of knots. Another popular definition is a closed, simple polygonal curve in euclidean three-space. In either case a knot is pictured as a ‘string’ in euclidean three-space. Equivalence of knots is usually defined up to a type of movement called ***ambient isotopy*** (see [4]). Equivalence by ambient isotopy of two knots means that there is a continuous deformation, via embeddings, from one embedding to the other. This type of deformation is sometimes called a ***level preserving isotopy***. The formal definition is rather complicated, but embodies the intuitive notion of manipulating a string in real life. An important result that will be presented later, allows us to use a much simpler form of knot equivalence called diagram equivalence.

Regular Projections

A drawing, or projection may be associated with a mathematical knot by projecting the embedding onto a plane. All of the above illustrations given as examples, are actually knot projections. A projection is called *regular* if each point on the projection that has more than one point mapped to it, has exactly two points mapped to it, and the set of these double points is a set of isolated points. This disallows for example projections of the types shown in Figure 3.

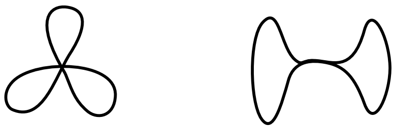


Figure 3. Non-regular projections

In this context we have the following result, stating that we can always look at a knot in space so as to see a regular projection of it, provided the knot is not infinitely knotted (as some recursive procedures might generate). Moreover the regular projections are generic, i.e. we would have to have some bad luck to see a non-regular projection.

Theorem 1 *:*

Every (tame) knot has a regular projection. In addition, the set of regular projections is dense in the set of all projections (see [5]).

Theorem 1 allows us to use knot projections without fear that we are restricting our scope. Moreover, the theorem states that finding a regular projection by randomly perturbing an existing (possibly non-regular) projection has a high probability of success. A knot *diagram* is a regular projection of the knot.

Combinatorial knot theory deals with proving or disproving knot equivalence through knot diagrams. In order to develop such a theory a notion of diagram equivalence must be introduced. Most importantly diagram equivalence should imply, and be implied by, equivalence (through ambient isotopy) of the knots that they represent. Theorem 2 by Alexander and Briggs [6] which is described below is the foundation of combinatorial knot theory.

Reidemeister Moves:

In Figure 4 six basic operations on knot diagrams, known as Reidemeister Moves [12], are depicted.

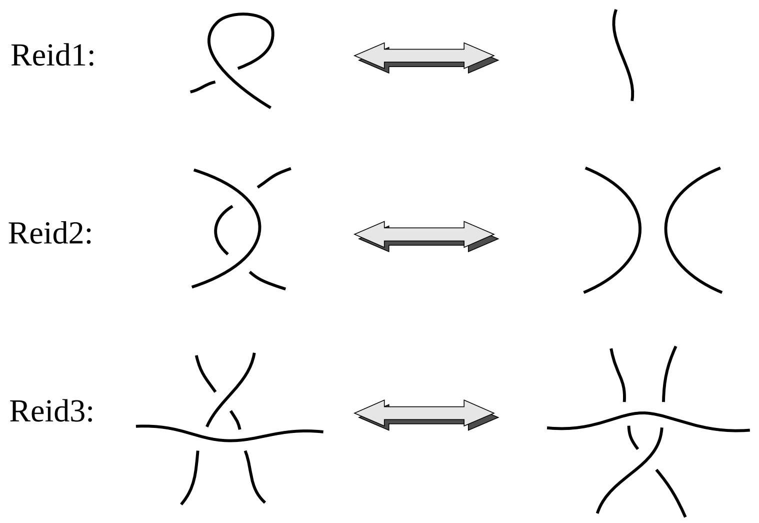


Figure 4. Reidemeister moves

Theorem 2 *:*

> *Two knots are equivalent (by ambient isotopy) iff their diagrams are related by a finite series of Reidemeister moves [6].*

This theorem may seem trivial at first glance, but some reflection shows that this is in fact a very deep result. Based on the above result, we may focus on knot diagrams (projections) and use the notion of diagram equivalence by Reidemeister moves.

There is no known algorithm that proves knot equivalence by explicitly showing a sequence of Reidemeister moves. Therefore combinatorial knot theory has concentrated on defining invariants of knots associated to their diagrams. Proving that something is an invariant of a knot boils down to showing that it is invariant under all the Reidemeister moves. Invariants can be used to show that two diagrams represent ***distinct*** knot types by showing that different values of the invariant correspond to the two knot diagrams. However, most invariants known do not distinguish between all distinct knot classes. Furthermore, most invariants are difficult to calculate. Recently, new “polynomial” invariants (e.g. the HOMFLY polynomial [7]) which are both powerful, and relatively easy to calculate have been introduced. The HOMFLY polynomial (and related polynomials) are very powerful knot invariants, and are capable of differentiating between many knot classes. Nevertheless, there remain both practical and aesthetic reasons to strive for proofs of equivalence via Reidemeister moves. A complete solution may still be far off because, in spite of the great effort that has been invested in pictorial knot theory, many of the basic questions are still unanswered.

One approach to solving the equivalence problem has been to define a canonical form for a knot class. Equivalence may be checked by first bringing two knots to their canonical forms, and then checking equality. One such canonical form is defined through the energy of a knot. Electric charge is assumed to be distributed over the knot and the resulting potential energy of the system is called the energy of the knot. The canonical form is an equivalent knot that has minimal energy. It was conjectured in [8] that this knot form is unique, and may thus be a good candidate for a canonical form. Simulated annealing has been applied in [9] to find this minimal energy knot. The annealing procedure was applied to knots represented by a piecewise linear curve in $\Re^3$.

We present in this paper a software tool for representing and manipulating knots. The platform may be used as a basis for implementing a variety of algorithms for manipulating knots. As a demonstration, a new simulated annealing algorithm that simplifies knots is implemented. The annealing algorithm exhibited good behavior for many large knots including those shown in [9]. In addition, a “divide-simplify-join” knot simplification algorithm is described which also shows promise for simplifying very large knots.

The TOK (Trivial or Knot) Framework for Knot Manipulation

Motivation Behind TOK:

TOK (Trivial Or Knot) is a tool that provides a framework for exploring knot theory through knot diagrams. It uses an abstract representation of knot diagrams and diagram moves for knot manipulation. A language is defined to enable systematic application of moves in an algorithmic manner to a knot representation.

The abstraction that represents the knot strips away information not relevant to the topology, i.e. the ‘knottedness’ of the knot. Thus, much of the work done by knot algorithms is already inherent in the representation. The terseness of the representation also leads to efficient algorithms for applying diagram moves to a knot.

Knot Representation

A knot is represented by a list of its crossings. Given an ***oriented*** knot diagram, the line segments are numbered from an arbitrary point as in Figure 5 (a). A crossing is represented by the smaller number that appears on the under-crossing, and the smaller number that appears on the over-crossing, as in Figure 5 (b).

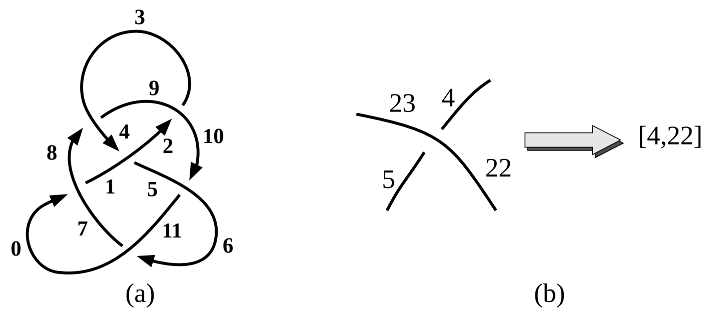


Figure 5. (a) Oriented knot diagram, numbered. (b) Crossing representation

The knot representation of the knot in Figure 5 (a) is: [0,7][4,1][2,9][8,3][10,5][6,11]

There is a one-to-one correspondence between this knot representation and the Dowker representation [13] which is a reinvention of a knot notation suggested by Gauss. In order to obtain the Dowker notation from our notation read off the pair of each segment starting with zero. For example the Dowker representation of the above knot is 7,4,9,8,1,10,11,0,3,2,5,6.

Theorem 3

The above knot representation is well defined.

This follows directly from Theorem 1 in [13] by breaking the knot into apparently prime sections and applying the theorem to each section. Note that the same list of crossings may represent a number of knot diagrams that all represent equivalent knots.

Rule Representation

A rule is a pair of two diagram sections: a *pre*, and a *post* diagram. A rule acts locally on a knot diagram by replacing a section of the knot diagram that is isomorphic to the *pre* diagram, with what appears in the *post* diagram. Such a replacement does not affect the rest of the knot diagram.

A rule is represented by a crossing pair description of its pre, and post diagrams. The area of the knot affected by the rule is called the *rule locality.* Segments entering the rule locality are assigned variables. The segments of the strands that make up the rule’s diagrams are numbered according to these variables according to the orientation of the knot. The variables are used to describe how the crossings are altered inside the rule locality. An example is shown in Figure 6.

All the Reidemeister moves can be represented by rules according to the above definition. A rule has an orientation, meaning that the orientation of the strands of the knot that take part in the rule are implied by the representation of the pre and post sections. In general a move such as a Reidemeister move may need a number of rules to represent the possible orientations of strands. Note that outside the rule locality, the knot is the same before and after the manipulation.

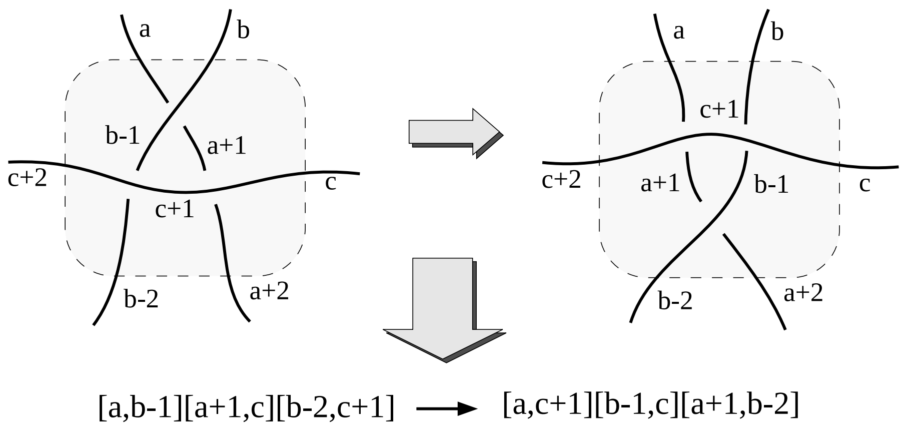


Figure 6. Rule definition example

Application of a Single Move

A rule is applied to a knot by replacing the crossing pairs in the knot that fit the description of the pre diagram with the crossing pairs specified in the post diagram description. A rule is said to be applicable to a knot if there is a set of crossings in the knot that fit the description of the pre diagram of the rule. Note that in general, a rule may change the type of a knot, since the whole knot can be replaced by another knot. A rule that does not change the knot type is called a *proper* move.

TOK applies an exhaustive search in order to locate a subset of crossings that fit a certain rule. The subsets of the knot representation are scanned in random order, to decrease the average search time. The complexity depends on the number of crossings in the pre side of the rule and the size of the knot. When a match is found, the matching crossings are replaced with the post side of the rule, and the entire knot is renumbered accordingly.

After a set of matching crossing have been found, the rule is applied to the knot via the following process: The segments of the knot are multiplied by a ***Segment Interval*** *(SI > 1),* which provides free indices for a rule to divide a segment with. The rule is then applied to the knot, and the resulting segments are sorted in increasing order and renumbered by consecutive integers.

For example:

```
knot k [0,7][1,4][2,9][8,3][10,5][6,11];
rule reid3a [b+1,a][b,c][a+1,c+1]->[a,c][b,a+1][b+1,c+1];
apply reid3a k;
```

Matching the pre side: a=9 b=1 c=4	**[2,9][1,4][10,5]**
Multiply by SI = 10:	**[20,90][10,40][100,50]**[00,70][80,30][60,110]
Apply the post side: a=90, b=10, c=40	**[90,40][10,91][11,41]**[00,70][80,30][60,110]
Renumber:	[9,4][1,10][2,5][0,7][8,3][6,11]

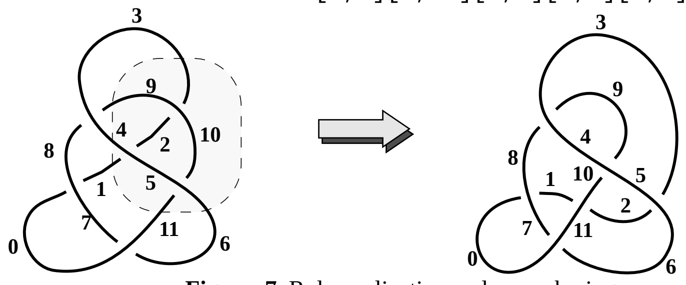
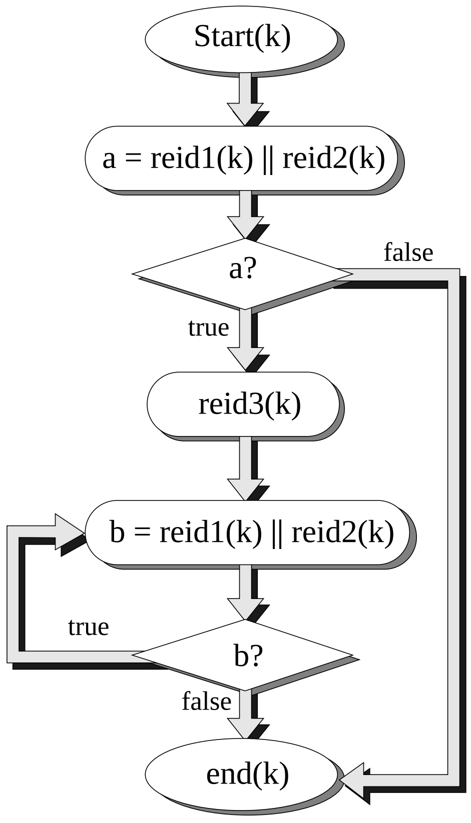


Figure 7. Rule application and renumbering

Systematic Knot Manipulation

A sequence of rules may be applied to a knot diagram to obtain a new diagram. If each rule is a proper rule, a diagram that represents a knot of the same class is obtained. The TOK tool defines a paradigm for applying moves to a diagram by means of a simple language. Following is an example that simplifies a knot diagram by applying Reidemeister moves as long as they are applicable.

The flow chart in Figure 8 describes an arbitrary algorithm for simplifying a knot ‘k’ by application of the various reidemeister moves. This algorithm applies the Reidemeister moves 1, or 2. If one of these was applicable, it applies Reidemeister move 3, and then applies moves 1 or 2 for as long as they are applicable. An example of the application of this algorithm on a knot is shown in Figure 9.



```
# Knot definition:
knot k [0,7][1,8][2,5][10,3][11,4][9,6];

# Rules definitions:
rule reid1a [a,b][a+1,b+1] -> NULL;
rule reid1b [b+1,a][b,a+1] -> NULL;

rule reid2a [a+1,a] -> NULL;
rule reid2b [a,a+1] -> NULL;

rule reid3a [b,a+1][a,c+1][b+1,c] ->
            [a+1,c][b,c+1][b+1,a];

# Systematic manipulation algorithm:
apply oneof(reid1a reid1b reid2a reid2b) k;
a = status;
if ( a ) {
  apply reid3a k;

  b = True;
  while ( b ) {
    apply oneof(reid1a reid1b
                reid2a reid2b) k;
    b = status;
  }
}
```

Figure 8. Simplification algorithm

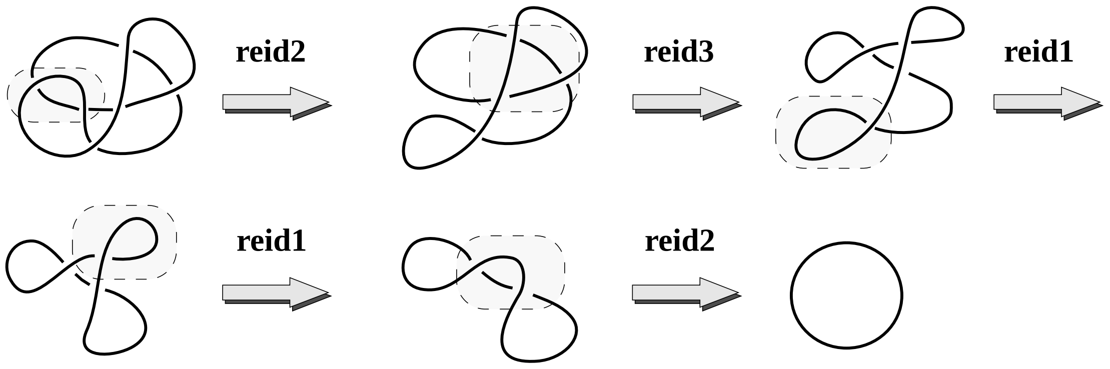


Figure 9. Application of the algorithm

Simulated Annealing

Simulated annealing is a global optimization technique that has been successfully applied to many difficult problems such as VLSI layout design, and the traveling salesman problem. The basis for the algorithm comes from observing the process of physical annealing. In this process a system with an initially given energy is slowly cooled. The state of the system at any moment is described by a given probability distribution that changes with the temperature. If the cooling process is slow enough, and the probability distribution is of a certain type, the system is guaranteed to settle into the least energy state.

The application of simulated annealing to an optimization problem is done by defining an energy function on the possible states of the system. The annealing process is then simulated on the system, minimizing the defined energy function. Many versions of simulated annealing exist. One of the most common is Boltzman annealing. The general algorithm is as follows:

1. Set the number of iterations *(n)* to 1. Set the temperature *(T)* to the initial temperature (T_0) . Set the current energy (E) to the energy of the current (given) state of the system.
2. While $T >$ *Final Temperature* repeat the following:
   a. Perturb the current system configuration by an amount that is proportional to the temperature.
   b. Calculate the energy of the new state *E'*. Let $\Delta E = E - E'$
   c. If $\Delta E > 0$ then accept the new configuration (Make the new configuration the current configuration and let *E'* be the current energy). Goto e.
   d. Else, accept the new configuration with a probability equal to $\left(1 + e^{-\frac{\Delta E}{kT}}\right)^{-1}$.
   e. Increment the number of iterations to *n+1*.
   f. Lower the temperature *T*. In Boltzman annealing the temperature is lowered according to $T = T_0 / \log n$.
   g. Return to 2.

Applying Simulated Annealing to Knots

Using the TOK system as a platform for knot manipulation we have implemented a simulated annealing algorithm that simplifies knots. The algorithm has been applied to large knots and has been shown to be effective at simplifying even very complicated knots. We note that annealing has been applied before to three dimensional models of knots [9]. The advantage of our algorithm is that our representation strips away all information that is not relevant to the topology, or the "knottedness" of the knot, so the state space is much smaller. In order to apply annealing we must define three things: the energy function, how the knot is perturbed, and the cooling process.

The Energy Function

In [9] simulated annealing was applied to a knot represented as a piecewise-linear curve. Two energy functions were used: electrostatic energy (as in [8]) and geometric energy. Both these energy functions are heavily based on the three dimensional representation of the knot, as they rely on the notion of distance. In each case the least energetic knot is what we would intuitively describe as the most 'simple' representation of the knot. In more precise terms the minimal energy knot has a regular projection that has a minimal number of crossings. The exact geometric placement of the lines between these crossings is determined by the specific energy function used.

The notion of either electrostatic or geometric energy cannot be applied to the representation of the knot diagram, as there is no meaningful 'distance' measure. Therefore a different energy function must be used. The energy function we use is based on the number of crossings in the knot diagram. The least possible number of crossings in a knot projection is called the crossing index of a knot (usually denoted $C(K)$). An energy function based on the number of crossings in a diagram makes sense since, surely, a diagram with few crossings will be the projection of a knot that has ***close*** to minimal electrostatic or geometric energy. The annealing procedure works best when the energy function does not have very sharp descents or ascents, so we refine the basic energy function as follows. A knot diagram is *alternating* if when traveling around the knot, underpasses and overpasses are met alternately. We define the energy function to minimize the number of crossings in the diagram, and to maximize the amount of alternation in the knot diagram. The relative weight of crossings is chosen larger than the weight of alternation in the energy function since alternation is sometimes opposed to minimizing the number of crossings, and obtaining fewer crossings remains our main objective. The reason that we seek alternating knot diagrams is that there are a number of very elegant theorems that we may apply when we reach a knot diagram that is alternating. These will be described in the next section.

Perturbing the Knot

According to the simulated annealing algorithm, the amount that the current state is perturbed is proportional to the temperature. The annealing procedure in TOK may apply a number of rules, one after the other, to perturb the state of a diagram. For each rule the rule ***wildness*** is defined as the number of crossings in the post-diagram minus the number of crossings in the pre-diagram. At each stage of the annealing process a number called the ***step*** is randomly chosen from a distribution whose variance decreases as the temperature drops. Rules are then selected according to a distribution that prefers rules with high wildness at higher temperatures, and rules with lower wildness at lower temperatures, until the total wildness of all rules that have been applied to the knot is larger than the step. In this way the effect of large changes at higher temperatures, and smaller, more modest changes at lower temperatures is obtained.

Cooling

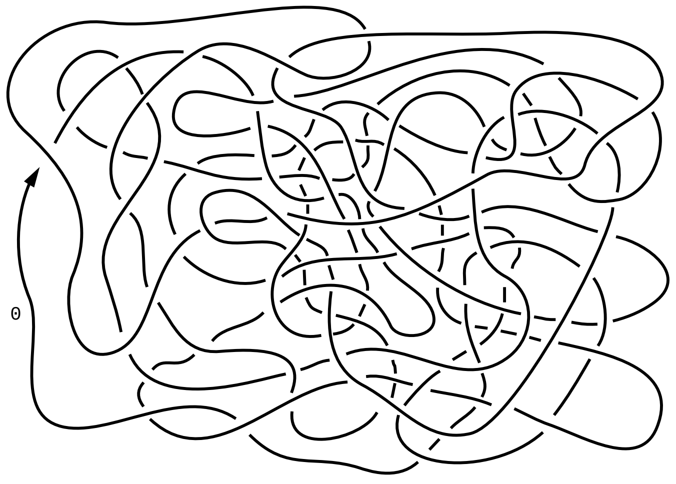
An effect that was noticed also in [9] is that the cooling process becomes very slow after the initial drop in the temperature because of the logarithmic decrease in the temperature. In our case, after the temperature is almost constant the state of the knot begins to oscillate around a base configuration that may not be the minimal energy configuration. This generally occurs when rules that have positive wildness are included in the list of rules with which the perturbations are done. This is because most positive wildness rules are applicable to the knot no matter what the current configuration. In order to speed up the cooling after reaching an equilibrium point we restart the cooling process by causing another logarithmic drop in the temperature. This is done after a constant number of iterations which we call the Cooling Restart Interval.

Experimental Results

Many knots may be simplified completely using systematic Reidemeister move applications. A simple TOK macro applies a Reid3 type move and then reduces crossings with any Reid1 or Reid2 type moves possible. This sequence is repeated until no further simplification is obtained. For example, all of the knots given as examples in [9] were completely simplified using this systematic method. Annealing is needed for very complicated knots, although in the same amount of time the systematic method also shows good behavior for some knots. One central advantage of the annealing is that it can prefer alternating diagrams which is important for the Divide-Simplify-Join algorithm described below. Another important advantage of annealing is that there are knots that require a ‘complication’ move (e.g. an ‘inverse’ Reid1 or Reid2 move) to be applied before any ‘simplification’ moves (e.g. Reid1 or Reid2 or Reid3) may be applied. An example of such a knot is given in Figure 11. Simulated annealing has the important quality that it can ‘bounce out’ of such local minima by temporarily raising the energy of the knot. The syntax in TOK for the annealing procedure allows any list of rules to be passed to the annealing routine to be used in the annealing process. In addition, annealing parameters such as the Cooling Restart Interval and the Initial Temperature may be controlled from within TOK. The following examples were simplified using only the basic Reidemeister moves, and with the default values for the annealing parameters.

Example 1:

The knot shown in Figure 10 is a 111 crossing trivial knot. The knot was simplified by an annealing procedure using Reidemeister moves. The simplification took approximately 6 minutes (on an RS6000 workstation). Deterministic move application simplified the knot to only around 65 crossings (note that there are quite a few immediate simplifications present in the knot).

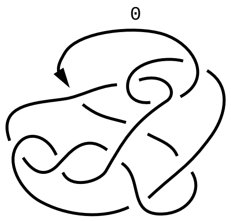


```
knot trivial
  [0,101][162,1][161,2][3,104][4,57][56,5]
  [187,6][166,7][41,8][9,52][10,93][11,194]
  [49,12][124,13][117,14][15,46][16,109]
  [17,156][18,221][220,19][111,20][21,128]
  [113,22][23,204][217,24][25,132][207,26]
  [208,27][28,133][29,150][30,201][31,214]
  [32,69][33,196][34,77][35,90][36,65]
  [170,37][38,81][39,54][40,167][42,95]
  [122,43][119,44][190,45][47,126][48,115]
  [50,193][51,94][168,53][186,55][58,83]
  [59,178][60,139][174,61][62,181][63,142]
  [183,64][66,79][67,92][195,68][213,70]
  [200,71][72,149][73,134][74,209][75,136]
  [76,145][78,91][169,80][185,82][84,103]
  [102,85][177,86][138,87][175,88][144,89]
  [96,121][97,120][189,98][108,99][157,100]
  [160,105][163,106][107,158][155,110]
  [112,129][114,153][116,125][118,191]
  [123,192][154,127][205,130][218,131]
  [210,135][137,176][173,140][141,180]
  [182,143][197,146][212,147][199,148]
  [216,151][203,152][164,159][188,165]
  [171,184][172,179][198,211][202,215]
  [219,206];
```

Figure 10. A 111 crossing trivial knot and its TOK representation

Example 2:

The trivial knot shown in Figure 11 has the interesting property that none of the simplifying, or Reid3 type Reidemeister moves can be applied until a complication move (inverse Reid1 or inverse Reid2) has been applied. This knot was simplified with an annealing process in around 30 seconds. Annealing is particularly well suited for simplification of this type of knot since during the initial 'hot' phase of the process the energy of the knot goes up (i.e. the number of crossings in the knot goes up). After this initial phase the knot in Figure 11 had 90 crossings. As the temperature goes down, the energy of the knot is reduced and the 'local minima' which was the initial configuration of the knot is not reached. Configurations such as the one shown in Figure 11 are often reached while simplifying large knots, but the 'hill climbing' ability of the annealing algorithm allows further simplification in such cases.



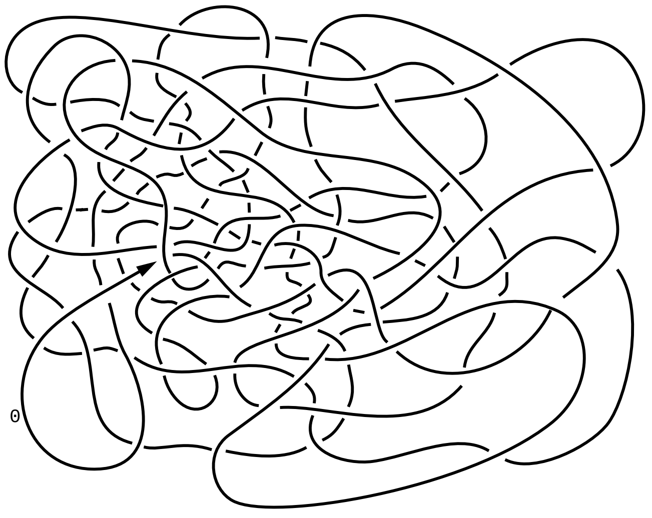
```
knot no_moves
  [0,7][1,10][2,15][14,3][4,11][12,5]
  [6,13][8,17][18,9][16,19];
```

Figure 11. A trivial knot needing complication moves in order to simplify

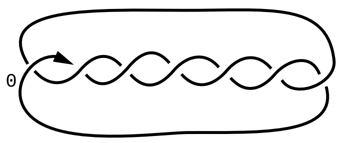
In addition to the two above examples, a number of trefoil knots with initial configurations ranging from 50 to 65 crossings were simplified, and all reached the minimal, three crossing configuration within 10 minutes. The number of annealing steps required to simplify even the most complicated knots that we tried was less that a few thousand. Changing the annealing parameters, and the set of rules that the annealing processes used affected the speed of the simplification process for a number of knots. We found that the number and type of complication moves, and the Cooling Restart Interval were the factors that influenced most intensely the simplification process.

Example 3:

Figure 12 depicts a 119 crossing complicated 2,7 torus knot, and its simplified version as found by an annealing process. The knot was simplified using Reidemeister moves and a number of “higher order” complication moves, i.e. complication moves with larger wildness than the complication Reidemeister moves. The simplification process was actually faster when these additional complication moves were introduced into the annealing process. We suspect that this is because such moves induce an effect of locality on complication, meaning that when a complication takes place, more complication is concentrated in a particular area of the knot. In order of obtain the same complication using only Reidemeister moves, the moves would have to be all applied to a particular area of the knot and this does not always happen in TOK’s rule application algorithm because of the random search. We have examined a number of examples where such concentrated application of complication moves would hasten the simplification process. Based on these observations, TOK’s rule application algorithm will be enhanced to enable the search for crossings fitting the pre-side of a rule to be done in a heuristic order. For example, the knot representation is enhanced so that crossings that have taken part in one of the last move applications are marked. The subset of marked crossings may be searched before the entire set of crossings, for a match to the next rule pre-side. Such a heuristic would have the effect of concentrating rule applications in a particular area of the knot.



```
knot complex_torus
  [187,0][18,1][235,2][70,3][4,97][214,5]
  [95,6][206,7][8,73][46,9][10,117][34,11]
  [12,43][13,122][14,195][15,38][16,237]
  [188,17][186,19][20,149][21,184][22,233]
  [23,176][24,101][25,54][26,139][64,27]
  [219,28][29,90][30,113][31,164][32,199]
  [33,78][44,35][193,36][120,37][196,39]
  [81,40][124,41][79,42][45,118][47,74]
  [48,93][49,170][50,203][218,51]
  [141,52][66,53][55,154][56,137][57,228]
  [58,107][59,132][131,60][61,226][62,157]
  [63,110][140,65][67,174][68,99][69,146]
  [71,212][72,207][75,168][115,76][166,77]
  [123,80][197,82][162,83][161,84][130,85]
  [86,225][87,158][88,111][89,220][91,202]
  [92,169][94,205][96,213][145,98][175,100]
  [231,102][178,103][151,104][134,105]
  [106,181][227,108][156,109][221,112]
  [201,114][167,116][119,192][194,121]
  [125,198][126,163][222,127][159,128]
  [129,224][133,182][135,180][136,229]
  [138,155][142,217][172,143][215,144]
  [234,147][185,148][183,150][179,152]
  [230,153][223,160][200,165][171,204]
  [173,216][232,177][189,236][209,190]
  [210,191][211,208];
```



```
knot torus
  [7,0][1,8][9,2][3,10][11,4][5,12][13,6];
```

Figure 12. A 119 crossing 2,7 torus knot and its simplified version

Knot Simplification by a “Divide-Simplify-Join” Process

As another example of knot algorithms that may be implemented using the TOK framework, we describe a knot simplification process which takes advantage of the fact that some knots may be simplified in sections. Concentrating on simplifying smaller sub-sections of a very large knot brings down the time needed to exhaustively search the knot for applicable moves. The algorithm is based on the observation that if a knot may be divided into *independent sections* (see below), then each section may be simplified separately.

Simplifying Independent Sections

A knot diagram may be divided into *independent sections* if there exists a closed simple curve in the plane of the knot projection that intersects the knot diagram in exactly two different segments. A diagram that cannot be divided into independent sections is called *apparently prime* (a *prime knot* is one that has **no** diagram that can be divided into independent sections).

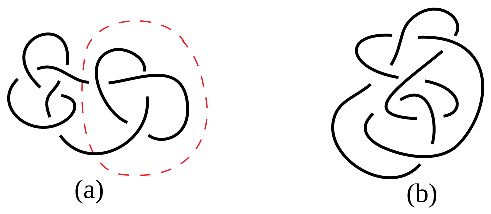


Figure 13. (a) Independent sections. (b) Apparently prime knot.

After identifying that a knot may be divided into independent sections, we may simplify the knot in pieces. Each section may be simplified independently of the others by replacing the rest of the knot with “straight” segments. After each piece is simplified they are joined together, and the process may be applied again to the resulting diagram. This *“divide and simplify”* method may be applied recursively to the knot, so a diagram may be simplified by division into *apparently prime atoms* which are then simplified and joined. After the divide-simplify-join process is iterated once, the resulting knot may have more possible simplifications, so the process is applied until there are no more simplifications to be done. The basic simplification algorithm that is used on the *apparently prime atoms* may be either an annealing process or deterministic simplification. The advantage of this procedure is a huge gain in the efficiency of the simplification algorithm. Since a rule is applied to a knot by using an exhaustive search of the crossings, the fewer crossings - the quicker the search. Since each section is independent of the others we know that simplifications found in a knot sub-section are applicable also on the larger knot. The procedure *SIMPLIFY* is described as follows:

```
SIMPLIFY(K)

1. $i = 0$.
2. Divide the knot $K$ into independent sections $K_1, K_2, \ldots, K_n$ using the procedure DIVIDE which
   is described below.
3. If the knot $K$ is apparently prime then goto 6.
4. For each $K_i$, $\hat{K}_i = SIMPLIFY(K_i)$.
5. Join the simplified knots $\hat{K}_1, \hat{K}_2, \ldots, \hat{K}_n$ back into a simplified knot $\hat{K}$ by concatenating the knots,
   and renumbering accordingly.
6. Use either systematic manipulation, or simulated annealing to simplify $\hat{K}$. Assign $K = \hat{K}$.
7. Check if $K$ is irreducible (a knot is irreducible if there is no diagram of $K$ with less crossings
   than the current one) using the procedure IRREDUCIBLE described below. If $K$ is irreducible
   then RETURN.
8. If $i <$ MAX-ITTERATIONS then $i = i + 1$ and goto 2 otherwise RETURN.
```

Figures 14, 15 and 16 follow an example of the divide-simplify-join process.

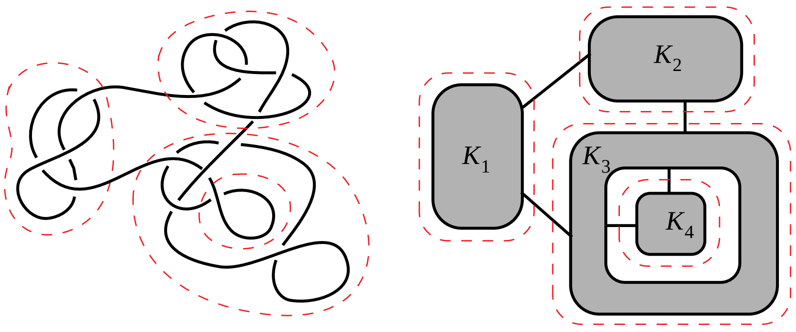


Figure 14. Knot division

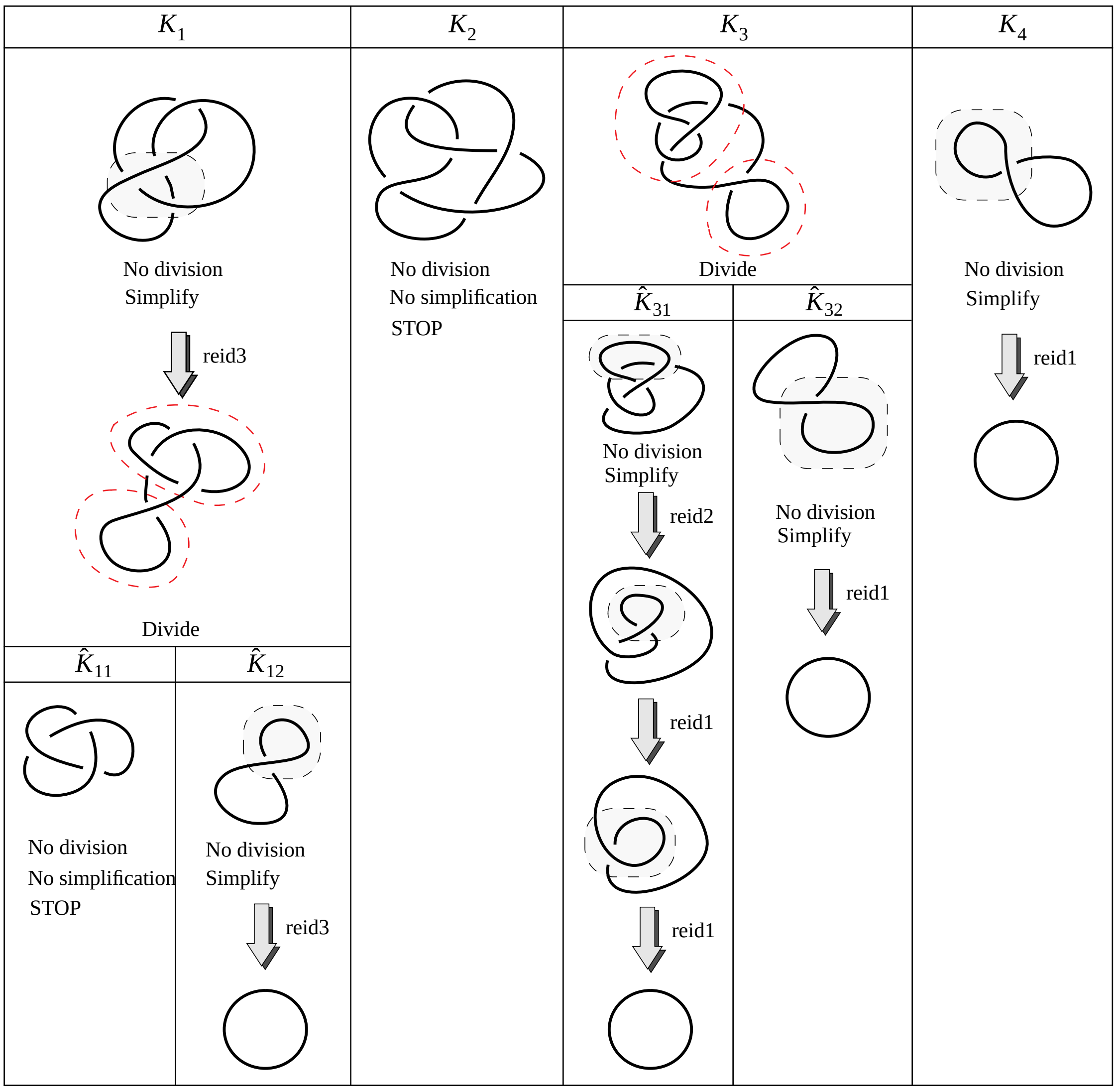


Figure 15. The Divide & Simplify procedure

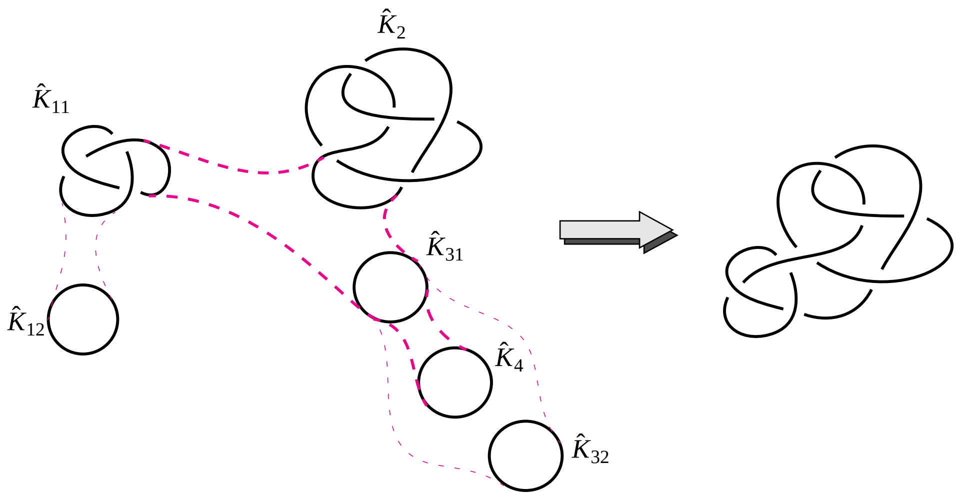


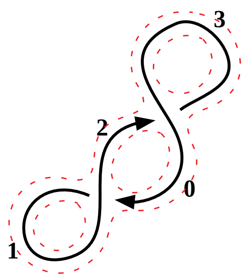
Figure 16. The Join procedure

In step 6 the knot K is perturbed so that there may be another division that is possible. A single knot move may be applied at this step, or a more complicated algorithm may also be used. Since the division into independent sections is very efficient in time (linear in the size of the knot), we should not attempt to obtain a complete simplification in step 6, but a partial one that either brings us to a knot that cannot be simplified (an irreducible knot) or may bring us to a knot from which we can continue to divide.

Independent sections are identified by invoking the procedure *DIVIDE* described below:

DIVIDE(K)

1. Build a 4-regular graph G from the knot representation of K, where each crossing is a vertex, and edges connect adjacent crossings. The graph is represented in adjacency list form.
2. Find a planar embedding of G, as an ordering of the edges of each vertex.
3. Using the planar embedding, extract a list F of all the *faces* of the embedding of G (see Figure 17). A face is represented by a set of segments.
4. $A = \varnothing$.
5. For each pair of faces f_i, f_j in F, if $|f_i \cap f_j| \geq 2$ then $A = A \cup (f_i \cap f_j)$, $F = F - \{f_i, f_j\}$.
6. A is a set of segments. Order the set A according to the numbering of the segments: $A = (a_1, a_2, \ldots, a_k)$ such that $a_1 < a_2 < \ldots < a_k$.
7. The set A partitions the set of crossings of the knot into independent sections. The set of crossings between a_i, a_{i+1} is an independent section for each $1 \leq i \leq k-1$ (the + is modulo the number of segments in the knot)



$$
\begin{aligned}
f_1 &= \langle 1 \rangle \\
f_2 &= \langle 2, 0 \rangle \\
f_3 &= \langle 3 \rangle \\
f_4 &= \langle 0, 1, 2, 3 \rangle \\
\\
A &= \{0, 2\}
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
K_1 &= [0, 1] \\
K_2 &= [2, 3]
\end{aligned}
$$

Figure 17. Example of the *DIVIDE* algorithm.

Since the graph G represents a knot diagram, it is always planar. We use a **linear** algorithm for embedding planar graphs using PQ-Trees [10][11]. Thus, the complexity of the division algorithm is linear in the number of crossings in the knot, and quadratic in the number of faces in the embedding found. Since the number of faces is at most the number of crossings times 2, the algorithm *DIVIDE* may be quadratic in the number of crossings, but in many cases the number of faces is significantly smaller than the maximum number.

In step 7 of the procedure *SIMPLIFY* we attempt to check if a given knot diagram is *irreducible.* We cannot always give an answer to this question, but in some cases which appear relatively frequently we can. The procedure *IRREDUCIBLE* makes use of the following famous theorem from knot theory:

Theorem 4 *(Tait Conjectures):*

If a knot K has a reduced alternating diagram (that is one that has no applicable Reid1 or Reid2 moves) then no other diagram of K has fewer crossings, all other reduced alternating diagrams of K have the same number of crossings, and furthermore, if K is prime then every non-alternating diagram of K has more crossings. (Proved by L. Kauffman, K Murasugi and M. Thistlethwaite)

If we have reached a reduced alternating diagram, then Theorem 4 implies that we may stop the simplification process for the diagram. Checking that a diagram is alternating is trivial from our knot representation (the polarity of crossing pairs alternate). In practice, most knots which are supplied to the *irreducibility* test procedure are not alternating, although if the division process succeeds in breaking the knot into small pieces the chance of encountering an alternating diagram grows. The *IRREDUCIBLE* procedure below is the real stopping condition of the algorithm, and the constant *MAX-ITTERATIONS* exists only because the algorithm for deciding irreducibility is not always applicable. Thus, improving the algorithm for deciding irreducibility would be an important, further contribution to the simplification algorithm.

```
IRREDUCIBLE(K)

1.  If K is alternating and there are no applicable Reid1 of Reid2 simplification moves, then return
    “irreducible”.
2.  else return “N/A”
```

Since we have an efficient algorithm for deciding if a diagram is apparently prime, we may derive an even stronger result from the last statement of theorem 4 using the following theorem, [14].

Theorem 5 *(W. Menasco):*

An alternating diagram represents a prime knot iff it is apparently prime.

Conclusion

TOK is a software framework on which many knot manipulation algorithms may be implemented. Two examples of such algorithms have been described. The knot simplification algorithms using TOK proved to be effective at simplifying very complicated knots. Further applications of the tool include building a tabulation of knot types by deciding knot equivalence with algorithms built on the TOK framework (e.g. as in [13]). The rule definition that has been presented can describe a wide range of manipulations. We note that there are knot algorithms (such as computation of the HOMFLY polynomial) which require knot manipulations that cannot be represented by our current syntax and application algorithm. Such moves are characterized by cutting and pasting segments in a local area in some predefined order. However, it is not hard to expand the TOK vocabulary, and algorithms to deal with these types of moves.

References

[1] C. C. Adams, *The Knot Book*, Freeman, New York 1994

[2] L. H. Kauffman, *On Knots*, Annals of Mathematics Studies 115, Princeton University Press, Princeton 1987.

[3] R.H Fox. *A Quick Trip Through Knot Theory.* Topology of 3-Manifolds (M.K. Fort Jr. , ed.), Prentice-Hall, Englewood Cliffs, N.J. 1962.

[4] G. Burde, H. Zieschang, *Knots*, de Gruyter Studies in Mathematics 5, Walter de Gruyter, Berlin-New York, 1985.

[5] C. Livingston, *Knot Theory*, The Carus Mathematical Monographs, The Mathematical Association of America, 1993.

[6] J.W. Alexander, G.B. Briggs, *On Types of Knotted Curves,* Ann. of Math **(2) 28** (1927) 562-586.

[7] P. Freyd, D. Yetter, J. Hoste, W. B. R. Lickorish, K. Millet, and A. Ocneanu, *A New Polynomial Invariant of Knots and Links*, Bull. American Math. Soc. **12** (1985), 239-246.

[8] S. Fukuhara, *Energy of a Knot*, A Fete of Topology, Academic Press, 1988.

[9] T. Ligocki, J.A. Sethian, *Recognizing Knots Using Simulated Annealing*, Journal of Knot Theory and its Ramifications, Vol. 3, No. 4, 1994.

[10] K. S. Booth, G. S. Lueker, *Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms*, Journal of Computer and System Sciences **13**, 335-379 (1976).

[11] S. Abe, T. Ozawa, *A Linear Algorithm for Embedding Planar Graphs Using PQ-Trees*, Journal of Computer and System Sciences **30**, 54-76 (1985).

[12] K. Reidemeister, *Knotentheorie*, Ergebnisse der Mathematik und ihrer Grenzgebiete, (Alte Folge), B 1, H 1.

[13] C. H. Dowker, Morwen B. Thistlethwaite, *Classification of Knot Projections*, Topology and its Applications 16 (1983) 19-31, North-Holland.

[14] W. Menasco, *Closed Incompressible Surfaces in Alternating Knot and Link Complement*s, Topology, Vol. 23, No. 1, 37-44, 1984.

Appendix A: TOK Syntax

Knot

A knot is defined by its segment pairs representation (See “Knot Representation”).

Syntax:

```
knot <identifier> [seg,seg][seg,seg]...;
```

Example:

```
knot trefoil [3,0][1,4][5,2];
```

Rule

A rule is defined by a description of the ‘pre’ and ‘post’ sides of the rule, using a variable for each of the threads that are part of the rule locality.

Variables may be used in the rule definition as part of simple arithmetic computations: subtraction and addition of constant integers or of other variables.

The left side of a rule (‘pre’) consists of one of the following items:

- Crossings list, represented by segments pairs.
- Segments, represented by ***seg(x)***.
- Adjacent segments, represented by ***adj(s1,s2)***.

The right side of a rule (‘post’) consists of one of the following items:

- Crossings list, represented by segments pairs.
- NULL, indicating no crossings on the ‘post’ side.

All the variables used in the right side of the rule must also exist in the left side rule.

Syntax:

```
rule <identifier> <before> -> <after>;
```

Examples:

```
rule reid3 [b,a][b+1,c-1][a+1,c-2] -> [a,c-1][b+1,a+1][b,c-2];

rule reid1 [xx,xx+1] -> NULL;

rule reid1r seg(s) -> [s,s+1];

rule comp adj(r,t) -> [r,t][r+1,t+1];
```

Commands

Commands may be used in the TOK shell to activate it's various features.
Each command must end with a semicolon (;).
A command can be any of the commands in the following list or an assignment.
Any executed command sets a global boolean variable called *status*, which holds the exit status of the last executed command. The status value can be saved in a different boolean variable by assigning it's value directly after the command's execution.
The commands are:

- `apply <rule_id> <knot_id>;`

  Apply a specific rule to a knot, if applicable. Exit status will be 'True' if the rule was actually applied 'False' otherwise.

- `apply oneof(<rule_id> [<rule_id> [...]]) <knot_id>;`

  Apply the rules in the list in a random order until one is successful, or all have been tried. Exit status will be 'True' if a rule was actually applied 'False' otherwise.

- `anneal <knot_id> with <rule_id> [<rule_id> [<rule_id> [...]]];`

  Start a simulated annealing process over a given knot with a set of rules. Exit status is 'True'.

- `display [<id> [<id> [...]]];`

  Display the value of any identifer(s). The value will be displayed according to the identifier's type. If no identifier is given as a parameter, all the currently defined identifiers will be displayed. Exit status is 'True'.

- `macro <macro_file>;`

  Execute commands from a predefined TOK shell script file. Exit status is set according to the status of the last executed command in the macro. Recursive calls are allowed.

- `echo "<any string>"`

  Echo strings to the standard output. Exit status is 'True'.

- `clear [<id> [<id> [...]]];`

  Clear existing identifiers (in order to free memory). If no identifier is given as a parameter, all identifiers will be cleared. Exit status is 'True';

- `verbose;`

  Run the TOK shell in verbose mode, displaying the new value of any variable that changed as a result of a command execution.

- `quiet;`

  Run the TOK shell in quiet mode, displaying nothing except the output of 'display' or 'echo' commands.

- `default;`

  Run the TOK shell in default mode, displaying selective information after the execution of some of the commands, that shows the knot(s) manipulation progress.

Boolean Variables and Logical Expressions

Any identifier that is being used during a TOK session, that was not defined as a knot or a rule identifier, is regarded as a boolean variable, who's initial setting is '***True***'.
A boolean variable may alter it's value by an assignment (See "Variables Assignment").
Logical expressions are made up of the basic logical operations: ***and***, ***or***, ***not***, and boolean variables.
Operator precedence is: ***not, and, or***. The precedence may be controlled by using parentheses.

Syntax:

```
and:   var1 && var2

or:    var1 || var2

not:   !var
```

Example:

```
!((a || b) && (c || d))
```

Variables Assignment

Assignment of variables allows duplication of any identifier entity such as a rule, a knot or a boolean variable.
By assigning a value to a variable, it inherits both the assigned value and it's type.
A variable may be assigned one of the following:

- Another identifier of any type.
- A boolean value: 'True' or 'False'.
- Boolean expression, in which case the logical expression will be computed first.

Syntax:

```
id1 = id2;

id = True; id = False;

id = <boolean expression>;
```

Examples:

```
saved_rule = reid1;

new_knot = knot1;

knot1 = True; (knot1 will change its type to Boolean)

x = a && (b || c);
```

Flow Control

Flow control is obtained by use of *if* and *while* sequences, based on the computed value of boolean expressions, or with the *repeat* construct. If the 'if', 'while', or 'repeat' keywords are used in an interactive shell, the TOK regular prompt (*tok* =>) is replaced with the 'waiting' prompt (*tok ?*) until the if or while body is completed.
Commands body that consists of more than a single command, should be enclosed in braces. For a single command

body, the braces are optional.

Syntax:

```
if (<boolean expression) command;

if (<boolean expression>) {
   <command>;
  [<command>;]
  [ ... ]
}

while (<boolean expression>) command;

while (<boolean expression>) {
   <command>;
  [<command>;]
  [ ... ]
}

repeat (lower, upper) command;

repeat (lower, upper) {
   <command>;
  [<command>;]
  [ ... ]
}
```

The lower and upper parameters of a repeat construct are the bounds on the number of times the block is executed. A number is chosen at random in the range each time the repeat construct is encountered.

Controlling the Annealing Parameters

A number of parameters used for annealing may be assigned an arbitrary value, other than their default value. The new value is set by a simple variable assignment, prior to the anneal command execution. These parameters are:

- **`MI`** - *Maximal number of Iterations*: Maximal number of iterations that are allowed for the annealing process. The annealing process will end if the temperature falls below the minimal temperature, or if MI iterations were completed. MI should be assigned an integer value.
- **`CRI`** - *Cooling Restart Interval*: The interval at which the cooling process is restarted (See “Cooling” on page 8). CRI should be assigned an integer value.
- **`IT`** - *Initial Temperature*: The temperature at the beginning of the annealing process. IT should be assigned a floating point value.
- **`MT`** - *Minimal Temperature*: The temperature at which the annealing process is stopped. MT should be assigned a floating point value.

Example

```
CRI = 100;
MI = 20;
anneal k with reid1 reid2 reid3 reid1r reid2r;
```

Comments

Comments may be placed in a macro file for documentation or for disabling some of it’s commands.
A comment is indicated by ‘#’. Anything written in the line after the ‘#’ sign is ignored.

TOK Command Line Options

- **`-s <num>`** : sets the random seed to <num>. This is useful in order to recreate the exact same scenarios as in a previous TOK session.
- **`-v`** : starts TOK in verbose mode.
- **`-q`** : starts TOK in quiet mode.

Appendix B: The 'init.tok' File

When the TOK shell is started, a macro called 'init.tok' is searched for in the current directory. If found, it will be run. Following is an example of a default init.tok file, including the definition of the various Reidemeister moves.

```
# Reidemeister Moves:

# Simplification Moves
rule reid1a [a,b][a+1, b+1] -> NULL;
rule reid1b [b+1, a][b, a+1] -> NULL;

rule reid2a [a, a+1] -> NULL;
rule reid2b [a+1, a] -> NULL;

# Weak Transformation Moves:
rule reid3a [b,a][a+1,c+1][b+1,c] -> [a,c][b,c+1][b+1,a+1];
rule reid3b [b,a+1][a,c+1][b+1,c] -> [a+1,c][b,c+1][b+1,a];
rule reid3c [b+1,a][a+1,c+1][b,c] -> [a,c][b+1,c+1][b,a+1];
rule reid3d [b,a][a+1,c][b+1,c+1] -> [a,c+1][b,c][b+1,a+1];
rule reid3e [b+1,a+1][a,c+1][b,c] -> [a+1,c][b+1,c+1][b,a];
rule reid3f [b,a+1][a,c][b+1,c+1] -> [a+1,c+1][b,c][b+1,a];
rule reid3g [b+1,a][a+1,c][b,c+1] -> [a,c+1][b+1,c][b,a+1];
rule reid3h [b+1,a+1][a,c][b,c+1] -> [a+1,c+1][b+1,c][b,a];

rule reid4a [a,b][a+1,c+1][b+1,c] -> [a,c][b,c+1][a+1,b+1];
rule reid4b [a+1,b][a,c+1][b+1,c] -> [a+1,c][b,c+1][a,b+1];
rule reid4c [a,b+1][a+1,c+1][b,c] -> [a,c][b+1,c+1][a+1,b];
rule reid4d [a,b][a+1,c][b+1,c+1] -> [a,c+1][b,c][a+1,b+1];
rule reid4e [a+1,b+1][a,c+1][b,c] -> [a+1,c][b+1,c+1][a,b];
rule reid4f [a+1,b][a,c][b+1,c+1] -> [a+1,c+1][b,c][a,b+1];
rule reid4g [a,b+1][a+1,c][b,c+1] -> [a,c+1][b+1,c][a+1,b];
rule reid4h [a+1,b+1][a,c][b,c+1] -> [a+1,c+1][b+1,c][a,b];

# Weak Complication Moves:
rule reid5a seg(a) -> [a,a+1];
rule reid5b seg(a) -> [a+1,a];

# Strong Complication Moves:
rule reid6a adj(a,b) -> [a,b][a+1,b+1];
rule reid6b adj(a,b) -> [a,b+1][a+1,b];
```

Appendix C

The actual knot shown in Figure 10