Benchmarking Algorithms for the Jones Polynomial

Evan Ashoori, Peter Bai, Hansen Shieh

Mentor: Ryan Maguire

October 18, 2025

Background: Knots and Knot Diagrams

Definition (Knot)

A *knot* is a closed loop in $\mathbb{R}^3$.

Background: Knots and Knot Diagrams

Definition (Knot)

A *knot* is a closed loop in $\mathbb{R}^3$.

Definition (Knot Diagrams)

A *knot diagram* is a projection of a knot into the plane.

Background: Knots and Knot Diagrams

Definition (Knot)

A *knot* is a closed loop in $\mathbb{R}^3$.

Definition (Knot Diagrams)

A *knot diagram* is a projection of a knot into the plane.

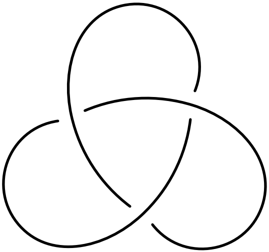


Figure: Knot diagram of the trefoil knot

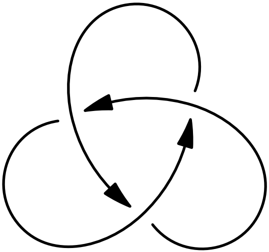


Figure: Oriented knot diagram of the trefoil knot

Definition (Equivalence)

We call two knot diagrams *equivalent* if one deform one into the other without breaking any part of the knot.

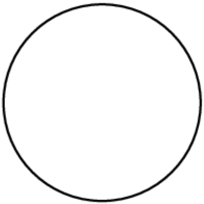


Figure: The unknot

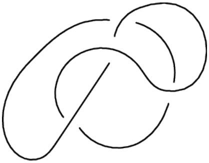


Figure: The unknot, but twisted

Background: Timeline of Knot Tabulation

- 165 different 10-crossing prime knots (late 1800s)

Background: Timeline of Knot Tabulation

- 165 different 10-crossing prime knots (late 1800s)
- $9,988$ different 13-crossing prime knots (early 1980s)

Background: Timeline of Knot Tabulation

- 165 different 10-crossing prime knots (late 1800s)
- $9,988$ different 13-crossing prime knots (early 1980s)
- $1,388,705$ different 16-crossing prime knots (1998)

Background: Timeline of Knot Tabulation

- 165 different 10-crossing prime knots (late 1800s)
- $9,988$ different 13-crossing prime knots (early 1980s)
- $1,388,705$ different 16-crossing prime knots (1998)
- $294,130,458$ different 19-crossing prime knots (2019)

Background: Timeline of Knot Tabulation

- 165 different 10-crossing prime knots (late 1800s)
- $9,988$ different 13-crossing prime knots (early 1980s)
- $1,388,705$ different 16-crossing prime knots (1998)
- $294,130,458$ different 19-crossing prime knots (2019)
- $1,847,319,428$ different 20-crossing prime knots (2025)

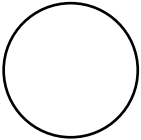
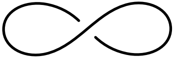
Background: Knot Invariants

Definition (Knot Invariants)

A *knot invariant* is a function of a knot diagram that is the same for any two equivalent knot diagrams.

Definition (Knot Invariants)

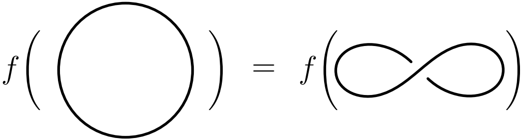
A *knot invariant* is a function of a knot diagram that is the same for any two equivalent knot diagrams.

$f\Big($  $\Big) = f\Big($  $\Big)$

Background: Knot Invariants

Definition (Knot Invariants)

A *knot invariant* is a function of a knot diagram that is the same for any two equivalent knot diagrams.

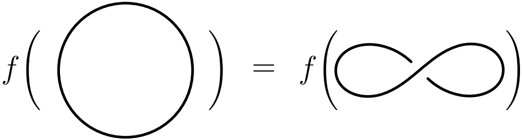


Examples include:

Background: Knot Invariants

Definition (Knot Invariants)

A *knot invariant* is a function of a knot diagram that is the same for any two equivalent knot diagrams.



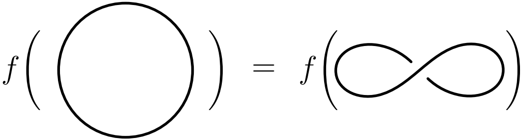
Examples include:

- Tricolorability

Background: Knot Invariants

Definition (Knot Invariants)

A *knot invariant* is a function of a knot diagram that is the same for any two equivalent knot diagrams.



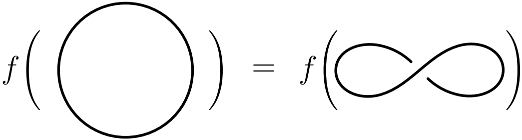
Examples include:

- Tricolorability
- Alexander Polynomial

Background: Knot Invariants

Definition (Knot Invariants)

A *knot invariant* is a function of a knot diagram that is the same for any two equivalent knot diagrams.



Examples include:

- Tricolorability
- Alexander Polynomial
- Jones Polynomial

Background: Links

Definition (Link)

A *link* is a collection of loops in $\mathbb{R}^3$.

Background: Links

Definition (Link)

A *link* is a collection of loops in $\mathbb{R}^3$.

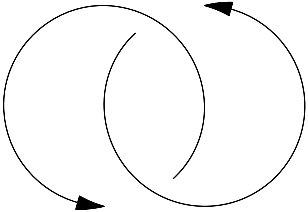


Figure: Projection of the Hopf Link

Kauffman Bracket

Definition (Kauffman Bracket)

The *Kauffman bracket polynomial* (KBP) of a link is defined by:

- $\langle \bigcirc \rangle = 1$

Kauffman Bracket

Definition (Kauffman Bracket)

The *Kauffman bracket polynomial* (KBP) of a link is defined by:

- $\langle \bigcirc \rangle = 1$
- $\langle \bigcirc\!\!\!\!\!\!\bigcirc \rangle = (-A^2 - A^{-2})\langle \bigcirc \rangle$

Kauffman Bracket

Definition (Kauffman Bracket)

The *Kauffman bracket polynomial* (KBP) of a link is defined by:

- $\langle$  $\rangle = 1$
- $\langle$  $\rangle = (-A^2 - A^{-2})\langle$  $\rangle$
- $\langle$  $\rangle = A\langle$  $\rangle + A^{-1}\langle$  $\rangle$

Kauffman Bracket

Definition (Kauffman Bracket)

The *Kauffman bracket polynomial* (KBP) of a link is defined by:

- $\langle$  $\rangle = 1$
- $\langle$  $\rangle = (-A^2 - A^{-2})\langle$  $\rangle$
- $\langle$  $\rangle = A\langle$  $\rangle + A^{-1}\langle$  $\rangle$

The Kauffman Bracket polynomial is *not* an invariant, *but* it's only ever off by a power of $(-A)^3$, and we can correct this with something called the *writhe*.

Writhe

Definition (Sign of Crossing)

The *sign* of a crossing is defined as shown below:

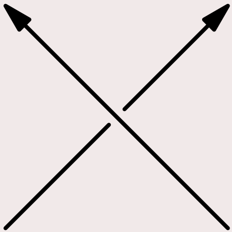


Figure: A negative (-1) crossing

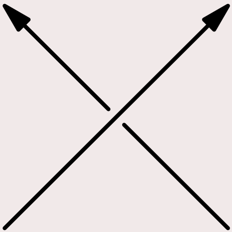


Figure: A positive (+1) crossing

Writhe

Definition (Sign of Crossing)

The *sign* of a crossing is defined as shown below:

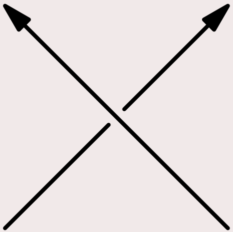


Figure: A negative (-1) crossing

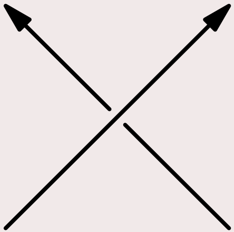


Figure: A positive (+1) crossing

Definition (Writhe)

The *writhe* of a link L, or $w(L)$, is the sum of the signs of all of the crossings of L.

Jones Polynomial: Definition

Definition (Jones Polynomial)

The *normalized bracket polynomial* of an oriented link L is given by $X(L) = (-A^3)^{-w(L)}\langle L \rangle$. The *Jones Polynomial* of L, denoted by $V(L)$, is obtained by setting $A = t^{-1/4}$.

- One of the most widely recognized knot invariants

Jones Polynomial: Definition

Definition (Jones Polynomial)

The *normalized bracket polynomial* of an oriented link L is given by $X(L) = (-A^3)^{-w(L)}\langle L \rangle$. The *Jones Polynomial* of L, denoted by $V(L)$, is obtained by setting $A = t^{-1/4}$.

- One of the most widely recognized knot invariants
- With another invariant, used to tabulate knots up to 20 crossings

Jones Polynomial: Goals

- Naive implementation for a knot with n crossings is $O(n2^n)$
- Computation can take hours or days as knots grow in size
- Faster/sub-exponential algorithms will aid in tabulating 21-crossing prime knots.

Tangles: Definitions

Definition (Tangle Diagram)

A *tangle diagram* is a section of a knot/link diagram embedded in a disk.

Example

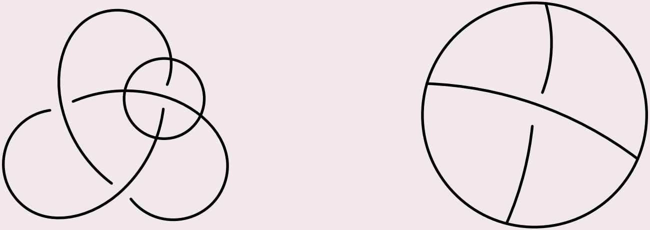


Figure: A tangle with a crossing from the trefoil

Tangles: Definitions

Definition

Let T be a tangle. The *type* of T consists of the disc, along with the points on the boundary of T. A tangle is a *base tangle* if it contains no crossings or loops.

Tangles: Definitions

Definition

Let T be a tangle. The *type* of T consists of the disc, along with the points on the boundary of T. A tangle is a *base tangle* if it contains no crossings or loops.

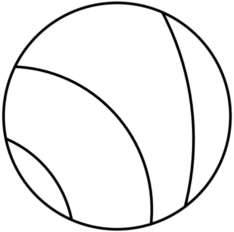


Figure: A base tangle with 6 boundary points

Tangles: Kauffman Bracket Polynomial

- Base tangles let us extend the KBP to tangles

Tangles: Kauffman Bracket Polynomial

- Base tangles let us extend the KBP to tangles
- For tangle on previous slide, $\langle$  $\rangle = A\langle$  $\rangle + A^{-1}\langle$  $\rangle$

Tangles: Kauffman Bracket Polynomial

- Base tangles let us extend the KBP to tangles
- For tangle on previous slide, $\langle$  $\rangle = A\langle$  $\rangle + A^{-1}\langle$  $\rangle$
- If a tangle has many crossings, we can first find the KBP of smaller tangles within, and glue the base tangles back in

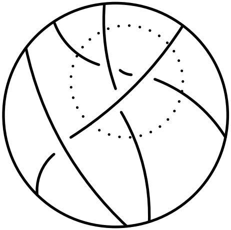


Figure: A smaller tangle with 3 crossings contained in a larger one

Tangles: Cutting

Let L be a link with n crossings. A *cutting* of L is a sequence of $n+1$ tangles T_0, T_1, ..., T_n such that:

- T_0 is empty
- T_n is L embedded in a disk
- For $0 \leq i < n$, the graph $T_{i+1} - T_i$ consists only of a crossing and the edges connecting the boundary of T_i to T_{i+1}

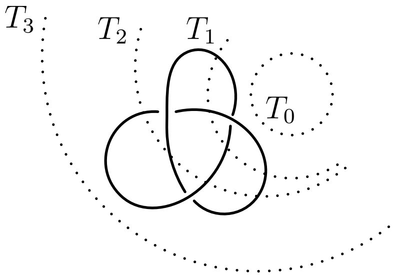


Figure: A cutting of the trefoil knot

Tangles: Algorithm

Algorithm (Ellenberg et al., 2013)

Input: A link L with n crossings and a cutting T_0, T_1, ..., T_n of L
Output: The Jones polynomial of L

- Start with empty tangle T_0

Tangles: Algorithm

Algorithm (Ellenberg et al., 2013)

Input: A link L with n crossings and a cutting T_0, T_1, ..., T_n of L
Output: The Jones polynomial of L

- Start with empty tangle T_0
- For $i = 0$ to $i = n - 1$:

Tangles: Algorithm

Algorithm (Ellenberg et al., 2013)

Input: A link L with n crossings and a cutting T_0, T_1, ..., T_n of L
Output: The Jones polynomial of L

- Start with empty tangle T_0
- For $i = 0$ to $i = n - 1$:
  - Take the KBP of T_i, expressed in terms of the base tangles

Tangles: Algorithm

Algorithm (Ellenberg et al., 2013)

Input: A link L with n crossings and a cutting T_0, T_1, ..., T_n of L
Output: The Jones polynomial of L

- Start with empty tangle T_0
- For $i = 0$ to $i = n - 1$:
  - Take the KBP of T_i, expressed in terms of the base tangles
  - Glue each base tangle into T_{i+1}

Tangles: Algorithm

Algorithm (Ellenberg et al., 2013)

Input: A link L with n crossings and a cutting T_0, T_1, ..., T_n of L
Output: The Jones polynomial of L

- Start with empty tangle T_0
- For $i = 0$ to $i = n - 1$:
  - Take the KBP of T_i, expressed in terms of the base tangles
  - Glue each base tangle into T_{i+1}
  - Remove the new crossing in T_{i+1} by using the rules for computing the KBP

Tangles: Algorithm

Algorithm (Ellenberg et al., 2013)

Input: A link L with n crossings and a cutting T_0, T_1, ..., T_n of L
Output: The Jones polynomial of L

- Start with empty tangle T_0
- For $i = 0$ to $i = n - 1$:
  - Take the KBP of T_i, expressed in terms of the base tangles
  - Glue each base tangle into T_{i+1}
  - Remove the new crossing in T_{i+1} by using the rules for computing the KBP
  - Obtain the KBP of T_{i+1} in terms of its base tangles

Tangles: Algorithm

Algorithm (Ellenberg et al., 2013)

Input: A link L with n crossings and a cutting T_0, T_1, ..., T_n of L
Output: The Jones polynomial of L

- Start with empty tangle T_0
- For $i = 0$ to $i = n - 1$:
  - Take the KBP of T_i, expressed in terms of the base tangles
  - Glue each base tangle into T_{i+1}
  - Remove the new crossing in T_{i+1} by using the rules for computing the KBP
  - Obtain the KBP of T_{i+1} in terms of its base tangles
- Compute the Jones polynomial of L using the *KBP* of T_n and the writhe of L

Tangles: Performance

Theorem (Djidjev and Vrťo, 2003)

Given a cutting of a knot with n crossings, the number of boundary vertices in any tangle of the cutting is bounded above by $c\sqrt{n}$, where $c = 6\sqrt{2} + 5\sqrt{3}$.

Tangles: Performance

Theorem (Djidjev and Vrťo, 2003)

Given a cutting of a knot with n crossings, the number of boundary vertices in any tangle of the cutting is bounded above by $c\sqrt{n}$, where $c = 6\sqrt{2} + 5\sqrt{3}$.

Proposition

Let T be a type with m boundary points. Up to isotopy equivalence, there are $C_{m/2}$ distinct base tangles with type T, where C_i is the ith Catalan number.

Tangles: Performance

Theorem (Djidjev and Vrťo, 2003)

Given a cutting of a knot with n crossings, the number of boundary vertices in any tangle of the cutting is bounded above by $c\sqrt{n}$, where $c = 6\sqrt{2} + 5\sqrt{3}$.

Proposition

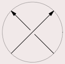
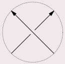
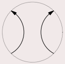
Let T be a type with m boundary points. Up to isotopy equivalence, there are $C_{m/2}$ distinct base tangles with type T, where C_i is the ith Catalan number.

Theorem (Ellenberg, Shieh, et al., 2025)

The time complexity for the tangle algorithm for a knot with n crossings is $O(n^{5/4}2^{c\sqrt{n}})$.

Definition (Jones)

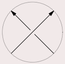
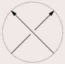
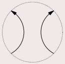
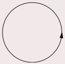
The Jones polynomial is the unique function satisfying

- $t^{-1}V$  $(t) - tV$  $(t) = (\sqrt{t} - \frac{1}{\sqrt{t}})V$  (t)

A Different Recursion for Jones

Definition (Jones)

The Jones polynomial is the unique function satisfying

- $t^{-1}V$  $(t) - tV$  $(t) = (\sqrt{t} - \frac{1}{\sqrt{t}})V$  (t)
- V  $(t) = 1$
- Notice that there are two terms with crossings in the above skein relation, so we can't induct on crossing number as with the original skein relation.

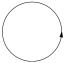
Unknots

- $V_{\bigcirc\, L}(t) = -(\sqrt{t} + \frac{1}{\sqrt{t}})V_L(t).$

Unknots

- $V_{\bigcirc \; L}(t) = -(\sqrt{t} + \frac{1}{\sqrt{t}})V_L(t).$
- Thus, we want to make all components into unknots.

Unknots

- V  $_{L}(t) = -(\sqrt{t} + \frac{1}{\sqrt{t}})V_L(t).$
- Thus, we want to make all components into unknots.

Proposition (Unknots)

If as one traverses the edges of a knot all crossings are first encountered as overcrossings, the knot is an unknot.

Unknots

- $V_{\bigcirc\ L}(t) = -(\sqrt{t} + \frac{1}{\sqrt{t}})V_L(t).$
- Thus, we want to make all components into unknots.

Proposition (Unknots)

If as one traverses the edges of a knot all crossings are first encountered as overcrossings, the knot is an unknot.

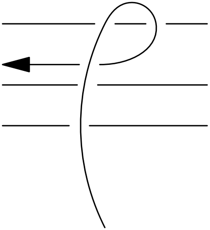


Figure: Part of an Unknot

Unknots

- We can decorate each overcrossing we encounter as we progress along each link component (and not skein decorated crossings) so that eventually we get all unknots. We can induct on the number of undecorated crossings instead of the number of crossings.

Unknots

- We can decorate each overcrossing we encounter as we progress along each link component (and not skein decorated crossings) so that eventually we get all unknots. We can induct on the number of undecorated crossings instead of the number of crossings.

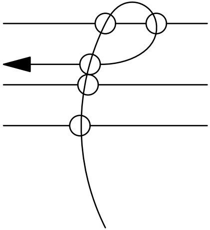


Figure: Decorated Part of Unknot

Skein Template Algorithm

Algorithm (Gouesbet, Meunier-Guttin-Cluzel, and Letellier, 1999)

Input: A link L, a starting crossing, and a starting direction.
Output: The Jones polynomial of L
Repeat the following procedure until I have only unlinks:

- If I encounter an overcrossing, decorate it (we won't touch it again).

Skein Template Algorithm

Algorithm (Gouesbet, Meunier-Guttin-Cluzel, and Letellier, 1999)

Input: A link L, a starting crossing, and a starting direction.
Output: The Jones polynomial of L
Repeat the following procedure until I have only unlinks:

- If I encounter an overcrossing, decorate it (we won't touch it again).
- If I encounter an undercrossing

Skein Template Algorithm

Algorithm (Gouesbet, Meunier-Guttin-Cluzel, and Letellier, 1999)

Input: A link L, a starting crossing, and a starting direction.
Output: The Jones polynomial of L
Repeat the following procedure until I have only unlinks:

- If I encounter an overcrossing, decorate it (we won't touch it again).
- If I encounter an undercrossing
  - Use the skein relation. Decorate the overcrossing made from switching the two strands (we won't touch it again).

Algorithm (Gouesbet, Meunier-Guttin-Cluzel, and Letellier, 1999)

Input: A link L, a starting crossing, and a starting direction.
Output: The Jones polynomial of L
Repeat the following procedure until I have only unlinks:

- If I encounter an overcrossing, decorate it (we won't touch it again).
- If I encounter an undercrossing
  - Use the skein relation. Decorate the overcrossing made from switching the two strands (we won't touch it again).
- Proceed to the next crossing in each case.

Time Complexity

Since each recursion splits the current link into two links with one fewer undecorated crossing and each step of the recursion takes polynomial time to compute, the worst case time complexity of the algorithm is $O^{\sim}(2^n)$.

Time Complexity

Since each recursion splits the current link into two links with one fewer undecorated crossing and each step of the recursion takes polynomial time to compute, the worst case time complexity of the algorithm is $O^{\sim}(2^n)$. In practice, we expect the algorithm to run in about $O^{\sim}(2^{\frac{n}{2}})$ since usually around half of all crossings are first encountered are undercrossings.

Load Balanced Algorithm

Definition (Kauffman Bracket Skein Relation)

$\langle$  $\rangle = A\langle$  $\rangle + A^{-1}\langle$  $\rangle$

Load Balanced Algorithm

Definition (Kauffman Bracket Skein Relation)

$\langle$  $\rangle = A\langle$  $\rangle + A^{-1}\langle$  $\rangle$

We can make the naive algorithm of recursively applying Skein relations much more efficient by:

Load Balanced Algorithm

Definition (Kauffman Bracket Skein Relation)

$\langle$  $\rangle = A\langle$  $\rangle + A^{-1}\langle$  $\rangle$

We can make the naive algorithm of recursively applying Skein relations much more efficient by:

- Performing immediate simplifications between iterations

Load Balanced Algorithm

Definition (Kauffman Bracket Skein Relation)

$$\langle \quad \rangle = A\langle \quad \rangle + A^{-1}\langle \quad \rangle$$

We can make the naive algorithm of recursively applying Skein relations much more efficient by:

- Performing immediate simplifications between iterations

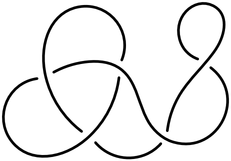


Figure: Trefoil with some twists

Load Balanced Algorithm

Definition (Kauffman Bracket Skein Relation)

$$\langle \quad \rangle = A\langle \quad \rangle + A^{-1}\langle \quad \rangle$$

We can make the naive algorithm of recursively applying Skein relations much more efficient by:

- Performing immediate simplifications between iterations

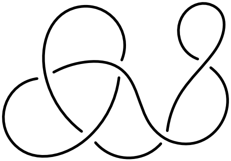


Figure: Trefoil with some twists

- Selecting crossings that yield immediate simplifications after applying a Skein relation

Load Balanced Algorithm

We can look for simple patterns with crossings that can be smoothed to remove *more* than one crossing at once:

Example

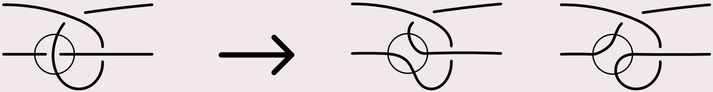


Figure: Smoothing a crossing

Load Balanced Algorithm

We can look for simple patterns with crossings that can be smoothed to remove *more* than one crossing at once:

Example

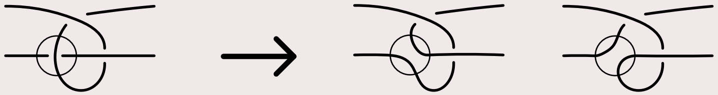


Figure: Smoothing a crossing

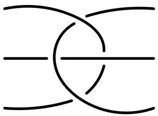


Figure: Triple

Load Balanced Algorithm

We can look for simple patterns with crossings that can be smoothed to remove *more* than one crossing at once:

Example

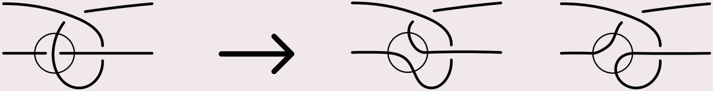


Figure: Smoothing a crossing

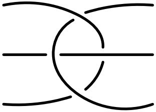


Figure: Triple

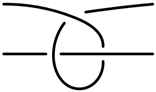


Figure: Gamma

Load Balanced Algorithm

We can look for simple patterns with crossings that can be smoothed to remove *more* than one crossing at once:

Example

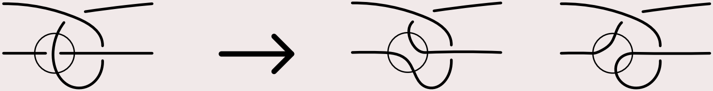


Figure: Smoothing a crossing

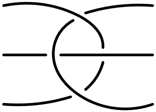


Figure: Triple

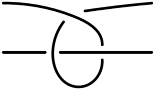


Figure: Gamma

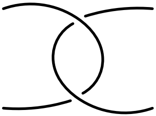


Figure: Bigon

Load Balanced Algorithm

Definition (Complexity Type)

Suppose that smoothing a crossing in a knot with n crossings and performing subsequent simplifications results in two knot diagrams with $n - a$ and $n - b$ crossings. Then, this smoothing has complexity type (a, b).

Load Balanced Algorithm

Definition (Complexity Type)

Suppose that smoothing a crossing in a knot with n crossings and performing subsequent simplifications results in two knot diagrams with $n-a$ and $n-b$ crossings. Then, this smoothing has complexity type (a, b).

Example

Smoothing the circled crossing has a complexity type of $(1, 2)$:

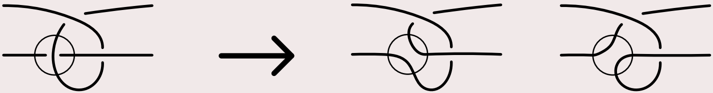


Figure: Smoothing a crossing in a gamma

Load Balanced Algorithm

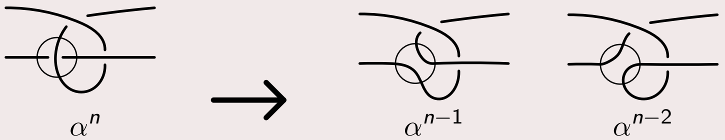
Estimate the time complexity of the algorithm as $\Theta(\alpha^n)$ for a knot diagram with n crossings and a hard-coded constant α

Load Balanced Algorithm

Estimate the time complexity of the algorithm as $\Theta(\alpha^n)$ for a knot diagram with n crossings and a hard-coded constant α

Example

Smoothing the circled crossing has a complexity type of $(1, 2)$:

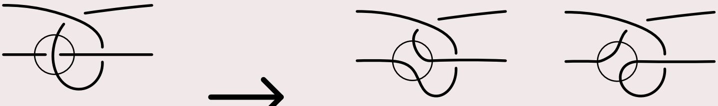


Load Balanced Algorithm

Estimate the time complexity of the algorithm as $\Theta(\alpha^n)$ for a knot diagram with n crossings and a hard-coded constant α

Example

Smoothing the circled crossing has a complexity type of $(1, 2)$:



α^n α^{n-1} α^{n-2}

Example (Comparing Complexity Types)

For $\alpha = 1.75$, we estimate a smoothing with complexity type $(2, 2)$ to be better than a smoothing with complexity type $(1, 4)$ because

$$\alpha^{-2} + \alpha^{-2} \approx 0.65 < 0.68 \approx \alpha^{-1} + \alpha^{-4}.$$

Load Balanced Algorithm

Algorithm (Ewing and Millet, 1991)

Input: A link L

Output: The KBP of L

- If L is the unknot, return 1.
- If L has a component with no crossings, get rid of this component and multiply the result by $-A^2 - A^{-2}$.

Load Balanced Algorithm

Algorithm (Ewing and Millet, 1991)

Input: A link L

Output: The KBP of L

- If L is the unknot, return 1.
- If L has a component with no crossings, get rid of this component and multiply the result by $-A^2 - A^{-2}$.
- Otherwise, search for patterns and their corresponding potential smoothings

Load Balanced Algorithm

Algorithm (Ewing and Millet, 1991)

Input: A link L
Output: The KBP of L

- If L is the unknot, return 1.
- If L has a component with no crossings, get rid of this component and multiply the result by $-A^2 - A^{-2}$.
- Otherwise, search for patterns and their corresponding potential smoothings
- Select the smoothing with complexity type (a, b) that minimizes $\alpha^{-a} + \alpha^{-b}$

Load Balanced Algorithm

Algorithm (Ewing and Millet, 1991)

Input: A link L

Output: The KBP of L

- If L is the unknot, return 1.
- If L has a component with no crossings, get rid of this component and multiply the result by $-A^2 - A^{-2}$.
- Otherwise, search for patterns and their corresponding potential smoothings
- Select the smoothing with complexity type (a, b) that minimizes $\alpha^{-a} + \alpha^{-b}$
- Evaluate the Skein relation and recurse

Benchmarking

Preliminary Results

Algorithm	Time (seconds)
Naive	33.819
Circle Counting	13.394
Skein Template Algorithm	1.611
Symbolic Calculus	0.121
Tangle Algorithm	0.052

Figure: Benchmarks on a random sample of 24-crossing knots

Benchmarking

Preliminary Results

Algorithm	Time (seconds)
Naive	33.819
Circle Counting	13.394
Skein Template Algorithm	1.611
Symbolic Calculus	0.121
Tangle Algorithm	0.052

Figure: Benchmarks on a random sample of 24-crossing knots

Future Work

- Implement and benchmark two additional algorithms

Benchmarking

Preliminary Results

Algorithm	Time (seconds)
Naive	33.819
Circle Counting	13.394
Skein Template Algorithm	1.611
Symbolic Calculus	0.121
Tangle Algorithm	0.052

Figure: Benchmarks on a random sample of 24-crossing knots

Future Work

- Implement and benchmark two additional algorithms
- Tabulate the Jones polynomial for all prime knots up to 20 crossings

Acknowledgments

- We would like to thank MIT PRIMES for providing us this opportunity.

Acknowledgments

- We would like to thank MIT PRIMES for providing us this opportunity.
- We would like to thank our mentor, Ryan Maguire, for teaching us knot theory, programming in C, and helping us at every step of our research.

Acknowledgments

- We would like to thank MIT PRIMES for providing us this opportunity.
- We would like to thank our mentor, Ryan Maguire, for teaching us knot theory, programming in C, and helping us at every step of our research.
- We would like to thank Kenta Suzuki for giving us feedback for our presentation.