COMPUTING FINITE TYPE INVARIANTS EFFICIENTLY

Abstract. We describe an efficient algorithm to compute finite type invariants of type k by first creating, for a given knot K with n crossings, a look-up table for all subdiagrams of K of size $\lceil \frac{k}{2} \rceil$ indexed by dyadic intervals in $[0, 2n-1]$. Using this algorithm, any such finite type invariant can be computed on an n-crossing knot in time $\sim n^{\lceil \frac{k}{2} \rceil}$, a lot faster than the previously best published bound of $\sim n^k$.

1. Introduction

Finite type invariants, also known as Vassiliev invariants [Vas90, Vas92], underlie many of the classical knot invariants, for instance they include the coefficients of the Jones, Alexander, and more generally HOMFLY-PT polynomials [BL93, BN95]. A knot invariant ζ is said to be of *finite type* k if it vanishes on all knots with at least $k+1$ double points, where ζ is extended to knots with double points by the formula:

$$\zeta(\times) = \zeta(\rightleftarrows) - \zeta(\rightrightarrows).$$

For example, the linking number of a two-component knot is a finite type invariant of type 1. In our main theorem, we provide an algorithm to compute finite type invariants from a planar projection of a knot in a surprisingly efficient time depending on the crossing number of the knot diagram.

Main Theorem. *Finite type invariants of type k can be computed on an n-crossing knot in time at most $\sim n^{\lceil k/2 \rceil}$.*

This is a surprising result as before this theorem, the fastest algorithm (known to the authors) to compute a type k invariant on a knot diagram with n crossings took time $\sim n^k$ [BNBNHS23], and it was commonly believed that this was the fastest possible. There are specific finite type invariants which can be computed much faster, such as the linking number and the coefficients of the Alexander polynomial, but these are special cases. The Man Theorem gives the current fastest known algorithm that works for *all* finite type invariants, and shows that the computational time can be reduced to roughly the square root of the previously fastest known algorithm. In a previous paper [BNBNHS23], we proved that finite type invariants can be computed efficiently using 3D methods. We argued that most knot invariants, including finite type invariants, should be more efficiently computed using 3D methods rather than 2D methods. However, the Main Theorem is significant as it is a 2D method that currently outperforms all known 3D methods to compute finite type invariants.

For complexity measurements in this paper, we measure only polynomial degree (i.e. we ignore constants and $\log(n)$ terms). We write $f(n) \sim g(n)$ to mean there exist natural numbers c, k, N so that for all $n > N$, we have that

$$\frac{1}{c} g(n)(\log(n))^{-k} < f(n) < c g(n)(\log(n))^k.$$

For example, for us, $n^4 \sim 5.4\ n^4(\log(n))^8$. Also, we write $g \gg f$ to mean that for all constants c and k, and for all large enough n we have that $cf(n)(\log(n))^k < g(n)$.

Key words and phrases. Finite type invariants, Gauss diagrams.

For impatient readers, the key formula in this paper is Equation (3). [1] The preceding pages include the definitions leading up to the formula, and the proof that the formula can be evaluated in time $\sim n^{\lceil k/2 \rceil}$.

Acknowledgements. The first author was supported by NSERC-RGPIN-2018-04350 and by the Chu Family Foundation (NYC). The third author was supported by the Natural Science Foundation Grant No. DMS-2302664. This material is also based upon work supported by the National Science Foundation under Grant No. DMS-1929284 while the fourth author was in residence at the Institute for Computational and Experimental Research in Mathematics in Providence, RI, during the Braids program. This project is partially sponsored by the Provost Office of Elon University. We would like to thanks ICERM for hosting the first and third authors for a week long visit.

2. Background

2.1. **Gauss diagrams.** A *Gauss diagram* of an n-crossing long knot diagram parametrized by the interval $I = (-1, 2n) \subseteq \mathbb{R}$ is given by the interval I along with n decorated arrows (equivalently, oriented perfect matchings of $2n$ points, where the arrows are the edges of the matching). Each arrow corresponds to one of the n crossings of the knot and has endpoints on integer points in I. The head of an arrow is at the point in I which parametrizes the lower strand of the crossing and the tail of the arrow is at the point which parametrizes the upper strand of the crossing. Each arrow is decorated with a sign corresponding to the sign of the crossing. Figure 1 (A) shows an example of a Gauss diagram. For a Gauss diagram D, the quantity $|D|$ is the number of arrows in D. Let $\mathcal{GD} = \langle \text{Gauss diagrams} \rangle$ denote the space of $\mathbb{Q}$-linear combinations of Gauss diagrams, and $\mathcal{GD}_k$ denote the subspace spanned by Gauss diagrams with k or fewer arrows.

Let D be a Gauss diagram with n arrows parametrized by $I = (-1, 2n)$. A *k-arrow subdiagram* of D is a diagram consisting of I and a subset of k decorated arrows from D. A subdiagram corresponds to a choice of k crossings in the knot diagram represented by the Gauss diagram D. An example is shown in Figure 1(B). Notice that a subdiagram D' of D keeps the original parametrization along the interval I and the $2k$ endpoints of D' will be spread out amongst the $2n$ points in I. A k-arrow subdiagram of D is *not* a proper Gauss diagram because of this parametrization issue. To make a k-arrow subdiagram D' into a Gauss diagram, we can apply the forgetful map ψ to D' which reparametrizes I to be $I = (-1, 2k)$, in essence forgetting how D' was realized as a subdiagram of D. We will call $\psi(D')$ a *reparametrized subdiagram* of D. An example is shown in Figure 1(C).

It will be useful within the proof of our main theorem to take two Gauss diagrams and superimpose them to make a larger diagram. Superimposing two diagrams requires not only the two input diagrams but gluing instructions for how to interweave the endpoints of the diagrams. For an interval I, we use the notation $I_{\mathbb{Z}}$ to denote $I \cap \mathbb{Z}$, the integer points of I. We will call a non-decreasing map on integers $\lambda : [0, 2\ell - 1]_{\mathbb{Z}} \to [0, 2k]_{\mathbb{Z}}$ a *placement map*, as it can be viewed as providing instructions on how to place one Gauss diagram relative to another when superimposing them. Given such a λ, a *superimposition map* $\#_\lambda : \mathcal{GD}_k \times \mathcal{GD}_\ell \to \mathcal{GD}_{k+\ell}$ creates a Gauss diagram with $k + \ell$ arrows given two diagrams D and D' of sizes $|D| = k$ and $|D'| = \ell$, as follows. Starting with the first diagram D parametrized by $I = (-1, 2k)$, enumerate the intervals between the $2k$ integral endpoints of D in increasing order, including $(-1, 0)$ as the 0-th interval and $(2k - 1, 2k)$ as the $2k$-th interval. The 2ℓ endpoints of D'

[1] As preliminary intuition, we note that the data structures used in our construction are similar but not quite the same as a hyperoctree.

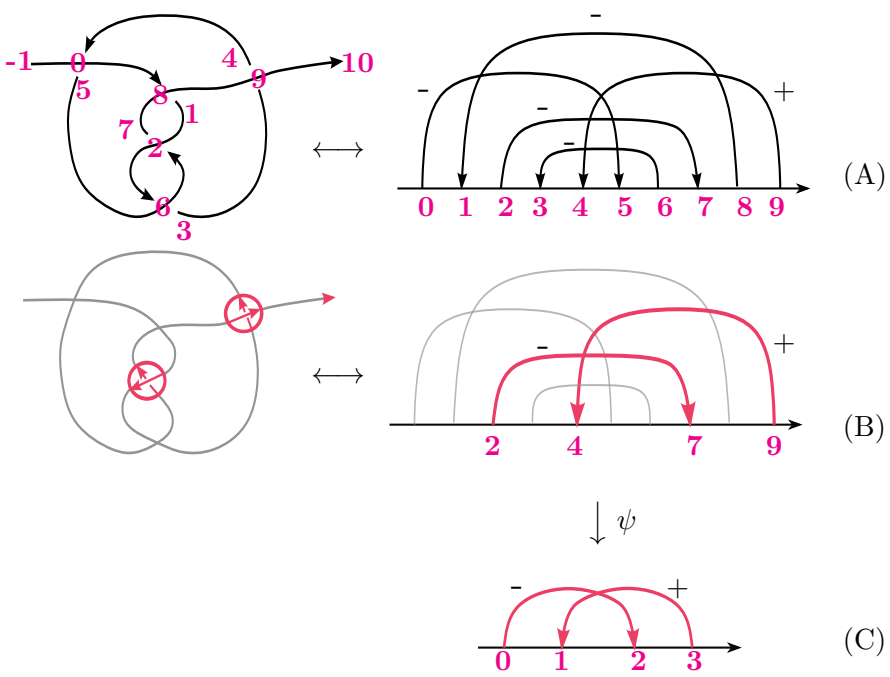

FIGURE 1. (A) An example of the Gauss diagram of a long knot diagram. (B) A 2-arrow subdiagram of a Gauss diagram. (C) The forgetful map ψ applied to a subdiagram yielding a reparametrized subdiagram.

are ordered and enumerated by $[0, 2\ell - 1]_{\mathbb{Z}}$. The function λ instructs how to place the 2ℓ endpoints of D' into the $2k + 1$ intervals of D. After the arrows of D' are glued into the intervals of D, the interval $I = (-1, 2k)$ is reparametrized to $(-1, 2(k + \ell))$ so that the resulting diagram $D\#_\lambda D'$ is a Gauss diagram. Two examples are shown in Figures 2(A) and 2(B). Superimposing two diagrams according to different placement maps λ can yield different outputs.

We denote by $\varphi_k : \{\text{knot diagrams}\} \to \mathcal{GD}_k$ the map which sends a knot diagram to the sum of all of the *reparametrized* subdiagrams of its Gauss diagram which have exactly k arrows. Note that $\varphi_k = \psi \circ \bar{\varphi}_k$ where $\bar{\varphi}_k$ sends a knot diagram to the sum of all of the subdiagrams of its Gauss diagram which have exactly k arrows, and ψ reparametrizes each summand to make it a Gauss diagram. Let $\varphi_{\leq k} := \sum_{i=1}^{k} \varphi_i$ be the map that sends a knot diagram to the sum of all of the reparametrized subdiagrams of its Gauss diagram which have *at most* k arrows. The maps φ_k and $\varphi_{\leq k}$ are *not* invariants of knots but every finite type invariant factors through $\varphi_{\leq k}$, as follows from the next theorem.

Theorem 2.1 (Goussarov-Polyak-Viro [GPV00], see also [Rou07])**.** *A knot invariant ζ is of type k if and only if there is a linear functional ω on $\mathcal{GD}_k$ such that $\zeta = \omega \circ \varphi_{\leq k}$.*

We show that $\varphi_{\leq k}$ can be computed in time $\sim n^{\lceil k/2 \rceil}$. This result, combined with the above theorem, proves that all finite type invariants can be computed in time $\sim n^{\lceil k/2 \rceil}$, which is the main result of this paper.

It is surprising that $\varphi_{\leq k}$ can be computed in time $\sim n^{\lceil k/2 \rceil}$ because, at first glance, it would seem that one must require time n^k. A Gauss diagram with n arrows has $\sum_{i=1}^{k} \binom{n}{i}$

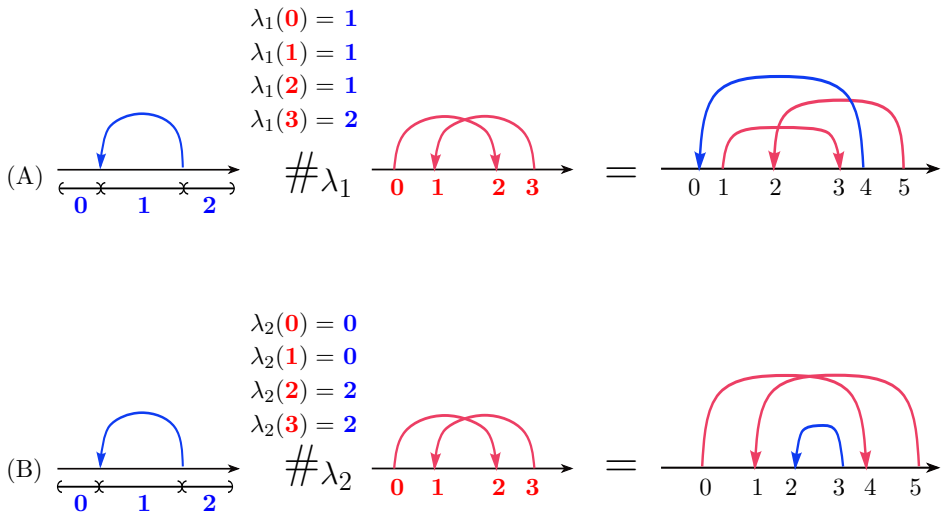

FIGURE 2. Two examples of superimposing the same diagrams along different λ gluing maps.

subdiagrams with k or fewer arrows. Because $\binom{n}{i} \sim n^i$, a Gauss diagram with n arrows has $\sum_{i=1}^{k} \binom{n}{i} \sim \sum_{i=1}^{k} n^i \sim n^k$ subdiagrams with k or fewer arrows. So $\bar{\varphi}_{\leq k}$ evaluated on a knot diagram with n crossings will be a sum of $\sim n^k$ subdiagrams. Note that the outputs of $\bar{\varphi}_k$ and φ_k have the same number of summands, but φ_k will have repeated terms and $\bar{\varphi}_k$ will not. So where do the computational savings come from? The idea is to break the computation of $\varphi_{\leq k}$ into two parts, one of which can be quickly pre-computed in a look-up table. The creation of this look-up table uses counting techniques taking advantage of dyadic intervals. These techniques are completely self-contained, and unrelated to finite type invariants and knot theory. In the next section, we describe these techniques that we will apply to prove that that $\varphi_{\leq k}$ can be computed in time $\sim n^{\lceil k/2 \rceil}$.

3. Computational Preliminaries: Counting Techniques using Dyadic intervals

3.1. **Counting with a look-up table.** A *look-up table* is a lexicographically-ordered list of (key $\mapsto$ value) entries, or more formally a lexicographically-ordered associative array. Below, Theorem 3.1 shows how to use a look-up table to count elements of a set inside $\bar{n}^\ell$, where $\bar{n} := [1, n]_{\mathbb{Z}}$. While we will need a generalized version of this Theorem, the proof of Theorem 3.1 showcases nicely how dyadic intervals are used to get computational savings.

Theorem 3.1. *Let Q be an enumerated subset of $\bar{n}^\ell$ with $1 \ll q = |Q| \ll n^\ell$. In time $\sim q$, a look-up table of size $\sim q$ can be created so that computing $|Q \cap R|$ will take time ~ 1 for any rectangular box $R \subset \bar{n}^\ell$.*

The straightforward approach to this theorem would be to create a look-up table with key-value pairs of the form $(R \mapsto |Q \cap R|)$ for all possible rectangular boxes R. A rectangular box in $\bar{n}^\ell$ is determined by choosing 2 interval endpoints in each coordinate of $\bar{n}^\ell$, so there are $\sim n^{2\ell}$ possible rectangular boxes in $\bar{n}^\ell$. This look-up table would take much longer than $\sim q$ to create. The trick is to create a restricted look-up table using only rectangular boxes whose

sides are dyadic intervals. To prove Theorem 3.1, let us first start with some preliminaries on dyadic intervals.

3.2. **Dyadic intervals.** For the purpose of this paper, a *dyadic interval* is a half-closed interval of the form $[2^p q, 2^p(q+1))_{\mathbb{Z}}$ for $p, q \in \mathbb{Z}_{\geq 0}$. A dyadic interval can be expressed via a binary expansion up to a certain accuracy, i.e. a fixed sequence of 0's and 1's followed by a fixed number of $*$'s or 'free' entries, and the integers contained in the interval are all possible completions of the expansions, i.e. all possible ways of replacing the $*$'s with 0's and 1's. For example $[2^2 2, 2^2 3)_{\mathbb{Z}} = \{8, 9, 10, 11\} = \{1000_2, 1001_2, 1010_2, 1011_2\}$, expressed in decimal and binary expansion respectively. Using the binary expansion, every number in the interval is of the form $10{*}{*}$ where the least significant two bits are free and the most significant two bits are fixed to be 10. We write u for a dyadic interval where u is a binary sequence followed by some number of $*$'s.

A dyadic interval u is *maximal* in $(b, c)_{\mathbb{Z}}$ if $u \subset (b, c)_{\mathbb{Z}}$ and any larger dyadic interval containing u is not contained in $(b, c)_{\mathbb{Z}}$. Notice if two dyadic intervals overlap, then one is contained in the other. Therefore, the maximal dyadic intervals of $(b, c)_{\mathbb{Z}}$ are disjoint. Every interval decomposes uniquely as a disjoint union of maximal dyadic intervals, as described in the following elementary Lemma.

Lemma 3.2. *For any interval $(b, c)_{\mathbb{Z}} \subset \mathbb{Z}_{\geq 0}$, there are at most $2\log_2(c-b)$ maximal dyadic intervals contained in $(b, c)_{\mathbb{Z}}$, and $(b, c)_{\mathbb{Z}}$ is a disjoint union of its maximal dyadic intervals.*

For a binary number, a truncation process can be used to compute dyadic intervals containing that binary number.

Truncation process: Given a binary number x with m bits of accuracy, starting from right to left, replacing one bit at a time with a $*$ generates a list of m dyadic intervals which contain x. For example, let $x = 7 = 00111_2$ where $m = 5$, then the truncation process generates the list of dyadic intervals $\{00111, 0011{*}, 001{*}{*}, 00{*}{*}{*}, 0{*}{*}{*}{*}\}$. If a number x (in decimal notation) is at most ℓ, then the number of bits needed to describe x in binary is at most $\log_2(\ell)$, and so x is contained in at most $\log_2(\ell)$ dyadic intervals coming from the truncation process with $\log_2(\ell)$ bits of accuracy.

3.3. **Proof and Corollary of Theorem 3.1.** We proceed with the proof the Theorem, as well as the main Corollary which we will use in the proof of the main theorem.

Proof of Theorem 3.1. A dyadic rectangular box in $\bar{n}^\ell$ is a product of ℓ dyadic intervals of $\bar{n}$. We describe a process to create a restricted look-up table of key-value pairs of the form $(R_d \mapsto |Q \cap R_d|)$ for only R_d that are dyadic rectangle in $\bar{n}^\ell$ and such that $|Q \cap R_d| > 0$.

To create the table, run through the enumerated elements $x \in Q$. For each x, from the truncation process described in Section 3.2, x is contained in at most $(\log_2(n))^\ell$ dyadic rectangular boxes R_d in $\bar{n}^\ell$. In the table, increment the value for R_d by 1 for each such R_d containing x, or create such an entry if it didn't already exist. Since there are q elements in Q, creating this table takes time $\sim q(\log_2(n))^\ell \sim q$, and there are $\sim q$ elements in the table. Using standard binary sorting techniques, accessing and modifying values in the table takes time $\sim \log_2 q \sim 1$.

A general non-dyadic rectangular box R in $\bar{n}^\ell$ is the disjoint union of at most $\sim (2\log_2(n))^\ell \sim 1$ maximal dyadic rectangular boxes, by Lemma 3.2. To count $|Q \cap R|$, one retrieves the values in the look-up table with key each maximal dyadic rectangular box in R, and then sums

together all those stored values to get $|Q \cap R|$. Since retrieval takes time ~ 1 and the sum is over ~ 1 elements, after the look-up table is completed, it takes time ~ 1 to compute $|Q \cap R|$. □

Viewing $|Q \cap R|$ as the sum $\sum_{x \in Q \cap R} 1$, Theorem 3.1 can be generalized, using almost exactly the same proof, to compute weighted sums $\sum_{x \in Q \cap R} w_x$ where the weights are in a $\mathbb{Q}$-vector-space. This generalization is stated in the Corollary below and is the version of the look-up table that will be used to prove the main result in Section 4.

Corollary 3.3. *Let Q be an enumerated subset of $\bar{n}^\ell$ with $1 \ll q = |Q| \ll n^\ell$. Let V be a $\mathbb{Q}$-vector space with $dim(V) \sim 1$, and let $\theta : \bar{n}^\ell \to V$ be a map that is zero outside of Q. Then, in time $\sim q$, a look-up table of size $\sim q$ can be created so that computing $\sum_R \theta$ will take time ~ 1 for any rectangular box $R \subset \bar{n}^\ell$.* [2]

4. Main Result

In this section, we state and prove the main theorem of this article. The strategy to quickly compute φ_k on a diagram D is to view a k-arrow subdiagram of D as the superimposition of two smaller subdiagrams E and F. The subdiagram E is placed inside D first, and instead of placing F, a look-up table is created to count in how many ways F could have been placed. Using Corollary 3.3, this look-up table can be computed and accessed very quickly, which ultimately gives the computational savings for the final result.[3]

Main Theorem. *Finite type invariants of type k can be computed on an n-crossing knot in time at most $\sim n^{\lceil k/2 \rceil}$.*

Proof. By Theorem 2.1, it suffices to show that $\varphi_{\leq k}$ can be computed in time $\sim n^{\lceil k/2 \rceil}$. Since $\varphi_{\leq k} = \sum_{i=1}^{k} \varphi_i$, it suffices to prove that φ_k can be computed in time $\sim n^{\lceil k/2 \rceil}$. This shows that $\varphi_{\leq k}$ can be computed in time $\sim k \cdot n^{\lceil k/2 \rceil} \sim n^{\lceil k/2 \rceil}$.

For a knot K with n crossings, let K be represented as a Gauss diagram with n arrows. By definition, $\varphi_k(K)$ is the sum of all reparametrized subdiagrams with exactly k arrows,

$$\varphi_k(K) = \sum_{\substack{D \subset K \\ |D| = k}} \psi(D),$$

where $D \subset K$ means D is a subdiagram of K.

Fixing a choice of $e, f \in \mathbb{Z}_{\geq 0}$ with $e + f = k$, every k-arrow subdiagram D can be viewed in $\binom{k}{e}$ ways as the superimposition of two smaller subdiagrams E and F of sizes e and f respectively. Note that $e, f, k \sim 1$. As discussed above, rather than breaking a specific k-arrow subdiagram down as a superimposition, one can instead build up all k-arrow subdiagrams by first choosing an e-arrow subdiagram E and then choosing an f-arrow subdiagram F

[2] There is a small caveat to Corollary 3.3 which requires that in some basis for V, the coefficients of θ must be quickly computable, which is not always the case. However, for the purposes of this paper and the application for which this Corollary is used, this issue does not arise and so we chose to leave this technical detail out of the statement.

[3] This is a theoretical result focusing on minimizing computational time. We do not take into consideration the space requirement to store the look-up table. The decomposition of D into E and F is trading some time-saving methods for storage space, which for large values could create practical limitations.

that lies in the complement of E in K. Summing over the possible choices of E, F, and superimpositions gives the next formula:

$$\varphi_k(K) = \sum_{\substack{D \subset K \\ |D| = k}} \psi(D) = \binom{k}{e}^{-1} \sum_{\substack{E \subset K \\ |E| = e}} \sum_{\lambda} \sum_{\substack{F \subset K,\ |F| = f \\ F_i \in (E_{\lambda(i)-1}, E_{\lambda(i)})}} \psi(E)\#_\lambda \psi(F) \tag{1}$$

Here, the endpoints of a subdiagram F are denoted by F_i, and the endpoints of E are E_j. The map $\lambda : [0, 2f-1]_{\mathbb{Z}} \to [0, 2e]_{\mathbb{Z}}$ is any placement map as described in the superposition map definition, and with the extra assignments $E_{-1} = -1$ and $E_{2e} = 2n$. The condition $E_{\lambda(i)-1} \leq F_i \leq E_{\lambda(i)}$ guarantees that the endpoints of the arrows in F lie in the complementary intervals of the endpoints of E, and in the correct order so that the superimposition along λ yields a subdiagram of K. This process overcounts every k-arrow subdiagram D of K $\binom{k}{e}$ times as every choice of splitting D into two pieces (i.e. a choice of E) occurs exactly once in the sum, which ultimately counts D $\binom{k}{e}$ times, instead of once.

Define $\theta_K : ([0, 2n-1]_{\mathbb{Z}})^f \to \mathcal{GD}_f$ by

$$(F_0, F_1, \cdots, F_{2f-1}) \mapsto \begin{cases} \psi(F) & \text{if } (F_0, F_1, \cdots, F_{2f-1}) \text{ are the ends of a subdiagram } F \subset K \\ 0 & \text{otherwise} \end{cases}$$

Now, the innermost sum from Equation (1) can be rewritten as a sum over θ_K as follows.

$$\sum_{\substack{F \subset K,\ |F| = f \\ F_i \in (E_{\lambda(i)-1}, E_{\lambda(i)})}} \psi(E)\#_\lambda \psi(F) \ = \ \psi(E)\#_\lambda \left(\sum_{\prod_i (E_{\lambda(i)-1}, E_{\lambda(i)})} \theta_K \right) \tag{2}$$

The upshot is that the sum on the right can be computed very quickly with a look-up table. Corollary 3.3 applies by taking Q to be the set of all $(F_0, F_1, \cdots, F_{2f-1}) \in ([0, 2n-1]_{\mathbb{Z}})^f$ that are the endpoints of a subdiagram of K. Here, $|Q| = \binom{n}{f} \sim n^f$ and the conclusion from the Corollary is that a look-up table can be created in time $\sim n^f$ so that computing the sum on the righthand side of Equation (2) takes time ~ 1.

Thus we arrive at our final equation,

$$\varphi_k(K) = \binom{k}{e}^{-1} \sum_{\substack{E \subset K \\ |E| = e}} \sum_{\lambda} \psi(E)\#_\lambda \left(\sum_{\prod_i (E_{\lambda(i)-1}, E_{\lambda(i)})} \theta_K \right). \tag{3}$$

To understand the computational complexity of Equation (3), we need to understand the complexity of each sum. We already showed that the innermost sum can be computed in time $\sim n^f$ by building a look-up table from Corollary 3.3. For the middle sum, λ can be any non-decreasing function $\lambda : [0, 2f-1]_{\mathbb{Z}} \to [0, 2e]_{\mathbb{Z}}$ of which there are $\binom{2e+2f}{2f} = \binom{2k}{2f}$ possible functions, which is $\sim$ to a power of k and does not add to the complexity of the total sum.

For the fixed choice of $e + f = k$, the outer sum in Equation (3) has at most $\sim \binom{n}{e} \sim n^e$ terms. Therefore, the total time of computing $\varphi_k(K)$, which includes first creating the look-up table and then computing the sum over all subdiagrams $D \subseteq K$ of size k, is $\sim n^e + n^f$.

This sum can be minimized for $e = \lceil \frac{k}{2} \rceil$ and $f = \lfloor \frac{k}{2} \rfloor$, in which case we get computation time $\sim 2n^{\lceil \frac{k}{2} \rceil} \sim n^{\lceil \frac{k}{2} \rceil}$.

□

References

[BL93] J. S. Birman and X.-S. Lin. Knot Polynomials and Vassiliev's Invariants. *Inventiones Mathematicae*, 111:225–270, 1993. 1

[BN95] D. Bar-Natan. On the Vassiliev Knot Invariants. *Topology*, 34:423–472, 1995. 1

[BNBNHS23] Dror Bar-Natan, Itai Bar-Natan, Iva Halacheva, and Nancy Scherich. Yarn Ball Knots and Faster Computations. *Journal of Applied and Computational Topology*, 8, 10 2023. 1

[GPV00] M. Goussarov, M. Polyak, and O. Viro. Finite Type Invariants of Classical and Virtual Knots. *Topology*, 39(5):1045–1068, 2000. 3

[Rou07] F. Roukema. Goussarov-Polyak-Viro Combinatorial Formulas for Finite Type Invariants. *ArXiv Preprint*, arXiv:0711.4001, 2007. 3

[Vas90] V. A. Vassiliev. *Cohomology of Knot Spaces*. American Mathematical Society, Providence, RI, 1990. 1

[Vas92] V.A. Vassiliev. *Complements of Discriminants of Smooth Maps: Topology and Applications*, volume 98 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 1992. 1