

A Fast Algorithm for the Hecke Representation of the Braid Group, and Applications to the Computation of the HOMFLY-PT Polynomial and the Search for Interesting Braids

Clément Maria   

Inria d'Université Côte d'Azur, Valbonne, France

Hoel Queffelec   

France-Australia Mathematical Sciences and Interactions ANU - CNRS International Research Laboratory, Canberra, Australia

Abstract

Knot theory is an active field of mathematics, in which combinatorial and computational methods play an important role. One side of computational knot theory, that has gained interest in recent years, both for complexity analysis and practical algorithms, is quantum topology and the computation of topological invariants issued from the theory.

In this article, we leverage the rigidity brought by the representation-theoretic origins of the quantum invariants for algorithmic purposes. We do so by exploiting braids and the algebraic properties of the braid group to describe, analyze, and implement a fast algorithm to compute the Hecke representation of the braid group. We apply this construction to design a parameterized algorithm to compute the HOMFLY-PT polynomial of knots, and demonstrate its interest experimentally. Finally, we combine our fast Hecke representation algorithm with Garside theory, to implement a reservoir sampling search and find non-trivial braids with trivial Hecke representations with coefficients in $\mathbb{Z}/p\mathbb{Z}$. We find explicitly several such braids, for the 4-strand and 5-strand braid groups.

2012 ACM Subject Classification Mathematics of computing → Geometric topology; Theory of computation → Fixed parameter tractability

Keywords and phrases Hecke representation of the braid group, parameterized algorithm, HOMFLY-PT polynomial of knots, reservoir sampling, faithfulness of Hecke representation

Digital Object Identifier 10.4230/LIPIcs.SoCG.2026.76

Related Version *Full Version:* <https://arxiv.org/abs/2512.06142> [29]

Supplementary Material *Software (Source code):* <https://doi.org/10.5281/zenodo.19349647>

Funding *Clément Maria:* Partially supported by the ANR project ANR-20-CE48-0007 (AlgoKnot).

Acknowledgements We would like to thank Asilata Bapat, Stepan Orevkov – who pointed us to Morton’s 1985 implementation [31] of the HOMFLY-PT algorithm –, Emmanuel Wagner and Oded Yacobi for helpful discussions. We would also like to thank the anonymous referees for the detailed comments and useful feedback. Some experiments presented in this paper were carried out using the Grid’5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see <https://www.grid5000.fr>).

1 Introduction

Geometrically, a *braid* on n strands is the embedding of n non-intersecting paths in the 3-dimensional space $\mathbb{R}^2 \times [0, 1]$, such that every path connects a point $(i, 0, 0)$, $i \in \{1, \dots, n\}$ in the bottom plane to a point $(j, 0, 1)$, $j \in \{1, \dots, n\}$ in the top plane, and every path grows monotonically along the z -axis. Two braids are *equivalent* if there is an ambient isotopy of



© Clément Maria and Hoel Queffelec;
licensed under Creative Commons License CC-BY 4.0

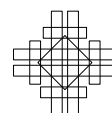
42nd International Symposium on Computational Geometry (SoCG 2026).

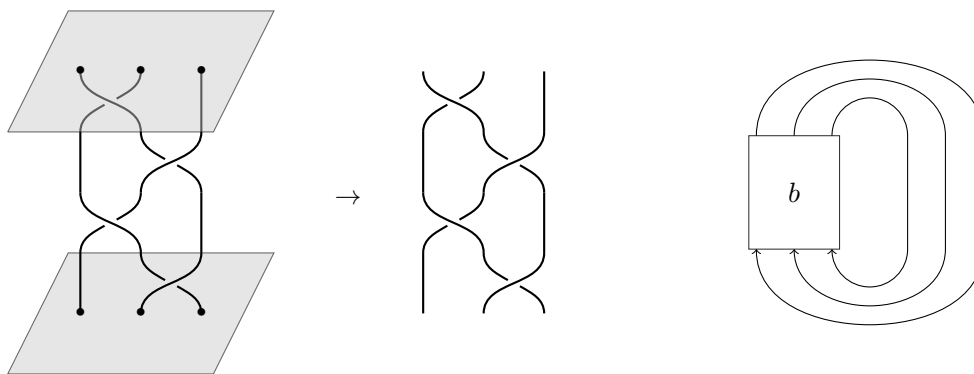
Editors: Hee-Kap Ahn, Michael Hoffmann, and Amir Nayyeri; Article No. 76; pp. 76:1–76:18

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 1** Left: Geometric braid on 3 strands in $\mathbb{R}^2 \times [0; 1]$, and natural diagram (projection on xz -plane). Right: Braid closure $\hat{b}$ of braid b .

$\mathbb{R}^2 \times [0, 1]$ fixing the bottom and top planes and taking one braid to the other. Braids are generally represented by *braid diagrams*, that are planar projections of a braid along the y -axis, keeping track of upper and under crossings; see Figure 1.

Braids are notably important in knot theory, as any link can be represented as the closure of a braid [1]. Knots have been studied extensively under the algorithmic lens. A famous problem is the computational complexity of recognizing the *trivial knot* from an input diagram, which is known to be in the complexity classes **NP** [19] and **coNP** [24], for which the best known worst case algorithm is exponential [19], but which experimentally exhibits a fast polynomial time behavior [11] with optimized implementation. In particular, the experimental aspects of computational knot theory play a fundamental role in the field, where mathematicians and computer scientists use efficient software, such as **Regina** [6, 9] and **SnapPy** [12], as well as computer-constructed census of knots [8], to guess and challenge profound conjectures, *e.g.* [5, 14, 16]. Consequently, an important side of computational knot theory is the design and implementation of fast, highly optimized algorithms.

Motivated by a finer understanding of the complexity of problems related to knots as well as practically fast computation, a recent route of research uses tools from *parameterized complexity* to compute *topological invariants* of knots. This approach has been particularly successful for invariants constructed via *quantum topology*, a field of topology using tools from quantum mechanics. In particular, the *treewidth* of a graph is a parameter measuring how close the graph is to a tree and, similarly, the *pathwidth* measures the proximity to a path. They can be extended to knot theory by considering the graph obtained from a diagram by putting a vertex on each crossing and an edge for each strand connecting crossings.

These are important parameters capturing a certain notion of sparsity of the input, and they can be combined with algorithmic techniques such as *dynamic programming* to design algorithms whose complexity depends exponentially on the tree/pathwidth and only polynomially in the size of the input: in quantum topology, such algorithms have been designed for the Jones and Kauffman polynomials [27], the Reshetikhin-Turaev invariants [28], and the HOMFLY-PT polynomial [7], which are **#P-hard** to compute in general [23]. Note that these quantum invariants, together with theoretically and practically fast algorithms, have been applied to construct knot censuses [8].

Contrary to knots, braids on n -strands have a natural algebraic description as a group, the *braid group* B_n , yielding rich algebraic properties that we exploit heavily in the following.

Our results. The goal of this article is to describe fast algorithms and data structures for braids and the braid group, related to quantum topology, and to apply them to computational knot theory and experimental mathematics. As opposed to algorithms on knots mentioned above, these algorithms rely heavily on the algebraic structure of the braid group.

Our starting point is to consider the *Hecke representation* of a braid. The *Hecke algebra* $\mathcal{H}_n$ is a fundamental concept in modern mathematics [30], deeply connected to group theory, number theory, and knot theory. The braid group B_n admits a representation into the Hecke algebra $\mathcal{H}_n$, i.e., a map $B_n \rightarrow \mathcal{H}$ respecting the group structure. Knowing whether this map (with $\mathbb{Z}$ -coefficients) is *faithful* (i.e., only the trivial braid has a trivial image) is a major open question [35, 21, 25, 4], related to the detection of the unknot by the Jones polynomial [20].

Our algorithm scans an input braid diagram from bottom to top, and updates its representation in a basis of $\mathcal{H}$. This strategy was already used by Morton and Short in the 80's [32]; the program we wrote can be seen as an optimized and parallelized version of theirs.

► **Theorem 1** (see Theorem 10). *Given a braid $b \in B_n$ on n strands and with N crossings, there is an algorithm to compute its Hecke representation in $O(n!(n \log n) \times N)$ operations and $O(n! \times N)$ algebraic operations, storing at most $n! + O(1)$ algebraic elements over the computation.*

Every knot or link can be obtained from the braid closure of a braid [1]; see Figure 1. Additionally, the HOMFLY-PT polynomial of the knot/link obtained from the braid closure can be computed from the Hecke representation of the braid by taking a *trace*. This is an alternative, more algebraic approach, to the usual definition of quantum invariants used in parameterized algorithms in the literature, relying either on *state sums* [10], *tensor networks* [28], or *skein relations* [7]. Here, we exploit the rigidity provided by the algebraic or representation-theoretic point of view to design more efficient algorithmic processes. In particular, the morphism spaces between representations are of much lower dimension than those between the underlying vector spaces, which helps reducing the algorithmic complexity.

We describe a fast implementation of the trace operation in the single pass of the coordinate vector representing the Hecke element associated to the braid.

► **Theorem 2** (see Theorem 19). *Given a braid $b \in B_n$ with m crossings, there is an algorithm to compute its HOMFLY-PT polynomial with the same complexity as Theorem 1.*

We compare experimentally this algorithm against Burton's implementation [7, 9] on large families of braid closures, and demonstrate its practical interest.

Finally, we use these fast implementations in experimental mathematics, running an extensive search for counter-examples to the Hecke faithfulness question in the case of Hecke representation with $\mathbb{Z}/p\mathbb{Z}$ coefficients.

Taking inspiration from earlier works [2, 3, 18], we have implemented a random algorithm to find counter-examples to faithfulness in the braid groups B_4 and B_5 , for different coefficients $\mathbb{Z}/p\mathbb{Z}$. Exploiting the favorable algebraic properties of braids, we have used *Garside theory* to generate increasingly complicated, non-trivial, braids for the search, running on each of them the algorithm from Theorem 10 in order to find braids whose Hecke representation was getting increasingly close to the trivial one. We have found explicit non-trivial braids in B_4 whose Hecke image is trivial in coefficients $\mathbb{Z}/2\mathbb{Z}$, $\mathbb{Z}/3\mathbb{Z}$ and $\mathbb{Z}/4\mathbb{Z}$. The non-faithfulness of the Hecke representation was known abstractly in these cases, using the explicit decomposition into simpler representations [21, Sections 5.7, 8.5], but our algorithm provides an efficient way to find examples of elements in the kernel. Similarly, we have explicitly found a non-trivial braid in B_5 whose Hecke image modulo 2 is trivial, while the status of the faithfulness of the Hecke representation of B_5 only relies on embedding B_4 into B_5 .

The code is available as part of the *KumQuAT* library [36].

Comparison with the literature. The HOMFLY-PT polynomial is a powerful topological invariants, and its computation has attracted the attention of mathematicians. For a knot diagram with N crossings, Kauffman gave a fast $O(2^N)$ skein template algorithm [22]. More recently, starting from Kauffman’s work, Burton [7] designed the first fixed parameter tractable algorithm in the treewidth of the knot diagram, running in $O((2\text{tw})!^4 \cdot \text{poly}(N))$, and implemented it in the software **Regina** [9]. This complexity bound should be compared to our bound from Theorem 1.

While treewidth, over all possible diagrams of a knot, is a generally smaller parameter than the braid index, it is common for them to be essentially equal, and the diagrams minimizing the treewidth are braid closures ; this is the case for *torus knots* for example [34, 26]. On the other hand, the complexity dependence of our algorithm in the number of strands is much lower than Burton’s dependence in the treewidth, which proves an advantage in practice on particular families of examples where treewidth and braid index are close.

Our approach to find counter-examples to the faithfulness of the Hecke representation is inspired from recent works [18] in computational mathematics, which have used a combination of curve-based random search with Garside theory to study the faithfulness of the *Bureau representation* of the braid group, and in particular to find braids with trivial representations. However, the Bureau representation of a braid is significantly less costly to compute, with a (small) polynomial dependence in both number of strands and number of crossings, while all known algorithms for the HOMFLY-PT polynomial, including this work, have a super-exponential dependence in the number of strands.

2 Preliminaries

2.1 Permutations and compact encoding

The *symmetric group* Σ_n is the group of permutations of n elements $\{1, \dots, n\}$. The group Σ_n is generated by the *transpositions*, *i.e.*, the permutations s_i of the two consecutive elements at index i and $i + 1$, for $1 \leq i \leq n - 1$. Every permutation $w \in \Sigma_n$ can be represented either by a unique *one-line word* $w(1)w(2) \cdots w(n)$, where $w(i)$ is the image of i under the permutation w , or by a non-unique product of generators $w = s_{i_k} \cdots s_{i_1}$.

An *inversion* in a permutation w is a pair of indices (i, j) such that $i < j$ and $w(i) > w(j)$. The (*Coxeter*) *length* of a permutation w , denoted by $\ell(w)$, is its total number of inversions.

The *Lehmer code* $L(w)$ of a permutation $w = w(1)w(2) \cdots w(n)$ is the sequence of positive integers $L(w) := (L(w)_1, \dots, L(w)_n)$ such that:

$$L(w)_i := \#\{j > i : w(j) < w(i)\},$$

i.e., $L(w)_i$ counts the number of inversions happening to the right of index i . In particular, $0 \leq L(w)_i \leq n - i$. Permutations are in bijection with their Lehmer code.

This last property allows us to represent a permutation with a single number, using the *factorial number system*. Specifically, given a sequence of numbers $\mathbf{a} = (a_1, \dots, a_n)$ such that $0 \leq a_i \leq n - i$ for all i , one can write uniquely the sequence as a single positive integer:

$$(a_1, \dots, a_n) \rightarrow F_{\mathbf{a}} = \sum_{i=1}^n a_i \times (n - i)! \quad (1)$$

In particular, the application $\Sigma \rightarrow \{0, \dots, n! - 1\}$ sending $w \mapsto F_{L(w)}$ is a bijection. In the following, we call the number $F_{L(w)}$ the *index of the permutation* w .

We prove, in the long version [29], the following basic facts about the encoding:

► **Lemma 3.**

- (i) Given an index $F_{L(w)} \in \{0, \dots, n! - 1\}$, there is an $O(n \log n)$ time algorithm to compute a one-line word presentation of w ,
- (ii) Given a permutation w represented as a one-line word $w(1) \cdots w(n)$, there is an $O(n \log n)$ time algorithm to compute its index $F_{L(w)}$,
- (iii) Let $w, w' \in \Sigma_n$ be permutations, then:

$$w(1) \cdots w(n) \leq_{\text{lex}} w'(1) \cdots w'(n) \text{ iff } F_{L(w)} \leq F_{L(w')},$$

where $\leq_{\text{lex}}$ denotes the lexicographic order.

2.2 Braids and Garside structure

We consider B_n the braid group on n strands:

$$B_n = \left\langle \sigma_1, \dots, \sigma_{n-1} \mid \begin{cases} \sigma_i \sigma_{i+1} \sigma_i = \sigma_{i+1} \sigma_i \sigma_{i+1} \\ \sigma_i \sigma_j = \sigma_j \sigma_i \text{ if } |i - j| > 1 \end{cases} \right\rangle.$$

The generators $\{\sigma_i^{\pm 1}\}_i$ are called the *Artin generators* of the braid group. The monoid with the same presentation, *i.e.*, generated by the $\{\sigma_i^{+1}\}_i$, is denoted by B_n^+ .

For some of our applications, we are interested in producing braids that we want to make sure get more and more complicated. The notion of Garside structure is instrumental, by providing us with an algorithmic-friendly normal form. We very briefly review the most important aspects, but refer to [13] for details.

Let us denote $[1, \Delta]$ the braids that realize positive lifts (that is, in B_n^+) of reduced words from the symmetric group. For example, $s_1 s_2 s_1 = s_2 s_1 s_2 \in \Sigma_n$ lifts to $\sigma_1 \sigma_2 \sigma_1 = \sigma_2 \sigma_1 \sigma_2 \in B_n$. It is a classical result that the map is well-defined (independent on the choice for the starting word). The half-twist $\Delta = \sigma_1 \cdots \sigma_{n-1} \sigma_1 \cdots \sigma_{n-2} \cdots \sigma_1$ is the so-called *Garside element*. One has a lattice structure [17, 13] on $[1, \Delta]$, extending to B_n^+ , with left divisibility given by:

$$b_1 \leq_L b_2 \Leftrightarrow \exists b_3 \in B_n^+, b_1 b_3 = b_2.$$

Similarly, one can define a notion of right divisibility, $\leq_R$. We are now ready to obtain the normal form $b_k \cdots b_1$ of a braid $b \in B_n^+$ by inductively defining:

$$b_i := \sup\{b' \in [1, \Delta], b' \leq_R (b b_1^{-1} \cdots b_{i-1}^{-1})\}.$$

If one adds the fact that any braid $b \in B_n$ can be written as $\Delta^r b^+$ with $b^+ \in B_n^+$, then one has a preferred expression for any braid.

At some point, we will be interested in using the Garside normal form to generate complicated braids. This will be made algorithmic thanks to the following lemma.

► **Lemma 4.** *The word $b_k \cdots b_1$ with $b_i \in [1, \Delta]$ is under normal form if and only if the following condition holds:*

$$\forall i \geq 2, \forall j, \quad \sigma_j \leq_R b_i \Rightarrow \sigma_j \leq_L b_{i-1}.$$

The divisibility in $[1, \Delta]$ being controlled by Σ_n , the computations can be made in the symmetric group. More details about the underlying automaton appear in Section 5.3.

2.3 Hecke algebra

The Hecke algebra is an ubiquitous object in representation theory. We introduce it as a quotient of the braid group, which makes it most natural for the computations we care about, and we refer to [29] and to Mathas' book [30] for further details and context.

► **Definition 5.** Let $\mathcal{H}_n$ be the $\mathbb{Z}[q^{\pm 1}]$ module obtained from the group algebra of the braid group as follows:

$$\mathcal{H}_n = \frac{\mathbb{Z}[q^{\pm 1}][B_n]}{\langle (\sigma_i - 1)(\sigma_i + q^2) \rangle}$$

We denote by $\psi : B_n \rightarrow \mathcal{H}_n$ the natural induced map. We quickly review some useful basic properties of the Hecke algebra that will be instrumental to us.

► **Lemma 6.** Denote by T_{s_i} the image of the i -th generator $\sigma_i \in B_n$. Then there is a well-defined map (of sets):

$$\Sigma_n \rightarrow \mathcal{H}_n \quad s = s_{i_1} \cdots s_{i_k} \mapsto T_s := T_{s_{i_1}} \cdots T_{s_{i_k}}.$$

Furthermore the image $\{T_s\}_{s \in \Sigma_n}$ is a basis of the Hecke algebra as a $\mathbb{Z}[q^{\pm 1}]$ -module.

The content of the above lemma is that the image of s does not depend on the word chosen to write it. This is simply because the T_s satisfy the same braiding relations as the elements in the symmetric group. To our eyes, the key property is that we have a basis indexed by permutations, which can be efficiently handled. For computation purposes, we also take advantage of the fact that the structure constants of this basis are quite simple, as expressed in the following classical lemmas [30, Theorem 1.11].

► **Lemma 7.**

$$\forall w \in \Sigma_n, \forall i, T_{s_i} T_w = \begin{cases} T_{s_i w} & \text{if } \ell(s_i w) > \ell(w); \\ (1 - q^2)T_w + q^2 T_{s_i w} & \text{if } \ell(s_i w) < \ell(w). \end{cases}$$

► **Lemma 8.**

$$\forall w \in \Sigma_n, \forall i, T_{s_i}^{-1} T_w = \begin{cases} (1 - q^{-2})T_w + q^{-2} T_{s_i w} & \text{if } \ell(s_i w) > \ell(w); \\ T_{s_i w} & \text{if } \ell(s_i w) < \ell(w). \end{cases}$$

When we compute knot invariants, it is convenient to take braid closures of braids. Assuming that P is a knot invariant, it is clear that $P(\widehat{b_1 b_2}) = P(\widehat{b_2 b_1})$. This remark motivates the introduction of an intermediary object $\text{Tr}(\mathcal{H}_n)$ which is obtained from $\mathcal{H}_n$ by formally adding the relation $b_1 b_2 = b_2 b_1$. This makes it possible to break the computation into a first step $\mathcal{H}_n \rightarrow \text{Tr}(\mathcal{H}_n)$, where elements are reduced as much as possible using the trace relation $b_1 b_2 = b_2 b_1$, before using information about the specific invariant we want to compute to go from $\text{Tr}(\mathcal{H}_n)$ to $\mathbb{Z}[q^{\pm 1}]$. Note that because $\psi : B_n \rightarrow \mathcal{H}_n$ is a representation, for any two braids $b_1, b_2 \in B_n$, we have $\text{Tr}(\psi(b_1 b_2)) = \text{Tr}(\psi(b_1)\psi(b_2)) = \text{Tr}(\psi(b_2)\psi(b_1)) = \text{Tr}(\psi(b_2 b_1))$.

Finally, when searching for counterexamples, we measure how far an element of $\mathcal{H}_n$ is from the identity. We use the following notion, that we adapt from [18].

► **Definition 9.** Let $x = \sum_w a_w T_w \in \mathcal{H}_n$. We define its projective length:

$$\text{projlength}(x) = \max\{\deg(a_w), w \in \Sigma_n\} - \min\{\text{val}(a_w), w \in \Sigma_n\},$$

where $\deg$ and val are the degree and valuation of a Laurent polynomial in the variable q .

Most of the notions we have defined here enjoy similar definitions with $\mathbb{Z}/p\mathbb{Z}$ in place of $\mathbb{Z}$. We use this context in Section 5.

3 Computing the Hecke representation of the braid group

The goal of the algorithm is to compute the image $\psi(b)$ in the Hecke algebra $\mathcal{H}_n$ of a braid $b = \sigma_{i_c}^{\epsilon_c} \cdots \sigma_{i_1}^{\epsilon_1} \in B_n$, given as an expression of the braid in the generators $\{\sigma_i\}$ of B_n .

Our algorithm is iterative, and maintains at step j the image in $\mathcal{H}_n$ of the braid $b_{(j)} := \sigma_{i_j}^{\epsilon_j} \cdots \sigma_{i_1}^{\epsilon_1}$, made of the first j crossings of the input braid.

3.1 Data structure and initialization

For a braid in B_n , we represent its image in $\mathcal{H}_n$ by an array of coordinates (Laurent polynomials) in the basis $\{T_w\}$. Specifically, a vector $x = \sum_{w \in \Sigma_n} a_w T_w \in \mathcal{H}_n$, with $a_w \in \mathbb{Z}[q^{\pm 1}]$, is represented by a length $n!$ array **hecke**, such that **hecke** $[j] = a_w$ whenever the permutation w has index $F_{L(w)} = j$.

Recall that the one-line word $w(1) \cdots w(n)$ of a permutation w , and its index $F_{L(w)}$ can be obtained from one another in $O(n \log n)$ operations. Note also that the maximal value $n! - 1$ of an index is below 2^{64} when $n \leq 20$, which will always be the case in this article ; permutation indices are stored as standard integers.

The image of σ_{i_1} is $T_{s_{i_1}} \in \mathcal{H}_n$, and Lemma 8, shows that the image of $\sigma_{i_1}^{-1}$ is $q^{-2}T_{s_{i_1}} + (1 - q^{-2}) \in \mathcal{H}_n$. Consequently, if $\epsilon_1 = 1$, at the initialization, only the coefficient corresponding to s_{i_1} is 1, all other ones being 0. If $\epsilon_1 = -1$, then we have the coefficient $1 - q^{-2}$ at index 0, q^{-2} at the index representing s_{i_1} , and 0 elsewhere.

3.2 Inductive step

We suppose that at the beginning of step j of the algorithm, **hecke** stores the coordinates of the image of $\sigma_{i_{j-1}}^{\epsilon_{j-1}} \cdots \sigma_{i_1}^{\epsilon_1}$, that we update with the next generator $\sigma_{i_j}^{\epsilon_j}$.

Since ψ is a group homomorphism, we have $\psi(b_{(j)}) = T_{\sigma_{i_j}}^{\epsilon_j} \psi(b_{(j-1)})$. We consequently need to update the coordinates of **hecke** following the local relations described in Lemmas 7 and 8. We do so in the single pass through **hecke**, without allocating more than $O(1)$ Laurent polynomials in memory:

Algorithm 1 From **hecke** representing $\psi(b_{(j-1)})$ to representing $\psi(b_{(j)})$.

```

for  $i \leftarrow 0$  to  $n! - 1$  do
  compute  $w$  such that  $F_{L(w)} = i$ ;
  if  $\ell(s_{i_j} w) > \ell(w)$  then
    set  $\text{idx}_w \leftarrow F_{L(w)}$ ;  $\text{idx}_{s_{i_j} w} \leftarrow F_{L(s_{i_j} w)}$ ;  $a_w \leftarrow \text{hecke}[\text{idx}_w]$ ;
     $a_{s_{i_j} w} \leftarrow \text{hecke}[\text{idx}_w]$ ;
    if  $\epsilon_j = +1$  then
      set  $\text{hecke}[\text{idx}_w] \leftarrow q^2 a_{s_{i_j} w}$ ; set  $\text{hecke}[\text{idx}_{s_{i_j} w}] \leftarrow a_w + (1 - q^2) a_{s_{i_j} w}$ ;
    end
    else
      set  $\text{hecke}[\text{idx}_w] \leftarrow (1 - q^{-2}) a_w + a_{s_{i_j} w}$ ; set  $\text{hecke}[\text{idx}_{s_{i_j} w}] \leftarrow q^{-2} a_w$ ;
    end
  end
end

```

3.3 Correctness

The correctness of the initialization phase is guaranteed by Lemmas 7 and 8. We focus on the inductive step. Consider any permutation $w \in \Sigma$. According to Lemmas 7 and 8, both $T_{s_{i_j}}^{\pm 1} T_w$ and $T_{s_{i_j}}^{\pm 1} T_{s_{i_j} w}$ give linear combinations of T_w and $T_{s_{i_j} w}$, because $s_{i_j}(s_{i_j} w) = w$. Additionally, for any permutation $w' \notin \{w, s_{i_j} w\}$, the product $T_{s_{i_j}}^{\pm 1} T_{w'}$ contains no term T_w or $T_{s_{i_j} w}$. Thus the pair of coordinates of **hecke** corresponding to permutations w and $s_{i_j} w$ can be updated independently from the rest of the array, and their new values is a combination of their previous values. We update them at once in the core of the (outer) **if** loop of Algorithm 1.

Finally, the condition of the (outer) **if** loop guarantees that we proceed to the update once. Indeed, the length ℓ being the number of inversions, we have $\ell(s_{i_j} w) \in \{\ell(w) - 1, \ell(w) + 1\}$. In consequence, considering the two permutations w and $(s_{i_j} w)$, multiplying by s_{i_j} exactly increases the length once and decreases the length once.

In light of the above, the induction algorithm does compute $\psi(b_{(j)})$ from $\psi(b_{(j-1)})$ and, at the end of the algorithm, we are left with $\psi(b) = \sum_{w \in \Sigma_n} a_w T_w$.

3.4 Complexity

► **Theorem 10.** *Given $b \in B_n$ represented by N generators (N crossings), the algorithm above computes the Hecke representation $\psi(b) = \sum_{w \in \Sigma_n} a_w T_w$ in $O(n!(n \log n) \times N)$ operations and $O(n! \times N)$ algebraic operations in $\mathbb{Z}[q^{\pm 1}]$, storing at most $n! + O(1)$ Laurent polynomials in memory over the computation.*

Proof. The initialization of the vector **hecke** requires $O(n!)$ operations, and parsing the first crossing requires $O(n \log n)$ operations to convert permutations into their indices, and $O(1)$ arithmetic operations in $\mathbb{Z}[q^{\pm 1}]$ to initialize a constant number of entries in **hecke**. The induction step reads the array **hecke** (of length $n!$) once, and every time converts a constant number of permutations into their indices ($O(n \log n)$) and performs at most $O(1)$ operations in $\mathbb{Z}[q^{\pm 1}]$. Finally, we run the induction exactly n times. ◀

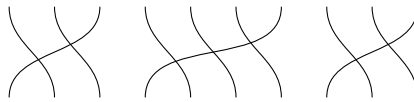
► **Remark 11.** Note that the algorithm above is rather simple, and the constant hidden by the big-O notation is rather small. This matters for running times in practice ; see Section 4.3.

► **Remark 12.** Note that the main **for** loop of Algorithm 1 can be naturally parallelized, considering that pairs of indices in **hecke**, corresponding to permutations w and $s_{i_j} w$, where s_{i_j} is the transposition corresponding to the new braid generator $\sigma_{i_j}^{\epsilon_j}$, are treated independently from other entries. We use a parallel and a non-parallel implementation of the **for** loop in Section 4.3.

4 Computing the HOMFLY-PT polynomial

4.1 The HOMFLY-PT polynomial

The HOMFLY-PT polynomial [15, 33] is a famous 2-variable knot invariant, that can be specialized to both the Jones polynomial and the Alexander polynomial. If one starts from a knot presented as a braid closure $\widehat{b}$ (see Figure 1), then, up to renormalization, the invariant is computed by taking a trace of the image of the braid b in the Hecke algebra. We now make this more precise, and we refer to the long version [29] for another presentation of the invariant.



■ **Figure 2** An annularly reduced element in Σ_{10} .

As in Section 3, we start from a braid $b = \sigma_{i_c}^{\epsilon_c} \cdots \sigma_{i_1}^{\epsilon_1}$, and pre-compute the array `hecke` encoding the Hecke representation $\psi(b) = \sum_{w \in \Sigma_n} a_w T_w$ with the algorithm of Theorem 10.

Our strategy is to simplify, in a single pass through `hecke`, the entries of the array, using the defining property of the trace, *i.e.*, $\text{Tr}(\psi(bb')) = \text{Tr}(\psi(b'b))$ for b, b' braids in B_n . We do not simplify the array all the way down, but instead only keep entries corresponding to permutations w with a simple form (called *annularly reduced*), and for which we know how to compute the participation to the final HOMFLY-PT polynomial.

Before describing the algorithm, we introduce some definitions and lemmas.

► **Definition 13.** An element $w \in \Sigma_n$ is said to be *annularly reduced* (see Figure 2) if it is a product of disjoint cycles r_i so that:

- r_i has support $(j_i, j_i + 1, \dots, j_i + l_i)$ for some j_i and l_i ;
- $r_i = (j_i + l_i, j_i, j_i + 1, \dots, j_i + l_i - 1)$.

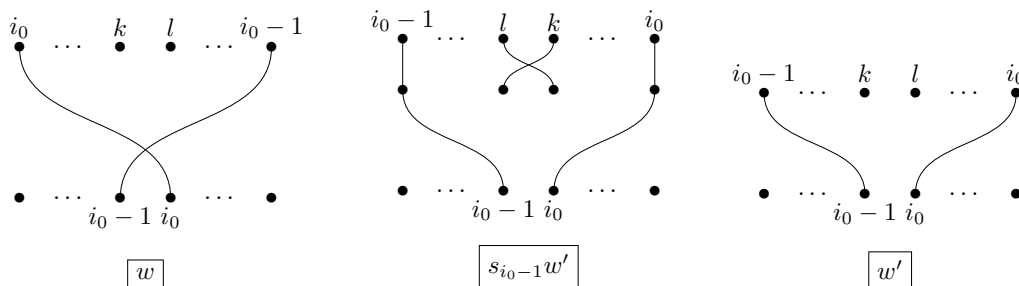
Our algorithm reduces braids to annularly reduced ones, using the following lemma.

► **Lemma 14.** Consider $w \in \Sigma_n$, and define, if it exists:

$$i_0 := \min\{i \mid w(i) < w(i) - 1\}.$$

Then $s_{i_0-1}ws_{i_0-1}$ and ws_{i_0-1} are lower than w in the lexicographic order.

Proof. We first argue that $w = w's_{i_0-1}$ as a reduced word. Indeed, from the definition of i_0 , $w(i_0 - 1) \geq i_0 - 2$ and thus $w(i_0 - 1) \geq w_{i_0}$. The relative order of $i_0 - 1$ and i_0 is exchanged under w and thus $w = w's_{i_0-1}$ with $\ell(w) = \ell(w') + 1$. We are thus in the following situation:

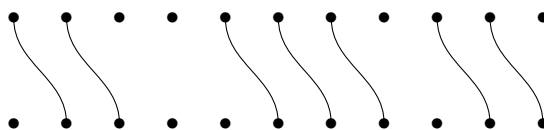


In both cases the lexicographic order gets lowered. ◀

The second piece of argument we need is the following one.

► **Lemma 15.** If $w \in \Sigma_n$ is so that $w(i) \geq i - 1$ for all i , then w is annularly reduced.

Proof. One first considers only the strands that move to the left. By hypothesis, those can only move one step to the left:



We are left with relating free ends with strands that can only go straight up or right. There is a unique solution, that consists of pairing the leftmost free end at the bottom with the leftmost free end at the top, and so on, which yields an annularly reduced element. ◀

We refer to the long version [29] for a precise definition of the HOMFLY-PT polynomial P . For our purpose, it suffices to say that the polynomial $P(\widehat{b})$ can be obtained from the vector `hecke` by:

- reducing the vector `hecke` in $\text{Tr}(\mathcal{H}_n)$ using trace moves $b_1 b_2 = b_2 b_1$. This leaves us with $\sum_{w \in \mathcal{A}} a'_w T_w$ where $\mathcal{A}$ denotes the set of annularly reduced words in Σ_n from Definition 13;
- evaluating annularly reduced elements on $\prod_i (a^{1-l_i} q^{l_i-1} \frac{a-a^{-1}}{q-q^{-1}})$ (with the notations of Definition 13). See [29] for a justification.

4.1.1 Simplifying the trace

We now want to simplify the vector `hecke` in a trace process. Given a basis element T_w , we want, when possible, to simplify its image $\text{Tr}(T_w) \in \text{Tr}(\mathcal{H}_n)$. We do that by ordering elements in w by the lexicographic order $\leq_{\text{lex}}$ of their one-line word $w(1) \cdots w(n)$, so that the simplification can be achieved in one scan through the set $\{T_w\}_{w \in \Sigma_n}$.

► **Remark 16.** Two elements that are annularly reduced might still be conjugate: the order of the cycles can be permuted. This results in keeping more non-zero entries in the vector we consider than necessary, but has the advantage of yielding a particularly simple algorithm.

We traverse the array `hecke` from right to left, *i.e.*, considering permutations w with *decreasing* position in lexicographic order: for every such w ,

1. look for the first i_0 so that $w(i_0) < i_0 - 1$.
2. If there is no such i_0 then w is already annularly reduced (Lemma 15).
3. Otherwise it can be written $w = w' s_{i_0-1}$. Replace T_w by $T_{i_0-1} T'_w$ which expresses as a single or a sum of two basis elements (Lemma 7). In both cases, the elements that appear are strictly lower in the lexicographic order (Lemma 14), and the coefficient of $T_{i_0-1} T'_w$ is removed and added to entries of `hecke` to the left of w .

4.1.2 Evaluating the polynomial

We now are left with $v = \sum_{w \in \mathcal{A}} a'_w T_w$. The result we are looking for is:

$$P(\widehat{b}) = \sum_{w \in \mathcal{A}} a'_w P(\widehat{b_w}) \quad (b_w \text{ is the positive lift of } w \in B_n).$$

The value for $P(\widehat{b_w})$ above is recorded in the following lemma.

► **Lemma 17.** *The HOMFLY-PT polynomial of an annularly reduced braid with cycles of lengths $l_1, \dots, l_k$ is $\prod_{i=1}^k (a^{1-l_i} q^{l_i-1} \frac{a-a^{-1}}{q-q^{-1}})$.*

Proof. The computation consists of taking the product of each cycle value, as computed in the long version [29]. ◀

The last step of the algorithm thus consists in replacing T_w by the product of the values of the HOMFLY-PT polynomial on the cycles of T_w , and taking the sum for all w 's, in a single pass through the array `hecke`.

4.2 Complexity and correctness

The correctness of the algorithm is guaranteed by the diverse lemmas proved along the way.

► **Theorem 18.** *Let `hecke` be the vector of (Laurent polynomials) coordinates of the Hecke representation of a braid $b \in B_n$, expressed in the $\{T_w\}_w$ basis. Given `hecke`, evaluating the HOMFLY-PT polynomial of b takes $O(n!(n \log n))$ operations and $O(n! \cdot n)$ algebraic operations, storing at most $n! + O(1)$ algebraic elements over the computation.*

Proof. Recall from Section 4.1.1 that we reduce the vector to only keep annularly reduced elements in a single pass on the array `hecke` of length $n!$. Computing w from its index and finding the first i_0 so that $w(i_0) < i_0 - 1$ takes $O(n \log n)$ operations. Then one finds w' so that $w = w' s_{i_0-1}$. This amounts to computing ws_{i_0-1} , which costs $O(1)$ operations. Then computing $T_{i_0-1}T_{w'}$ costs two multiplications in $\mathbb{Z}[q^{\pm 1}]$. In total, for this phase, we use $O(n!n \log n)$ combinatorial operations and $O(n!)$ arithmetic operations in $\mathbb{Z}[q^{\pm 1}]$.

Finally, thanks to Lemma 17, it suffices to know the number of cycles and their lengths to evaluate an annularly reduced element in $O(n)$ arithmetic operations in $\mathbb{Z}[q^{\pm 1}]$ at most (there are at most n disjoint cycles). Identifying cycles and their lengths is done directly by reading the one-line word of a permutation w ($O(n \log n)$ operations to compute the one-line word from w 's index, then $O(n)$ to identify cycles), yielding a last step in $O(n! \cdot n \log n)$ combinatorial operations and $O(n! \cdot n)$ arithmetic operations in $\mathbb{Z}[q^{\pm 1}]$. Summing the complexity gives the theorem. ◀

► **Theorem 19.** *Given a braid $b \in B_n$ with N crossings, the above algorithm computes its HOMFLY-PT polynomial in $O(n!(n \log n) \times N)$ operations and $O(n! \times N)$ algebraic operations, storing $n! + O(1)$ algebraic elements.*

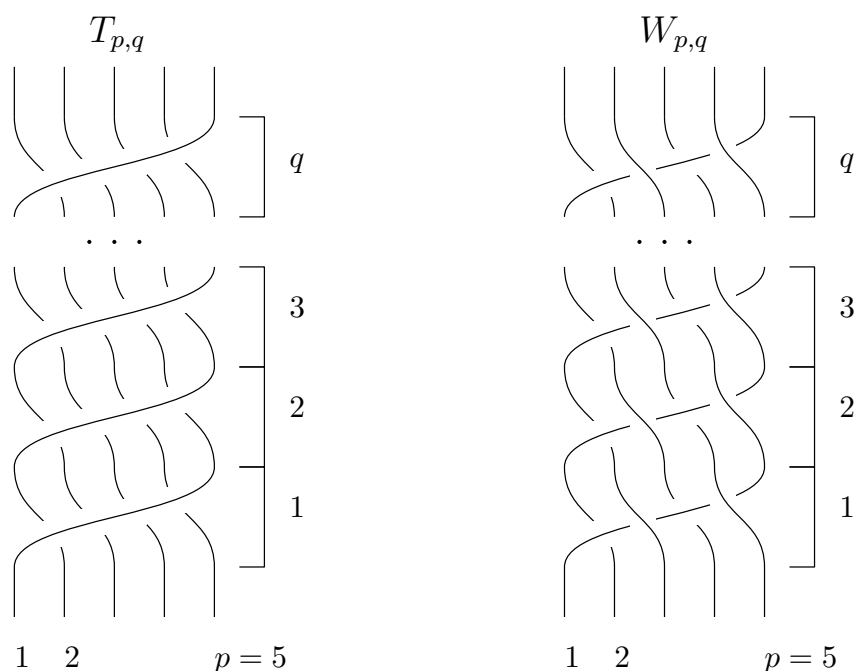
Proof. Summing the complexity of Theorem 10 and Theorem 18, all steps of the algorithm are covered; assuming $n \leq N$, Theorem 10 dominates the computation. ◀

4.3 Experimental performance

We have compared our code with the advanced C++ library **Regina** [9], implementing Burton's state of the art algorithm [7] for the HOMFLY-PT polynomial. We have run our experiments on an 8 cores, 32Gb RAM computer; parallel implementations use the library `Intel::TBB`. We restrict our attention to braids whose closures lead to knots. The computation of the Hecke representation, which largely dominates the timings for our computation of the HOMFLY-PT polynomial, can be implemented in parallel, as described in Remark 12, while **Regina** does not admit a parallel implementation and the algorithm it implements may not parallelize as naturally as ours.

We have generated p -strands torus braids $T_{p,q}$ and p -strands weaving braids $W_{p,q}$, in their standard form (see Figure 3), keeping only those parameters q for which the closure is a knot. We have used them as is in our algorithm. **Regina** working with knot diagrams and treewidth, we have run simplification heuristics and the construction of the tree decomposition, before computing the HOMFLY-PT polynomial (and before starting the timer). However, for these knots, the braid presentation is expected to be minimal for both size and parameter.

In Figure 4 we compare the running time of our algorithm *not using parallelism*, against the algorithm from **Regina** for torus braids $T_{p,q}$ and weaving braids $W_{p,q}$, for $3 \leq p < 6$ and all values of $q = p + 1, \dots, q_{\max}$, such that the closures of the braids lead to knots (i.e., $\gcd(p, q) = 1$), with $q_{\max}$ picked such that the braids $T_{p,q_{\max}}$ and $W_{p,q_{\max}}$ have approximately



■ **Figure 3** Torus braid $T_{p,q}$ and weaving braid $W_{p,q}$.

600 crossings. The results are very similar for torus and weaving braids, highlighting the fact that the running times depends mostly on the combinatorics of the input diagrams, and not so much on the algebraic computations. Our implementation consistently outperforms **Regina** on braids with more than 4 strands, by a factor ranging from 3 (shorter braids) to 9 (longer braids) for braids on 4 strands, a factor ranging from 3 to 4 for 5-braids.

In Figure 4, **Regina** is faster on the smaller 3-strands examples, by a factor 3 on shorter braids, and a smaller factor 1.3 on the larger braids ; all these timings are below 0.3 seconds, and may be explained by a slower initialization of data on our side. The global tendency is that our algorithm, even on its non-parallel form, gets better on longer braids (the slopes α of interpolating lines is $0.78 \leq \alpha \leq 0.8$ on all examples) and on braids with more strands.

On braids with ≥ 6 strands, **Regina** starts to **swap** due to excessive memory consumption, leading to impractical running times ; to the contrary, our implementation continues to be practical, Table 1 and see Figure 5 for much larger examples.

In Figure 5, we compare the performance of our implementation with and without parallelism. Recall that the parallelization is on the array **hecke** of length $n!$ (working with an n -strands braid). In consequence, we see the gain of parallelism increasing with the number of strands: first, the gain range from a factor 1.6 to 2.4 on 3-strands, from 3.3 to 4.54 4-strands, and from 3.8 to 6.25 on 5-strands. On braids with 6, 7, 8-strands, the gain ranges between a factor 5 to 7.14, near the optimal 8 (on 8 cores); see Figure 5.

Finally, the memory consumption of our implementation is much more frugal than **Regina**, and remains very applicable for large number of strands. The memory cost depends mostly on the number of strands; however, for a fixed number of strands, longer braids tend to consume slightly more memory because the Laurent polynomials stored in the array **hecke** become more complicated; the memory consumption of longer braids is shown in Table 1).

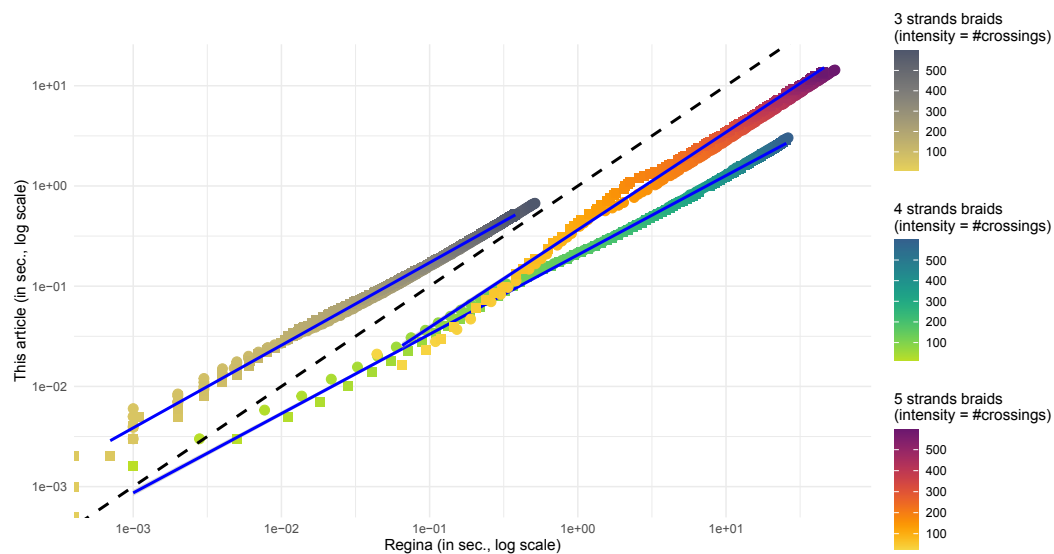


Figure 4 Timings of our algorithm (without parallelism) and Regina. Circle points represent torus braids, and square points represent weaving braids. The dashed line is $x = y$.

Table 1 Timing and peak memory usage of our algorithm using parallelism on large torus braids (similar results are found for weaving braids).

Braid	Num. of crossings	Timing parallel (8 cores)	Timing non-parallel	Peak memory
$T_{6,41}$	205	1.6 sec.	9.6 sec.	8.1 Mb
$T_{6,119}$	595	13.4 sec.	95.1 sec.	23.5 Mb
$T_{7,36}$	216	10.8 sec.	74 sec.	45.6 Mb
$T_{7,99}$	594	88.9 sec.	643 sec.	109 Mb
$T_{8,29}$	203	74.5 sec.	502 sec.	257 Mb
$T_{8,49}$	343	272 sec.	2265 sec.	435 Mb

In conclusion, our implementation is very competitive for knots represented by braids, even with a fairly large number of strands. It also combines very well with parallelism. We apply our parallel implementation in experimental mathematics in Section 5 to investigate algebraic properties of the Hecke algebra.

5 Proving the non-faithfulness of the Hecke representation in $\mathbb{Z}/m\mathbb{Z}$

With a bucket search procedure, coupled with the fast algorithm described in Section 3, we have been looking for non-trivial braids with trivial Hecke representation in several contexts.

Following [3, 18, 2], we generate a sample of non-trivial braids from B_n in Garside normal form, and store them in *buckets* of bounded size, one for each represented value of *augmented projlength* (see below). At iteration k of the search, the buckets store braids of Garside length k . For initialization, the Garside length 1 braids are in bijection with permutations in $\Sigma_n \setminus \{\text{id}, \Delta\}$, and we store them in the buckets. Inductively, if we have generated a sample of Garside length k braids, we attempt to extend each of them into Garside length $k + 1$ braids by appending an extra compatible Garside letter. Note that, when attempting to insert a new generated braid, we use a *reservoir sampling* [37] approach to guarantee that we extract an iid sample. In Section 5.3, we describe details on the optimized extension of

76:14 A Fast Algorithm for the Hecke Representation, and Applications

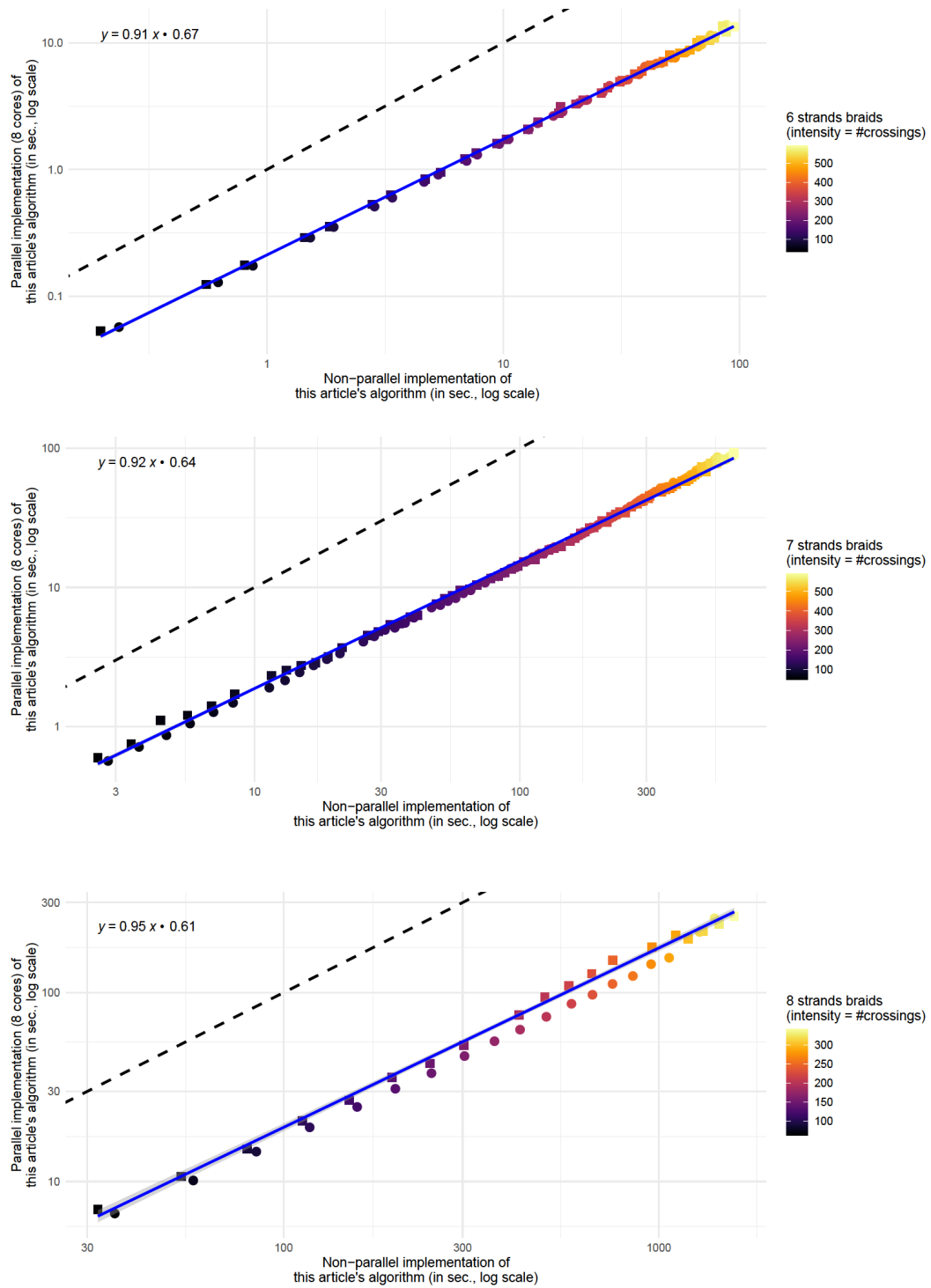


Figure 5 Timings of our algorithm with parallelism against our algorithm without parallelism. The interpolation line are near $x = y/7.14$ (we use 8 cores).

Garside normal forms we have implemented. In particular, we prove,

► **Lemma 20.** *Given a braid $b = b_k \dots b_1 \in B_n$ in Garside normal form, there is an $O(n \log n)$ algorithm to pick a Garside letter b_{k+1} iid among all Garside letters compatible with b_k .*

We define the *augmented projlength* of a positive braid b to be

$$\widehat{\text{projlength}}(b) : \sup_{k \leq 0} \{\text{projlength}(\psi(\Delta^k b))\},$$

and we try to minimize this quantity over the search. To avoid computing the $\widehat{\text{projlength}}$ for infinitely many negative powers of Δ , we (heuristically) stop the computation of $\widehat{\text{projlength}}(b)$ when we encounter a power $k_0 < 0$ for which $\psi(\Delta^{k_0} b)$ has a coordinate with negative degree.

The bucket search algorithm, if successful, finds positive braids b (not of the form Δ^l) such that there exists of negative power k of Δ satisfying: $\Delta^k \cdot b$ has $\widehat{\text{projlength}} = 0$. In all cases where we found such braids, they had an image in the Hecke algebra (in the basis used from Lemma 6) with a single non-trivial coordinate idx equal to 1. This coordinate corresponding to the positive permutation s^{-1} , the braid $s^{-1} \cdot \Delta^k \cdot b$ has trivial Hecke representation.

We have run our searches in parallel on a cluster of 2 CPUs, 128 cores/CPU, and 1024GB RAM.

5.1 Non-faithful Hecke representations of B_4

The decomposition of $\mathcal{H}_4$ into irreducible B_4 representations only contains the Burau representation as possibly faithful summand. The latter being not faithful at $p = 2, 3, 5$ implies the same for the Hecke representation. We indeed found such counterexamples for $p = 2$ and 3, as well as with $p = 4$, that we present in the long version [29].

5.2 Non-faithful Hecke representations of B_5

The same search performed on B_5 with base ring $\mathbb{Z}/2\mathbb{Z}$ also yielded a counterexample to faithfulness. In this case, non-faithfulness follows by including the B_4 kernel into B_5 . Here we could run a direct search in B_5 to find *explicit* counterexamples.

► **Theorem 21.** *The braid $s_9^{-1} \Delta^{-9} b$ is in the kernel of the Hecke representation of B_5 with coefficients in $\mathbb{Z}/2\mathbb{Z}[q^{\pm 1}]$, with:*

$$\begin{aligned} b = & \sigma_3 \sigma_4 \sigma_2 \sigma_3 \sigma_1 \sigma_2 \sigma_2 \sigma_4 \sigma_2 \sigma_3 \sigma_1 \sigma_2 \sigma_4 \sigma_2 \sigma_3 \sigma_3 \sigma_2 \sigma_3 \sigma_1 \sigma_3 \sigma_1 \sigma_2 \sigma_2 \sigma_3 \sigma_1 \sigma_3 \sigma_1 \sigma_2 \sigma_2 \sigma_3 \sigma_1 \sigma_3 \sigma_1 \sigma_2 \sigma_4 \sigma_3 \\ & \sigma_2 \sigma_3 \sigma_4 \sigma_1 \sigma_4 \sigma_3 \sigma_2 \sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_4 \sigma_3 \sigma_2 \sigma_3 \sigma_1 \sigma_2 \sigma_3 \sigma_3 \sigma_2 \sigma_3 \sigma_4 \sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_4 \sigma_3 \sigma_4 \sigma_2 \sigma_1 \sigma_2 \sigma_3 \sigma_3 \sigma_4 \sigma_2 \\ & \sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_4 \sigma_3 \sigma_4 \sigma_1 \sigma_2 \sigma_3 \sigma_3 \sigma_4 \sigma_2 \sigma_3 \sigma_4 \sigma_1 \sigma_4 \sigma_3 \sigma_4 \sigma_1 \sigma_2 \\ s_9^{-1} = & \sigma_2^{-1} \sigma_3^{-1} \sigma_4^{-1}, \quad \Delta^{-1} = \sigma_4^{-1} \sigma_3^{-1} \sigma_2^{-1} \sigma_1^{-1} \sigma_4^{-1} \sigma_3^{-1} \sigma_2^{-1} \sigma_4^{-1} \sigma_3^{-1} \sigma_4^{-1}. \end{aligned}$$

b is of Garside length 19, and the resulting counterexample is a braid with 186 crossings.

5.3 Garside automaton

To generate braids, we use the following finite state automaton.

- **Definition 22.** *Let **Garside** be the finite state automaton with:*
- *states, permutations s in Σ_n (or equivalently elements in $[1, \Delta]$);*
 - *s' -labeled arrows $s \rightarrow s'$ if $\forall s_j \leq_R s'$ then $s_j \leq_L s$.*

We create increasingly complicated braids by applying to a Garside normal form $b_k \cdots b_1$ any (possibly randomly chosen) letter from $\mathbf{Garside}(b_k)$. The construction of the automaton can be made very efficient thanks to the following observation.

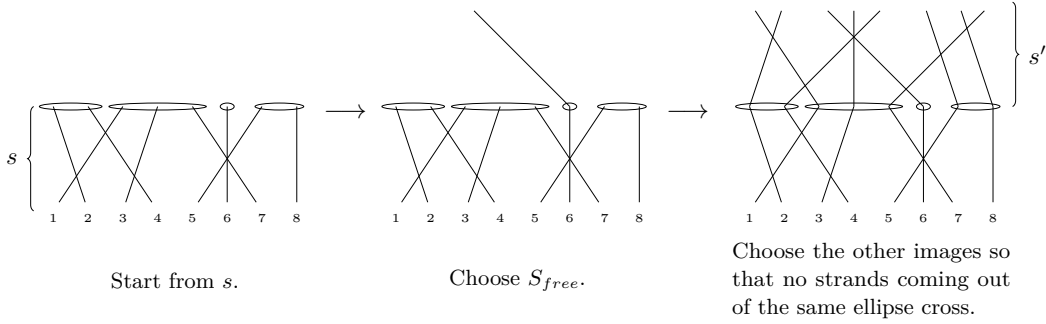
► **Lemma 23.** *Given $s = (s(1), \dots, s(n))$ and $s' = (s'(1), \dots, s'(n))$, s' can be applied at the state s if and only if the following property holds:*

$$s' \neq \text{id and } \forall i \in \llbracket 1, n \rrbracket \text{ with } s'(i) > s'(i+1), s^{-1}(i) > s^{-1}(i+1).$$

Proof. This restates the fact that right descents from s' need to be left descents of s . ◀

From a state s , we want to generate admissible arrows s' . All pairs $\{(i_r, i_r + 1) \mid \exists l < k, i_r = s(k), i_r + 1 = s(l)\}$ partition $\llbracket 1, n \rrbracket$ into intervals $\llbracket i_r + 1, i_{r+1} \rrbracket$. Lemma 23 prevents two entries from the same interval to cross in s' . This suggests the following (see Figure 6):

- for length 1 intervals ($i_r + 1 = i_{r+1}$), freely choose $s'(i_r)$ in $\llbracket 1, n \rrbracket$ (forming S_{free});
- images of the other intervals are disjoint ordered subsets of $\llbracket 1, n \rrbracket \setminus S_{free}$. These choices can be made one interval at a time in any order. Starting from the first interval $\llbracket i_r + 1, i_{r+1} \rrbracket$ of length $p' > 1$, one chooses a random ordered subset $S_{p'}$ of p' elements in $\llbracket 1, n - p \rrbracket$, and then decide that $s'(i_r + k)$ is the $S_{p'}(k)$ -th element in $\llbracket 1, n \rrbracket \setminus S_{free}$. Then one replaces S_{free} by $S_{free} \cup S_{p'}$ and goes to the next interval of size greater than 1.



■ **Figure 6** Generating descents.

A major advantage of the above construction is the following.

► **Lemma 24.** *Given a braid $b \in B_n$ in Garside normal form $b_k \cdots b_1$, there is an $O(n \log n)$ algorithm to pick a Garside letter b_{k+1} iid among all Garside letters compatible with b_k .*

Proof. If one can make uniform random choices of p unordered elements in $\llbracket 1, n \rrbracket$, and uniform random choices of p ordered elements in $\llbracket 1, p' \rrbracket$, then the procedure described above the lemma produces uniform choices in $\mathbf{Garside}(s)$. Picking iid random subsets can be done in time $O(n \log n)$ using, for example, a reservoir sampling algorithm. ◀

References

- 1 J. W. Alexander. A lemma on systems of knotted curves. *Proceedings of the National Academy of Sciences*, 9(3):93–95, March 1923. doi:10.1073/pnas.9.3.93.
- 2 A. Bapat and H. Queffelec. Some remarks about the faithfulness of the Burau representation of Artin–Tits groups, 2024. arXiv:2409.00144.
- 3 S. Bigelow. The Burau representation is not faithful for $n=5$. *Geometry & Topology*, 3(1):397–404, 1999.

- 4 S. Bigelow. Braid groups and iwahori-hecke algebras. *Problems on mapping class groups and related topics*, 74:285–299, 2006. [arXiv:1411.5418](#).
- 5 M. Brittenham and S. Hermiller. Unknotting number is not additive under connected sum, 2025. [arXiv:2506.24088](#).
- 6 B A Burton. Introducing Regina, the 3-manifold topology software. *Experiment. Math.*, 2004.
- 7 Benjamin A. Burton. The HOMFLY-PT Polynomial is Fixed-Parameter Tractable. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2018.18.
- 8 Benjamin A. Burton. The Next 350 Million Knots. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2020.25.
- 9 Benjamin A. Burton, Ryan Budney, William Pettersson, et al. Regina: Software for 3-manifold topology and normal surface theory. <http://regina.sourceforge.net/>, 1999–2018.
- 10 Benjamin A. Burton, Clément Maria, and Jonathan Spreer. Algorithms and complexity for Turaev–Viro invariants. *Journal of Applied and Computational Topology*, 2(1–2):33–53, August 2018. doi:10.1007/s41468-018-0016-2.
- 11 Benjamin A. Burton and Melih Ozlen. A fast branching algorithm for unknot recognition with experimental polynomial-time behaviour, 2014. [arXiv:1211.1079](#).
- 12 M Culler, N M Dunfield, and J R Weeks. SnapPy, a computer program for studying the geometry and topology of 3-manifolds. <http://snappy.computop.org/>, 1991–2018.
- 13 P. Dehornoy, F. Digne, E. Godelle, D. Krammer, and J. Michel. *Foundations of Garside theory*, volume 22. European Mathematical Society Zürich, 2015.
- 14 Renaud Detcherry. A quantum obstruction for purely cosmetic surgeries. *Annales de l'Institut Fourier*, 2025. Online first. doi:10.5802/aif.3673.
- 15 P. Freyd, D. Yetter, J. Hoste, W. B. R. Lickorish, K. Millett, and A. Ocneanu. A new polynomial invariant of knots and links. *Bull. Am. Math. Soc., New Ser.*, 12:239–246, 1985. doi:10.1090/S0273-0979-1985-15361-3.
- 16 Stavros Garoufalidis and Yueheng Lan. Experimental evidence for the volume conjecture for the simplest hyperbolic non-2-bridge knot. *Algebraic & Geometric Topology*, 5(1):379–403, May 2005. doi:10.2140/agt.2005.5.379.
- 17 F. A Garside. The braid group and other groups. *The Quarterly Journal of Mathematics*, 20(1):235–254, 1969.
- 18 J. Gibson, G. Williamson, and O. Yacobi. 4-Strand Burau is Unfaithful Modulo 5, 2023. [arXiv:2310.02403v1](#).
- 19 Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *Journal of the ACM*, 46(2):185–211, March 1999. doi:10.1145/301970.301971.
- 20 T. Ito. A kernel of a braid group representation yields a knot with trivial knot polynomials. *Mathematische Zeitschrift*, 280(1):347–353, 2015. [arXiv:1402.2028](#).
- 21 V. FR Jones. Hecke algebra representations of braid groups and link polynomials. *Annals of Math.*, 126:335–388, 1987.
- 22 Louis H. Kauffman. State models for link polynomials. *Enseignement Mathématique*, 36(1–2):1–37, 1990.
- 23 Greg Kuperberg. How hard is it to approximate the Jones polynomial? *Theory of Computing*, 11(1):183–219, 2015. doi:10.4086/toc.2015.v011a006.
- 24 Marc Lackenby. The efficient certification of knottedness and thurston norm. *Advances in Mathematics*, 387:107796, August 2021. doi:10.1016/j.aim.2021.107796.

- 25 G. I Lehrer and N. Xi. On the injectivity of the braid group in the hecke algebra. *Bulletin of the Australian Mathematical Society*, 64(3):487–493, 2001.
- 26 Coentun Lunel and Arnaud de Mesmay. A Structural Approach to Tree Decompositions of Knots and Spatial Graphs. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry (SoCG 2023)*, volume 258 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2023.50.
- 27 J.A. Makowsky and J.P. Mariño. The parametrized complexity of knot polynomials. *Journal of Computer and System Sciences*, 67(4):742–756, 2003. Parameterized Computation and Complexity 2003. doi:10.1016/S0022-0000(03)00080-1.
- 28 Clément Maria. Parameterized Complexity of Quantum Knot Invariants. In Kevin Buchin and Éric Colin de Verdière, editors, *37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2021.53.
- 29 Clément Maria and Hoel Queffelec. A fast algorithm for the hecke representation of the braid group, and applications to the computation of the homfly-pt polynomial and the search for interesting braids, 2025. arXiv:2512.06142.
- 30 A. Mathas. *Iwahori-Hecke algebras and Schur algebras of the symmetric group*, volume 15 of *Univ. Lect. Ser.* Providence, RI: American Mathematical Society, 1999.
- 31 H. Morton. br9z. <https://www.liverpool.ac.uk/~su14/knotprogs.html>, 1985.
- 32 Hugh R Morton and HB Short. Calculating the 2-variable polynomial for knots presented as closed braids. *Journal of Algorithms*, 11(1):117–131, 1990. doi:10.1016/0196-6774(90)90033-B.
- 33 J. H. Przytycki and P. Traczyk. Invariants of links of Conway type. *Kobe J. Math.*, 4(2):115–139, 1987.
- 34 Saul Schleimer, Arnaud de Mesmay, Jessica S. Purcell, and Eric Sedgwick. On the tree-width of knot diagrams. *J. Comput. Geom.*, 10(1):164–180, 2019. doi:10.20382/JOCG.V10I1A6.
- 35 C. C Squier. Matrix representations of artin groups. *Proceedings of the American Mathematical Society*, 103(1):49–53, 1988.
- 36 The KumQuAT Project. KumQuAT: Algorithms in quantum topology, 2026. URL: <https://kumquat.inria.fr/>.
- 37 Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. doi:10.1145/3147.3165.