A calculus for monomials in Chow group of zero cycles in the moduli space of stable curves*

Jiayue Qi[0000−0003−0542−4972]

[1] Doctoral Program "Computational Mathematics" W1214, Johannes Kepler University Linz.
[2] Research Institute for Symbolic Computation, Johannes Kepler University Linz.
jiayue.qi@dk-compmath.jku.at

Abstract. We introduce an algorithm for computing the value of all monomials in the Chow group of zero cycles in the moduli space of stable curves. This computation is a concrete but complicated algorithmic question in the field. (The algorithm was published as an extended abstract, see [1].)

Keywords: Monomials in the Chow ring · Tree representations · Recursive algorithm on a forest.

1 Introduction

Let $n \in \mathbb{N}$, $n \geq 3$, set $N := \{1, \ldots, n\}$. $\mathcal{M}_n$ denotes the moduli space of stable n-pointed curves of genus zero. A bipartition $\{I, J\}$ of N where both cardinalities of I and J are at least 2 is called a **cut**; I and J are **parts** of this cut. For every cut $\{I, J\}$, there is a variety $D_{I,J}$ in $\mathcal{M}_n$; denote by $\delta_{I,J}$ its corresponding element in the Chow ring. It is a graded ring — denote it as $A^*(n)$ — we have $A^*(n) = \bigoplus_{r=0}^{n-3} A^r(n)$. These homogeneous components are defined as Chow groups (of M_n); $A^r(n)$ is the **Chow group of rank** r. It is known that $A^r(n) = \{0\}$ for $r > n-3$ and $A^{n-3}(n) \cong \mathbb{Z}$. We denote this isomorphism by $\int : A^{n-3}(n) \to \mathbb{Z}$.

The set $\{\delta_{I,J} \mid \{I, J\} \text{ is a cut }\}$ generates group $A^1(n)$, and also the whole ring $A^*(n)$ (when they are used as ring generators). Then, $\prod_{i=1}^{n-3} \delta_{I_i,J_i}$ can be viewed as an element in $A^{n-3}(n)$. Let $M := \prod_{i=1}^{n-3} \delta_{I_i,J_i}$, we define the **value of** M to be $\int(\prod_{i=1}^{n-3} \delta_{I_i,J_i})$. **In this paper we calculate the value of a given monomial** $M = \prod_{i=1}^{n-3} \delta_{I_i,J_i}$.

The problem originally showed up as a sub-problem for counting the realization of Laman graphs (minimally-rigid graphs) on a sphere [3], when we wanted to improve the algorithm given in [3]. We see this problem fundamental, standing on its own; we find the algorithm elegant and concise, may as well be helpful for other similar or even further-away problems. Therefore, we formulate it on its own. Our algorithm is implemented in Python, see [4].

* The research was funded by the Austrian Science Fund (FWF): W1214-N15, project DK9.

Among the generators of $A^*(n)$, we say that the two generators $\delta_{I_1,J_1}, \delta_{I_2,J_2}$ fulfill **Keel's quadratic relation** [2] if the following four conditions hold: $I_1 \cap I_2 \neq \emptyset$; $I_1 \cap J_2 \neq \emptyset$; $J_1 \cap I_2 \neq \emptyset$; $J_1 \cap J_2 \neq \emptyset$. In this case, $\delta_{I_1,J_1} \cdot \delta_{I_2,J_2} = 0$. Then we know that an easy case for our value computing problem is when two factors of the monomial fulfill Keel's quadratic relation — we simply get zero. Hence we only need to consider the monomials where no two factors fulfill Keel's quadratic relation; we call this type of monomials **tree monomials** since there is a one-to-one correspondence between these monomials and *loaded trees* (see Theorem 1).

Definition 1. *A* **loaded tree with** n **labels and** k **edges** *is a tree* (V, E) *together with a labeling function* $h : V \to 2^N$ *and an edge multiplicity function* $m : E \to \mathbb{N}^+$ *such that the following three conditions hold:*

1. $\{h(v)\}_{v \in V, h(v) \neq \emptyset}$ *form a partition of* N*;*
2. $\sum_{e \in E} m(e) = k$*;*
3. *For every* $v \in V$, $\deg(v) + |h(v)| \geq 3$*, note that here multiple edges are only counted once for the degree of its incident vertices.*

Theorem 1. *There is a one-to-one correspondence between tree monomials* $M = \prod_{i=1}^{k} \delta_{I_i,J_i}$ *and loaded trees with* n *labels and* k *edges, where* $I_i \cup J_i = N$ *for all* $1 \leq i \leq k$*.*

Correspondingly to the above theorem, we provide an algorithm obtaining the corresponding loaded tree of a given tree monomial, see Algorithm 1. Note that in this algorithm we restrict the ambient group to be $A^{n-3}(n)$, but it can apply to a much wider context — transferring any tree monomial to its corresponding loaded tree. After this step, we start from that loaded tree, go through several transformations, until finally the value of the given tree monomial is obtained.

Because of this one-to-one correspondence, we define **value of a loaded tree** T as $\int(M_T)$, where M_T is its corresponding monomial of T; denote it as $\int(T)$. Our calculus contains two parts. First: Check whether the given monomial contains a pair of factors which fulfills Keel's quadratic relation. If yes, the value of the monomial is 0; if no, we continue with the second part. We need the following known result to explain the second part.

Theorem 2. *If all factors are distinct in the tree monomial* $M = \Pi_{i=1}^{n-3} \delta_{I_i,J_i}$*, then* $\int(M) = 1$*. We call this type of tree monomials* ***clever monomials*** *and their corresponding loaded trees* ***clever trees****.*

In the second part, first step is to check whether the given monomial is a clever monomial. If yes, we know that its value is 1; if no, we go to the next step. The next step contains several sub-steps. Here we give the sketch of the second part: **Input:** a loaded tree with n labels and $n-3$ edges. **Output:** a natural number. (1) Transfer the loaded tree to a **semi-redundancy tree**. (2) Calculate the **sign of the tree value**. (3) Construct a **redundancy forest** from the semi-redundancy tree. (4) Apply a recursive algorithm to this redundancy forest,

Algorithm 1: monomial to tree

input : a tree monomial M in $A^{n-3}(n)$
output: a loaded tree with n labels and $n-3$ edges
$C \leftarrow$ collection of any cut that corresponds to some factor of M;
$P \leftarrow$ collection of all the parts of cuts in C;
$c \leftarrow$ any element $c = \{I, J\} \in C$;
for *each element* $p \in P \setminus \{I, J\}$ **do**
  if $p \subset I$ *or* $p \subset J$ **then**
    $c := c \cup \{p\}$
  end if
end for
$H \leftarrow$ the Hasse diagram of elements in c with respect to set containment order;
Consider H as a graph (V, E);
for *each vertex* V *of* H **do**
  Define labelling set $h(V)$ as its corresponding element in c;
  Update the labelling set: $h(V) := h(V) \setminus h(V_1)$ if V_1 is less than V in H
end for
$E := E \cup \{\{I, J\}\}$;
Attach this labelling function h to H;
Set the multiplicity function value $m(e)$ for each edge e as the power of its corresponding factor in M;
return $H = (V, E, h, m)$

obtaining the absolute tree value. (5) Product of the sign and absolute value gives us the tree value.

Now we explain those terminologies. Given a loaded tree $LT = (V, E, h, m)$. Define a weight function $w : V \cup E \to \mathbb{N}$ as follows: For any $v \in V$, $w(v) := \deg(v) + |h(v)| - 3$. For any $e \in E$, $w(e) := m(e) - 1$. Then the **semi-redundancy tree** (of LT) is $SRT := (V, E, w)$. Start from this semi-redundancy tree, let S be the sum of vertex weight (or edge weight) of LT. Then the **sign of the tree value** of loaded tree LT is $(-1)^S$. It is not hard to verify that weight sum of the edges and that of the vertices are the same when the given loaded tree has the number of labels three more than the sum of multiplicity of its edges.

Next, how do we transfer a semi-redundancy tree (V, E, w) (assume $LT = (V, E, h, m)$) to a redundancy forest? Replace each edge by a length-two edge with a new vertex connecting them carrying the same weight as the replaced edge. Then we obtain the redundancy tree (of loaded tree LT) $RT := (V \cup E, E_1, w_1)$. Deleting the weight-zero vertices and their adjacent edges from RT gives us the **redundancy forest** of LT.

Starting from the redundancy forest, we can compute its value (i.e. the absolute value of the loaded tree LT) by a recursive algorithm. Let $RF = (V, E, w)$ be the redundancy forest of a loaded tree LT. We define the **value of** RF as follows: Pick any leaf $l \in V$ of this forest, denote the unique parent of l as l_1. If $w(l) > w(l_1)$, return 0 and terminate the process; otherwise, remove l from RF and assign the weight $(w(l_1) - w(l))$ to l_1, replacing its previous weight. Denote

this new forest as RF_1. The value of RF is the product of binomial coefficient $\binom{w(l_1)}{w(l)}$ and the value of RF_1. Whenever we reach a degree-zero vertex, if it has non-zero weight, return 0 and terminate the process; otherwise return 1. Product of the value of the corresponding redundancy forest of LT and sign of its tree value gives us the value of LT. In the sequel, we explain this process with an example, to show it more intuitively.

Example 1. Given a loaded tree LT (leftmost of Figure 1). Follow the definition (construction) of semi-redundancy tree, we obtain the tree in the middle of Figure 1. Sum of vertex weight $S = 1+4+1+0+1 = 7$, so the sign of LT value is $(-1)^7 = -1$. From SRT, follow the construction for the redundancy forest, we obtain the forest shown on the rightmost of Figure 1. Finally we get the absolute value of RF as $[\binom{1}{1} \times 1] \times [\binom{2}{1} \times \binom{4}{1} \times \binom{4}{3} \times \binom{1}{1} \times 1] = 32$. Combining with the sign -1, we obtain the value of LT as -32.

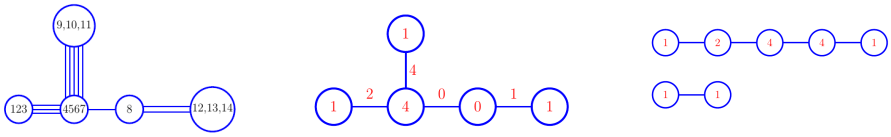

Fig. 1: Leftmost is the loaded tree LT with 14 labels and 11 edges, with labels tagged in black. In the middle is the semi-redundancy tree SRT of LT, while rightmost is the redundancy forest RF, both with weight function marked in red.

It is not hard to verify that the process of our calculus terminates. It is well-defined — the result is independent of the sequence of leaves we choose (to apply the recursive algorithm on the redundancy forest) — because of the following identity in binomial coeffients: $\binom{c}{a} \cdot \binom{c-a}{b} \equiv \binom{c}{b} \cdot \binom{c-b}{a}$. Therefore it is indeed an algorithm. We call it the **forest algorithm**.

Theorem 3. *The forest algorithm calculates the value of a given loaded tree, or equivalently, the value of its corresponding (tree) monomial.*

Consider an input of $n-3$ many generators $\delta_{I,J}$. The first part of our algorithm is polynomial in n, while the second part is linear in n. Our algorithm is polynomial in n, more precisely, linear in n^2.

Acknowledgement

I thank Josef Schicho for introducing to me the problem background, the general discussion, particularly for helping me with the proof of correctness of forest algorithm.

References

1. Jiayue Qi. A calculus for monomials in Chow group of zero cycles in the moduli space of stable curves. *ACM Communications in Computer Algebra*, 54.3 (2021): 91-94.
2. Sean Keel. Intersection theory of moduli space of stable n-pointed vurves of genus zero. *Transaction of the American Mathematical Society*, **330** (1992), no. 2, 545-574.
3. M. Gallet, G. Grassegger, J. Schicho. Counting realizations of Laman graphs on the sphere. *The Electronic Journal of Combinatorics*, Volume 27, Issue 2 (2020).
4. `https://github.com/muronghezi/integral-chow-ring-monomial`