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

Jiayue Qi[1]

Research Institute for Symbolic Computation

CASC 2022, Gebze, Turkey

[1] This research was funded by the Austrian Science Fund (FWF): W1214-N15, project DK9.

basic setting

- Let $n \in \mathbb{N}$, $n \geq 3$, we call $N := \{1, \ldots, n\}$ the **labeling set** and elements of N **labels**.
- A bipartition $\{I, J\}$ of N where the cardinalities of I and J are both at least 2 is called a **cut** (of M_n). And I, J are called two **parts** of the cut $\{I, J\}$.
- This talk focus on the Chow ring of M_n, where M_n is the moduli space of n-marked stable curves of genus zero.
- For each bipartition $\{I, J\}$, there exists a codimension one hypersurface $D_{I,J} \subset M_n$. Denote by $\delta_{I,J}$ the class of $D_{I,J}$.
- However, we will not focus on the details of M_n in this talk, what is important here is the properties of this Chow ring.
- We denote the Chow ring of M_n as $A^*(n)$.

ambient ring

- It is a graded ring with $n-2$ homogeneous pieces $A^*(n) = \bigoplus_{k=0}^{n-3} A^k(n)$; the homogeneous component $A^r(n)$ is the **Chow group of rank** r.
- Fact 1: $A^r(n) = \{0\}$ for $r > n-3$.
- Fact 2: $A^{n-3}(n) \cong \mathbb{Z}$, we denote this isomorphism as $\int : A^{n-3}(n) \longrightarrow \mathbb{Z}$; we call the image under this map the **integral value** of the given monomial.
- $\{\delta_{I,J} \,|\, \{I,J\} \text{ is a cut}\}$ is a set of generators for $A^1(n)$; they are also generators for $A^*(n)$, when viewed as ring generators. The product $\prod_{i=1}^{n-3} \delta_{I_i,J_i}$ is an element in $A^{n-3}(n)$.
- Goal: calculate the integral value of this monomial, i.e., $\int(\prod_{i=1}^{n-3} \delta_{I_i,J_i})$.

motivation

- This calculus shows up as a subproblem when we want to improve an algorithm for realization-counting of Laman graphs on the sphere.
- With the help of this integral value calculus, we invent another algorithm for the same goal.
- However, by efficiency it does not seem faster or better than the existing one.
- But we see that this problem is fundamental, standing on its own, and may be helpful for other problems in the future.
- Then we focus on it, and formalize it as a self-contained result.

Keel's quadratic relation

Among the generators of $A^*(n)$, we say the two generators $\delta_{I_1,J_1}, \delta_{I_2,J_2}$ fulfill **Keel's quadratic relation** if the following conditions hold:

- $I_1 \cap I_2 \neq \emptyset$;
- $I_1 \cap J_2 \neq \emptyset$;
- $J_1 \cap I_2 \neq \emptyset$;
- $J_1 \cap J_2 \neq \emptyset$.

And when they are fulfilled, we have $\delta_{I_1,J_1} \cdot \delta_{I_2,J_2} = 0$. In this case, the ambient varieties have empty intersection.

- An easy example: when $n = 5$, $\delta_{12,345} \cdot \delta_{13,245} = 0$ but $\delta_{12,345}$ and $\delta_{123,45}$ does not fulfill this relation.

Keel's quadratic relation

- Inspired by this property, we know that if any two factors of the monomial fulfills this relation, the whole integral will be zero.
- Now we only need to focus on those monomials where no two factors fulfill this quadratic relation, we call those monomials **tree monomials**.
- Since there is a one-to-one correspondence between these monomials and a type of tree, which we define as **loaded tree**.

loaded tree

A **loaded tree with n labels and k edges** is a tree (V, E, h, m), where h denotes the labeling function from V to the power set of N and m denotes the multiplicity function for edges. The following conditions must hold:

- Non-empty labels $\{h(v)\}_{v \in V}$ form a partition of N;
- Number of edges is k, edges are counted with multiplicity, i.e., $\sum_{e \in E} m(e) = k$;
- $\deg(v) + |h(v)| \geq 3$ holds for every $v \in V$.

This loaded tree would correspond to a monomial in $A^k(n)$. In the classic notation, this concept coincides with the dual tree of an element in the moduli space M_n, but allowing multiple edges.

loaded tree: examples

Figure: This is a loaded tree with 5 labels and 2 edges.

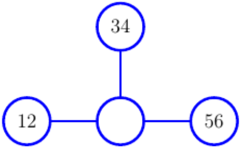


Figure: This is a loaded tree with 6 labels and 3 edges.

monomial of a given tree

- We define the **monomial of a given loaded tree** as follows:
- Remove an edge e, we collect the labels in the two connected components respectively to form I and J. And we say $\{I, J\}$ is the corresponding cut for the edge e.
- The monomial of this given loaded tree (V, E, h, m) is $\prod_{e \in E}^{m(e)} \delta_{I,J}$, where $\{I, J\}$ is the corresponding cut of edge e, and $m(e)$ is the multiplicity of e.
- We can see that it is well-defined and each loaded tree has a unique monomial representation.

monomial of a given tree

Figure: This is a loaded tree with 5 labels and 2 edges. Its corresponding monomial: $\delta_{12,345} \cdot \delta_{123,45}$.

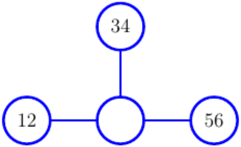


Figure: This is a loaded tree with 6 labels and 3 edges. Its corresponding monomial: $\delta_{34,1256} \cdot \delta_{12,3456} \cdot \delta_{56,1234}$.

one-to-one correspondence

Theorem

There is a one to one correspondence between tree monomials $T = \prod_{i=1}^{m} \delta_{I_i,J_i} (1 \leq m \leq n-3)$ and loaded trees with n labels and m edges.

We also have an algorithm converting the monomial to tree, we call it *tree algorithm*.

tree algorithm

- Input: a tree monomial M in $A^k(n)$
- Output: a loaded tree with n labels and k edges
- Step 1: collect all cuts in each factor of the monomial in set C.
- Step 2: collect all parts of those cuts in set P.
- Step 3: pick any cut from set C, say $c = (I, J) \in C$.
- Step 4: go through all elements in P, find those that is either a subset of I or a subset of J, collect them together in set P_1.
- Step 5: create a Hasse diagram H of elements in P_1 w.r.t. set containment order.
- Step 6: consider H as a graph (V, E). Each element in P_1 has a corresponding vertex in H. We denote the vertex v_I for $I \in P_1$.

tree algorithm

- Step 7: For each vertex v of H, define the labeling set $h(v)$ as its corresponding element in P_1.
- Step 8: Go through the vertices again, update the labeling function: $h(v) := h(v) \setminus h(v_1)$ if v_1 is less than v in H (in the Hasse diagram relation).
- Step 9: $E = E \cup \{v_I, v_J\}$. This edge corresponds to the cut we pick in Step 3.
- Step 10: set the multiplicity value $m(e)$ for each edge e as the power of its corresponding factor in M.
- Step 11: return $H = (V, E, h, m)$.

tree algorithm: an example

Example

- Given a tree monomial (in $A^6(9)$)
  $\delta^3_{123,456789} \cdot \delta_{12345,6789} \cdot \delta_{1234589,67} \cdot \delta_{1234567,89}$.
- Then we obtain the labeling set $N := \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- We collect the parts to set
  $P := \{123, 456789, 12345, 6789, 1234589, 67, 1234567, 89\}$
  and we pick any cut $c = \{12345, 6789\}$.
- Then we collect together all parts which are either contained in 12345 or 6789, we obtain the set
  $P_1 = \{12345, 6789, 123, 67, 89\}$.
- Note that for convenience, we simplify the set notation sometimes. For instance, by 123 we mean the set $\{1, 2, 3\}$.
- Then we construct the corresponding Hasse diagram H for the set P_1, see the figure below.

tree algorithm: an example

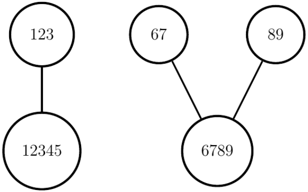


Figure: This is the Hasse diagram of set $\{12345, 6789, 123, 67, 89\}$ with respect to set containment order.

- Now, we still need to update the labeling function for each vertex. $h(v) := h(v) \setminus h(v_1)$ if v_1 is less than v in H (in the Hasse diagram relation).
- Another mission is to attach edge multiplicity to each edge, simply by copying the power of the corresponding factor in M.

tree algorithm: an example

Example

- The corresponding loaded tree see the figure below.
- It is easy to see that if we go back from the tree constructing monomial, we get the same one as the given one.

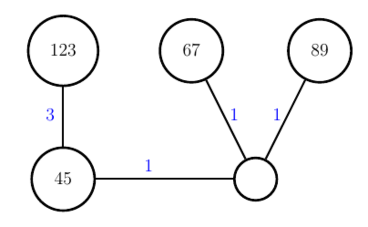


Figure: This is the corresponding loaded tree of monomial $\delta^3_{123,456789} \cdot \delta_{12345,6789} \cdot \delta_{1234589,67} \cdot \delta_{1234567,89}$. Multiplicity function values are written in blue.

the calculus (first half)

- Input: $M := \prod_{i=1}^{n-3} \delta_{I_i,J_i}$. (any monomial in $A^{n-3}(n)$)
- Output: the integral value of the given monomial, $\int(\prod_{i=1}^{n-3} \delta_{I_i,J_i})$, which is an integer.
- Step 1: Check if any two factors of M fulfill Keel's quadratic relation. If yes, return 0, terminate the process. Otherwise, continue. This step is in the worst case quadratic in n.
- Step 2: Apply tree algorithm to the monomial, transfer it to a loaded tree (with n labels and $n-3$ edges). As far as I know, constructing a Hasse diagram is at most quadratic.

Hence, the first part of our calculus is at most quadratic in n.

the calculus – second half

- Input: a loaded tree LT with n labels and $n-3$ edges.
- Output: the integral value of its corresponding monomial, which is an integer.
- This half mainly contains two parts, one for the absolute value and one for the sign.
- We will show it with a running example.

weighted tree

- Given a loaded tree $LT = (V, E, h, m)$.
- We define its corresponding **weighted tree** $WT = (V, E, w)$ by attaching a weight function to each vertex and edge.
- $w(e) := m(e) - 1$ and $w(v) := \deg(v) + |h(v)| - 3$.
- Assume $WT = (V, E, w)$ is **a weighted tree of some loaded tree with n labels and $n - 3$ edges**, then we can verify the following identity about the weight function w.
- $\sum_{v \in V} w(v) = \sum_{e \in E} w(e)$.

weight identity

$$
\begin{aligned}
\sum_{v\in V} w(v) &= \sum_{v\in V} (\deg(v) + |h(v)| - 3) \\
&= \sum_{v\in V} \deg(v) + \sum_{v\in V} |h(v)| - 3\cdot|V| \\
&= 2\cdot|E| + n - 3\cdot|V| \\
&= 2\cdot|E| + n - 3\cdot|E| - 3 \\
&= n - 3 - |E| \\
\sum_{e\in E} w(e) &= \sum_{e\in E} (m(e) - 1) \\
&= \sum_{e\in E} m(e) - |E| \\
&= n - 3 - |E|
\end{aligned}
$$

the calculus – second half: running example

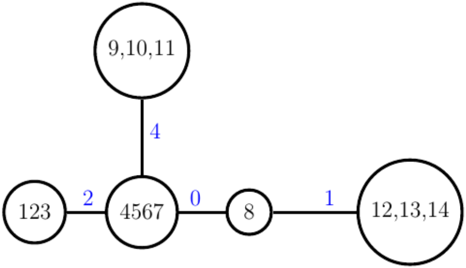


Figure: This is a loaded tree LT with 14 labels and 11 edges.

- **Step 1: Transfer it to a weighted tree.**
- Recall: $w(e) := m(e) - 1$ and $w(v) := \deg(v) + |h(v)| - 3$.

running example: weighted tree

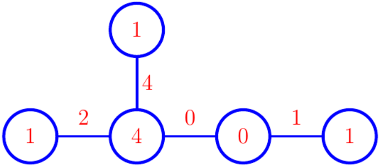


Figure: This is the weighted tree WT of the loaded tree LT, where the weights of vertices and edges are tagged in red.

Step 2: Compute the sign, which is $(-1)^S$. Here S denotes the weight sum of vertices (or equivalently, of edges) of WT.

running example: sign

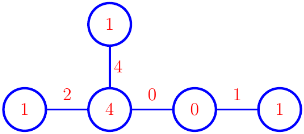


Figure: This is the weighted tree of the loaded tree LT, where the weights of vertices and edges are tagged in red.

Sum of vertex weight $S = 1 + 4 + 1 + 0 + 1 = 7$, so the sign of the monomial value is $(-1)^7 = -1$.

redundancy tree

- Step 3: Replace each edge by a length-two edge with a new vertex connecting them which has the same weight as the replaced edge.
- Then we obtain the redundancy tree RT (of loaded tree LT).

running example: redundancy tree

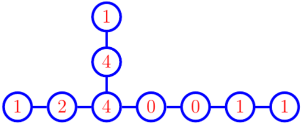


Figure: This is the redundancy tree RT of loaded tree LT, the weights of vertices are tagged in red.

Step 4: Omit those vertices with weight zero and their adjacent edges, we obtain the redundancy forest of LT.

running example: redundancy forest

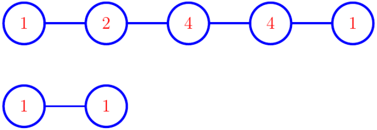


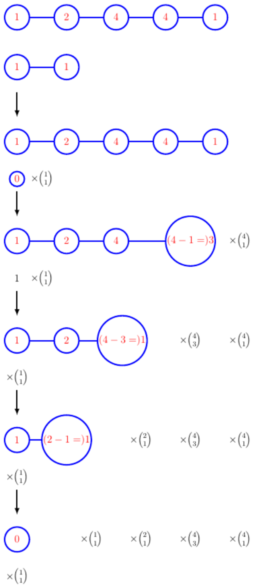
Figure: This is the redundancy forest *RF* of loaded tree *LT*, which contains two trees and the weight of vertices of are tagged in red.

Step 5: Apply a recursive algorithm to the redundancy forest, obtaining the absolute value (of the integral value).

recursive algorithm?

- Let $RF = (V, E, w)$ be the redundancy forest of a loaded tree LT.
- We define the value of RF as the following:
- Pick any leaf of this forest, say $l \in V$, 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 a new weight $(w(l_1) - w(l))$ to l_1, replacing its previous weight. Denote the new forest by RF_1.
- The value of RF is defined to be the product of binomial coefficient $\binom{w(l_1)}{w(l)}$ and the value of RF_1.
- Base cases: whenever we reach a degree-zero vertex, if it has non-zero weight, return 0 and terminate the process; otherwise, return 1.
- Value of RF is then the absolute value of LT.

running example: absolute value



running example: integral value

- Finally we get the absolute value as $1 \times \binom{1}{1} \times \binom{2}{1} \times \binom{4}{3} \times \binom{4}{1} \times \binom{1}{1} = 32$.
- Combining with the sign -1, we obtain the value of LT as -32.

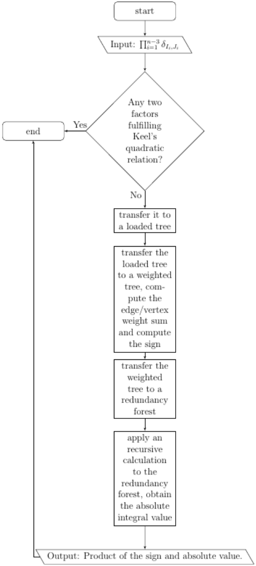
Step 6: Product of the sign and absolute value gives us tree value.

the calculus – second half

- Input: a loaded tree with n labels and $n-3$ edges.
- Output: the integral value of the given loaded tree.
- Transfer the loaded tree to a weighted tree.
- Calculate the sign of the integral value.
- Transfer the weighted tree to a redundancy forest.
- Apply the recursive algorithm to this redundancy forest, obtaining the absolute integral value.
- Product of the sign and absolute value gives us the integral value.

We call this part of the calculus the **forest algorithm**. This part is linear in n. The calculus is then in the worst case quadratic in n.

the calculus – flow chart



correctness

Theorem

The forest algorithm is correct.

Reference

I thank Prof. Josef Schicho for helping me with the correctness proof of the forest algorithm.

Jiayue Qi.
A graphical algorithm for the integration of monomials in the Chow ring of the moduli space of stable marked curves of genus zero. preprint arXiv:2102.03575

Thank You