A calculus for monomials in Chow group $A^{n-3}(n)$

Jiayue Qi

Joint work with Josef Schicho (University of Linz)

2020.07.02. Fika Webinar KTH

Doctoral Program
Computational Mathematics
Numerical Analysis and Symbolic Computation

JOHANNES KEPLER
UNIVERSITY LINZ

Der Wissenschaftsfonds.

basic setting

- Let $n \in \mathbb{N}$, $n \geq 3$, set $N := \{1, \ldots, n\}$.
- A partition (I, J) of N where both cardinality of I and J are 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 stable n-pointed curves of genus zero.
- Denote $\delta_{I,J}$ as the class of a cut subvariety $D_{I,J}$ of M_n.
- We will not focus on the details of M_n, what is important for this talk is the properties of this Chow ring.
- We denote the Chow ring of M_n as $A^*(n)$.

basic setting

- It is a graded ring, we have $A^*(n) = \bigoplus_{k=0}^{n-3} A^k(n)$; and these homogeneous components are defined as Chow groups (of M_n). Here, for instance, we say $A^r(n)$ is a **Chow group of rank** r.
- Fact1: $A^r(n) = \{0\}$ for $r > n-3$.
- Fact2: $A^{n-3}(n) \cong \mathbb{Z}$, we denote this isomorphism as $\int : A^{n-3}(n) \longrightarrow \mathbb{Z}$.
- $\{\delta_{I,J} \mid \{I,J\} \text{ is a cut}\}$ is a set of generators for $A^1(n)$; they are also generators for $A^*(n)$.
- $\prod_{i=1}^{n-3} \delta_{I_i,J_i}$ can be viewed as an element in $A^{n-3}(n)$ since we are in a graded ring.
- Goal: calculate the integral value of this monomial, i.e., $\int(\prod_{i=1}^{n-3} \delta_{I_i,J_i})$.

motivation

- For me, this calculus shows up as a subproblem when I want to improve an algorithm for realization-counting of Laman graphs on the sphere.
- With the help of the integral value calculation, I 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, may be helpful for other similar problems, or even further-away problems.
- Then we focus on it, and try to formalize it as a result on its own.

two important properties of $A^*(n)$

- Quadratic relations between the generators.
- Linear relations between the generators.

Keel's quadratic relation

Among the generators of $A^*(n)$, $\delta_{I_1,J_1} \cdot \delta_{I_2,J_2} = 0$ and we say these two generators fulfill **Keel's quadratic relation** if the following conditions hold:

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

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 monomial**.
- This name also has a reason!
- 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 $v \in V$.

(Hint: this tree would correspond to a monomial in the Chow group $A^k(n)$.)

loaded tree

See some examples of loaded trees. (check with definitions)

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

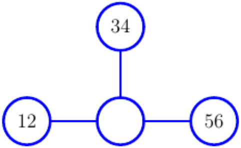


Figure: This is a loaded tree with 6 labels and 3 edges.

monomial of a given tree

- We define the **monomial of a given loaded tree** as the following:
- For each edge we collect the labels on one side of it to form I and labels on the other side of it to form J. And we say (I, J) is the corresponding cut for this edge.
- The monomial of this given loaded tree is $\prod_{i=1}^{m} \delta_{I_i, J_i}$, where m is the number of edges.
- Each edge of the tree contributes to the monomial a factor $\delta_{I,J}$ if (I, J) is the corresponding cut for this edge.
- 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, the corresponding tree of tree monomial $\delta_{12|345} \cdot \delta_{123|45}$.

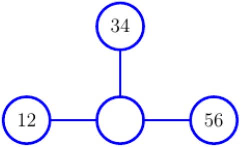


Figure: This is a loaded tree with 6 labels and 3 edges, the corresponding tree of tree monomial $\delta_{34|1256} \cdot \delta_{12|3456} \cdot \delta_{56|1234}$.

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 call the corresponding tree of a tree monomial **tree of the given tree monomial**.

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 cubset 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

Let's see an example.

Example

- Given a tree monomial $\delta^3_{123,456789} \cdot \delta_{12345,6789} \cdot \delta_{1234589,67} \cdot \delta_{1234567,89}$.
- Obviously we have the labeling set $N := \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- We collect the parts in set $P := \{123, 456789, 12345, 6789, 1234589, 67, 1234567, 89\}$ and we pick any cut $c = \{12345, 6789\}$.
- After collecting all parts which are either contained in 12345 or 6789, we obtain $P_1 = \{12345, 6789, 123, 67, 89\}$.
- Then we construct the corresponding Hasse diagram for c, see the figure below.

tree algorithm: an example

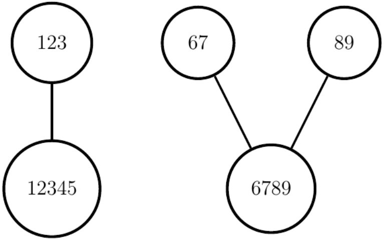


Figure: This is the Hasse diagram of set $\{12345, 6789, 123, 67, 89\}$ with respect to set containment order.

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.

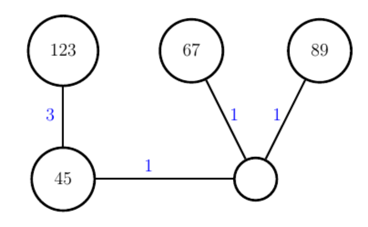


Figure: This is the corresponding loaded tree of monomial $\delta^3_{123,456789} \cdot \delta_{12345,6789} \cdot \delta_{1234589,67} \cdot \delta_{1234567,89}$. Multiplicity function values are written in blue.

value of a loaded tree

- Goal: Calculate $\int(T)$ for any tree monomial $T \in A^{n-3}(n)$.
- Recall: $\int$ represents the isomorphism from $A^{n-3}(n)$ to $\mathbb{Z}$
- Because of this one-to-one correspondence, now we define **value of a loaded tree** as $\int(T)$, where T is the corresponding monomial of this loaded tree.
- Given a loaded tree with n labels and $n-3$ edges, we want to calculate its value.

weighted tree

- $LT = (V, E, h)$: loaded tree with n labels and k edges.
- its corresponding *weighted tree*: $WT = (V, E, w)$.
- weight function $w : V \cup E \to \mathbb{N}$, $w(v) := deg(v) + |h(v)| - 3$ for all $v \in V$ and $w(e) :=$multipicity$(e) - 1$ for all $e \in E$.
- Tree monomial $\longrightarrow$ Loaded tree $\longrightarrow$ Weighted tree
- Since permuting/renaming the labels doesn't change the integral value, we see that if two monomial have the same weighted tree, they also must have the same value.

weighted tree

- Given a weighted tree of some tree monomial, its value is defined to be the value of the monomial.
- Our goal can also be to calculate the value of the weighted tree (of some loaded tree with n labels and $n-3$ edges).
- 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

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

$$\begin{aligned}\sum_{e\in E} w(e) &= \sum_{e\in E} (multiplicity(e) - 1) \\ &= \sum_{e\in E} multiplicity(e) - |E| \\ &= n - 3 - |E|\end{aligned}$$

clever tree

For tree monomials in $A^{n-3}(n)$, there is a known result.

Theorem

If all factors are distinct in $T := \prod_{i=1}^{n-3} \delta_{I_i, J_i}$*, then* $\int(T) = 1$*. We call this type of tree monomial* **clever monomial** *and its corresponding loaded tree* **clever tree***.*

Remark

For clever trees, we know that they have value 1*. What about non-clever trees? Stage time for Keel's linear relation.*

Recall: two important properties of $A^*(n)$

- Quadratic relations between the generators.
- Linear relations between the generators.

Keel's linear relation

Denote $\epsilon_{ij|kl} := \sum_{i,j\in I,k,l\in J} \delta_{I,J}$. Then we have the equality relations $\epsilon_{ij|kl} = \epsilon_{il|kj} = \epsilon_{ik|jl}$, we call it **Keel's linear relation**.

Example

When $n = 6$, we have $\epsilon_{12|35} = \epsilon_{13|25} = \epsilon_{15|23}$, i.e.,

$$\delta_{12,3456} + \delta_{124,356} + \delta_{126,345} + \delta_{1246,35}$$

$$= \delta_{13,2456} + \delta_{134,256} + \delta_{136,245} + \delta_{1346,25}$$

$$= \delta_{15,2346} + \delta_{145,236} + \delta_{156,234} + \delta_{1456,23}$$

Remark

In the motivation, we mention the realization-counting of Laman graphs pn the sphere, actually this integral value is the key thing to solve there: $\int \prod_{r=1}^{n-3} \epsilon_{i_r j_r|k_r l_r}$. *We can then see that the goal in our talk is indeed a subproblem of it.*

Keel's linear relation

Example

When $n = 6$, we have $\epsilon_{12|35} = \epsilon_{13|25} = \epsilon_{15|23}$, i.e.,

$$\begin{aligned} & \delta_{12,3456} + \delta_{124,356} + \delta_{126,345} + \delta_{1246,35} \\ = & \delta_{13,2456} + \delta_{134,256} + \delta_{136,245} + \delta_{1346,25} \\ = & \delta_{15,2346} + \delta_{145,236} + \delta_{156,234} + \delta_{1456,23} \end{aligned}$$

Remark

From the example above we easily see that we can replace some $\delta_{I,J}$, say $\delta_{12|3456}$, by $\epsilon_{13|25} - (\epsilon_{12|35} - \delta_{12|3456})$. Basicly we can replace $\delta_{I,J}$ by a sum of $(2^{n-3} - 1)$ many $(\pm)\delta_{I',J'}$.

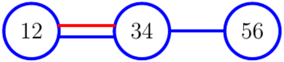
non-clever trees?

Example

- Given: $\delta^2_{12|3456} \cdot \delta_{1234|56}$
- Corresponding tree see below, note that for the multiplicity function of edges, sometimes we draw all the edges down, sometimes just write the multiplicity besides it.
- use Keel's linear relation:

$$\delta^2_{12|3456} \cdot \delta_{1234|56} = \delta_{12|3456} \cdot \delta_{1234|56} \cdot (\epsilon_{13|25} - \delta_{124|356} - \delta_{126|345} - \delta_{1246|35})$$

- After cancellations caused by Keel's quadratic relation, we get $\delta^2_{12|3456} \cdot \delta_{1234|56} = -\delta_{12|3456} \cdot \delta_{1234|56} \cdot \delta_{124|356}$.
- obtain tree value/monomial value: -1



non-clever trees?

- Above is the newly generated tree in the example.
- We can also see from the process that whenever we substitute some term with the linear relation, we create a negative sign for the value.
- We call it **one-step reduction**.
- Meanwhile, we generate one or more loaded tree of the same type.
- However, the weight sum of the edges (equivalently, of the vertices) is reduced by one.

non-clever trees?

- Then we apply again this reduction to each one of the newly generated trees.
- When at some stage, it is reduced to zero, we directly know the absolute value, by counting the number of clever trees in total.
- As for the sign, it is simply -1 to the power of weight sum of the given loaded tree.
- In this example, weight sum is reduced from 1 to 0.
- The sign is $(-1)^1 = -1$ and one clever tree is generated. Therefore, the value of the given tree is -1.
- We can also see that in order to compute the value of a loaded tree, the tricky part is to figure out its absolute value.
- Based on this idea, we have an algorithm for computing all tree monomials in $A^{n-3}(n)$. We call it **forest algorithm**.

forest algorithm

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

running example: loaded tree

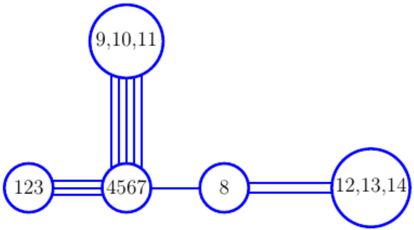


Figure: This is a loaded tree LT with 14 labels and 11 edges.

Let's figure out its weighted tree!

running example: weighted tree

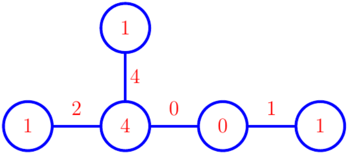


Figure: This is the weighted tree of the loaded tree LT, where the weight of vertices and edges are tagged in red.

running example: sign of the tree value

- Given a weighted tree $WT = (V, E, w)$.
- Let S be the sum of vertex weight (or equivalently, sum of edge weight) of LT.
- Sign of the tree value is $(-1)^S$.

sign of the tree value

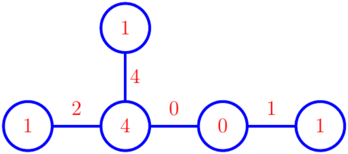


Figure: This is the weighted tree of the loaded tree LT, where the weight of vertices and edges are tagged in red.

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

redundancy forest

- How do we transfer a weighted tree (V, E, w) (assume its corresponding loaded tree $LT = (V, E, h)$) to a redundancy forest?
- 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 (of loaded tree LT) $RT := (V \cup E, E_1, w_1)$.
- Omit those vertices with weight zero and their adjacent edges, we then obtain the redundancy forest of LT.

running example: weighted tree

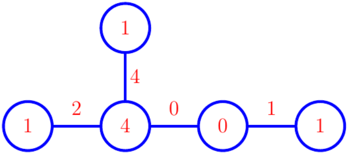


Figure: This is the weighted tree of the loaded tree LT, where the weight of vertices and edges are tagged in red.

- First let's figure out its redundancy tree on the whiteboard!
- Let's figure out its redundancy forest!

running example: redundancy forest

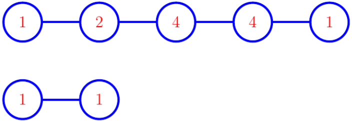


Figure: This is the redundancy forest RF of loaded tree LT, which contains two trees and the weight of vertices of RF are tagged in red.

What is the recursive algorithm mentioned above for absolute tree 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 weight $(w(l_1) - w(l))$ to l_1, replacing its previous weight. Denote the new forest as RF_1.
- Value of RF is 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.

absolute value

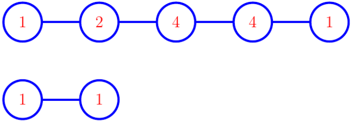
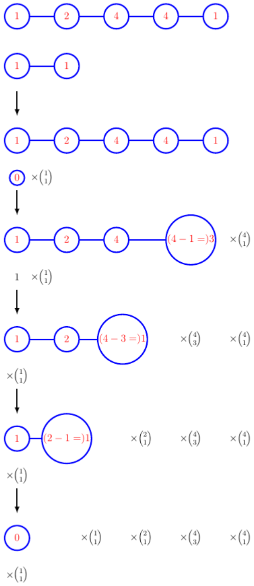


Figure: This is the redundancy forest *RF* of loaded tree *LT*, which contains two trees and the weight of vertices of *RF* are tagged in red.

Let's figure out how to apply the recursive algorithm to obtain the absolute value!

absolute value



tree value

- Finally we get the absolute value of RF 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.

forest algorithm

- Input: a loaded tree with n labels and $n-3$ edges
- Output: a natural number
- Transfer the loaded tree to a weighted tree.
- Calculate the sign of the tree value.
- Construct a redundancy forest from the weighted tree.
- Apply a recursive algorithm to this redundancy forest, obtaining the absolute tree value.
- Product of the sign and absolute value gives us tree value.
- Implemented in Python; based on forest algorithm, computation of $\int \prod_{r=1}^{n-3} \epsilon_{i_r j_r | k_r l_r}$ is also implemented in Python.

well-definedness; termination

- Not hard to verify that at every step it does not matter from which leaf we start and base cases are well-defined. Hence forest algorithm is well-defined.
- Input is a tree with $(n-2)$ vertices maximally, the redundancy forest can have at most $(n-2)+(n-3)=2n-5$ many vertices, which is finite.
- The recursive algorithm strictly reduces the number of vertices by 1 in each step, obtaining a proper sub-forest.
- Hence the algorithm terminates and is well-defined.

correctness

Theorem

Forest algorithm is correct.

Thank You

Happy Birthday Kathlén