EasyChair Preprint

№ 8975

F-Polynomials and Newton Polytopes

Gleb Koshevoy and Denis Mironov

October 3, 2022

F-POLYNOMIALS AND NEWTON POLYTOPES

GLEB KOSHEVOY AND DENIS MIRONOV *

Abstract. We provide an effective algorithmic method for computation of Gross-Keel-Hacking-Kontsevich potential, F-polynomials and Bernstein-Kazhdan decoration function and it's complexity bounds. For simply laced Lie algebras we make conjecture and provide experimental evidence that Newton polytopes for Gross-Keel-Hacking-Kontsevich potential do not contain any interior lattice points.

1. F-polynomials and Gross-Hacking-Keel-Kontsevich potentials

Let G be a group with the Lie algebra of simply-laced type, B_+ and B_- be its Borel subgroups, with the set of simple roots α_a, $a \in I$, W the Weyl group. The Gross-Hacking-Keel-Kontsevich potential (GHKK for short) W_{GHKK} is a function on the double Bruhat cell $G^{w_0,e} = B_- \cap B_+ \overline{w_0} B_+$, defined using cluster algebra's tools [6]. Because of validity the Fock-Goncharov conjecture in such cases [5], we get the polyhedral parametrization of canonical bases of the ring of regular functions on G/B arising from the tropicalizations of the potential.

Specifically, the ring of regular functions on the double Bruhat cell is endowed with the cluster algebra structure. Namely, for a reduced decomposition $\mathbf{i}$ of the longest element $w_0 \in W$ with length N, let $\Sigma_{\mathbf{i}}$ be a corresponding X-cluster seed and $Q_{\mathbf{i}}$ be the corresponding quiver (due to [1]). Then W_{GHKK} is a polynomial in the cluster variables $\Sigma_{\mathbf{i}}$([10]).

Namely, a seed Σ with underlying quiver Q is *optimal for a frozen vertex* $a \in I$ if after deleting arrows between frozen vertices and a, the vertex a becomes a source of the quiver Q.

For the optimal seed Σ, the ath part of the GHKK-potential is equal to the value of the corresponding frozen,

$$W_a = Y_a. \tag{1}$$

For a frozen $a \in I$, there exists an appropriate reduced word $\mathbf{i}'$, such that $\Sigma_{\mathbf{i}'}$ is an optimal seed for a.

Because of that, for a given reduced decomposition $\mathbf{i}$, one can compute the half of the GHKK-potential

$$W_{GHKK} = \sum_{a \in I} W_a$$

using cluster mutations corresponding to 3-braid moves between the reduced decompositions of w_0 (for l and k, such that $a_{lk} = -1$, $s_k s_l s_k = s_l s_k s_l$). Namely, for computing W_a, we apply a sequence of cluster mutations corresponding to 3-braid moves which transform $\Sigma_{\mathbf{i}}$ into an

Institute for Information Transmission Problems Russian Academy of Sciences, Russian Federation

Moscow center for Continuous Mathematical Education

*Corresponding author

E-mail addresses: koshevoyga@gmail.com, mironovd@poncelet.ru.

This work was supported by the Russian Science Foundation under grant no. 21-11-00283.

optimal seeds for a, then W_a is the X-cluster variable at the frozen vertex labeled by a in the optimal seed computed in the variables of the seed $\Sigma_{\mathbf{i}}$. In variables of the seed $\Sigma_{\mathbf{i}}$, such an X-cluster variable is equal to the specification of the F-polynomial (see [4, 9]) and takes the form

$$W_a = Y_1^{c_{1a}(t)} \cdots Y_N^{c_{Na}(t)} \prod_i F_i(t)(Y_1, \cdots, Y_N)^{b_{ia}(t)}. \quad (2)$$

In the above formula we take notations of [9], where t means the end vertex of the path in the mutation graph from the optimal seed for a to $\Sigma_{\mathbf{i}}$ and Y_j's are cluster variables of $\Sigma_{\mathbf{i}}$.

Precisely (see [10]) W_a is of the form product of the frozen $Y_a(t)$ and an F-polynomial.

2. Newton polytopes

We are interested of properties of the Newton polytopes of the individual terms W_a, $a \in I$, of the half-potential W_{GHHK}, as well as the Newton polytope of W_{GHHK}.

Fei in [3] conjectured that the Newton polytope of an F-polynomial has no interior integer points.

For minuscule weight $a \in I$, the validity of this conjecture for F-polynomials corresponding to terms W_a follows from Remark 5.17 [7]. Namely in such a case, the Newton polytope is a geometric realisation of a distributive lattice of the corresponding decorated graph DG_a. Since such a polytope is a convex hull of a subset of the vertices of a unit cube the claim follows.

An integer polytope without interior integer points is *void*.

We state the following

Conjecture. For a simply-laced group G, and any reduced decomposition $\mathbf{i}$ of w_0, the Newton polytope of W_{GHKK} is void.

Note that validity of this conjecture implies that the Newton polytopes of F-polynomials for frozens are void.

We made computer verification of Conjecture for the following cases

A_n, $n = 3, 4, 5, 6, 7$, D_n, $n = 4, 5, 6, 7$, E_6, E_7

Note that the cluster computation of W_{GHKK} is time consuming, because the division of Laurent polynomials in many variables is time consuming.

We use another approach. Namely, because of Theorem 1 in [5], we compute W_{GHKK} by applying the algorithm of [8] for computing the Berenstein-Kazhdan decoration function Φ_{BK}.

Theorem 1. *For simply-laced G, and a given reduced decomposition $\mathbf{i}$, the Newton polytopes Φ_{BK} and W_{GHKK} are isomorphic under a unimodular transformation.*

Corollary 1. *The Newton polytopes Φ_{BK} is void if and only if the Newton polytopes W_{GHKK} is void.*

Thus, for the numeric verification of Conjecture we compute the Newton polytope Φ_{BK} using the algorithm [8] and Polymake.

3. Algorithm description

The algorithm for computation Berenstein-Kazhdan decoration function Φ_{BK} is based on Theorem 4.4 in [8]. For the input data consisting of a group G and reduced word $w_0 = \mathbf{i}$. Let W be a corresponding Weyl group, I be the set of simple roots, Λ_i be i-th fundamental weight.

We compute Φ_{BK} as sum:

$$\Phi_{BK} = \sum_{j \in I} \Delta_{w_0\Lambda_j, s_j\Lambda_j},$$

where Δ is generalized minor function (see Defenition 2.2 in [8]). By Theorem 4.4. [8] it is possible to compute all monomials in $\Delta_{w_0\Lambda_i, s_i\Lambda_i}$ by

```
set initial algebraic structures:
                Weyl group,
                Cartan matrix,
                ring of Laurent polynomials
set initial value of word w

compute source monomial
     compute weyl_action(i_1, i_2, ...i_j)(α_j) for all j
     find index h(j) of one equal to exactly a_j
     set t_{h(j)} as starting monomial

start graph enumeration
set new variables
     St=[[source monomial]] //layers of associahedron
     Gs=[] //graph structure

until new_nodes=[1]
     //until we get only one new monomial equal to 1)
     mutate all monomials in last layer of St
     add new monomials to new layer of St
     add arrows to Gs for all mutations
          with marks corresponding to A_k^{-1} factors
return sum(flatten(St))
```

Listing 1: main loop of the algorithm

consequently applying multiplication by monomials A_k^{-1}, where

$$A_k = t_j t_{j^+} \prod_{j<l<j^+} t^{c_{i_l,i_j}}$$

and

$$k+ := min\{l \in [1,N] | w_l = w_k, l > k\} \cup \{\infty\}$$
$$k- := max\{l \in [1,N] | w_l = w_k, l < k\} \cup \{0\}$$

(3.7 [8]) starting from predefined source monomial. The algorithm performs graph enumeration for graph with vertices being monomials of $\Delta_{w_0\Lambda_j, s_j\Lambda_j}$ and edges are relations of monomials being different by multiplication by A_k^{-1}.

Computation of each $\Delta_{w_0\Lambda_i, s_i\Lambda_i}$ in pseudo-code using SageMath computer algebras system is shown in listing 1.

The mutation procedure in listing 2 checks which mutations are applicable to a given monomial due to Theorem 4.4 [8] and computes new monomials. To speed up computation we compute the b-vector b^0 of the source monomial $t_{h(i)}$ and then we apply Lemma 5.2 [8]. Computed b-vectors are stored for later use. For +-notation see (3.5) [8]. The pseudo-code for b-vectors implementation with reference to SageMath functions is shown in listing 3.

After $\Delta_{w_0\Lambda_i, s_i\Lambda_i}$ are computed for all simple weights all is left is sum them to obtain Φ_{BK}. Computation of it's Newton polytope using Polymake is straightforward. We check it's interior integer lattice points using Polymake method `N_INTERIOR_LATTICE_POINTS`.

For computation of monomials in W_{GHKK} it is necessary to compute each set of monomials W_a from Φ_{BK} computation data. Using

$$W_a = Y_1^{c_{1a}(t)} \cdots Y_N^{c_{Na}(t)} \prod_i F_i(t)(Y_1, \cdots, Y_N)^{b_{ia}(t)}.$$

```
def mutate_node(M)
     //M = ∏_{m=1}^{N} t_m^{d_m}
     compute b_j for node if it was not computed previously
     set initial result as empty list
     for all simple roots α_l
        for all k in R_l
           if k+ < ∞
              if d_k<2 and b_{k+}>0
                 new_monomial = M * A_k^{-1}
                 compute new_monomial b_j:
                    Lemma 5.2 [8]
                    new_monomial b_j = old_monomial b_j
                    new_monomial b_k +=1
                    new_monomial b_{k+} -=1
                 cache new_monomial b_j
                 add new_monomial and graph edge to result
              if d_k == d_{k+}
                 set lookup depth h=2
                 while k^{+h} < ∞ and d_{k^{+h}} = 0 and b_{k^{h+}} = 0
                    h++
                 if d_{k^{+h}} = -1 and b_{k^{+h}} = 1
                    new_monomial = M * A_k^{-1}
                    compute new_monomial b_i:
                       [Lemma 3.4]
                       new_monomial b_j = old_monomial b_j
                       new_monomial b_k +=1
                       new_monomial b_{k+} -=1
                    cache new_monomial b_j
                    add new_monomial and graph edge to result
     return result
```

Listing 2: mutation procedure

```
def b(M)
     //M = ∏_{m=1}^{N} t_m^{d_m}
     set b_N = d_N +
          fundamental_weights[j].weyl_action(α_j).
                         scalar(simple_coroots(i_N))
     for t from N-1 to 1
          b_t=d_t+
               fundamental_weights[j].weyl_action(α_j).
                         scalar(simple_coroots(i_t))
          for l from t to N
          b_t -= b[l]*a_{i_t,i_{l+1}}
     return b
```

Listing 3: b-vector computation

```
compute GHKK support
      b_start = get b_i vector of source monomial in Δ_{w0Λ_i,s_iΛ_i}
      b_stop = get b_i vector of stop monomial in Δ_{w0Λ_i,s_iΛ_i}
      ee = basis spanned by e_k − e_{k+}
      GHKK_support = coordinates of b_stop-b_start in ee

compute Y_frozen
      Y_frozen=Y[number last occurrence of j in i]

compute W_j
      compute start monomial
            Y_start = Y_frozen * ∏_{0<k<=len(i)} Y_k
      W = dictionary with keys in monomials in Δ_{w0Λ_i,s_iΛ_i}
      W[start monomial of Δ_{w0Λ_i,s_iΛ_i}] = Y_start
      enumerate edges in graph Gs starting from source monomial:
            v_b = start of the edge
            v_e = end of the edge
            k = mark on the edge
            W[v_e]=W[e_b]*Y[k]
      return set(values(W))
```

Listing 4: computation of W_{GHKK} monomials

we Because in A-cluster variables $t_m = \frac{X_m}{X_{m^-}}$ ([2] Corollary 4.10). Recall, that cluster variables are generalized minors computed at the twisted matrix corresponding to $\theta^-_{\mathbf{i}}(t_1,\cdots,t_N)$). Namely, at the vertex m

$$X_m := \Delta_{\Lambda_{i_m},s_N\cdots s_{m+1}\Lambda_{i_m}}(\psi^{w_0,e}(X)),$$

where $X = \theta^-_{\mathbf{i}}(t_1,\cdots,t_N)$, and $\psi^{w_0,e}(X)$ is the twist, $\psi^{w_0,e}(X) \in U^{e,w_0} = U \cap B_-\overline{w}_0B_-$. Because of that relation to A cluster variables X's, we get by direct computations, using p-map, the following relations to X-cluster variables, Y's

$$A_j = t_jt_{j^+}\prod_{j<l<j^+} t_l^{a_{i_l,i_j}} =$$

$$\frac{X_j}{X_{j^-}}\frac{X_{j^+}}{X_j}\prod_{j<l<j^+}(\frac{X_l}{X_{l^-}})^{a_{i_l,i_j}}$$

$$= \frac{\prod_{l'\to j}X_{l'}}{\prod_{j\to l''}X_{l''}} = Y_j.$$

Similarly, we get

$$t_Jt_{J+1}^{a_{i_{J+1},i}}\cdots t_N^{a_{i_N,i}} = Y_1^{c_{1a}(t)}\cdots Y_N^{c_{Na}(t)} = Y_{w_0i}.\ Y_a \text{ (exchanging factor } A_k^{-1} \text{ with } Y_k).$$

Note, that the monomial in Y's variables corresponding to t_k is the following

$$Y_{w_0i}\prod_{s\in S}Y_s,$$

where S is defined by the decomposition of b vectors

$$b(t_k) - b(t_Jt_{J+1}^{a_{i_{J+1},i}}\cdots t_N^{a_{i_N,i}}) = \sum_{s\in S}(e_s - s_{s^+}).$$

Because of above relation between t's coordinates and Y's coordinates and since the 3-braid moves correspond to cluster mutations, we apply the same line of arguments as in [5] and get that $\Delta_{w_0\Lambda_i,s_i\Lambda_i}\circ\theta^-_{\mathbf{i}}(t_1,\cdots,t_N)$ transformed in Y_m's coordinates will coincide with $Y_k(t)$. Therefore

$$\sum_i \Delta_{w_0\Lambda_i,s_i\Lambda_i}\circ\theta^-_{\mathbf{i}}(t_1,\cdots,t_N)$$

coincides with W'_{GHKK} wrt the change of variables t_m's into Y_m's.

Now it is possible to apply same sequence of mutations corresponding to edges in `Gs` graph starting from frozen

To determine starting monomial of F-polynomial part it is needed to compute GHKK support – coordinates of $b_{source} - b_{stop}$ vector in basis $e_k - e_{k+}$ ((3.5) [8]), where b_{source} is b-vector of source monomial and b_{stop} is b-vector of last computed monomial in $\Delta_{w_0\Lambda_i,s_i\Lambda_i}$ (listing 4).

Last step is to compute monomials in W_{GHKK} as $\bigcup_{j\in I} W_j$. Again, checking if Newton polytope of W_{GHKK} is void is done via Polymake using same routine.

The implementation using SageMath is provided in [13].

4. Algorithm complexity

We establish a bound on the complexity of the algorithm computing $\Delta_{w_0\Lambda_i,s_i\Lambda_i}$ and W_j with respect to the number of monomials in $\Delta_{w_0\Lambda_i,s_i\Lambda_i}$. Let K be the number of such monomials, and r be the rank of the Lie algebra.

In the mutation function, each node can produce at most $N = length(\mathbf{i})$ mutations, $N \sim O(r^2)$. Note that mutation branch with lookup in $\mathbf{i}$ need no more than r cycles and $k+/k-$ operations can also be computed in r operations if $\mathbf{i}$ splitting into slices of same root number is cached. With monomial multiplication complexity ($O(N)$), we get $O(r^4K)$.

The main tree search can be done in linear time using prefix trees for string representation of monomials in alphabet t_j is of complexity $O(r^2K)$, because monomials can have only N variables. Therefore total complexity of generating the monomials of $\Delta_{w_0\Lambda_i,s_i\Lambda_i}$ is of complexity

$$O(r^4K) \sim O(r^2 * \textbf{length of string representation}).$$

W_j computation is bounded by multiplication complexity and number of edges in Gs: $O(r^2) * O(K * r^2) \sim O(r^4K)$. For a fixed r, this complexity is the lowest possible complexity being linear with respect to actual complexity to print out the answer. Complexity of lattice point counting in polytopes has theoretical exponential upper bound with respect of number of inequalities defining polytope [11].

Actual computing speed is mostly determined by speed of Polymake operations. Average time of $\Delta_{w_0\Lambda_i,s_i\Lambda_i}$ and W_j computation for single simple root of D_6 with polytope checking is around 2 seconds, for E_7 – 8 seconds. For comparison, one computation for single simple root of D_6 without any Polymake operations takes around 70ms. For all computations we used a PC with dual 3.8 Ghz Intel® Xeon® Gold 5222 CPU running Ubuntu Linux.

Computing all half-potentials and W_{GHKK} for given Dynkin type relies on enumeration of all reduced decompositions of longest element in Weyl group. This is done using SageMath library with **BraidOrbit**[12] function. Original **BraidOrbit** function provides full list of reduced decompositions, which can be enormous and not feasible to compute in real applications (even for D_5). This can be resolved making it generator function, returning a new reduced decomposition each call. Second difficulty is related to algorithm computing all reduced decompositions. Function loops trough all braid moves and applies them to given reduced decomposition, then compares new decompositions to ones that were produced and stored before. This leads to extensive memory consumption when computing all half-potentials for given Lie group. As can be seen from algorithm, if two reduced decompositions can be transformed one into another using only 2-braid moves, then the results of computation differ only by variable exchange. If it is possible to enumerate all classes of reduced decompositions up to equivalence by 2-braid moves, it can drastically reduce number of cases for computation (cf. for

D_4 number of classes is 182 and total number of reduced words is 2316, for D_5 - 13198 and 12985968). (Number of classes of equivalence fot type D can be found in A180607 OEIS sequence.) Thus we propose an open problem to find fast algorithm to produce only relevant reduced decompositions which give different half-potentials.

Problem. *Find algorithm to enumerate quotient by 2-braid moves equivalence set of set of all reduced decompositions of longest element of Weyl group and produce one reduced decomposition from each class with respect of equivalence by far commuting braid moves. Complexity of one step of generation should be polynomial with respect of rank of Lie-group.*

References

[1] A.Berenstein, S.Fomin, and A.Zelevinsky, Cluster algebras.III. Upper bounds and double Bruhat cells. Duke Math. J. 126(1), 1–52 (2005)

[2] A. Berenstein and A. Zelevinsky, Tensor product multiplicities, canonical bases and totally positive varieties, Invent. Math. 143(1), 77–128 (2001)

[3] J. Fei. Combinatorics of F-polynomials Preprint. arXiv:1909.10151.

[4] S.Fomin and A. Zelevinsky, Cluster algebras.IV. Coefficients. Compos. Math.143,112–164(2007)

[5] V. Genz, G. Koshevoy, and B. Schumann, Polyhedral parametrizations of canonical bases & cluster duality, Advances in Mathematics 369 (2020), p. 107178

[6] M. Gross, P. Hacking, S.Keel, and M. Kontsevich, Canonical bases for cluster algebras. J. Am. Math. Soc. 31, 497–608 (2018)

[7] Y.Kanakubo, G.Koshevoy and T.Nakashima, An algorithm for Berenstein-Kazhdan decoration functions and trails for minuscule representations, J. of Algebra (2022)

[8] Y.Kanakubo, G.Koshevoy and T.Nakashima, An algorithm for Berenstein-Kazhdan decoration functions and trails for classical groups, arXiv:2207.08065

[9] Keller B., Cluster algebras and derived categories, arXiv:1202.4161

[10] G.Koshevoy and B.Schumann, Redundancy in string cone inequalities and multiplicities in potential functions on cluster varieties, J. of Algebraic Combinatorics (2022)

[11] J.Loerab, R.Hemmeckeb, J.Tauzera, R.Yoshida, Effective lattice point counting in rational convex polytopes, J. of Symbolic Computation, 38, 1273–1302 (2004)

[12] https://doc.sagemath.org/html/en/reference/combinat/sage/combinat/root_system/braid_orbit.html, https://github.com/sagemath/sage/blob/develop/src/sage/combinat/root_system/braid_orbit.pyx

[13] https://github.com/mironovd/crystal