Gröbner Fan, related ideas and an algorithm

José Abel Castellanos Joo

December 26, 2018

Abstract

In this technical report we discuss the Gröbner fan construction. We examine the connections between the Gröbner basis algorithm, monomial orderings, Euclidean geometry and linear programming used for the Gröbner fan construction. We also provide some examples and a detailed explanation of the algorithm by Mora and Robbiano [1].

Keywords— commutative algebra, computational algebraic geometry, linear programming

1 Introduction

The Gröbner fan of an ideal was introduced in [1]. This idea was motivated to study all possible reduced Gröbner basis for all monomial orders since the latter heavily depends on a particular monomial order [2] since the main component in many Gröbner basis algorithms relies on a division algorithm. A simple (and quite expected) observation notices that different monomial orders lead different Gröbner basis. Moreover, the performance of certain algorithms is faster using certain monomial orders. In addition, current complexity results [3] indicate instances where th runtime for ideals with polynomials of degree at most d is $\mathcal{O}(2^{2^d})$. Hence, it is interesting to obtain a characterization of how to transform one Gröbner basis to another with different monomial order due to the expensive nature of the algorithm. For the latter, some techniques which are based on the Gröbner fan, like Gröbner walk, provide a solution. More efficient methods might rely on the use of Universal Gröbner basis. However, the Gröbner fan has proven to be useful in other areas of mathematics like tropical algebra, auction design and optimization problems [4].

Among of the main outcomes of [1] are:

1. There is a one-to-one correspondence between reduced marked Gröbner bases with initial ideals.
2. The set of initial ideals is finite (hence the set of all reduced marked Gröbner bases is finite too).
3. For every initial ideals in $k[x_1, \ldots, x_n]$ there is a corresponding positive vector in $\mathbb{R}^n$.

Similarly, in [5], the authors motivated the above points through three questions. An important fact mentioned in both [5, 6], which received little attention, was about the encoding of monomials orders. I consider this crucial since the main focus of our study is to characterize Gröbner bases independently of the monomial order, hence it is necessary to give a formal representation of the latter. For instance in [5] this was achieved by representing monomial orders using matrices, whereas [6] considered a recursive definition using *vectors* and the standard *dot product* in $\mathbb{Z}^n$ assuming the existence of an arbitrary term order, which typically is the lexicographical order. In the end, both approaches are useful since the main property of these encodings is the relevance of the first row of the matrix (initial vector respectively) to define an implicit Gröbner basis, which is the so called *marked Gröbner basis*.

Given all these properties, the algorithm by Mora and Robianno [1] for computing Gröbner fans exploits the fact that the Gröbner basis are finite and so a naive algorithm of three steps will always terminate. Their approach can be studied as follows:

1. Given a set of polynomials S, enumerating all possible initial monomials and filter the ones that are feasible using linear programming techniques. The enumeration is a combinatorial process of choosing a initial monomial m for each polynomial f in S and making the correspondent inequalities with each non-initial monomial in f.

2. Extend the initial set of polynomials to several marked Gröbner bases (each for every possible monomial order previously computed) using a Gröbner basis algorithm. This step is motivated by a result in [5] which states that the Gröbner fan covers the positive orthant of $\mathbb{R}^n$. Hence, if this step cannot find more extensions it means the set of marked Gröbner basis has already covered this section of $\mathbb{R}^n$.

3. Filter the previous marked Gröbner bases by computing reduced Gröbner bases. This step give us additionally a complete geometrically characterization of the reduced marked Gröbner bases and it is important since many of the marked Gröbner bases computed in the previous step might be the equivalent with different monomial orders.

We will discuss several examples to illustrate the above algorithm in Section 4. In Section 2 we will discuss the relevance of the monomial orders, initial monomials, and its relevance with the geometric structure that entails the finiteness of the construction [1]. In Section 3 we will discuss an implementation of the Mora and Robbiano algorithm using Python will additional support of the computer algebra system *Sage* [8] and the linear programming library scipy [9].

2 The Gröbner Fan of an ideal: A mix of commutative algebra, combinatorics, and linear programming

For this discussion, we will choose to represent monomial orders as matrices with real entries. There are different conditions for particular entries in such matrices to define a monomial order [5].

In order to compare two terms it is just necessary to use their exponents. Let M be a matrix and α, β the exponents of two polynomials. A matrix order works as follows: $\alpha \prec_M \beta$ if there exists a row vector ω in M, say the ith row, such that $\alpha \cdot \omega < \beta \cdot \omega$ and all previous rows $\omega^{'}$ in M we have that $\alpha \cdot \omega^{'} = \beta \cdot \omega^{'}$.

Example 2.0.1. *Let us consider the following matrix* $M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. *This matrix encodes the* graded lexicographical order $x > y > z$ *for the polynomial ring* $k[x, y, z]$. *We can see that the first row essentially compares the total grade in the exponent of terms and the rest of the rows break tie using the lexicographic order.*

Example 2.0.2. *On the other hand, not all matrices define a monomial order. For instance, let us consider the matrix* $M^{'} = \begin{pmatrix} 1 & 2 & -1 & -2 \\ 2 & -1 & -2 & 1 \\ -1 & -2 & 1 & 2 \\ -2 & 1 & 2 & -1 \end{pmatrix}$. *We notice that multiplying* $M^{'}$ *with any vector with the same elements in each entry will give us a zero vector. Hence* $M^{'}$ *cannot distinguish these set of vectors, which is problematic according to the definition of a monomial order. In general it should be desirable that the kernel of these matrices intersecting the correspondent* $\mathbb{N}^n$ *is empty. The latter will provide injectivity to the matrix in order to distinguish different vectors. The latter property is common and shared among different monomial order in matrices with different domains (rational, real, etc).*

We will like to study Gröbner bases without *explicitly* specifying the monomial ordering. For the latter the idea of a *marked Gröbner basis* was introduced.

Definition 2.1. *[5] A* (reduced) marked Gröbner basis *is a set of pairs of polynomials and monomials* $\{(f_i, g_i) | i = 1, \ldots, r\}$ *such that* $\{f_i | i = 1, \ldots, r\}$ *is a (reduced) Gröbner basis and* $g_i = LT_{\prec}(f_i)$ *for some monomial order* $\prec$.

[1] It is worth mentioning the set of monomial ideals is finite for the commutative case. However, for the non commutative case this construction is not finite as noted by Weispfenning in [7]

The latter definition helps providing an implicit monomial ordering, i.e. the monomial order used to compute the Gröbner basis is not available. Is this information enough to uniquely determine the original monomial order used to compute the Gröbner basis? As shown in [5], having the information of the leading monomial we can produce a set of inequalities that, if satisfiable, the solution of such system of inequalities corresponds to the first row of matrix order.

Example 2.1.1. *We will highlight the leading monomial of a polynomial using parenthesis. Let us consider the marked Gröbner basis* $\{(xy)-z^2,(wyz)-x^3,(wy^2)-x^2z,(wz^3)-x^4\}$*. The set of inequalities entailed by the marked Gröbner basis is:*

- $(0,1,1,0)\cdot(a,b,c,d) \geq (0,0,0,2)\cdot(a,b,c,d)$
- $(1,1,1,0)\cdot(a,b,c,d) \geq (0,3,0,0)\cdot(a,b,c,d)$
- $(1,0,2,0)\cdot(a,b,c,d) \geq (0,2,0,1)\cdot(a,b,c,d)$
- $(1,0,0,3)\cdot(a,b,c,d) \geq (0,4,0,0)\cdot(a,b,c,d)$

Using a linear solver we compute a solution for the above system of inequalities:

```
>>> linprog(np.array([1, 1, 1, 1]), A_ub = np.array([[0, -1, -1, 2], [-1, 2, -1, 0], [-1, 2, -2, 1], [-1, 4, 0, -3]]), b_ub = np.array([0, 0, 0, 0]), bounds = (0.01, None))
     fun: 0.040000000000000008
 message: 'Optimization terminated successfully.'
     nit: 8
   slack: array([  0.00000000e+00,   0.00000000e+00,   2.31296463e-18,
         2.31296463e-18,   0.00000000e+00,   2.89120579e-19,
         5.78241159e-19,   0.00000000e+00])
  status: 0
 success: True
       x: array([ 0.01,  0.01,  0.01,  0.01])
```

Figure 1: The solution found by scipy.optimize.linprog [9] is $x = [0.01, 0.01, 0.01, 0.01]$

We notice then that the first row for the previous matrix order are all the same elements. Thus we can suspect an equi-graded order (i.e. an order which doesn't prefer an indeterminate while computing the total grade) is used. This phenomenon happens quite frequently, specifically with equi-graded orders. For example, consider the graded lexicographical order $x > y > z$ and the graded lexicographical order $y > z > x$. The first row of their matrix order will be the same, but the rest of their matrices will be different.

From the previous observation, if we take all the vectors in the $\mathbb{R}^n$ such that they satisfy the set of inequalities demanded by a matrix order we will obtain a *cone* in $\mathbb{R}^n$. In [5] it is shown that the geometry properties of these cones form a fan, which is a structure such that the intersection of cones (known as faces) belong to the structure as well as each of the faces of the cones belong to the structures. In hindsight, the faces of this structure correspond to elements of the cones such that the the dot product cannot distinguish between exponent vectors, hence it is important to rely on the interior points of these cones to fully distinguish/classify exponent vectors.

3 Mora and Robianno Algorithm: Discussion and Implementation in Sage

Here is an implementation [2] of algorithm by Mora and Robbiano for computing Gröbner fan of an ideal:

```
# 'inputBasis' is an array of polynomials
def groebnerFan(inputBasis):

    # Initialization
    L = ([], [], [], {}, [])
    Lnew = [L]
    for polynomial in inputBasis:
        Lold = Lnew
        Lnew = []
```

[2]Comments in Python begin with #

```
        for (G, M, E, Psi, B) in Lold:
            for leadingMonomial in polynomial.monomials():
                Gnew = G[:]
                Gnew.append(polynomial)
                Mnew = M[:]
                Mnew.append(leadingMonomial)
                Enew = E[:]
                for nonLeadingMonomial in polynomial.monomials():
                    if(nonLeadingMonomial != leadingMonomial):
                        # We substract the Leading Monomial to the
                        # Non Leading Monomials because the LP solver
                        # has <= as default inequalities
                        Enew.append(subtractExponents(nonLeadingMonomial,
                                                      leadingMonomial))
                Psinew = copy.deepcopy(Psi)
                Psinew[polynomial] = leadingMonomial
                Bnew = B[:]
                for g in G:
                    Bnew.append((g, polynomial))
                if isNotEmptyTO(Enew):
                    L = (Gnew, Mnew, Enew, Psinew, Bnew)
                    Lnew.append(L)

    # Computation of the Groebner Bases
    Lwork = Lnew
    Lpartial = []
    while (Lwork != []):
        G, M, E, Psi, B = Lwork.pop()
        f, g = B.pop()
        T = lcm(Psi[f], Psi[g])
        gCoeffPsiG = g.monomial_coefficient(Psi[g])
        fCoeffPsiF = f.monomial_coefficient(Psi[f])
        h = gCoeffPsiG*T*f//Psi[f] - fCoeffPsiF*T*g//Psi[g]
        check, subtract = minimalPolynomialCheck(G, Psi, h)
        while check:
            h = h - subtract
            check, subtract = minimalPolynomialCheck(G, Psi, h)
        if h == 0 :
            if (B == []):
                Lpartial.append((G, M, E, Psi))
            else:
                Lwork.append((G, M , E, Psi, B))
        else:
            for leadingMonomial in h.monomials():
                Gnew = G[:]
                Gnew.append(h)
                Mnew = M[:]
                Mnew.append(leadingMonomial)
                Enew = E[:]
                for nonLeadingMonomial in h.monomials():
                    if(nonLeadingMonomial != leadingMonomial):
                        # We substract the Leading Monomial to the
                        # Non Leading Monomials because the LP solver
                        # has <= as default inequalities
                        Enew.append(subtractExponents(nonLeadingMonomial,
                                                      leadingMonomial))
                Psinew = copy.deepcopy(Psi)
                Psinew[h] = leadingMonomial
                Bnew = B[:]
                for g in G:
```

```
70                      Bnew.append((g, h))
71                  if isNotEmptyTO(Enew):
72                      Lwork.append((Gnew, Mnew, Enew, Psinew, Bnew))
73
74      # Computation of the Reduced Groebner Bases
75      # and of the Groebner Region
76      Loutput = []
77      Mon = []
78      while (Lpartial != []):
79          G, M, E, Psi = Lpartial.pop()
80          if (not membershipIdealArrayTest(M, Mon)):
81              polynomial, check = reducibilityCheck(G, M, Psi)
82              while check:
83                  G.remove(polynomial)
84                  M.remove(Psi[polynomial])
85                  del Psi[polynomial]
86                  polynomial, check = reducibilityCheck(G, M, Psi)
87              for g in G:
88                  G.remove(g)
89                  M.remove(Psi[g])
90                  gnew = red(G, M, g)
91                  coeffGNew = gnew.monomial_coefficient(Psi[g])
92                  gnew = 1/coeffGNew*gnew
93                  G.append(gnew)
94                  M.append(Psi[g])
95                  tempPsiG = Psi[g]
96                  del Psi[g]
97                  Psi[gnew] = tempPsiG
98              E = []
99              for g in G:
100                 for monomial in g.monomials():
101                     if(monomial != Psi[g]):
102                         E.append(subtractExponents(Psi[g],
103                                                    monomial))
104             Loutput.append((G, M, E, Psi))
105             Mon.append(M)
106     return Loutput
```

As mentioned before, the algorithm has three main components. The implementation is straightforward once is understood the high level idea. Many additional methods where needed to implement separately in order to provide a clean design.

It might be worth mentioning the main data structure used in the algorithm. In many parts of the algorithm we will see the common decomposition of an array of elements into the tuple (G, M, E, Psi, B). These components stand for the following:

- G is the set of polynomials for a Gröbner basis.
- M is the set of monomials keeping track of the leading monomials for the respective G.
- E is the set of inequalities produced by enumerating the constraints by G and M.
- Psi is a map (dictionary structure in Python) that associates an element in G with an element in M.
- B keeps track of the pair of elements in G that have not reduced using Buchberger's criterion in order to avoid unnecessary computations.

In order to test if a set of inequalities define a monomial order (lines 29, 71) we used the linear programming library 'scipy.optimize.linprog' to find solution to the set of inequalities. We realize that the library uses non-strict inequalities and the lack of attention of the latter produced bugs in the program since the linear solver was accepting set of inequalities that do not define monomial orders. The latter was fixed by including a small epsilon value to the non-strict inequalities to convert them into strict inequalities.

Additionally, we used a theorem in [2], to be more precise a corollary about two monomial ideals being equivalent, to code line 80 in the previous implementation.

4 Some examples

First we will compute some examples using knowledge previously discussed about the Gröbner fan.

Example 4.0.1. *Let us consider the polynomial ring* $\mathbb{Q}[x, y, z]$ *and the ideal* $\{yz + x, xy + z, x^2 - z^2\}$. *We can start the computation of the marked Gröbner basis by taking the graded lexicographical order for all the possible variable orderings. We obtain:*

- *Graded reverse lexicographic order* $x > y > z$: $\{(yz) + x, (xy) + z, (x^2) - z^2\}$
- *Graded reverse lexicographic order* $x > z > y$: $\{(yz) + x, (xy) + z, (x^2) - z^2\}$
- *Graded reverse lexicographic order* $y > x > z$: $\{(yz) + x, (x^2) - z^2, (xy) + z\}$
- *Graded reverse lexicographic order* $y > z > x$: $\{(xy) + z, (z^2) - x^2, (yz) + x\}$
- *Graded reverse lexicographic order* $z > x > y$: $\{(xy) + z, (yz) + x, (z^2) - x^2\}$
- *Graded reverse lexicographic order* $z > y > x$: $\{(xy) + z, (yz) + x, (z^2) - x^2\}$

Essentially there are only two different marked Gröbner bases. Are these all the marked Gröbner bases? We can check this in two ways, using a geometric approach or using the algorithm above. The geometric approach it is indeed inefficient since it requires to check if the set of cones cover the positive orthant of $\mathbb{R}^n$. *If the entire positive orthant of* $\mathbb{R}^n$ *is covered then we stop, otherwise we choose a vector in the complement area of the cones space generated so far.*

We will use the geometric approach. For the latter, we notice the current set of cones produced by the marked Gröbner bases is shown in Figure 2 (Left):

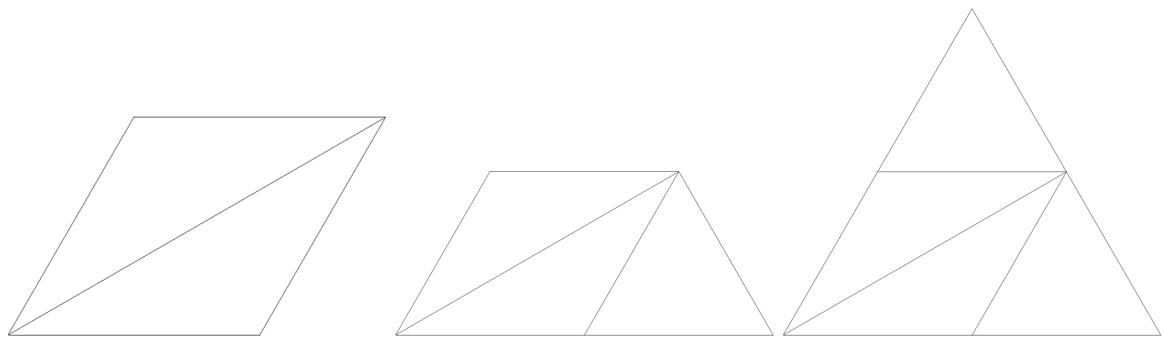

Figure 2: Cones of the progressive computation for the Gröbner fan of the ideal $\{yz + x, xy + z, x^2 - z^2\}$

Using the gfanInterface *[10] from Macaulay2 [11] we observe the render produced intersect the cone with the hyper-plane* $x + y + z = 1$ *in order to produce a 2D plot. From this fact we know the set of cones cover the positive orthant of* $\mathbb{R}^3$ *when the render forms a triangle. Hence, we can choose a vector* $(w = (1, 0, 0))$ *from the bottom right side. From the latter we obtain the marked Gröbner basis (the correspondent set of cones is in Figure 2 (Center))* $\{(y^2 * z) - z, (x) + y * z\}$. *Now we try to extend the cones with the vector* $w = (0, 0, 1)$ *and we obtain the marked Gröbner basis* $\{(x * y^2) - x, (z) + x * y\}$. *The final set of cones is shown in Figure 2 (Right). Since the latter intersects with the triangle mention before we are sure we have computed all the marked Gröbner bases for the ideal* $\{yz + x, xy + z, x^2 - z^2\}$.

Example 4.0.2. *Let us consider the polynomial ring* $\mathbb{Q}[x, y, z]$ *and the ideal* $\{x*y-x, x^2+x*z, y^2*z+x\}$. *For this example we will use our implementation of the Mora and Robbiano algorithm.*

- *First we compute all the possible monomial orders. Enumerating all the possibilities these are the potential candidates:*
  - $\{(x * y) - x, (x^2) + x * z, (y^2 * z) + x\}$
  - $\{(x * y) - x, (x^2) + x * z, y^2 * z + (x)\}$
  - $\{(x * y) - x, x^2 + (x * z), (y^2 * z) + x\}$
  - $\{(x * y) - x, x^2 + (x * z), y^2 * z + (x)\}$
  - $\{x * y - (x), (x^2) + x * z, (y^2 * z) + x\}$
  - $\{x * y - (x), (x^2) + x * z, y^2 * z + (x)\}$
  - $\{x * y - (x), x^2 + (x * z), (y^2 * z) + x\}$
  - $\{x * y - (x), x^2 + (x * z), y^2 * z + (x)\}$

  However, not all the previous candidates will define monomial orders. Using a linear programming solver we obtain the following set of polynomials that define a monomial order:
  - $\{(x * y) - x, (x^2) + x * z, (y^2 * z) + x\}$
  - $\{(x * y) - x, (x^2) + x * z, y^2 * z + (x)\}$
  - $\{(x * y) - x, x^2 + (x * z), (y^2 * z) + x\}$
- *Now we extend each set of polynomials to marked Gröbner bases. We obtain:*
  - $\{(x * y) - x, x^2 + (x * z), (y^2 * z) + x\}$
  - $\{(x * y) - x, (x^2) + x * z, y^2 * z + x, (-y^3 * z) + y^2 * z\}$
  - $\{(x * y) - x, (x^2) + x * z, (y^2 * z) + x\}$
- *Finally, we obtain the reduced marked Gröbner basis, this step help to eliminate marked Gröbner bases than define the same monomial order. In this case, we kept the same number of reduced marked Gröbner basis but we eliminated redundant polynomials:*
  - $\{(x^2) + x * z, (x * y) - x, (y^2 * z) + x\}$
  - $\{(-y^3 * z) + y^2 * z, y^2 * z + (x)\}$
  - $\{x^2 + (x * z), (x * y) - x, (y^2 * z) + x\}$

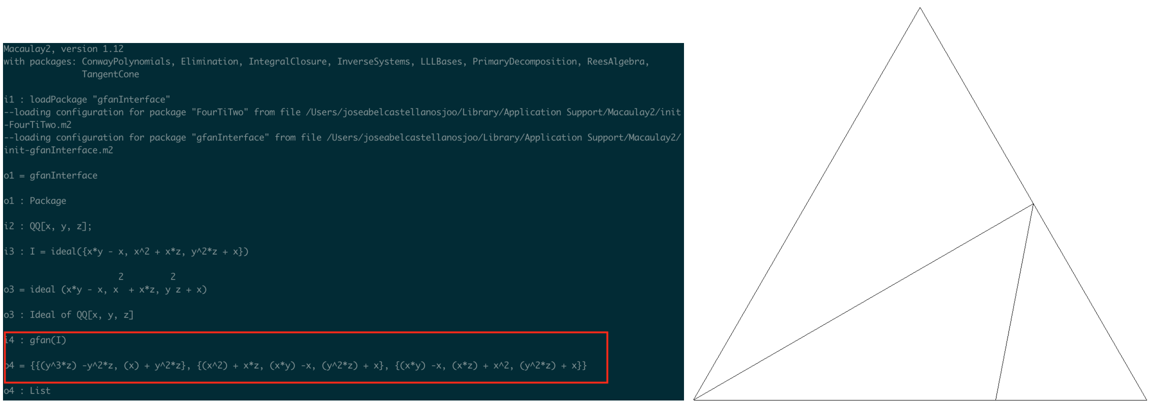

Figure 3: (Left) Using gfan to compute the Gröbner fan of the ideal $\{x*y-x, x^2+x*z, y^2*z+x\}$; (Right) Visualization of the Gröbner fan of the ideal $\{x * y - x, x^2 + x * z, y^2 * z + x\}$

Example 4.0.3. *Let us consider the polynomial ring* $\mathbb{Q}[x, y, z]$ *and the ideal* $\{x^2 - y, y^2 - x * z - y * z\}$*. For this example we will use our implementation of the Mora and Robbiano algorithm. We will repeat the same steps as in the previous example highlighting the output produced at each step:*

- *Valid monomial orders:*
  - $\{(x^2) - y, (y^2) - x * z - y * z\}$
  - $\{(x^2) - y, y^2 - (x * z) - y * z\}$
  - $\{(x^2) - y, y^2 - x * z - (y * z)\}$
  - $\{x^2 - (y), (y^2) - x * z - y * z\}$
  - $\{x^2 - (y), y^2 - x * z - (y * z)\}$
- *Marked Gröbner basis:*
  - $\{x^2 - (y), y^2 - x * z - (y * z), x^4 - (x^2 * z) - x * z\}$
  - $\{x^2 - (y), (y^2) - x * z - y * z, x^4 - (x^2 * z) - x * z\}$
  - $\{x^2 - (y), (y^2) - x * z - y * z, (x^4) - x^2 * z - x * z\}$
  - $\{(x^2) - y, y^2 - x * z - (y * z)\}$
  - $\{(x^2) - y, y^2 - (x * z) - y * z, -x * y^2 + y^3 - (y^2 * z) + y * z\}$
  - $\{(x^2) - y, y^2 - (x * z) - y * z, (-x * y^2) + y^3 - y^2 * z + y * z, -y^4 + 2 * y^3 * z - (y^2 * z^2) + y * z^2\}$
  - $\{(x^2) - y, y^2 - (x * z) - y * z, (-x * y^2) + y^3 - y^2 * z + y * z, (-y^4) + 2 * y^3 * z - y^2 * z^2 + y * z^2\}$
  - $\{(x^2) - y, (y^2) - x * z - y * z\}$
- *Reduced marked Gröbner basis:*
  - $\{(y^2) - x * z - y * z, (x^2) - y\}$
  - $\{y^2 - (x * z) - y * z, (-y^4) + 2 * y^3 * z - y^2 * z^2 + y * z^2, (x * y^2) - y^3 + y^2 * z - y * z, (x^2) - y\}$
  - $\{y^2 - (x * z) - y * z, -y^4 + 2 * y^3 * z - (y^2 * z^2) + y * z^2, (x * y^2) - y^3 + y^2 * z - y * z, (x^2) - y\}$
  - $\{y^2 - (x * z) - y * z, (x^2) - y, x * y^2 - y^3 + (y^2 * z) - y * z\}$
  - $\{y^2 - x * z - (y * z), (x^2) - y\}$
  - $\{(x^4) - x^2 * z - x * z, -x^2 + (y)\}$
  - $\{x^4 - (x^2 * z) - x * z, -x^2 + (y)\}$

Figure 4: (Left) Using gfan to compute the Gröbner fan of the ideal $\{x^2 - y, y^2 - x * z - y * z\}$; (Right) Visualization of the Gröbner fan of the ideal $\{x^2 - y, y^2 - x * z - y * z\}$

5 Conclusions

In this technical report we discussed the geometric structure of the Gröbner fan, the motivation for this research topic, and an algorithm for computing the latter. In few words, the Gröbner fan of an ideal is the collection of the cones associated with each of the reduced marked Gröbner basis for all possible monomial orders. With this project I had the opportunity to learn about the connections between Gröbner bases and other branches of mathematics, its importance, and several tools that are useful for this discipline.

For future work I would like come up with heuristic to enhance performance of the current algorithms available. As a first step, I consider important to study a recent technique that exploits some symmetries in the geometric structure of the Gröbner fan [12].

I also learned many intricate details about the Mora and Robbiano algorithm by implementing their algorithm and doing examples. The latter helped me to consolidate and fix a wrong picture I used to have about the method.

References

[1] Teo Mora and Lorenzo Robbiano. The gröbner fan of an ideal. *Journal of Symbolic Computation*, 6(2):183 – 208, 1988.

[2] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Springer Publishing Company, Incorporated, 4th edition, 2015.

[3] Ernst W Mayr and Albert R Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305 – 329, 1982.

[4] Nikolai Krivulin. Tropical optimization problems. *arXiv e-prints*, page arXiv:1408.0313, August 2014.

[5] David A. Cox, John Little, and Donal O'Shea. *Using Algebraic Geometry.* Springer Publishing Company, Incorporated, 2th edition, 2004.

[6] Bernd Sturmfels. Grobner bases and convex polytopes. *SERBIULA (sistema Librum 2.0)*, 12 1995.

[7] Volker Weispfenning. Constructing universal gröbner bases. In Llorenç Huguet and Alain Poli, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 408–417, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[8] Generic multivariate polynomials. `http://doc.sagemath.org/html/en/reference/polynomial_rings/sage/rings/polynomial/multi_polynomial_element.html`. Accessed: 2018-12-23.

[9] scipy.optimize.linprog. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html`. Accessed: 2018-12-23.

[10] Anders N. Jensen. Gfan, a software system for Gröbner fans and tropical varieties. Available at `http://home.imf.au.dk/jensen/software/gfan/gfan.html`.

[11] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at `https://faculty.math.illinois.edu/Macaulay2/`.

[12] Komei Fukuda, Anders N. Jensen, and Rekha R. Thomas. Computing Groebner fans. *Mathematics of Computation*, 76:2189–2212, December 2007.