# Computing Pretropisms for the Cyclic $n$-Roots Problem*

Jeff Sommars† Jan Verschelde‡

**Abstract**

The cyclic $n$-roots problem is an important benchmark problem for polynomial system solvers. We consider the pruning of cone intersections for a polyhedral method to compute series for the solution curves.

## 1 Introduction

The cyclic $n$-roots problem asks for the solutions of a polynomial system, commonly formulated as

$$\begin{cases} x_0 + x_1 + \cdots + x_{n-1} = 0 \\ i = 2, 4, \ldots, n-1 : \sum_{j=0}^{n-1} \prod_{k=j}^{j+i-1} x_{k \bmod n} = 0 \\ x_0 x_1 x_2 \cdots x_{n-1} - 1 = 0. \end{cases} \quad (1)$$

This problem is important in the study of biunimodular vectors, a notion that traces back to Gauss, as stated in [10]. In [3], Backelin showed that if $n$ has a divisor that is a square, i.e. if $d^2$ divides $n$ for $d \geq 2$, then there are infinitely many cyclic $n$-roots. The conjecture of Björck and Saffari [5], [10, Conjecture 1.1] is that if $n$ is not divisible by a square, then the set of cyclic $n$-roots is finite.

As shown in [1], the result of Backelin can be recovered by polyhedral methods. Polyhedral methods to solve a polynomial system consider the Newton polytopes of the polynomials in the system. The *Newton polytope* of a polynomial in several variables is the convex hull of the exponent tuples of the monomials that appear with nonzero coefficient in the polynomial. Looking for positive dimensional solution sets, we look for series developments of the solutions, and in particular we look for Puiseux series. The leading exponents of Puiseux series are called *tropisms*. A *pretropism* is a vector that forms the minimal inner product with a face of every one of the given polytopes, where none of the faces are 0-faces. Pretropisms are candidates for being tropisms, but not every pretropism is a tropism, as pretropisms depend only on the Newton polytopes of the system, see e.g. [13] for a mathematical background on tropical algebraic geometry.

*This material is based upon work supported by the National Science Foundation under Grant No. 1440534.

†Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, sommars1@uic.edu

‡Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, janv@uic.edu.

Our problem can thus be stated as follows. Given a tuple of Newton polytopes, compute all pretropisms. In [15] we examined the case where all polytopes are in general position with respect to each other. In this paper we focus on the Newton polytopes of the cyclic $n$-roots problem.

**Prior and related work.** In [6], the computation of pretropisms is defined as the common refinement of the normal fans of the Newton polytopes [18]. The software Gfan [12] relies on cddlib [11] in its application of reverse search algorithms [2].

## 2 Pruning Cone Intersections

To introduce our algorithms, consider Figure 1. For three Newton polytopes $(P_1, P_2, P_3)$, the leaves of the trees represent cones of pretropisms. Nodes without children that are not leaves correspond to cone intersections that contain only the zero dimensional cone.

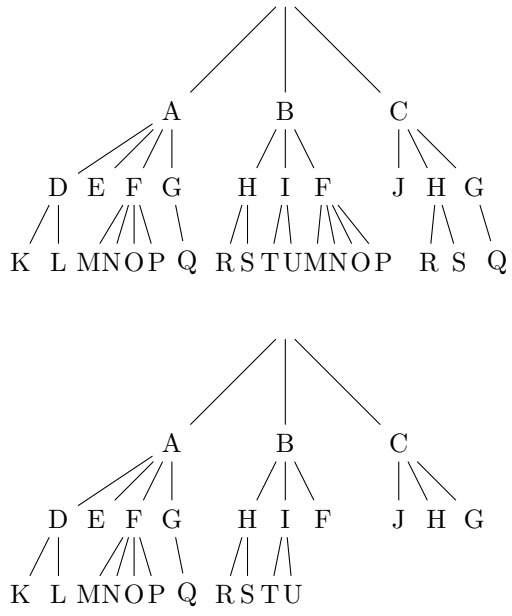


Figure 1: Nodes A, B, C represent cones to $P_1$. Intersections of those cones with the cones of $P_2$ are represented by nodes D through J. Duplicate nodes are removed from the second tree.

The removal of duplicate nodes eliminates many cone intersections at deeper levels in the tree.

## 3 Algorithms

Our algorithm takes as input the edge skeletons of a set of Newton polytopes; the edge skeleton of a polytope can be computed by a polynomial-time algorithm, presented in [9]. In our implementation, we use edge objects that have vertices, references to their neighboring edges, and the cone of the set of inner normals of all of the facets on which the edge rests. Note that since the cyclic-$n$ polytopes are not all full dimensional, we included generating rays of the lineality spaces as needed.

Algorithm 2 sketches the outline of the algorithm to compute all pretropisms of a tuple of $n$ polytopes. Along the lines of the gift wrapping algorithm, for every edge of the first polytope we take the plane that contains this edge and consider where this plane touches the second polytope. Algorithm 1 starts exploring the edge skeleton defined by the edges connected to the vertices in this touching plane.

**Algorithm 1** Explores the skeleton of edges to find pretropisms of a polytope $P$ and a cone $C$.

```
1:  function ExploreEdgeSkeleton(P, C)
2:      r := a random ray inside C
3:      m := min{⟨a, r⟩, a ∈ P}
4:      in_r(P) := {a ∈ P, ⟨a, r⟩ = m}
5:      EdgesToTest := edges e of P: e ∩ in_r(P) ≠ ∅
6:      Cones := ∅
7:      TestedEdges := ∅
8:      while EdgesToTest ≠ ∅ do
9:          E := pop an edge from EdgesToTest
10:         C_E := normal cone to E
11:         ShouldAddCone := False
12:         if C_E contains C then
13:             ConeToAdd := C
14:             ShouldAddCone := True
15:         else if C ∩ C_E ≠ {0} then
16:             ConeToAdd := C ∩ C_E
17:             ShouldAddCone := True
18:         end if
19:         if ShouldAddCone then
20:             Cones := Cones ∪ ConeToAdd
21:             Edges := Edges ∪ E
22:             for each neighboring edge e of E do
23:                 if e ∉ TestedEdges then
24:                     EdgesToTest := EdgesToTest∪e
25:                 end if
26:             end for
27:         end if
28:         TestedEdges := TestedEdges ∪ E
29:     end while
30:     return Cones
31: end function
```

The exploration of the neighboring edges corresponds to tilting the ray $r$ in Algorithm 1, as in rotating a hyperplane in the gift wrapping method. One may wonder why the exploration of the edge skeleton in Algorithm 1 needs to continue after the statement on line 5. This is because the cone $C$ has the potential to intersect many cones in $P$, particularly if $P$ has small cones. Furthermore it is reasonable to wonder why we bother checking cone containment when computing the intersection of two cones provides more useful information. Checking cone containment means checking if each of the generators of $C$ is contained in $C_E$, which is a far less computationally expensive operation than computing the intersection of two cones.

In the Newton-Puiseux algorithm to compute series expansions, we are interested only in the edges on the lower hull of the Newton polytope, i.e. those edges that have an upward pointing inner normal [17]. For Puiseux for space curves, the expansions are normalized so that the first exponent in the tropism is positive. Algorithm 2 is then adjusted so that calls to the edge skeleton computation of Algorithm 1 are made with rays that have a first component that is positive.

**Algorithm 2** Finds pretropisms for a given tuple of polytopes $(P_1, P_2, \ldots, P_n)$.

```
1:  function FindPretropisms(P_1, P_2, ..., P_n)
2:      Cones := set of normal cones to edges in P_1
3:      for i := 2 to n do
4:          NewCones := ∅
5:          for Cone in Cones do
6:              NewCones := NewCones ∪
                    ExploreEdgeSkeleton(P_i, Cone)
7:          end for
8:          Cones := NewCones
9:      end for
10:     Pretropisms := set of generating rays for each
          cone in Cones
11:     return Pretropisms
12: end function
```

### 3.1 Correctness

To see that this algorithm will do what it claims, we must define an additional term. A *pretropism graph* is the set of edges for a polytope that have normal cones intersecting a given cone. We will now justify why the cones output by Algorithm 1 correspond to the set of cones that live on a pretropism graph.

**Theorem 1** *Pretropism graphs are connected graphs.*

*Proof.* Let $C$ be a cone, and let $P$ be a polytope with edges $e_1, e_2$ such that they are in the pretropism graph of $C$. Let $C_1$ be the cone of the intersection

of the normal cone of $e_1$ with $C$, and let $C_2$ be the cone of the intersection of the normal cone of $e_2$ with $C$. If we can show that there exists a path between $e_1$ and $e_2$ that remains in the pretropism graph, then the result will follow.

Let $n_1$ be a normal to $e_1$ that is also in $C_1$ and let $n_2$ be a normal to $e_2$ that is also in $C_2$. Set $n = tn_1 + (1-t)n_2$ where $0 \leq t \leq 1$. Consider varying $t$ from 0 to 1; this creates the cone $C_n$, a cone which must lie within $C$, as both $n_1$ and $n_2$ lie in that cone. As $n$ moves from 0 to 1, it will progressively intersect new faces of $P$ that have all of their edges in the pretropism graph. Eventually, this process terminates when we reach $e_2$, and we have constructed a path from $e_1$ to $e_2$. Since a path always exists, we can conclude that pretropism graphs are connected graphs. ■

Since pretropism graphs are connected, Algorithm 1 will find all cones of edges on the pretropism graph. In Algorithm 2, we repeatedly explore the edge skeleton of polytope $P_i$, and use the pruned set of cones to explore $P_{i+1}$. From this, it is clear that Algorithm 2 will suffice to find all pretropisms.

## 4 Comparison With Our Previous Algorithm

Our algorithm in [15] restricted the pruning of the cone intersections in a vertical fashion: nodes in the tree with cone intersections that yield only $\{0\}$ will not have any children. That algorithm works well for polytopes with randomly generated coordinates.

In this paper we consider polytopes that are not in generic position. In this situation, intersecting normal cones to edges may lead to cones of normals of higher dimensional cones. At the same level in the tree we can then have duplicate cones or cones that are contained in other cones. In those cases were one cone is contained in another, the smaller cone can be pruned from the tree. We call this type of pruning horizontal pruning. For generic polytopes horizontal pruning would not reduce the number of cone intersections. However, in special cases like the cyclic $n$-root problem, there is the potential to dramatically reduce the scope of the problem through horizontal pruning.

To illustrate horizontal pruning, consider Figure 1. These graphs illustrate computing the pretropisms for three fictitious, non-generic polytopes $P_1$, $P_2$, $P_3$ with the two distinct algorithms. Nodes A, B, C represent the cones of the edges of $P_1$, the row below that represents the resulting cones from performing Algorithm 1 with $P_2$ and A, B, or C. The row below that varies in the two figures. In the tree at the top of Figure 1, the process iterates and Algorithm 1 is performed with $P_3$ and each of the input cones D through J. The tree at the bottom of Figure 1 shows how the horizontal pruning has the potential to improve over the previous algorithm. Since there are duplicate nodes for F, G, and H, each of these paths only needs to be followed once. Though this does not lead to dramatic improvements in this fictitious case, as the number of polytopes increases, the benefit of pruning compounds.

## 5 Computational Experiments

We developed a preliminary version of Algorithm 2 in Sage [16], using its modules for lattice polytopes [14], and polyhedral cones [7]; Sage uses PPL [4] to compute cone intersections. Our preliminary code is available at https://github.com/sommars/GiftWrap. We ran the code on a Red Hat Enterprise Linux workstation of Microway, with Intel Xeon E5-2670 processors at 2.6 GHz.

Instead of directly calculating the pretropisms of the Newton polytopes of the cyclic $n$-root problem, we chose to calculate pretropisms of the reduced cyclic $n$-root problem. This reformulation [8] is obtained by performing the substitution $x_i = \frac{y_i}{y_0}$ for $i = 0 \ldots n-1$. Clearing the denominator of each equation leaves the first $n-1$ equations as polynomials in $y_1, \ldots y_{n-1}$. We compute pretropisms of the Newton polytopes of these $n-1$ equations because they yield meaningful sets of pretropisms. Calculating with the reduced cyclic $n$-roots problem has the benefit of removing much of the symmetry present in the standard cyclic $n$-roots problem, as well as decreasing the ambient dimension by one. Unlike the standard cyclic $n$-roots problem, some of the polytopes of the reduced cyclic $n$-roots problem are full dimensional, which leads to calculation speed ups. A simple transformation can be performed on the pretropisms we calculate of reduced cyclic $n$-root problem to convert them to the pretropisms of cyclic $n$-root problem, so calculating the pretropisms of reduced cyclic $n$-roots problem is equivalent to calculating the pretropisms of the cyclic $n$-roots problem.

In Table 1, we have recorded the number of cone intersections performed and the number of times cone containments let us avoid performing additional intersections for each of the reduced cyclic $n$-root systems with $n \leq 10$. Table 1 also contains a comparison between the two sums, which acts as a way of evaluating the quality of the algorithms. We consider the unit of work of each algorithm to be the total number of intersections performed, as that is the constraining part of the algorithm. As $n$ increases, the ratio tends to increase as well, demonstrating that Algorithm 2 represents a substantial improvement over our previous algorithm. We expect that the result would become even more dramatic with higher $n$, but in our testing, our previous algorithm was too inefficient to terminate for $n > 10$.

| $n$ | intersections | containments | sum | intersections | containments | sum | ratio |
|---|---|---|---|---|---|---|---|
| 4 | 63 | 2 | 65 | 54 | 2 | 56 | 1.16071 |
| 5 | 750 | 20 | 770 | 395 | 5 | 400 | 1.92500 |
| 6 | 4,531 | 1,232 | 5,763 | 2,982 | 291 | 3,273 | 1.76076 |
| 7 | 105,982 | 5,767 | 111,749 | 18,798 | 343 | 19,141 | 5.83820 |
| 8 | 479,640 | 181,507 | 661,147 | 145,125 | 3,922 | 149,047 | 4.43582 |
| 9 | 9,232,384 | 1,993,049 | 11,225,433 | 1,101,563 | 16,313 | 1,117,876 | 10.04175 |
| 10 | 70,026,302 | 23,838,851 | 93,865,153 | 8,846,353 | 165,203 | 9,011,556 | 10.41608 |

Table 1: Columns two through four contain results when our previous algorithm is applied to the reduced cyclic $n$-roots problem while columns five through seven contain the results of Algorithm 2. The final column represents the ratio of the previous sum to the sum of Algorithm 2.

## 6 Comparison with Gfan

As we reported in [15], on randomly generated polytopes, our code was competitive with Gfan [12]. Although the additional pruning criteria presented in this paper are promising, on the specific cyclic $n$-roots problem, our Python prototype is not as good as the compiled code of Gfan. Our code tends to be slower by a factor of two, but we hope to be more competitive if we improve our ability to exploit the symmetry of the polytopes.

The computational complexity is such that high level parallelism is effective. Instead of iterating through all of the cones from line 5 of Algorithm 2, we can create a queue of them and then perform Algorithm 1. We then initialize some number of processes and give them successive cones from the queue until the queue is empty. Once the queue is empty, the resulting cones are pruned and combined and the algorithm iterates. We have incorporated this parallelism into our prototype Sage code.

## References


[1] D. Adrovic and J. Verschelde. Computing Puiseux series for algebraic surfaces. In J. van der Hoeven and M. van Hoeij, editors, *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation (ISSAC 2012)*, pages 20–27. ACM, 2012.

[2] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8(3):295–313, 1992.

[3] J. Backelin. Square multiples n give infinitely many cyclic n-roots. Reports, Matematiska Institutionen 8, Stockholms universitet, 1989.

[4] R. Bagnara, P. Hill, and E. Zaffanella. The Parma Polyhedral Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[5] G. Björck and B. Saffari. New classes of finite unimodular sequences with unimodular Fourier transforms. Circulant Hadamard matrices with complex entries. *C. R. Acad. Sci. Paris, Série I*, 320:319–324, 1995.

[6] T. Bogart, A. Jensen, D. Speyer, B. Sturmfels, and R. Thomas. Computing tropical varieties. *Journal of Symbolic Computation*, 42(1):54–73, 2007.

[7] V. Braun and M. Hampton. `polyhedra` module of Sage. The Sage Development Team, 2011.

[8] I. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, University of California at Berkeley, Berkeley, 1994.

[9] I. Emiris, V. Fisikopoulos, and B. Gärtner. Efficient edge-skeleton computation for polytopes defined by oracles. *Journal of Symbolic Computation*, 73:139–152, 2016.

[10] H. Führ and Z. Rzeszotnik. On biunimodular vectors for unitary matrices. *Linear Algebra and its Applications*, 484:86–129, 2015.

[11] K. Fukuda and A. Prodon. Double description method revisited. In *Selected papers from the 8th Franco-Japanese and 4th Franco-Chinese Conference on Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer-Verlag, 1996.

[12] A. Jensen. Computing Gröbner fans and tropical varieties in Gfan. In M. Stillman, N. Takayama, and J. Verschelde, editors, *Software for Algebraic Geometry*, volume 148 of *The IMA Volumes in Mathematics and its Applications*, pages 33–46. Springer-Verlag, 2008.

[13] D. Maclagan and B. Sturmfels. *Introduction to Tropical Geometry*, volume 161 of *Graduate Studies in Mathematics*. American Mathematical Society, 2015.

[14] A. Novoseltsev. `lattice_polytope` module of Sage. The Sage Development Team, 2011.

[15] J. Sommars and J. Verschelde. Exact gift wrapping to prune the tree of edges of Newton polytopes to compute pretropisms. `arXiv:1512.01594`.

[16] W. Stein et al. *Sage Mathematics Software (Version 6.9)*. The Sage Development Team, 2015. `http://www.sagemath.org`.

[17] R. Walker. *Algebraic Curves*. Princeton University Press, 1950.

[18] G. Ziegler. *Lectures on Polytopes*. Springer-Verlag, 1995.