Computing Tropical Prevarieties in Parallel*

Anders Jensen† Jeff Sommars‡ Jan Verschelde§

23 June 2017

Abstract

The computation of the tropical prevariety is the first step in the application of polyhedral methods to compute positive dimensional solution sets of polynomial systems. In particular, pretropisms are candidate leading exponents for the power series developments of the solutions. The computation of the power series may start as soon as one pretropism is available, so our parallel computation of the tropical prevariety has an application in a pipelined solver.

We present a parallel implementation of dynamic enumeration. Our first distributed memory implementation with forked processes achieved good speedups, but quite often resulted in large variations in the execution times of the processes. The shared memory multithreaded version applies work stealing to reduce the variability of the run time. Our implementation applies the thread safe Parma Polyhedral Library (PPL), in exact arithmetic with the GNU Multiprecision Arithmetic Library (GMP), aided by the fast memory allocations of TCMalloc.

Our parallel implementation is capable of computing the tropical prevariety of the cyclic 16-roots problem. We also report on computational experiments on the n-body and n-vortex problems; our computational results compare favorably with Gfan.

1 Introduction

Given one polynomial in two variables, the Newton-Puiseux algorithm computes series expansions for the algebraic curve defined by the polynomial, departing from the edges of the Newton polyhedron. Given a polynomial system, the rays in the tropical prevariety may lead to the series expansions for the positive dimensional solution sets of the system. This paper presents a parallel algorithm to compute the tropical prevariety. We refer to [24] for a textbook introduction to tropical algebraic geometry.

The *Newton polytope* $\mathrm{NP}(f)$ of a polynomial f in n variables is the convex hull of the exponent vectors appearing with nonzero coefficient in f. The exponent vectors have integer coordinates. Each face of $\mathrm{NP}(f)$ has a normal cone. The set of all normal cones constitutes the *normal fan* of $\mathrm{NP}(f)$ which is a polyhedral fan in $\mathbb{R}^n$. The *tropical hypersurface* $\mathrm{T}(f)$ is the subfan of non-maximal cones. Given a polynomial system as a tuple of polynomials, the *tropical prevariety* that we wish to compute is the intersection of all tropical hypersurfaces of polynomials in the tuple.

The tropical prevariety is mainly a combinatorial object depending only on the Newton polytopes of the polynomials in the system, whereas the cancellation properties of the coefficients are captured by the *tropical variety* of the polynomial ideal of the system, which we shall not consider here.

For a general introduction to polytopes, we refer to [32]. In [5], the tropical prevariety was defined via the common refinement of normal fans. To be formally correct, we use the same definition. Given two

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

†Aarhus University

‡University of Illinois at Chicago

§University of Illinois at Chicago

fans F_1 and F_2, their *common refinement* $F_1 \wedge F_2$ is defined as

$$F_1 \wedge F_2 = \{C_1 \cap C_2 | (C_1, C_2) \in F_1 \times F_2\}. \quad (1)$$

As the common refinement of two fans is again a fan, the common refinement of three fans F_1, F_2, and F_3 may be computed as $(F_1 \wedge F_2) \wedge F_3$.

The *support* of a polyhedral fan is the union of its cones. We clarify the definition of the tropical prevariety of a tuple of polynomials $(f_1, f_2, \ldots, f_N)$ by defining it as the support of $T(f_1) \wedge T(f_2) \wedge \cdots \wedge T(f_N)$. The nonzero vectors in the prevariety are called *pretropisms*. The pretropisms are exactly the vectors normal to positive dimensional faces of each polytope in a tuple of Newton polytopes.

Problem Statement. Given a tuple of Newton polytopes
$(P_1, P_2, \ldots, P_N)$ with normal fans $(F_1, F_2, \ldots, F_N)$, efficiently compute the tropical prevariety of the fans with a parallel implementation in order to compute more challenging examples. Though the postprocessing of the cones of the prevariety is embarassingly parallel, computing the tropical prevariety is not, mainly because the cones in the output share parts of other cones.

Relation to Mixed Volumes. Our problem can be considered as a generalization of the mixed volume computation. For the relation between triangulations and mixed subdivisions, we refer to [8]. In the mixed volume computation, one intersects normal cones to the edges of each polytope. Linear programming models to prune superfluous edge-edge combinations were proposed first in [10]. Further developments can be found in [16] and [27]. A recent complexity study appeared in [25], along with a report on a parallel implementation of the mixed volume computation.

Most relevant for the algorithms presented in this paper is the dynamic enumeration introduced in [27].

Software. A practical study on various software packages for exact volume computation of a polytope is described in [6]. The authors of [11] present an experimental study of approximate polytope volume computation. In [12], a total polynomial-time algorithm is presented to compute the edge skeleton of a polytope.

Free dedicated software packages to compute mixed volumes are MixedVol [17] and DEMiCS [26]. In [26], the computation of the mixed volume for the cyclic 16-roots problem was reported for the first time. The mixed volume computation is included in PHCpack [30], pss5 [25], and gfanlib [23]. Gfan [22] contains software to compute the common refinement of the normal fans of the Newton polytopes. Gfan relies on cddlib [15] and optionally SoPlex [31] for lower level polyhedral computations.

The external software we used in our computations is the Parma Polyhedral Library (PPL) [2], and in particular its thread safe multithreading capabilities. Speedups were obtained with TCMalloc [18].

Our contributions. This paper extends the results of the last two authors, presented in [28, 29], extending the pruning algorithms with dynamic enumeration and a work stealing strategy. Our parallel implementation gives the first computation of the tropical prevariety for the cyclic 16-roots problem.

Acknowledgments. We thank Enea Zaffanella for developing a thread safe version of PPL.

2 Half Open Cones

We represent the support of a fan as a set of mutually disjoint cones. Such representations were also used in [7]. The key element in the representation is the notion of a half open cone, which we will define next.

While a (closed) polyhedral cone in $\mathbb{R}^n$ is a set of the form

$$\{\ x \in \mathbb{R}^n \mid Ax \leq 0\ \} \quad (2)$$

with $A \in \mathbb{R}^{m \times n}$ and the comparison done coordinatewise, a half open polyhedral cone is a set of the form

$$\{\ x \in \mathbb{R}^n \mid Ax \leq 0 \wedge A'x < 0\ \} \quad (3)$$

with $A \in \mathbb{R}^{m \times n}$ and $A' \in \mathbb{R}^{m' \times n}$.

2.1 Constructing Half Open Cones

We provide algorithms that divide the support of fans defined by a convex polytope P into a disjoint set of half open cones. We first explain how this is done for the normal fan of P and later how it is done for the tropical hypersurface of $T(P)$ of P, by which we mean the tropical hypersurface of a polynomial with Newton polytope P. Figure 2 shows an example of dividing the support of the normal fan of a single polytope into half open cones.

We begin by orienting the edge graph of the polytope using a random vector $r \in \mathbb{R}^n$ and then ordering the vertices by inner product with r. Assuming that these inner products are different, we find a unique sink orientation of the graph by giving each edge of the polytope a direction based on the vertex ordering, as in Figure 1.

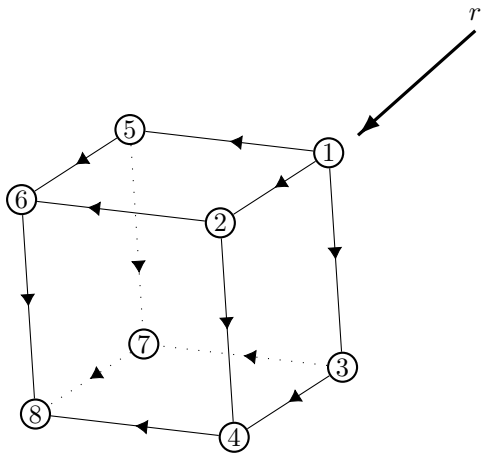

Figure 1: A cube oriented by a vector r.

Using the unique sink orientation, we define a half open normal cone to each vertex v of P in this way: for each edge e incident to v, create an inequality from it, making it strict if e is outgoing and non-strict if it is ingoing. The collection of all cones obtained as v varies divides $\mathbb{R}^n$ into a disjoint union. For the cones to cover $\mathbb{R}^n$, it is essential that the oriented graph is cycle free. This is guranteed by the construction.

To write the tropical hypersurface $T(P)$ as a disjoint union of half open cones, call Algorithm 1 below on each half open cone C constructed above. The output is a collection of half open cones covering exactly the non-interior points of C. Taking the union of all these collections as C varies results in a set of cones covering the support of $T(P)$ exactly and containing one cone for each edge of P.

Algorithm 1 Create half open cones of codimension one from a full dimensional half open cone.

Input: An inequality description of a full dimensional half open cone C
Output: A collection of disjoint half open cones with union equal to the boundary of C

```
function CreateHalfOpenCones(C)
    if C has only strict constraints then return ∅
    else
        Choose a non-strict constraint c of C
        C_< := C but with c being strict
        C_= := C but with c being an equation
        return C_= ∪ CreateHalfOpenCones(C_<)
    end if
end function
```

2.2 Represention as a Closed Cone

Linear programming involves sets of equations and non-strict inequalities defining closed polyhedra. Therefore we represent a half open cone C defined by matrices A and A' from (3) by the closed cone $C' \subseteq \mathbb{R}^{n+1}$ defined by the matrix

$$\begin{bmatrix} A & 0 \\ A' & 1 \end{bmatrix} \tag{4}$$

with a column of zeros and ones appended. Then $C = \pi(C' \cap (\mathbb{R}^n \times \mathbb{R}_{>0}))$ where $\pi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ denotes the projection forgetting the last coordinate. Observing that for all $x \in C$ we have $x \times \varepsilon \in C'$ for $\varepsilon > 0$ sufficiently small, we obtain $\overline{C} \subseteq \pi(C' \cap (\mathbb{R}^n \times \{0\}))$, while the converse inclusion holds if and only if C is non-empty. This happens if and only if $C' \cap (\mathbb{R}^n \times \mathbb{R}_{>0}) \neq \emptyset$ and in that case $\dim(C') = \dim(C) + 1$. Non-emptyness and other properties can be determined with linear programming.

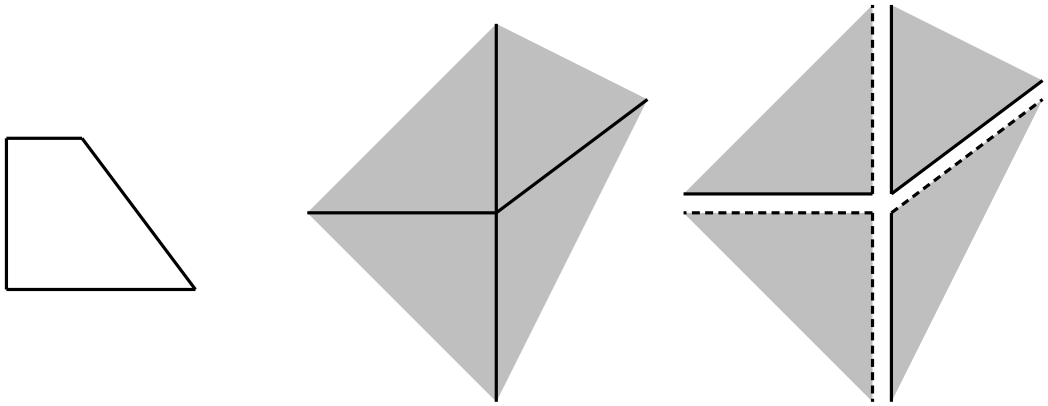

Figure 2: *Left*: A Newton polytope P. *Center*: The normal fan F of P. *Right*: F split apart into four half open cones. The dashed lines represent boundaries that are not contained by a cone, while the solid lines represent boundaries that are contained by a cone. In this example, the upper right cone contains the origin.

3 Static Enumeration

The common refinement (1) in the definition of the tropical prevariety has a constructive formulation: compute the intersection of every combination of N cones, one from each fan, and add the non-empty intersections to the output. This combinatorial algorithm requires $\prod_{i=1}^{N} |P_i|$ cone intersections, where $|P_i|$ is the number of edges of polytope P_i.

The recursive formulation in Algorithm 2 (defined as *static enumeration*) performs substantially fewer cone intersections.

Algorithm 2 Static enumeration

Input: A list F of fans $F_1, \dots F_N$ in $\mathbb{R}^n$ where each F_i is represented by a list of cones covering the support of F_i.

Output: A list of cones covering the support of $F_1 \wedge \cdots \wedge F_N$.

```
    procedure StaticEnumeration(Cone C, Index i)
        if C ≠ ∅ then
            if i > |F| then
                Output C
 5:         else
                for each cone D in F_i do
                    StaticEnumeration(C ∩ D, i + 1)
                end for
            end if
10:     end if
    end procedure
    StaticEnumeration(ℝ^n, 1)
```

The recursive execution of Algorithm 2 leads to a tree of cone intersections with a default depth first traversal. At the ith level of the tree are a set of cones comprising the common refinement of the first i fans. If the cones in Algorithm 2 are closed, then the intersection C is always non-empty. In particular one may want to remove duplicate cones at each level of the tree. If the fans F_i in Algorithm 2 each consist of mutually disjoint half open cones, then the occurrence of duplicate intersections is avoided and cone intersections can indeed be empty. Thus, working with half open cones in Algorithm 2 brings another substantial improvement.

4 Dynamic Enumeration

Inspired by [16] and [27], we reduce the number of cone intersections by dynamic enumeration. Dynamic enumeration can be viewed as a greedy method to reorder the fans during the computation, thereby affecting the shape of the tree of cone intersections. A random permutation of the N polytopes before the start of Algorithm 2 defines a sub-optimal order in which to intersect cones. Using a greedy metric, defined in Section 4.1, we determine with which fan it is best to start.

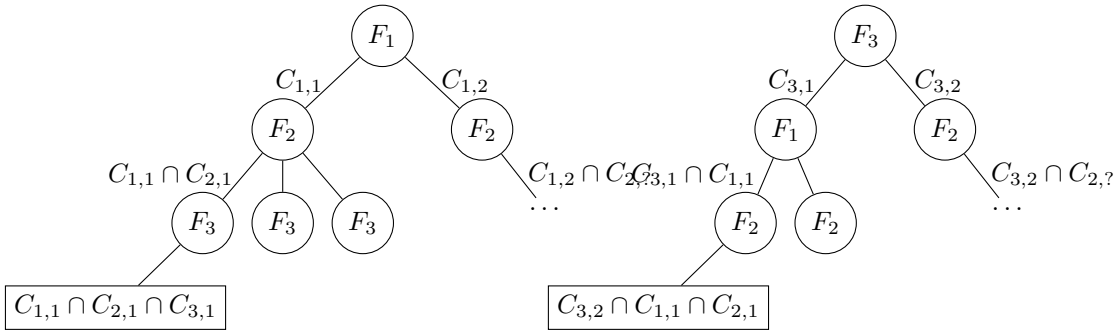

Figure 3: These trees illustrate the difference between static and dynamic enumeration for three fans F_1, F_2, and F_3. The left tree represents static enumeration, where the ordering of the fans is established and does not change. The right tree represents dynamic enumeration, where the starting fan is greedily selected, and each cone greedily chooses which fan to be intersected with next.

Algorithm 3 Dynamic enumeration

Input: A list F of fans $F_1, \dots F_N$ in $\mathbb{R}^n$ where each F_i is represented by a list of cones covering the support of F_i.

Output: A list of cones covering the support of $F_1 \wedge \cdots \wedge F_N$.

```
    procedure DynamicEnumeration(Cone C, Set I)
        if C ≠ ∅ then
            if I = ∅ then
                Output C
 5:         else
                Greedily choose index i ∈ I.
                for each cone D in F_i do
                    DynamicEnumeration(C ∩ D, I \ {i})
                end for
10:         end if
        end if
    end procedure
    DynamicEnumeration(ℝ^n, {1, ..., |F|})
```

Every intermediate cone intersection C must be intersected with each remaining fan in order to contribute to the final common refinement, but it can be intersected with the fans in any order. We greedily choose the next fan to intersect with C, as a fan with which C is expected to have few non-empty intersections. After intersecting with the fan we get a new set of cones, each contained within C. Each of these new cones must be intersected with the remaining fans, but this can happen in whatever order seems to be the most efficient, determined through greedy selection. This process ends when C is the result of an intersection of a cone from each fan.

Algorithm 3 and Algorithm 2 have recursion trees with different shapes. In the setting of mixed volume computation it was observed in [26] that the tree of Algorithm 3 has far fewer vertices. Figure 3 demonstrates the difference between the tree traversal of static enumeration and dynamic enumeration.

4.1 Greedy Selection

The basic unit of work for Algorithm 2 and Algorithm 3 is intersecting a pair of polyhedral cones. Every greedy choice we make is done to minimize the number of necessary intersections while avoiding adding

an additional computational burden. To this end, the choice of the first fan is easy: pick the fan with the fewest cones.

The greedy metric used during the tree traversal is more difficult. To facilitate the choices, we use relation tables (introduced in [16]), tables that store whether or not pairs of cones could intersect. Before we define a relation table, it is useful to introduce additional notation: call $C_{i,j}$ the jth cone of fan $T(P_i)$. For a cone C of fan $T(P)$, we define *the relation table* $R(i,j)$ to be a boolean array such that

$$R(i,j) = \begin{cases} 1, & \text{if } C \cap C_{i,j} \neq \emptyset \\ 0, & \text{if } C \cap C_{i,j} = \emptyset \\ 0, & \text{if } P = P_i \end{cases} \tag{5}$$

where $1 \leq i \leq N$ and $1 \leq j \leq \#\text{Edges}(P_i)$. It is faster to check if $C \cap D = \emptyset$ than it is to compute $C \cap D$ with redundant inequalities removed. Checking if an intersection is empty is equivalent to checking the feasibility of a linear system.

Before our algorithm begins, we initialize each cone's relation table by intersecting each cone C with every cone not in the same fan. We store this information compactly on a bit array on each cone object. When we intersect cone objects in Algorithm 5, not only do we intersect the polyhedral cones, but we also intersect their associated relation tables. Intersecting relation tables requires creating a new bit array that is equal to a bitwise AND of the two input bit arrays, as shown in Figure 4. As this is a bit operation, it is very fast.

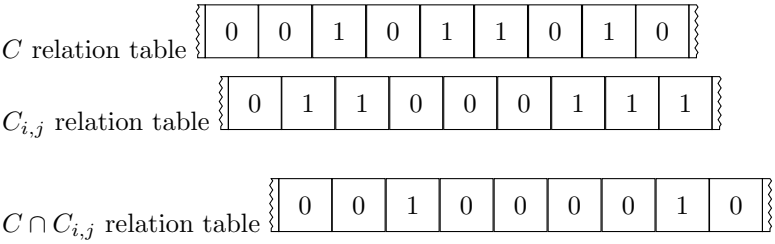

Figure 4: Sample intersection of two relation tables. Each relation table is an array of bits, so intersecting a pair of relation tables is a bitwise AND.

To make the greedy choices that our algorithm requires, our software consults a cone's relation table to find the unused fan with the fewest cones with which it could intersect. This greedy selection requires minimal additional computation and leads to large speedups as will be shown in Section 6. An additional benefit of the relation tables is that they allow us to avoid intersecting cones that are already known to not intersect. This is illustrated in Algorithm 4.

Algorithm 4 Full dynamic enumeration algorithm

Input: A list F of fans $F_1, \dots F_N$ in $\mathbb{R}^n$ where each F_i is represented by a list of cones covering the support of F_i.

Output: A list of cones covering the support of $F_1 \wedge \cdots \wedge F_N$.

```
procedure DynamicEnumeration(Cone C, Set I)
    if C ≠ ∅ then
        if I = ∅ then
            Output C
        else
            Choose index i ∈ I such that F_i has fewest
              cones which C could intersect.
            for each cone D in F_i do
                if C's relation table allows C ∩ D ≠ ∅ then
                    Intersect C's relation table with D's
                      relation table, and store on C ∩ D
                    DynamicEnumeration(C ∩ D, I \ {i})
                end if
            end for
        end if
    end if
end procedure
Compute relation tables for R^n and the cones in F
DynamicEnumeration(R^n, {1, ..., |F|})
```

5 Parallel Implementation

The problem of computing a tropical prevariety can be seen as traversing a tree, which can run in parallel.

While the polytopes, fans, and cones in the tropical prevariety all live in $\mathbb{R}^n$, we point out that their defining data is exact, spanned by points with *integer* coordinates. All our computations are performed with arbitrary precision integer arithmetic. The rapid coefficient growth is not polynomial in n, but n is relatively small in our applications.

5.1 Software Setup

Our choices of software packages both enables and limits our options for a parallel implementation.

We chose to use the Parma Polyhedra Library (PPL) for all polyhedral computations [2]. We first call it to find the vertices of the Newton polytopes, and we later call it many times to compute intersections of polyhedral cones. PPL uses arbitrary precision integers during its computations via the GNU Multi Precision Arithmetic Library (GMP) [13]. If we had chosen to use a library that did not use arbitrary precision integers, the software would need to exercise care and certify that there were no incorrect answers due to floating point error. Since computing a prevariety requires intersecting many polyhedral cones in sequence, as the depth of the tree increases, the likelihood of floating point error also increases. Using a polyhedral library that uses GMP integers avoids this challenge.

PPL has recently become threadsafe, which allows us to use it in a parallel implementation. Unfortunately, if multiple threads are intersecting pairs of polyhedra simultaneously, linear speedups will not be achieved. This is due to the fact that when multiple threads attempt to allocate GMP integers from the heap, a linear speedup is not attained. Modest improvements in speedup come from using the allocator TCMalloc [18].

5.2 Design Considerations

For computing a tropical prevariety, we distinguish three stages in the algorithm. For each of the three stages, we consider its parallel execution.

The algorithm begins by computing the vertices of the Newton polytopes, which is necessary to determine the normal fans. Computing the vertices of distinct polytopes can be done in parallel, but we have found it not to be necessary. The polynomial systems of interest are sparse, with small Newton polytopes spanned by relatively few monomials; if this were not the case, determining the vertices would be more difficult. In our most computationally intensive benchmark, computing the vertices takes less than a second for a single thread. Therefore, this component of the algorithm is not considered to run in parallel.

The second stage we consider is the computation of the relation tables. Filling the relation tables requires computing the intersection of many pairs of polyhedral cones and testing if that intersection is non-empty. This amounts to a job queue, where each job is the intersection of two polyhedral cones. The queue is filled with all of the necessary jobs, then each process pops a job, records the result of the intersection and returns to the queue. This process continues until the job queue has been emptied.

The third stage of the algorithm leads to the greatest benefit: developing a parallel version of the recursion in Algorithm 4. This will be addressed in the following sections.

5.3 Coarse Grained Parallelism

A first, coarse grained parallel version of Algorithm 4 was implemented using forked processes, dividing the cones of the starting fan among several processes.

Each process took its starting cones and performed Algorithm 4 on them, terminating when finished. This approach was a natural starting point, as it did not require communication between threads and it was straightforward to implement. Since the processes were distinct, each thread had its own heap, so we were closer to achieving linear speedups in the polyhedral computations. However, the time required for each process varied dramatically. For a run of the cyclic-16 roots polynomial system with twenty threads, the fastest threads finished in less than a day while the slowest thread took more than three weeks to finish. This was an inefficient use of resources, as a good parallel implementation uses all of a computer's available resources for the duration of the computation.

5.4 Work Stealing

Our current parallel implementation applies work stealing [4], using the run time parallel library provided by PPL.

The first barrier to creating a work stealing implementation of the dynamic enumeration method is that Algorithm 4 is a recursive algorithm, so it lacks a job queue. We define a single job to be taking a cone and intersecting it with the normal cones of a polytope; Algorithm 5 transforms Algorithm 4 into an algorithm with a work queue of these jobs. This version of Algorithm 4 will find the prevariety with the same number of cone intersections, but it will find the cones of the prevariety in a different order.

There are two benefits to finding cones in the tropical prevariety quickly. Once a cone in the prevariety has been discovered, it can be printed to file, so the memory that it consumed can be freed. Additionally, when cones in the prevariety are found, post-processing can begin, which may vary depending on the application. One application of interest is computing power series expansions of positive dimensional solution sets of polynomial systems. Each cone could lead to several distinct power series, thus this process could begin as soon as a single cone has been found.

To find cones in the prevariety as quickly as possible, the job queue is implemented as in Figure 5, thereby also making it more stack-like. When Algorithm 5 begins, cones are placed into an initial subqueue,

Algorithm 5 Iterative version of dynamic enumeration

Input: A list of fans $F_1, \ldots F_N$ in $\mathbb{R}^n$ where each F_i is represented by a list of cones covering the support of F_i.

Output: A list of cones covering the support of $F_1 \wedge \cdots \wedge F_N$.

```
    Compute relation tables
    F := fan with fewest cones
    Cones := Cones from F
    while Cones≠ ∅ do
 5:     C := remove an element from Cones
        Choose fan F' not used to produce C such that F'
            has fewest cones with which C could intersect.
        for each cone D in F' do
            if C's relation table allows C ∩ D ≠ ∅ then
10:             Compute C ∩ D
                if C ∩ D ≠ ∅ then
                    if C ∩ D used all fans then
                        Output C ∩ D
                    else
15:                     Intersect C's relation table with D's
                            relation table, and store on C ∩ D
                        Add C ∩ D to Cones
                    end if
                end if
20:         end if
        end for
    end while
```

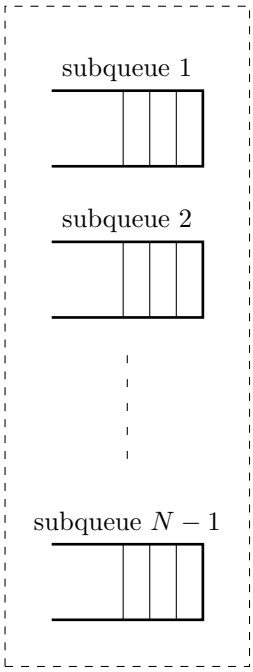

Figure 5: A queue made up of subqueues. Subqueue 1 contains cones from the starting polytope, while subqueue $N-1$ contains cones that are that are the intersection of $N-1$ cones.

subqueue 1. When a cone from subqueue 1 is removed and has been intersected with a set of cones from another polytope, the resulting cones are put into subqueue 2. To achieve the goal of finding cones in the prevariety quickly, when choosing the next job, an optimal strategy requires picking a cone from the subqueue with the highest index.

To transform this algorithm into a multi-threaded work stealing algorithm, each thread must have its own queue in the style of Figure 5. When picking a job to execute, a thread first looks to its own queue and picks a cone from the subqueue of highest index. When a thread's queue is empty, it looks to steal from another thread's queue, but steals the job from the subqueue of lowest index, as to require stealing less often. Furthermore, if there are j total threads, the ith thread looks to steal from threads in the following order: $i+1, i+2, \ldots, j, 1, 2, \ldots i-1$. This avoids having all threads attempting to steal from the same thread, keeping the theft spread out among different threads and requiring fewer total robberies.

Since there is no communication between the various branches of the enumeration tree in Algorithm 4, an alternative to dealing with work stealing directly is to phrase the recursive part of the algorithm as an abstract tree traversal and apply a general purpose parallel tree traverser. Thereby the parallelization aspects are entirely seperated from the problem domain. This approach was used in [23] for the situation of mixed volume computation.

6 Experimental Results

For finding isolated solutions of polynomial systems, there exist many standard benchmark problems. However, few of these problems have positive dimensional components as well, which makes them inappropriate test cases for tropical prevarieties. We will mention results from three standard benchmark problems with positive dimensional solution components as well as one problem from tropical geometry.

The code is available at `https://github.com/sommars/DynamicPrevariety`. We compare it against Gfan which contains a single threaded and less efficient implementation of a variant of the dynamic enumeration algorithm. Except for the Gfan timings, all computations were done on a 2.2 GHz Intel Xeon E5-2699 processor in a CentOS Linux workstation with 256 GB RAM using varying numbers of threads.

6.1 n-body and n-vortex Problems

For equal masses, the central configurations in the classical n-body problem are solutions to the $\binom{n}{2}$ Albouy-Chenciner equations obtained by clearing denominators of the equations

$$\sum_{k=1}^{n}(x_{ik}^{-3}-1)(x_{jk}^2-x_{ik}^2-x_{ij}^2)+(x_{jk}^{-3}-1)(x_{ik}^2-x_{jk}^2-x_{ij}^2)=0 \tag{6}$$

indexed by i and j where $1 \leq i < j \leq n$, and there are $\binom{n}{2}$ pairwise distance variables $x_{12} \dots x_{(n-1)n}$ and $x_{ij} = x_{ji}$.

The n-vortex problem [21] arose from a generalization of a problem from fluid dynamics that attempted to model vortex filaments. In this setting, the $\binom{n}{2}$ Albouy-Chenciner equations are obtained by clearing denominators of

$$\sum_{k=1}^{n}(x_{ik}^{-2}-1)(x_{jk}^2-x_{ik}^2-x_{ij}^2)+(x_{jk}^{-2}-1)(x_{ik}^2-x_{jk}^2-x_{ij}^2)=0. \tag{7}$$

n	Static Enum	Dyn. Enum.	#Rays	Gfan	1 thread	10 threads	20 threads
4	114	114	2	0.020s	0.008s	0.017s	0.028s
5	682	676	0	0.058s	0.036s	0.053s	0.073s
6	2,286	2,254	8	0.22s	0.10s	0.11s	0.16s
7	7,397	7,163	28	0.64s	0.29s	0.26s	0.37s
8	19,619	18,315	94	2.87s	0.79s	0.49s	0.70s
9	63,109	50,584	276	13.0s	2.8s	1.2s	1.4s
10	269,223	160,203	712	1m22s	9.8s	4.4s	3.7s
11	1,625,520	827,469	2,244	9m17s	50s	16.8s	20.3s
12	11,040,912	5,044,441	5,582	82m33s	5m2s	1m5s	1m3s
13		36,633,391	14,872		46m59s	8m30s	6m20s
14		264,463,730	49,114		6h22m56s	67m31s	46m37s
15		1,852,158,881	145,276			10h25m45s	7h43m57s
16		13,715,434,028	527,126			84h20m37s	62h36m31s

Table 1: This table contains results from experiments with the cyclic-n roots problem. The left half contains the number of cone intersections required in static enumeration and dynamic enumeration, as well as the number of rays i.e. 1-dimensional cones in the produced fan. The right half of the table contains timings of Gfan and the dynamic prevariety software run with 1, 10, or 20 threads. The Gfan timings are for a single thread running on an Intel Xeon E2670 CPU. SoPlex [31] was enabled in the Gfan timings, providing a speed up of roughly a factor 3.

Computing a tropical prevariety was essential in the argument of finiteness of the relative equilibria in the 4-body problem [20]. Tables 2 and 3 contain data from experiments with our implementation run on the n-body and n-vortex equations above. Since the problems increase in difficulty quickly, we can only compute few prevarieties in the family.

n	#Rays	1 thread	20 threads
3	4	0.014s	0.038s
4	57	0.77s	0.61s
5	2908	2m37s	34s

Table 2: n-body problem: number of rays and timings of the new software run with 1 or 20 threads. The 6-body problem did not terminate in two days when run with 20 threads.

n	#Rays	1 thread	20 threads
3	4	0.011s	0.03s
4	27	0.44s	0.42s
5	643	30.1s	10.9s
6	152,514		3h16m13s

Table 3: n-vortex problem: number of rays and timings of the new software run with 1 or 20 threads. The 7-vortex problem did not terminate in two days when run with 20 threads.

6.2 4×4 minors of a 5×5 matrix

In [9], the authors pose the question of whether or not the 4×4 minors of a 5×5 matrix form a tropical basis. It was answered in the affirmative [7], where one proof strategy required computing the prevariety defined by the 4×4 minors of the following matrix:

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{pmatrix}. \tag{8}$$

In [7], the 4×4 minors prevariety was computed in two weeks, using four threads, or eight weeks of computation time if run linearly. They also exploited symmetry of the problem, which reduced the computation time by an expected factor between 2-10x.

The dynamic prevariety software ran in under ten hours using twenty threads. However, these two trials cannot fairly be compared, as processor speeds have increased in the eight years since the computation was run in [7].

6.3 Cyclic-n roots

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

$$\begin{cases} x_0 + x_1 + \cdots + x_{n-1} = 0 \\ i = 2, 3, \ldots, n-1 : \displaystyle\sum_{j=0}^{n-1} \prod_{k=j}^{j+i-1} x_{k \bmod n} = 0 \\ x_0 x_1 x_2 \cdots x_{n-1} - 1 = 0. \end{cases} \tag{9}$$

This problem is important in the study of biunimodular vectors, a notion that traces back to Gauss, as stated in [14]. In [1], 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 [3], [14, Conjecture 1.1]

is that if n is not divisible by a square, then the set of cyclic n-roots is finite. If the dimension is a prime number, then the number of solutions if finite, as proven in [19] with an explicit count of the number of solutions given.

The cyclic-n roots problem scales slowly, so it is a good case study to examine the effectiveness of Algorithm 5 in detail. The first two columns of Table 1 show that as the size of the problem increases, the benefit of dynamic enumeration increases as well.

From the right hand portion of Table 1, it can be seen that there is a speedup as the number of threads increases. In cyclic-14, a speedup of 5.67 was achieved with ten threads while a speedup of 8.21 was achieved with twenty threads. These speedups are not linear, due to the GMP integer allocation issue. However, computing in parallel dramatically reduces computation time, so it is beneficial in practice.

As the number of cones increase, postprocessing the results becomes increasingly difficult. For cyclic-16, there are many cones of high dimension, which makes the prevariety more challenging to compute, see Table 4.

Dim.	#Maximal Cones
1	0
2	768
3	114,432
4	1,169,792
5	1,007,616
6	2,443,136
7	4,743,904
8	109,920

Table 4: Number of maximal cones in the prevariety of cyclic-16 by dimension.

References

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

[2] R. Bagnara, P.M. 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.

[3] 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.

[4] R. Blumofe and C. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.

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

[6] B. Büeler, A. Enge, and K. Fukuda. Exact volume computation for polytopes: a practical study. In G. Kalai and G.M. Ziegler, editors, *Polytopes – Combinatorics and Computation*, volume 29 of *DMV Seminar*, pages 131–154. Springer-Verlag, 2000.

[7] M. Chan, A. Jensen, and E. Rubei. The 4x4 minors of a 5xn matrix are a tropical basis. *Linear Algebra and its Applications*, 435(7):1598–1611, 2011.

[8] J.A. De Loera, J. Rambau, and F. Santos. *Triangulations. Structures for Algorithms and Applications*, volume 25 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2010.

[9] M. Develin, F. Santos, and B. Sturmfels. On the rank of a tropical matrix. *Discrete Comput. Geom.*, 52:213–242, 2005.

[10] I.Z. Emiris and J.F. Canny. Efficient incremental algorithms for the sparse resultant and the mixed volume. *Journal of Symbolic Computation*, 20(2):117–149, 1995.

[11] I.Z. Emiris and V. Fisikopoulos. Efficient random-walk methods for approximating polytope volume. In *Proceedings of the thirtieth annual symposium on computational geometry (SoCG'14)*, pages 318–327. ACM, 2014.

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

[13] Torbjörn Granlund et al. GNU multiple precision arithmetic library 6.1.2, 2016.

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

[15] 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.

[16] T. Gao and T.Y. Li. Mixed volume computation for semi-mixed systems. *Discrete Comput. Geom.*, 29(2):257–277, 2003.

[17] T. Gao, T.Y. Li, and M. Wu. Algorithm 846: MixedVol: a software package for mixed-volume computation. *ACM Trans. Math. Softw.*, 31(4):555–560, 2005.

[18] S. Ghemawat and Menage P. Tcmalloc: Thread-caching malloc. `http://goog-perftools.sourceforge.net/doc/tcmalloc.html`, 2005.

[19] U. Haagerup. Cyclic p-roots of prime length p and related complex Hadamard matrices. Preprint available at `arXiv:0803.2629`, 2008.

[20] M. Hampton and R. Moeckel. Finiteness of relative equilibria of the four-body problem. *Inventiones Mathematicae*, 163:289–312, 2006.

[21] M. Hampton and R. Moeckel. Finiteness of stationary configurations of the four-vortex problem. *Transactions of the American Mathematical Society*, 361(3):1317–1332, 2009.

[22] A.N. Jensen. Computing Gröbner fans and tropical varieties in Gfan. In M.E. 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.

[23] A.N. Jensen. An implementation of exact mixed volume computation. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – ICMS 2016, 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings*, volume 9725 of *Lecture Notes in Computer Science*, pages 198–205. Springer-Verlag, 2016.

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

[25] G. Malajovich. Computing mixed volume and all mixed cells in quermassintegral time. *Found. Comput. Math.*, pages 1–42, 2016.

[26] T. Mizutani and A. Takeda. DEMiCs: a software package for computing the mixed volume via dynamic enumeration of all mixed cells. In M.E. Stillman, N. Takayama, and J. Verschelde, editors, *Software for Algebraic Geometry*, volume 148 of *The IMA Volumes in Mathematics and Its Applications*, pages 59–79. Springer-Verlag, 2008.

[27] T. Mizutani, A. Takeda, and M. Kojima. Dynamic enumeration of all mixed cells. *Discrete Comput. Geom.*, 37(3):351–367, 2007.

[28] J. Sommars and J. Verschelde. Computing pretropisms for the cyclic n-roots problem. In *32nd European Workshop on Computational Geometry (EuroCG'16)*, pages 235–238, 2016.

[29] J. Sommars and J. Verschelde. Pruning algorithms for pretropisms of Newton polytopes. In V.P. Gerdt, W. Koepf, W.M. Seiler, and E.V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing, 18th International Workshop, CASC 2016, Bucharest, Romania*, volume 9890 of *Lecture Notes in Computer Science*, pages 489–503. Springer-Verlag, 2016.

[30] J. Verschelde. Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.*, 25(2):251–276, 1999.

[31] Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. `http://www.zib.de/Publications/abstracts/TR-96-09/`.

[32] G.M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag, 1995.