# libtropicon: A scalable library for computing intersection points of generic tropical hyper-surfaces

Tianran Chen[1]

Auburn University at Montgomery, USA
ti@nranchen.org,
www.tianranchen.org

**Abstract.** The computation of intersection points of generic tropical hyper-surfaces is a fundamental problem in computational algebraic geometry. An efficient algorithm for solving this problem will be a basic building block in many higher level algorithms for studying tropical varieties, computing mixed volume, enumerating mixed cells, constructing polyhedral homotopies, etc. libtropicon is a library for computing intersection points of generic tropical hyper-surfaces that provides a unified framework where the several conceptually opposite approaches coexist and complement one another. In particular, great efficiency is achieve by the data cross-feeding of the "pivoting" and the "elimination" step — data by-product generated by the pivoting step is selectively saved to bootstrap the elimination step, and vice versa. The core algorithm is designed to be naturally parallel and highly scalable, and the implementation directly supports multi-core architectures, computer clusters, and GPUs based on CUDA or ROCm/OpenCL technology. Many-core architectures such as Intel Xeon Phi are also partially supported. This library also includes interface layers that allows it to be tightly integrated into the existing ecosystem of software in computational algebraic geometry.

**Keywords:** Tropical hypersurfaces, Mixed volume, Mixed cells, BKK bound, Polyhedral homotopy

## 1 Introduction

Finding common solutions to a multivariate nonlinear polynomial system is a fundamental problem in computational mathematics with a great variety of important applications. While the most meaningful space for searching for such common solutions is the field of complex numbers $\mathbb{C}$ due to the rather special feature of being *algebraic closed*, the vastness of $\mathbb{C}$, however, poses challenges for computer programs. For instance, from a measure theory point of view, almost no complex number can be represented exactly using floating point numbers. In this light, the "tropical intersection problem" can be viewed as a particularly attractive discretization of this problem. In this tropical version, $\mathbb{C}$ is replaced by a *tropical semiring* [8] which is the extended real numbers with the addition

and multiplication operations defined as minimum and addition respectively. The notion of polynomial functions and their zero sets can be extended to this semiring, and we thus have the "tropicalized" problem of finding intersections of zero sets for multivariate nonlinear polynomial systems in this context. Among several different computational aspects in this rich framework, libtropicon focuses on the problem of computing "generic intersections". The finite and combinatorial nature of the generic intersection problem make it particularly suitable for numerical computation (using floating point arithmetic). Yet great deal of information of about the original algebraic problems can be uncovered from these tropical intersections. For instance, generic root count (intersection number) of a given polynomial system can be directly extracted from tropical intersections. More importantly, from a computational view point, the rich data structures produced by such tropical intersection points are also the necessary ingredient for constructing a polyhedral homotopy method [3] for numerically locating all complex roots of the original polynomial system.

## 2 Problem statement

The mathematical problem that libtropicon solves is the computation of intersection points of generic tropical hypersurfaces. Interestingly, this problem coincide with several different problems with seemingly very different origins. We briefly describe some of the key formulations.

### 2.1 Tropical formulation

Consider the *tropical semiring* $\mathbb{T} = (\mathbb{R} \cup \{\infty\}, \oplus, \odot)$ with $a \oplus b = \min(a, b)$ and $a \odot b = a + b$. Given a Laurent polynomial (a polynomial with potentially negative exponents) $p(\mathbf{x}) = \sum_{\mathbf{a} \in S} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}$ where $\mathbf{x}^{\mathbf{a}} = (x_1, \ldots, x_n)^{(a_1, \ldots, a_n)^\top} = x_1^{a_1} \cdots x_n^{a_n}$, it gives rise to the piecewise linear function

$$L(\mathbf{x}) = \min \{ \, c_{\mathbf{a}} + \langle \mathbf{a}, \mathbf{x} \rangle \mid \mathbf{a} \in S \, \}, \tag{1}$$

if addition and multiplication are to be interpreted as the tropical operations $\oplus$ and $\odot$ respectively. The "zero set" $\mathcal{V}(p)$ of $p$ over $\mathbb{T}$ is then defined to be the set of points where $L$ is not differentiable, i.e., points where two or more linear pieces of $L(\mathbf{x})$ meet. Following the terminology from algebraic geometry (over $\mathbb{C}$), such a zero set will be known as a *tropical hypersurface* when it is of codimension one.

*Problem 1.* Given sets of monomials $M_1, \ldots, M_r \subset \mathbb{R}[x_1^{\pm}, \ldots, x_n^{\pm}]$, we consider generic Laurent polynomials $p_1, \ldots, p_r$ in these sets of monomials, i.e., each $p_k$ is a linear combination of monomials in $M_k$ with generic (nonzero) real coefficients. We want to find the intersection of the tropical hypersurface $\bigcap_{k=1}^{r} \mathcal{V}(p_k)$.

### 2.2 Mixed cells formulation

The above problem has a nearly equivalent formulation rooted from generalized coherent subdivision problem for point configurations: Given finite sets $S_1, \ldots, S_r \subset \mathbb{R}^n$, an $r$-tuple $(t_1, \ldots, t_r) \in (\mathbb{Z}^+)^r$ with $\sum_{k=1}^r t_k = n$, and functions $\omega_1, \ldots, \omega_r$ with each $\omega_k : S_k \to \mathbb{R}$ having generic images, we define $\hat{S}_k = \{(\mathbf{a}, \omega_k(\mathbf{a})) \mid \mathbf{a} \in S_k\} \subset \mathbb{R}^{n+1}$ for $k = 1, \ldots, r$. The main problem is to find all the $r$-tuple of faces of $\mathrm{conv}(\hat{S}_1), \ldots, \mathrm{conv}(\hat{S}_r)$ matching the dimensions given by $(t_1, \ldots, t_r)$ that can share the same "upward pointing" inner normal vector:

*Problem 2.* Given $S_1, \ldots, S_r$ and $\omega_1, \ldots, \omega_r$ described above, we want to find *all* possible $r$-tuples $(\{\mathbf{a}_{i_{1,0}}, \ldots, \mathbf{a}_{i_{1,t_1}}\}, \ldots, \{\mathbf{a}_{i_{r,0}}, \ldots, \mathbf{a}_{i_{r,t_r}}\})$ of subsets of $\hat{S}_1, \ldots, \hat{S}_r$ respectively for which there exists an $\alpha \in \mathbb{R}^n$ such that for each $k = 1, \ldots, r$

$$\begin{aligned}
&\langle \mathbf{a}_{i_{k,0}}, \alpha + \omega_k(\mathbf{a}_{i_{k,0}}) = \langle \mathbf{a}_{i_{k,j}}, \alpha + \omega_k(\mathbf{a}_{i_{k,j}}) \quad \text{for} j = 1, \ldots, t_k \\
&\langle \mathbf{a}_{i_{k,1}}, \alpha + \omega_k(\mathbf{a}_{i_{k,1}}) < \ \langle \mathbf{a}, \alpha \ + \omega_k(\mathbf{a}) \quad \text{for } \mathbf{a} \in S_k \setminus \{\mathbf{a}_{i_{k,0}}, \ldots, \mathbf{a}_{i_{k,t_k}}\}.
\end{aligned}$$

It can be verified that in the $r = n$ case, Problems 2 and 1 are equivalent despite the rather different presentations. The $r$-tuples of points are known as *mixed cells* [3] of type $(t_1, \ldots, t_r)$, and they play a crucial role in the construction of polyhedral homotopies for solving a Laurent polynomial system.

### 2.3 Incremental Cayley's trick formulation

Yet another formulation of this problem is connected to the well known *Cayley's trick* and the *phase one problem* in linear programming. By introducing a new set of variables $h_k := \langle \mathbf{a}_{i_{k,0}}, \alpha \rangle + \omega_k(\mathbf{a}_{i_{k,0}})$ for $k = 1, \ldots, r$ as in Problem 2, we obtain the equivalent system

$$\begin{aligned}
\langle \mathbf{a}_{i_{k,j}}, \alpha \rangle - h_k &= -\omega_k(\mathbf{a}_{i_{k,j}}) \quad \text{for } j = 1, \ldots, t_k \\
\langle \mathbf{a}, \alpha \rangle - h_k &> -\omega_k(\mathbf{a}) \quad \text{for } \mathbf{a} \in S_k \setminus \{\mathbf{a}_{i_{k,1}}, \mathbf{a}_{i_{k,2}}\},
\end{aligned}$$

With this, we get a reformulation of Problem 2 that resembles a generalized "Phase One" problem in linear programming:

*Problem 3.* Given $S_1, \ldots, S_r$, $(t_1, \ldots, t_r)$, and $\omega_1, \ldots, \omega_r$ described above, let $\check{A}_k$ to be the matrix whose rows are $(\mathbf{a}, -1)$ for points $\mathbf{a} \in S_k$, and let $\mathbf{c}_k$ be the column vector with corresponding entries of $-\omega_k(\mathbf{a})$ for $\mathbf{a} \in S_k$. We want to find *all* possible $(\alpha, h_1, \ldots, h_r)$ such that

$$\check{A}_k \begin{bmatrix} \alpha \\ h_k \end{bmatrix} \geq \mathbf{c}_k \text{ with } t_k \text{ equalities hold, for each } k = 1, \ldots, r. \tag{2}$$

Here, the term "incremental" Cayley's trick refers to the fact that each group of inequality in (2) is embedded into $\mathbb{R}^{n+1}$ separately. Note that for each $k$, (2) is a generalized version of the Phase-One problem in linear programming. Therefore Problem 3 is a problem of simultaneous Phase-One problem: it requires the solutions $(\alpha, h_1), \ldots, (\alpha, h_r)$ to the $r$ different generalized Phase-One problem to share the same projection onto the first $n$ coordinates.

## 3 Applications

While finding the generic intersection points of tropical hypersurfaces is an interesting problem in its own right [5, 9], the result also has a variety of important applications in computational mathematics which we shall outline below.

### 3.1 Root counting problem and the BKK bound

One direct application of the generic tropical intersection points is the root counting problem. For each intersection point (component) in Problem 1, an intersection number (multiplicity) can be defined. The generic intersection number of the tropical hypersurfaces is defined to be the sum of all the intersection numbers at all the intersection points.

**Theorem 1 (Huber & Sturmfels [3]).** *Given a system of $n$ Laurent polynomials $f_1, \ldots, f_n$ in $n$ variables with generic coefficients, the total number of isolated common zeros in $(\mathbb{C}^*)^n$ equals the generic intersection number of the $n$ tropical hypersurfaces defined by $f_1, \ldots, f_n$.*

Here $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$. This generic root count is known as the BKK bound or the mixed volume bound. This restriction on the domain of the root counting problem is minor and has since been removed [4, 7, 12].

### 3.2 Polyhedral homotopy

The constructive proof of Theorem 1 gives rise to a numerical homotopy method for solving polynomial systems. Given a system of (Laurent) polynomials $F = (f_1, \ldots, f_n)$ in the $n$ variables $\mathbf{x} = (x_1, \ldots, x_n)$ with generic coefficients, finding all its isolated complex solutions is a fundamental problem in computational mathematics. If we write the $i$-th polynomial in $F$ as $f_i = \sum_{\mathbf{a} \in S_i} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}$ using the multi-index notation as before, we could consider the homotopy $H = (h_1, \ldots, h_n)$ given by

$$h_i(\mathbf{x}, t) = \sum_{\mathbf{a} \in S_i} c_{i,\mathbf{a}} \mathbf{x}^{\mathbf{a}} \, t^{\omega_1(\mathbf{a})} \quad \text{for } i = 1, \ldots, n \tag{3}$$

where $\omega_i : S_i \to \mathbb{R}$ are functions with generic images that play the same roles as the lifting functions in Problem 2. Clearly, $H$ is continuous in $\mathbf{x}$ and $t$ for $t > 0$, and $H(\mathbf{x}, 1) = F(\mathbf{x})$. Moreover as $t$ varies between 0 and 1, the isolated solutions of $H(\mathbf{x}, t) = \mathbf{0}$ in $(\mathbb{C}^*)^n$ also move smoothly and form solution paths reaching the solutions of $F(\mathbf{x}) = H(\mathbf{x}, 1) = \mathbf{0}$ at $t = 1$. The end points of these paths include *all* solutions of the original system $F(\mathbf{x}) = \mathbf{0}$ in $(\mathbb{C}^*)^n$. Numerical *continuation method* can therefore be applied to trace these paths to reach the solutions if their starting points are known.

The difficulty, however, is that the starting points of these solution paths cannot be identified directly since at $t = 0$, $H(\mathbf{x}, 0)$ becomes identically zero. An ingenious observation in [4] is that the tropical intersection points are exactly the right tool to resolve this difficulty: Under the genericity assumption of

the functions $\omega_i$, all solution paths escape $(\mathbb{C}^*)^n$ as $t \to 0$, and the asymptotic behavior of each path is characterized by $(y_1\, t^{\alpha_1}, \ldots, y_n\, t^{\alpha_n})$ for some $\mathbf{y} = (y_1, \ldots, y_n) \in (\mathbb{C}^*)^n$ and a tropical intersection point $\alpha = (\alpha_1, \ldots, \alpha_n)$ in Problem 1. Finding the tropical intersection points is therefore the key step in bootstrapping the polyhedral homotopy construction.

## 4 Underlying theory

To briefly outline the underlying theory behind libtropicon, we shall focus on the formulation given in Problem 3. At a solution $\mathbf{x}$ to the system $A\mathbf{x} \geq \mathbf{c}$ with $\operatorname{rank} A = N = n + 1$, a row $\mathbf{a}$ of $A$ is said to be *active* if $\langle \mathbf{a}, \mathbf{x} \rangle = c_k$ where $c_k$ is the corresponding entry in $\mathbf{c}$. If there are $N$ linear independent active rows, $\mathbf{x}$ is known as a *basic feasible solution*.

A *level-k basic feasible solution* is simply a basic feasible solution of the combined system

$$\check{A}_{i_j} \begin{bmatrix} \alpha \\ h_{i_j} \end{bmatrix} \geq \mathbf{c}_{i_j} \quad \text{with at least } t_{i_j} \text{ equalities hold for each } j = 1, \ldots, k \tag{4}$$

for a subset $\{i_1, \ldots, i_k\}$ of the indices[1] $1, \ldots, r$.

### 4.1 The intersection-elimination-pivot scheme

A series of successful software packages [1, 6, 10, 11] for computing intersections of tropical hypersurfaces share a common basic incremental scheme that, in hindsight, could be described as an "intersection-elimination-pivot" scheme. This scheme starts with all the basic feasible solutions of $\check{A}_i\mathbf{x} \geq \mathbf{c}_i$ for certain $i \in \{1, \ldots, n\}$ and attempt to extend each into level-$k$ extended basic feasible solutions for increasingly higher values of $k$ until reaching all the level-$r$ extended basic feasible solutions.

This scheme consists of several complicated algorithms. The organization and detail of each algorithm will be outside the scope of this extended abstract. We only outline the general mathematical problem behind each step and highlight some of the new improvements over existing implementations. We refer to [1] for the complex organization of these steps.

**Intersection** Using the information from two level-1 basic feasible solutions of $\check{A}_i\mathbf{x} \geq \mathbf{c}_i$ and $\check{A}_j\mathbf{y} \geq \mathbf{c}_j$ respectively, the intersection step seeks to construct a point $\alpha \in \mathbb{R}^n$ together with $h_i$ and $h_j$ such that

$$\check{A}_i \begin{bmatrix} \alpha \\ h_i \end{bmatrix} \geq \mathbf{c}_i \qquad \text{and} \qquad \check{A}_j \begin{bmatrix} \alpha \\ h_j \end{bmatrix} \geq \mathbf{c}_j. \tag{5}$$

[1] Note that the level-$k$ basic feasible solution defined here is but a simplified prototype of the much more technical concept of "level-$k$ subfaces" that is actually used in family of algorithms [1, 2, 6, 10] whence libtropicon inherits much of the core ideas.

This step also generalizes to the problem of finding a level-$(k_1+k_2)$ basic feasible solution using a level-$k_1$ and level-$k_2$ basic feasible solutions. This question can be loosely interpreted as a local version of the tropical intersection problem.

**Elimination** Using information produced from the previous step, the goal of the elimination step is to eliminate the possibility of certain level-$k$ basic feasible solutions with minimum computational cost. This step closely resembles some of the key ideas in integer programming.

The *relation table* proposed in [2] is one of the first data structure designed for fast elimination. This simple idea has been proved to be extremely effective. It also sparked a series of related works on this idea. The technique developed in [11] is a major improvement on the relation table based method. libtropicon adopts a far generalization of this general idea to drastically improve the effectiveness of the elimination step with little computational cost. We will simply state the main theorem behind this idea in Section 4.2.

**Pivot** Finally the pivot step walks from one level-$k$ basic feasible solution to another. For a fixed $k$, the level-$k$ basic feasible solutions are organized into a (not necessarily connected) graph. The pivot step is therefore constructed as a graph walking algorithm.

### 4.2 The conic elimination method

Compare to other existing implementations, one distinguishing feature of libtropicon is the adoption of the "conic elimination method" which brings substantial improvement in the effectiveness of the elimination step.

Conic elimination is a series of tests that can quickly eliminate many candidates for level-$(k+1)$ basic feasible solutions using only the local geometric information encoded in a level-$k$ basic feasible solution. For brevity, we only state one version of such conic elimination tests. Recall that in linear programming, the set of indices of active constraints at a basic feasible solution is called *basic indices*, and the sub-matrix formed by active constraints is the *basic matrix*. Naturally, the inverse of the basic matrix is known as the *basic inverse*.

**Theorem 2.** *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be two level-1 basic feasible solutions to* $\check{A}_i\mathbf{x} \geq \mathbf{c}_i$ *and* $\check{A}_j\mathbf{y} \geq \mathbf{c}_j$ *respectively. Let* $\Delta_{\mathbf{x}}$ *and* $\Delta_{\mathbf{y}}$ *be the set of basic indices at* $\mathbf{x}$ *and* $\mathbf{y}$ *respectively. Also let* $D_i$ *and* $D_j$ *be the corresponding basic inverse matrix with columns of* $D_j$ *denoted by* $\mathbf{d}_\ell$ *for* $\ell \in \Delta_j$*. For two sets* $F_i \subseteq \Delta_{\mathbf{x}}$ *and* $F_j \subseteq \Delta_{\mathbf{y}}$*, if there exists some* $b_1 \in F_i$ *and* $b_2 \in \Delta_{\mathbf{x}}$ *such that*

$$\begin{aligned} \langle \mathbf{d}_\ell, \check{\mathbf{a}}_{b_2} - \check{\mathbf{a}}_{b_1} \rangle &\leq 0 \qquad \text{for each } \ell \in \Delta_{\mathbf{y}} \setminus F_j \\ \langle \mathbf{y} - \mathbf{x}, \check{\mathbf{a}}_{b_2} - \check{\mathbf{a}}_{b_1} \rangle &< 0 \end{aligned}$$

*then the constraints indexed by* $F_i$ *and* $F_j$ *cannot belong to the same level-2 basic feasible solutions.*

This test can be easily extended to be used for eliminating general level-$k$ basic feasible solutions.

# 5 Technical contribution

## 5.1 Data cross-feeding

One notable technical feature of libtropicon, compared to previous implementations, is that data from different algorithms are shared and reused. In particular, the data by-product of the "intersection" step are selectively saved and used in the elimination step and vice versa. Memory occupied by data that are no longer needed can also be detected and freed immediately resulting in much more efficient memory usage.

## 5.2 Parallelization on shared-memory architectures via task graphs

Today, parallel computation is an integral part of any high performance software for numerical computation. Parallel computation on shared-memory architectures such as modern multi-core systems are directly supported via a task-based model (independent calculations are organized into "tasks" that can be scheduled to run in parallel). Task-based models are generally considered to be much more scalable and flexible than thread-based models. While libtropicon inherited the basic "task pool" framework adopted in Hom4PS-3[1], this framework is refined using the more flexible "task graphs" which organize the tasks into a dependency graph. This change allows libtropicon to be scaled to systems with more processor cores.

## 5.3 Low latency GPU implementations

An increasingly important trend in high performance computing is the use of GPU (graphics processing units) devices in general purpose computing tasks. GPU based parallel computation is also supported by libtropicon through NVidia's CUDA and AMD's ROCm OpenCL framework[2]. In the previous preliminary GPU-based implementation [1] the data structure that completely describes the level-$k$ basic feasible solutions are transferred back and forth between CPU and GPU devices. Such data structures contains several $N \times N$ matrices in double precision floating point numbers. Therefore, in hindsight, this is clearly the main cause of the rather high latency. In libtropicon, only the very short *hash keys* (256 bits by default) that identifies the basic feasible solutions are passed between CPU and GPU devices while the actual data remains in the GPU devices. This change greatly reduced the CPU-to-GPU latency.

[2] ROCm is AMD's latest implementation of the OpenCL standard, an open standard for general purpose GPU computation. Currently, only ROCm have been tested. Support for other implementations of OpenCL could be added in the future with minimum changes to the code.

## References


1. T. Chen, T. L. Lee, and T. Y. Li. Mixed cell computation in Hom4PS-3. *Journal of Symbolic Computation*, 79:516–534, mar 2017.
2. T. Gao, T. Y. Li, and M. Wu. Algorithm 846. *ACM Transactions on Mathematical Software*, 31(4):555–560, 2005.
3. B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. *Mathematics of Computation*, 64(212):1541–1555, 1995.
4. B. Huber and B. Sturmfels. Bernsteins theorem in affine space. *Discrete & Computational Geometry*, 17(2):137–141, 1997.
5. A. N. Jensen. A presentation of the Gfan software. In A. Iglesias and N. Takayama, editors, *Mathematical Software - ICMS 2006*, number 4151 in Lecture Notes in Computer Science, pages 222–224. Springer Berlin Heidelberg, jan 2006.
6. T. L. Lee, T. Y. Li, and C. H. Tsai. HOM4PS-2.0: A software package for solving polynomial systems by the polyhedral homotopy continuation method. *Computing (Vienna/New York)*, 83(2-3):109–133, 2008.
7. T. Li and X. Wang. The BKK root count in $\mathbb{C}^n$. *Mathematics of Computation of the American Mathematical Society*, 65(216):1477–1484, 1996.
8. D. Maclagan and B. Sturmfels. *Introduction to tropical geometry*, volume 161. American Mathematical Soc., 2015.
9. G. Malajovich. Computing Mixed Volume and All Mixed Cells in Quermassintegral Time. *Foundations of Computational Mathematics*, pages 1–42, may 2016.
10. T. Mizutani and A. Takeda. DEMiCs: A Software Package for Computing the Mixed Volume Via Dynamic Enumeration of all Mixed Cells. In M. Stillman, J. Verschelde, and N. Takayama, editors, *Software for Algebraic Geometry*, number 148 in The IMA Volumes in Mathematics and its Applications, pages 59–79. Springer, jan 2008.
11. T. Mizutani, A. Takeda, and M. Kojima. Dynamic enumeration of all mixed cells. *Discrete & Computational Geometry*, 37(3):351–367, mar 2007.
12. J. M. Rojas and X. Wang. Counting affine roots of polynomial systems via pointed newton polytopes. *Journal of Complexity*, 12(2):116–133, jun 1996.