Maximum Consensus Floating Point Solutions for Infeasible Low-Dimensional Linear Programs with Convex Hull as the Intermediate Representation

MRIDUL AANJANEYA, Rutgers University, USA
SANTOSH NAGARAKATTE, Rutgers University, USA

This paper proposes a novel method to efficiently solve *infeasible* low-dimensional linear programs (LDLPs) with billions of constraints and a small number of unknown variables, where all the constraints cannot be satisfied simultaneously. We focus on infeasible linear programs generated in the RLibm project for creating correctly rounded math libraries. Specifically, we are interested in generating a floating point solution that satisfies the maximum number of constraints. None of the existing methods can solve such large linear programs while producing floating point solutions.

We observe that the convex hull can serve as an intermediate representation (IR) for solving infeasible LDLPs using the geometric duality between linear programs and convex hulls. Specifically, some of the constraints that correspond to points on the convex hull are precisely those constraints that make the linear program infeasible. Our key idea is to split the entire set of constraints into two subsets using the convex hull IR: (a) a set $\mathcal{X}$ of feasible constraints and (b) a superset $\mathcal{V}$ of infeasible constraints. Using the special structure of the RLibm constraints and the presence of a method to check whether a system is feasible or not, we identify a superset of infeasible constraints by computing the convex hull in 2-dimensions. Subsequently, we identify the key constraints (*i.e.*, basis constraints) in the set of feasible constraints $\mathcal{X}$ and use them to create a new linear program whose solution identifies the maximum set of constraints satisfiable in $\mathcal{V}$ while satisfying all the constraints in $\mathcal{X}$. This new solver enabled us to improve the performance of the resulting RLibm polynomials while solving the corresponding linear programs significantly faster.

CCS Concepts: • **Mathematics of computing → Solvers**.

Additional Key Words and Phrases: RLIBM, infeasible linear programs, convex hull, math libraries

ACM Reference Format:
Mridul Aanjaneya and Santosh Nagarakatte. 2024. Maximum Consensus Floating Point Solutions for Infeasible Low-Dimensional Linear Programs with Convex Hull as the Intermediate Representation. *Proc. ACM Program. Lang.* 8, PLDI, Article 197 (June 2024), 25 pages. https://doi.org/10.1145/3656427

1 INTRODUCTION

Linear programs (LPs) arise in many domains, such as robotics [13, 47], databases [10], machine learning [41, 48, 53, 61], computer graphics [32], etc. They are also widely used in the programming languages community for analyzing vulnerabilities in C source code [26], designing correctly rounded math libraries [2, 3, 37, 38, 40], repair of deep neural networks [52, 55], Presburger arithmetic for polyhedral compilation [46], and many other problems. A linear program is called *feasible* if all the constraints can be satisfied simultaneously, otherwise, it is called *infeasible*. Given the widespread use of linear programs, numerous seminal algorithms and methods have

Authors' addresses: Mridul Aanjaneya, Rutgers University, Piscataway, USA, mridul.aanjaneya@rutgers.edu; Santosh Nagarakatte, Rutgers University, Piscataway, USA, santosh.nagarakatte@cs.rutgers.edu.

ACM 2475-1421/2024/6-ART197
https://doi.org/10.1145/3656427

been developed to solve linear programs with real values [59]. Modern solvers can easily solve linear programs with several thousand constraints. However, linear programs that have billions of constraints and a large number of variables are beyond the capabilities of modern LP solvers.

Low-dimensional linear programs. A class of linear programs known as *low-dimensional linear programs* (LDLPs) have a large number of constraints in comparison to the number of variables. Low-dimensional linear programs arise in the design of correctly-rounded math libraries [2, 37, 38, 40], fast training of support vector machines (SVMs) for regression and classification in machine learning [7–9], and parametric model fitting for 3D reconstruction and inverse procedural modeling [27, 36, 57]. Meggido [42], Clarkson [20], and Seidel [51] have designed seminal effective algorithms for feasible low-dimensional linear programs with real values. However, generating solutions with floating-point coefficients is still a challenge.

Maximum consensus solutions. In many domains, it is necessary to find a solution that satisfies the maximum number of constraints of an infeasible linear program - this solution is known as the *maximum consensus solution* [15, 35, 62]. For infeasible linear programs, the problem of computing the solution that satisfies the maximum number of constraints is known to be NP-hard [49] even when solutions are explored with real values. The problem of finding solutions with a machine-supported representation, such as the floating point (FP) representation, is even more challenging. In the context of LDLPs with billions of constraints, existing algorithms do not identify maximum consensus solutions for infeasible linear programs.

LDLPs in the RLibm project. In this paper, we focus on LDLPs in the context of the RLibm project [37, 40], which computes polynomial approximations for elementary functions that produce the correctly rounded result for all floating point inputs. The RLibm project makes a case for approximating the correctly rounded result instead of the real value, the key insight being that there is an interval of real values around the correctly rounded result such that any value in this interval rounds to the correct result (see Figure 1). This interval bounds the results of the polynomial approximation being generated. The RLibm project uses these intervals for every input to create a linear program and generate polynomial approximations of a particular degree by solving them. This linear program is an LDLP because there can be a few billion constraints corresponding to inputs in the 32-bit FP representation but the number of variables (which are the coefficients of the polynomial approximation) are small. The resulting linear programs in the RLibm project are infeasible linear programs for a significant number of elementary functions. Further, the RLibm project attempts to produce a single polynomial approximation that produces correct results for multiple representations and rounding modes by approximating the round-to-odd result [40], which produces more constrained intervals and is also a reason behind infeasible linear programs.

In the presence of infeasible linear programs, the RLibm project attempts to generate a solution that satisfies the majority of constraints. Each constraint that is not satisfied by the obtained solution is added as a special case using branch statements (*i.e.*, biased branches with `__builtin_expect` intrinsics). These special cases reduce the performance of the resulting math libraries.

This paper. We propose a novel method to produce FP solutions that satisfy the maximum number of constraints in an infeasible low-dimensional linear program with a large number of constraints. To solve this problem, our high-level strategy is to split the entire universe of constraints ($\mathcal{A}$) into two sets: (1) a set of feasible constraints ($\mathcal{X}$) and (2) a small superset of infeasible constraints ($\mathcal{V}$). Given a system of LDLP constraints, we have a method to check if the system is feasible or not. We use the adaptations of the Clarkson's method from the RLibm project [2, 20] to check if the system is a feasible LDLP.

Once, we generate these two sets, we can borrow ideas from the *maximum consensus formulations* from the computer vision community [34] to create another linear program for the constraints in the second set (*i.e.*, $\mathcal{V}$) with slack variables, such that the solution to the modified linear program

satisfies all constraints in the first set (*i.e.*, $\mathcal{X}$) and the slack variables allow some of the additional constraints in the second set to be satisfied. In contrast to our approach, the prior maximum consensus formulation [34] cannot be directly applied to the entire set of constraints because it destroys the low-dimensional nature of the linear program. The resulting linear program that has billions of constraints without the low-dimensional nature cannot be solved by any existing solver.

Convex hull as the intermediate representation. To split the set of constraints into the two sets described above, our idea is to use the convex hull as the intermediate representation (IR). Specifically, we exploit the *geometric duality* [14, 30] between linear programs and convex hulls and observe that some of the constraints that correspond to points on the convex hull are precisely those constraints that make the low-dimensional linear program infeasible. We compute the convex hull $\mathcal{V}$ of the entire system of constraints. Then, we split $\mathcal{A}$ into $\mathcal{V}$ and $\mathcal{X} = \mathcal{A} - \mathcal{V}$. We use the RLibm's adaptation of the Clarkson's method to check if $\mathcal{X}$ is a feasible LDLP. If so, we have split the entire set of constraints $\mathcal{A}$ into a feasible LDLP $\mathcal{X}$ and a superset of violated constraints $\mathcal{V}$. Otherwise, we iteratively compute the convex hull of the set $\mathcal{X}$ and the new set of violated constraints $\mathcal{V}'$. If the set $\mathcal{X} - \mathcal{V}'$ is a feasible LDLP, then we have found our partition and $\mathcal{V} \cup \mathcal{V}'$ is the superset of the infeasible constraints. The above procedure continues until we end up with a feasible LDLP. This procedure terminates as $\mathcal{A}$ is finite.

Exactly computing the convex hull of n points that are embedded in the k-dimensional Euclidean space $\mathbb{R}^k$ has an asymptotic run-time complexity of $O(n^k)$ [22], which makes it intractable when n is in the order of billions. To address this issue, we observe that the RLibm constraints have a special structure and utilize it to make the problem tractable. The RLibm linear constraint for an input x is of the form, $l \leq C_0 + C_1x + C_2x^2 + C_3x^3 + .. + C_kx^k \leq h$, where C_i's are unknowns and l/h represents lower/upper bounds of the rounding interval (see Section 2.1). Using geometric duality transformations (see Section 2.3), these constraints represent the points, $(x, x^2, x^3, ..., x^d, l)$ or $(x, x^2, x^3, ..., x^d, h)$, whose convex hull is being computed in our approach. We observe that these points are intrinsically 2-dimensional because the points $(x, x^2, x^3, ..., x^d)$ are parameterized by a single parameter x and represent a 1-dimensional curve in high dimensions. Further, l (or h) indicates the freedom to satisfy the constraint. Hence, we propose to project points in $\mathbb{R}^k$ to a 2-dimensional space (*i.e.*, $\mathbb{R}^2$) and compute the convex hull in $\mathbb{R}^2$. The complexity of computing the convex hull of n points in $\mathbb{R}^2$ with our projections is $O(kn \log F)$, where F is the number of points on the convex hull [33]. This makes our approach extremely fast in practice and can still handle billions of constraints. Theoretically, we have to compute the convex hull with 2-D projections iteratively to identify the superset of infeasible constraints in $\mathcal{A}$. Empirically, we found that computing the convex hull with a 2-D projection identified a small superset of the infeasible constraints in $\mathcal{A}$ in a single iteration.

Our prototype can solve infeasible linear programs with billions of constraints significantly faster while producing the best solutions. Using the solutions from our solver, we have also improved the performance of correctly rounded RLibm's math libraries.

2 BACKGROUND ON LINEAR PROGRAMS IN THE RLIBM PROJECT

We provide background on our RLibm project, the linear programs generated with the RLibm approach, and the notion of geometric duality between convex hulls and linear programs that will be necessary to understand our solution for infeasible linear programs.

2.1 The RLibm Project

The RLibm project proposes a new method to build correctly rounded math libraries by generating polynomial approximations that approximate the correctly rounded result rather than the real value

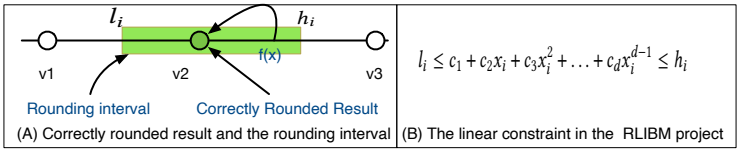


Fig. 1. (A) The rounding interval of a correctly rounded result v2. (B) The linear constraint generated to produce a correctly rounded result for input x_i where the interval $[l_i, h_i]$ is 1 ULP (units in the last place).

of an elementary function [2, 3, 37, 38, 40]. Typically, the implementation of a correctly rounded function for a 32-bit float representation uses the 64-bit double precision representation internally. Given a correctly rounded result for a 32-bit float input, there is an interval of values in the double precision representation around the correctly rounded result such that any value in that interval rounds to the correctly rounded value (see Figure 1(A)), which is called the *rounding interval*. The rounding interval for an input x_i is represented as $[l_i, h_i]$, where l_i is the lower bound and h_i is the upper bound. When the goal is to generate a polynomial of degree $d-1$ with d terms, the rounding interval specifies the linear constraint shown in Figure 1(B) on the result of the polynomial for input x_i. The RLIBM project generates a system of linear inequalities corresponding to all 32-bit inputs and their corresponding rounding intervals. Then, the goal is to identify the coefficients (*i.e.*, c_i's) of a polynomial of a particular degree that satisfies these inequalities. The RLIBM project uses an LP solver to solve this system of inequalities.

Before generating such polynomial approximations, it is necessary to reduce the original input from the domain of a 32-bit float representation (*i.e.*, $[2^{-150}, 2^{128})$] to a small domain (*e.g.*, $[0, 1]$). This step is called range reduction. The original input x is range reduced to x'. Subsequently, the polynomial that we wish to generate approximates the result for x', which is then output compensated to compute the final output for x. The RLIBM project uses specific range reduction methods that ensure that the reduced inputs are positive, which we leverage in our approach.

The RLIBM project has also proposed a method to generate a single polynomial approximation that can produce correctly rounded results for all FP representations up to n-bits [40]. The key idea is to generate a polynomial that produces correctly rounded results for the $(n+2)$-bit representation (which has 2 additional precision bits compared to the n-bit representation) using the *round-to-odd* rounding mode. When that result is double-rounded to any representation with less than n-bits, it produces correct results for the target representation. The system of linear constraints generated to produce correctly rounded results for the $(n+2)$-bit representation with the round-to-odd mode for all inputs also makes them infeasible in many cases.

2.2 Solving Low-Dimensional Linear Programs

The trajectory of our RLIBM project has been shaped by the ability to solve linear programs with billions of constraints. Initially the RLIBM project used piecewise polynomials because there were no publicly available solvers that could handle such a large number of constraints. Another challenging issue is to generate floating-point (FP) solutions because a real-valued solution from an LP solver when rounded to the FP representation may not satisfy all the constraints.

Given a reduced input x_i and its rounding interval $[l_i, h_i]$, the linear constraint generated to create a polynomial of degree $d-1$ with d terms is shown in Figure 1(B). We will canonicalize the above constraint in the standard LP format that has just one inequality in each constraint. Putting all the unknown polynomial coefficients in a column vector $\boldsymbol{\theta}$, the constraint in Figure 1(B) can be re-written in the following equivalent form:

$$\begin{aligned} \boldsymbol{a_i^T}\boldsymbol{\theta} - h_i &\leq 0 \\ \boldsymbol{a_{i+1}^T}\boldsymbol{\theta} + l_i &\leq 0 \end{aligned}$$

where the row vector $\boldsymbol{a_i^T} = [1, x_i, x_i^2 \ldots, x_i^{d-1}]$ and $\boldsymbol{a_{i+1}^T} = [-1, -x_1, -x_i^2 \ldots, -x_i^{d-1}]$.

There are two inequalities for each RLibm constraint after canonicalization. The number of unknown variables (*i.e.*, c_i's) is also known as the *dimension* of the linear system.

Low-dimensional linear program. When the dimension k of θ is much smaller than the number of constraints, then the linear program is called a *low-dimensional linear program* (LDLP). There is a large amount of redundancy in LDLPs, in the sense that a few key constraints determine the solution for all the other constraints [19].

Solving feasible low-dimensional linear programs. The seminal results from Meggido [42] and Clarkson [20] describe algorithms to solve large feasible LDLPs. If the LDLP is feasible, then the small dimension property can be used to design a fast randomized algorithm [20] for computing the solution with real values. The RLibm project uses Clarkson's method [20] to solve feasible LDLPs with floating-point solutions [2]. The essence of this algorithm is to employ *weighted* random sampling [23] to choose a small subset of $6k^2$ constraints $\mathcal{S}$ from the entire set of constraints $\mathcal{A}$. It associates a weight with each constraint. Initially the weight is 1 for all constraints. The algorithm is as follows:

- **Step 1.** Sample $\mathcal{S}$ constraints from $\mathcal{A}$ with weighted random sampling where $|\mathcal{S}| = 6k^2$, where k is the number of terms in the polynomial.
- **Step 2**: Solve the sample $\mathcal{S}$ optimally using an LP solver to compute the sample solution $\mathbf{x}^\star$.
- **Step 3**: Exit if $\mathbf{x}^\star$ satisfies all constraints in $\mathcal{A}$, otherwise check how many constraints of $\mathcal{A}$ are not satisfied by $\mathbf{x}^\star$. If the sum of the weights of the violated constraints is less than or equal to $1/(3k-1)$ fraction of the combined sum of the weights of constraints satisfied by $\mathbf{x}^\star$, then discard this sample and go to Step 1. Otherwise, double the weights of all violated constraints and then go to Step 1.

These three steps are repeated until a solution $\mathbf{x}^\star$ satisfies all constraints in $\mathcal{A}$. Clarkson [20] provides seminal results on the convergence of this procedure. Specifically, with probability at least $1/2$, $\mathbf{x}^\star$ can only violate $1/3k$ constraints in $\mathcal{A}$. In other words, the above procedure converges to a solution that satisfies all the constraints in $\mathcal{A}$ in $6k \log n$ iterations in expectation.

To solve feasible LDLPs with the above algorithm, the RLibm project uses the SOPLEX solver to generate the solution for the sample. SOPLEX solves the sample with real values (*i.e.*, with GMP rational numbers). When the sample solution is rounded to floating point coefficients, some constraints in the sample may not be satisfied. In such cases, the RLibm project restricts the intervals to check if the sample can still be solved. The need to generate a floating point solution can also turn the feasible LDLP into an infeasible LDLP.

What happens with infeasible LDLPs? When the LDLP is infeasible, Clarkson's algorithm [20] adapted by the RLibm project is not guaranteed to converge to a solution. Running more iterations of the above algorithm with an infeasible LDLP does not help in reducing the number of constraints violated by the sample solution. The above algorithm uses weighted random sampling and doubles the weight for the violated constraints. Although this doubling of weights for the violated constraints increases the probability of them being selected in the next sample, the entire sample becomes infeasible. In many LDLPs, it fails to generate an acceptable solution that satisfies the maximum number of constraints. In fact, the state-of-the-art approach is to run this algorithm for a sufficiently long time and choose the best result that has been generated so far.

2.3 Geometric Duality and Linear Programming

Our proposed method to solve infeasible LDLPs uses ideas from computational geometry, specifically the geometric duality transformations between points and lines. We provide a brief background on it. We refer the reader to excellent books on this subject for detailed information [11, 30].

Geometric duality transformations. For simplicity of exposition, consider the 2-D case first. A geometric duality $d : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is a transformation that maps a set of lines to a set of points and vice versa, in a one-to-one manner. Let $p = (a, b)$ be a 2-D point, then the dual of p is denoted by $p^\star$ and is defined as the line $y = ax - b$. The dual of a line $y = mx + c$ is defined as the point $(m, -c)$. Duality transformations have several nice properties that can be useful for analyzing geometric structures. For example, duality transforms are *incidence preserving*, i.e., a point p lies on a line l if and only if the dual point $l^\star$ lies on the dual line $p^\star$. Duality transforms are also *order preserving*, i.e., a point p lies above a line l if and only if the dual point $l^\star$ lies above the dual line $p^\star$. Geometric duality allows one to reason about lines by converting the problem into another equivalent problem about points.

For example, three lines l_1, l_2 and l_3 are concurrent in the primal space if their dual points $l_1^\star, l_2^\star$ and $l_3^\star$ are collinear in the dual space. More generally, three lines l_1, l_2 and l_3 are approximately concurrent in the primal space if their corresponding dual points $l_1^\star, l_2^\star$ and $l_3^\star$ are approximately collinear in the dual space, as shown in Figure 2. The latter problem can be efficiently solved using linear regression. A popular result in computational geometry, which our proposed approach builds upon, is that the lower envelope of an arrangement of lines (*i.e.*, the lowest partition of $\mathbb{R}^2$ that is produced by the arrangement) corresponds to the upper convex hull of the dual points [12], as shown in Figure 4(B). In $d > 2$ dimensions, geometric duality $d : \mathbb{R}^d \rightarrow \mathbb{R}^d$ maps a hyperplane h to a point $h^\star$ and vice versa, as defined below:

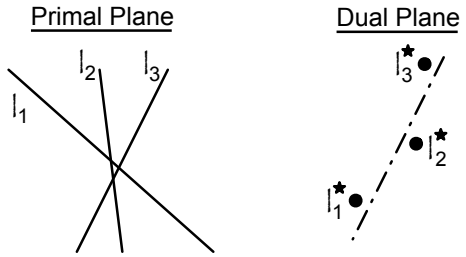


Fig. 2. Lines l_1, l_2, l_3 are approximately concurrent in the primal space if and only if the points $l_1^\star, l_2^\star, l_3^\star$ are approximately collinear in the dual space.

$$h \coloneqq a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{id}x_d - b_i = 0 \quad \Leftrightarrow \quad h^\star \coloneqq (a_{i1}, a_{i2}, \ldots, a_{id}, b_i) \tag{1}$$

Visualizing Linear Programs through the lens of Geometric Duality. For visualizing constraints in an LDLP from a geometric viewpoint, consider the constraint below:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{id}x_d \leq b_i \tag{2}$$

Equation (2) represents a *halfspace* in $\mathbb{R}^d$, *i.e.*, a partition of $\mathbb{R}^d$ into two pieces by a hyperplane that only includes one piece. If the coefficients $a_{i1}, \ldots, a_{id}$ are all positive, then equation (2) describes an upper halfspace, i.e., it is bounded from above, as shown in Figure 4(A)(b). Similarly, if the coefficients $a_{i1}, \ldots, a_{id}$ are all negative, then equation (2) describes a lower halfspace, i.e., it is bounded from below, as shown in Figure 4(A)(a). Thus, the solution space for a system of linear constraints, which is the intersection of all these halfspaces, can

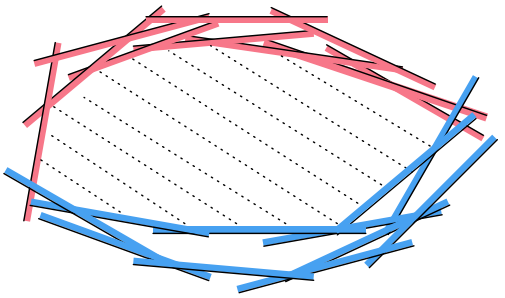
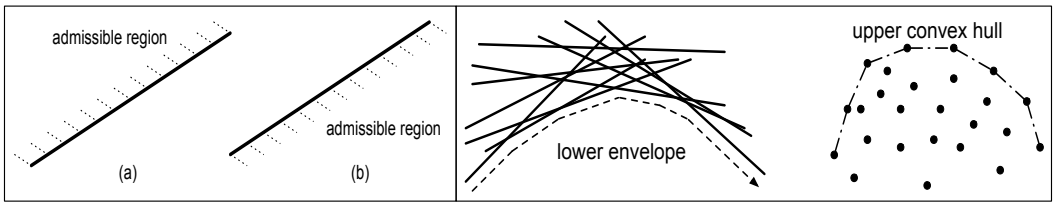


Fig. 3. The shaded region denotes the solution space for an LDLP that is bounded from above by upper halfspaces (red) and from below by lower halfspaces (blue).



(A) Interpreting constraints as half spaces (B) Mapping the lower envelope of half spaces to the upper convex hull

Fig. 4. We use geometric duality transforms to map halfspaces to points in a dual space. Then we can compute the lower envelope of the halfspaces by computing the upper convex hull of the points in the dual space.

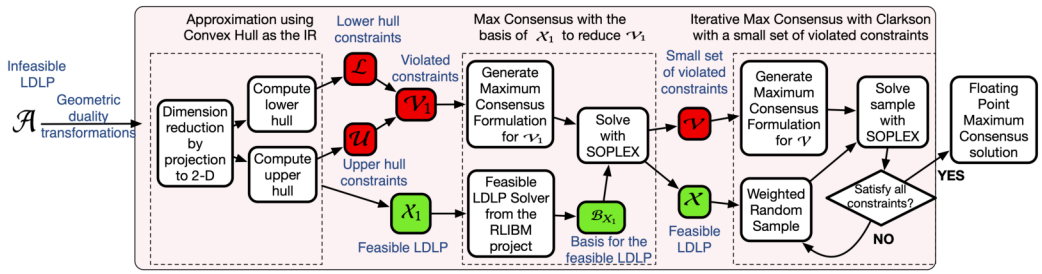


Fig. 5. Our approach to generate an FP solution for an infeasible LDLP that satisfies the maximum number of constraints. The first step is to split the entire set of constraints ($\mathcal{A}$) into two sets $\mathcal{X}_1$ and $\mathcal{V}_1$ where $\mathcal{X}_1$ is a feasible LDLP. Using the convex hull as the IR, we compute the superset of the infeasible constraints,$\mathcal{V}_1$, using geometric duality. Second, we use the basis of the solution for $\mathcal{X}_1$ (*i.e.*, $\mathcal{B}_{X_1}$) and a new linear program with slack variables that satisfies the maximum number of constraints in $\mathcal{V}_1$ to get a new feasible LDLP $\mathcal{X}$ and a small set $\mathcal{V}$. Third, we perform an iterative algorithm with weighted random sampling from $\mathcal{X}$ with the maximum consensus formulation for constraints in $\mathcal{V}$ to produce an FP solution that satisfies the maximum number of constraints in $\mathcal{A}$.

be viewed as being bounded from above by upper halfspaces, and being bounded from below by lower halfspaces, as shown in Figure 3. Constraints that border the solution space are precisely the set of key constraints that determine the solution for all the other constraints. This visualization also presents an opportunity to efficiently compute the set of constraints that border the solution space. Specifically, if we only consider the set of upper halfspaces, then their lower envelope corresponds to the set of constraints that border the solution space (see Figure 3). This lower envelope can be efficiently computed using the upper convex hull of the dual points [12], as shown in Figure 4(B). In a similar fashion, the set of lower halfspaces that border the solution space can be efficiently identified by computing the upper envelope of the lower halfspaces, which corresponds to the lower convex hull of the dual points.

Identifying infeasible constraints in an LDLP. For an infeasible LDLP, the shaded region shown in Figure 3 does not exist, or in other words, the intersection of the lower envelope of the upper halfspaces and the upper envelope of the lower halfspaces is empty. In this case, identifying the two envelopes can still provide crucial information regarding the set of constraints that make the LDLP infeasible. Specifically, constraints that border the two envelopes are the *most* conflicting constraints, and removing them is the first step towards making the LDLP feasible. If the LDLP is still infeasible, then this process can be repeated until the LDLP becomes feasible (see Figure 6).

3 AN OVERVIEW OF OUR APPROACH

Given an infeasible low-dimensional linear program with a large number of constraints (*i.e.*, billions of them) but with a small number of unknown variables, our goal is to find a floating point solution that satisfies the maximum number of constraints. Our approach is motivated by such infeasible LDLPs generated in the RLibm project to generate polynomial approximations with floating point coefficients to produce correctly rounded math libraries. This section provides an overview of our approach before going into the details in the subsequent sections (as illustrated in Figure 5).

Existing methods for infeasible LDLPs. Since we have an infeasible linear program, there does not exist a single solution that satisfies all the constraints. Our work is inspired by the maximum consensus formulations in the computer vision community where they create another feasible linear program by using slack variables whose solution satisfies the maximum number of constraints in the original linear program [34]. This approach creates four additional constraints for each original linear constraint and destroys the low-dimensional nature of the linear program. Hence, it works only with a small number of constraints typically in the order of thousands of constraints. When we tried that formulation with our infeasible LDLPs, it could not produce a solution even with real values. The huge number of constraints requires us to explore linear or near-linear time algorithms (*i.e.*, even an algorithm with a time complexity of $O(n^2)$ makes it impractical for our purposes) while preserving the low-dimensional nature of the linear program.

Insights. Our high-level idea to solve infeasible LDLPs with a large number of constraints is to partition the set of constraints into two sets: a set of constraints that is feasible and a small superset of infeasible constraints that makes the original set infeasible. This partitioning enables us to intelligently combine existing solvers for feasible LDLPs from the RLibm project [2] and the maximum consensus formulation from the vision community [34] to produce maximum consensus solutions for infeasible LDLPs with approximately billions of constraints. Figure 5 illustrates our approach to solve a large infeasible LDLP in the RLibm context. The research challenges that we have to address are the following: (1) how do we perform the above partitioning quickly? (2) how do we combine the maximum consensus formulation that handles small number of constraints with the solver for feasible LDLPs such that we can generate maximum consensus solutions for a system with billions of constraints?

3.1 Convex Hull as the IR to Create a Feasible LDLP

One of the key ideas of this paper is to use the convex hull as the intermediate representation (IR) to perform the partitioning of the constraints. Geometric duality transforms, which we described in Section 2, allow us to represent a linear constraint as a point in the dual space. Then, finding the feasible region (*i.e.*, a solution) for all the constraints can be done by identifying the convex hull in the dual space. Thus, we use the convex hull as the IR to solve large infeasible LDLPs. It also enables us to perform further optimizations and approximations that would not have been feasible otherwise, similar to how compiler IRs enable canonicalization, optimization, and approximation while compiling high-level languages.

We represent every linear constraint as a point in the dual space. Subsequently, we identify the upper hull of the points in the dual space corresponding to the constraints that specify the lower bound and identify the lower hull of the points in the dual space corresponding to the constraints that specify the upper bound. *Our insight is that the constraints that make the LDLP infeasible must lie either on the upper or the lower hull in the dual space (see Figure 6).*

Given the original set of constraints $\mathcal{A}$, we compute the lower hull $\mathcal{L}$ and the upper hull $\mathcal{U}$, and their union *i.e.*, $\mathcal{V}_1 = \mathcal{L} \cup \mathcal{U}$. Subsequently, we compute the set $\mathcal{X}_1 = \mathcal{A} - \mathcal{V}_1$. We check if the set $\mathcal{X}_1$ is a feasible LDLP using the Clarkson method adapted to the RLibm context [2]. If so, we

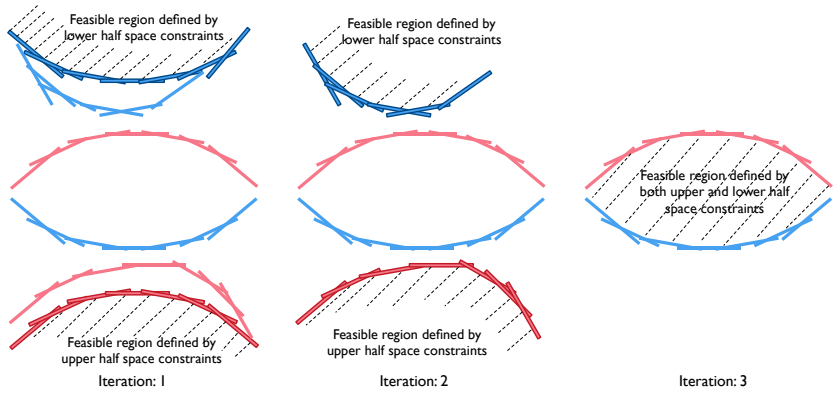


Fig. 6. Intuition on why computing the lower hull and upper hull iteratively enables the identification of infeasible constraints. Each line represents a constraint. The bordered lines represent constraints on the convex hull of the dual points. After we iteratively remove the constraints on the convex hull, we end up with a feasible LDLP.

have successfully partitioned $\mathcal{A}$ into a feasible LDLP $\mathcal{X}_1$ and a superset of infeasible constraints $\mathcal{V}_1$. Otherwise, we compute the lower hull and upper hull of $\mathcal{X}_1$ and check if removing them from $\mathcal{X}_1$ makes it feasible. We repeat this procedure iteratively until we end up with a feasible LDLP. The union of all the lower hull and upper hull constraints identified in the intermediate steps constitutes the superset of the infeasible constraints. Given that the set $\mathcal{A}$ is finite, the process terminates either when we identify a feasible LDLP or when the size of the superset of infeasible constraints identified exceeds some threshold (*e.g.*, 20,000). Figure 6 pictorially shows how removing the lower and upper hull constraints identifies the infeasible constraints and eventually results in a feasible LDLP.

Approximation using the convex hull IR. Computing the convex hull with n constraints and in k dimensions has time complexity $O(n^k)$, which makes the problem intractable when we have billions of constraints in the RLibm context. We observe that RLibm constraints have a special structure; they are intrinsically 2-dimensional because a given constraint depends on a single input x_i (which is positive) and the bounds l_i or h_i (*i.e.*, $c_0 + c_1x_i + c_2x_i^2 + .. + c_kx_i^k \leq h_i$). Hence, we propose to project points from k-dimensions to 2-dimensions and compute the convex hull in 2-dimensions. We subsequently identify the feasible LDLP using the computed 2-dimensional hulls with an iterative process that we described above. We empirically show that computing the convex hull in 2-dimensions in a single iteration is sufficient to identify a superset of infeasible constraints in the RLibm context. Our use of the convex hull as the IR and approximation over it by projecting to 2-dimensions, makes our method extremely fast in practice with time complexity $O(kn \log F)$, where F is the number of vertices on the convex hull.

3.2 Maximum Consensus using the Basis of the Feasible LDLP

After computing the feasible LDLP $\mathcal{X}_1$ and a superset of the infeasible constraints $\mathcal{V}_1$ using the convex hull, we now want to identify the maximum number of constraints in $\mathcal{V}_1$ that can be satisfied while also satisfying all the constraints in $\mathcal{X}_1$. Our high level strategy is to create a new linear program with slack variables that satisfies all the constraints in $\mathcal{X}_1$ and the maximum number of constraints in $\mathcal{V}_1$ inspired by prior maximum consensus formulations [34]. However, we cannot directly use the maximum consensus formulation because we have more than a billion constraints.

Using the maximum consensus formulation for all the billion constraints will violate the "small dimension" property (small number of variables and a large number of constraints). General linear programs with billions of constraints cannot be solved by any modern LP solver.

To address this issue, we leverage the insight that any feasible LDLP has a small set of "key" basis constraints and any solution (with real values) that satisfies these key constraints will satisfy all other constraints in the LDLP [19]. We use the feasible LDLP solver from the RLibm project to identify (an overapproximation of) the basis constraints $\mathcal{B}_{X_1}$, which has $6k^2$ constraints where k is the number of unknowns in the LDLP (*i.e.*, dimension of the LDLP).

Next, we generate a new LP formulation that uses slack variables for the constraints in $\mathcal{V}_1$ and the original constraints for the basis constraints of $\mathcal{B}_{X_1}$. The use of slack variables makes the new LP feasible. Because the number of variables in the new LP increases the number of unknowns, the new linear program is not a low-dimensional linear program. Given that the superset of the infeasible constraints computed using the convex hull has already reduced $\mathcal{V}_1$ to a few thousand constraints and we use the basis of $\mathcal{X}_1$ that just has $6k^2$ constraints, the new feasible LP in total has a few thousand constraints, which can be easily solved by modern LP solvers. Section 5 provides detailed explanation to create the maximum consensus solution using $\mathcal{B}_{X_1}$ and $\mathcal{V}_1$.

When we solve the above new linear program using any existing LP solver (*e.g.*, SOPLEX), it produces solutions with real values. Our goal is to produce floating point maximum consensus solutions. When we round the real-valued solution to floating point values, it may violate some of the constraints in $\mathcal{B}_{X_1}$ or $\mathcal{V}_1$. In those cases, we attempt to solve the system by strengthening the constraint to account for rounding errors (*i.e.*, decreasing the upper bound or increasing the lower bound). It forces the LP solver to find a solution for a much stricter constraint. When we are able to find a solution, we have identified a nearly maximum consensus solution. The middle part of Figure 5 illustrates these steps.

3.3 Combining Clarkson's Method with the Maximum Consensus Approach

The solution obtained from the above step is the *nearly-maximum consensus* solution (not the maximum consensus solution) due to FP rounding errors. The challenging issue in generating FP solutions arise when the solution from the above step satisfies $\mathcal{B}_{X_1}$ and the maximum number of constraints in $\mathcal{V}_1$ but violates another constraint in $\mathcal{X}_1$. To identify the maximum consensus solution, we split the entire set of constraints into two sets: $\mathcal{X}$ consists of constraints that are satisfied by the nearly maximum consensus solution and $\mathcal{V}$ consists of constraints violated by the nearly maximum consensus solution.

We iteratively combine Clarkson's method for feasible LDLPs in the RLibm project [2] (*i.e.*, for $\mathcal{X}$) and the maximum consensus formulation for violated constraints (*i.e.*, $\mathcal{V}$). The goal is to identify an FP solution that satisfies all the constraints in $\mathcal{X}$ and the maximum number of constraints in $\mathcal{V}$. We could have directly used this iterative method combining Clarkson's method and the maximum consensus formulation on the feasible set $\mathcal{X}_1$ and the violated set $\mathcal{V}_1$ identified using the convex hull. However, each iteration of the Clarkson's method would have been significantly slower because there are thousands of violated constraints in $\mathcal{V}_1$. The nearly maximum consensus solution ideally reduces the cardinality of violated constraints from thousands to a handful.

We assign weights to each constraint in $\mathcal{X}$. We sample $6k^2$ constraints from $\mathcal{X}$ and use all the constraints from $\mathcal{V}$ with the maximum consensus formulation and ask an off-the-shelf LP solver for a solution. The use of slack variables for the constraints in $\mathcal{V}$ enables us to create a feasible linear program for the sample. We validate if the sample solution satisfies all the constraints in $\mathcal{X}$. If they are violated by the sample solution, then the weights of those constraints in $\mathcal{X}$ are doubled. When all the constraints in $\mathcal{X}$ are satisfied, we have an FP solution that satisfies the maximum number of constraints. The rightmost part of Figure 5 shows the iterative loop.

4 FINDING THE SUPERSET OF INFEASIBLE CONSTRAINTS WITH THE CONVEX HULL

We make a case for using the convex hull as the intermediate representation to generate maximum consensus solutions for infeasible low-dimensional linear programs. We achieve this by using geometric duality transformations to represent a constraint as a point in the dual space [30]. Looking at the convex hull of the points in the dual space can help in the identification of feasible regions in the case of feasible LDLPs [51]. As explained in Section 2, *we observe that by visualizing constraints as halfspaces, the lower envelope of all upper halfspaces (resp. upper envelope of all lower halfspaces) includes all constraints that make the LDLP infeasible. The upper and lower envelopes can be efficiently computed by using geometric duality and computing the lower and upper hull of the dual points*. Hence, the convex hull serves as a good intermediate representation for computing maximum consensus solutions for infeasible linear programs.

Linear constraints to points in the dual space. The first step in our approach is to represent each constraint as a point in the dual space. Given a linear constraint from the RLibm project of the form $l_i \leq c_1 + c_2x_i^1 + c_3x_i^2 + c_4x_i^3 \leq h_i$, we first canonicalize it as two constraints: $c_1 + c_2x_i^1 + c_3x_i^2 + c_4x_i^3 - h_i \leq 0$ and $-c_1 - c_2x_i^1 - c_3x_i^2 - c_4x_i^3 + l_i \leq 0$. The points in the dual space for these constraints are $(1, x_i^1, x_i^2, x_i^3, h_i)$ and $(-1, -x_i^1, -x_i^2, -x_i^3, -l_i)$, respectively. Since the first coordinate is the same for all the dual points (1 for dual points of constraints that specify the upper bound and -1 for dual points of constraints that specify the lower bound), we can ignore the first coordinate, making the mapping from $\mathbb{R}^k$ to $\mathbb{R}^k$.

Next we want to compute the lower hull of the dual points corresponding to constraints that specify the upper bound (*i.e.*, $(1, x_i^1, x_i^2, x_i^3, h_i)$, which is now represented as (x_i, x_i^2, x_i^3, h_i)). Similarly, we want to compute the upper hull of the dual points corresponding to constraints that specify the lower bound (*i.e.*, $(-1, -x_i^1, -x_i^2, -x_i^3, -l_i)$, which is now represented as $(-x_i, -x_i^2, -x_i^3, -l_i)$).

Special structure of the RLibm constraints and computing the convex hull with 2-D projections. Computing the convex hull in k-dimensions has complexity $O(n^k)$, which becomes intractable when n is in order of billions. To address this issue, we leverage the special structure of the RLibm constraints and compute the convex hull in 2-dimensions. We project the point set $\mathcal{P}$ that is embedded in a high-dimensional Euclidean space $\mathbb{R}^k$ to a lower-dimensional Euclidean space $\mathbb{R}^2$. The RLibm constraints are of the form, for example, $(x, x^2, x^3, ..., x^d, l)$ or $(x, x^2, x^3, ..., x^d, h)$ where l and h are real numbers. These points are intrinsically 2-dimensional because the points $(x, x^2, x^3, ..., x^d)$ are parameterized by a single parameter x and represent a 1-dimensional curve in high dimensions.

As we show next in Theorem 1, the points $(x, x^2, x^3, ..., x^d)$ also lie on a convex manifold. Moreover, they are monotonically increasing. Hence, if we ignore the last coordinate (l, h), then all the constraints are satisfiable. Intuitively, l and h represent the freedom provided by the RLibm approach for computing the correctly rounded result. If this freedom is large, then it makes it easier for a single polynomial approximation to exist. Thus, it is only because of the last coordinate l (or h) that some constraints become infeasible. Our approach of computing the 2-dimensional hull is based on the intrinsic 2-dimensionality of the point set described above. Technically, the constraint with the highest l coordinate in any direction should be below the constraint with the lowest h coordinate in that direction for the LDLP to become feasible. This observation is our main motivation for using the 2-D convex hull for finding infeasible constraints (*i.e.*, with the extreme l and h coordinates).

We now explain the details behind this projection by first proving the following theorem:

Theorem 1. *All points of the form (x^r, x^s), where $1 \leq r < s$, lie on a convex curve.*

Proof. The points (x^r, x^s) lie on the graph of the function $y = x^{s/r}$. The curvature of this function can be computed as:

$$\frac{d^2}{dx^2}x^{s/r} = \frac{s}{r}\left(\frac{s}{r}-1\right)x^{s/r-2} = \frac{s}{r}\left(\frac{s}{r}-1\right)\frac{x^{s/r}}{x^2} \tag{3}$$

In the context of the RLibm project, x is the reduced input, which is always positive [38, 39]. Furthermore, since $s > r \geq 1$, it follows that the curvature is always positive. The positivity of the curvature implies that the function $y = x^{s/r}$ is convex. □

The convex hull of a point set $\mathcal{P}$ is the smallest convex set containing $\mathcal{P}$. However, if $\mathcal{P}$ was sampled from a convex curve, then by definition, all points in $\mathcal{P}$ lie on the convex hull. An argument similar to that made in Theorem 1 can also be made for all triplets, quadruplets, *etc.*, of powers of x, implying that they lie on a convex manifold. However, we skip this proof for simplicity of exposition as it follows directly from the structure of cyclic polytopes [1].

The above argument leads us to individually consider the pairs $(x_i, h_i), (x_i^2, h_i), \ldots, (x_i^d, h_i)$ when computing the convex hull for the dual points corresponding to constraints that specify the upper bound. In a similar fashion, we only consider the pairs $(-x_i, l_i), (-x_i^2, -l_i), \ldots, (-x_i^d, -l_i)$ when computing the convex hull for the dual points corresponding to constraints that specify the lower bound. Intuitively, this means that it is the lower and upper bounds l_i and h_i that ultimately decide whether the linear program is feasible or infeasible. Indeed, as the gap between l_i and h_i increases for all inputs, so does the likelihood of a possible solution for the linear program.

We provide a proof for the general case (*i.e.*, without the special structure of the RLibm constraints) that points identified using the convex hull of the 2-D projection maps to the convex hull of the dual points $(x_i, x_i^2, \ldots, x_i^d, h_i)$ or $(x_i, x_i^2, \ldots, x_i^d, l_i)$ in k-dimensions. For this purpose, we recall that a point p lies on the convex hull, if and only if there exists a hyperplane H that passes through p and all the other points lie only on one side of H, *i.e.*, in one halfspace defined by H [11].

Theorem 2. *Consider a point set $\mathcal{P}$ in k-dimensions and let $\mathcal{Q}$ be a 2-D point set that is computed by projecting all points in $\mathcal{P}$ to a 2-D space by choosing two of the k coordinates. Then, any point on the convex hull of $\mathcal{Q}$ maps to a point on the convex hull of $\mathcal{P}$.*

Proof. Without loss of generality, assume that the point set $\mathcal{Q}$ was computed by projecting all the k-dimensional points in $\mathcal{P}$ along the *first* two coordinates. Let $p \equiv (x_0, x_1)$ be a point on the convex hull of Q. By definition of the convex hull, there is a hyperplane H that passes through p and all other points in $\mathcal{Q}$ lie only on one side of H. Let $n \equiv (n_0, n_1)$ be the normal vector for H. The above definition implies that $n_0 \cdot x_0 + n_1 \cdot x_1 \leq n_0 \cdot y_0 + n_1 \cdot y_1$ for all other points $q \equiv (y_0, y_1)$ in $\mathcal{Q}$. If we consider the normal vector $n \equiv (n_0, n_1, 0, \ldots, 0)$ in k-dimensions, then it passes through the point $p' \equiv (x_0, x_1, x_2, \ldots, x_{k-1})$ which was projected to p. Moreover, for all points $q' \equiv (y_0, y_1, y_2, \ldots, y_{k-1})$ in $\mathcal{P}$, we still have the inequality $x_0 \cdot n_0 + x_1 \cdot n_1 + x_2 \cdot 0 + \ldots + x_{k-1} \cdot 0 \leq y_0 \cdot n_0 + y_1 \cdot n_1 + y_2 \cdot 0 + \ldots + y_{k-1} \cdot 0$ from the definition of the 2-D convex hull, i.e., the point p' lies on the convex hull of $\mathcal{P}$. □

In summary, without the structure of the RLibm constraints, points identified via the 2-D hull lie on the k-D hull, but not vice versa. The specific structure of the RLibm constraints (*i.e.*, they are inherently 2-dimensional) enables us to identify the infeasible constraints using the 2-D hull.

Identifying a superset of infeasible constraints by computing the convex hull with 2-D projections. To identify the superset of infeasible constraints, our approach is to use an iterative procedure, which we described in Section 3.1. We compute the convex hull with the 2-D projection and check if the remaining constraints after removing the constraints on the 2-D hull are feasible using the RLibm's feasible LDLP solver. This process repeats until we end up with a feasible LDLP or the total number of constraints accumulated by computing the 2-D hull iteratively exceeds the user-defined threshold. Crucially, our approach always reports a superset of infeasible constraints

by construction because we have a definitive way to check whether a system is a feasible LDLP using RLIBM's solver. We empirically show in Section 7 that the convex hull computed with the 2-D projection computes a superset of infeasible constraints in a single iteration in the RLIBM context.

Computing the convex hull in 2-dimensions with our projections can be done in $O(kn \log F)$ time using the Kirkpatrick-Seidel algorithm [33] where F is the number of vertices on the convex hull. This computation is performed with high precision arithmetic using the CGAL library [56]. With this approach, we are able to split the set of constraints $\mathcal{A}$ into a feasible set of constraints $\mathcal{X}_1$ and a superset of infeasible constraints $\mathcal{V}_1$.

5 COMPUTING NEARLY MAXIMUM CONSENSUS SOLUTIONS

Using the convex hull of the constraints in the dual space, we have partitioned the entire set of constraints (*i.e.*, $\mathcal{A}$) into a set of feasible constraints (*i.e.*, $\mathcal{X}_1$) and a superset of infeasible constraints (*i.e.*, $\mathcal{V}_1$). The set $\mathcal{V}_1$ can have a few thousand constraints. Now, the goal is to identify a solution that satisfies all the constraints in the first set and the maximum number of constraints in the second set.

Identifying the basis of the feasible LDLP $\mathcal{X}_1$. Given that the feasible set is an LDLP, there exists a small set of basis constraints (*i.e.*, $\mathcal{B}_{X_1}$ in Figure 5) whose solution satisfies all constraints in the feasible LDLP $\mathcal{X}_1$ [19]. For the feasible set of constraints, we can use the solver from the RLIBM project for feasible low-dimensional linear programs [2] to identify the basis constraints. Inspired by maximum consensus formulations from the computer vision community [34], we design a new linear program whose solution in real values satisfies all the constraints in $\mathcal{B}_{X_1}$ and the maximum number of constraints in the set $\mathcal{V}_1$. We describe our adaptation of the maximum consensus formulation in Section 5.1. The new linear program for determining maximum consensus introduces two new variables for each original constraint. Hence, it cannot be applied to all the constraints in $\mathcal{A}$ as modern LP solvers cannot solve regular LP problems (*i.e.*, number of variables are proportional to the number of constraints) with more than a few thousand constraints.

Transitioning from real valued solutions to FP solutions. The solution that we obtain by solving the basis constraints and the new LP formulation for the constraints in $\mathcal{V}_1$ is with real values. When we round the real valued solution to a floating point representation, it may fail to satisfy some constraints in $\mathcal{B}_{X_1}$. In such cases, we reduce the bound (*i.e.*, b_i) of the constraint by 1 ULP (units in the last place) in double precision (*i.e.*, the representation used for the implementation of the math library) for constraints of the form ($a_i^T \theta - b_i \leq 0$). This is analogous to shrinking of the intervals in the RLIBM project. By doing so, we ask the solver to solve a much stricter constraint. We repeat this process until we are able to solve the set $\mathcal{B}_{X_1} \cup \mathcal{V}_1$. This process terminates because $\mathcal{B}_{X_1}$ is a feasible LDLP. When the solution with floating point values satisfies all constraints in $\mathcal{B}_{X_1}$, we have a nearly maximum consensus solution.

The solution may still not be the maximum consensus solution for the entire set of constraints $\mathcal{A}$ because the rounding error introduced by rounding the real value to a FP solution may violate a few constraints in $\mathcal{A}$. The important point to note is that the number of violated constraints will significantly shrink from a few thousands to a handful by identifying the nearly maximum consensus solution. Subsequently, we employ an iterative algorithm that combines maximum consensus with Clarkson's method [20] to find the maximum consensus solution, which we describe in Section 6.

5.1 A New Linear Program for Maximum Consensus Among $\mathcal{V}_1$ and $\mathcal{B}_{X_1}$

Given the set of basis constraints $\mathcal{B}_{X_1}$ and an overapproximation of violated constraints $\mathcal{V}_1$, the set $\mathcal{B}_{X_1} \cup \mathcal{V}_1$ is an infeasible LDLP. We want to identify a solution that satisfies the maximum number of constraints in $\mathcal{V}_1$ while satisfying all constraints in $\mathcal{B}_{X_1}$. Mathematically,

$$\max_{\boldsymbol{\theta} \in \mathbb{R}^n, \mathcal{I} \subseteq \mathcal{V}_1} |\mathcal{I}|$$
$$\text{subject to} \quad \boldsymbol{a_i}^T \boldsymbol{\theta} - b_i \leq 0 \;\; \forall i \in \mathcal{I} \cup \mathcal{B}_{X_1} \tag{4}$$

This formulation, as stated above by itself, is not a linear program. The objective function is counting over discrete sets. Our goal in this section is to create an LP formulation.

Slack variables to move to a feasible LP and indicator variables to count the number of violated constraints. We want to create a new feasible LP from an infeasible LP. There are four key ideas in creating this formulation. First, we add slack variables to each constraint. Each original constraint $\boldsymbol{a_i}^T \boldsymbol{\theta} - b_i \leq 0$ now becomes $\boldsymbol{a_i}^T \boldsymbol{\theta} - b_i \leq s_i$, where $s_i \geq 0$ is a slack variable. The slack variable provides a bit more freedom to satisfy the constraint. We want the slack variable to be non-zero only when the constraint is violated, which we accomplish by adding additional constraints that we describe next. When the i^{th} constraint is violated in the original LP, we have $\boldsymbol{a_i}^T \boldsymbol{\theta} - b_i > 0$. With the slack variables, we have $\boldsymbol{a_i}^T \boldsymbol{\theta} - b_i \leq s_i$. Hence, s_i will be non-zero when the constraint is violated.

Second, we add indicator variables u_i for each constraint to indicate whether the constraint is violated or not. With the use of indicator variables, finding a solution that satisfies the maximum number of constraints boils down to minimizing the sum of the indicator variables (see equation (5)).

Third, we want the indicator variable to be 1 when the constraint is violated. Since the slack variable is non-zero when the constraint is violated, we force the indicator variable u_i to be 1 with the constraint $s_i(1 - u_i) = 0$.

Finally, we want to force the indicator variable u_i to be 0 and the slack variable s_i to be 0 when the i^{th} constraint is satisfied. We accomplish this with the constraint $u_i(s_i - \boldsymbol{a_i}^T \boldsymbol{\theta} + b_i) = 0$. When the original LP constraint is satisfied, $\boldsymbol{a_i}^T \boldsymbol{\theta} - b_i \leq 0$. Hence, $-\boldsymbol{a_i}^T \boldsymbol{\theta} + b_i \geq 0$. Given that $s_i \geq 0$, we have $s_i - \boldsymbol{a_i}^T \boldsymbol{\theta} + b_i \geq 0$, which forces u_i to be zero. The constraint $s_i(1 - u_i) = 0$ forces s_i to be zero when u_i is 0. The constraint $u_i(s_i - \boldsymbol{a_i}^T \boldsymbol{\theta} + b_i) = 0$ forces $s_i - \boldsymbol{a_i}^T \boldsymbol{\theta} + b_i = 0$ when the i^{th} constraint is violated because u_i is 1 in that case. Further, it also ensures that the slack variable s_i captures the amount by which the constraint $\boldsymbol{a_i}^T \boldsymbol{\theta} + b_i \leq 0$ is off in the original LP.

Summarizing these ideas, the formulation to identify the maximum set of constraints in $\mathcal{V}_1$ while satisfying all constraints in $\mathcal{B}_{X_1}$ is as follows:

$$\min_{\boldsymbol{u}, \boldsymbol{s} \in \mathbb{R}^M, \boldsymbol{\theta} \in \mathbb{R}^n} \sum u_i \tag{5}$$
$$\text{subject to} \quad \boldsymbol{a_i}^T \boldsymbol{\theta} - b_i \leq s_i \quad \forall i \in V_1 \tag{6}$$
$$\boldsymbol{a_j}^T \boldsymbol{\theta} - b_j \leq 0 \quad \forall j \in B_{X_1} \tag{7}$$
$$u_i(s_i - \boldsymbol{a_i}^T \boldsymbol{\theta} + b_i) = 0, \tag{8}$$
$$s_i(1 - u_i) = 0, \tag{9}$$
$$1 - u_i \geq 0, \tag{10}$$
$$s_i, u_i \geq 0. \tag{11}$$

Note that all constraints from the basis of $\mathcal{X}_1$ (*i.e.*, $\mathcal{B}_{X_1}$) are used unmodified from the original LP (see equation (7)). Slack and indicator variables are only introduced for the constraints in $\mathcal{V}_1$.

Removing non-linearity with iterative predictor and corrector steps. Equations (5)-(11) are still non-linear due to the constraints in equations (8) and (9). We use the idea from prior work [34] of removing this non-linearity by moving the non-linear constraints in equations (8)

and (9) into the objective function. These new constraints are controlled by the penalty parameter α, which forces as many constraints to be satisfied as possible.

$$\begin{aligned} \min_{\boldsymbol{u},\boldsymbol{s}\in\mathbb{R}^M,\boldsymbol{\theta}\in\mathbb{R}^n} \quad & \textstyle\sum_i u_i + \alpha\left[u_i(s_i - \boldsymbol{a_i}^T\boldsymbol{\theta} + b_i) + s_i(1-u_i)\right] & \\ \text{subject to} \quad & \boldsymbol{a_i}^T\boldsymbol{\theta} + b_i \le s_i, & \forall i \in V_1 \\ & \boldsymbol{a_j}^T\boldsymbol{\theta} - b_j \le 0, & \forall j \in B_{X_1} \\ & 1 - u_i \ge 0, & \\ & s_i, u_i \ge 0. & \end{aligned} \tag{12}$$

The above system has a non-linear objective function. The idea is to solve it in an iterative fashion with a predictor-corrector loop. In the predictor step, we fix the values of u_i, which makes them constant and the system linear. The resulting LP can be solved by an LP solver to obtain the values of $\boldsymbol{s_i}$ and $\boldsymbol{\theta}$. Subsequently in the corrector step, we use the values for $\boldsymbol{s_i}$ and $\boldsymbol{\theta}$ from the predictor step and make them constant and create a linear program and solve them for $u_i's$.

The linear program being solved by the predictor step after fixing the variables u_i with constants is shown in equation (13), which has been simplified with constant propagation for the $\boldsymbol{u}$ variables and some algebraic simplification.

$$\begin{aligned} \min_{\boldsymbol{s}\in\mathbb{R}^M,\boldsymbol{\theta}\in\mathbb{R}^n} \sum_i \quad & s_i - u_i(\boldsymbol{a_i}^T\boldsymbol{\theta} - b_i) & \\ \text{subject to} \quad & \boldsymbol{a_i}^T\boldsymbol{\theta} - b_i \le s_i, & \forall i \in V_1 \\ & \boldsymbol{a_j}^T\boldsymbol{\theta} - b_j \le 0, & \forall j \in B_{X_1} \\ & s_i \ge 0. & \end{aligned} \tag{13}$$

After obtaining the solution to equation (13), the corrector step fixes the variables $\boldsymbol{s}$ and $\boldsymbol{\theta}$ and solves for the following set of constraints in equation (14) to determine the u_i's.

$$\begin{aligned} \min_{\boldsymbol{u}\in\mathbb{R}^M} \sum_i u_i\left[1 - \alpha(\boldsymbol{a_i}^T\boldsymbol{\theta} - b_i)\right] & \\ \text{subject to} \quad 0 \le u_i \le 1. & \end{aligned} \tag{14}$$

Interestingly, this equation can be solved in closed form in a way that also ensures that $\boldsymbol{u} \in \{0,1\}^M$. Specifically, if $\left[1 - \alpha(\boldsymbol{a_i}^T\boldsymbol{\theta} - b_i)\right] \le 0$, set $u_i = 1$, else set $u_i = 0$. In the subsequent iterations, the predictor uses these values of $\boldsymbol{u}$. The α parameter, when set to a large value, will force the u_i's to be 1, unless the constraint is satisfied. The predictor will try to satisfy as many violated constraints as possible in the next iteration. We alternate between the predictor and the corrector steps until the objective function of the predictor step in equation (13) does not change across iterations.

5.2 Illustration of Maximum Consensus with $\mathcal{B}_{X_1}$ and $\mathcal{V}_1$

We illustrate our above formulation to find the nearly maximum consensus solution with a small infeasible LP with three basis and two violated RLIBM constraints. Assume that the polynomial $p(x)$ that we are generating from the RLIBM project has the form $p(x) = c_0 + c_1x + c_2x^2$. Let us say the basis constraints (*i.e.*, $\mathcal{B}_{X_1}$) corresponding to the inputs x_1, x_2, and x_3 are:

$$
\begin{aligned}
l_1 &\leq c_0 + c_1 x_1 + c_2 x_1^2 \leq h_1 \\
l_2 &\leq c_0 + c_1 x_2 + c_2 x_2^2 \leq h_2 \\
l_3 &\leq c_0 + c_1 x_3 + c_2 x_3^2 \leq h_3
\end{aligned} \tag{15}
$$

Let's say the violated constraints, $\mathcal{V}_1$, corresponding to inputs x_4 and x_5 are:

$$
\begin{aligned}
l_4 &\leq c_0 + c_1 x_4 + c_2 x_4^2 \leq h_4 \\
l_5 &\leq c_0 + c_1 x_5 + c_2 x_5^2 \leq h_5
\end{aligned} \tag{16}
$$

After canonicalization, each original basis constraint from equation (15) becomes,

$$
\begin{aligned}
c_0 + c_1 x_1 + c_2 x_1^2 - h_1 &\leq 0, \qquad & -c_0 - c_1 x_1 - c_2 x_1^2 + l_1 &\leq 0 \\
c_0 + c_1 x_2 + c_2 x_2^2 - h_2 &\leq 0, \qquad & -c_0 - c_1 x_2 - c_2 x_2^2 + l_2 &\leq 0 \\
c_0 + c_1 x_3 + c_2 x_3^2 - h_3 &\leq 0, \qquad & -c_0 - c_1 x_3 - c_2 x_3^2 + l_3 &\leq 0
\end{aligned} \tag{17}
$$

Similarly, the violated constraints from equation (16) after canonicalization become,

$$
c_0 + c_1 x_4 + c_2 x_4^2 - h_4 \leq 0, \qquad -c_0 - c_1 x_4 - c_2 x_4^2 + l_4 \leq 0 \tag{18}
$$

$$
c_0 + c_1 x_5 + c_2 x_5^2 - h_5 \leq 0, \qquad -c_0 - c_1 x_5 - c_2 x_5^2 + l_5 \leq 0 \tag{19}
$$

Our goal is to find a nearly maximum consensus solution that satisfies all the basis constraints and the maximum number of the violated constraints. Obviously, in this particular scenario, it is possible to exhaustively test whether the solution is maximal by inserting each of the violated constraints one by one and checking if the linear program is feasible. However, in practical scenarios, this is not a feasible approach when there are several billions of constraints.

We introduce slack and indicator variables for each of the constraints in the set of violated constraints. We use s_{4l} and s_{4h} to represent slack variables for the violated constraint in equation (18). Similarly, s_{5l} and s_{5h} represent slack variables used for the violated constraint in equation (19). Similarly, u_{4l}, u_{4h}, u_{5l}, and u_{5h} represent indicator variables corresponding to the violated constraints in equations (18) and (19), respectively.

Initially, we can set indicator variables (*i.e.*, u_i's) to either 0 or 1 for the predictor step when we try to find the s_i's and θ. The following linear equation from the predictor step (*i.e.*, equation (13)) attempts to find and satisfy the maximum number of violated constraints while satisfying all the basis constraints:

$$
\begin{aligned}
min \;\; & s_{4l} - u_{4l}(-c_0 - c_1 x_4 - c_2 x_4^2 - l_4) + s_{4h} - u_{4h}(c_0 + c_1 x_4 + c_2 x_4^2 - h_4) \\
& + s_{5l} - u_{5l}(-c_0 - c_1 x_5 - c_2 x_5^2 - l_5) + s_{5h} - u_{5h}(c_0 + c_1 x_5 + c_2 x_5^2 - h_5) \\
\text{subject to} \quad & -c_0 - c_1 x_4 - c_2 x_4^2 + l_4 \leq s_{4l}, \qquad c_0 + c_1 x_4 + c_2 x_4^2 - h_4 \leq s_{4h} \\
& -c_0 - c_1 x_5 - c_2 x_5^2 + l_5 \leq s_{5l}, \qquad c_0 + c_1 x_5 + c_2 x_5^2 - h_5 \leq s_{5h} \\
& c_0 + c_1 x_1 + c_2 x_1^2 - h_1 \leq 0, \qquad -c_0 - c_1 x_1 - c_2 x_1^2 + l_1 \leq 0 \\
& c_0 + c_1 x_2 + c_2 x_2^2 - h_2 \leq 0, \qquad -c_0 - c_1 x_2 - c_2 x_2^2 + l_2 \leq 0 \\
& c_0 + c_1 x_3 + c_2 x_3^2 - h_3 \leq 0, \qquad -c_0 - c_1 x_3 - c_2 x_3^2 + l_3 \leq 0 \\
& s_{4l} \geq 0, s_{4h} \geq 0, s_{5l} \geq 0, s_{5h} \geq 0
\end{aligned} \tag{20}
$$

Note that basis constraints are used without any slack or indicator variables in equation (20). The introduction of the slack variables $s_{4l}, s_{4h}, s_{5l}, s_{5h}$ ensures that the constraints $[l_4, h_4]$ and $[l_5, h_5]$ may not be exactly satisfied by the computed solution at either the lower or the upper bound, while ensuring that the linear program defined in equation (20) still computes a non-trivial solution, i.e., some of the slack variables may be non-zero. This is in stark contrast to prior solvers for feasible LPs [2], which will simply fail to find a solution if all the constraints cannot be exactly satisfied.

The corrector stage uses the solution from the LP problem above and computes u_{4l}, u_{4h}, u_{5l}, and u_{5h} using equation (14). We set α to a large pre-determined value to provide higher penalty to violated constraints. The process is now repeated with the corrected u values until the objective function in equation (13) does not change or the user-specified number of constraints are satisfied.

6 ITERATIVELY FINDING THE MAXIMUM CONSENSUS FP SOLUTION

The new linear program for maximum consensus with the basis and an overapproximation of the violated constraints is solved with real values and the solution is rounded to floating point values. The floating point solution satisfies all the basis constraints and the maximum number of violated constraints. However, due to rounding errors, the floating point solution may violate some constraints in $\mathcal{X}_1$, the feasible LP. There is also some bias introduced by the fixed basis $\mathcal{B}_{\mathcal{X}_1}$ because the basis computation was decoupled from the set of violated constraints $\mathcal{V}_1$. Hence, the solution from the formulation presented in Section 5 is a solution that is close to the maximum consensus solution but not the maximum consensus solution itself. Using the above solution, we partition the entire set of constraints into two sets: a set of constraints $\mathcal{X}$ that is satisfied by the nearly maximum consensus solution and a set of violated constraints $\mathcal{V}$. The set of violated constraints $\mathcal{V}$ is significantly smaller than the overapproximation of the violated constraints that we started with (*i.e.*, $\mathcal{V}_1$). Next, we propose an iterative combination of the maximum consensus formulation with Clarkson's algorithm [20] for feasible LDLPs. Clarkson's method tries to remove the bias introduced by any fixed basis by iterating over many different bases for $\mathcal{X}$, in an effort to find the basis that is best suited for satisfying the maximum number of constraints in $\mathcal{V}$. Each iteration with the sets $\mathcal{X}$ and $\mathcal{V}$ is significantly faster in comparison to running our iterative algorithm with $\mathcal{X}_1$ and $\mathcal{V}_1$, which were generated by our approximation using the convex hull.

Here are the steps of our iterative algorithm, which uses Clarkson's method for feasible LDLPs from the RLIBM project and combines it with the maximum consensus formulation that we described in Section 5.1. Given the feasible LDLP $\mathcal{X}$ and a small set of violated constraints $\mathcal{V}$, our goal is to generate a solution with floating point values that satisfies the maximum number of constraints in $\mathcal{V}$ while satisfying all the constraints in $\mathcal{X}$.

- **Step 1**: We maintain weights with each constraint from $\mathcal{X}$. We initialize the weights to 1.
- **Step 2**: Now we sample $6k^2$ constraints from $\mathcal{X}$ using weighted random sampling, where k is the number of unknowns. We use the original LP for the sampled constraints. We generate the maximum consensus LP formulation for the constraints in $\mathcal{V}$. We ask the LP solver to solve the sampled constraints from $\mathcal{X}$ along with the maximum consensus constraints of the form in equation (13) for those in $\mathcal{V}$. This modified linear program only has $k + 2|\mathcal{V}|$ variables, which is much smaller than the total number of constraints $|\mathcal{A}|$, thereby maintaining a low-dimensional structure.
- **Step 3**: We check how many constraints of $\mathcal{X}$ are not satisfied by the sample solution. If the sum of the weights of the constraints in $\mathcal{X}$ that are violated by the sample solution is more than $1/(3k - 1)$ fraction of the combined sum of the weights of constraints in $\mathcal{X}$ satisfied by the sample solution, then we discard this sample and go to Step 2. Otherwise, we double the weights of the constraints violated by the sample solution in $\mathcal{X}$ and go back to Step 2.

- **Step 4**: The algorithm terminates when the sample solution satisfies all constraints in $\mathcal{X}$ and the number of violated constraints does not change for a fixed number of iterations.

7 EXPERIMENTAL EVALUATION

We describe our prototype [4] for solving infeasible LDLPs, our experimental methodology for evaluating its ability to solve linear programs, and the performance of the resulting math libraries with the maximum consensus solution from our solver.

Prototype and methodology. Our solver for infeasible LDLPs takes linear programs as input and produces floating point solutions in double precision for all the unknown variables. It is written in C++ and handles linear programs with several billion constraints. The prototype uses the SOPLEX solver, which uses exact rational arithmetic to solve small linear programs internally. To evaluate our infeasible LDLP solver, we use linear programs generated from the RLibm project for generating correctly rounded elementary functions. A single polynomial approximation for an elementary function from the RLibm project produces correctly rounded results for all rounding modes and multiple representations (up to 32-bits). When we create a feasible subset of the original infeasible LDLP, we use the solver for feasible linear programs from the RLibm project [2], which implements Clarkson's method [20] to produce floating point solutions. We use the computational geometry package, CGAL [56], to compute the convex hull using our 2-D projections.

We experimented with the following LP solvers to solve the linear programs generated in the RLibm project: Gurobi, CPLEX, and SOPLEX. All these solvers failed to solve any of the the LPs from the RLibm project. For the experimental evaluation, we compare our solver with RLibm's solver for feasible linear programs [2] and our implementation of the maximum consensus solution from the computer vision community [34]. When given an infeasible linear program, RLibm's solver does not terminate. We modified it to print out the number of constraints satisfied and violated by the best solution after every iteration. We ran the RLibm's solver for 500 iterations and use the best solution produced for our evaluation. We also implemented the maximum consensus solution for small programs from the vision community for the entire LDLP [34] because there are no publicly available versions. We use the wall clock time to measure the time taken to produce a solution.

We incorporated the resulting floating point solutions generated by our solver for each elementary function into RLibm's math libraries. We validated that the resulting functions produce correctly rounded results for all representations and for all inputs. During this process, we also improved the implementation of range reduction and output compensation code in the RLibm project. We evaluate the performance benefits from these changes along with those from the maximum consensus solutions. We have committed the resulting functions to the RLibm git repository.

Checking the validity of the resulting functions can be completed within a minute. Subsequently, we measure the performance of the resulting functions. We conducted our experiments on a 2.10GHz Intel Xeon(R) Silver 4310 server with 256GB of RAM running Ubuntu 22.04.3 LTS that has both Intel turbo boost and hyper-threading disabled to minimize perturbation. We compiled our resulting math libraries with `-march=native -O3` flags. For measuring performance, we use hardware performance counters using the `perf` tool to count the number of cycles taken to compute the result for each input. We aggregate these counts for all inputs to compute the total time taken for each elementary function.

Ability to solve infeasible linear programs and produce maximum consensus solutions. Table 1 compares the ability of the three solvers: our solver, the solver from the RLibm project, and the MCS solver from the computer vision community to solve linear programs for generating correctly rounded implementations for 16 functions. The maximum consensus solver from the vision community [34] failed to produce a solution for any of the functions. The SOPLEX solver used to solve the MCS LP formulation times out without producing a solution. This is because

Table 1. This table reports whether the solvers produce the maximum consensus solution (*i.e.*, ✓) or not (*i.e.*, ✗). We report the type of LP and the number of RLIBM constraints (*i.e.*, $l_i \leq P(x_i) \leq h_i$) for a particular function. We compare our solver to the RLIBM project's solver [2] and our implementation of the maximum consensus solver from the computer vision community [34]. We report the speedup in generating a solution with our solver in comparison to RLIBM's solver when it generates an acceptable solution with less than 30 violated constraints.

function	LP type	Total RLIBM constraints	**Our solver**	RLIBM solver [2]	MCS solver [34]	Speedup over RLIBM solver
`logf`	Infeasible	7,165,657	✓	✗	✗	**73.2** ×
`log10f`	Infeasible	7,165,657	✓	✗	✗	**66.7** ×
`log2f`	Feasible	7,165,657	✓	✓	✗	**0.9** ×
`expf`	Infeasible	503,402,009	✓	✗	✗	**144.5** ×
`exp10f`	Infeasible	504,492,234	✓	✗	✗	**98.3** ×
`exp2f`	Infeasible	286,174,228	✓	✗	✗	**31.6** ×
`sinpif`	Feasible	126,086,339	✓	✓	✗	**1.1** ×
`cospif`	Feasible	48,373,323	✓	✓	✗	**1.1** ×
`sinhf`	Infeasible	256,614,352	✓	✗	✗	**2.5** ×
`coshf`	Feasible	255,080,290	✓	✓	✗	**1.2** ×
`sinf`	Infeasible	1,122,326,490	✓	✗	✗	**74.2** ×
`cosf`	Infeasible	1,045,258,544	✓	✗	✗	**113** ×
`tanf`	Infeasible	1,324,683,332	✓	✗	✗	**174** ×

that formulation destroys the low-dimensional nature of the LP problem and no solver can solve LPs with billions of unknowns. In contrast, *our solver is able to generate the best known maximum consensus solution for both infeasible and feasible LDLPs corresponding to these functions.*

For feasible LDLPs, there exists a solution that satisfies all the constraints. RLIBM's solver can solve such feasible LDLPs in similar time as our solver (*i.e.*, for `log2f`, `cospif`, `sinpif`, and `coshf`). In the case of an infeasible LDLP, there does not exist a single solution that satisfies all the constraints. Our goal is to identify a solution that satisfies the maximum number of constraints (*i.e.*, only a few violated constraints). RLIBM's solver cannot generate the solution produced by our solver within the specified number of iterations. In contrast, our solver is able generate a better solution in significantly faster time (174× faster for `tanf`). Overall, our solver is able to generate a better solution than RLIBM's solver in at least 10× shorter time for infeasible LDLPs.

Analysis of the solutions from the various stages of our solver. For each function, Table 2 reports the number of canonicalized constraints, which is 2× the number of RLIBM constraints. The number of constraints for each function is a function of range reduction. For trigonometric functions, one needs to use $\frac{1}{\pi}$ with a large number of bits to produce correctly rounded results [43], which causes each original input to result in a unique reduced input (and a constraint) after range reduction. Hence, trigonometric functions have more than 2 billion canonicalized constraints.

The third, fourth, and fifth columns of Table 2 report the number of extreme points on the convex hull that approximates the number of infeasible constraints (*i.e.*, $|\mathcal{V}_1|$ from Figure 5), the number of constraints violated by the solution generated by using the basis and the new LP for the nearly maximum consensus FP solution (*i.e.*, $|\mathcal{V}|$ from Figure 5), and the number of constraints violated by the solution with the iterative loop for maximum consensus, respectively. The use of convex hull to over-approximate the infeasible constraints reduces the number of constraints from billions to a few thousands (*i.e.*, 11,773 in the worst case with `exp10f`), which allows us to solve the new LP for maximum consensus with SOPLEX.

Table 2. This table reports the total number of canonical constraints, the number of extreme points on the convex hull, violated constraints in the nearly maximum consensus solution and our final maximum consensus solution, the number of violated constraints in the RLIBM solution, and the time taken to solve the infeasible LDLP with our approach.

function	Total canonical constraints	Convex hull extreme points	Nearly-MCS violated constraints	Final MCS violated constraints	Best RLIBM violated constraints	Total time (seconds)
`logf`	14,331,314	6,166	13	**10**	24	216
`log10f`	14,331,314	6,162	39	**8**	21	140
`log2f`	14,331,314	6,726	0	**0**	0	121
`expf`	1,006,804,018	11,714	14	**1**	23	2047
`exp10f`	1,008,984,468	11,773	4	**3**	11	1929
`exp2f`	572,348,456	14,348	1	**1**	3	636
`sinpif`	252,172,678	4,827	0	**0**	0	275
`cospif`	96,746,646	4,451	0	**0**	0	116
`sinhf`	513,228,704	6,597	2	**1**	6	319
`coshf`	510,160,580	7,452	0	**0**	0	108
`sinf`	2,244,652,980	7,113	7	**5**	11	1732
`cosf`	2,090,517,088	8,521	8	**4**	13	1141
`tanf`	2,649,366,664	11,073	11	**5**	15	2175

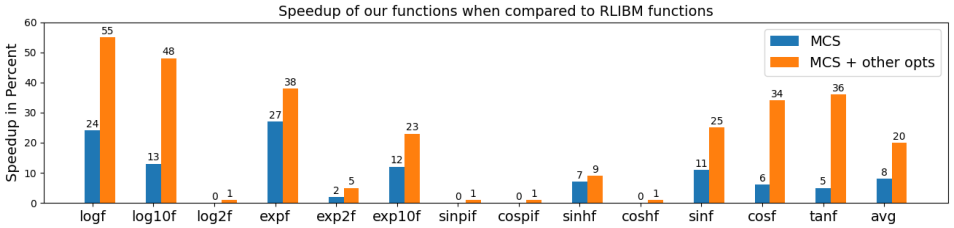


Fig. 7. Speedup of the resulting correctly rounded functions generated using our approach. The first bar reports the speedup just due to the reduction in the number of special cases with our maximum consensus solution (MCS). The second bar reports the speedup due to reduction in the special cases along with other optimizations to range reduction and output compensation when compared to functions in the RLIBM project.

For feasible LDLPs, our nearly-maximum consensus formulation produces the final solution. For infeasible LDLPs, the solution reported by our nearly-maximum consensus solution is almost as good as the best solution that we generate. The only exception being the `log10f` function where our iterative loop for maximum consensus is needed to account for both the rounding errors resulting from rounding an exact rational solution to an FP solution and reducing the bias caused by a fixed basis, which was computed independently of the set of violated constraints $\mathcal{V}_1$. The number of violated constraints reduces from 39 to 8 with our iterative maximum consensus loop.

The sixth column of Table 2 reports the number of violated constraints in the best solution generated by the RLIBM solver. The RLIBM solver did not generate the best solution reported by our solver. Finally, the seventh column of Table 2 reports the time taken to produce the best solution with our solver. The time taken increases with the increase in the number of constraints. We are able to generate the best solution in approximately 36 minutes in the worst case (for `tanf`), which is two orders of magnitude faster than RLIBM's solver (see seventh column of Table 1).

Improvements in the performance of the resulting elementary functions. Figure 7 reports the speedup of the resulting elementary functions as a result of: (1) maximum consensus solutions generated by our solver and (2) other improvements to range reduction and output compensation along with the maximum consensus solutions. Each constraint that is violated by our solution is added as a branch condition. We use `__builtin_expect` intrinsics to provide hints to the compiler to optimize the common case. On average, our implementations with maximum consensus solutions are 8% faster than the corresponding RLibm functions with the same range reduction and output compensation code. The feasible LDLPs do not see any improvement in performance. The performance improvement is significant for infeasible LDLPs with a large number of special cases. The `logf` function has 24% speedup because it has a significant reduction in the number of constraints violated between the RLibm solution (*i.e.*, 24) and our solution (*i.e.*, 10). When our improvements to range reduction and output compensation are combined with maximum consensus solutions, we improved the performance of the resulting implementations by 20% on average when compared to previous RLibm functions.

8 RELATED WORK

We describe closely related work on solving linear programs and on computing the convex hull.

Solving Linear Programs. Detailed exposition on various approaches to solve linear programs ranging from the simplex algorithm to interior-point and ellipsoid methods can be found in the textbooks [59]. As a result of algorithmic and engineering advances, modern LP solvers can easily solve several thousand constraints and unknowns. Meggido [42] and Clarkson [20] made seminal advances by observing that certain classes of "feasible" linear programs that have low-dimensions can be solved much more effectively with randomized algorithms. Based on the advances of Clarkson, Seidel [51] observed that computing the convex hull can help in solving low-dimensional feasible linear programs with a simpler analysis. When these algorithms are applied to infeasible linear programs, they will not terminate and do not generate the maximum consensus solution. Further, all these algorithms work with real values and may not produce floating point solutions. In our prior work in the RLibm project, we used the ideas from Clarkson's method and designed a solver for feasible LDLPs that produce floating point solutions with the solve-and-refine loop [2]. The key idea in RLibm's adaptation of Clarkon's method is to solve the sample with real values and tighten the constraints when the sample solution with real values rounded to floating point values does not satisfy the sample. When given an infeasible LDLP, RLibm's solver fails to terminate and find the maximum consensus solution similar to Clarkson's method. We use our previous RLibm solver for feasible LDLPs internally once we create a feasible subset of the infeasible LDLP.

Irreducible infeasible subsets. A subset C of constraints that itself is infeasible, but any proper subset of C is feasible, is called an *irreducible infeasible subset* (IIS). The concept of an IIS was first introduced for general optimization problems and later introduced for linear programs [29, 58]. The first practical methods for computing them were presented in the seminal work of Chinneck and Dravnieks [18]. Since then, after identifying an IIS in a linear program, various methods have been proposed for removing constraints until the system becomes feasible [17, 18, 54, 60].

Maximum consensus formulations. The concept of *maximum feasible subsets* (MAXFS) of constraints were first introduced by Amaldi *et al.* [6]. Some equivalent formulations for MAXFS include the minimum unsatisfied linear relation problem [5] and the minimum cardinality IIS set-covering problem [16]. All these formulations are known to be NP-hard [49]. Greenberg *et al.* [28] have proposed a mixed-integer linear program computing the MAXFS for linear constraints. The vision community has explored similar maximum consensus solutions for many model-fitting problems on real-world data, which is contaminated by noise and outliers. Similar to the LDLP formulation of RLibm [2], there are several "hypothesize-and-verify" methods in computer vision

as well, predominantly RANSAC [25] and its variants. They do not provide good solutions for infeasible linear programs. Hence, globally optimal algorithms for maximum consensus, such as branch-and-bound search [35, 62], tree search [15], or exhaustive search [24, 45] have been explored. However, they work only for small problem sizes. Our formulation is inspired by the prior MCS work [34] which decomposed the maximum consensus problem into two separate linear programs using a penalty formulation. However, this MCS formulation [34] destroys the low-dimensional property, which is crucial for solving LPs with billions of constraints. Hence, it does not solve any infeasible or feasible system as we show in our evaluation in Section 7.

Computing the convex hull. Exactly computing the convex hull of n points that are embedded in d-dimensional Euclidean space has an asymptotic run-time complexity of $O(n^d)$ [22]. The fastest known algorithm for computing the convex hull has a complexity of $O(n^2 + |F| \log n)$ [50], where $|F|$ is the number of faces on the convex hull. This is still prohibitively expensive when n is in the order of billions of constraints. The complexity of exactly computing the convex hull was recently recognized by Müller et al. [44] in the context of formal verification of deep neural networks where they propose an approximation algorithm for computing the convex hull.

For identifying a superset of the infeasible constraints, we only need to identify the points that lie on the convex hull and do not require knowledge of the connectivity structure (*i.e.*, the topology of the convex hull). Some researchers have recognized this problem to be easier than computing the convex hull and termed it as the *frame problem* [21]. A good discussion of the differences in complexity of the convex hull and the frame problem is provided in [31]. The method proposed by Dulá *et al.* [21] requires a complex initialization procedure and is serial in nature, where a linear program is needed to be solved per iteration. This approach is very slow for problem sizes with billions of constraints. Thus, we designed a different approach that can very quickly identify a superset of the infeasible constraints by computing the convex hull in the dual space.

Correctly rounded math libraries. The seminal book [43] by Muller provides a detailed survey on generating correctly rounded math libraries. The standard approach to develop correctly rounded math libraries prior to the RLibm project was to create polynomial approximations that minimizes the maximum error across all inputs using the Remez algorithm. In contrast to minimax methods, the RLibm project makes a case for directly approximating the correctly rounded result and creating a rounding interval, which naturally leads to an LDLP. The RLibm project has shown that the freedom available to the polynomial generator is much larger than minimax methods. Hence, solving large LDLPs with billions of constraints with few violated constraints is crucial for performance of the resulting functions. Given that infeasible LDLPs are common with the RLibm project, our solver makes the resulting math libraries much faster by producing maximum consensus solutions.

9 CONCLUSION

We propose a new method for solving infeasible low-dimensional linear programs with billions of constraints to generate floating point solutions that satisfy the maximum number of constraints. Our key idea is to create a superset of infeasible constraints by computing the convex hull and subsequently create a new linear program with slack variables whose solution satisfies the maximum number of constraints in the original LDLP. In the context of LDLPs generated in the RLibm project, our method not only produces the best solution but also does it significantly faster. The resulting solutions from our solver along with other optimizations to range reduction and output compensation helped us improve the performance of RLibm's math libraries by 20% on average. Our approach to solve large infeasible LDLPs with floating point solutions will likely be useful in various domains such as neural network verification, robotics, and computer vision, which we plan to explore in the future.

ACKNOWLEDGMENTS

We thank the PLDI reviewers, Bill Zorn, and members of the Rutgers Architecture and Programming Languages (RAPL) lab for their feedback on this paper. This material is based upon work supported in part by the National Science Foundation with grants: 2110861, 2312220, and 1908798 and a research gift from Intel Corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or Intel.

REFERENCES

[1] [n. d.]. *Cyclic Polytope.* https://en.wikipedia.org/wiki/Cyclic_polytope

[2] Mridul Aanjaneya, Jay P. Lim, and Santosh Nagarakatte. 2022. Progressive Polynomial Approximations for Fast Correctly Rounded Math Libraries. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (San Diego, CA, USA) *(PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 552–565. https://doi.org/10.1145/3519939.3523447

[3] Mridul Aanjaneya and Santosh Nagarakatte. 2023. Fast Polynomial Evaluation for Correctly Rounded Elementary Functions Using the RLIBM Approach. In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization* (Montréal, QC, Canada) *(CGO 2023)*. Association for Computing Machinery, New York, NY, USA, 95–107. https://doi.org/10.1145/3579990.3580022

[4] Mridul Aanjaneya and Santosh Nagarakatte. 2024. *RLIBM's Maximum Consensus LP Solver.* https://github.com/rutgers-apl/The-RLIBM-Project/tree/main/mcs-lp-solver

[5] Edoardo Amaldi. 1994. From finding maximum feasible subsystems of linear systems to feedforward neural network design. (01 1994). https://doi.org/10.5075/epfl-thesis-1282

[6] Edoardo Amaldi, Marc E. Pfetsch, and Leslie E. Trotter. 1999. Some Structural and Algorithmic Properties of the Maximum Feasible Subsystem Problem. In *Integer Programming and Combinatorial Optimization*, Gérard Cornuéjols, Rainer E. Burkard, and Gerhard J. Woeginger (Eds.). 45–59. https://doi.org/10.1007/3-540-48777-8_4

[7] José L. Balcázar, Yang Dai, Junichi Tanaka, and Osamu Watanabe. 2008. Provably Fast Training Algorithms for Support Vector Machines. *Theory of Computing Systems* 42, 4 (01 May 2008), 568–595. https://doi.org/10.1007/s00224-007-9094-6

[8] José L. Balcázar, Yang Dai, and Osamu Watanabe. 2001. Provably Fast Training Algorithms for Support Vector Machines. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM '01)*. IEEE Computer Society, USA, 43–50. https://doi.org/10.1109/ICDM.2001.989499

[9] José L. Balcázar, Yang Dai, and Osamu Watanabe. 2001. A Random Sampling Technique for Training Support Vector Machines. In *Proceedings of the 12th International Conference on Algorithmic Learning Theory (ALT '01)*. Springer-Verlag, Berlin, Heidelberg, 119–134. https://doi.org/10.1007/3-540-45583-3_11

[10] Colin Bell, Anil Nerode, Raymond T. Ng, and V. S. Subrahmanian. 1992. Implementing Deductive Databases by Linear Programming. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (San Diego, California, USA) *(PODS '92)*. Association for Computing Machinery, New York, NY, USA, 283–292. https://doi.org/10.1145/137097.137892

[11] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed. ed.). Springer-Verlag TELOS, Santa Clara, CA, USA. https://doi.org/10.1007/978-3-540-77974-2

[12] Kevin Q. Brown. 1978. Fast Intersection of Half Spaces. https://apps.dtic.mil/sti/citations/ADA058787 CMU Department of Computer Science Technical Report ADA058787.

[13] Luca Carlone and Daniel Lyons. 2014. Uncertainty-constrained robot exploration: A mixed-integer linear programming approach. *Proceedings - IEEE International Conference on Robotics and Automation*, 1140–1147. https://doi.org/10.1109/ICRA.2014.6906997

[14] Bernard Chazelle, Leo J. Guibas, and D. T. Lee. 1985. The power of geometric duality. *BIT Numerical Mathematics* 25, 1 (01 Mar 1985), 76–90. https://doi.org/10.1007/BF01934990

[15] Tat-Jun Chin, Pulak Purkait, Anders Eriksson, and David Suter. 2015. Efficient globally optimal consensus maximisation with tree search. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2413–2421. https://doi.org/10.1109/TPAMI.2016.2631531

[16] John W. Chinneck. 1996. An effective polynomial-time heuristic for the minimum-cardinality IIS set-covering problem. *Annals of Mathematics and Artificial Intelligence* 17 (1996), 127–144. https://doi.org/10.1007/BF02284627

[17] John W. Chinneck. 1997. Finding a Useful Subset of Constraints for Analysis in an Infeasible Linear Program. *INFORMS J. Comput.* 9 (1997), 164–174. https://doi.org/10.1287/ijoc.9.2.164

[18] John W. Chinneck and Erik W. Dravnieks. 1991. Locating Minimal Infeasible Constraint Sets in Linear Programs. *INFORMS J. Comput.* 3 (1991), 157–168. https://doi.org/10.1287/ijoc.3.2.157

[19] Vasek Chvatal. 1983. *Linear Programming*. W. H. Freeman.
[20] Kenneth L. Clarkson. 1995. Las Vegas Algorithms for Linear and Integer Programming When the Dimension is Small. *J. ACM* 42, 2 (March 1995), 488–499. https://doi.org/10.1145/201019.201036
[21] J.H. Dulá and R.V. Helgason. 1996. A new procedure for identifying the frame of the convex hull of a finite collection of points in multidimensional space. *European Journal of Operational Research* 92, 2 (1996), 352–367. https://doi.org/10.1016/0377-2217(94)00366-1
[22] Herbert Edelsbrunner. 1987. *Algorithms in Combinatorial Geometry*. Springer. https://doi.org/10.1007/978-3-642-61568-9
[23] Pavlos S. Efraimidis and Paul G. Spirakis. 2006. Weighted random sampling with a reservoir. *Inform. Process. Lett.* 97, 5 (2006), 181–185. https://doi.org/10.1016/j.ipl.2005.11.003
[24] Olof Enqvist, Erik Ask, Fredrik Kahl, and Kalle Åström. 2012. Robust Fitting for Multiple View Geometry. In *ECCV*. https://doi.org/10.1007/978-3-642-33718-5_53
[25] Martin A. Fischler and Robert C. Bolles. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM* 24, 6 (1981), 381–395. https://doi.org/10.1145/358669.358692
[26] Vinod Ganapathy, Somesh Jha, David Chandler, David Melski, and David Vitek. 2003. Buffer Overrun Detection Using Linear Programming and Static Analysis. In *Proceedings of the 10th ACM Conference on Computer and Communications Security* (Washington D.C., USA) *(CCS '03)*. Association for Computing Machinery, New York, NY, USA, 345–354. https://doi.org/10.1145/948109.948155
[27] W. Shane Grant, Randolph Voorhies, and Laurent Itti. 2013. Finding planes in LiDAR point clouds for real-time registration. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013), 4347–4354. https://doi.org/10.1109/IROS.2013.6696980
[28] Harvey Greenberg and Frederic Murphy. 1991. Approaches to Diagnosing Infeasible Linear Programs. *INFORMS Journal on Computing* 3 (08 1991), 253–261. https://doi.org/10.1287/ijoc.3.3.253
[29] Harvey J. Greenberg. 1993. Enhancements of ANALYZE: a computer-assisted analysis system for linear programming. *ACM Trans. Math. Softw.* 19, 2 (1993), 233–256. https://doi.org/10.1145/152613.152619
[30] Sariel Har-peled. 2011. *Geometric Approximation Algorithms*. American Mathematical Society.
[31] Christian Helbling. 2010. *Extreme points in medium and high dimensions*. Master Thesis. ETH Zürich. 1–83 pages. https://doi.org/10.3929/ethz-a-006250404
[32] Qixing Huang, Leonidas J. Guibas, and Niloy J. Mitra. 2014. Near-Regular Structure Discovery Using Linear Programming. *ACM Trans. Graph.* 33, 3, Article 23 (jun 2014), 17 pages. https://doi.org/10.1145/2535596
[33] David G. Kirkpatrick and Raimund Seidel. 1986. The Ultimate Planar Convex Hull Algorithm? *SIAM J. Comput.* 15, 1 (1986), 287–299. https://doi.org/10.1137/0215021
[34] Huu Le, Tat-Jun Chin, and David Suter. 2017. An Exact Penalty Method for Locally Convergent Maximum Consensus. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 379–387. https://doi.org/10.1109/CVPR.2017.48
[35] Hongdong Li. 2009. Consensus set maximization with guaranteed global optimality for robust geometry estimation. In *2009 IEEE 12th International Conference on Computer Vision*. 1074–1080. https://doi.org/10.1109/ICCV.2009.5459398
[36] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, L. Yi, and Leonidas J. Guibas. 2019. Supervised Fitting of Geometric Primitives to 3D Point Clouds. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), 2647–2655. https://doi.org/10.1109/CVPR.2019.00276
[37] Jay P. Lim, Mridul Aanjaneya, John Gustafson, and Santosh Nagarakatte. 2021. An Approach to Generate Correctly Rounded Math Libraries for New Floating Point Variants. *Proceedings of the ACM on Programming Languages* 6, POPL, Article 29 (Jan. 2021), 30 pages. https://doi.org/10.1145/3434310
[38] Jay P. Lim and Santosh Nagarakatte. 2021. High Performance Correctly Rounded Math Libraries for 32-bit Floating Point Representations. In *42nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'21)*. https://doi.org/10.1145/3453483.3454049
[39] Jay P Lim and Santosh Nagarakatte. 2021. RLIBM-32: High Performance Correctly Rounded Math Libraries for 32-bit Floating Point Representations. arXiv:2104.04043 Rutgers Department of Computer Science Technical Report DCS-TR-754.
[40] Jay P. Lim and Santosh Nagarakatte. 2022. One Polynomial Approximation to Produce Correctly Rounded Results of an Elementary Function for Multiple Representations and Rounding Modes. *Proceedings of the ACM on Programming Languages* 6, POPL, Article 3 (Jan. 2022), 28 pages. https://doi.org/10.1145/3498664
[41] O.L. Mangasarian and David Musicant. 2002. Large Scale Kernel Regression via Linear Programming. *Machine Learning* 46 (01 2002), 255–269. https://doi.org/10.1023/A:1012422931930
[42] Nimrod Megiddo. 1984. Linear Programming in Linear Time When the Dimension Is Fixed. *J. ACM* 31, 1 (Jan. 1984), 114–127. https://doi.org/10.1145/2422.322418

[43] Jean-Michel Muller. 2016. *Elementary Functions: Algorithms and Implementation.* Sprinder, 3rd edition. https://doi.org/10.1007/978-1-4899-7983-4

[44] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2022. PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. *Proc. ACM Program. Lang.* 6, POPL, Article 43 (jan 2022), 33 pages. https://doi.org/10.1145/3498704

[45] Carl Olsson, Olof Enqvist, and Fredrik Kahl. 2008. A polynomial-time bound for matching and registration with outliers. *2008 IEEE Conference on Computer Vision and Pattern Recognition* (2008), 1–8. https://doi.org/10.1109/CVPR.2008.4587757

[46] Arjun Pitchanathan, Christian Ulmann, Michel Weber, Torsten Hoefler, and Tobias Grosser. 2021. FPL: Fast Presburger Arithmetic through Transprecision. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 162 (oct 2021), 26 pages. https://doi.org/10.1145/3485539

[47] Arthur Richards, Eric Feron, Jonathan How, and Tom Schouwenaars. 2002. Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming. *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM* 25 (07 2002). https://doi.org/10.2514/2.4943

[48] Rómer Rosales and Glenn Fung. 2006. Learning Sparse Metrics via Linear Programming. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Philadelphia, PA, USA) *(KDD '06)*. Association for Computing Machinery, New York, NY, USA, 367–373. https://doi.org/10.1145/1150402.1150444

[49] Jayaram K Sankaran. 1993. A note on resolving infeasibility in linear programs by constraint relaxation. *Operations Research Letters* 13, 1 (1993), 19–20. https://doi.org/10.1016/0167-6377(93)90079-V

[50] R Seidel. 1986. Constructing Higher-Dimensional Convex Hulls at Logarithmic Cost per Face. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing (STOC '86)*. 404–413. https://doi.org/10.1145/12130.12172

[51] Raimund Seidel. 1990. Linear Programming and Convex Hulls Made Easy. In *Proceedings of the Sixth Annual Symposium on Computational Geometry* (Berkley, California, USA) *(SCG '90)*. Association for Computing Machinery, New York, NY, USA, 211–215. https://doi.org/10.1145/98524.98570

[52] Matthew Sotoudeh and Aditya V. Thakur. 2021. Provable Repair of Deep Neural Networks. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 588–603. https://doi.org/10.1145/3453483.3454064

[53] Umar Syed, Michael Bowling, and Robert E. Schapire. 2008. Apprenticeship Learning Using Linear Programming. In *Proceedings of the 25th International Conference on Machine Learning* (Helsinki, Finland) *(ICML '08)*. Association for Computing Machinery, New York, NY, USA, 1032–1039. https://doi.org/10.1145/1390156.1390286

[54] M. Tamiz, S.J. Mardle, and D.F. Jones. 1996. Detecting IIS in infeasible linear programmes using techniques from goal programming. *Computers and Operations Research* 23, 2 (1996), 113–119. https://doi.org/10.1016/0305-0548(95)00018-H

[55] Zhe Tao, Stephanie Nawas, Jacqueline Mitchell, and Aditya V. Thakur. 2023. Architecture-Preserving Provable Repair of Deep Neural Networks. *Proc. ACM Program. Lang.* 7, PLDI, Article 124 (jun 2023), 25 pages. https://doi.org/10.1145/3591238

[56] The CGAL Project. 2023. *CGAL User and Reference Manual* (5.6 ed.). CGAL Editorial Board. https://doc.cgal.org/5.6/Manual/packages.html

[57] Trung-Thien Tran, Van-Toan Cao, and Denis Laurendeau. 2015. Extraction of cylinders and estimation of their parameters from point clouds. *Computers & Graphics* 46 (02 2015), 345–357. https://doi.org/10.1016/j.cag.2014.09.027

[58] J.N.M. van Loon. 1981. Irreducibly inconsistent systems of linear inequalities. *European Journal of Operational Research* 8, 3 (1981), 283–288. https://doi.org/10.1016/0377-2217(81)90177-6

[59] Robert J. Vanderbei. 2020. *Linear programming: Foundations and extensions.* Springer. 1–465 pages. https://doi.org/10.1007/978-1-4614-7630-6

[60] Jian Yang. 2008. Infeasibility resolution based on goal programming. *Comput. Oper. Res.* 35, 5 (2008), 1483–1493. https://doi.org/10.1016/j.cor.2006.08.006

[61] Li Zhang and Wei-Da Zhou. 2011. Sparse ensembles using weighted combination methods based on linear programming. *Pattern Recognition* 44, 1 (2011), 97–106. https://doi.org/10.1016/j.patcog.2010.07.021

[62] Yinqiang Zheng, Shigeki Sugimoto, and Masatoshi Okutomi. 2011. Deterministically maximizing feasible subsystem for robust model fitting with unit norm constraint. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1825 – 1832. https://doi.org/10.1109/CVPR.2011.5995640

Received 2023-11-16; accepted 2024-03-31