Research Article BEU Fen Bilimleri Dergisi 15(2): 615-629 (2026) DOI: 10.17798/bitlisfen.1804064

BİTLİS EREN UNIVERSITY
FEN BİLİMLERİ DERGİSİ
ISSN: 2147-3129 / e-ISSN: 2147-3188

GEOMETRIC INITIALIZATION: A STRATEGY FOR ACCELERATING THE SIMPLEX METHOD

Ahmet AGAOGLU

Yeditepe University, Mechanical Engineering Department, Istanbul, Türkiye, ahmet.agaoglu@yeditepe.edu.tr

Article Info

Received: *October 15, 2025*
Revised: *April 22., 2026*
Accepted: *April 24, 2026*

Keywords
Linear Programming,
Simplex,
Polytope,
Initialization.

ABSTRACT

The performance of the simplex method in solving linear programming (LP) problems is heavily influenced by the choice of the initial basic feasible solution. While Phase I of the simplex method identifies a feasible starting point, it does not explicitly aim to optimize this choice in terms of proximity to the optimal solution. This paper presents a geometry-driven initialization strategy that leverages the convex polytope structure of the feasible region to improve the quality of the starting point and reduce the number of simplex iterations.

The method begins from a feasible point and performs a series of interior moves to approach a boundary point near optimality. Once such a point is reached, the algorithm locates the nearest vertex, which is then used as the starting point for Phase II of the simplex method. The proposed approach preserves compatibility with the classical simplex framework and introduces no additional pivoting logic. Numerical experiments on synthetically generated LP instances, including problems with up to 200 variables and 400 constraints, demonstrate that the method consistently reduces both iteration counts and computation times. The method achieves an average iteration savings of 46.5% and a time savings of 30.4% compared to classical simplex implementations, with improvements being especially pronounced in large-scale problems.

1. INTRODUCTION

The simplex method has long been the cornerstone of linear programming (LP) due to its robustness and effectiveness in practice. Despite its exponential worst-case complexity [1], [2], its average-case performance has been shown to be polynomial under certain probabilistic models [3]. Modern simplex-based solvers further benefit from advanced pivot rules and heuristics that allow them to solve LP problems of various sizes with high efficiency [4], [5], [6], [7]. As a result, the simplex method remains widely used, especially in small to medium-scale applications [8].

Geometrically, the simplex method explores the feasible region, which is a convex polytope, by moving from vertex to vertex along its edges. It operates in two main phases:

Phase I: Identify an initial vertex of the polytope, i.e., a basic feasible solution.

Phase II: Starting from the initial vertex, move to an adjacent vertex that improves the objective value. This process is repeated until no such improvement is possible, indicating optimality.

Despite its practical efficiency, the vertex-to-vertex traversal of the simplex method can become computationally expensive, particularly when the number of variables and constraints is large. In such cases, the boundary of the feasible region becomes highly intricate, and the number of vertices grows exponentially with problem size [2], leading to a longer simplex path in terms of iteration count.

Significant research has focused on enhancing the core algorithm through improved pivot rules [9], [10], advanced dual simplex variants [11], [12], [13], and interior search directions within the feasible region while remaining within the simplex framework [14], [15], [16], [17]. These modifications aim to reduce the number of iterations or improve numerical stability.

License: CC BY-NC 4.0

An alternative approach is to optimize the choice of the initial vertex, which constitutes the second critical factor influencing the overall efficiency. While a poorly chosen starting point may result in inefficient traversal through many suboptimal vertices, a strategically chosen one can significantly shorten the simplex path (see Figure 1). However, the standard Phase I procedure of the simplex method typically does not aim to select an initial vertex that is close to optimality.

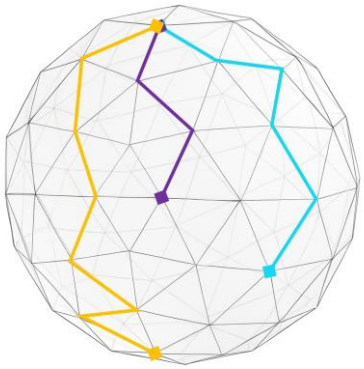


Figure 1. Simplex paths from different initial vertices to the optimal solution on the same polytope.

Motivated by this observation, several researchers have focused on improving the quality of starting point. Stojkovic and Stanimirovic [18], Junior and Lins [19] proposed methods to identify near-optimal bases by analyzing the geometric relationship between constraints and the objective function. Their key insight is that constraints forming the smallest angles with the gradient of the objective function are more likely to define the optimal vertex. This cosine criterion prioritizes constraints that are geometrically aligned with the optimization direction, often yielding a basis closer to the optimal solution. Later, Hu [20] simplified the basis selection process in Junior and Lins' method by reducing reliance on angle computations, making it more scalable for large-scale problems.

Luh and Tsaih [21] introduced a novel search direction combining two components: an interior direction pointing toward the feasible region's interior to maintain feasibility, and a gradient direction aligning with the negative objective gradient to drive improvement. This composite direction, computed iteratively from boundary points, balances exploration of the feasible region with convergence toward optimality. Chaderjian and Gao [22] later refined this method by addressing edge cases, such as zero denominators, and demonstrated that initializing with the negative gradient direction further improves performance.

Al-Najjar and Malakooti [23] introduced the Hybrid-LP method, a two-phase approach that combines interior-point and simplex techniques to enhance initialization. In the first phase, Hybrid-LP moves through the interior of the feasible region using computationally efficient pivots to bypass multiple vertices and reach an improved vertex of the feasible region. This advanced starting point is then used in the second phase, where traditional simplex or other LP methods are applied to find the optimal solution. The Hybrid-LP method is particularly notable for its adaptability, as it can start from any feasible point, not necessarily a vertex, and avoids the boundaries of the feasible region to improve efficiency.

The proposed method draws on the strengths of these strategies while avoiding their main limitations. Starting from a strictly feasible interior point, it performs a sequence of geometrically informed line searches to approach a near-optimal boundary point, bypassing a significant portion of the simplex path. A subsequent boundary walk then identifies the nearest vertex, which serves as the starting point for Phase II of the simplex method. Unlike angle-based approaches, the method is not sensitive to degeneracy; and unlike hybrid techniques, it introduces no additional pivoting logic, maintaining full compatibility with the standard simplex framework. Experimental results on problems with up to 200 variables and 400 constraints demonstrate consistent reductions in both iteration count and total solve time. Unlike presolve reductions or crash heuristics, which aim to simplify the problem or quickly construct a feasible basis, the proposed method focuses on improving the geometric quality of the starting point for the simplex method.

2. PRELIMINARIES

Consider an LP problem in the standard form:

$$\begin{aligned} \min_{x} \quad & c^{\top}x \\ \text{s.t.} \quad & \tilde{A}x \leq \tilde{b}, \\ & Dx = e, \\ & \ell \leq x \leq u, \end{aligned} \tag{1}$$

where the variable bounds $\ell \leq x \leq u$ can be reformulated as additional linear inequality constraints. Incorporating these into the system yields the equivalent form:

$$\begin{aligned} \min_{x} \quad & c^{\top}x \\ \text{s.t.} \quad & Ax \leq b, \\ & Dx = e, \end{aligned} \tag{2}$$

where $x \in \mathrm{R}^n$ denotes the vector of decision variables, and $c \in \mathrm{R}^n$ is the cost vector defining the linear objective function. The matrix $A \in \mathrm{R}^{mi \times n}$ and the vector $b \in \mathrm{R}^{mi}$ represent the inequality constraints, which include both the original constraints and those derived from variable bounds. The matrix $D \in \mathrm{R}^{me \times n}$ and the vector $e \in \mathrm{R}^{me}$ define the set of equality constraints.

From a geometric perspective, each equality constraint $D_j x = e_j$, where D_j denotes the jth row of matrix D, defines a hyperplane in R^n. The vector D_j serves as the normal vector of the hyperplane, and e_j determines its offset from the origin. For a point x to satisfy all equality constraints, it must lie on the intersection of all such hyperplanes. This intersection defines an affine subspace, which corresponds to a translated null space of D.

In contrast, each inequality constraint $A_i x \leq b_i$ defines a closed half-space in R^n. The boundary of this half-space is the hyperplane $A_i x = b_i$, which separates feasible and infeasible regions. The intersection of all such half-spaces forms a convex polytope in R^n. The feasible set F is therefore the intersection of this polytope and the affine subspace defined by the equality constraints:

$$\mathcal{F} = \{x \in \mathbb{R}^n \mid Ax \leq b, Dx = e\}, \tag{3}$$

which geometrically corresponds to a lower-dimensional polytope embedded within the affine subspace determined by $Dx = e$ (see Figure 2).

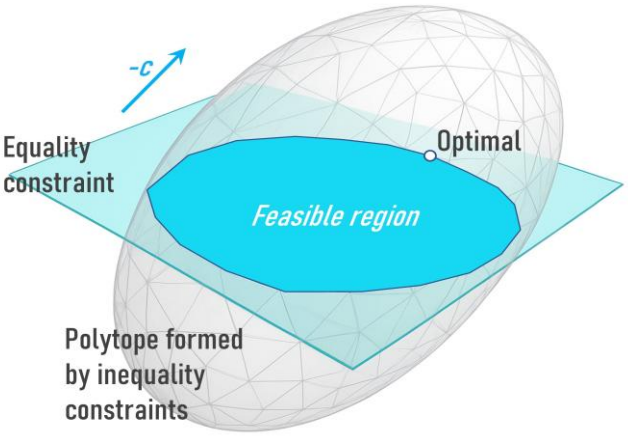


Figure 2. Geometric interpretation of a linear program. The polytope formed by the inequality constraints defines the ambient feasible region, while the equality constraints restrict the solution to an affine subspace. Their intersection yields the feasible region F, shown in solid turquoise. The optimal solution lies on a vertex of this lower-dimensional polytope in the direction of −c.

3. ALGORITHM DESIGN AND IMPLEMENTATION

Starting from a feasible point $x_0 \in F$, the algorithm systematically generates a sequence of feasible points that monotonically decrease the objective function value, efficiently exploring the feasible region. We assume that the linear program is bounded and has a unique global minimizer, which is a common scenario in practical applications. The proposed method performs line searches of the form

$$x_{k+1} = x_k + \alpha_k d_k, \tag{4}$$

where $x_k \in R^n$ denotes the current feasible point, $d_k \in R^n$ is a *feasible and descent* search direction at iteration k, and $\alpha_k \geq 0$ represents the maximal step size ensuring x_{k+1} remains within the feasible region F. The set of descent directions is defined as

$$\mathcal{D}^d = \{d \in \mathbb{R}^n \mid c^T d < 0\}, \tag{5}$$

comprising directions that strictly reduce the objective function. Let

$$\mathcal{I}_k = \{i \mid A_i x_k = b_i\} \tag{6}$$

denote the index set of inequality constraints active at the current point x_k. The feasible directions at x_k form the set

$$\mathcal{D}_k^f = \{d \in \mathbb{R}^n \mid Dd = 0, A_{\mathcal{I}_k} d \leq 0\}. \tag{7}$$

Note that all equality constraints remain inherently active throughout the iterations, i.e., $Dx_k = e$ for all k, and are strictly maintained by enforcing $Dd_k = 0$ at each step. Consequently, a direction d_k is feasible and descent if it belongs to the intersection $\mathcal{D}^d \cap \mathcal{D}_k^f$.

The maximal step size α_k is computed such that the updated point x_{k+1} remains within the feasible region F. Specifically, feasibility requires

$$A_i x_{k+1} = A_i(x_k + \alpha_k d_k) \leq b_i, \forall i \tag{8}$$

Solving this inequality for αk yields

$$\alpha_k \leq \frac{b_i - A_i x_k}{A_i d_k}, \forall i \tag{9}$$

Since $x_k \in F$, it follows that $b_i - A_i x_k \geq 0$. To ensure forward progress ($\alpha_k \geq 0$), we must have $A_i d_k > 0$. Hence, the maximal feasible step size is given by

$$\alpha_k = \min_{\substack{i \\ A_i d_k > 0}} \frac{b_i - A_i x_k}{A_i d_k}. \tag{10}$$

We now present the core mechanism of the proposed algorithm. For clarity, the algorithm is introduced in two distinct phases:

> ***Phase A*:** Starting from a strictly feasible point x_0, obtain a boundary point of F that is near-optimal relative to x_0.
>
> ***Phase B*:** Given the near-optimal boundary point from Phase A, identify the closest vertex of F along the steepest descent direction $-c$.

Note that the starting point x_0 must be strictly feasible with respect to the inequality constraints, explicitly, $Dx_0 = e$ and $Ax_0 < b$. In practice, such a point can be obtained using a procedure similar to the standard Phase I of the simplex method. To ensure strict feasibility, we slightly tighten the inequality constraints by introducing a small parameter $\tau > 0$ and solve the following modified auxiliary problem:

$$\begin{aligned} \min_{x,\xi} \quad & \xi \\ \text{s.t.} \quad & Dx = e \\ & Ax \leq b - \tau + 1\xi, \\ & \xi \geq 0. \end{aligned} \tag{11}$$

where $\xi \in \mathrm{R}$ is an auxiliary scalar variable and $\mathbf{1} \in \mathrm{R}^{mi}$ is the vector of ones. If the optimal solution yields $\xi^* = 0$, then the resulting point x_0 is strictly feasible. The complete workflow of the proposed approach is summarized in Figure 3.

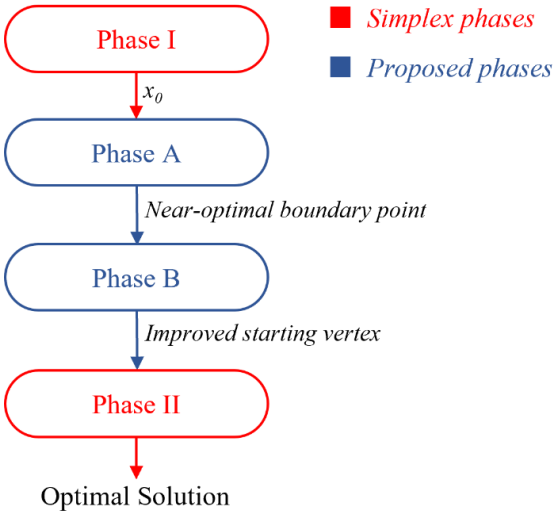


Figure 3. Overview of the proposed geometric initialization framework.

3.1. Phase A: Generating a Near-Optimal Boundary Point

Since no inequality constraint is active at x_0, the initial search direction d_0 is determined solely based on the conditions $Dd_0 = 0$ and $c^{\mathrm{T}}d_0 < 0$. Specifically, to ensure descent while preserving feasibility, we project the steepest descent direction $-c$ onto the null space of the equality constraints:

$$d_0 = ZZ^{\mathrm{T}}(-c), \tag{12}$$

where $Z \in \mathrm{R}^{n \times p}$ is an orthonormal basis for null(D), with p = dim(null(D)). Note that $d_0 \in \mathrm{D}_0^f$ since

$$Dd_0 = DZZ^{\mathrm{T}}(-c) = 0 \tag{13}$$

by construction, and $d_0 \in \mathrm{D}^d$ since

$$c^{\mathrm{T}}d_0 = -c^{\mathrm{T}}ZZ^{\mathrm{T}}c = -\| Z^{\mathrm{T}}c \|^2 < 0, \tag{14}$$

provided $Z^{\mathrm{T}}c \neq 0$. This guarantees that d_0 is both feasible and descent. If $Z^{\mathrm{T}}c = 0$, then $-c$ lies entirely in the row space of D, indicating that no descent direction exists within the feasible subspace, which is an exceptional case excluded under our assumptions.

Once d_0 is obtained, the next step is to compute the maximal step size α_0 and update the iterate to x_1. This line search necessarily terminates at one or more inequality constraints, making x_1 a boundary point of F. Consequently, in the next iteration(s), the search direction d_k must satisfy the additional condition $A_{Ik}d_k \leq 0$, in addition to lying in the null space of the equality constraints. Satisfaction of this condition yields two possibilities:

a) If $A_{Ik}d_k = 0$, then d_k is tangent to the boundary of F at x_k, as is typical in the classical simplex method.

b) If $A_{Ik}d_k < 0$, then d_k points strictly into the interior of F, which is the behavior encouraged by the proposed strategy.

To obtain such a strictly interior search direction, we begin by constructing a *boundary direction* d_k^ℓ corresponding to case (a). This is accomplished by treating the active inequality constraints as equality constraints and projecting $-c$ onto the intersection of the null spaces of these constraints. Accordingly,

$$d_k^\ell = N_k N_k^\mathsf{T}(-c), \tag{15}$$

where $N_k \in \mathrm{R}^{n\times s}$ is an orthonormal basis for the null space of the active constraint matrix

$$P_k = \begin{bmatrix} D \\ A_{\mathcal{I}_k} \end{bmatrix}. \tag{16}$$

Similar to the analysis in Equations (13) and (14), it follows that $d_k^\ell \in \mathcal{D}_k^f \cap \mathcal{D}_d$, provided $N_k^\mathsf{T} c \neq 0$. A perturbation is then applied to d_k^ℓ to steer the direction away from the boundary and into the interior of F. To this end, we employ spherical linear interpolation (slerp) between the boundary direction d_k^ℓ and the negation of the previous search direction $-d_{k-1}$. The geometric intuition behind this interpolation is illustrated in Figure 4.

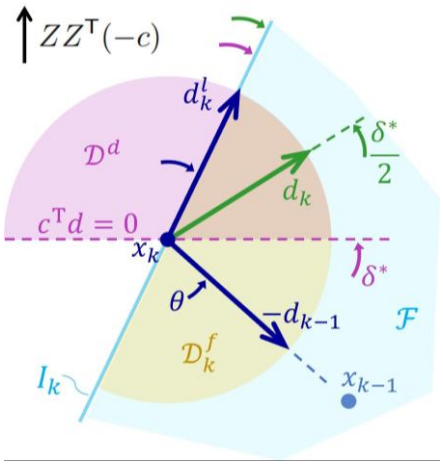


Figure 4. Geometric construction of the search direction dk at iteration k. The magenta and yellow hemispheres represent the descent cone Dd and the feasible cone $\mathcal{D}_k^f$, respectively. Their intersection defines directions that are both descent and feasible. The final direction dk is obtained by interpolating between the boundary-aligned direction d_k^ℓ and the previous direction −dk−1.

Assume both d_k^ℓ and $-d_{k-1}$ are normalized to unit vectors. Let θ_k be the angle between them, computed as

$$\theta_k = \cos^{-1}\left(-d_{k-1}^\mathsf{T} d_k^\ell\right), \tag{17}$$

where $\theta_k \in (0,\pi)$. Then, the interpolated search direction is given by

$$d_k = \frac{\sin(\theta_k - \delta_k)}{\sin\theta_k} d_k^\ell + \frac{\sin\delta_k}{\sin\theta_k}(-d_{k-1}), \tag{18}$$

where $\delta_k \in (0,\theta_k)$ is the perturbation angle. Since only the direction of d_k matters (not its magnitude), this simplifies to

$$d_k = \sin(\theta_k - \delta_k) d_k^\ell - \sin\ \delta_k d_{k-1}. \tag{19}$$

We now analyze the properties of the interpolated direction d_k. First, we show that $d_k \in \mathcal{D}_k^f$. Then, we derive an upper bound on the perturbation angle δ_k to ensure that $d_k \in \mathrm{D}^d$.

Consider the direction $-d_{k-1} = (x_{k-1} - x_k)/\alpha_{k-1}$. Since $Dx_{k-1} = e$ and $Dx_k = e$, we have

$$D(-d_{k-1}) = \frac{Dx_{k-1} - Dx_k}{\alpha_{k-1}} = 0. \tag{20}$$

Additionally, assuming forward progress was made in the previous iteration (i.e., $\alpha_{k-1} > 0$), we have

$$A_{\mathcal{I}_k} x_k = b_{\mathcal{I}_k} \text{ and } A_{\mathcal{I}_k} x_{k-1} < b_{\mathcal{I}_k}, \tag{21}$$

which implies

$$A_{\mathcal{I}_k}(-d_{k-1}) = \frac{A_{\mathcal{I}_k} x_{k-1} - A_{\mathcal{I}_k} x_k}{\alpha_{k-1}} < 0. \tag{22}$$

Therefore, $-d_{k-1}$ is a strictly feasible direction (case b) and belongs to $\mathcal{D}_k^f$.

Both $d^\ell{}_k$ and $-d_{k-1}$ lie in the feasible direction set D_k^f, which forms a convex cone. Since the interpolation coefficients $\sin(\theta_k - \delta_k)$ and $\sin\delta_k$ are positive for $\delta_k \in (0, \theta_k)$ and $\theta_k \in (0, \pi)$, the resulting vector d_k is a conical combination of these two directions, implying $d_k \in \mathrm{D}_k^f$. Moreover, because $-d_{k-1}$ is strictly interior, the interpolation preserves this strict interiority, ensuring that $A_{1k} d_k < 0$.

Although the interpolation guarantees feasibility, the resulting vector may not lie in the descent direction set D^d, since $-d_{k-1}$ is not a descent direction. In the next step, we derive a bound $\delta_k{}^*$ on the perturbation angle to ensure that $d_k \in \mathrm{D}^d$. In particular, the boundary between descent and non-descent directions is defined by

$$c^{\mathsf{T}} d_k = c^{\mathsf{T}} \left(\sin(\theta_k - \delta_k) d_k^\ell - \sin \delta_k d_{k-1} \right) = 0. \tag{23}$$

Define

$$a_k = c^{\mathsf{T}} d_k^\ell \text{ and } b_k = c^{\mathsf{T}} d_{k-1}. \tag{24}$$

Then Equation (23) becomes

$$a_k \sin(\theta_k - \delta_k) - b_k \sin \delta_k = 0. \tag{25}$$

Using the sine subtraction identity,

$$\sin(\theta_k - \delta_k) = \sin \theta_k \cos \delta_k - \cos \theta_k \sin \delta_k, \tag{26}$$

Equation (25) becomes

$$a_k \sin \theta_k \cos \delta_k = (a_k \cos \theta_k + b_k) \sin \delta_k. \tag{27}$$

Since $d^\ell{}_k$ is a descent direction, any direction orthogonal to it cannot be descent. Thus, we restrict $\delta_k \in (0, \pi/2)$ to ensure $\cos\delta_k \neq 0$. Dividing both sides of Equation (27) by $\cos\delta_k$ yields

$$\tan \delta_k = \frac{a_k \sin \theta_k}{a_k \cos \theta_k + b_k}. \tag{28}$$

Thus, the unique threshold angle is given by

$$\delta_k^* = \arctan\left(\frac{a_k \sin\,\theta_k}{a_k \cos\,\theta_k + b_k}\right). \tag{29}$$

For $d_k \in \mathrm{D}^d$, the perturbation angle must satisfy

$$0 < \delta_k < \delta_k^*. \tag{30}$$

Consider the allowable range for δ_k. When δ_k is small, the interpolated direction d_k approaches the boundary direction $d^{\ell}{}_k$, meaning that d_k remains close to the boundary of F. In contrast, a larger δ_k causes d_k to deviate significantly from the descent direction, potentially compromising the decrease in the objective function.

Drawing on concepts from directional statistics, and assuming a symmetric distribution over the unit sphere such as the von Mises–Fisher distribution, it is natural to select δ_k as the midpoint of the angular support, corresponding to the mean direction of the distribution. Hence, we set

$$\delta_k = \frac{\delta_k^*}{2} \tag{31}$$

This choice provides a d_k that balances the trade-off between remaining sufficiently interior to F and preserving a strong descent characteristic.

After d_k is determined, the step size α_k is computed, and the current iterate is updated. This process is repeated, and the algorithm progressively moves from one boundary point to another, approaching a near-optimal solution.

Although rare, it is possible for the line search to terminate at a point lying at the intersection of n linearly independent constraints, corresponding to a vertex of F. In such a case, the null space associated with the active constraint matrix becomes trivial, resulting in an empty N_k and consequently a dimensionless direction d_k^{ℓ}. The algorithm proceeds by computing the Lagrange multipliers λ_k corresponding to the active constraints by solving the nonsingular system

$$P^{\mathsf{T}}\lambda_k = -c. \tag{32}$$

If all components of λ_k corresponding to the inequality constraints are nonnegative, then x_k satisfies the optimality conditions of the LP, and the algorithm terminates. However, if any multiplier associated with an inequality constraint is negative, the algorithm removes the constraint from P_k corresponding to the most negative multiplier and begins a new iteration.

Excluding this extreme case, as the iterations progress, the improvement in the objective value becomes negligible, thereby necessitating termination of the process. The primary termination criterion is based on a functional tolerance ε_{tol}, ensuring that the process terminates when the relative absolute change in the objective value satisfies:

$$\frac{|c^T x_{k+1} - c^T x_k|}{|c^T x_k|} < \varepsilon_{\text{tol}}. \tag{33}$$

In rare cases where progress becomes excessively slow or numerical instability arises, a maximum iteration limit $k_{\max}$ is imposed as a safeguard, ensuring termination when $k \geq k_{\max}$. Regardless of whether termination is triggered by the functional tolerance or the iteration limit, the process concludes by returning a boundary point that is near-optimal relative to the initial point x_0, marking the end of Phase A. The implementation of Phase A is presented in Algorithm 1 (Appendix).

3.2. Phase B: Vertex Identification via Boundary Walk

The subsequent objective is to identify the nearest vertex of F along the steepest descent direction, which lies at the intersection of n linearly independent constraints. This is accomplished by moving along the

boundary in the direction of $-c$ using line searches, similar to those in classical active-set methods. At this stage, the search direction is no longer computed using the interpolation strategy defined in Equation (19); instead, it is set directly as $d^{\ell}{}_k$. All other algorithmic steps presented in Subsection 3.1 remain unchanged. The process terminates when the number of linearly independent active constraints at the current iterate reaches n.

Note that if stagnation occurs (when $\alpha_k \leq 0$), the search direction is recomputed as

$$d_k = N_k N_k^{\mathsf{T}}(-c + r), \tag{34}$$

and the iteration repeats. Here, $r \in \mathrm{R}^n$ is a random vector with a magnitude that is sufficiently small relative to that of c, thereby providing a perturbation to overcome the stagnation while maintaining being a descent direction. Algorithm 2 (Appendix) presents the pseudocode implementation of Phase B. To clarify these concepts, Figure 5 provides a visual demonstration of the proposed method.

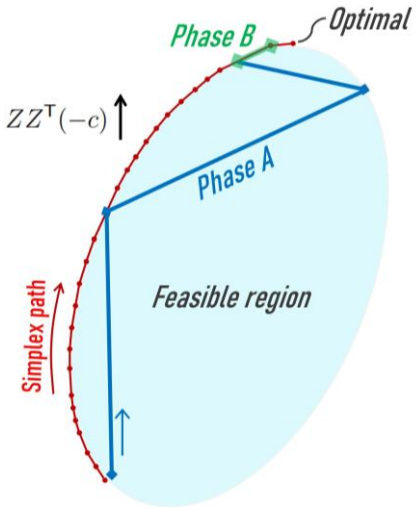


Figure 5. A visual example showing the sequence of points visited at each iteration.

4. EXPERIMENTAL RESULTS

This section presents the performance evaluation of the proposed algorithm and compares the results with those of the classical simplex method.

The Phase II component of the simplex method is implemented using the revised simplex algorithm, as described in [24]. Its Phase I routine solves the standard auxiliary linear program to compute the starting vertex.

To evaluate the efficiency, we consider two primary performance metrics: computational time and the number of simplex iterations. The total computational time for the classical simplex method (T_{CSM}) is calculated as the sum of the execution times of its Phase I and Phase II components. In contrast, the total time for the proposed method (T_{PM}) includes the durations of the modified Phase I, Phase A, Phase B, and Phase II routines.

To quantify the improvement, we define the *time savings* metric *TS* as

$$TS = \frac{T_{\mathrm{CSM}} - T_{\mathrm{PM}}}{T_{\mathrm{CSM}}} \times 100\%, \tag{35}$$

Additionally, to assess iteration-level efficiency, we define the *iteration savings* metric *IS* in a similar way as

$$IS = \frac{I_{\mathrm{CSM}} - I_{\mathrm{PM}}}{I_{\mathrm{CSM}}} \times 100\%. \tag{36}$$

Here, I_{CSM} and I_{PM} denote the number of iterations performed during Phase II of the classical simplex method and the proposed method, respectively.

All experiments were conducted on a system equipped with an Intel(R) Core(TM) i7-13700H 2.40 GHz processor, 32 GB DDR4 3200 MHz RAM, running MATLAB R2024a [25].

The proposed method involves several tunable parameters, which are set as follows:

- Tolerance for strict feasibility in the auxiliary problem: $\tau = 1 \times 10^{-6}$
- Functional tolerance for termination: $\varepsilon_{tol} = 1 \times 10^{-6}$
- Maximum number of iterations: $k_{max} = 10(m + n)$
- Random perturbation magnitude: $\| r \| = \frac{1}{10^3} \| c \|$

Three sets of problems are used to compare the performance of the proposed method and the classical simplex method:

- Randomly generated general LP problems,
- Randomly generated generalized transportation problems,
- Standard NETLIB test problems.

4.1. Randomly Generated General LP and Generalized Transportation Problems

For each type of problem, a collection of synthetically generated linear programs is constructed. The number of variables *n* ranges from 25 to 200, while the total number of constraints *m* (including both equality and inequality constraints) is varied to produce different problem sizes (small, medium, and large). All problems are generated to ensure a unique optimal solution exists.

Table 1. Iteration savings (IS), time savings (TS), and runtime breakdown for randomly generated general LP problems. Phase A, Phase B, and Phase II values are reported as percentages of the total proposed method time T_{PM}. The remainder corresponds to the modified Phase I duration. Density is fixed at 0.5. Each configuration is averaged over 100 repetitions.

Problem Size	n	m	IS	TS	Phase A	Phase B	Phase II
Small	25	13	38.1	2.2	18.2	10.8	48.5
	25	25	44.2	21.4	12.5	7.8	56.2
	25	50	68.5	40.2	16.8	11.2	45
Medium	100	50	26.8	-0.7	15.2	10.3	62
	100	100	36	27.7	5.8	3.7	75.5
	100	200	65.3	62.3	4.2	2.8	79.5
Large	200	100	26.4	8.3	10.5	7	71.5
	200	200	40.2	39.4	4	2.8	88
	200	400	73.1	72.8	2.2	1.5	93
Average			**46.5**	**30.4**	**9.9**	**6.4**	**68.8**

The sparsity of the constraint matrices is controlled via a density parameter, which is tested at values of 0.3, 0.5, and 0.7. Preliminary experiments indicate that the density has negligible influence on the number of simplex iterations and only a minor impact on the computational time. Therefore, for clarity and conciseness, only results corresponding to a density of 0.5 are reported.

To mitigate the effects of randomness and ensure statistical robustness, each experimental configuration is repeated 100 times. The reported results represent the mean values obtained over these repetitions. Table 1 reports the performance of the proposed method across various randomly generated *general* LP problems.

Table 2 reports the performance of the proposed method across various randomly generated generalized transportation problems.

Table 2. Iteration savings (IS), time savings (TS), and runtime breakdown for randomly generated transportation problems. Density is fixed at 0.5. Each configuration is averaged over 100 repetitions.

Problem Size	n	m	IS	TS	Phase A	Phase B	Phase II
Small	25	13	34.2	1.5	12.3	7.6	45.1
	25	25	40.5	18.3	10.1	6.2	58.9
	25	50	64.1	36.8	16.1	9.8	48
Medium	100	50	23.4	0.8	8.5	5.2	53.5
	100	100	32.6	23.1	4.6	2.8	79
	100	200	61.7	57.5	3.7	2.3	87.9
Large	200	100	23.1	5.6	6.9	4.2	61.9
	200	200	36.8	35.7	1.8	1.1	94.1
	200	400	69.5	68.4	2.4	1.5	94.8
Average			**42.9**	**27.5**	**7.4**	**4.5**	**69.2**

4.2. Standard NETLIB Test Problems

To further evaluate the proposed method on structured, real-world problems, we test it on a selection of 19 problems from the NETLIB benchmark suite [26]. These problems exhibit a variety of sizes, density levels, and degeneracy structures that are not captured by synthetic instances. Unlike the random test sets, each NETLIB problem is solved once, and IS and TS values are reported individually. Table 3 summarizes the results.

Table 3. Iteration savings (IS) and time savings (TS) for selected NETLIB benchmark problems. Negative TS values indicate that the proposed method is slower than the classical simplex method due to overhead costs exceeding iteration savings.

Problem	m	n	Non-zeros	IS	TS
ADLITTLE	57	97	465	33.5	22.8
AGG	489	163	2541	7.2	-9.4
AGG2	517	302	4515	16.8	8.3
AGG3	517	302	4531	58.4	51.2
BEACONFD	174	262	3476	2.1	-5.3
BLEND	75	83	521	6.8	1.2
E226	224	282	2767	3.5	-2.7
ISRAEL	175	142	2358	2.4	-4.1
LOTFI	154	308	1086	55.1	43.6
SC105	106	103	281	45.3	21.4
SC205	206	203	552	60.7	47.5
SC50A	51	48	131	71.2	50.6
SC50B	51	48	119	53.8	32.4
SCAGR25	472	500	2029	4.6	2.8
SCAGR7	130	140	553	3.8	0.5
SCFXM1	331	457	2612	2.9	-3.2
SHARE1B	118	225	1182	18.6	10.1
SHARE2B	97	79	730	3.2	-7.8
STOCFOR1	118	111	474	4.5	-6.3
Average				**23.9**	**13.3**

4.3. Parameter Sensitivity

The proposed method introduces several parameters, including the strict feasibility tolerance τ, the termination tolerance ε_{tol}, the iteration limit k_{max} and the perturbation magnitude r. These parameters serve different purposes within the algorithm.

The parameter τ is used only in the auxiliary Phase I problem to obtain a strictly feasible starting point and is applied identically in both the classical simplex method and the proposed approach. Therefore, it does not affect the relative performance comparison between the two methods.

To assess the sensitivity of the algorithm to the termination tolerance ε_{tol}, we conducted a small experiment by varying this parameter across several orders of magnitude while keeping the other parameters fixed. Table 4 reports the resulting average IS and TS over the random LP test set. The results indicate that the method is largely insensitive to the choice of ε_{tol} within the tested range.

Table 4. Sensitivity of the proposed method to the termination tolerance ε_{tol}, averaged over all randomly generated general LP problems.

ε_{tol}	IS	TS
10^{-4}	45.8	29.6
10^{-6}	46.5	30.4
10^{-8}	46.6	30.1

The parameter k_{max} is introduced only as a safeguard to prevent excessive iterations in rare cases where progress becomes very slow. In practice, this limit was not reached in the vast majority of test instances. Additional experiments with larger values of k_{max} produced nearly identical results, indicating that the performance of the algorithm is not sensitive to this parameter.

The perturbation vector r is used only in stagnation situations to avoid numerical degeneracy during the boundary walk in Phase B. In the extensive numerical experiments conducted in this study, such stagnation events were observed only rarely. Moderate variations in the magnitude of r did not produce any noticeable impact on the overall performance metrics.

5. DISCUSSION

Tables 1 and 2 present the runtime distribution across the three main phases of the proposed method. A clear pattern emerges: for small-scale problems (n = 25), Phase A and Phase B together account for 18–29% of the total proposed method time, leaving Phase II as the dominant but not overwhelming component. This substantial overhead explains why TS remains modest despite high IS values in small problems. As the problem size increases, the overhead fraction drops sharply: for n = 200 and m = 400, Phase A and Phase B together consume only 3.5–3.9% of the total time, while Phase II accounts for 93–95%. This convergence indicates that the computational cost of the geometric initialization becomes negligible relative to the simplex iterations it saves.

Based on these results, we identify a practical threshold for the applicability of the proposed method. The ratio of constraints to variables, m/n, emerges as the primary indicator of when the method becomes advantageous. When $m/n \geq 1$, the method consistently achieves TS values exceeding approximately 20% across all tested values of n, including small-scale problems. In contrast, when $m/n \leq 0.5$, the overhead of Phases A and B often dominates the iteration savings, resulting in marginal or negligible time savings regardless of problem size. This pattern is consistent across both random LP and transportation problem classes, though the absolute savings are moderately lower for transportation problems due to their structured constraint matrices, which already facilitate efficient simplex traversal.

In comparison, previous initialization strategies such as those in Refs. [19], [20], [21] report average IS of 30–34% but are limited to small-scale problems, with experiments conducted on instances containing up to 30 variables and 50 constraints. Furthermore, these studies do not report time-related metrics, leaving the overall computational benefit unclear. A similar limitation is observed in Ref. [20], which claims IS of up to 50% without providing any analysis of computational time (TS). In contrast, the

hybrid-LP method proposed in Ref. [23] includes experiments on a broader range of problem sizes (up to 200 variables and 60 constraints) and achieves average savings of 34.4% in IS and 24.9% in TS. Compared to these methods, our approach has been tested more extensively, covering instances with up to 200 variables and 400 constraints. It outperforms the aforementioned techniques, achieving 46.5% average IS and 30.4% average TS.

The NETLIB benchmark results (Table 3) further confirm these findings on real-world problem structures. The proposed method achieves an average IS of 23.9% and TS of 13.3%, compared to 22.7% IS and 10.6% TS reported for the hybrid-LP method in Ref. [23]. Notably, both methods exhibit negative TS on several NETLIB instances (7 out of 19 for the proposed method vs. 8 for hybrid-LP), reflecting the inherent challenge of overcoming initialization overhead on problems with complex degeneracy patterns. Nevertheless, on well-conditioned problems such as SC50A, SC205, and LOTFI, the proposed method achieves TS values exceeding 40%, demonstrating its strong potential when the problem structure aligns with the geometric assumptions of the method.

Modern simplex implementations in commercial solvers typically include several preprocessing and initialization techniques, such as presolve reductions and crash basis heuristics. Presolve reductions simplify the problem by removing redundant constraints and fixing variables, while crash procedures select an initial basis by identifying columns that are likely to remain basic at optimality, typically using triangularity or angle-based criteria. The proposed geometric initialization strategy addresses a different aspect of the problem: improving the quality of the starting vertex through interior exploration of the feasible region. Therefore, the proposed method is complementary to these techniques rather than a replacement. In practical implementations, presolve reductions could first simplify the LP model, after which the proposed initialization strategy could be applied before the standard simplex iterations begin.

6. CONCLUSION

This paper introduced a geometry-driven initialization strategy designed to enhance the efficiency of the simplex method for linear programming problems. By performing interior moves from a feasible point toward a boundary point near optimality and subsequently identifying the nearest vertex, the proposed method provides a high-quality starting point for Phase II of the simplex method. The approach requires no modification to the simplex algorithm itself and maintains full compatibility with existing solvers.

Extensive numerical experiments demonstrated that the proposed strategy significantly reduces the number of iterations and total computation time when compared to other existing initialization techniques. The performance improvements are especially prominent in large-scale problems. On test problems with up to 200 variables and 400 constraints, the method achieved an average iteration reduction of 46.5% and an average time reduction of 30.4% relative to the classical simplex method.

Future work may include extending this strategy to handle degenerate cases more robustly, integrating the method with dual simplex variants, incorporating machine learning-enhanced search direction selection, and exploring its application in structured LP problems arising in real-world optimization contexts.

Statement of Research and Publication Ethics

The study is complied with research and publication ethics.

Artificial Intelligence (AI) Contribution Statement

AI tools (ChatGPT, OpenAI) were used for language editing and clarity improvement. All content was reviewed and approved by the author, who takes full responsibility for the manuscript.

REFERENCES

[1] V. Klee and G. J. Minty, “How good is the simplex algorithm?,” in *Inequalities III*, O. Shisha, Ed. New York: Academic Press, 1972, pp. 159–175.

[2] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, 2nd ed. New York: Springer, 2001.

[3] K.-H. Borgwardt, "The average number of steps required by the simplex method is polynomial," *Zeitschrift für Operations Research*, vol. 26, no. 1, pp. 157–177, 1982.

[4] J. Hall and Q. Huangfu, "A high performance dual revised simplex solver," in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, Eds. Berlin, Heidelberg: Springer, 2012, pp. 143–151.

[5] J. B. E. Etoa, "New optimal pivot rule for the simplex algorithm," *Advances in Pure Mathematics*, vol. 6, pp. 647–658, 2016, doi: 10.4236/apm.2016.610054.

[6] N. Ploskas and N. Samaras, "Pivoting rules for the revised simplex algorithm," *Yugoslav Journal of Operations Research*, vol. 24, pp. 321–332, 2014, doi: 10.2298/YJOR140228016P.

[7] H. Nabli, "An overview on the simplex algorithm," *Applied Mathematics and Computation*, vol. 210, no. 2, pp. 479–489, 2009, doi: 10.1016/j.amc.2009.01.013.

[8] T. Illés and T. Terlaky, "Pivot versus interior point methods: Pros and cons," *European Journal of Operational Research*, vol. 140, no. 2, pp. 170–190, 2002, doi: 10.1016/S0377-2217(02)00061-9.

[9] R. G. Bland, "New finite pivoting rules for the simplex method," *Mathematics of Operations Research*, vol. 2, no. 2, pp. 103–107, 1977, doi: 10.1287/moor.2.2.103.

[10] P.-Q. Pan, "Practical finite pivoting rules for the simplex method," *Operations Research Spectrum*, vol. 12, no. 4, pp. 219–225, 1990, doi: 10.1007/BF01721801.

[11] C. E. Lemke, "The dual method of solving the linear programming problem," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 36–47, 1954, doi: 10.1002/nav.3800010107.

[12] K. Paparrizos, N. Samaras, and G. Stephanides, "A new efficient primal dual simplex algorithm," *Computers & Operations Research*, vol. 30, no. 9, pp. 1383–1399, 2003, doi: 10.1016/S0305-0548(02)00077-1.

[13] A. Koberstein, "Progress in the dual simplex algorithm for solving large scale LP problems: techniques for a fast and stable implementation," *Computational Optimization and Applications*, vol. 41, no. 2, pp. 185–204, 2008, doi: 10.1007/s10589-008-9207-4.

[14] G. Mitra, M. Tamiz, and J. Yadegar, "Experimental investigation of an interior search method within a simplex framework," *Communications of the ACM*, vol. 31, no. 12, pp. 1474–1482, 1988, doi: 10.1145/53580.214953.

[15] H. A. Eiselt and C.-L. Sandblom, "External pivoting in the simplex algorithm," *Statistica Neerlandica*, vol. 39, no. 4, pp. 327–341, 1985, doi: 10.1111/j.1467-9574.1985.tb01152.x.

[16] H. A. Eiselt and C.-L. Sandblom, "Experiments with external pivoting," *Computers & Operations Research*, vol. 17, no. 4, pp. 325–332, 1990, doi: 10.1016/0305-0548(90)90010-5.

[17] Y. Fathi and K. G. Murty, "Computational behavior of a feasible direction method for linear programming," *European Journal of Operational Research*, vol. 40, no. 3, pp. 322–328, 1989, doi: 10.1016/0377-2217(89)90424-4.

[18] N. V. Stojković and P. S. Stanimirović, "Two direct methods in linear programming," *European Journal of Operational Research*, vol. 131, no. 2, pp. 417–439, 2001, doi: 10.1016/S0377-2217(00)00083-7.

[19] H. V. Junior and M. P. E. Lins, "An improved initial basis for the simplex algorithm," *Computers & Operations Research*, vol. 32, no. 8, pp. 1983–1993, 2005, doi: 10.1016/j.cor.2004.01.002.

[20] J.-F. Hu, "A note on 'An improved initial basis for the simplex algorithm'," *Computers & Operations Research*, vol. 34, no. 11, pp. 3397–3401, 2007, doi: 10.1016/j.cor.2006.02.004.

[21] H. Luh and R. Tsaih, "An efficient search direction for linear programming problems," *Computers & Operations Research*, vol. 29, no. 2, pp. 195–203, 2002, doi: 10.1016/S0305-0548(00)00069-1.

[22] B. J. Chaderjian and T. Gao, "Comments on 'An efficient search direction for linear programming problems' by H. Luh and R. Tsaih," *Computers & Operations Research*, vol. 30, no. 8, pp. 1255–1258, 2003, doi: 10.1016/S0305-0548(02)00110-7.

[23] C. Al-Najjar and B. Malakooti, "Hybrid-LP: Finding advanced starting points for simplex, and pivoting LP methods," *Computers & Operations Research*, vol. 38, no. 2, pp. 427–434, 2011, doi: 10.1016/j.cor.2010.07.001.

[24] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Belmont, MA: Athena Scientific, 1997.

[25] The MathWorks, Inc., *MATLAB Version 24.1.0 (R2024a)*. [Online]. Available: https://www.mathworks.com

[26] NETLIB LP test problems. [Online]. Available: https://www.netlib.org/lp/

APPENDIX

Algorithm 1 Phase A

Input: c, A, b, D, x_0, ε_{tol}, $k_{\max}$
Set $k \leftarrow 0$
while termination conditions are not satisfied **do**
  if $k = 0$ **then**
    Compute the orthonormal basis Z for null(D)
    Compute the initial search direction d_0 ▷ Equation 12
  else
    Identify the active inequality constraints $\mathcal{I}_k$ ▷ Equation 6
    Compute the orthonormal basis N_k for null(P_k)
    while N_k is empty **do**
      Compute the Lagrange multipliers λ_k ▷ Equation 32
      Set λ_k^I to the entries of λ_k corresponding to $A_{\mathcal{I}_k}$
      if all entries of $\lambda_k^I \geq 0$ **then**
        return x_k ▷ Optimal solution found
      else
        Drop the constraint in $\mathcal{I}_k$ with the most negative λ_k^I
        Recompute the orthonormal basis N_k
      end if
    end while
    Compute the limit direction d_k^ℓ ▷ Equation 15
    Compute the angle θ_k between d_k^ℓ and $-d_{k-1}$ ▷ Equation 17
    Compute the upper bound $\delta_k^\star$ for δ_k ▷ Equation 29
    Set the perturbation angle $\delta_k \leftarrow \delta_k^\star/2$
    Compute the interpolated search direction d_k ▷ Equation 19
  end if
  Compute the maximal step size α_k ▷ Equation 10
  Update the solution: $x_{k+1} \leftarrow x_k + \alpha_k d_k$ ▷ Equation 4
  Evaluate the objective function at x_{k+1}
  Set $k \leftarrow k + 1$
end while
Output: Near-optimal boundary point x_k

Algorithm 2 Phase B

Input: Near-optimal boundary point x_k, c, A, b, D
while number of independent active constraints $< n$ **do**
  Identify the active inequality constraints $\mathcal{I}_k$ ▷ Equation 6
  Compute the orthonormal basis N_k for null(P_k)
  Compute d_k^ℓ and set $d_k \leftarrow d_k^\ell$ ▷ Equation 15
  Compute the maximal step size α_k ▷ Equation 10
  while $\alpha_k \leq 0$ **do**
    Generate a random vector r with $\|r\| \ll \|c\|$
    Compute the modified direction d_k ▷ Equation 34
    Recompute the step size α_k ▷ Equation 10
  end while
  Update the solution: $x_{k+1} \leftarrow x_k + \alpha_k d_k$
  Set $k \leftarrow k + 1$
end while
Output: Initial vertex x_k