Scientific Research Publishing

American Journal of Operations Research, 2018, 8, 112-131
http://www.scirp.org/journal/ajor
ISSN Online: 2160-8849
ISSN Print: 2160-8830

The Sliding Gradient Algorithm for Linear Programming

Hochung Lui, Peizhuang Wang

College of Intelligence Engineering and Mathematics, Liaoning Technical University, Fuxin, China
Email: hcluieip@gmail.com

How to cite this paper: Lui, H.C. and Wang, P.Z. (2018) The Sliding Gradient Algorithm for Linear Programming. *American Journal of Operations Research*, **8**, 112-131.
https://doi.org/10.4236/ajor.2018.82009

Received: February 26, 2018
Accepted: March 27, 2018
Published: March 30, 2018

Abstract

The existence of strongly polynomial algorithm for linear programming (LP) has been widely sought after for decades. Recently, a new approach called Gravity Sliding algorithm [1] has emerged. It is a gradient descending method whereby the descending trajectory slides along the inner surfaces of a polyhedron until it reaches the optimal point. In R^3, a water droplet pulled by gravitational force traces the shortest path to descend to the lowest point. As the Gravity Sliding algorithm emulates the water droplet trajectory, it exhibits strongly polynomial behavior in R^3. We believe that it could be a strongly polynomial algorithm for linear programming in R^n too. In fact, our algorithm can solve the Klee-Minty deformed cube problem in only two iterations, irrespective of the dimension of the cube. The core of gravity sliding algorithm is how to calculate the projection of the gravity vector $\boldsymbol{g}$ onto the intersection of a group of facets, which is disclosed in the same paper [1]. In this paper, we introduce a more efficient method to compute the gradient projections on complementary facets, and rename it the Sliding Gradient algorithm under the new projection calculation.

Keywords

Linear Programming, Mathematical Programming, Complexity Theory, Optimization

1. Introduction

The simplex method developed by Dantiz [2] has been widely used to solve many large-scale optimizing problems with linear constraints. Its practical performance has been good and researchers have found that the expected number of iterations exhibits polynomial complexity under certain conditions [3] [4] [5] [6]. However, Klee and Minty in 1972 gave a counter example showing that its

worst case performance is $\mathcal{O}\left(2^n\right)$ [7]. Their example is a deliberately constructed deformed cube that exploits a weakness of the original simplex pivot rule, which is sensitive to scaling [8]. It is found that, by using a different pivot rule, the Klee-Minty deformed cube can be solved in one iteration. But for all known pivot rules, one can construct a different deformed cube that requires exponential number of iterations to solve [9] [10] [11]. Recently, the interior point method [12] has been gaining popularity as an efficient and practical LP solver. However, it was also found that such method may also exhibit similar worse case performance by adding a large set of redundant inequalities to the Klee-Minty cube [13].

Is it possible to develop a strongly polynomial algorithm to solve the linear programming problem, where the number of iterations is a polynomial function of only the number of constraints and the number of variables? The work by Barasz and Vempala shed some light in this aspect. Their AFFINE algorithm [14] takes only $\mathcal{O}\left(n^2\right)$ iterations to solve a broad class of deformed products defined by Amenta and Ziegler [15] which includes the Klee-Minty cube and many of its variants.

In certain aspect, the Gravity Sliding algorithm [1] is similar to the AFFINE algorithm as it also passes through the interior of the feasible region. The main difference is in the calculation of the next descending vector. In the gravity falling approach, a gravity vector is first defined (see Section 3.1 for details). This is the principle gradient descending direction where other descending directions are derived from it. In each iteration, the algorithm first computes the descending direction, then it descends from this direction until it hits one or more facets that forms the boundary of the feasible region. In order not to penetrate the feasible region, the descending direction needs to be changed. The trajectory is likened a water droplet falling from the sky but is blocked by linear planar structures (e.g. the roof top structure of a building) and needs to slide along the structure. The core of gravity sliding algorithm is how to calculate the projection of the gravity vector $\boldsymbol{g}$ onto the intersection of a group of facets. This projection vector lies on the intersection of the facets and hence lies on the null space defined by these facets. Conventional approach is to compute the null space first and then find the projection of $\boldsymbol{g}$ onto this null space. An alternative approach is disclosed in [1] which operates directly from the subspace formed by the intersecting facets. This direct approach is more suitable to the Gravity Sliding algorithm. In this paper, we further present an efficient method to compute the gradient projections on complementary facets and also introduce the notion of selecting the steepest descend projection among a set of candidates. With these refinements, we rename the Gravity Sliding algorithm as the Sliding Gradient algorithm. We have implemented our algorithm and tested it on the Klee-Minty cube. We observe that it can solve the Klee-Minty deformed cube problem in only two iterations, irrespective of the dimension of the cube.

This paper is organized as follows: Section 2 gives an overview of the

Cone-Cutting Theory [16], which is the intuitive background of the Gravity Sliding algorithm. Section 3 discusses the Sliding Gradient algorithm in details. The pseudo-code of this algorithm is summarized in Section 4 and Section 5 gives a walk-through of this algorithm using the Klee-Minty as an example. This section also discusses the practical implementation issues. Finally, Section 6 discuss about future work.

2. Cone-Cutting Principle

The cone-cutting theory [16] offers a geometric interpretation of a set of inequality equations. Instead of considering the full set constraint equations in a LP problem, the cone-cutting theory enables us to consider a subset of equations, and how an additional constraint will shape the feasible region. The geometric insight forms the basis of our algorithm development.

2.1. Cone-Cutting Principle

In an m-dimension space $\mathbb{R}^m$, a hyperplane $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\tau}=c$ cuts $\mathbb{R}^m$ into two half spaces. Here $\boldsymbol{\tau}$ is the normal vector of the hyperplane and c is a constant. We denote the positive half space $\left\{\boldsymbol{y}\,|\,\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\tau}\geq c\right\}$ the *accepted zone* of the hyperplane and the negative half space where $\left\{\boldsymbol{y}\,|\,\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\tau}<c\right\}$ is *rejected zone.* Note that the normal vector $\boldsymbol{\tau}$ points to accepted zone area and we call the hyperplane with such orientation a *facet* $\alpha:(\boldsymbol{\tau},c)$. When there are m facets in $\mathbb{R}^m$ and $\left\{\boldsymbol{\tau}_1,\boldsymbol{\tau}_2,\cdots,\boldsymbol{\tau}_m\right\}$ are linear independent, this set of linear equations has a unique solution which is a point $\boldsymbol{V}$ in $\mathbb{R}^m$. Geometrically, $\left\{\alpha_1,\alpha_2,\cdots,\alpha_m\right\}$ form a cone and $\boldsymbol{V}$ is the vertex of the cone. We now give a formal definition of a cone, which is taken from [1].

Definition 1. Given m hyperplanes in $\mathbb{R}^m$, with rank $r\left(\alpha_1,\cdots,\alpha_m\right)=m$ and intersection V, $C=C\left(\boldsymbol{V};\alpha_1,\cdots,\alpha_m\right)=\alpha_1\bigcap\cdots\bigcap\alpha_m$ is called a *cone* in $\mathbb{R}^m$. The area $\left\{\boldsymbol{y}\,|\,\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\tau}_i\geq c_i\left(i=1,2,\cdots,m\right)\right\}$ is called the *accepted zone* of C. The point V is the *vertex* and α_j is the *facet plane*, or simply the *facet* of C.

A cone C also has m edge lines. They are formed by the intersection of $(m-1)$ facets. Hence, a cone can also be defined as follows.

Definition 2. Given m rays $R_j=\left\{\boldsymbol{V}+t\boldsymbol{r}_j\,|\,0\leq t<+\infty\right\}\left(j=1,\cdots,m\right)$ shooting from a point V with rank $r\left(\boldsymbol{r}_1,\cdots,\boldsymbol{r}_m\right)=m$, $C=C\left(\boldsymbol{V};\boldsymbol{r}_1,\cdots,\boldsymbol{r}_m\right)=c\left[R_1,\cdots,R_m\right]$, the convex closure of m rays is called a *cone* in $\mathbb{R}^m$. R_j is the *edge*, $\boldsymbol{r}_j$ the *edge direction*, and $R_j^+=\left\{\boldsymbol{V}+t\boldsymbol{r}_j\,|\,\infty<t<+\infty\right\}$ the *edge line* of the cone C.

The two definitions are equivalent. Furthermore, P.Z. Wang [11] has observed that R_i^+ and α_i are opposite to each other for $i=1,\cdots,m$. Edge-line R_i^+ is the intersection of all C-facets except α_i, while facet α_i is bounded by all C-edges except R_i^+. This is the duality between facets and edges. For $i=1,\cdots,m,\left\{\boldsymbol{\tau}_i,R_i\right\}$ is called a *pair* of cone C.

It is obvious that $\boldsymbol{r}_j^{\mathrm{T}}\boldsymbol{\tau}_i=0$ (for $i\neq j$) since $\boldsymbol{r}_j$ lies on α_i. Moreover, we have

$$\boldsymbol{r}_i^{\mathrm{T}}\boldsymbol{\tau}_i\geq 0\left(\text{for } i=1,\cdots,m\right) \tag{1}$$

2.2. Cone Cutting Algorithm

Consider a linear programming (LP) problem and its dual:

$$\text{(Primary):}\quad \max\left\{\tilde{\boldsymbol{c}}^{\mathrm{T}}\boldsymbol{x} \mid A\boldsymbol{x} \le \boldsymbol{b};\ \boldsymbol{x} \ge 0\right\} \tag{2}$$

$$\text{(Dual):}\quad \min\left\{\boldsymbol{y}^{\mathrm{T}}\boldsymbol{b} \mid \boldsymbol{y}^{\mathrm{T}}A \ge \tilde{c};\ \boldsymbol{y} \ge 0\right\} \tag{3}$$

In the following, we focus on solving the dual LP problem. The standard simplex tableau can be obtained by appending an $m \times m$ identity matrix $I_{m\times m}$ which represents the slack variables as shown below:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} & 1 & \cdots & 0 & b_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & \cdots & a_{mn} & 0 & \cdots & 1 & b_m \\ \tilde{c}_1 & \cdots & \tilde{c}_n & 0 & \cdots & 0 & 0 \end{bmatrix}$$

We can construct a facet tableau whereby each column is a facet denoted as $\alpha_j : (\boldsymbol{\tau}_j, c_j)$, where $\boldsymbol{\tau}_i = (a_{1i}, a_{2i}, \cdots, a_{mi})^{\mathrm{T}}$ and

$$c_i = \begin{cases} \tilde{c}_i & \text{for } 1 \le i \le n \\ 0 & \text{for } n < i \le m+n \end{cases} \tag{4}$$

The facet tableau is depicted as follow. The last column $(b_1, b_2, \cdots, b_m, 0)^{\mathrm{T}}$ is not represented in this tableau.

$$\begin{array}{cccc} \alpha_1 & \alpha_2 & \cdots & \alpha_{m+n} \\ \left[\begin{array}{cccc} \boldsymbol{\tau}_1 & \boldsymbol{\tau}_2 & \cdots & \boldsymbol{\tau}_{m+n} \\ \hline c_1 & c_2 & \cdots & c_{m+n} \end{array}\right] \end{array}$$

When a cone $C = C(\boldsymbol{V}; \alpha_1, \cdots, \alpha_m) = C(\boldsymbol{V}; \boldsymbol{r}_1, \cdots, \boldsymbol{r}_m)$ is intersected by another facet α_j, the i^{th} edge of the cone is intersected by α_j at certain point $\boldsymbol{q}_{ij}$. We call α_j cuts the cone C and the cut points $\boldsymbol{q}_{ij}$ can be obtained by the following equations:

$$\boldsymbol{q}_{ij} = \boldsymbol{V} + t_i \boldsymbol{r}_i \quad \text{where } t_i = \left(c_i - \boldsymbol{V}^{\mathrm{T}}\boldsymbol{\tau}_j\right) \Big/ \boldsymbol{r}_i^{\mathrm{T}}\boldsymbol{\tau}_j \tag{5}$$

The intersection is called *real* if $t_i \ge 0$ and *fictitious* if $t_i < 0$. Cone cutting greatly alters the accepted zone, as can be seen from the simple 2-dimension example as shown in Figures 1(a)-(e). In 2-dimension, a facet $\alpha : (\boldsymbol{\tau}, c)$ is a line. The normal vector $\boldsymbol{\tau}$ is perpendicular to this line and points to the accepted zone of this facet. Furthermore, a cone is formed by two non-parallel facets in 2-dimension. Figure 1(a) shows such a cone $C(\boldsymbol{V}; \alpha_1, \alpha_2)$. The accepted zone of the cone is the intersection of the two accepted zones of facets α_1 and α_2. This is represented by the shaded area A in Figure 1(a). In Figure 1(b), a new facet α_3 intersects the cone at two cut points $\boldsymbol{q}_{13}$ and $\boldsymbol{q}_{23}$. They are both real cut points. Since the arrow of normal vector $\boldsymbol{\tau}_3$ points to the same general direction of the cone, $\boldsymbol{V}$ lies in the rejected zone of α_3 and we say α_3 rejects $\boldsymbol{V}$. Moreover, the accepted zone of α_3 intersects with the accepted zone of the cone so that the overall accepted zone is reduced to the shaded area marked as B. In Figure 1(c), $\boldsymbol{\tau}_3$ points to the opposite direction. α_3 accepts $\boldsymbol{V}$ and the overall

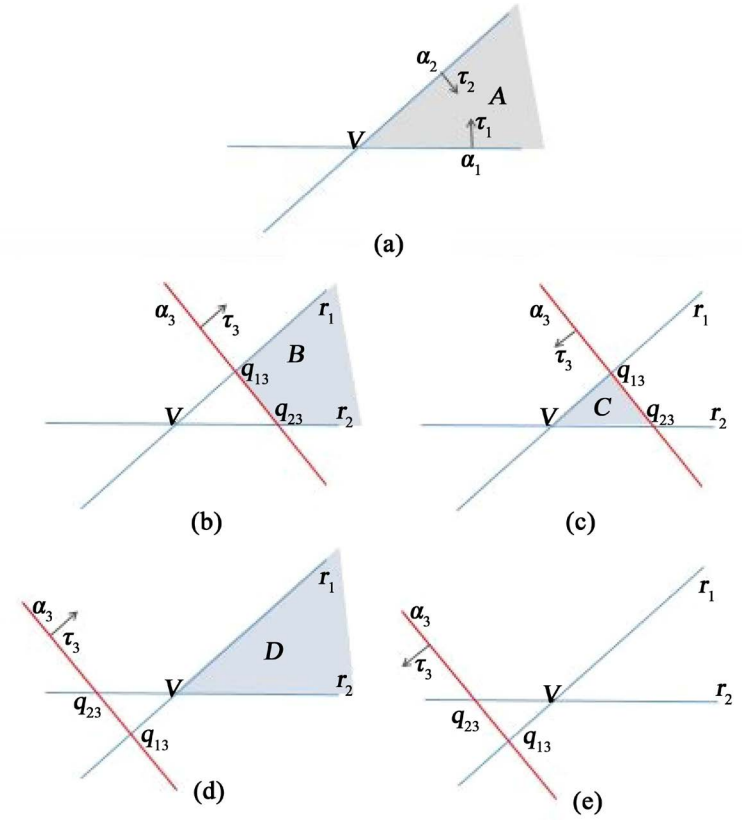

Figure 1. Accepted zone area of a cone and after it is cut by a facet.

accepted zone is confined to the area marked as C. As the dual feasible region $\mathcal{D}$ of a LP problem must satisfy all the constraints, it must lie within area C. In Figure 1(d), α_3 cuts the cone at two fictitious points. Since τ_3 points to the same direction of the cone, V is accepted by α_3. However, the accepted zone of α_3 covers that of the cone. As a result, α_3 does not contribute to any reduction of the overall accepted zone area, and so it can be deleted for further consideration without affecting the LP solution. In Figure 1(e), τ_3 points to the opposite direction of the cone. The intersection between the accepted zone of α_3 and that of the cone is an empty set. This means that the dual feasible region $\mathcal{D}$ is empty and the LP is infeasible. This is actually one of the criteria that can be used for detecting infeasibility.

Based on this cone-cutting idea, P.Z. Wang [16] [17] have developed a cone-cutting algorithm to solve the dual LP problem. Each cone is a combination of m facets selected from $(m+n)$ choices. Let Δ denotes the index set of facets of C, (*i.e.* if $\Delta(i)=j$, then $\tau_{\Delta(i)}=\tau_j$). The algorithm starts with an initial coordinate cone C_o, then finds a facet α_{in} to replace one of the existing facet α_{out} thus forming a new cone. This process is repeated until an optimal point is found. The cone-cutting algorithm is summarized in Table 1 below.

This algorithm finds a facet that rejects V the least as the cutting facet in steps 2 and 3. This facet cuts the edges of the cone at m points. In step 4 and 5, the real cut point q_{I^*} that is closest to the vertex V is identified. This becomes the

Table 1. Cone-cutting algorithm.

steps	**Input:** A, b and $\tilde{c}$ ***Output**: either V as the optimal point & $V^{\mathrm{T}}b$ as optimal value or declare the LP problem as infeasible.*
0	$\boldsymbol{V} = \boldsymbol{V}_0 = (0,0,\cdots,0)$ % the coordinate cone; $\Delta(j) = n + j \ \ for \ \ j = 1,\cdots,m$ $r_1 = (1,0,\cdots,0), r_2 = (0,1,\cdots,0),\cdots, r_m = (0,0,\cdots,1)$;
1	**while** (true)
2	% For all facets not in C & reject V, find one that rejects V the least. $\bar{\Delta} = \{1:(m+n)\} \setminus \Delta$; % $\bar{\Delta}$ are facets not in C $j^* = \arg\min_j \left\{ e_j \mid e_j = \left(V^{\mathrm{T}} \tau_{\bar{\Delta}(j)} - c_{\bar{\Delta}(j)} \right), j \in \bar{\Delta} \right\}$
3	**if** $e_{j^*} \geq 0$ return ($\boldsymbol{V}$ & $\boldsymbol{V}^{\mathrm{T}}\boldsymbol{b}$) as optimal vertex & optimal value **else** $J^* = \bar{\Delta}(j^*)$; % j^* is the facet index to enter
4	$t_i = \dfrac{c_{J^*} \boldsymbol{V}^{\mathrm{T}} \boldsymbol{\tau}_{J^*}}{\boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{\tau}_{J^*}}; i = 1,\cdots,m$; $i = 1,\cdots,m$ }; $\boldsymbol{q}_i = \boldsymbol{V} + t_i \boldsymbol{r}_i$;
5	**if** $(t_i < 0 \ \forall i = 1,\cdots,m)$ **return**("LP is infeasible") **else** % For all the real cuts $\boldsymbol{q}_i$, find the $\boldsymbol{q}_{I^*}$ that is closest to $\boldsymbol{V}$ $I^* = \arg\min_i \{ \boldsymbol{q}_i^{\mathrm{T}} \boldsymbol{b} \mid t_i > 0 \}$ % I^* is the facet index to leave
6	% Form new cone C_{k+1} by updating $\boldsymbol{V}_{k+1}$; edge vectors and facets $V_{k+1} = \boldsymbol{q}_{I^*}$; $r_i = \begin{cases} r_i & i = I^* \\ sign(t_i)[\boldsymbol{q}_i - V_{k+1}] & i \neq I^* \end{cases}$ $\Delta(I^*) = J^*$;
7	**end**

vertex of a new cone. This new cone retains all the facets of the original cone except that the cutting facet replaces the facet corresponding to the edge I^*. Yet the edge I^* is retained but the rest of the edges must be recomputed as shown in step 6. Amazingly, P.Z. Wang shows that when $\boldsymbol{b} > 0$, this algorithm produces exactly the same result as the original simplex algorithm proposed by Dantz [2]. Hence, the cone-cutting theory offers a geometric interpretation of the simplex method. More significantly, it inspires the authors to explore new approach to tackle the LP problem.

3. Sliding Gradient Algorithm

Expanding on the cone-cutting theory, the Gravity Sliding Algorithm [1] was developed to find the optimal solution of the LP problem from a point within the feasible region $\mathcal{D}$. Since then, several refinements have been made and they are presented in the following sections.

3.1. Determining the General Descending Direction

The feasible region $\mathcal{D}$ is a convex polyhedron formed by constraints

$\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\tau}_j \geq c_j; 1 \leq j \leq n+m\right)$, and the optimal feasible point is at one of its vertices. Let $\Omega = \left\{\boldsymbol{V}_i \,|\, \boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{\tau}_j \geq c_j;\, j = 1, \cdots, n+m\right\}$ be the set of feasible vertices. The dual LP problem (3) can then be stated as: $\min\left\{\boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{b} \,|\, \boldsymbol{V}_i \in \Omega\right\}$. As $\boldsymbol{V}_i^{\mathrm{T}}\boldsymbol{b}$ is the inner-product of vertex $\boldsymbol{V}_i$ and $\boldsymbol{b}$, the optimal vertex $\boldsymbol{V}^*$ is the vertex that yields the lowest inner-product value. Thus we can set the principle descending direction $\boldsymbol{g}_0$ to be the opposite of the $\boldsymbol{b}$ vector (*i.e.* $\boldsymbol{g}_0 = -\boldsymbol{b}$) and this is referred to as the gravity vector. The descending path then descends along this principle direction inside $\mathcal{D}$ until it reaches the lowest point in $\mathcal{D}$ viewed along the direction of $\boldsymbol{b}$. This point is then the optimal vertex $\boldsymbol{V}^*$.

3.2. Circumventing Blocking Facets

The basic principle of the new algorithm can be illustrated in Figure 2. Notice that in 2-dim, a facet is a line. In this figure, these facets (lines) form a closed polyhedron which is the dual feasible region $\mathcal{D}$. Here the initial point $\boldsymbol{P}_0$ is inside $\mathcal{D}$. From $\boldsymbol{P}_0$, it attempts to descend along the $\boldsymbol{g}_0 = -\boldsymbol{b}$ direction. It can go as far as $\boldsymbol{P}_1$ which is the point of intersection between the ray $\boldsymbol{R} = \boldsymbol{P}_0 + t\boldsymbol{g}_0$ and the facet α_1. In essence, α_1 is blocking this ray and hence it is called the blocking facet relative to this ray. In order not to penetrate $\mathcal{D}$, the descending direction needs to change from $\boldsymbol{g}_0$ to $\boldsymbol{g}_1$ at $\boldsymbol{P}_1$, and slides along $\boldsymbol{g}_1$ until it hits the other blocking facet α_2 at $\boldsymbol{P}_2$. Then it needs to change course again and slides along the direction $\boldsymbol{g}_2$ until it hits $\boldsymbol{P}_3$. In this figure, $\boldsymbol{P}_3$ is the lowest point in this dual feasible region $\mathcal{D}$ and hence it is the optimal point $\boldsymbol{V}^*$.

It can be observed from Figure 2 that $\boldsymbol{g}_1$ is the projection of $\boldsymbol{g}_0$ onto α_1 and $\boldsymbol{g}_2$ is the projection of $\boldsymbol{g}_0$ onto α_2. Thus from $\boldsymbol{P}_1$, the descending path slides along α_1 to reach $\boldsymbol{P}_2$ and then slides along α_2 to reach $\boldsymbol{P}_3$. Hence we call this algorithm Sliding Gradient Algorithm. The basic idea is to compute the new descending direction to circumvent the blocking facets, and advance to find the next one until it reaches the bottom vertex viewed along the direction of $\boldsymbol{b}$.

Let σ_t denotes the set of blocking facets at the t^{th} iteration. From an initial point $\boldsymbol{P}_0$ and a gradient descend vector $\boldsymbol{g}_0$, the algorithm iteratively performs the following steps:

1) compute a gradient direction $\boldsymbol{g}_t$ based on σ_t. In this example, the initial set

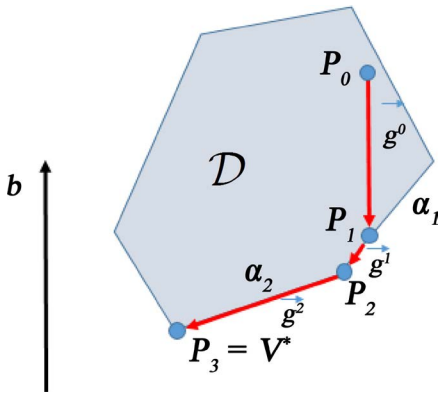

Figure 2. Sliding gradient illustration.

of blocking facets σ_0 is empty and $\boldsymbol{g}_0 = -\boldsymbol{b}$.

2) move $\boldsymbol{P}_t$ to $\boldsymbol{P}_{t+1}$ along $\boldsymbol{g}_t$ where $\boldsymbol{P}_{t+1}$ is a point at the first blocking facet.

3) Incorporate the newly encountered blocking facet to σ_t to form σ_{t+1}.

4) go back to step 1.

The algorithm stops when it cannot find any direction to descend in step (1). This is discussed in details in Section 3.6 where a formal stopping criterion is given.

3.3. Minimum Requirements for the Gradient Direction g_t

For the first step, the gradient descend vector $\boldsymbol{g}_t$ needs to satisfy the following requirements.

Proposition 1. $\boldsymbol{g}_t$ must satisfy $(\boldsymbol{g}_t)^{\mathrm{T}} \boldsymbol{g}_0 \geq 0$ so that the dual objective function $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{b}$ will be non-increasing when $\boldsymbol{y}$ move from $\boldsymbol{P}_t$ to $\boldsymbol{P}_{t+1}$ along the direction of $\boldsymbol{g}_t$.

Proof. Since $(\boldsymbol{g}_t)^{\mathrm{T}} \boldsymbol{g}_0 \geq 0$, $\boldsymbol{g}_t$ aligns to the principle direction of $\boldsymbol{g}_0$. As $\boldsymbol{P}_{t+1} = \boldsymbol{P}_t + t\boldsymbol{g}_t$, $\boldsymbol{P}_{t+1}$ moves along the principle direction of $\boldsymbol{g}_0$ when $t > 0$.

Since $\boldsymbol{P}_{t+1}^{\mathrm{T}}\boldsymbol{b} = \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{b} + t(\boldsymbol{g}_t)^{\mathrm{T}}\boldsymbol{b} = \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{b} - t(\boldsymbol{g}_t)^{\mathrm{T}}\boldsymbol{g}_0$, $\boldsymbol{P}_{t+1}^{\mathrm{T}}\boldsymbol{b} \leq \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{b}$ when $(\boldsymbol{g}_t)^{\mathrm{T}}\boldsymbol{g}_0 \geq 0$. **END**

This means that if $(\boldsymbol{g}_t)^{\mathrm{T}}\boldsymbol{g}_0 \geq 0$, then $\boldsymbol{P}_{t+1}$ is "lower than" $\boldsymbol{P}_t$ when viewed along the $\boldsymbol{b}$ direction.

Proposition 2. If $\boldsymbol{P}_0 \in \mathcal{D}$, $\boldsymbol{g}_t$ must satisfy $\left(\tau_{\sigma_t(j)}\right)^{\mathrm{T}} \boldsymbol{g}_t \geq 0$ for all $j \in \sigma_t$ to ensure that $\boldsymbol{P}_{t+1}$ remains dual feasible (*i.e.* $P_{t+1} \in \mathcal{D}$).

Proof. If for some *j*, $\left(\tau_{\sigma_t(j)}\right)^{\mathrm{T}} \boldsymbol{g}_t < 0$, this means that $\boldsymbol{g}_t$ is in the opposite direction of the normal vector of facet $\alpha_{\sigma_t(j)}$ so a ray $\boldsymbol{Q} = \boldsymbol{P}_t + t\boldsymbol{g}_t$ will eventually penetrate this facet for certain positive value of *t*. This means that $\boldsymbol{Q}$ will be rejected by $\alpha_{\sigma_t(j)}$ and hence $\boldsymbol{Q}$ is no longer a dual feasible point. **END**

3.4. Maximum Descend in Each Iteration

To ensure that $\boldsymbol{P}_{t+1} \in \mathcal{D}$, we need to make sure that it won't advance too far. The following proposition stipulates the requirement.

Proposition 3. Assuming that $\mathcal{D}$ is non-empty and $\boldsymbol{P}_0 \in \mathcal{D}$. If $\boldsymbol{g}_t$ satisfies Propositions 1 and 2; and not all $\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j = 0$ for $j = 1, \cdots, m+n$, then $\boldsymbol{P}_{t+1} \in \mathcal{D}$ provided that the next point $\boldsymbol{P}_{t+1}$ is determined according to (6) below:

$$\boldsymbol{P}_{t+1} = \boldsymbol{P}_t + t_{j^*}\boldsymbol{g}_t \tag{6}$$

where $j^* = \arg\min_j \left\{ t_j \,|\, t_j = \dfrac{c_j - \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j}; t_j > 0; j = 1, \cdots, m+n \right\}$.

Proof. The equation for a line passing through $\boldsymbol{P}$ along the direction $\boldsymbol{g}$ is $\boldsymbol{P} + t\boldsymbol{g}$. If this line is not parallel to the plane (*i.e.* $\boldsymbol{g}^{\mathrm{T}}\boldsymbol{\tau} \neq 0$), it will intersect a facet $\alpha : (\boldsymbol{\tau}, c)$ at a point $\boldsymbol{Q}$ according to the following equation:

$$\boldsymbol{Q} = \boldsymbol{P} + t\boldsymbol{g} \quad \text{where } t = \left(c - \boldsymbol{P}^{\mathrm{T}}\boldsymbol{\tau}\right) \Big/ \boldsymbol{g}^{\mathrm{T}}\boldsymbol{\tau}. \tag{7}$$

We call t the displacement from $\boldsymbol{P}$ to $\boldsymbol{Q}$. So $t_j = \dfrac{c_j - \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j}$ is the displacement from $\boldsymbol{P}_t$ to α_j. The condition $t_j > 0$ ensures that $\boldsymbol{P}_{t+1}$ moves along the direction $\boldsymbol{g}_t$ but not the opposite direction. j^* is the smallest of all the displacements thus α_{j^*} is the first blocking facet that is closest to $\boldsymbol{P}_t$.

To show that $\boldsymbol{P}_{t+1} \in \mathcal{D}$, we need to show $\boldsymbol{P}_{t+1}^{\mathrm{T}}\boldsymbol{\tau}_j - c_j \geq 0$ for $j = 1, \cdots, m+n$. Note that

$$\boldsymbol{P}_{t+1}^{\mathrm{T}}\boldsymbol{\tau}_j - c_j = \left(\boldsymbol{P}_t + t_{j^*}\boldsymbol{g}_t\right)^{\mathrm{T}}\boldsymbol{\tau}_j - c_j = \left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) + t_{j^*}\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j.$$

Since $\boldsymbol{P}_t \in \mathcal{D}$, $\left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) \geq 0$ for $j = 1, \cdots, m+n$, so we need to show that $t_{j^*}\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j \geq 0$ for $j = 1, \cdots, m+n$.

The displacements t_j can be split into two groups. For those displacements where $t_j < 0$, $\dfrac{c_j - \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j} = t_j < 0$ so $\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j = \left(c_j - \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j\right)/t_j = -\left(c_j - \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j\right)/k_1$, where $k_1 = -t_j$ is a positive constant. Since $t_{j^*} > 0$.

$$t_{j^*}\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j = -\frac{t_{j^*}}{k_1}\left(c_j - \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j\right) = k_2\left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) \geq 0 \quad \text{since } \boldsymbol{P}_t \in \mathcal{D} \ \ \& \ \ k_2 > 0.$$

For those displacements where $t_j \geq 0$, we have that t_{j^*} is the minimum of all t_j in this group. Let k_3 be the ratio between t_{j^*} and t_j. Obviously, $k_3 = \dfrac{t_{j^*}}{t_j} \leq 1$

$$\begin{aligned}
&\left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) + t_{j^*}\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j \\
&= \left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) + k_3 t_j \boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j = \left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) + k_3 \frac{c_j - \boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j}\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j \\
&= \left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) - k_3\left(\boldsymbol{P}_t^{\mathrm{T}}\boldsymbol{\tau}_j - c_j\right) \geq 0
\end{aligned}$$

So $\boldsymbol{P}_{t+1} \in \mathcal{D}$. **END**

If $\boldsymbol{g}_t^{\mathrm{T}}\boldsymbol{\tau}_j = 0$, $\boldsymbol{g}_t$ is parallel to α_j. Unless all facets are parallel to $\boldsymbol{g}_t$, Proposition 3 can still find the next descend point $\boldsymbol{P}_{t+1}$. If all facets are parallel to $\boldsymbol{g}_t$, this means that facets are linearly dependent with each other. The LP problem is not well formulated.

3.5. Gradient Projection

We now show that the projection of $\boldsymbol{g}_0$ onto the set of blocking facets σ_t satisfies the requirements of Proposition 1 and 2. Before we do so, we discuss the projection operations in subspace first.

3.5.1. Projection in Subspaces

Projection is a basic concept defined in vector space. Since we are only interested in the gradient descend direction of $\boldsymbol{g}_t$ but not the actual location of the projection, we can ignore the constant c in the hyperplane $\left\{\boldsymbol{y} \mid \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\tau} = c\right\}$. In other

words, we focus on the subspace $V(\tau)$ spanned by $\boldsymbol{\tau}$ and its null space $N(\tau)$ rather than the affine space spanned by the hyperplanes.

Let Y be the vector space in $\mathbb{R}^m$, $V(\tau)=\{\boldsymbol{y} \mid \boldsymbol{y}=t\boldsymbol{\tau}; t\in\mathbb{R}\}$ and its corresponding null space is $N(\tau)=\{\boldsymbol{x} \mid \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}=0 \text{ for } \boldsymbol{x}\in Y \text{ and } \boldsymbol{y}\in V(\tau)\}$. Extending to k hyperplanes, we have $V(\tau_1,\tau_2,\cdots,\tau_k)=\left\{\boldsymbol{y} \mid \boldsymbol{y}=\sum_j^k t_j\boldsymbol{\tau}_j; t_j\in R\right\}$ and the null space is $N(\tau_1,\tau_2,\cdots,\tau_k)=\{\boldsymbol{x} \mid \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}=0 \text{ for } \boldsymbol{x}\in Y \text{ and } \boldsymbol{y}\in V(\tau_1,\tau_2,\cdots,\tau_k)\}$. It can be shown that $N(\tau_1,\tau_2,\cdots,\tau_k)=N(\tau_1)\bigcap N(\tau_2)\bigcap\cdots\bigcap N(\tau_k)$. Since $V(\tau_1,\tau_2,\cdots,\tau_k)$ and $N(\tau_1,\tau_2,\cdots,\tau_k)$ are the orthogonal decomposition of the whole space Y, a vector $\boldsymbol{g}$ in $\mathbb{R}^m$ can be decomposed into two components: the projection of $\boldsymbol{g}$ onto $V(\tau_1,\tau_2,\cdots,\tau_k)$ and the projection of $\boldsymbol{g}$ onto $N(\tau_1,\tau_2,\cdots,\tau_k)$. We use the notation $g\downarrow_{[\tau_1,\tau_2,\cdots,\tau_k]}$ and $g\downarrow_{\alpha_1\cap\cdots\cap\alpha_k}$ to denote them and they are called *direct projection* and *null projection* respectively.

The following definition and theorem were first presented in [1] and is repeated here for completeness.

Let the set of all subspaces of $Y=\mathbb{R}^m$ be $\mathcal{N}$, and let $\mathcal{O}$ stand for 0-dim subspace, we now give an axiomatic definition of projection.

Definition 3. The projection defined on a vector space Y is a mapping

$$*\downarrow_{\#}: Y\times\mathcal{N}\to Y$$

where $*$ is a vector in Y, # is a subspace X in $\mathcal{N}$ satisfying that

(N.1) (Reflectivity).

For any $\boldsymbol{g}\in Y, \boldsymbol{g}\downarrow_Y=\boldsymbol{g}$;

(N.2) (Orthogonal additivity).

For any $\boldsymbol{g}\in Y$ and subspaces $X,Z\in\mathcal{N}$, if X and Z are orthogonal to each other, then $\boldsymbol{g}\downarrow_X+\boldsymbol{g}\downarrow_Z=\boldsymbol{g}\downarrow_{X+Z}$, where $X+Z$ is the direct sum of X and Z.

(N.3) (Transitivity).

For any $\boldsymbol{g}\in Y$ and subspaces $X,Z\in\mathcal{N}$, $\left(\boldsymbol{g}\downarrow_X\right)\downarrow_{X\cap Z=}\boldsymbol{g}\downarrow_{X\cap Z}$,

(N.4) (Attribution).

For any $\boldsymbol{g}\in Y$ and subspace $X\in\mathcal{N}$, $\boldsymbol{g}\downarrow_X\in X$, and especially,

(N.5) For any $\boldsymbol{g}\in Y$ and subspace $X\in\mathcal{N}$, $\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}\downarrow_X\geq 0$.

A convention approach to find $\boldsymbol{g}\downarrow_{\alpha_1\cap\cdots\cap\alpha_k}$ is to compute it directly from the null space $N(\tau_1,\tau_2,\cdots,\tau_k)$. We now show another approach that is more suitable to our overall algorithm.

Theorem 1. For any $\boldsymbol{g}\in Y$, we have

$$\boldsymbol{g}\downarrow_{\alpha_1\cap\cdots\cap\alpha_k}=\boldsymbol{g}-\sum_i^k\boldsymbol{g}^{\mathrm{T}}\boldsymbol{o}_i \tag{8}$$

where $\{\boldsymbol{o}_1,\cdots,\boldsymbol{o}_k\}$ are an orthonormal basis of subspace $V(\tau_1,\tau_2,\cdots,\tau_k)$.

Proof. Since $\alpha_1\bigcap\cdots\bigcap\alpha_k$ and $[\tau_1,\tau_2,\cdots,\tau_k]$ are the orthonormal decomposition of Y, according to (N.2) and (N.1) we have

$$\boldsymbol{g}=\boldsymbol{g}\downarrow_{[\tau_1,\cdots,\tau_k]}+\boldsymbol{g}\downarrow_{\alpha_1\cap\cdots\cap\alpha_k}.$$

According to (N.2) and (N.4), the first term becomes

$$\boldsymbol{g}\downarrow_{[\tau_1,\cdots,\tau_k]} = \boldsymbol{g}^{\mathrm{T}}\boldsymbol{o}_1 + \cdots + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{o}_k .$$

Hence (8) is true. **END**

The following theorem shows that the projection of $\boldsymbol{g}_0$ onto the set of all blocking facets σ_t always satisfies Propositions 1 and 2. First, let us simplify the notation and use σ to represent σ_t in the following section and $\boldsymbol{g}_0\downarrow_\sigma$ to stand for $\boldsymbol{g}_0\downarrow_{\alpha_{\sigma(1)}\cap\cdots\cap\alpha_{\sigma(k)}}$ where $k=|\sigma|$ is the number of elements in σ.

Theorem 2. $\left(\tau_{\sigma(j)}\right)^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_\sigma\right)=0$ and $\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_\sigma\right)\geq 0$ for all $j=1,\cdots,k$.

Proof. Since $\boldsymbol{g}_0\downarrow_\sigma$ lies on the intersection of $\left(\alpha_{\sigma(1)}\cap\cdots\cap\alpha_{\sigma(k)}\right)$, it lies on each facet $\alpha_{\sigma(j)}$ for $j=1,\cdots,k$. Thus $\boldsymbol{g}_0\downarrow_\sigma$ is perpendicular to the normal vector of $\alpha_{\sigma(j)}$ (*i.e.* $\left(\tau_{\sigma(j)}\right)^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_\sigma\right)=0$). So it satisfies Proposition 2.

According to (N.5), $\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_\sigma\right)\geq 0$. So it satisfies proposition 1 too. **END**

As such, $\boldsymbol{g}_0\downarrow_\sigma$, the projection of $\boldsymbol{g}_0$ onto all the blocking facets, can be adopted as the next gradient descend vector $\boldsymbol{g}_t$. Hence, $\boldsymbol{g}_0\downarrow_\sigma$, the projection of $\boldsymbol{g}_0$ onto all the blocking facets, can be adopted as the next gradient descend vector $\boldsymbol{g}_t$.

3.5.2. Selecting the Sliding Gradient

In this section, we explore other projection vectors which also satisfy Propositions 1 and 2. Let the j^{th} complement blocking set σ_j^c be the blocking set σ excluding the j^{th} element; *i.e.* $\sigma_j^c=\alpha_1\cap\cdots\cap\alpha_{j-1}\cap\alpha_{j+1}\cap\cdots\cap\alpha_k$. We examine the projection $\boldsymbol{g}_0\downarrow_{\sigma_j^c}$ for $j=1,\cdots,k$. Obviously, $\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_{\sigma_j^c}\right)\geq 0$ according to (N.5) as $\boldsymbol{g}_0\downarrow_{\sigma_j^c}$ is a projection of $\boldsymbol{g}_0$. So if $\left(\tau_{\sigma(j)}\right)^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_{\sigma_j^c}\right)\geq 0$ for all $j\in\sigma$, it satisfies Proposition 2 and hence is a candidate for consideration. For all the candidates, including $\boldsymbol{g}_0\downarrow_\sigma$, which satisfy this proposition, we can compute the inner product of each candidate with the initial gradient descend vector $\boldsymbol{g}_0$, (*i.e.* $\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_{\sigma_j^c}\right)$) and select the maximum. This inner product is a measure of how close or similar a candidate is to $\boldsymbol{g}_0$ so taking the maximum means getting the steepest descend gradient. Notice that if a particular $\boldsymbol{g}_0\downarrow_{\sigma_j^c}$ is selected as the next gradient descending vector, the corresponding α_j is no longer a blocking facet in computing $\boldsymbol{g}_0\downarrow_{\sigma_j^c}$. Thus α_j needs to be removed from σ_t to form the set of *effective blocking facets* σ_t^*. The set of blocking facets for the next iteration σ_{t+1} is σ_t^* plus the newly encountered blocking facet. In summary, the next gradient descend vector $\boldsymbol{g}_t$ is:

$$\boldsymbol{g}_t=\max\left(\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_\sigma\right),\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_{[1]}\right),\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_{[2]}\right),\cdots,\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_{[k]}\right)\right) \tag{9}$$

where $\boldsymbol{g}_{[j]}=\boldsymbol{g}_0\downarrow_{\sigma_j^c};\ j\in\sigma$ with $\left(\tau_{\sigma(j)}\right)^{\mathrm{T}}\left(\boldsymbol{g}_0\downarrow_{\sigma_j^c}\right)\geq 0$ and $k=|\sigma|$.

The effective blocking set σ_t^* is

$$\sigma_t^*=\begin{cases}\sigma_t & \text{if } \boldsymbol{g}_{t+1}=\boldsymbol{g}_0\downarrow_\sigma\\ \sigma_t\setminus\{\alpha_j\} & \text{if } \boldsymbol{g}_{t+1}=\boldsymbol{g}_0\downarrow_{\sigma_j^c}\end{cases}. \tag{10}$$

At first, this seems to increase the computation load substantially. However, we now show that once $\boldsymbol{g}_0\downarrow_\sigma$ is computed, $\boldsymbol{g}_0\downarrow_{\sigma_j^c}$ can be obtained efficiently.

3.5.3. Computing the Gradient Projection Vectors

This session discusses a method of computing $\boldsymbol{g}\downarrow_\sigma$ and $\boldsymbol{g}\downarrow_{\sigma_j^c}$ for any vector $\boldsymbol{g}$. According to (8), $\boldsymbol{g}\downarrow_\sigma = \boldsymbol{g}\downarrow_{\alpha_1\cap\cdots\cap\alpha_k} = \boldsymbol{g} - \sum_i^k \boldsymbol{g}^{\mathrm{T}}\boldsymbol{o}_i$. The orthonormal basis $\{\boldsymbol{o}_1, \boldsymbol{o}_2, \cdots, \boldsymbol{o}_k\}$ can be obtained from the Gram Schmidt procedure as follows:

$$\boldsymbol{o}_1 = \boldsymbol{\tau}_1;\ \boldsymbol{o}_1 = \boldsymbol{o}_1 / |\boldsymbol{o}_1| \tag{11}$$

$$\boldsymbol{o}_2 = \boldsymbol{\tau}_2 - \boldsymbol{\tau}_2^{\mathrm{T}}\boldsymbol{o}_1;\ \boldsymbol{o}_2 = \boldsymbol{o}_2 / |\boldsymbol{o}_2| \tag{12}$$

Let us introduce the notation $\boldsymbol{a}\downarrow\boldsymbol{b}$ to denote the projection of vector $\boldsymbol{a}$ onto vector $\boldsymbol{b}$. We have $\boldsymbol{a}\downarrow\boldsymbol{b} = \left(\frac{\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}}{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}}\right)\boldsymbol{b}$, then $\boldsymbol{o}_2 = \boldsymbol{\tau}_2 - \boldsymbol{\tau}_2\downarrow\boldsymbol{o}_1$ as $\left(\boldsymbol{o}_1^{\mathrm{T}}\boldsymbol{o}_1\right) = 1$. Likewise,

$$\boldsymbol{o}_j = \boldsymbol{\tau}_j - \sum_{i=1}^{j-1}\boldsymbol{\tau}_j\downarrow\boldsymbol{o}_i;\ \boldsymbol{o}_j = \boldsymbol{o}_j / |\boldsymbol{o}_j|. \tag{13}$$

Thus from (8),

$$\boldsymbol{g}\downarrow_\sigma = \boldsymbol{g} - \sum_i^k \boldsymbol{g}^{\mathrm{T}}\boldsymbol{o}_i = \boldsymbol{g} - \sum_{i=1}^k \boldsymbol{g}\downarrow\boldsymbol{o}_i. \tag{14}$$

After evaluating $\boldsymbol{g}\downarrow_\sigma$, we can find $\boldsymbol{g}\downarrow_{\sigma_j^c}$ backward from $j = k$ to 1. Firstly,

$$\boldsymbol{g}\downarrow_{\sigma_k^c} = \boldsymbol{g} - \sum_{i=1}^{k-1}\boldsymbol{g}\downarrow\boldsymbol{o}_i = \boldsymbol{g}\downarrow_\sigma + \boldsymbol{g}\downarrow\boldsymbol{o}_k. \tag{15}$$

Likewise, it can be shown that

$$\boldsymbol{g}\downarrow_{\sigma_j^c} = \boldsymbol{g} - \sum_{i=1}^{j-1}\boldsymbol{g}\downarrow\boldsymbol{o}_i - \sum_{i=j+1}^{k}\boldsymbol{g}\downarrow\boldsymbol{o}_i^{(j)} \text{ for } j = 2 \text{ to } k. \tag{16}$$

The first summation is projections of $\boldsymbol{g}$ onto existing orthonormal basis $\boldsymbol{o}_i$. Each term in this summation has already been computed before and hence is readily available. However, the second summation is projections on new basis $\boldsymbol{o}_i^{(j)}$. Each of these basis must be re-computed as the facet α_j is skipped in σ_j^c. Let

$$T_k = \boldsymbol{g} + \boldsymbol{g}\downarrow\boldsymbol{o}_k;\ S_k = 0 \tag{17}$$

$$T_j = T_{j+1} + \boldsymbol{g}\downarrow\boldsymbol{o}_j;\ S_j = \sum_{i=j+1}^{k}\boldsymbol{g}\downarrow\boldsymbol{o}_i^{(j)}. \tag{18}$$

Then we can obtain $\boldsymbol{g}\downarrow_{\sigma_j^c}$ recursively from $j = k, k-1, \cdots, 1$ by:

$$\boldsymbol{g}\downarrow_{\sigma_j^c} = T_j - S_j. \tag{19}$$

To compute $o_i^{(j)}$, some of the intermediate results in obtaining the orthonormal basis can also be reused.

Let $\mu_{j,1} = 0$ for all $j = 2, \cdots, k$ and $\mu_{j,i} = \mu_{j,i-1} + \boldsymbol{\tau}_j\downarrow\boldsymbol{o}_i$ for $i = 1, \cdots, j-1$, then we have

$$\boldsymbol{o}_j = \boldsymbol{\tau}_j - \mu_{j,j-1};\ \boldsymbol{o}_j = \boldsymbol{o}_j / |\boldsymbol{o}_j| \text{ for } j = 2, \cdots, k. \tag{20}$$

The intermediate terms $\mu_{j,i}$ can be reused in computing $\boldsymbol{o}_m^{(j)}$ as follows:

$$\boldsymbol{o}_m^{(j)} = \boldsymbol{\tau}_m - \mu_{m,j-1} - \sum_{i=j+1}^{m-1} \boldsymbol{\tau}_m \downarrow \boldsymbol{o}_i^{(j)};\ \boldsymbol{o}_m^{(j)} = \frac{\boldsymbol{o}_m^{(j)}}{\left|\boldsymbol{o}_m^{(j)}\right|} \text{ for } m = +1, \cdots, k. \tag{21}$$

By using these intermediate results, the computation load can be reduced substantially.

3.6. Termination Criterion

When a new blocking facet is encountered, it will be added to the existing set of blocking facets. Hence both σ_t and σ_t^* will typically grow in each iteration unless one of the $\boldsymbol{g}_0 \downarrow_{\sigma_j^c}$ is selected as $\boldsymbol{g}_t$. In this case, $\alpha_{\sigma(j)}$ is deleted from σ_t according to (10). The following theorem, which was first presented in [1] shows that when $\left|\sigma_t^*\right| = m$, the algorithm can stop.

Theorem 3 (Stopping criterion) Assuming that the dual feasible region $\mathcal{D}$ is non-empty, let $\boldsymbol{P}_t \in \mathcal{D}$ and is descending along the initial direction $\boldsymbol{g}_0 = -\boldsymbol{b}$; let $\left|\sigma_t^*\right|$ be the number of effective blocking facets in σ_t^* at the t^{th} iteration. If $\left|\sigma_t^*\right| = m$ and the rank $r\left(\sigma_t^*\right) = m$, then $\boldsymbol{P}_t$ is a lowest point in the dual feasible region $\mathcal{D}$.

Proof. If $\left|\sigma_t^*\right| = m$ and the rank $r\left(\sigma_t^*\right) = m$, then the m facets in σ_t^* form a cone C with vertex $\boldsymbol{V} = \boldsymbol{P}_t$. Since the rank is m, its corresponding null space contains only the zero vector. So $\boldsymbol{g}_0 \downarrow_\sigma = \boldsymbol{g}_0 \downarrow_{\alpha_{\sigma(1)} \cap \cdots \cap \alpha_{\sigma(k)}} = 0$.

As mentioned about the facet/edge duality in Section 2, for $j = 1, \cdots, m$, edge-line R_i^+ is the intersection of all C-facets except α_i. That means $R_i^+ = \sigma_i^c$. Since an edge-line is a 1-dimensional line, the projection of a vector $\boldsymbol{g}_0$ onto R_i^+ equals to $\pm\boldsymbol{r}_i$ and hence $\boldsymbol{g}_t \downarrow_{\sigma_i^c} = \boldsymbol{g}_t \downarrow_{R_i^+} = \pm\boldsymbol{r}_i$. Since $\boldsymbol{g}_t \downarrow_{\sigma_i^c}$ are projections of $\boldsymbol{g}_0$, according to (N.5), $\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_t \downarrow_{\sigma_i^c}\right) \geq 0$.

Since $\left|\sigma_t^*\right| = m$, it means that $\boldsymbol{g}_t \downarrow_{\sigma_i^c}$ does not satisfy Proposition 2 for all $i = 1, \cdots, m$. Otherwise, one of the $\boldsymbol{g}_t \downarrow_{\sigma_i^c}$ would have been selected as the next gradient descend vector and, according to (10), it would be deleted from σ_t^* and hence $\left|\sigma_t^*\right|$ would be less than m. This means that at least one of $j \in \sigma$ has a value $\boldsymbol{\tau}_j^{\mathrm{T}}\left(\boldsymbol{g}_t \downarrow_{\sigma_i^c}\right) < 0$. However, for all $k \neq i$, $\boldsymbol{g}_t \downarrow_{\sigma_i^c}$ is in the null space of α_k so $\boldsymbol{\tau}_k^{\mathrm{T}}\left(\boldsymbol{g}_t \downarrow_{\sigma_i^c}\right) = 0$. This leaves $\boldsymbol{\tau}_i^{\mathrm{T}}\left(\boldsymbol{g}_t \downarrow_{\sigma_i^c}\right) < 0$. If $\boldsymbol{g}_t \downarrow_{\sigma_i^c} = \boldsymbol{r}_i$, then $\boldsymbol{\tau}_i^{\mathrm{T}}\left(\boldsymbol{g}_t \downarrow_{\sigma_i^c}\right) = \boldsymbol{\tau}_i^{\mathrm{T}}\boldsymbol{r}_i < 0$. This contradicts to the fact that $\boldsymbol{\tau}_i^{\mathrm{T}}\boldsymbol{r}_i \geq 0$ in (1). Therefore, $\boldsymbol{g}_t \downarrow_{\sigma_j^c} = -\boldsymbol{r}_i$. Since $\boldsymbol{g}_0^{\mathrm{T}}\left(\boldsymbol{g}_t \downarrow_{\sigma_i^c}\right) \geq 0$, $\boldsymbol{g}_0^{\mathrm{T}}\left(-\boldsymbol{r}_i\right) = \left(-\boldsymbol{g}_0\right)^{\mathrm{T}}\boldsymbol{r}_i \geq 0$. Note that $\left(-\boldsymbol{g}_0\right)^{\mathrm{T}}\boldsymbol{r}_i$ means that edge $\boldsymbol{r}_i$ is in opposite direction of $\boldsymbol{g}_0$. As this is true for all edges, there is no path for $\boldsymbol{g}_t$ to descend further from this vertex. It is obvious that the vertex $\boldsymbol{V}$ is the lowest point of C when viewed in the $\boldsymbol{b}$ direction.

Since $\boldsymbol{P}_t$ is dual feasible, and $\boldsymbol{V}$ is a vertex of $\mathcal{D}$. Cone C coincides with the dual feasible region $\mathcal{D}$ in a neighborhood N of $\boldsymbol{V}$, it is obvious that $\boldsymbol{P}_t$ is the lowest point of $\mathcal{D}$ when viewed in the $\boldsymbol{b}$ direction. **END**

In essence, when the optimal vertex $\boldsymbol{V}^*$ is reached, all the edges of the cone will be in opposite direction of the gradient vector $\boldsymbol{g}_0 = -\boldsymbol{b}$. There is no path to descend further so the algorithm terminates.

4. The Pseudo Code of the Sliding Gradient Algorithm

The entire algorithm is summarized as follows in Table 2.

Step 0 is the initialization step that sets up the tableau and the starting point $\boldsymbol{P}$. Step 2 is to find a set of initial blocking facets σ in preparation of step 4. In the inner loop, Step 4 calls the *Gradient Select* routine. It computes $\boldsymbol{g}_0 \downarrow_{\sigma}$ and $\boldsymbol{g}_0 \downarrow_{\sigma_j^c}$ in view of σ using Equations (11) to (21) and select the best gradient vector $\boldsymbol{g}$ according to (9). This routine not only returns $\boldsymbol{g}$ but also the effective blocking facets σ^* and $\boldsymbol{g}_0 \downarrow_{\sigma_j^c}$ for subsequent use. Theorem 3 states that when the size of σ^* reaches m, the optimal point is reached. So when it does, step 5 returns the optimal point and the optimal value to the calling routine. Step 6 is to find the closest blocking facet according to (6). Because $\boldsymbol{P}$ lies on every facets of σ, $t_j = 0$ for $j \in \sigma$. Hence, we only need to compute those t_j where $j \notin \sigma$. The newly found blocking facet is then included in σ in step 7 and the

Table 2. The sliding gradient algorithm.

steps	**Input: A, b and $\tilde{c}$; and $\mathcal{D}$ is non-empty and P_0 is inside $\mathcal{D}$** **Output: *OptPt* & *OptVal***
0	Construct the Facet Tableau $[\boldsymbol{\tau}]$ from A, b and $\tilde{c}$ $\boldsymbol{P} = \boldsymbol{P}_0$; $\boldsymbol{g}_0 = -\boldsymbol{b}$
1	$d_j = \boldsymbol{P}^{\mathrm{T}}\tau_j - c_j$ for $j = 1, \cdots, m+n$ }
2	$\sigma = \{ j \mid \lvert d_j \rvert < \delta \}$; % δ is a small constant $\sigma^* = \sigma$; % σ^* is set of blocking facet
3	**while** (true)
4	**if** $\sigma \neq \phi$ % compute $\boldsymbol{g}, \boldsymbol{g}\downarrow_{\sigma_j^c}$ and update σ^* $\left[\boldsymbol{g}, \boldsymbol{g}\downarrow_{\sigma_j^c}, \sigma^*\right]$ = Gradient Select($\boldsymbol{g}_0, \sigma, [\boldsymbol{\tau}], \boldsymbol{P}, c$) **else** $\boldsymbol{g} = \boldsymbol{g}_0$
5	**if** $\lvert \sigma^* \rvert == m$ $OptPt = \boldsymbol{P}; OptVal = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{b}$; return (*OptPt, OptVal*) **else** $\sigma = \sigma^*$
6	$t_j = \dfrac{c_j \boldsymbol{P}^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}^{\mathrm{T}}\boldsymbol{\tau}_j}$; for $j \in \{1, 2, \cdots, m+n\} \setminus \sigma$ $j^* = \arg\min_j \{ t_j \mid t_j > 0 \}$; $\boldsymbol{Q} = \boldsymbol{P} + t_{j^*}\boldsymbol{g}$; $\boldsymbol{P} = \boldsymbol{Q}$
7	$\sigma = \sigma^* \cup \{ i \mid \lvert t_i - t_{j^*} \rvert < \epsilon \}$ % update blocking facets
8	**end**

inner loop is repeated until the optimal vertex is found.

5. Implementation and Experimental Results

5.1. Experiment on the Klee-Minty Problem

We use the Klee-Minty example presented in [18][1] to walk through the algorithm in this section. An example of the Klee-Minty Polytope example is shown below:

$$\max 2^{m-1}x_1 + 2^{m-2}x_2 + \cdots + 2x_{m-1} + x_m .$$

Subject to

$$\begin{array}{lllllllll}
x_1 & & & & & & & \le & 5^1 \\
4x_1 & + & x_2 & + & & & & \le & 5^2 \\
8x_1 & + & 4x_2 & + & x_3 & & & \le & 5^3 \\
\vdots & & & & & & & \vdots & \vdots \\
2^m x_1 & + & 2^{m-1}x_2 & + & 2^{m-2}x_3 & + \cdots & x_m & \le & 5^m
\end{array}$$

For the standard simplex method, it needs to visit all 2^{m-1} vertices to find the optimal solution. Here we show that, with a specific choice of initial point $\boldsymbol{P}_0$, the Sliding Gradient algorithm can find the optimal solution in two iterations—no matter what the dimension m is.

To apply the Sliding Gradient algorithm, we first construct the tableau. For an example with $m = 5$, the simplex tableau is:

1					5
4	1				25
8	4	1			125
16	8	4			625
32	16	8	4	1	3125
16	8	4	2	1	

The $\boldsymbol{b}$ vector is $\boldsymbol{b} = [5, 25, 125, 625, 3125]^{\mathrm{T}}$. After adding the slack variables, the facet tableau becomes:

α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
1	0	0	0	0	1	0	0	0	0
4	1	0	0	0	0	1	0	0	0
8	4	1	0	0	0	0	1	0	0
16	8	4	1	0	0	0	0	1	0
32	16	8	4	1	0	0	0	0	1
16	8	4	2	1	0	0	0	0	0

[1]Other derivations of the Klee-Minty formulas have also been tested and the same results are obtained.

Firstly, notice that α_5 and α_{10} have the same normal vector (*i.e.* $\boldsymbol{\tau}_5 = \boldsymbol{\tau}_{10}$) so we can ignore α_{10} for further consideration. This is true for all value of *m*.

If we choose $\boldsymbol{P}_0 = M\boldsymbol{b}$, where M is a positive number (e.g. $M = 100$), It can be shown that $\boldsymbol{P}_0$ is inside the dual feasible region. The initial gradient descend vector is: $\boldsymbol{g}_0 = -\boldsymbol{b}$.

With $\boldsymbol{P}_0$ and $\boldsymbol{g}_0$ as initial conditions, the algorithm proceeds to find the first blocking facet using (6). The displacements t_j for each facet can be found by:

$$t_j = \frac{c_j - \boldsymbol{P}_0^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_0^{\mathrm{T}}\boldsymbol{\tau}_j} = \frac{c_j}{-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} - \frac{M\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}{-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} = \frac{c_j}{-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} + M = M - \frac{c_j}{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}.$$

With $\boldsymbol{P}_0$ and $\boldsymbol{g}_0$ as initial conditions, the algorithm proceeds to find the first blocking facet using (6). The displacements t_j for each facet can be found by:

$$t_j = \frac{c_j - \boldsymbol{P}_0^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_0^{\mathrm{T}}\boldsymbol{\tau}_j} = \frac{c_j}{-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} - \frac{M\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}{-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} = \frac{c_j}{-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} + M = M - \frac{c_j}{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}. \tag{22}$$

We now show that the minimum of all displacements is t_m.

First of all, at = *m*, $\boldsymbol{\tau}_m = [0,\cdots,0,1]^{\mathrm{T}}$, $c_m = 1$ and $b_m = 5^m$, so $t_m = M - 5^{-m}$.

For $m < j \le 2m-1$, $c_j = 0$, so $t_j = M > t_m$.

For $1 \le j < m$, $c_j = 2^{m-j}$, and the elements of $\boldsymbol{\tau}_j$ are:

$$\tau_{ij} = \begin{cases} 0 & \text{if } i < j \\ 1 & \text{if } i = j \\ 2^{i-j+1} & \text{if } j < i \le m \end{cases}.$$

The 2nd term of Equation (22) can be re-written as:

$$\frac{c_j}{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} = \frac{1}{\boldsymbol{b}^{\mathrm{T}}\left(\frac{\boldsymbol{\tau}_j}{c_j}\right)} = \frac{1}{\boldsymbol{b}^{\mathrm{T}}\left(\frac{\boldsymbol{\tau}_j}{2^{m-j}}\right)}$$

The inner product of the denominator is:

$$\boldsymbol{b}^{\mathrm{T}}\left(\frac{\boldsymbol{\tau}_j}{2^{m-j}}\right) = \sum_{i=1}^{m} b_i\left(\frac{\tau_{ij}}{2^{m-j}}\right) = \sum_{i=1}^{m-1} b_i\left(\frac{\tau_{ij}}{2^{m-j}}\right) + b_m\left(\frac{2^{m-j+1}}{2^{m-j}}\right) = \sum_{i=1}^{m-1} b_i\left(\frac{\tau_{ij}}{2^{m-j}}\right) + 2b_m$$

Since all the elements in the $\boldsymbol{b}$ vector and the $\boldsymbol{\tau}$ are positive, the summation is a positive number. Thus

$$\boldsymbol{b}^{\mathrm{T}}\left(\frac{\boldsymbol{\tau}_j}{2^{m-j}}\right) = \sum_{i=1}^{m-1} b_i\left(\frac{\tau_{ij}}{2^{m-j}}\right) + 2b_m > b_m$$

Since the value of the denominator is bigger than $b_m = 5^m$, we have

$$\frac{1}{\boldsymbol{b}^{\mathrm{T}}\left(\frac{\boldsymbol{\tau}_j}{2^{m-j}}\right)} < 5^{-m}$$

So

$$t_j = M - \frac{c_j}{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j} = M - \frac{1}{\boldsymbol{b}^{\mathrm{T}}\left(\frac{\boldsymbol{\tau}_j}{2^{m-j}}\right)} > M - 5^{-m} = t_m .$$

Hence t_m is the smallest displacement. For the case of $m=5$, their values are shown in the first row (first iteration) of the following Table 3.

Thus α_m is the closest blocking facet. Hence, $\sigma_1^* = \sigma_1 = \{\alpha_m\}$. For the next iteration,

$$\boldsymbol{P}_1 = \boldsymbol{P}_0 + t_m \boldsymbol{g}_0 = M\boldsymbol{b} + \left(M - 5^{-m}\right)\left(-\boldsymbol{b}\right) = M\boldsymbol{b} - M\boldsymbol{b} + 5^{-m}\boldsymbol{b} = 5^{-m}\boldsymbol{b} .$$

The gradient vector $\boldsymbol{g}_1$ is $\boldsymbol{g}_0$ projects onto α_m. Because $\boldsymbol{\tau}_m = [0,\cdots,1]^{\mathrm{T}}$ is already an orthonormal vector, we have according to (8)

$$\boldsymbol{g}_1 = \boldsymbol{g}_0 - \left(\boldsymbol{g}_0^{\mathrm{T}}\boldsymbol{\tau}_m\right)\boldsymbol{\tau}_m = \boldsymbol{g}_0 - \left[0,\cdots,-5^m\right]^{\mathrm{T}} = \left[-5,-5^2,\cdots,-5^{m-1},0\right]^{\mathrm{T}} .$$

In other words, $\boldsymbol{g}_1$ is the same as $-\boldsymbol{b}$ except that the last element is zeroed out. Using $\boldsymbol{P}_1$ and $\boldsymbol{g}_1$, the algorithm proceeds to the next iteration and evaluates the displacements t_j again. For $j = m+1$ to $2m-1$, since $c_j = 0$ and $\boldsymbol{\tau}_j$ is a unit vector with only one non-zero entry at the j^{th} element,

$$t_j = -\frac{\boldsymbol{P}_1^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j} = -\frac{5^{-m}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j} = -5^{-m}\frac{b_j}{-b_j} = 5^{-m} \quad \text{for } m+1 \le j \le 2m-1 .$$

Thus the displacements t_m to t_{2m-1} have the same value of 5^{-m}.

For $1 \le j < m$, we have:

$$t_j = \frac{c_j - \boldsymbol{P}_1^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j} = \frac{c_j - 5^{-m}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j} = 5^{-m}\frac{5^m c_j - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j} .$$

As mentioned before, $\boldsymbol{g}_1$ is the same as $-\boldsymbol{b}$ except that the last element is zero, we can express $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j$ in terms of $\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j$ as follows:

$$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j = -\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j + b_m\tau_{mj} .$$

The numerator then becomes:

$$5^m c_j - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j = 5^m c_j + \boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j - b_m\tau_{mj} .$$

Since $c_j = 2^{m-j}$, $c_j = 2^{m-j}$ and $\tau_{mj} = 2^{m-j+1}$, substituting these values to the above equation, the numerator becomes

$$5^m c_j - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j = 5^m 2^{m-j} - 5^m 2^{m-j+1} + \boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j = \boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j - 5^m 2^{m-j} .$$

Thus

$$t_j = 5^{-m}\frac{5^m c_j - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{\tau}_j}{\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j} = 5^{-m}\left(1 - \frac{5^m 2^{m-j}}{\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j}\right).$$

Table 3. Displacement values t_i in each iterations for $m=5$.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
1	99.9999	99.9999	99.9999	99.9998	99.9997	100	100	100	100
2	0.0018	0.0018	0.0018	0.0035	0	0.00032	0.00032	0.00032	0.00032

Notice that all elements in $\boldsymbol{g}_1$ are negative but all of $\boldsymbol{\tau}_j$ are positive. So the inner product $\boldsymbol{g}_1^{\mathrm{T}}\boldsymbol{\tau}_j$ is a negative number. As a result, the last term inside the bracket is a positive number which makes the whole value inside the bracket bigger than one and hence $t_j > 5^{-m}$ for $1 \le j < m-1$. Moreover, t_m is zero as $\boldsymbol{g}_1$ lies on α_m. The actual displacement values for the case of $m=5$ are shown in the second row of Table 3.

Since t_m to t_{2m-1} have the same lowest displacement value, all of them are blocking facets so $\sigma_2^* = \sigma_2 = \{\alpha_m\} \bigcup \{\alpha_{m+1}, \cdots, \alpha_{2m-1}\} = \{\alpha_m, \alpha_{m+1}, \cdots, \alpha_{2m-1}\}$. Also,

$$\boldsymbol{P}_2 = \boldsymbol{P}_1 + t_{m+1}\boldsymbol{g}_1 = 5^{-m}\boldsymbol{b} + 5^{-m}\left[-5, -5^2, \cdots, -5^{m-1}, 0\right]^{\mathrm{T}} = \left[0, 0, \cdots, 0, 1\right]^{\mathrm{T}}.$$

Now $\left|\sigma_t^*\right| = m$, so $\boldsymbol{P}_2$ has reached a vertex of a cone. According to Theorem 3, the algorithm stops. The optimal value is $\boldsymbol{P}_2^{\mathrm{T}}\boldsymbol{b} = 5^m$, which is the last element of the $\boldsymbol{b}$ vector.

Thus with a specific choice of the initial point $\boldsymbol{P}_0 = M\boldsymbol{b}$, the Sliding Gradient algorithm can solve the Klee-Minsty LP problem in two iterations, and it is independent of m.

5.2. Issues in Algorithm Implementation

The Sliding Gradient Algorithm has been implemented in MATLAB and tested on the Klee-Minty problems and also self-generated LP problems with random coefficients. As a real number can only be represented in finite precision in digital computer, care must be taken to deal with the round-off issue. For example, when a point $\boldsymbol{P}$ lies on a plane $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{\tau} = c$, the value $d = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{\tau} - c$ should be exactly zero. But in actual implementation, it may be a very small positive or negative number. Hence in step 2 of the aforementioned algorithm, we need to set a threshold δ so that if $|d| < \delta$, we regard that point $\boldsymbol{P}$ is laid on the plane. Likewise for the Klee-Minty problem, this algorithm relies on the fact that in the second iteration, the displacement values t_i for $i = m+1$ to $2m-1$ should be the same and they should all be smaller than the values of t_j for $j = 1$ to $m-1$. Due to round-off errors, we need to set a tolerant level to treat the first group to be equal and yet if this tolerant level is set too high, then it cannot exclude members of the second group. The issue is more acute as m increases. It will require higher and higher precision in setting the tolerant level to distinguish these two groups.

6. Conclusions and Future Work

We have presented a new approach to tackle the linear programming problem in this paper. It is based on the gradient descend principle. For any initial point inside the feasible region, it will pass through the interior of the feasible region to reach the optimal vertex. This is made possible by projecting the gravity vector to a set of blocking facets and using that as descending vector in each iteration. In fact, the descending trajectory is a sequence of line segments that hug either a single blocking facet or the intersections of them, and each line segment is ad-

vancing towards the optimal point. It should be noted that there is no parameters (such as step-size, ..., etc.) to tune in this algorithm although one needs to take care of numerical round-off issue in actual implementation.

This work opens up many areas of future research. On the one hand, we are extending this algorithm so that it can relax the constraint of starting from a point inside the feasible region. Promising development has been achieved in this area though more thorough testing on obscure cases need to be carried out.

On the theoretical front, we are encouraged that, from the algorithm walk-through on the Klee-Minty example, this algorithm exhibits strongly polynomial complexity characteristics. Its complexity does not appear to depend on the bit sizes of the LP coefficients. However, more rigorous proof is needed and we are working towards this goal.

Acknowledgements

The authors wish to thank all his friends for their valuable critics and comments on the research. Special thanks are given to Prof. Yong Shi, Prof. Sizong Guo for their supports. This study is partially supported by the grants (Grant Nos. 61350003, 70621001, 70531040, 90818025) from the Natural Science Foundation of China, and grant (Grant No. L2014133) from Department of Education of Liaoning Province.

References

[1] Wang, P.Z., Lui, H.C., Liu, H.T. and Guo, S.C. (2017) Gravity Sliding Algorithm for Linear Programming. *Annals of Data Science*, **4**, 193-210. https://doi.org/10.1007/s40745-017-0108-1

[2] Dantzig, G.B. (1963) Linear Programming and Extensions. Princeton University Press, Princeton. https://doi.org/10.1515/9781400884179

[3] Megiddo, N. (1987) On the Complexity of Linear Programming. In: Bewley, T., Ed., *Advances Economic Theory*, 5*th World Congress*, Cambridge University Press, Cambridge, 225-268. https://doi.org/10.1017/CCOL0521340446.006

[4] Megiddo, N. (1984) Linear Programming in Linear Time When the Dimension Is Fixed. *Journal of ACM*, **31**, 114-127. https://doi.org/10.1145/2422.322418

[5] Smale, S. (1983) On the Average Number of Steps of the Simplex Method of Linear Programming. *Mathematical Programming*, **27**, 251-262. https://doi.org/10.1007/BF02591902

[6] Todd, M. (1986) Todd, Polynomial Expected Behavior of a Pivoting Algorithm for Linear Complementarity and Linear Programming Problems. *Mathematical Programming*, **35**, 173-192. https://doi.org/10.1007/BF01580646

[7] Klee, V. and Minty, G.J. (1972) How Good Is the Simplex Method. In: Shisha, O., Ed., *Inequalities III*, Academic Press, New York, 159-175.

[8] Chvatal, V. (1983) Linear Programming. W.H. Freeman and Company, New York.

[9] Goldfarb, D. and Sit, W. (1979) Worst Case Behavior of the Steepest Edge Simplex Method. *Discrete Applied Mathematics*, **1**, 277-285. https://doi.org/10.1016/0166-218X(79)90004-0

[10] Jeroslow, R. (1973) The Simplex Algorithm with the Pivot Rule of Maximizing Im-

provement Criterion. *Discrete Mathematics*, **4**, 367-377. https://doi.org/10.1016/0012-365X(73)90171-4

[11] Zadeh, N. (1980) What Is the Worst Case Behavior of the Simplex Algorithm? Technical Report 27, Dept. Operations Research, Stanford University, Stanford.

[12] Karmarkar, N.K. (1984) A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, **4**, 373-395. https://doi.org/10.1007/BF02579150

[13] Deza, A., Nematollahi, E. and Terlaky, T. (2008) How Good Are Interior Point Methods? Klee-Minty Cubes Tighten Iteration-Complexity Bounds. *Mathematical Programming*, **113**, 1-14.

[14] Barasz, M. and Vempala, S. (2010) A New Approach to Strongly Polynomial Linear Programming.

[15] Amenta, N. and Ziegler, G. (1999) Deformed Products and Maximal Shadows of Polytopes. *Contemporary Mathematics*, **223**, 57-90. https://doi.org/10.1090/conm/223/03132

[16] Wang, P.Z. (2011) Cone-Cutting: A Variant Representation of Pivot in Simplex. *Information Technology & Decision Making*, **10**, 65-82.

[17] Wang, P.Z. (2014) Discussions on Hirsch Conjecture and the Existence of Strongly Polynomial-Time Simplex Variants. *Annals of Data Science*, **1**, 41-71. https://doi.org/10.1007/s40745-014-0005-9

[18] Greenberg, H.J. (1997) Klee-Minty Polytope Shows Exponential Time Complexity of Simplex Method. University of Colorado at Denver, Denver.