

# A Facet Enumeration Algorithm for Convex Polytopes

Yaguang Yang<sup>1</sup>

NASA, Goddard Space Flight Center  
8800 Greenbelt Rd, Greenbelt, 20771 MD, USA.  
`yaguang.yang@verizon.net`

**Abstract.** This paper proposes a simple facet enumeration algorithm for convex polytopes. The complexity of the algorithm is analyzed. The algorithm is implemented in Matlab. Some polytopes with known H-representations and V-representations are used as test examples. The preliminary numerical test shows the effectiveness and efficiency of the proposed algorithm. Due to the duality between the vertex enumeration problem and facet enumeration problem, we expect that this method can also be used to solve the vertex enumeration problem.

**Keywords:** Algorithm · convex polytope · facet enumeration.

## 1 Introduction

For convex polytopes, there are two different but equivalent representations: (a) H-representation, and (b) V-representation. H-representation uses a set of closed half-spaces to define a convex polytope, i.e., a polytope is given as  $\mathbf{Ax} \leq \mathbf{b}$ , where  $\mathbf{A} \in \mathbf{R}^{m \times d}$  and  $\mathbf{b} \in \mathbf{R}^m$  are given, and  $\mathbf{x} \in \mathbf{R}^d$  satisfying the inequality constraints is the set of points in the convex polytope. A  $k$ -face of a polytope is the set of  $\mathbf{x}$  that meet two conditions: (1)  $\mathbf{Ax} \leq \mathbf{b}$ , and (2) for some  $d - k$  independent rows of  $\mathbf{A}_k$ ,  $\mathbf{A}_k \mathbf{x} = \mathbf{b}$ . For  $k = 0$ , a 0-face of the polytope is a vertex; for  $k = 1$ , a 1-face of the polytope is an edge; for  $k = d - 2$ , a  $(d - 2)$ -face of the polytope is a ridge; for  $k = d - 1$ , a  $(d - 1)$ -face of the polytope is a facet. In the remainder of the paper, we use vertex, edge, and facet explicitly, but we do not make distinction for other  $k$ -faces in general. V-representation uses a set of vertices  $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$  to define a convex polytope as a convex hull of  $\mathcal{V}$ . Given one of the representations, there is a need, in many applications [1, 8, 13], to know what the other corresponding representation is. The transformation from (a) to (b) is known as the vertex enumeration and the other from (b) to (a) is known as the facet enumeration.

Vertex enumeration is clearly related to the simplex method of linear programming. The so-called double-description method can be dated back to Fourier (1827) [30] and was reinvented by Motzkin [22]. This method constructs the polytope sequentially by adding one constraint at a time. All new vertices produced must lie on the hyperplanes bounding the constraint currently being inserted.

A more popular vertex enumeration method is based on pivoting, which was discussed by Chand and Kapur [10], Dyer [14], Swart [24], among others. The most popular method in this category is the reverse search method proposed by Avis and Fukuda [2]. Their method starts at an “optimum vertex” and traces out a tree in depth order by “reverse” Bland’s rule [5].

Noticing that the dual problem of a vertex (resp. facet) enumeration problem is the facet (resp. vertex) enumeration problem for the same polytope where the input and output are simply interchanged, Bremner, Fukuda and Marzetta [7] pointed out that the vertex enumeration methods can be used for the facet enumeration problem. They suggested using a primal-dual approach proposed in [16] that includes two purely combinatorial algorithms for enumerating all faces of a  $d$ -polytope based on the combinatorial vertices’ description and some information on edges. In [3], Avis and Jordan published a scalable parallel vertex/facet enumeration code. Avis and Jordan’s paper is also a good source to find the related works.

Although vertex enumeration can be used for facet enumeration using duality theory, converting a primal problem to a dual problem to use vertex enumeration and then converting it back would need significant effort. Developing a direct facet enumeration algorithm is therefore necessary. The convex hull computation can be used for this purpose. Given  $n$  points in  $d$ -dimensional Euclidean space (some of them may not be vertices), convex hull algorithms find the H-representation that has  $h \leq n$  vertices, and the convex polytope contains all  $n$  points. Some algorithms may represent the convex hull by an intersection of a set of half spaces. For  $d \leq 3$ , Chan’s algorithm [9] and the Kirkpatrick–Seidel algorithm [20] are well-known convex hull algorithms because of their low computational complexity  $\mathcal{O}(n \log n)$ . For  $d > 3$ , the complexity of convex hull algorithms might be higher according to McMullen’s Upper Bound Theorem [21]. In [11], the complexity upper bound is conservatively estimated as  $\mathcal{O}(n^{\lfloor d/2 \rfloor})$ . In reality, the upper bound of a carefully implemented beneath-and-beyond method can be tightly related to the sizes of the input and output as we will see in Section 4. In [17], the author implemented a beneath-and-beyond algorithm for vertex/facet enumerations as a software framework (iB4e)<sup>1</sup>. For a fixed  $d$ , to obtain a single facet, Seidel [23] proposed an algorithm, whose computational effort is estimated as  $\mathcal{O}(n^2 + F \log(n))$ , where  $F$  is the number of facets. Clearly, if  $n$  given points are all vertices, the convex hull problem is reduced to the facet enumeration problem. Therefore, in principle, convex hull algorithms, such as the Quickhull algorithm [4] and its variants, for example [2, 11, 20, 24, 25], can be applied to the facet enumeration problem.

But given additional information that all input points are vertices of the convex polytope, we should make full use of the information to develop more efficient algorithm(s) specifically for the facet enumeration problem. In addition, a complexity analysis of the corresponding enumeration algorithm should be more accurate. In this paper, we consider the facet enumeration problem without

<sup>1</sup> It is worthwhile noting that Beneath-and-Beyond [6, 18] has been widely used in convex hull algorithms to facilitate the construction of facets.

using duality of the vertex enumeration. This is a “direct” facet enumeration method which avoids converting back and forth primal and dual problems. The algorithm first finds all edges of the polytope, which provides a vertex/edge diagram (adjacent matrix) that connects every vertex to its neighbor vertices. Then the information of the diagram (adjacent matrix) is used to effectively construct all facets of the polytope and derive the complexity bound of the proposed algorithm.

The first step of the proposed algorithm is similar to the edge-skeleton algorithm described in [15]. Given a set of edges (each edge connects two adjacent vertices), together with all edge directions of a polytope  $P$ , the edge-skeleton algorithm of [15] computes edge-skeleton in polynomial time. In contrast, our proposed algorithm does not require prior knowledge of edge directions of a polytope  $P$  while still achieving the same goal in polynomial time.

The second step is similar to the reconstruction of a simple polytope from its edge-skeleton graph discussed in [19]. But our proposed algorithm does not limit the reconstruction for only simple polytopes as required in [19] (see also [15]). To efficiently construct all facets, we store the vertex-edge information in an adjacent matrix  $\mathbf{D}$ , which can be used to efficiently generate a connected tree. For any vertex, using matrix  $\mathbf{D}$ , one can create a branch with depth of  $d - 1$  (the branch connects  $d$  vertices). If vertices in the branch of the tree from a root vertex to an end vertex form a hyperplane, using exactly the strategy of Beneath-and-Beyond, one can check if all vertices in  $\mathcal{V}$  are in a half space defined by the hyperplane. If the answer is true, the hyperplane contains a facet of the polytope, otherwise, it does not. Repeating these steps for all vertices will find all half spaces whose intersection forms the polytope.

Our experience shows that finding all edges using the proposed method is negligible compared to determining all the facets of the polytope. Once we find all the edges of the polytope, the number of candidate facets is significantly reduced, thereby making the overall algorithm more efficient. Finally, we would like to mention that using the duality between the vertex enumeration problem and facet enumeration problem, we expect that this method can also be used to solve the vertex enumeration problem with some minor tweaks but that is not the purpose of this paper.

The remainder of the paper is organized as follows: Section 2 discusses the edge detection. Section 3 provides details of the proposed algorithm. Section 4 is focused on the complexity analysis for the proposed algorithm. Section 5 presents some numerical examples to show the effectiveness and efficiency of the algorithm. Concluding remarks are summarized in Section 6.

## 2 Edge detection

Assume that a  $d$ -dimensional polytope has  $n$  vertices. Each vertex  $\mathbf{v}_i$  is represented as a **row vector**, and the  $n$  vertices are stored in a matrix  $\mathbf{V} \in \mathbf{R}^{n \times d}$ . To have an efficient algorithm and simplify the discussion, we set the centroid

of the polytope  $\mathbf{v}_0$  as the origin which is an interior point of the polytope. Let

$$\mathbf{u}_i = \mathbf{v}_i - \mathbf{v}_0, \quad i = 1, \dots, n. \quad (1)$$

We will denote by  $\mathcal{U} = \{\mathbf{u}_1, \dots, \mathbf{u}_n\}$  the set of vertices which form a polytope with its center as the origin of the coordinate system. We also use a matrix

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_n \end{bmatrix} \in \mathbf{R}^{n \times d}$$

to store these vertices. Our idea is to calculate a **column vector**  $\mathbf{h}_i \in \mathbf{R}^d$ , or the hyperplane  $\mathbf{h}_i^T \mathbf{x} = 1$ , where  $\mathbf{x} \in \mathbf{R}^d$  is the set of  $d$ -dimensional column vectors that spans the hyperplane, so that the  $m$  half spaces (associated with the  $m$  hyperplanes)

$$\mathbf{H}\mathbf{x} := [\mathbf{h}_1, \dots, \mathbf{h}_m]^T \mathbf{x} \leq \mathbf{e} \quad (2)$$

defines the polytope, where  $\mathbf{e} \in \mathbf{R}^m$  is a **column vector** of all 1's.

The algorithm first establishes a vertex/edge diagram that describes all vertex/edge relations of a given polytope. The process is as follows: given any vertex  $\mathbf{u}_i$  on the polytope, the proposed algorithm identifies its neighbor vertices which are connected by edges. Let  $\mathbf{u}_i$  and  $\mathbf{u}_j$  be two vertices of the polytope. The line pass both  $\mathbf{u}_i$  and  $\mathbf{u}_j$  can be defined by

$$\mathcal{L} = \{\mathbf{u}_i + t(\mathbf{u}_j - \mathbf{u}_i) \mid t \in \mathbf{R}\}. \quad (3)$$

Assuming that a point  $\mathbf{z}$  is on the line of  $\mathcal{L}$  and the **row vector**  $\mathbf{z} - \mathbf{0} = \mathbf{z}$  (where  $\mathbf{0}$  is the origin and the center of the polytope) is perpendicular to the line  $\mathcal{L}$ , then  $\mathbf{z}$  satisfies the following equations:

$$\mathbf{z} = \mathbf{u}_i + t(\mathbf{u}_j - \mathbf{u}_i), \quad \langle \mathbf{z}, \mathbf{u}_j - \mathbf{u}_i \rangle = 0. \quad (4)$$

Solving (4) for  $t$  yields

$$t = -\frac{\mathbf{u}_i(\mathbf{u}_j - \mathbf{u}_i)^T}{\|(\mathbf{u}_j - \mathbf{u}_i)\|^2}, \quad (5)$$

which gives

$$\mathbf{z} = \mathbf{u}_i - \frac{\mathbf{u}_i(\mathbf{u}_j - \mathbf{u}_i)^T}{\|(\mathbf{u}_j - \mathbf{u}_i)\|^2}(\mathbf{u}_j - \mathbf{u}_i). \quad (6)$$

We have the following claim:

**Lemma 1.**  $\mathbf{z} \neq \mathbf{0}$  if  $\mathcal{L}$  does not cross the centroid (origin).

*Proof.* Since  $\mathbf{z}$  is on  $\mathcal{L}$  and  $\mathcal{L}$  does not cross  $\mathbf{0}$ , it must have  $\mathbf{z} \neq \mathbf{0}$ . □

Thus,  $\mathbf{z} = \mathbf{0}$  implies that  $\mathbf{u}_j$  is not a neighbor vertex of  $\mathbf{u}_i$  (this will reduce the computational effort if such a scenario appears). Therefore, we may assume, without loss of generality, that  $\mathbf{z} \neq \mathbf{0}$ . Denote a **row vector**  $\mathbf{a} = \mathbf{z}/\|\mathbf{z}\| \neq \mathbf{0}$  and

a constant  $c = \mathbf{az}^T = \frac{\mathbf{zz}^T}{\|\mathbf{z}\|} = \|\mathbf{z}\| \neq 0$ . Then, a hyperplane  $\mathcal{P}$  passing line  $\mathcal{L}$  with the normal direction  $\mathbf{a}^T$  is given by

$$\mathbf{a}(\mathbf{x} - \mathbf{z}^T) = \mathbf{ax} - c = 0, \quad (7)$$

where  $\mathbf{a}$  is a known **row vector**,  $c$  is a constant, and  $\mathbf{x} \in \mathbf{R}^d$  is any point on the hyperplane  $\mathcal{P}$ . Dividing both sides by  $c$  and denote  $\mathbf{h}^T = \mathbf{a}/c$ , i.e.,  $\mathbf{h}$  is a **column vector**, we get

$$\mathbf{h}^T \mathbf{x} = 1. \quad (8)$$

Intuitively, from the construction of  $\mathcal{P}$ , if for all  $\mathbf{u}_k$ , we have

$$\mathbf{h}^T \mathbf{u}_k^T \leq 1 \quad (9)$$

with equality hold for only  $\mathbf{u}_i$  and  $\mathbf{u}_j$ , then the segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  is an edge of the polytope and  $\mathbf{u}_i$  and  $\mathbf{u}_j$  are adjacent vertices. We summarize the discussion as the following theorem.

**Theorem 1.** *Let  $\mathbf{u}_i$  and  $\mathbf{u}_j$  be two vertices of a convex polytope. Let  $\mathcal{L}$  be the line as defined in (3). For  $t$  given in (5),  $\mathbf{z} = \mathbf{u}_i + t(\mathbf{u}_j - \mathbf{u}_i)$  is perpendicular to  $\mathcal{L}$ . Denote  $\mathbf{a} = \mathbf{z}/\|\mathbf{z}\|$  and  $c = \mathbf{az}^T$  ( $\mathbf{a}$  and  $\mathbf{z}$  are row vectors). Let  $\mathbf{h}^T = \mathbf{a}/c$ . Then,*

$$\mathbf{h}^T \mathbf{x} = 1, \quad \forall \mathbf{x} \in \mathbf{R}^d \quad (10)$$

*is a hyperplane passing  $\mathbf{u}_i$  and  $\mathbf{u}_j$ , and the following claims hold:*

(a) *If inequality (9) holds for all  $\mathbf{u}_k$ , and the equality holds for only  $\mathbf{u}_i$  and  $\mathbf{u}_j$ , then the segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  is an edge of the polytope, i.e.,  $\mathbf{u}_i$  and  $\mathbf{u}_j$  are adjacent vertices.*

(b) *If inequality (9) holds for all  $\mathbf{u}_k$ , and the equality holds for more than two but less than  $d$  vertices, then the line segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  is on a  $k$ -face which is part of the hyperplane.*

(c) *If inequality (9) holds for all  $\mathbf{u}_k$ , and the equality holds for at least  $d$  vertices, then the line segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  is on the hyperplane and a facet of the polytope is part of the hyperplane.*

*Proof.* We show part (a) only because parts (b) and (c) follows similar argument. Since (9) holds for all vertices  $k = 1, \dots, n$ , all points of the polytope are inside of the half space  $\mathbf{h}^T \mathbf{x} \leq 1$ . The half space contains the polytope. Since  $\mathbf{u}_i$  and  $\mathbf{u}_j$  are on the hyperplane, all points on the line segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  are on the hyperplane. Since  $\mathbf{h}^T \mathbf{u}_k^T < 0$  holds for all  $\mathbf{u}_k$  satisfying  $\mathbf{u}_i \neq \mathbf{u}_k \neq \mathbf{u}_j$ , for any point in the convex hull such that

$$\sum_{k=1}^n \lambda_k \mathbf{u}_k$$

with at least one  $\lambda_k > 0$  and  $k \notin \{i, j\}$ , we have  $\mathbf{h}^T \sum_{k=1}^n \lambda_k \mathbf{u}_k^T < 1$  because  $\sum_{k=1}^n \lambda_k = 1$ . Therefore, all those points in the convex polytope are not on the hyperplane. Since for all points of the polytope, only line segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  are on the hyperplane, the line segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  is an edge of the polytope.  $\square$

*Remark 1.* For the purpose of facet enumeration, we are only interested in cases (a) and (c) because case (a) of Theorem 1 implies that  $\mathbf{u}_j$  is a neighbor vertex of  $\mathbf{u}_i$ , which is used to construct vertex/edge diagram, and case (c) will be recorded to reduce computational effort.

If inequality (9) does not hold for at least one of  $\mathbf{u}_k$ ,  $k = 1, \dots, n$ , then we need to be a little bit more careful. Denote the middle point between  $\mathbf{u}_j$  and  $\mathbf{u}_i$  as a **row vector**  $\mathbf{r} = [r_1, r_2, \dots, r_d] = \frac{\mathbf{u}_j + \mathbf{u}_i}{2}$ , which is also the vector from the origin to the middle point of  $\mathbf{u}_j$  and  $\mathbf{u}_i$ <sup>2</sup>. Therefore, for a **row vector**

$$\mathbf{y} \neq \mathbf{r}, \quad (11)$$

then,  $\mathbf{y} - \mathbf{r}$  is a vector starting from the middle point of  $\mathbf{u}_j$  and  $\mathbf{u}_i$  and pointing to  $\mathbf{y}$ . We consider all  $\mathbf{y}$  such that  $\mathbf{y} - \mathbf{r}$  is perpendicular to  $\mathbf{u}_j - \mathbf{u}_i$ . This implies that  $\mathbf{y}$  satisfies

$$\left\langle \mathbf{y} - \mathbf{r}, \mathbf{u}_j - \mathbf{u}_i \right\rangle = 0, \quad (12)$$

where  $\langle \cdot, \cdot \rangle$  denotes the inner product of its arguments<sup>3</sup>. This is equivalent to

$$\mathbf{y}(\mathbf{u}_j - \mathbf{u}_i)^T = \mathbf{r}(\mathbf{u}_j - \mathbf{u}_i)^T. \quad (13)$$

We want to find a bounded vector  $\mathbf{y} - \mathbf{r}$  such that the smallest angle between  $\mathbf{y} - \mathbf{r}$  and  $\mathbf{u}_k - \mathbf{r}$ , for  $k \notin \{i, j\}$ , is maximized, which is equivalent to solving the following min-max problem:

$$\min_{\mathbf{y}} \max_{k \notin \{i, j\}} \left\langle \frac{\mathbf{y} - \mathbf{r}}{\|\mathbf{y} - \mathbf{r}\|}, \frac{\mathbf{u}_k - \mathbf{r}}{\|\mathbf{u}_k - \mathbf{r}\|} \right\rangle. \quad (14)$$

However, in stead of considering the nonlinear problem (14), we prefer to deal with a simpler problem. Let

$$f = \max_{k \notin \{i, j\}} \left\langle \frac{\mathbf{y} - \mathbf{r}}{\|\mathbf{y} - \mathbf{r}\|}, \frac{\mathbf{u}_k - \mathbf{r}}{\|\mathbf{u}_k - \mathbf{r}\|} \right\rangle. \quad (15)$$

Since  $\|\mathbf{y} - \mathbf{r}\|$  is independent of  $k$ , we may consider the following linear programming problem:

$$\begin{aligned} \min_{\mathbf{y}, f} \quad & f \\ \text{s.t.} \quad & (\mathbf{u}_j - \mathbf{u}_i)\mathbf{y}^T = \mathbf{r}(\mathbf{u}_j - \mathbf{u}_i)^T, \\ & (\mathbf{u}_k - \mathbf{r})(\mathbf{y} - \mathbf{r})^T \leq f\|\mathbf{u}_k - \mathbf{r}\|, \quad k \notin \{i, j\}, \\ & -\mathbf{1} \leq \mathbf{y} - \mathbf{r} \leq \mathbf{1}, \end{aligned} \quad (16)$$

<sup>2</sup> If  $\frac{\mathbf{u}_j + \mathbf{u}_i}{2} = \mathbf{0}$ , then, according to Lemma 1,  $\{\mathbf{u}_i, \mathbf{u}_j\}$  is not an edge and will not be considered further.

<sup>3</sup> Here, we abuse the notation of inner product by not restricting its arguments to be **column vectors**. The only restriction is that they must have the same dimension.

where  $\mathbf{1} \in \mathbf{R}^d$  is a **row vector** of all ones. Let  $\mathbf{y}^*$  and  $f^*$  be the solution of (16) ( $\mathbf{y}^*$  is a constant **row vector** to be determined), then the **column vector**  $\mathbf{x}$  in the following equation

$$(\mathbf{y}^* - \mathbf{r})(\mathbf{x} - \mathbf{r}^T) = 0 \quad (17)$$

defines a plane that is perpendicular to  $\mathbf{y}^* - \mathbf{r}$  and passes the line segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$ . Substituting  $\mathbf{x}^T = \mathbf{u}_i$  to the left side of (17) and applying (12) gives

$$-\frac{1}{2} \left\langle \mathbf{y}^* - \mathbf{r}, \mathbf{u}_j - \mathbf{u}_i \right\rangle = 0. \quad (18)$$

Therefore,  $\mathbf{u}_i$  is on the plane defined by (17). Substituting  $\mathbf{x}^T = \mathbf{u}_j$  to the left side of (17) and applying (12) gives

$$\frac{1}{2} \left\langle \mathbf{y}^* - \mathbf{r}, \mathbf{u}_j - \mathbf{u}_i \right\rangle = 0. \quad (19)$$

Therefore,  $\mathbf{u}_j$  is on the plane defined by (17). Let  $\mathbf{h}^T = \mathbf{y}^* - \mathbf{r}$ , according to (11), it must have  $\mathbf{h} \neq \mathbf{0}$ . Assuming that the middle point of  $\mathbf{u}_i$  and  $\mathbf{u}_j$  is not in the origin, i.e.,  $\mathbf{r} \neq \mathbf{0}$ , we can write the plane of (17) as

$$\mathbf{h}^T \mathbf{x} = \mathbf{h}^T \mathbf{r}^T = c \neq 0. \quad (20)$$

If the minimum of (16)  $f^* < 0$ , it means that all  $\mathbf{u}_k$ ,  $k \notin \{i, j\}$ , are not on the plane, and they are all on one side of the plane, therefore, the line segment connecting  $\mathbf{u}_j$  and  $\mathbf{u}_i$  is an edge of the polytope. If  $f^* = 0$ , it means that besides  $\mathbf{u}_j$  and  $\mathbf{u}_i$ , there are additional vertices  $\mathbf{u}_k$ ,  $k \notin \{i, j\}$  on the plane; although all vertices are on the same side of the plane defined by (17),  $\mathbf{u}_j$  and  $\mathbf{u}_i$  do not form an edge of the polytope. If  $f^* > 0$ , it means that it is impossible to find a plane which passes  $\mathbf{u}_j$  and  $\mathbf{u}_i$ , such that all  $\mathbf{u}_k$  are on the same side of the plane. Therefore,  $\mathbf{u}_j$  and  $\mathbf{u}_i$  do not form an edge of the polytope.

Summarizing the above discussion, we have the following theorem.

**Theorem 2.** *Let  $\mathbf{u}_i$  and  $\mathbf{u}_j$  be any two vertices of a convex polytope,  $\mathbf{y}^*$  and  $f^*$  be the optimal solution of (16). Denote that  $c = \mathbf{h}^T \left( \frac{\mathbf{u}_i + \mathbf{u}_j}{2} \right)^T$ , then the following claims hold:*

(a) *If  $f^* < 0$ , then, the line segment connecting  $\mathbf{u}_j$  and  $\mathbf{u}_i$  is an edge of the polytope.*

(b) *If  $f^* = 0$  and  $\mathbf{y}^* - \mathbf{r} \neq \mathbf{0}$ , then,  $\mathbf{u}_j$  and  $\mathbf{u}_i$  do not form an edge of the polytope and the hyperplane contains either a  $k$ -face if less than  $d$  vertices are on the hyperplane or a facet if at least  $d$  vertices are on the hyperplane.*

(c) *If  $f^* > 0$ , then,  $\mathbf{u}_j$  and  $\mathbf{u}_i$  do not form an edge of the polytope and the hyperplane does not form a  $k$ -face or a facet of the polytope.*

*Remark 2.* If  $f^* = 0$  and  $\mathbf{y}^* - \mathbf{r} = \mathbf{0}$ , this solution of (16) does not give us the necessary information to use Theorem 2, in this case, we may add an additional

inequality constraint:

$$\sum_{k=1}^d (y_k - r_k) \geq 1 \quad (21)$$

in the linear programming problem (16) to enforce  $\mathbf{y}^* - \mathbf{r} \neq \mathbf{0}$ .

*Remark 3.* It is worthwhile to point out that the Matlab function `linprog` oftentimes fails to find the optimal solution of (16), while the facet pivot method developed in [29] has no problem for all tested problems.

*Remark 4.* Assume that  $\mathbf{y}^* \neq \mathbf{r}$  and  $f^* = 0$  be the solution of (16). Denote the index set  $\mathcal{K} = \{k \mid (\mathbf{u}_k - \mathbf{r})(\mathbf{y}^* - \mathbf{r})^T = f^* \|\mathbf{u}_k - \mathbf{r}\|, k \notin \{i, j\}\}$ . Then,  $\mathcal{F} = \{\mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_{\mathcal{K}}\}$  forms a  $k$ -face. In addition, if  $|\mathcal{F}| \geq d$ , then,  $\mathcal{F}$  forms a facet. This information will save some computational effort in the next section.

### 3 Constructing facets from adjacent matrix

First, we discuss the information storage, which is important for algorithm design. Let  $\mathbf{D}$  be the  $n \times n$  adjacency matrix whose  $(i, j)$  element is one if the line segment between  $\mathbf{u}_i$  and  $\mathbf{u}_j$  is an edge or is zero otherwise. Matrix  $\mathbf{D}$  is obtained by the process described in Theorem 1 or Theorem 2. Since  $\mathbf{D}$  is symmetric, using this property will reduce the computational effort to find all edges of the polytope.

Let us consider a vertex tree associated with  $\mathbf{D}$  and starting from vertex  $\mathbf{u}_1$ , which connects a series of adjacent vertices. Clearly the structure of the tree is defined by matrix  $\mathbf{D}$ . To efficiently carry out the search, we denote by  $\mathcal{U}_o$  the set of vertices for which all relevant facets have been found, by  $\mathcal{U}_c$  the set of vertices in the current iteration for which the associated hyperplanes are to be determined, by  $\mathcal{U}_t$  the set of the vertices that are not in  $\mathcal{U}_c$  yet and therefore whose facets have not been examined.  $\mathcal{U}_o = \emptyset$  and  $\mathcal{U}_c = \mathbf{u}_1$  before the iteration starts. The proposed algorithm has two loops. The outer loop uses breadth-first search. For a vertex  $\mathbf{u}_i \in \mathcal{U}_c$ , the algorithm in the inner loop finds all facets that intersect at  $\mathbf{u}_i$ , once all facets containing  $\mathbf{u}_i$  are found,  $\mathbf{u}_i$  is moved from  $\mathcal{U}_c$  to  $\mathcal{U}_o$ . While finding facets containing  $\mathbf{u}_i \in \mathcal{U}_c$ , new vertices on each of these new facets become members of  $\mathcal{U}_c$ . This process is repeated for every  $\mathbf{u}_i \in \mathcal{U}_c$  until for every vertex, its intersecting facets are found, i.e., the process will terminate when  $\mathcal{U}_o$  includes all vertices of  $\{\mathbf{u}_1, \dots, \mathbf{u}_n\}$ .

Having the vertex/edge diagram of  $\mathbf{D}$  that connects all the vertices in  $\mathcal{V}$ , the inner loop uses the following method to identify all the facets that intersects at  $\mathbf{u}_i \in \mathcal{U}_c$ . Since every facet that contains  $\mathbf{u}_i$  can be associated with an edge, for each of these edges directly connected to  $\mathbf{u}_i$ , we can create a branch of length  $d-1$  (with  $d-1$  edges) as follows. Note that nonzero  $(i, j)$  elements of the adjacency matrix  $\mathbf{D}$  define the branch under the vertex  $\mathbf{u}_i$ . For each  $\mathbf{u}_i$ , assuming that  $\mathbf{u}_i$  is the  $i$ th vertex, we look at the  $i$ th row of  $\mathbf{D}$  to select the next vertex  $j$  among all  $(i, j) = 1$  such that the  $j$ th vertex has not been used in the construction of the current hyperplane; once vertex  $j$  is selected, we look at the  $j$ th row of  $\mathbf{D}$

to select the next vertex  $k$  among all  $(j, k) = 1$  such that the  $k$ th vertex has not been used in the construction of the current hyperplane; repeat this step until  $d$  vertices are found.

Since hyperplane (10) is uniquely defined by  $\mathbf{h}$ , we will loosely use  $\mathbf{h}_i \in \mathbf{R}^d$  for the  $i$ th facet of the polytope if the hyperplane  $\mathbf{h}_i$  contains the facet. Let  $\mathbf{u}_{i_1}$  be one of vertices in the vertex set  $\mathcal{U}_c$  in current iteration. We say that  $\{\mathbf{u}_{i_1}, \mathbf{u}_{i_2}, \dots, \mathbf{u}_{i_d}\}$  is a branch of length  $d - 1$  in the tree under  $\mathbf{u}_{i_1}$  if for  $i_j \in \{i_2, \dots, i_{d-1}\}$ ,  $\mathbf{u}_{i_j}$  is connected to only  $\mathbf{u}_{i_{j-1}}$  and  $\mathbf{u}_{i_{j+1}}$  in the set of  $\{\mathbf{u}_{i_1}, \mathbf{u}_{i_2}, \dots, \mathbf{u}_{i_d}\}$ . Given these  $d$  vertices  $\{\mathbf{u}_{i_1}, \dots, \mathbf{u}_{i_d}\}$  which are on the branch of the tree starting from  $\mathbf{u}_{i_1}$ , if the base matrix formed by  $\{\mathbf{u}_{i_1}, \dots, \mathbf{u}_{i_d}\}$  is linearly independent, one can solve the linear system

$$\begin{bmatrix} \mathbf{u}_{i_1} \\ \vdots \\ \mathbf{u}_{i_d} \end{bmatrix} \mathbf{h}_i := \mathbf{U}_i \mathbf{h}_i = \mathbf{e}. \quad (22)$$

for a candidate facet  $\mathbf{h}_i$ . If  $\mathbf{h}_i^T \mathbf{u}_k^T \leq 1$  for  $k = 1, \dots, n$ , then  $\mathbf{h}_i$  is the hyperplane that includes a facet. Since (22) may also create an unwanted hyperplane, these unwanted hyperplanes can be identified using one of the following rules<sup>4</sup>: first, inequality  $\mathbf{h}_i^T \mathbf{u}_i^T \leq 1$  is violated, i.e.,

$$\mathbf{U} \mathbf{h}_i \leq \mathbf{e} \quad (23)$$

does not hold for some  $\mathbf{u}_i \in \mathbf{U}$ ; second, the hyperplane has been found earlier in this process (in this case, the hyperplane will not be added to the matrix  $\mathbf{H}$ ). If the newly found hyperplane contains a facet of the convex polytope that is not in the matrix  $\mathbf{H}$ , it is then added to  $\mathbf{H}$ . Otherwise, discard the hyperplane and continue the search in the tree.

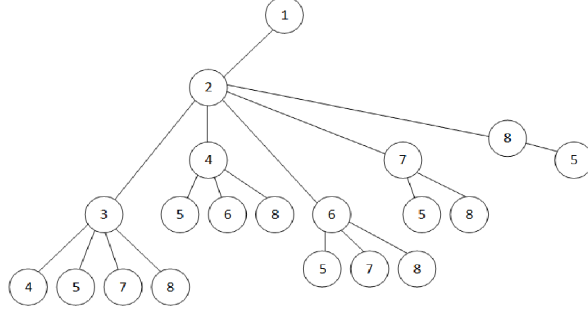
For  $\mathbf{u}_i \in \mathcal{U}_c$ , the idea of the proposed algorithm is to examine all branches  $\mathbf{u}_{i_1}, \dots, \mathbf{u}_{i_d}$  under  $\mathbf{u}_i = \mathbf{u}_{i_1} \in \mathcal{U}_c$  in a systematical method to reduce the effort to find facets associated with  $\mathbf{u}_i$  that has not been found. However, given the vertex  $\mathbf{u}_i \in \mathcal{U}_c$ , we may not need to calculate all hyperplanes associated with it because some facets associated with  $\mathbf{u}_i$  may be on other hyperplanes defined by some  $\mathbf{h}_j$  which have been found in current or previous iterations.

We use cross-polytope of Example 4 to describe the process, for which the  $\mathbf{D}$  matrix is obtained by using the method given in the previous section:

$$\mathbf{D} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}. \quad (24)$$

<sup>4</sup> This is essentially the same idea of beneath-and-beyond [6, 19]; but because of the use of the adjacency matrix  $\mathbf{D}$ , the search number for potential facets has been significantly reduced.

Assume  $\mathcal{U}_c = \{\mathbf{u}_1\}$ , the branches are constructed as follows: In view of the first



**Fig. 1.** The branches under  $\mathbf{u}_i = \mathbf{u}_1 \in \mathcal{U}_c$ .

row of  $\mathbf{D}$ ,  $\mathbf{u}_1$  is connected to  $\mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4, \mathbf{u}_5, \mathbf{u}_6, \mathbf{u}_7$ . We consider the branch with its first edge between  $\mathbf{u}_1$  and  $\mathbf{u}_2$  as an example. The second row of  $\mathbf{D}$  indicates that  $\mathbf{u}_2$  is also connected to  $\mathbf{u}_3, \mathbf{u}_4, \mathbf{u}_6, \mathbf{u}_7, \mathbf{u}_8$ , we select the first vertex such that the branch includes  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ . The third row of  $\mathbf{D}$  indicates that  $\mathbf{u}_3$  is connected to  $\mathbf{u}_4, \mathbf{u}_5, \mathbf{u}_7, \mathbf{u}_8$ , again, we select the first vertex such that  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4$  forms the first branch with length  $d = 3$ . Using the aforementioned method, we can examine if this branch forms a hyperplane. If it does, save it; otherwise, discard it. To construct the next branch, we move one level up at a time and check if there is any vertex that has not been considered on this level. In this case,  $\mathbf{u}_5$  is the first vertex that has not been considered, a new branch  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_5$  is formed; and the new branch is examined to see if it forms a new hyperplane. Repeating this process, the order of the selected branches is described in Figure 1 that is from the left to the right. After all the branches under  $\mathbf{u}_1$  has been examined, 8 facets are found. Set  $\mathcal{U}_0 = \{\mathbf{u}_1\}$ . From Figure 1, it is clear that  $\mathcal{U}_c = \{\mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4, \mathbf{u}_5, \mathbf{u}_6, \mathbf{u}_7, \mathbf{u}_8\}$ . The above process can be repeated for every  $\mathbf{u}_i \in \mathcal{U}_c$  until  $\mathcal{U}_0 = \{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4, \mathbf{u}_5, \mathbf{u}_6, \mathbf{u}_7, \mathbf{u}_8\}$ .

Denote the number of members of a set  $\mathcal{U}$  by  $|\mathcal{U}|$ . The proposed algorithm is therefore given as follows:

### Algorithm 1

- 1: *Data: Vertex matrix  $\mathbf{V}$ .*
- 2: *Initial step: Calculate centroid  $\mathbf{v}_0$  and vertex set  $\mathcal{U}$ , adjacent matrix  $\mathbf{D}$ , and initial facet matrix  $\mathbf{H}$ . Set up initial  $\mathcal{U}_0, \mathcal{U}_c$ , and  $\mathcal{U}_t$ .*
- 3: **while**  $|\mathcal{U}_0| < n$  **do**
- 4:     set  $\mathcal{U}_2 = \emptyset$ .
- 5:     **for**  $i = 1: |\mathcal{U}_c|$  **do**

- 6: *Step 1: Start from  $\mathbf{u}_{i_1}$ , from row  $i_1$  of  $\mathbf{D}$ , find the first vertex  $\mathbf{u}_{i_2}$  that has not been considered in the branches under  $\mathbf{u}_{i_1}$  so that the new branch includes  $\mathbf{u}_{i_1}$  and  $\mathbf{u}_{i_2}$ .*
- 7: *Step 2: Repeat the process until a new branch  $\mathbf{u}_{i_1}, \dots, \mathbf{u}_{i_d}$  is formed.*
- 8: *Step 3: Solve (22) to get a candidate facet  $\mathbf{h}_i$ . Test if  $\mathbf{h}_i$  is indeed a new facet. If it is, add it to  $\mathbf{H}$ ; otherwise discard it.*
- 9: *Step 4: Add new vertices to the set  $\mathcal{U}_2$  whose members are on the hyperplanes of the new branches but are not in  $\mathcal{U}_c$ .*
- 10: *Step 5: Move one level up at a time in the current branch until either (1) there is a vertex on this level that has not been included in a branch under  $\mathbf{u}_{i_1}$ , then repeat Steps 2-4; or (2) all branches under  $\mathbf{u}_{i_1}$  have been investigated, increase  $i$  by 1.*
- 11: **end for**
- 12: *Update  $\mathcal{U}_o = \mathcal{U}_o \cup \mathcal{U}_c$ . Set  $\mathcal{U}_c = \mathcal{U}_2$ .*
- 13: **end while**
- 14: *Recover the polytope by shifting the origin  $\mathbf{H}(\mathbf{x} - \mathbf{v}_0) \leq \mathbf{e}$ .*

## 4 Complexity analysis

First, it is easy to derive the complexity for edge-detection process.

**Theorem 3.** *Given  $n$  vertices of the polytope, it needs to solve  $\frac{n(n-1)}{2}$  linear programming problems (16) to find all edges of the polytope to form the adjacency matrix  $\mathbf{D}$ .*

*Proof.* Given  $n$  vertices, to find all edges of the polytope, one needs to examine all pairs of vertices. According to Theorem 2, one needs to solve

$$(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$$

linear programming problems in the form of (16), with  $d+1$  variables,  $n-2$  inequality, 1 equality, and  $2n$  boundary constraints.  $\square$

*Remark 5.* Our edge detection process solves roughly the same amount of linear programming problems as in [15] but our method does not need to know all edge direction information as required by the edge-skeleton method of [15].

*Remark 6.* There are two popular types of algorithms for solving the linear programming problem, namely the simplex method [12], and the interior-point algorithms [26, 28]. It is widely believed that for large-scale linear programming problems with tens of thousands variables and constraints, the interior-point algorithms with complexity of  $O(\sqrt{n} \log(\frac{1}{\epsilon}))$  are more efficient [27]; but for smaller linear programming problems, simplex algorithms are better choices, because of the McMullen's upper bound theorem [21], it is impracticable to solve facet enumeration problems for polytope in tens of thousands dimensional space. Therefore, simplex method is considered in our implementation. We selected a recently developed facet pivot simplex method because of its merits discussed in [29].

The complexity analysis for reconstructing the facets from adjacent matrix  $D$  is slightly trickier. Assume that a branch includes vertices  $\mathbf{u}_{i_1}, \mathbf{u}_{i_2}, \dots, \mathbf{u}_{i_d}$ . In view of (24), a *feasible branch* is defined by  $d - 1$  elements

$$\{D(i_1, i_2), D(i_2, i_3), \dots, D(i_{d-1}, i_d)\}$$

in the adjacent matrix  $\mathbf{D}$  with  $D(i_1, i_2) = D(i_2, i_3) = \dots = D(i_{d-1}, i_d) = 1$ , where  $D(i_\alpha, i_\beta)$  is the  $(i_\alpha, i_\beta)$  elements of the  $\mathbf{D}$  matrix, and  $\{i_1, i_2, \dots, i_{d-1}, i_d\}$  are distinct. This means that  $\{D(i_1, i_2), D(i_2, i_3), \dots, D(i_{d-1}, i_d)\}$  are in different rows and different columns in the adjacent matrix  $\mathbf{D}$ .

**Theorem 4.** *Assume that the adjacent matrix  $\mathbf{D}$  (that describes the structure of the polytope) is given. Let  $N$  be the total number of all feasible branches. To reconstruct the  $H$ -representation of the polytope, Algorithm 1 needs to examine at most  $N$  branches by using Beneath-and-Beyond method.*

*Proof.* We start to consider a cross-polytope of Example 4, and examine a set of branches starting with  $\mathbf{u}_1$  and composed of  $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_5$  or denoted by  $1, 2, 3, 5$  for brevity. There are 6 possible branches  $\{1 \rightarrow 2 \rightarrow 3 \rightarrow 5, 1 \rightarrow 2 \rightarrow 5 \rightarrow 3, 1 \rightarrow 3 \rightarrow 2 \rightarrow 5, 1 \rightarrow 3 \rightarrow 5 \rightarrow 2, 1 \rightarrow 5 \rightarrow 2 \rightarrow 3, 1 \rightarrow 5 \rightarrow 3 \rightarrow 2\}$ . Among them, only  $\{1 \rightarrow 2 \rightarrow 3 \rightarrow 5, 1 \rightarrow 2 \rightarrow 5 \rightarrow 3\}$  are feasible branches because  $D(1, 2)D(2, 3)D(3, 5) = 1$ ,  $D(1, 5)D(5, 3)D(3, 2) = 1$ ,  $D(2, 5) = 0$ , and  $D(5, 2) = 0$ . This idea can be applied to any branch starting from any vertex for the cross-polytope of Example 4. Moreover, given the adjacent matrix  $\mathbf{D}$  (that describes the structure of the polytope), this idea can be applied to any convex polytope. This shows that Algorithm 1 needs to examine at most  $N$  branches.  $\square$

*Remark 7.* From the proof of Theorem 4, one can see that there may be multiple feasible branches that represent the same facet. Therefore, in the implementation, for every new feasible branch obtained in Step 2 of the algorithm, we may first check before Step 3 if all its vertices in the feasible branch are considered in some identified facet. If the answer is yes. We may ignore the remainder steps and search for a new feasible branch.

From this remark, the upper bound in Theorem 4 can be improved further. For all feasible branches, we may group them as  $M$  mutually exclusive subsets as follows: in any of the subsets, every branch includes the same vertices. We name these subsets as *equivalent sets*. For each of these equivalent sets, we need only to examine one branch.

**Theorem 5.** *Assume that the adjacent matrix  $\mathbf{D}$  (that describes the structure of the polytope) is given. Let  $N$  be the total number of all feasible branches which are grouped as  $M$  equivalent sets. To reconstruct the  $H$ -representation of the polytope, Algorithm 1 needs to examine at most  $M$  branches, and these branches are from distinct equivalent sets.*

The upper bound presented in Theorems 5 eliminates many non-feasible and equivalent feasible branches in the process of determining facets. This is particularly useful when  $\mathbf{D}$  is a sparse matrix, and we would like to point out that in many cases, matrix  $\mathbf{D} \in \mathbf{R}^{n \times n}$  is a sparse matrix. Considering Example 6, if  $p = 18$  ( $p$  is defined in Example 6, for this problem, it has  $2p + 2 = 38$  vertices), the matrix  $\mathbf{D} \in \mathbf{R}^{38 \times 38}$  is given as follows:

[illegible]

Given  $n$  vertices of a polytope in  $d$ -dimensional space, without the edge information in matrix  $\mathbf{D}$ , the search by brute force for all facets may need to examine as many as  $n^{C(n-1, d-1)}$  hyperplanes (where  $C(n-1, d-1)$  denotes the combination of  $n-1$  choosing  $d-1$ ), while using the information provided in  $\mathbf{D}$ , we examined 738 branches of length 2 (facets) in Example 6, a tremendous saving!

We conclude this section by providing an explicit overall complexity result.

**Theorem 6.** *Given a convex polytope with  $n$  input vertices (V-representation) and  $m$  output facets (H-representation), Algorithm 1 requires solving  $\frac{n(n-1)}{2}$  linear programming problems to construct vertex/edge diagram (adjacent matrix  $\mathbf{D}$ ). Assume that the adjacent matrix  $\mathbf{D}$  contains  $M > m$  equivalent sets. Algorithm 1 needs to examine at most  $M$  feasible branches by using Beneath-and-Beyond method to identify  $m$  facets.*

*Remark 9.* By examining the proofs in this section, we conclude that the complexity bound is both accurate and attainable, which distinguishes it from the *estimated* upper bounds commonly found in the literature. However,  $M$  is not an *a priori* known quantity, and the gap between  $M$  and  $m$  remains unknown. Determining the gap should be a future research topic.

## 5 Numerical examples

Several examples are provided in this section. Testing was performed on an old Dell Inc 2014, Intel(R) Core(TM) i5-4440 CPU @3.10GHz, 3101Mhz, 4 Core(s), 4 Logical Processor(s).

*Example 1.* The first example is a  $d$ -dimensional simplex convex polytope. For  $d = 2$ , it is a triangle whose vertices are given by:

$$\mathbf{V} = \begin{bmatrix} 0 & 0 \\ 3 & 0 \\ 0 & 3 \end{bmatrix}$$

The centroid is  $(1, 1)$ . Algorithm 1 finds the H-representation as

$$\mathbf{H} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \\ 1 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

in 0.0050 seconds. For  $d > 2$ , using Matlab notation, the vertices of the simplex are given by:

$$\mathbf{V} = \begin{bmatrix} \text{zeros}(1, d+1) \\ \text{eys}(d+1) \end{bmatrix}.$$

Algorithm 1 finds the H-representation as expected.

*Example 2.* The second example is a 3-dimensional convex polytope that is a cube. Its vertices are given by:

$$\mathbf{V} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

The centroid is  $(0.5, 0.5, 0.5)$ . Algorithm 1 finds the H-representation as

$$\mathbf{H} = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \\ 0 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

*Example 3.* This example is a 3-dimensional convex polytope which is a octahedron. Its vertices are given by:

$$\mathbf{V} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

The centroid is  $(0, 0, 0)$ . Algorithm 1 finds the H-representation as

$$\mathbf{H} = \begin{bmatrix} -1 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & -1 \\ 1 & -1 & -1 \\ 1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

It is worthwhile noting that this is a degenerate polytope (there are 4 facets intercepting at the same vertex of the polytope in the 3-dimensional space). Since the proposed algorithm finds all edges of the polytope which are used to construct the facets, it has no difficulty at all for degenerate polytopes.

*Example 4.* This example is a 4-dimensional convex polytope which is a cross-polytope. Its vertices are given by:

$$\mathbf{V} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

The centroid is  $(0,0,0,0)$ . Algorithm 1 finds the H-representation as

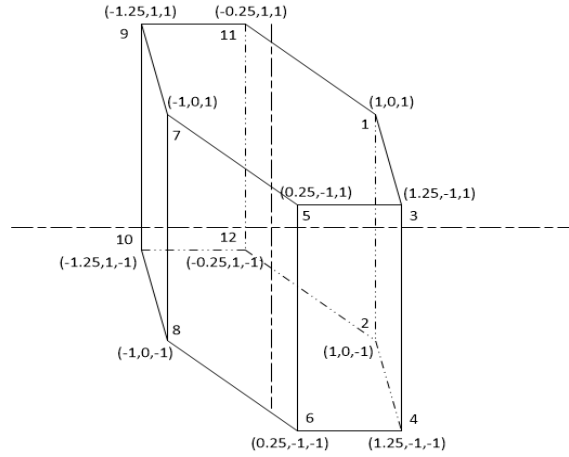
$$\mathbf{H} = \begin{bmatrix} -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

For the above problems, using Theorem 1 can find all the edges of the corresponding polytopes. However, for the following convex polytope, we need Theorem 2 to find some edges.

*Example 5.* This example is a 3-dimensional convex polytope. Its vertices are given by:

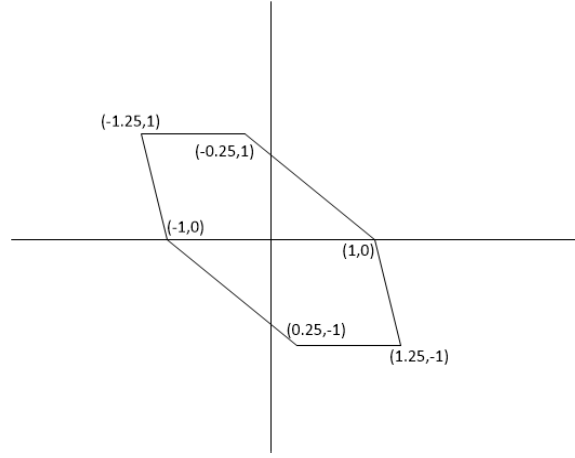
$$\mathbf{V} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & -1 \\ 1.25 & -1 & 1 \\ 1.25 & -1 & -1 \\ 0.25 & -1 & 1 \\ 0.25 & -1 & -1 \\ -1 & 0 & 1 \\ -1 & 0 & -1 \\ -1.25 & 1 & 1 \\ -1.25 & 1 & -1 \\ -0.25 & 1 & 1 \\ -0.25 & 1 & -1 \end{bmatrix}$$

The centroid is  $(0,0,0)$ . The polytope is depicted as in Figure 2. Each vertex is numbered and its coordinate is provided in the figure. There are 12 vertices, 8 facets and 18 edges in this polytope. Let  $\{i, j\}$  represent the line segment that connects vertices  $i$  and  $j$ . Using Theorem 1, we can determine 14 edges. However, for the rest of the 4 edges,  $\{1, 2\}$ ,  $\{5, 6\}$ ,  $\{7, 8\}$ , and  $\{11, 12\}$ , we have to use Theorem 2 to identify them.



**Fig. 2.** The polytope of Problem 5.

The projected figure of the polytope in the x-y plane is depicted in Figure 3 which may help us verify the upcoming results. First, we consider the  $\{1, 2\}$  edge.



**Fig. 3.** The projected figure of Problem 5.

Given  $i = 1$  and  $j = 2$ , i.e.,  $\mathbf{u}_1 = [1, 0, 1]$  and  $\mathbf{u}_2 = [1, 0, -1]$ , using Theorem 1, we cannot determine if  $\{1, 2\}$  is an edge. It is not difficult (but tedious) to verify that the corresponding linear programming problem (16) can be written as

$$\begin{aligned}
 & \min_{\mathbf{y}, f} f \\
 & s.t. \quad \begin{bmatrix} 0.25 & -1 & 1 \\ 0.25 & -1 & -1 \\ -0.75 & -1 & 1 \\ -0.75 & -1 & -1 \\ -2 & 0 & 1 \\ -2 & 0 & -1 \\ -2.25 & 1 & 1 \\ -2.25 & 1 & -1 \\ -1.25 & 1 & 1 \\ -1.25 & 1 & -1 \end{bmatrix} \begin{bmatrix} y_1 - 1 \\ y_2 \\ y_3 \end{bmatrix} \leq f \quad \begin{bmatrix} 1.43614 \\ 1.43614 \\ 1.60078 \\ 1.60078 \\ 2.23607 \\ 2.23607 \\ 2.65753 \\ 2.65753 \\ 1.88746 \\ 1.88746 \end{bmatrix}, \\
 & \quad [0, 0, -2][y_1, y_2, y_3]^T = 0, \\
 & \quad \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix} \leq \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \leq \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}
 \end{aligned} \tag{26}$$

which has the optimal solution  $\mathbf{y}^* = [2, 0.6821, 0]$  and  $f^* = -0.300878$ . According to Theorem 2.(a), the line segment  $\{1, 2\}$  is an edge.

Given  $i = 1$  and  $j = 5$ , i.e.,  $\mathbf{u}_1 = [1, 0, 1]$  and  $\mathbf{u}_5 = [0.25, -1, 1]$ , using Theorem 1, we cannot determine if  $\{1, 5\}$  is an edge. It is not difficult (but tedious) to verify that the corresponding linear programming problem (16) can

be written as

$$\begin{aligned}
& \min_{\mathbf{y}, f} f \\
& s.t. \quad \begin{bmatrix} 0.375 & 0.5 & -2 \\ 0.625 & -0.5 & 0 \\ 0.625 & -0.5 & -2 \\ -0.375 & -0.5 & -2 \\ -1.625 & 0.5 & 0 \\ -1.625 & 0.5 & -2 \\ -1.875 & 1.5 & 0 \\ -1.875 & 1.5 & -2 \\ -0.875 & 1.5 & 0 \\ -0.875 & 1.5 & -2 \end{bmatrix} \begin{bmatrix} y_1 - 0.625 \\ y_2 + 0.5 \\ y_3 - 1 \end{bmatrix} \leq f \\
& \quad \begin{bmatrix} -0.75, -1, 0 \end{bmatrix} [y_1, y_2, y_3]^T = 0.0313, \\
& \quad \begin{bmatrix} -0.375 \\ -1.5 \\ 0 \end{bmatrix} \leq \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \leq \begin{bmatrix} 1.625 \\ 0.5 \\ 2 \end{bmatrix}
\end{aligned} \tag{27}$$

which has the optimal solution  $\mathbf{y}^* = [0.625, -0.5, 1]$  and  $f^* = 2.7756e - 16$ . According to Theorem 2.(a), the line segment  $\{1, 5\}$  is not an edge.

Using the same strategy, we can determine that  $\{5, 6\}$ ,  $\{7, 8\}$ , and  $\{11, 12\}$  are also edges. Algorithm 1 finds the H-representation as

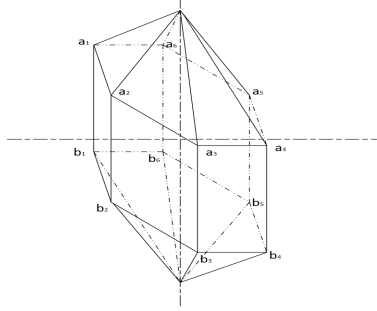
$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0.25 & 0 \\ 0 & -1 & 0 \\ -1 & -1.25 & 0 \\ -1 & -0.25 & 0 \\ 0 & 1 & 0 \\ 1 & 1.25 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

*Example 6.* The last example is a convex polytope in 3-dimensional space as described as in Figure 4, where  $a_1, a_2, \dots, a_p$  and  $b_1, b_2, \dots, b_p$  are two regular  $p$ -polygons. The polytope has  $3p$  facets,  $2p+2$  vertices, and  $5p$  edges. For  $p \geq 20$ , the CPU time is given in Table 1.

| p           | 20  | 40  | 80  | 100 | 120  | 140  | 160  | 180  |
|-------------|-----|-----|-----|-----|------|------|------|------|
| # of facets | 60  | 120 | 240 | 300 | 360  | 420  | 480  | 540  |
| CPU (s)     | 0.2 | 0.6 | 3.4 | 6.2 | 10.7 | 16.9 | 24.5 | 36.3 |

**Table 1.** CPU time increases with  $p$ .

This problem is highly degenerate, for  $p = 180$ , the top and bottom vertices each intercepts with 180 facets, the algorithm finds all facets of the polytope without any difficult (in about 36 second on this very old PC).

**Fig. 4.** Polytope of problem 6.

In summary, for all tested convex polytopes with known H-representations and V-representation, the algorithm performs as expected.

## 6 Conclusion

In this paper, an intuitive and novel facet enumeration algorithm is proposed. The idea is purely based on geometric observation; therefore, it is easy to understand. The outer loop of algorithm is based on breadth-first search which eventually covers all the vertices of the polytope. The inner loop of the algorithm is based on depth-first search which will find the facets associated with the vertex under consideration. The complexity of the algorithm is analyzed which shows the efficiency of the proposed algorithm. The algorithm is implemented in Matlab<sup>5</sup> and numerical test shows the efficiency and effectiveness of the algorithm. Due to the duality between the vertex enumeration problem and facet enumeration problem, we expect that this method can also be used to solve the vertex enumeration problem with some minor tweaks.

**Disclosure of Interests.** The author declares that he has no conflict of interest.

## References

1. Avis, D.: <http://cgm.cs.mcgill.ca/~avis/C/lrs.html>, last accessed 2023/10/25
2. Avis, D., Fukuda, K.: A pivoting algorithm for convex hull and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom* **8**(3), 295 – 313 (1992)
3. Avis, D., Jordan, C.: mpls: A scalable parallel vertex/facet enumeration code. *Mathematical Programming Computation*, **10**(2), 267 – 302 (2018)

<sup>5</sup> The implemented code is available in Github <https://github.com/yaguangyang/facetEnumeration>.

4. Barber, C. B., Dobkin, D. P., Huhdanpaa, H.: The quickhull algorithm for convex hulls, *ACM Transactions on Mathematical Software*, **22**(4), 469 – 483 (1996)
5. Bland, R. G.: New Finite Pivoting Rules for the Simplex Method. *Mathematics of Operations Research*, **2**(2), 103 – 107 (1977)
6. Borgwardt, K.: Average-case analysis of the double description method and the beneath-beyond algorithm. *Discrete & Computational Geometry* 37 (2): 175–204 (2007)
7. Bremner, D., Fukuda, K., Marzetta, A.: Primal-dual methods for vertex and facet enumeration. *Discrete Comput. Geom.*, **20**(10), 333 – 357 (1998)
8. Ceder, G., Garbulsky, G., Avis, D., Fukuda, K.: Ground states of a ternary fcc lattice model with nearest- and next-nearest-neighbor interactions. *Phys Rev B Condens Matter* **49**(1), 1 – 7 (1994)
9. Chan, T.: Optimal output-sensitive convex hull algorithms in two and three dimensions, *Discrete & Computational Geometry*, **16**(4), 361 – 368 (1996)
10. Chand, D. R., Kapur, S.: An algorithm for convex polytopes, *J. Assoc. Comput. Mach.*, **17**(1), 78 – 86 (1970)
11. Chazelle, B.: An optimal convex hull algorithm in any fixed dimension, *Discrete & Computational Geometry*, **10**(1), 377 – 409 (1993)
12. Dantzig, G.B.: *Linear programming and extension*. Princeton University Press, New Jersey (1963)
13. Deza, M.M., Laurent, M.: *Geometry of cuts and metrics*. Springer New York, (1997)
14. Dyer, M.E.: The complexity of vertex enumeration methods. *Mathematics of Operations Research*, **8**(3), 381-402, (1983)
15. Emiris, I. Z., Fisikopoulos, V., Gärtner, B.: Efficient edge-skeleton computation for polytopes defined by oracles, *Journal of Symbolic Computation*, Volume 73, 139-152, (2016)
16. Fukuda, K., Rosta, V.: Combinatorial face enumeration in convex polytopes. *Computational Geometry: Theory and Applications*, **4**(4), 191-198 (1994)
17. Huggins, P.: iB4e: A Software Framework for Parametrizing Specialized LP Problems, . In: Iglesias, A., Takayama, N. (Eds.), *Mathematical Software – ICMS*. In: *Lect. Notes Comput. Sci.*, vol. 4151. 245–247 (2006)
18. Joswig M.: Beneath-and-Beyond Revisited, In: Joswig, M., Takayama, N. (eds) *Algebra, Geometry and Software Systems*. Springer, Berlin, Heidelberg, 2003.
19. Joswig, M., Kaibel, V., Korner, F.: On the k-systems of a simple polytope, *Israel Journal of Mathematics*, vol. 129, 109 - 117 (2002)
20. Kirkpatrick, D., Seidel, R.: The ultimate planar convex hull algorithm? *SIAM Journal on Computing*. **15**(1), 287–299 (1986)
21. McMullen, P.: The maximum numbers of faces of a convex polytope, *Mathematika*, **17**(34), 179-184 (1970)
22. Motzkin, T.S., Raiffa, H., Thompson, G.L., Thrall, R. M.: The Double Description Method. *Annals of Math. Studies* 8, 51-73, Princeton University Press, (1953)
23. Seidel, R.: Constructing higher-dimensional convex hulls at logarithmic cost per face, *Proceedings of the eighteenth annual ACM symposium on theory of computing*, (1986)
24. Swart, G. F.: Finding the convex hull facet by facet. *J. Algorithms*, **6**(1) 17 – 48 (1985)
25. Tzeng, S., Owens, J.D.: Finding convex hulls using Quickhull on the GPU, *arXiv preprint arXiv:1201.2936* (2012)
26. Wright, S.: *Primal-dual interior-point methods*. SIAM, Philadelphia (1997)

- 27. Yang, Y.: Two computationally efficient polynomial-iteration infeasible interior-point algorithms for linear programming, Numerical Algorithms, Vol. 79, pp. 957–992 (2018).
- 28. Yang, Y.: Arc-search techniques for interior-point method. CRC Press, Florida (2020)
- 29. Yang, Y.: On the pivot simplex method for linear programming I: algorithms and numerical test, arXiv:2107.08468v2, (2020)
- 30. Ziegler, G.M.: Lectures on polytopes. Springer, New York (1995)