

Chapter 2

New efficient zonotope vertex enumeration: an edge-based approach

Introduction

In this thesis, one major objective is to reconstruct muscles or cables according to given postures and associated force feasible sets. The formal definition of a force feasible set located at the end-effector is as follows: for $\mathcal{T} \subset \mathbb{R}^m$ the set of all exorable tensions by m muscles, the force feasible set $\mathcal{F} \subset \mathbb{R}^3$ at the end effector of a n degrees-of-freedom kinematic chain is the set

$$\mathcal{F} = \{\mathbf{f} \in \mathbb{R}^3 \mid J^T \mathbf{f} = -L^T \mathbf{t} - \mathbf{G}, \quad \mathbf{t} \in \mathcal{T}\}$$

where $J^T \in \mathbb{R}^{n \times 3}$ is the projection of the end-effector Cartesian forces onto the torque space, $L^T \in \mathbb{R}^{n \times m}$ is the projection of muscle tensions onto the torque space and $\mathbf{G} \in \mathbb{R}^n$ is the gravity torque vector.

The reconstruction process occurs in silico. Seen as an optimization process, we shall compute force feasible sets of a musculoskeletal model in various postures and compare them with reference ones. If some musculoskeletal parameters minimize a metric between in silico and experimental force feasible sets, then it can be assumed that the reconstruction succeeded for the given postures and the given metric. In the same manner, we can apply the reconstruction process onto the *torque feasible set*, corresponding to $\{-L^T \mathbf{t} - \mathbf{G}, \quad \mathbf{t} \in \mathcal{T}\}$.

In any case, we suppose the tension set \mathcal{T} to be convex, inducing the force and torque feasible sets to share this property. Consequently, the metric used for comparison should be able to match their surfaces. There remains to compute the surface, which depends greatly on the shape of \mathcal{T} .

In this chapter, it will be assumed that \mathcal{T} takes the shape of an m -orthotope *i.e.* a hyperrectangle in m dimensions. This definition implies the two following statements:

1. For each muscle tension t_i of a muscle m_i , there exist positive lower and upper bounds, noted \underline{t}_i and \bar{t}_i , defining the feasible tensions exorable by muscle m_i . The upper bound corresponds to the feasible muscle tension when it is fully activated, while the lower bound relates to its current passive tension when it is not activated;

2. The shape of an hyperrectangle models how muscles can act together. Essentially, it allows all muscles to exert their maximal tension at the same time.

The second point is biomechanically a strong assumption. While in a cable robot it could be considered, human muscle activations differ by a neuromotor control whose law is yet to be fully understood.

Geometrically, when \mathcal{T} is an orthotope, the torque feasible set is a *zonotope* *i.e.* a specific type of polytope described as a projection of a higher-dimensional cube. A m -orthotope being the image of a m -cube under an invertible affine transformation, we can without loss of generality assimilate \mathcal{T} to the m -dimensional cube $[0, 1]^m$.

Zonotopes have multiple possible representations including either a non-squared matrix, a set of vertices, delimiting hyperplanes, or even *cells* which characterize the cube vertices whose projections are the zonotope vertices. Several algorithms exist to transit from one representation to another, whose goals are to be efficient in time and space. This chapter offers a new perspective on representing zonotopes using their edges, and describes an algorithm to compute them directly from a matrix representation of a zonotope. To achieve this goal, let's first dive onto zonotope formalism.

Let $\mathcal{Z} \subset \mathbb{R}^n$ be a n -zonotope, *i.e.* the Minkowski sum of n -dimensional segment vectors $\mathcal{Z} = \mathbf{c} + \bigoplus_{i=1}^m \alpha_i \mathbf{g}_i$ for $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{g}_i \in \mathbb{R}^n$ for $i = 1, \dots, m$ such that all the \mathbf{g}_i span \mathbb{R}^n , and $\alpha_i \in [0, 1]$. The vectors \mathbf{g}_i are called *generators* and are usually concatenated into columns of a matrix $G \in \mathbb{R}^{n \times m}$. For consiveness, a zonotope is denoted using directly its translation and its generators as $\mathcal{Z}(\mathbf{c}, G)$, or $\mathcal{Z}(G)$ if there is no translation. The notation $\mathcal{Z}(n, m)$ is also convenient to directly refer to the size of the matrix G . It is assumed that generators are non-null and that any combination of n of them span \mathbb{R}^n . In this case, the generators are said to be in *general position*.

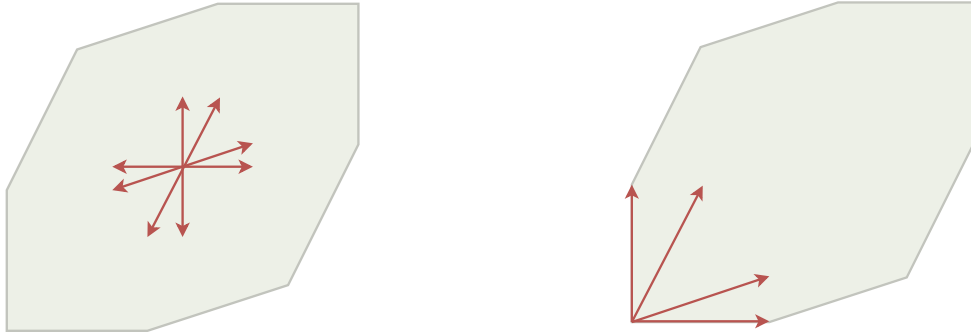


Figure 2.1: Both constructions lead to the same 2D zonotope generated by 4 generators: the only difference lies in a rescale of the generators and the domain of α_i .

A zonotope is a specific type of convex polytope, so it can be described via its vertices (\mathcal{V} -representation) or a set of inequalities (\mathcal{H} -representation), as shown in figure 2.2. Several zonotope applications can be found as bounding volumes in collision detection [28], as bounding disturbances and measurement errors [49], but also in approximating the domain of a function of several variables [53]. More recently in robotics, a neural network was used to predict a path of reachable sets in an environment crowded with dynamic obstacles modeled as zonotopes [50].

The process of retrieving zonotope vertices is a combinatorial problem described as

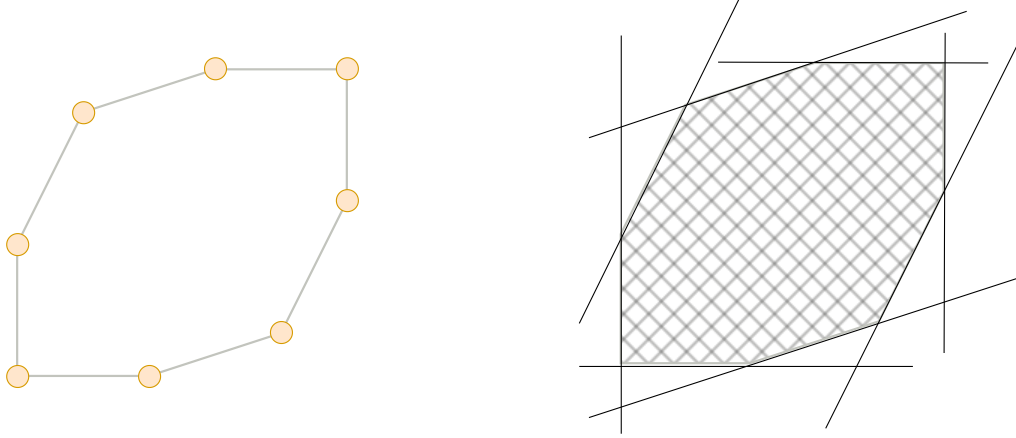


Figure 2.2: Any zonotope is uniquely described by its vertices. Equivalently, bounding hyperplanes can represent it.

the *vertex enumeration problem*. The explicit enumeration of zonotope vertices is required for instance in the fixed rank integer quadratic program [17], in signal processing [38] and in linear regression models with interval data [7].

Unfortunately, converting zonotope generators to vertices or bounding hyperplanes is a combinatorial problem. As the dimension or the number of generators increases, the number of vertices does too: theorem 3.1 from Ferrez et al. [17] recalls that the number of vertices of a n -zonotope \mathcal{Z} with m generators is $\leq 2 \sum_{i=0}^{n-1} \binom{m-1}{i}$, with equality satisfied when the generators are in general position. For instance, a 3D zonotope can have up to 92 vertices if it is defined by 10 generators; 2452 if it has 50 generators; and 9902 vertices for 100 generators. Similarly, for a n -zonotope with 50 generators, it has up to 39300 vertices for $n = 4$, 562949953421312 for $n = 25$ and 1125899906842624 for $n = 50$. In a biomechanical context, by considering the Stanford’s upper limb musculoskeletal model consisting of 50 muscles and 7 degrees-of-freedom [30], the torque feasible set has at most 32244452 vertices. Consequently, the required space to describe all the vertices becomes large very quickly, independently of the efficiency of an algorithm enumerating them.

The following section 2.1 presents a novel efficient zonotope edge enumeration algorithm, called *EdgeEnum* which can be used to enumerate vertices of a zonotope. Section 2.2 shows that *EdgeEnum* is indeed theoretically *efficient* by proving the polynomiality of both its time and space complexity when n is fixed. An asymptotic growth comparison is also performed over various recent enumeration algorithms adapted to the zonotope vertex enumeration problem. Section 2.3 compares the time benchmark of multiple algorithms to show that in practice *EdgeEnum* is faster for zonotopes described by $m \geq 10$ generators in dimensions $n \geq 6$. Section 2.4 offers some insights on time computation of approximated vertex enumeration algorithms, in order to show that in our context, the current computation times are not fast enough, whether using approximated or exact algorithms. Finally, section 2.5 summarizes the results and advantages offered by our approach, such as the easyness of implementation and the possible parallelism in comparison to existing methods, while bringing some light upon the effectiveness of using vertices to describe zonotope characteristics of interest such as its global shape and orientation.

2.1 The edge enumeration algorithm

While enumerating zonotope vertices is the main goal, we shall present an algorithm which enumerate its edges, then describe an edges-to-vertices conversion algorithm which is negligible in computation time. The edge enumeration algorithm takes as input the generator matrix of a n -zonotope \mathcal{Z} with m generators in general position and returns a set of m -cube edges which map to the edges of \mathcal{Z} .

Vertices *do not* contain information about other vertices, whereas edges are always describing at least 2 vertices by providing a link between them. Our novel approach use this information in order to make vertex enumeration faster, by enumerating zonotope edges. Since a zonotope is the projection of a higher-dimensional cube, its edges are projection of some cube edges, so constructing edges of the cube in higher dimension is required.

From cube edges to zonotope edges

The construction of the edges of a m -dimensional hypercube, also called a m -cube, is an iterative process over dimensions 2 to m (cf. figure 2.3). Starting from a square (or the 2-cube), each of its edges are embedded in 3D then duplicated and translated by 1 along the newly created dimension. Once done, there remains to create new edges arising from the duplication: for each vertex v in the previous step, a segment is created between v and its newly duplicated and translated version v' . The process is then iterated until the m th dimension is reached: all of the m -cube edges are enumerated.

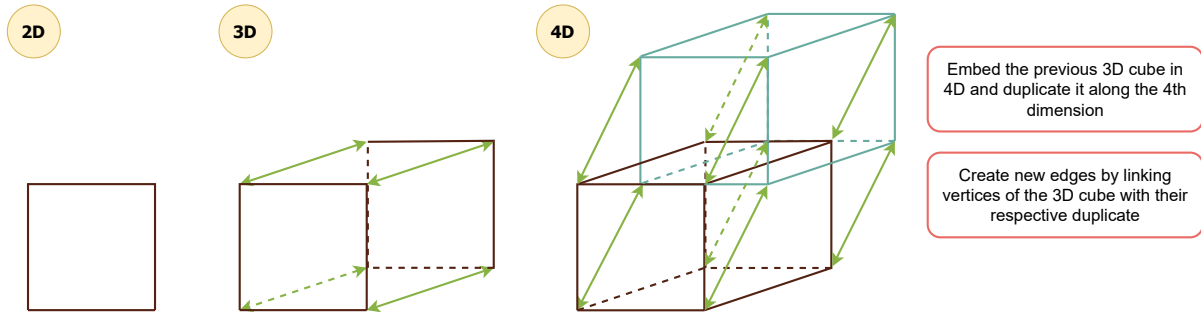


Figure 2.3: Constructing the edges of a zonotope with 4 generators requires constructing the edges of a 4-dimensional hypercube. The process is iterative and starts from the square to retrieve the cube, then the 4-cube. The creation of new edges arising at each new step correspond to the green double arrows.

Once all m -cube edges are enumerated, projecting them via the generator matrix of the considered zonotope gives us something similar to the figure 2.4: a bunch of segments which seems to encapsulate the zonotope.

This phenomenon is due to a parallelism property specific to zonotopes. We recall that two segments or lines are defined to be *parallel* if their direction vectors are colinear.

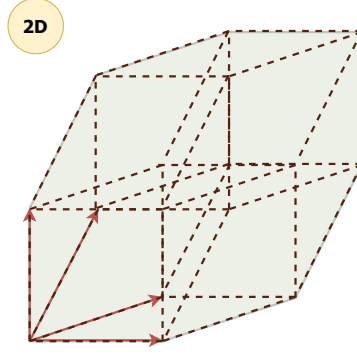


Figure 2.4: Projecting the 4-cube edges via the generator matrix of a 2D zonotope with 4 generators.

Lemma 2.1.1: Parallelism of zonotope edges with the generators

For any n -dimensional zonotope $\mathcal{Z} \subset \mathbb{R}^n$ with m generators, then each of its edge is parallel to one of its generator.

Moreover, each projected edge of the m -cube is also parallel to one of the generators.

Proof. A linear or affine map preserves parallelism. Consider a zonotope \mathcal{Z} described by generators $\mathbf{g}_1, \dots, \mathbf{g}_m \in \mathbb{R}^n$ in general position. Denote by G the generator matrix. If $n \geq 2$, then edges of the m -cube surject onto the zonotope edges clearly. Since edges of the m -cube can be grouped by parallelism by a representing vector $\mathbf{r}_i = \mathbf{e}_i \in \mathbb{R}^m$ where \mathbf{e}_i corresponds to the i^{th} vector of the canonical basis of \mathbb{R}^m , we have that $G\mathbf{r}_i = \mathbf{g}_i$. This ensures that when an edge cube is projected via G , it necessarily will be parallel to one of the generator. \square



Figure 2.5: To the left: each edge of a zonotope \mathcal{Z} is parallel to one of its generator. To the right: projecting a 4-cube edge via the generator matrix of \mathcal{Z} produces a line segment parallel to one of the generator.

Unfortunately, for a zonotope $\mathcal{Z}(\mathbf{c}, G)$, we could not enumerate each edge \mathbf{e} of an m -cube and project them via $\mathbf{e} \mapsto \mathbf{c} + G\mathbf{e}$ to obtain the zonotope edges, due to the high combinatorics involved in the number of possible edges. This amounts to $m2^{m-1}$ edges for a m -cube, which is greater than 2^m , the amount of vertices and would make our approach worse than the naive algorithm.

Our novel approach uses a straight-forward elimination technique to remove useless cube edges as described in the following paragraphs.

The convex hull of parallel lines

As seen in lemma 2.1.1, in a zonotope each edge is parallel to one of the generator. More than that: each edge of the m -cube will map to a segment parallel to a generator. This means that it is possible to *group* the projected cube edges by parallelness; the group representant being one of the generator. For each group, the extremal segments define the edges of the zonotope parallel to the considered generator. Suppose we group these projected edges by parallelism. The next question is: *how do I select only the ones on the zonotope surface?* While it is a very difficult question to answer for a set consisting of various segments, we are working with projected edges which are all parallel to a generator so that this difficulty is leveraged.

We shall call the process of retrieving extremal edges as the *convex hull of parallel lines*. Usually and in practice, convex hulls are implemented for sets of points. In this specific case of parallelism, we shall establish an algorithm which selects extremal edges by modifying the problem as a classic convex hull of a set of points.

Let's introduce it by a 2D example which will generalize well for any dimension: proceeding step by step, consider the first iteration of the algorithm. Edges of a square are created, then they are embedded in 3D and projected via the submatrix of G consisting of the two first columns. This gives three group of edges as shown by the edge colours in figure 2.6.

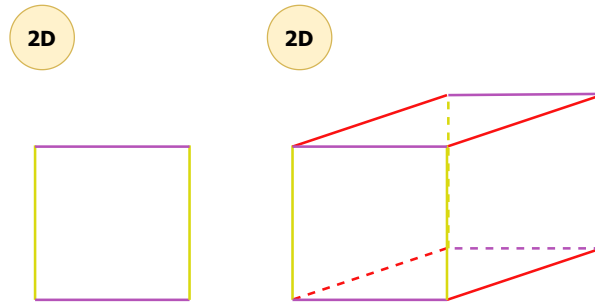


Figure 2.6: The edges on the right are created from embedding and duplicating the square edges to create a cube, then are projected onto the plane via the two first columns of G .

Now, consider a group of parallel edges, the purple ones. We shall consider them as lines. Then, an arbitrary point in the zonotope space is selected and is projected orthogonally onto each line. At any step of the algorithm, these projected points span at most a $n - 1$ -dimensional space if the zonotope is of dimension n . The convex hull operation is thus applied on this set of projected points and only the edges associated to points on the convex hull are kept for the next iteration (cf. figure 2.7).

We shall repeat this process for all group of edges, which amounts to the number of dimensions the underlying cube is defined as, as shown in figure 2.8.

Once all groups have been treated, we are left with only the necessary edges required to produce the edges of the zonotope generated by the three first columns of G . We shall proceed the embedding, duplication and translation process to create the new edges in $4D$, and project them in this time via the submatrix of G consisting of the four first columns. The algorithm stops when after multiple iterations, the selected cube edges are from the m -cube (cf. figure 2.9).

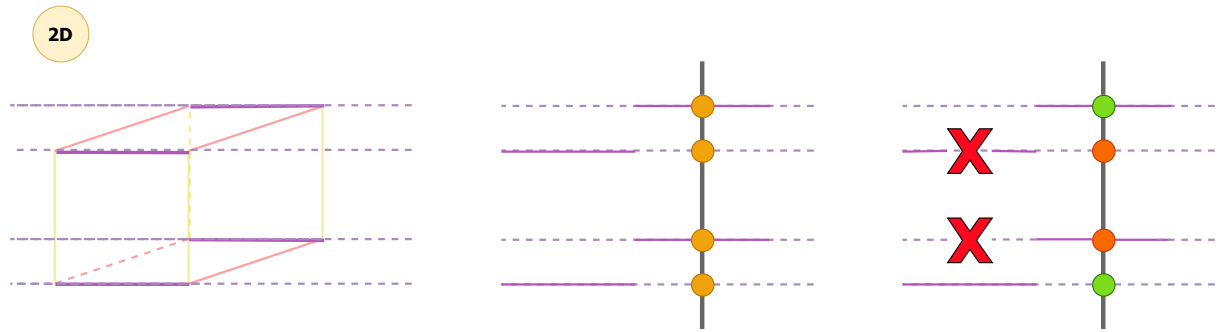


Figure 2.7: From left to right: select a group of parallel edges and consider them as lines. Then project orthogonally an arbitrary point onto these lines. Finally, apply the convex hull operation onto these points to retrieve the extremal associated edges. This works in any dimension, as long as a set of parallel lines is considered.

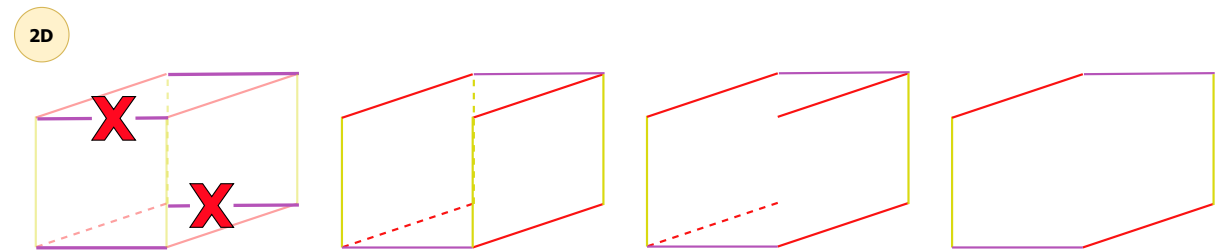


Figure 2.8: Selection of the extremal edges for each group of parallel edges by applying the convex hull of parallel lines. First purple edges, then yellow ones and finally red ones.

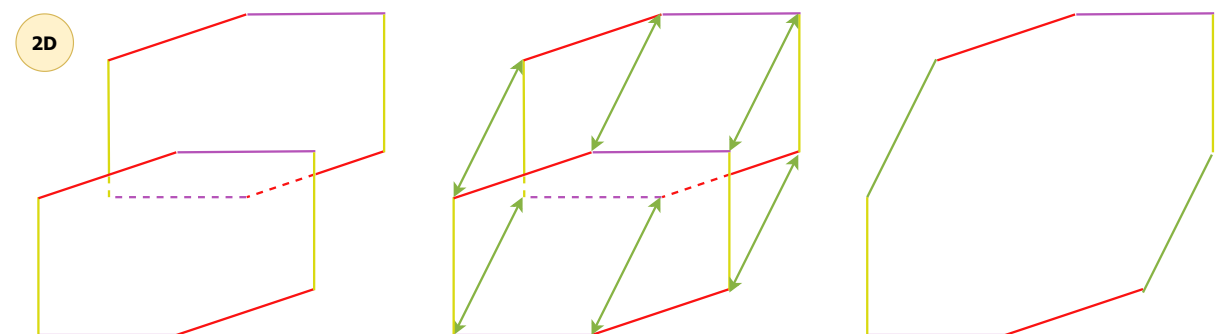


Figure 2.9: After exactly $m - 2$ iterations, the returned edges are exactly the zonotope edges.

The EdgeEnum algorithm

In the next algorithms, an edge of the m -cube can be represented by two m -dimensional vectors: a direction vector and an initial position point. Algorithm 1 computes how to create a new set of edges in one higher dimension $d + 1$ from a set of given edges in \mathbb{R}^d , using the construction of edges of a higher dimensional cube. Algorithm 2, called EdgeEnum, constructs zonotope edges following the reasoning previously described.

Algorithm 1: Duplication, translation and link of edges

Data: E_d a set of edges in \mathbb{R}^d
Result: E_{d+1} a set of edges in \mathbb{R}^{d+1}
Function duplicateAndLinkEdges(E_d):
 $E_{d+1} \leftarrow \emptyset$
 foreach $e \in E_d$ **do**
 $e_0 \leftarrow \text{embedInOneHigherDimension}(e)$
 $e_1 \leftarrow \text{translateAlongNewDimensionBy1}(e_0)$
 $v_0^1, v_0^2 \leftarrow \text{verticesAtExtremities}(e_0)$
 $v_1^1, v_1^2 \leftarrow \text{verticesAtExtremities}(e_1)$
 $e_{01}^1 \leftarrow \text{createEdge}(v_0^1, v_1^1)$
 $e_{01}^2 \leftarrow \text{createEdge}(v_0^2, v_1^2)$
 $E_{d+1} \leftarrow E_{d+1} \cup \{e_0, e_1, e_{01}^1, e_{01}^2\};$
 end
 return E_{d+1}
End Function

Algorithm 2: EdgeEnum: Zonotope edge enumeration algorithm

Data: A matrix $N \in \mathbb{R}^{n \times m}$, with $n \geq 2$ and $m \geq 2$
Result: Edges of the zonotope $\mathcal{Z}(\mathbf{c}, N)$
 $d \leftarrow 2$
 $N_d \leftarrow N[:, : d]$
 $\text{Edges}(\mathcal{C}_d) \leftarrow \{\text{edges of the 2D cube}\} // \mathcal{C}_d \text{ is the cube } [0, 1]^d.$
 $\text{Edges}(\mathcal{Z}(\mathbf{c}, N_d)) \leftarrow \{\mathbf{c} + N_d \mathbf{e} \mid \mathbf{e} \in \text{Edges}(\mathcal{C}_d)\}$
 $d \leftarrow 3$
while $d \leq m$ **do**
 $N_d \leftarrow N[:, : d]$
 $\text{Edges}(\mathcal{C}_{d-1}) \leftarrow \text{Edges}(\mathcal{C}_d)$
 $\text{Edges}(\mathcal{C}_d) \leftarrow \text{duplicateAndLinkEdges}(\text{Edges}(\mathcal{C}_{d-1})) // \text{cf. algorithm 1}$
 $\text{Edges}(\mathcal{Z}(\mathbf{c}, N_d)) \leftarrow \{\mathbf{c} + N_d \mathbf{e} \mid \mathbf{e} \in \text{Edges}(\mathcal{C}_d)\}$
 foreach *group of parallel edges* $G \in \text{Edges}(\mathcal{Z}(\mathbf{c}, N_d))$ **do**
 foreach $\mathbf{c} + N_d \mathbf{e} \notin \text{ext}(\text{conv}(G))$ **do**
 $\text{Edges}(\mathcal{C}_d) \leftarrow \text{Edges}(\mathcal{C}_d) \cup \{\mathbf{e}\}$
 end
 end
 $d \leftarrow d + 1$
end
 $\text{Edges}(\mathcal{Z}(\mathbf{c}, N_d)) \leftarrow \{\mathbf{c} + N_d \mathbf{e} \mid \mathbf{e} \in \text{Edges}(\mathcal{C}_d)\}$
return $\text{Edges}(\mathcal{C}_d), \text{Edges}(\mathcal{Z}(\mathbf{c}, N_d))$

2.2 Theoretical complexity

In this section, various theoretical results about our new algorithm are shown, using complexity as the main tool of comparison. While a thorough time and space complexity analysis is given, the asymptotic time growth as the number of generators increases is our main interest since it is related to a larger number of muscles considered to compute the torque feasible set. This means that n , the dimension of the ambient space the zonotope lives in, is considered to be of fixed value while the number of generators m is assumed to be large in general.

Before diving into the analysis, we recall some basics over algorithmic terms and the asymptotic growth usage:

Asymptotic growth and Big O notation. An *asymptotic growth* describes how a function behaves as its input becomes very large and we denote by a *Big O* notation the dominant terms. For instance, the function $f: \mathbb{N} \rightarrow \mathbb{N}$ defined by $f(n) = 4n^5 + n^5 \log(n)$ is dominated by the term $n^5 \log(n)$, meaning that when n tends to $+\infty$, $f(n)$ will approximately have the same growth rate as the function $n \mapsto n^5 \log(n)$. We use the Big O notation to precise this growth rate comparison by stating f is in $O(n^5 \log(n))$ or $f(n) = O(n^5 \log(n))$. To compare the asymptotic growth of two functions, their respective growth are calculated and the ratio of these rates is evaluated to decide which of the two has a slower growth than the other. If the ratio tends to a constant value, then both algorithms have asymptotically the same growth. More generally, we consider O to represent a class of functions which are bounded by a given formula: in our example, f is in $O(n^5 \log(n))$ but it is also in higher classes such as $O(n^6)$ or $O(n^{20})$.

Algorithmic complexity. For algorithms, which are essentially functions with inputs and outputs, two types of complexity are commonly described: the *time* and the *space* complexity. The time complexity evaluates theoretically the required time of each operation, while the space complexity evaluates the required storage.

Efficient algorithm. An algorithm is *polynomial* or *efficient* if its time complexity is upper bounded by a polynomial expression of both the input size and the output size [19]. In other words, the running time of the algorithm doesn't grow exponentially as the input size increases, making it generally efficient for solving problems.

Compact algorithm. An algorithm is *compact* if its space complexity is polynomial in the input size: this means that a compact algorithm does not store *all* of the output and can stream the results directly without keeping in storage the output. It is a very useful property, especially when the size of the output is huge. In the literature, two main approaches can be found: reverse-search-based algorithms and iterative algorithms. Algorithms are theoretically more valuable if they are compact, especially in high dimensions, so this approach is preferred and commonly found [1], [27] and [44], but it should not exclude non-compact algorithms depending on the application context.

Optimal algorithm. An algorithm is *optimal* (in time and/or in space) if its complexity has been proven to be the lowest ever reachable. For instance, it has been recalled in theorem 3.2 of Ferrez et al. [17] that there exists a time-optimal algorithm which

enumerate all zonotope vertices in time $O(m^{n-1})$, which has been implemented in 1983 by Edelsbrunner et al. [15]. Unfortunately, such an optimal algorithm may not be practical [17]: firstly, it uses an incremental strategy which requires to store all extreme points of a subproblem based on a lesser number of generators. Secondly, it is complicated to implement and needs to store all faces and their incidence (*i.e.* if the intersection of two faces is another face or not). In other words, Edelsbrunner et al.'s algorithm is time optimal and efficient, but space deficient.

In our context, we mainly care about time efficiency. As the next theorems show, our new algorithm is time efficient even when considering the edges-to-vertices transition. It is also space efficient but it is not compact. Before diving into complexity calculations, let's describe a very useful result concerning the number of specific faces in a zonotope. A zonotope \mathcal{Z} in dimension n has $n - 1$ type of faces: the 0-faces are called *vertices*, the 1-faces *edges*, the $(n - 2)$ -faces *ridges*, and the $(n - 1)$ -faces *facets*. In specific cases such as in dimension 2, the edges correspond to the facets. In 3D, edges correspond to ridges.

Lemma 2.2.1: Upper bound on the number of k -faces of a zonotope

For $G \in \mathbb{R}^{n \times m}$, the number of k -faces of the zonotope $\mathcal{Z}(G)$ for $k = 1, \dots, n - 1$ is upper bounded by $f_k(\mathcal{Z})$ such that

$$f_k(\mathcal{Z}) = 2 \binom{m}{k} \sum_{i=0}^{n-k-1} \binom{m-k-1}{i}$$

The bound is attained if \mathcal{Z} is in general position.

Proof. This result can be found with proof in [19], [13] and [25]. □

2.2.1 Time complexity

Theorem 2.2.2: Time complexity of EdgeEnum

For a n -zonotope \mathcal{Z} with m generators in general position, the edge enumeration algorithm is in time complexity

$$O(nm^2 f_1(\mathcal{Z}))$$

where $f_1(\mathcal{Z})$ denotes the number of edges of \mathcal{Z} .

Proof. Each iteration of the algorithm (*i.e.* $d = 3, \dots, m$) consider a new zonotope \mathcal{Z}_d generated by the first d columns of the generator matrix $G \in \mathbb{R}^{n \times m}$ and construct its edges based on the set of edges of \mathcal{Z}_{d-1} . We are going to process every time computation in the algorithm:

1. **Embed, duplicate and translate edges:** In the first inner-loop, a new dimension is added to the edges of \mathcal{Z}_{d-1} , which can be done by concatenating a 0 to both the direction and position vector describing each of them. These embedded edges are then duplicated and the copied versions have their direction and position last coordinates mutated to 1. New edges are created by linking the respective extremity points from the

embedded edges to their **translate**. This whole step can be done in time complexity O for each substep, in a loop over each of the previous edges, so it is $O(f_1(\mathcal{Z}_{d-1}))$.

2. Grouping edges by parallelism: The newly created set of edges consists of exactly $4f_1(\mathcal{Z}_{d-1})$ edges. Since colinearity between cube edges is preserved under linear transformation, grouping edges of zonotope \mathcal{Z}_d by parallelism only requires to group the cube edges by directions. In Python, using a dictionary structure, the grouping process is $O(f_1(\mathcal{Z}_{d-1}))$.

3. Applying the convex hull over each group of edges: Each group of edge consists of $2f_1(\mathcal{Z}_{d-1})/(d-1)$ edges, and there are d groups. For each group, each edge will be projected onto \mathbb{R}^n via the generator matrix of \mathcal{Z}_d . This corresponds to one matrix-vector operation of time complexity $O(nd)$ per edge. The projection done, the orthogonal projection of the origin in \mathbb{R}^n onto the line spanned by the projected edge is computed in time complexity $O(n)$ (the dimension of the ambient space where the line lies). The convex hull of the projected edges is computed as the convex hull of the orthogonal projections of the origin onto the lines spanned by the edges. Let's consider the time complexity of Chan's convex hull algorithm in n dimensions. It is an output-sensitive algorithm, which means that its complexity depends on the size of the output. Its time complexity is $O(n_v \log h)$ with n_v the number of given points and h the known number of points on the convex hull. Since the zonotope \mathcal{Z}_d is in general position, the number of edges in each group is $f_1(\mathcal{Z}_d)/d$. The total time complexity of step 3 is then of order $d(\frac{2f_1(\mathcal{Z}_{d-1})}{d-1}nd + \frac{2f_1(\mathcal{Z}_{d-1})}{d-1}n + \frac{2f_1(\mathcal{Z}_{d-1})}{d-1} \log(\frac{f_1(\mathcal{Z}_d)}{d}))$, which is $O(f_1(\mathcal{Z}_{d-1})nd + f_1(\mathcal{Z}_{d-1}) \log(f_1(\mathcal{Z}_d)/d))$.

After d iterations, the complexity is of order $\sum_{d=3}^m f_1(\mathcal{Z}_{d-1})nd + f_1(\mathcal{Z}_{d-1}) \log(f_1(\mathcal{Z}_d)/d)$ and since $d \leq m$ and $f_1(\mathcal{Z}_d) \leq f_1(\mathcal{Z})$, we have the following upper bound:

$$\begin{aligned} \sum_{d=3}^m f_1(\mathcal{Z}_{d-1})nd + f_1(\mathcal{Z}_{d-1}) \log(f_1(\mathcal{Z}_d)/d) &\leq f_1(\mathcal{Z}) \left[n \sum_{d=3}^m d + \sum_{d=3}^m \log \left(\frac{f_1(\mathcal{Z})}{d} \right) \right] \\ &\leq f_1(\mathcal{Z}) \left[nm \frac{m+1}{2} + \log \left(\prod_{d=1}^m \frac{f_1(\mathcal{Z})}{d} \right) \right] \\ &\leq f_1(\mathcal{Z}) \left[n \frac{m^2 + m}{2} + \log \left(\frac{f_1(\mathcal{Z})^m}{m!} \right) \right] \end{aligned}$$

So the edge-based algorithm is upper bounded in $O \left(nm^2 f_1(\mathcal{Z}) + f_1(\mathcal{Z}) \log \left(\frac{f_1(\mathcal{Z})^m}{m!} \right) \right)$.

The second part of the proof is to show that $nm^2 f_1(\mathcal{Z})$ is the dominant term in this upper bound. Let's show that for any growth of n and m , $f_1(\mathcal{Z}) \log \left(\frac{f_1(\mathcal{Z})^m}{m!} \right) / (nm^2 f_1(\mathcal{Z}))$

is upper-bounded by a function whose growth tends to be constant (meaning it is $O(1)$).

$$\begin{aligned}
\frac{f_1(\mathcal{Z}) \log \left(\frac{f_1(\mathcal{Z})^m}{m!} \right)}{nm^2 f_1(\mathcal{Z})} &\leq \frac{1}{nm^2} \log (f_1(\mathcal{Z})^m) \\
&= \frac{1}{nm} \log (f_1(\mathcal{Z})) \\
&= \frac{1}{nm} \log \left(2 \binom{m}{1} \sum_{i=0}^{n-2} \binom{m-2}{i} \right), \quad \text{using theorem 2.2.1} \\
&\leq \frac{1}{nm} \log (m 2^{m-1}), \quad \text{using } \sum_{i=0}^{n-2} \binom{m-2}{i} \leq \sum_{i=0}^{m-2} \binom{m-2}{i} = 2^{m-2} \\
&= \frac{\log(m)}{nm} + \frac{(m-1) \log(2)}{nm}
\end{aligned}$$

There are now three cases to study:

1. $n \rightarrow +\infty$ and m grows at a smaller rate than n ($\lim_{m,n \rightarrow +\infty} \frac{m}{n} = 0$):

$$\lim_{m,n \rightarrow +\infty} \frac{\log(m)}{nm} = \lim_{m,n \rightarrow +\infty} \frac{1}{nm} = 0 \quad \text{and} \quad \lim_{m,n \rightarrow +\infty} \frac{(m-1) \log(2)}{nm} = \lim_{n \rightarrow +\infty} \frac{1}{n} = 0$$


2. $m \rightarrow +\infty$ and n grows at a smaller rate than m ($\lim_{m,n \rightarrow +\infty} \frac{n}{m} = 0$):

$$\lim_{m,n \rightarrow +\infty} \frac{\log(m)}{nm} = \lim_{m,n \rightarrow +\infty} \frac{1}{nm} = 0 \quad \text{and} \quad \lim_{m,n \rightarrow +\infty} \frac{(m-1) \log(2)}{nm} = \lim_{m,n \rightarrow +\infty} \frac{1}{n} = 0$$

3. $m \rightarrow +\infty$ and $n \rightarrow +\infty$ at an equivalent rate ($\lim_{m,n \rightarrow +\infty} \frac{n}{m} = C$, for C a positive constant):

$$\lim_{m,n \rightarrow +\infty} \frac{\log(m)}{nm} = \lim_{m,n \rightarrow +\infty} \frac{1}{nm} = 0 \quad \text{and} \quad \lim_{m,n \rightarrow +\infty} \frac{(m-1) \log(2)}{nm} = \lim_{m,n \rightarrow +\infty} \frac{1}{n} = 0$$

All cases confirm that $f_1(\mathcal{Z}) \log \left(\frac{f_1(\mathcal{Z})^m}{m!} \right)$ is dominated in growth by $nm^2 f_1(\mathcal{Z})$ for any cases of growth for m and n . This means that the complexity of our edge enumeration algorithm is $O(nm^2 f_1(\mathcal{Z}))$. \square

Now, let  show an intermediate result which will be used in the proof of the next theorem 2.2.4.

Lemma 2.2.3: Upper asymptotic growth for the number of k -faces

For \mathcal{Z} a n -zonotope with m generators in general position, then $f_k(\mathcal{Z})$ the number of k -faces of \mathcal{Z} when k and n are fixed and m is large has an asymptotic growth upper-bounded in $O(m^{n-1})$.

While it was already well-known that the number of vertices $f_0(\mathcal{Z})$ is asymptotically bounded in $O(m^{n-1})$ (proven by Zaslavsky in [57]), our result extends this finding to higher dimensions in order to find an asymptotic upper-bound for the number of k -faces,

as long as k and n are considered constant. This theorem stipulates that the number of vertices, edges, ridges, hyperplanes and all type of k -faces of a general zonotope are all upper bounded by the same type of growth as m increases. We use this theorem **in a few pages** to compute the asymptotic growth of our edge enumeration algorithm in the case of n fixed, which will help us to compare our algorithm against the state-of-the-art.

Proof. We shall proceed by finding an upper bound of $f_k(\mathcal{Z})$ which is $O(m^{n-1})$ for n fixed. First, notice that for $0 \leq n \leq m$, m^n is an upper bound of $\binom{m}{n}$:

$$\binom{m}{n} = \frac{m!}{n!(m-n)!} = \frac{1}{n!} \cdot m(m-1)(m-2) \dots (m-n+1) \leq m^n \quad (2.1)$$

Also, when considering m large and k, n fixed, it is reasonable to assume $(n-k-1) \leq (m-k-1)/2$ so that

$$\binom{m-k-1}{i} \leq \binom{m-k-1}{n-k-1}, \quad \text{for any } i = 1, \dots, n-k-1 \quad (2.2)$$

Hence the following:

$$\begin{aligned} f_k(\mathcal{Z}) &= 2 \binom{m}{k} \sum_{i=0}^{n-k-1} \binom{m-k-1}{i} \\ &\leq 2 \binom{m}{k} \sum_{i=0}^{n-k-1} \binom{m-k-1}{n-k-1}, \quad \text{using inequality 2.2} \\ &\leq 2m^k (m-k-1)^{n-k-1} (n-k), \quad \text{using inequality 2.1} \end{aligned}$$

The last inequality is equivalent to state that $f_k(\mathcal{Z})$ is $O(m^{n-1})$, for k, n fixed and m large. \square

Theorem 2.2.4: Time complexity of EdgeEnum when n is fixed

If n is fixed and m is large, the time complexity of the edge enumeration is $O(m^{n+1})$. Also, this means that EdgeEnum is an algorithm of polynomial time when n fixed.

Proof. This directly is deduced from theorem 2.2.3 stating that the number of faces $f_1(\mathcal{Z})$ is $O(m^{n-1})$ when n is fixed. Using the time complexity of EdgeEnum in theorem 2.2.2, we have for n fixed: $O(nm^2 f_1(\mathcal{Z})) = O(m^2 m^{n-1}) = O(m^{n+1})$. \square

From edges to vertices

Theorem 2.2.5: Time complexity to convert edges to vertices

For a n -zonotope \mathcal{Z} with m generators in general position, the time complexity of an algorithm enumerating vertices from its edges is in $O(nm f_1(\mathcal{Z}))$.

Proof. Every edge of \mathcal{Z} is linked to two vertices, which corresponds to its extremities. The transformation of all the edges to vertices can be realized in time complexity $O(f_1(\mathcal{Z})nm)$: indeed, the two extremities of each cube edge must be projected via the generator matrix so this accounts for $2f_1(\mathcal{Z})nm$ $O(1)$ steps. An iteration of all these vertices can be done as well to ensure the uniqueness of each vertex (2 edges can lead to a common vertex!). This step of verification can be done in $O(f_1(\mathcal{Z}))$ in Python, using a hash structure. \square

Since $nm^2f_1(\mathcal{Z})$ dominates $nmf_1(\mathcal{Z})$, we can safely say that the conversion step from edges to vertices does not have a theoretical time impact in a worst-case scenario if this step is added after the edge-enumeration algorithm.

2.2.2 Space complexity

Theorem 2.2.6: Space complexity of EdgeEnum

For a n -zonotope \mathcal{Z} with m generators in general position, the edge enumeration algorithm is in space complexity

$$O(n^2m^2f_1(\mathcal{Z}))$$

where $f_1(\mathcal{Z})$ denotes the number of edges of \mathcal{Z} .

Proof. The algorithm starts with a matrix in $\mathbb{R}^{n \times m}$ so requires an input space in $O(nm)$. In the first steps; the definition of the 2-cube edges and their projection is in $O(n)$ clearly.

1. Embed, duplicate, translate edges: A cube edge can be described in two parts: a direction and a position in \mathbb{R}^m . A direction of a cube edge is in $O(1)$, since it is necessarily assimilated to a vector $e = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^m$ (due to parallelism of edges to the canonical basis of \mathbb{R}^m). The position, however, can be more general so let's take a worst case scenario of $O(m)$.

Each edges at the begining of the loop is copied, then embedded, duplicated and new edges are created. There are maximum $f_1(\mathcal{Z}_{d-1})$ edges at the begining so these steps require a space in $O((d-1) \cdot f_1(\mathcal{Z}_{d-1}) + 3 \cdot d \cdot f_1(\mathcal{Z}_{d-1})) = O(d \cdot f_1(\mathcal{Z}_{d-1}))$.

2. Grouping edges by parallelism: First, all newly created edges need to be projected onto the space generated by N_d , so it gives a new set of projected edges in $O(ndf_1(\mathcal{Z}_{d-1}))$. Using a dictionary in Python to pair edges with a projected direction (in this case corresponding to a generator of N_d), this requires $O(d)$ space with d corresponding to the number of generators in the current iteration. The total space in this step is thus in $O(ndf_1(\mathcal{Z}_{d-1}))$.

3. Applying the convex hull over each group of edges: For each of the d group of edges (consisting of $O(nf_1(\mathcal{Z}_{d-1}))$ edges), Chan's convex hull algorithm has a space time complexity of $O(n \cdot nf_1(\mathcal{Z}_{d-1})) = O(n^2f_1(\mathcal{Z}_{d-1}))$. This leads to a total space complexity of $O(dn^2f_1(\mathcal{Z}_{d-1}))$.

To conclude, since $d \leq m$ and $f_1(\mathcal{Z}_d) \leq f_1(\mathcal{Z}_m)$, we have for all the loops a space complexity in $O(nm + n + \sum_{d=3}^m df_1(\mathcal{Z}_{d-1}) + ndf_1(\mathcal{Z}_{d-1}) + dn^2f_1(\mathcal{Z}_{d-1}))$ and since the

terms in the sum are dominated by $dn^2 f_1(\mathcal{Z}_{d-1})$, itself dominated by $mn^2 f_1(\mathcal{Z}_d)$, we have a total space complexity of $O(n^2 m^2 f_1(\mathcal{Z}_d))$. \square

This leads directly to the following result:

Theorem 2.2.7: Space complexity to convert edges to vertices

For a n -zonotope \mathcal{Z} with m generators in general position, the space complexity of an algorithm enumerating vertices from its edges is in $O(n f_1(\mathcal{Z}))$.

Proof. Consider $f_1(\mathcal{Z})$ the maximum number of edges of a zonotope \mathcal{Z} . The input of the conversion algorithm is thus $O(n f_1(\mathcal{Z}))$ which is represented as a dictionary of a zonotope generator and a position vector in \mathbb{R}^n . Now, for each of these edges we shall create extremal vertices, which include the position vector and the addition of the position vector and the associated generator, so that the space complexity is still in $O(n f_1(\mathcal{Z}))$. \square

This ensures that the total space complexity from the edge enumeration to the vertex enumeration is $O(n^2 m^2 f_1(\mathcal{Z}))$.

Theorem 2.2.8: Space complexity of EdgeEnum when n is fixed

If n is fixed and m is large, the space complexity of the edge enumeration is $O(m^{n+1})$. Also, this means that EdgeEnum is an algorithm of polynomial space when n is fixed.

Proof. Since the space complexity is $O(n^2 m^2 f_1(\mathcal{Z}))$ and that $f_1(\mathcal{Z})$ is $O(m^{n-1})$ when n is fixed, we have that $O(n^2 m^2 f_1(\mathcal{Z}))$ is $O(m^{n+1})$, which is a polynomial bounding both the input size nm and the output size $O(m^{n-1})$. \square

2.2.3 Time-theoretic comparison with other algorithms

The proposed algorithm exhibits time complexity of $O(m^{n+1})$, while the optimal lower bound is $O(m^{n-1})$. Despite this discrepancy, the algorithm demonstrates several desirable properties, such as being easily implementable with standard scientific programming languages and packages. Nevertheless, there exist another significant algorithms focused on this problem that we shall described succinctly.

Avis and Fukuda's pivoting algorithm [1]. The *pivoting* algorithm, a type of *reverse-search* algorithms created by the same authors, was initially created to enumerate the vertices of a polytope described by a set of inequalities. Broadly, the algorithm starts with a vertex, then a neighboring vertex is found through a *pivot rule*, which decides which vertex to visit next based on the current vertex and the polytope structure. The new vertex is then marked as visited, and through iteration over each newly found vertex, all polytope vertices are enumerated. Numerous variant of this algorithm are possible by adapting the pivot rule. This algorithm also has been extended to compute vertices of a zonotope using only its generators, and is implemented in C++ with a Python wrapper in *libzonotope* by Ingvason [56]. Also, an effective parallelized variant of the pivoting algorithm as been done by Weibel in 2010 [54], and can be used through the C++ implementation *Minksum* [55].

The initial reverse-search algorithms were not described as a zonotope vertex enumeration. They were implemented to solve two kinds of problems: computing the convex hull of a set of points and computing the *cell enumeration of a central hyperplane arrangement problem*. The latest corresponds to an alternative description of a zonotope. There is a one-to-one correspondance between the combinatorial face structure of a zonotope and its central hyperplane arrangement. The idea is to consider the hyperplanes normal to each generator of a zonotope: the *cells* of this arrangement correspond to the regions created by the hyperplanes intersections. Describing and enumerating these cells (via a sign vector as shown in [17], [44]) is equivalent to finding the vertices of a zonotope. Figures 2.10 and 2.11 summarize this correspondance by giving some intuition on how to construct it from a zonotope and how to compute vertices from the regions of such an arrangement. This construction generalizes well for zonotope generators in any dimension. For a more profound understanding of this duality and further details on this correspondance, the reader is invited to read the *Duality* section of Ferrez et al. [17], or more generally in Grünbaum's dedicated chapter *Arrangement of hyperplanes* [25].

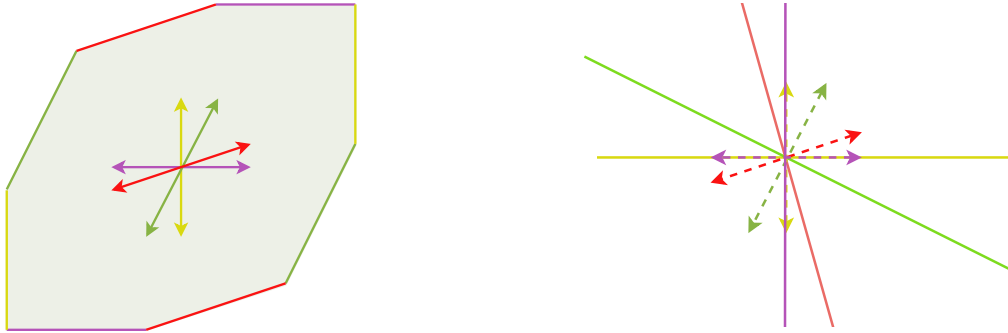


Figure 2.10: To the left: a zonotope with four generators. To the right: the normal hyperplanes to each generator. This corresponds to its *central hyperplane arrangement*.

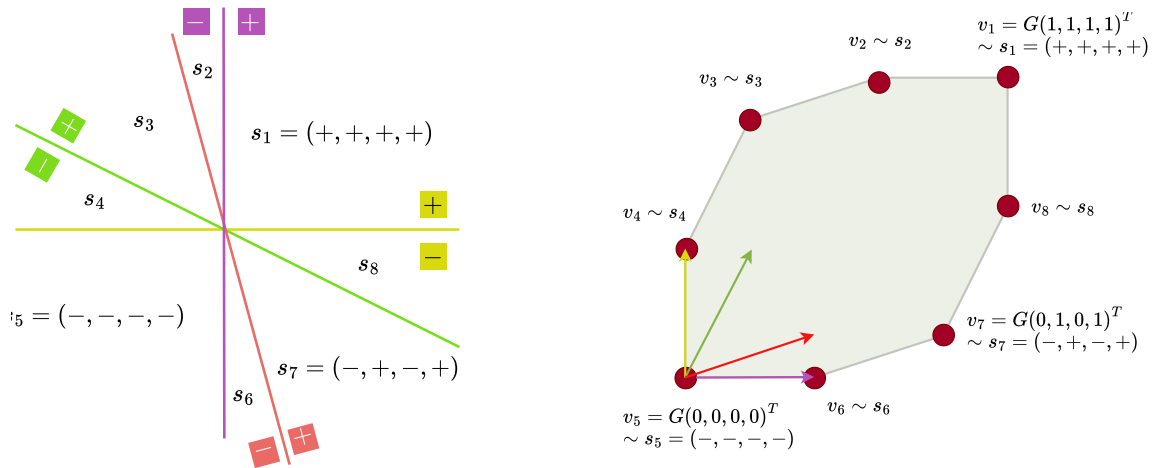


Figure 2.11: Let G be the matrix for the yellow, purple, red then green generators. In its central hyperplane arrangement, each region (or *cell*) is represented by a sign vector s_i . Indeed, any n -dimensional space can be separated in exactly 2 by any hyperplane. There are exactly as many cells as there are vertices in the underlying zonotope. Even more: they are in one-to-one correspondance, as shown by the symbol \sim .

Thus, due to the simplicity of working with sign vectors, there is a focus in computation geometry to create algorithms dedicated to enumerate cells of a central hyperplane

arrangements. The most recent ones include Rada and Cerny [44], Gu et Koenker [26] but also Gu et al.'s 2022 algorithm [27] whose practical time performance are significantly than the two previous one. We shall describe it next.

Gu et al.'s GRS algorithm [27]. This algorithm find cells of a central hyperplane arrangement iteratively over sub-dimensional arrangements. The idea is to consider a *witness* points in each cell, whose sign correspond the a cell description. Then move one hyperplane from the arrangement and project all other hyperplanes onto the space it generates. The previous witness points are then also projected onto it and they correspond to *witness point* of this sub-arrangement, up to duplication. The figure 2.12 summarizes the process in two dimensions, where instead of projecting onto subspaces, the witness points are constructed iteratively.

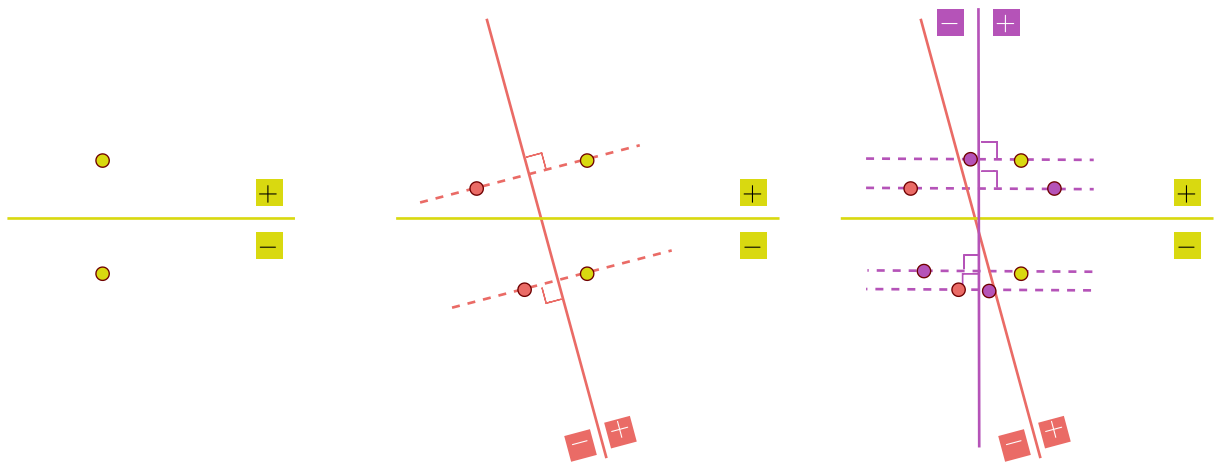


Figure 2.12: On the left: two points are created, on the respective sides of the hyperplane. At the middle: another hyperplane is added and the previously created points are copied on the other side of the new hyperplane. Gu et al found a distance computation to ensure that the new points are in the closest region. To the right: repeat with another added hyperplane and remove duplicates *i.e.* with the same sign vector.

The two previous cell enumeration methods are interesting, in regard to the correspondence, but there exist also other algorithms converting zonotope generators to a different representation, such as its bounding hyperplanes. However, since we concentrate on enumerating vertices, the conversion between bounding hyperplanes and vertices should be taken into account. Avis and Fukuda offer such a conversion algorithm in [1] in time complexity $O(n_h n n_v)$, where n is the dimension of n_h hyperplanes, and n_v is the number of vertices created by intersecting them. Such a method of bounding hyperplanes is the following:

Gouttefarde and Krut's hyperplane shifting method (HS) [23]. The number of bounding hyperplanes of a zonotope $\mathcal{Z}(n, m)$ in general position is $f_{n-1}(\mathcal{Z}) = 2\binom{m}{n-1}$, *i.e.* the $(n-1)$ -dimensional faces of the m -cube projects directly onto the zonotope bounding hyperplanes. Since the position of each $(n-1)$ -faces in the m -cube is known, there remain to project the position and the $(n-1)$ -space via the generator matrix, then compute the normal vectors. Using symmetry, the algorithm only requires to enumerate $\binom{m}{n-1}$ hyperplanes. Its time complexity is thus $O(\binom{m}{n-1})$, since all linear operations in the

algorithm (computing normals, projecting etc.) are necessarily bounded by $\binom{m}{n-1}$. We shall use the Python implementation available in Skuric et al's package Pycapacity [51].

We now present in table 2.1 the time complexity the three presented methods against EdgeEnum.

Algorithm	Pivot [1]	HS [23]	GRS [27]	EdgeEnum
Initial enumeration type	Cell	Hyperplane	Cell	Edge
Time complexity	$mf_0\text{lp}(m, n)$	$\binom{m}{n-1}$	$mf_0\text{mat}(m)$	nm^2f_1
Additional complexity to convert to vertices	nmf_0	$\binom{m}{n-1}nf_0$ described in [1]	nmf_0	nmf_1
Total complexity for n fixed	$m^{n+2.38}$	m^{2n-1}	m^{n+1}	m^{n+1}

Table 2.1: Comparison of time complexity between combinatorially equivalent enumeration algorithms when considering a n -dimensional zonotope with m generators in general position. f_0 corresponds to its maximal number of vertices and f_1 of edges. Pivot denotes Avis and Fukuda's pivoting algorithm [1], HS stands for Huetfarde and Krut's hyperplane shifting method [23] and GRS is Gu et al' [27]. $\text{lp}(m, n)$ denotes the time complexity to resolve a linear program of m equations of n variables, and $\text{mat}(m)$ denotes the time cost to determine whether two vectors of length m are identical. Cohen et al. [9] created an algorithm solving a linear program of m equations and n variables in a time complexity of the matrix multiplication, so about $m^{2.38}$. Also, using a programming language such as R or Python, we have $\text{mat}(m) = m$. An algorithm for converting a set of n_h hyperplanes to f_0 vertices has been created in time $O(n_hnf_0)$ in Avis and Fukuda [1]. The total complexity of HS is computed using Stirling's approximation of $\binom{m}{n}$ when m gets large as $\binom{m}{n-1} \approx m^{n-1}/(n-1)! = O(m^{n-1})$ if n is fixed.

It should be noted that the case of m fixed and n large is not of interest: if the number of generators m is smaller than the dimension of the associated zonotope, then the enumeration of the zonotope vertices necessarily requires to enumerate **all** of the m -cube vertices (which amounts to 2^m). So any algorithm can not be more optimal than a naive algorithm listing all the m -cube vertices.

However, a very strong theoretical limit on the case where m and n grows at the same rate. For a rough analysis, it must be first noticed that when n has approximately the same growth as m , $f_0(\mathcal{Z})$ is upper-bounded in $O(2^m)$ and $f_1(\mathcal{Z})$ is upper-bounded in $O(m2^{m-1})$ in the worst-cases (which is when $m = n$). Thus, in this case the best algorithm in time complexity is Gu et al., 2022 (which grows in $O(m^22^m)$), while EdgeEnum is far behind, even preceded by a naive algorithm (of time complexity $O(m2^m)$) with theoretical time complexity bounded in $O(m^42^{m-2})$.

An interpretation of these results is that EdgeEnum theoretically performs similarly to the best state-of-the-art enumeration algorithm (Gu et al., 2022 [27]) when there are much more generators than the ambient dimension of the produced zonotope, which is actually our scenario when considering a large number of muscles to produce the torque feasible set.



2.3 Time benchmarks

Theoretical time complexity analysis is not sufficient for a full time study: a time benchmark has to be realized over the different algorithms in order to evaluate the practical relevancy of the edge-based vertex enumeration algorithm. For instance, there exist cell enumeration algorithms such as Rada and Cerny [44] and Gu et Koenker [26] which have the same asymptotic bounds as Avis and Fukuda [1] for hyperplane arrangements in dimensions greater or equal to 3 [27] but seem to show faster computation time in practice.

2.3.1 Benchmark results

A computational time benchmark has been performed to evaluate the relevancy of a zonotope edge-based approach vertex enumeration for faster computation. A Dell XPS15 laptop computer has been used using WSL2 with a Ubuntu 22.01 operating system. This computer is equipped with 11th Gen Intel i9-11950H processors at 2.60GHz. Each core has 2 threads. The benchmark is implemented using Python 3.10 with the library *numpy* 1.26.4 and default packages such as *itertools* for generating rapidly cube vertices. *HS*, *GRS* and *EdgeEnum* algorithms are implemented in Python, whereas *Pivot* is implemented directly in C++ and the use of the package *libzonotope* allows for a Python interface. Our edge-based algorithm requires a convex hull computation, which uses *QuickHull*, available in Python through the library *scipy*.

Table 2.2 summarizes the means and standard deviations in seconds for each considered algorithm over 10 generator matrices $G \in \mathbb{R}^{n \times m}$ whose values are sampled from a uniform distribution between -1 and 1 .

(n, m)	Pivot [1]	HS [23]	GRS [27]	EdgeEnum
(2, 25)	0.02 ± 0.00	$< \mathbf{0.01}$	$< \mathbf{0.01}$	0.02 ± 0.00
(2, 50)	0.08 ± 0.00	$< \mathbf{0.01}$	$< \mathbf{0.01}$	0.09 ± 0.00
(3, 25)	0.45 ± 0.01	$\mathbf{0.03} \pm \mathbf{0.00}$	0.34 ± 0.01	0.14 ± 0.01
(3, 50)	4.25 ± 0.05	$\mathbf{0.14} \pm \mathbf{0.00}$	5.70 ± 0.11	1.44 ± 0.03
(4, 15)	0.66 ± 0.01	$\mathbf{0.08} \pm \mathbf{0.00}$	0.90 ± 0.02	0.14 ± 0.01
(4, 20)	2.41 ± 0.07	$\mathbf{0.21} \pm \mathbf{0.00}$	5.31 ± 0.09	0.48 ± 0.02
(5, 15)	3.60 ± 0.08	$\mathbf{0.80} \pm \mathbf{0.01}$	9.02 ± 0.11	0.97 ± 0.01
(5, 20)	17.99 ± 0.14	$\mathbf{4.25} \pm \mathbf{0.06}$	> 30	6.55 ± 0.29
(6, 10)	0.73 ± 0.02	0.58 ± 0.05	0.66 ± 0.01	$\mathbf{0.43} \pm \mathbf{0.01}$
(6, 11)	1.48 ± 0.05	1.39 ± 0.12	1.80 ± 0.03	$\mathbf{0.92} \pm \mathbf{0.01}$
(6, 12)	2.73 ± 0.07	2.86 ± 0.22	4.57 ± 0.14	$\mathbf{1.80} \pm \mathbf{0.05}$

Table 2.2: Means and standard deviations computation time (in seconds) for 10 randomly generated zonotopes for different zonotope enumeration algorithms. All generators are in general positions and all algorithms returned the expected number of vertices f_0 . The conversion time from a specific representation to vertices is taken into account for each algorithm.

It is noticeable that for any $n > 2$ and any m , EdgeEnum is faster than the standard Avis and Fukuda’s pivot algorithm. Surprisingly, Gouttefarde and Krut’s hyperplane shifting algorithm is the fastest in almost all cases for $n \leq 5$, even though theoretically it has the worst time complexity. However, as the dimension n increases, its complexity

is combinatorially explosive so this could explain why EdgeEnum is the fastest from dimension 6.

2.3.2 Parallelization of EdgeEnum

In order to improve the time performance of EdgeEnum, *parallelization* could be used. This refers to distributing some parts of an algorithm on multiple processors. Not all algorithms can be parallelized, and time computations can be *drastically* improved.



In EdgeEnum, there is one part which can be distributed: the inner loop in which the convex hull of each group of edges. Each new edges gathered after an iteration of this loop do not influence the gathering of other iterations, so that it is parallelizable. For instance, using the same computational setup as previously described, the next table 2.3 summarizes how EdgeEnum performs using either a single processor or two.

(n, m)	EdgeEnum	EdgeEnum Parallel
(6, 10)	0.43 ± 0.01	0.14 ± 0.01
(6, 11)	0.92 ± 0.01	0.28 ± 0.02
(6, 12)	1.80 ± 0.05	0.54 ± 0.03

Table 2.3: Means and standard deviations computation time (in seconds) for 10 randomly generated generator matrix $G \in \mathbb{R}^{n \times m}$ per tuple (n, m) . The parallelization of EdgeEnum is over two processors. The Python package *multiprocessing* has been used in this regard.

It must be noted that only from $n \geq 6$ the parallelism seemed to have a non-negligible effect. For lower n , parallelization results in worst time computation due to the overhead required to copy and transfer data between processors. We shall not compare the parallelized version with other algorithms as they are implemented in a non-parallelized manner.

2.3.3 Conclusion on practical time computation

To conclude, computing the vertices of a zonotope $\mathcal{Z}(n, m)$ requires long computation time, even when our  edge-based approach is in practice faster than the state-of-the-art algorithms for $n \geq 6$. This is due to the combinatorial explosion of the number of  a zonotope vertices when its number of generators increases. In our context of musculoskeletal model reconstruction, we shall at some point (*cf.* chapter 5) generate torque feasible sets for various muscle configurations in order to reproduce given zonotopes. As the number of muscles increases, the search space increases and a thorough sampling of it is required. For each muscle configuration, a zonotope must be computed. We will show that in our context, a time computation of zonotope vertices ≤ 0.2 seconds leads to hours of optimization when considering a reasonably sufficient number of muscles (≥ 20).

Time computations being for now a major obstacle, we shall concentrate on approximations of the vertex set of a zonotope and conclude if they are relevant in time computation but also in the shape they produce.

2.4 Approximation of the vertex set

While the previously described methods provide the *exact* vertices of a zonotope, there are cases in which the quantity of vertices is the main computational issue of any algorithms: the number of vertices are too many. This always happens when the number of generators is large, for any dimension n . Thus, it is desirable to leverage the computing time through approximations.

Approximating the surface of a zonotope can be done in multiple ways, including bounding boxes or various ellipsoid approximations ([6], [21], [34], [29]). However, their major drawback is to not necessarily preserve the shape of the zonotope. Usually, the quality of the approximation is related to the time computation.

We shall concentrate on conserving the shape of a zonotope, by approximating its vertex set. They can be approximated in two ways: either compute the necessary subset of vertices which sufficiently express the zonotope shape, or find points closest to each zonotope vertex up to some tolerance. These are respectively the two following algorithms:

Stinson et al. randomized vertex enumeration[53]. For a zonotope $\mathcal{Z}(G)$ with $G \in \mathbb{R}^{n \times m}$, this algorithm samples vectors in the zonotope space \mathbb{R}^n using a standard Gaussian distribution in \mathbb{R}^m . These vectors are then projected onto \mathbb{R}^m via $\mathbf{x} \mapsto \text{sign}(G^T \mathbf{x})$, with sign assigning -1 if its value is negative and 1 otherwise. So the returning vector in \mathbb{R}^m corresponds to some vertex of the m -cube. The author showed that the probability of such a created vertex will map to a zonotope vertex with probability 1. This approach is necessarily fast since it requires only a sampling of a chosen amount of \mathbb{R}^n vectors. However, a major caveat appears if the zonotope generators have mutual angles close to each other, as shown in figure 2.13. In this case, many randomly chosen vectors can map to the same zonotope vertex and rarely to other vertices. To leverage this, the sampling size must be increased.

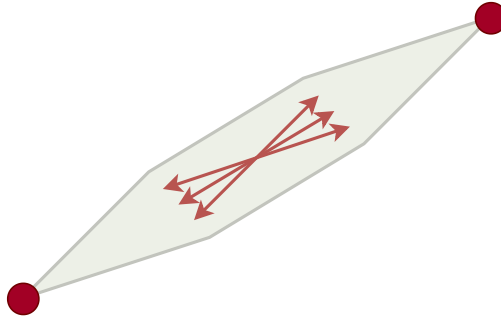


Figure 2.13: Stinson et al.’s randomized vertex enumeration algorithm. This is an example where the generators are very close angle-wise to each other. In this case, the chosen random vertices have a high probability to correspond to the same zonotope vertex (in red). The sampling size must be increased to avoid this effect.

Skuric et al.’s iterative convex hull method (ICH) [52]. Recently, a novel approximation of a polytope surfaces has been proposed in Skuric et al. [52]. The following figures 2.14 summarize this method. For a zonotope $\mathcal{Z}(n, m)$ in \mathbb{R}^n , this algorithm begins with a selection of n direction vectors in \mathbb{R}^n then compute the intersection of their associated line with the zonotope itself. While this requires to describe a set of bounding

hyperplanes of \mathcal{Z} , (which we do not have when using a generator matrix representation), it can be leveraged by working in the m -cube whose bounding hyperplanes can be enumerated easily. Indeed, the previously generated line in \mathbb{R}^n is also a line in \mathbb{R}^m , so that it remains to intersect it with the m -cube to return two vectors whose projection will map onto the line intersection with the zonotope bounding hyperplanes. Once all intersection points found, compute their convex hull and describe their bounding hyperplanes. Redo the intersection process from the line whose direction are the hyperplane normals. The algorithm stops when the produced points in \mathbb{R}^m are sufficiently close, up to some tolerance, to a zonotope vertex. A Python implementation is available and described in [51].

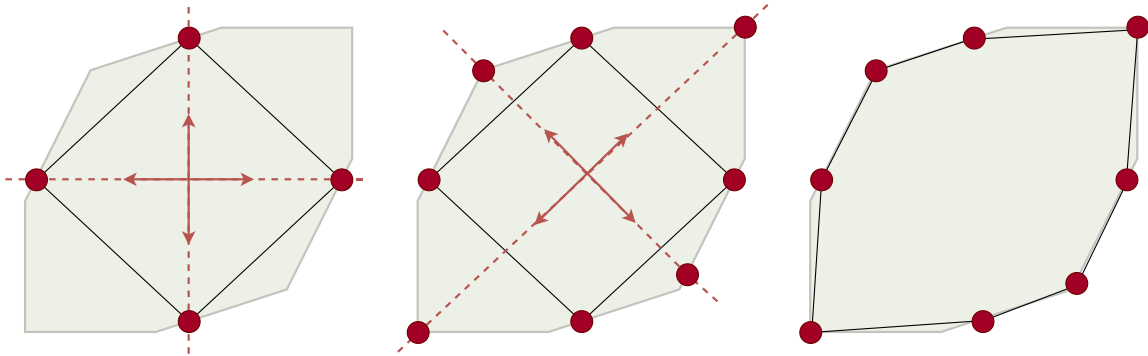


Figure 2.14: In black: the zonotope returned by the Iterative Convex Hull method. To the left: choose $n = 2$ vectors in general position, and compute by a linear program its intersection with the given zonotope. Compute the faces of the newly created polytope. At the middle: for each new face, consider the line directed by its normal and intersect it with \mathcal{Z} using a linear program. To the right: repeat the process until the distance between a created face and the zonotope is less than a specified tolerance.

To compare the quality of produced vertices from each of these algorithm, a metric should be defined. While there exists a computable distance between sets of points involving the euclidean distance from every point of a set to every point of another set (called *Hausdorff* distance), it is not necessarily a relevant metric. Indeed, Skuric et al's method returns a sensible amount of points which are close to the surface of the zonotope, much more than the expected number of vertices. Since it is the global shape of the zonotope which interests us, comparing the relative distance of returned points and the exact vertices would not give a strong information on whether or not the zonotope's surface is well approximated. Instead, we shall use the *Jaccard index*, also called the Jaccard similarity coefficient. It is defined for any two sets A and B as

$$J(A, B) = \frac{\text{Vol}(A \cap B)}{\text{Vol}(A \cup B)}$$

where Vol denotes the volume. If two sets are superposed, then the Jaccard index equals 1. For both presented algorithms, the returning points are on the surface of the given zonotope, so that the intersection and union can be computed easily.

The two next tables summarize both algorithm computation time for 4-dimensional zonotopes with either $m = 15$ or $m = 20$ generators. Table 2.4 shows results for Stinson et al.'s algorithm and table 2.5 for Skuric et al.'s. All computations have been performed using the same setup as the previous time benchmark in subsection 2.3.1.

(n, m)	Number of samples	Computation time	Number found points / exact number of vertices	Jaccard index
(4, 15)	10	< 0.01	0.41 ± 0.03	0.98 ± 0.01
	100	< 0.01	0.67 ± 0.05	1.00 ± 0.00
	1000	0.05 ± 0.00	0.83 ± 0.02	1.00 ± 0.00
	10000	0.30 ± 0.01	0.93 ± 0.01	1.00 ± 0.00
(4, 20)	10	< 0.01	0.28 ± 0.01	0.97 ± 0.00
	100	0.02 ± 0.00	0.56 ± 0.03	1.00 ± 0.00
	1000	0.08 ± 0.00	0.78 ± 0.03	1.00 ± 0.00
	10000	0.39 ± 0.02	0.89 ± 0.01	1.00 ± 0.00

Table 2.4: Stinson’s et al randomized algorithm. 10 randomly generated 4-dimensional zonotopes with m generators are computed for each line. The number of samples column correspond to the number of vertices randomly drawn at each iteration. Computation time are in seconds.

(n, m)	Tolerance	Computation time	Number found points / exact number of vertices	Jaccard index
(4, 15)	1	0.03 ± 0.01	0.03 ± 0.01	0.56 ± 0.05
	0.1	0.29 ± 0.02	0.72 ± 0.10	0.98 ± 0.00
	0.01	0.39 ± 0.01	1.94 ± 0.6	1.00 ± 0.00
	0.001	0.39 ± 0.02	2.12 ± 0.04	1.00 ± 0.00
(4, 20)	1	0.06 ± 0.01	0.03 ± 0.01	0.68 ± 0.05
	0.1	0.54 ± 0.03	0.61 ± 0.05	0.99 ± 0.00
	0.01	0.64 ± 0.04	1.36 ± 0.07	1.00 ± 0.00
	0.001	0.62 ± 0.04	1.37 ± 0.07	1.00 ± 0.00

Table 2.5: Skuric et al ICH algorithm. 10 randomly generated 4-dimensional zonotopes with m generators are computed for each line. The tolerance refers to how close could be a computed point from the zonotope surface. Computation time are in seconds.

These tables confirm that both algorithms can produce a set of points assuring a similar shape to a given zonotope. Stinson’s et al produces such a set very fastly compared to exact algorithms, and with a number of required points much less than the exact amount. However, Skuric et al’s method requires a to set a low tolerance in order to ensure its produced points are close to the zonotope surface, inducing longer computation time related to a higher amount of generated points.

While results concerning exact and approximation algorithms are related to the surface description of a zonotope, they are strictly different problems. This section showed that in our context of describing the shape of the torque feasible set, an exact enumeration algorithm should **not** be used when considering a large amount of muscles. However, working with exact algorithms allowed to have more insights upon the geometric process occurring when describing the projection of a tension set when it is modeled as an orthotope. This understanding is the key to generalize the projection process of any convex shape for \mathcal{T} , and is the focus of the next chapter 3.

2.5 Conclusion

In this chapter, we focused on computing explicitly the torque feasible set $\{-L^T \mathbf{t} - \mathbf{G} \in \mathbb{R}^n \mid \mathbf{t} \in \mathcal{T} \subset \mathbb{R}^m\}$ assuming that \mathcal{T} was shaped as an orthotope, and without loss of generality as a m -dimensional cube. The produced set is thus a zonotope, which can be described through various representations including a set of vertices. However, in our case the torque feasible set is associated to the matrix $-L^T$, which do not directly describe the global shape of its associated zonotope. We then proposed an efficient algorithm, called EdgeEnum, to compute the exact vertices of a zonotope based on its edge enumeration. It was compared theoretically in time and space complexity against various state-of-the-art zonotope enumeration algorithms, to show that we created a theoretically performant algorithm for worst case scenarii, *i.e.* when the number of columns of $-L^T$ is large compared to its number of rows. In simpler terms, when $n \ll m$.

However, in practice our algorithm can not be performant due to the unavoidable combinatorial amount of vertices to enumerate. A time benchmark has been performed to show that EdgeEnum can be faster than state-of-the-art algorithms when $n \geq 6$ and $m > 10$, but still requires a considerable amount of time in our context. Nevertheless, the next paragraphs summarize the benefits and limits of using an edge-based algorithm.

2.5.1 Strengths of EdgeEnum

EdgeEnum showed several advantages from its creation process to its implementation.

Easy to implement. EdgeEnum is straight-forward to implement. While a recursive version is possible (and implemented in its associated Python package), we preferred to show an implementation based on an iterative construction of edges of a m -cube. This is to emphasize readability as well as the geometric intuition. The only requirement needed for an implementation in any language is the availability of the convex hull operation, which fortunately is already implemented in R, MatLAB, Python and Julia which are common scientific programming languages.

Parallelism. A strong advantage of EdgeEnum compared to recent development such as Gu et al. [27] algorithm is the possibility to parallelize *easily* EdgeEnum. This step occurs during the grouping process, in which all current edges during an iteration are separated into groups according to their directions: applying the convex hull on a group of edge do not require the resulting convex hull of another group. Our Python implementation offers the parallel and non-parallel version of EdgeEnum.

Theoretical efficiency when n is fixed. Theorems 2.2.4 and 2.2.8 showed that our algorithm was polynomially bounded in the input and output sizes both in time and space complexity. This is fundamental as it qualifies EdgeEnum as an efficient algorithm in time and space.

Faster than the state-of-the-art for large m and $n > 6$. While related to the theoretical notion of efficiency, the numerical time benchmarks show that in practice EdgeEnum is indeed faster than state-of-the-arts algorithms when dealing with a zonotope of generator matrix in $\mathbb{R}^{n \times m}$. It should be noted that for $n = 2$, our algorithm can never

be better than Gu et al.’s algorithm, since they reach almost optimality in this specific case.

EdgeEnum handles degeneracy. Degeneracy of a zonotope describes if its generators are in general position or not. If they are, the zonotope is not degenerate. The only condition to handle it is the ability of the convex hull operation to return multiple points which are located at the same position. In this case, the convex hull time complexity is not $O(n \log h)$ where n is the number of points and h the number of points on the convex hull: it is instead $O(n \log n)$ on average, using an algorithm like *QuickHull* [2]. When applying this new bound into our computed theoretical complexities, this actually does not modify the returning result. However, our computed complexities becomes *average* bounds, and not worst-case bounds.

However, there are several drawbacks as we shall describe it in the paragraphs.

2.5.2 Limits

We can state that in our case, there are three types of limits related to theoretical algorithmic results, extension of our method to polytopes, and to our context thesis of grasping knowledge about a zonotope surface.

Non-compactness. An algorithm is *compact* if its space complexity is polynomially bounded by the input size only [19]. While our new approach is polynomially bounded in time and space for n fixed, it requires to store all edges found in each iteration. Our algorithm does not have the property of *compactness*, which means that its space complexity is polynomially bounded by the input size only [19]. Indeed, the input size is the generator matrix, which is of size nm , and even for n fixed, since the space complexity is in this case $O(m^{n+2})$, it is not bounded by some polynomial $(nm)^\alpha$, for α a positive real number. In other words, our algorithm is not able to stream all edges once found, since an edge can be rejected in a next iteration.

As explained by Ferrez et al. [17], our algorithm lies in the *incremental* strategy category. This means that the edge enumeration problem is solved inductively, by maintaining a list of edges at a certain state and the memory requirement is a critical disadvantage of this kind of approach. So the time efficiency of our algorithm is actually counteracted by its required space.

Approximation algorithms are better-suited for a fast zonotope surface description. Even if EdgeEnum remains relevant for practical problems ([27], [19], [28]), the description of a zonotope surface is not necessary one of them. Section 2.4 showed that there exist algorithms which are sufficiently performant in time to find points on a zonotope surface and whose convex hull is sufficiently close to what should be expected (in volume).

EdgeEnum does not (always) extend to the vertex enumeration of zonotope sections. While we showed results concerning the torque feasible set, our main interest as always been the force feasible set *i.e.* a section of a zonotope. If p the dimension of this section equals $n - 1$, then it is possible to construct some vertices of the polytope resulting from this section (*cf.* figure 2.15). If the affine subspace sectionning the zonotope

is sufficiently in general position (*i.e.* it does not cross any vertices of the zonotope), then it is even possible to enumerate all of its vertices. However, in our context we usually have the force feasible set described in $3D$, the torque space in $7D$ and the tension space in mD for $m \geq 7$, so it is not possible to return all vertices. This is mainly due to the fact the polytope vertices are produced from intersecting this affine space with higher dimensional faces of the zonotope, and not necessarily *only* edges.

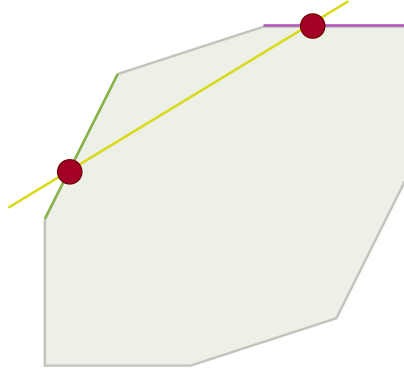



Figure 2.15: In yellow: an affine subspace L of dimension 1 which do not intersect zonotope vertices. In this case, and only because $1 = n - 1$ with $n = 2$ the dimension of the zonotope, it is possible to construct the vertices of this zonotope section by intersecting L with crossing edges.

2.5.3 On the cubic representation of the tension set

Our novel algorithm is theoretically relevant in the vertex enumeration subfield of Computational Geometry. It sheds new lights onto enumeration techniques. To our knowledge, this is the first time an algorithm uses cube edges combinatorics in order to retrieve vertices. It can be assumed that edges are not necessarily treated as a subject of interest due to their amount (the total number of m -cube edges is always much larger than its number of vertices).

In practice, the number of zonotope vertices combinatorially explodes with its number of generators, making *any* enumeration algorithm inefficient in regard to the time computation. In section 2.4, approximations of the zonotope surface have been considered in order to decrease much further the computation time and leveraging the number of points required to describe the zonotope surface. These approximations are relevant in this thesis context: to reconstruct a musculoskeletal model whose torque feasible sets fit given maximal torque capacities in various postures, we must be able to generate in silico a large amount of torque feasible sets and compare them.

However, whether are not approximations or exact algorithms are used, it was underlyingly hypothesized that the tension set has a cubic shape. Biomechanically, this means that all muscles produce their maximum tensions at the same time. In the next chapter, we do not assume this and we diligently focus on the force feasible set shape, for any possible tension set. The creation process of EdgeEnum is a first step in this direction: we described the global shape of the torque feasible set by navigating between the tension set and its projection onto the torque space. In a more general manner, the shape of the tension set and its projection (followed or not by an intersection) are strongly linked.

They are even *inextricable*, and the major results of the next chapter serve as argumenting on why **does** the shape of the tension set does not strictly matter when a large number of muscles is considered. Surprisingly, the understanding of these results led to a  approximation of *any* projected force feasible set, and a deeper understanding of muscle activation interactions.