On Computing the Vertex Centroid of a Polyhedron*

Khaled Elbassioni[†] Hans Raj Tiwary[‡]
elbassio@mpi-sb.mpg.de hansraj@cs.uni-sb.de

April 11, 2008

Abstract

Let $\mathcal{P}$ be an $\mathcal{H}$-polytope in $\mathbb{R}^d$ with vertex set V. The vertex centroid is defined as the average of the vertices in V. We prove that computing the vertex centroid of an $\mathcal{H}$-polytope is #P-hard. Moreover, we show that even just checking whether the vertex centroid lies in a given halfspace is already #P-hard for $\mathcal{H}$-polytopes. We also consider the problem of approximating the vertex centroid by finding a point within an ϵ distance from it and prove this problem to be #P-easy by showing that given an oracle for counting the number of vertices of an $\mathcal{H}$-polytope, one can approximate the vertex centroid in polynomial time. We also show that any algorithm approximating the vertex centroid to *any* "sufficiently" non-trivial (for example constant) distance, can be used to construct a fully polynomial approximation scheme for approximating the centroid and also an output-sensitive polynomial algorithm for the Vertex Enumeration problem. Finally, we show that for unbounded polyhedra the vertex centroid can not be approximated to a distance of $d^{\frac{1}{2}-\delta}$ for any fixed constant $\delta > 0$.

1 Introduction

An intersection of a finite number of closed halfspaces in $\mathbb{R}^d$ defines a polyhedra. A polyhedra can also be represented as $conv(V) + cone(Y)$, the Minkowski sum of the convex hull of a finite set of points V and the cone of a finite set of rays. A bounded polyhedra is called a polytope. In what follows, we will discuss mostly polytopes for simplicity and refer to the unbounded case explicitly only towards the end.

Let $\mathcal{P}$ be an $\mathcal{H}$-polytope in $\mathbb{R}^d$ with vertex set V. Various notions try to capture the essence of a "center" of a polytope. Perhaps the most popular notion is that of the center of gravity of $\mathcal{P}$. Recently Rademacher proved that computing the center of gravity of a polytope is #P-hard [7]. The proof essentially relies on the fact that the center of gravity captures the volume of a polytope perfectly and that computing the volume of a polytope is #P-hard [3]. Note that, polynomial algorithms exist that approximate the volume of a polytope within any arbitrary factor [4]. It is also easy to see that the center of gravity can be approximated by simply sampling random points from the polytope, the number of samples depending polynomially on the desired approximation (See Algorithm 5.8 of [4]).

In this paper we study a variant of the notion of "center" defined as the centroid (average) of the vertices of P. Despite being quite a natural feature of polytopes, this variant seems to have received very little attention both from theoretical and computational perspectives. Throughout this paper we will refer to the vertex centroid just as centroid. The reader should note that

---

in popular literature the word centroid refers more commonly to the center of gravity. We nevertheless use the same terminology for simplicity of language. Our motivation for studying the centroid stems from the fact that the centroid encodes the number of vertices of a polytope. As we will see, this also makes computing the centroid hard.

The parallels between centroid and the center of gravity of a polytope mimic the parallels between the volume and the number of vertices of a polytope. Computing the volume and the number of vertices are both #P-complete ([2, 3, 6]) and so are the problems of computing the corresponding centroids ([7], Theorem 1). The volume can be approximated quite well but approximating the number of vertices of a polytope is an interesting open problem. Similarly, the center of gravity can be approximated quite well but (as we will see in this paper) obtaining a polynomial algorithm for approximating the centroid would be a very interesting achievement.

The problem of enumerating vertices of an $\mathcal{H}$-polytope has been studied for a long time. However, in spite of years of research it is neither known to be hard nor is there an output sensitive polynomial algorithm for it. A problem that is polynomially equivalent to the Vertex Enumeration problem is to decide if a given list of vertices of an $\mathcal{H}$-polytope is complete [1]. In this paper we show that any algorithm that approximates the centroid of an arbitrary polytope to any "sufficiently" non-trivial distance can be used to obtain an output sensitive polynomial algorithm for the Vertex Enumeration problem.

The main results of this paper are the following:

- Computing the centroid of an $\mathcal{H}$-polytope is #P-hard.

- Even just deciding whether the centroid of an $\mathcal{H}$-polytope lies in a halfspace remains #P-hard.

- Approximating the centroid of an $\mathcal{H}$-polytope is #P-easy.

- Any algorithm approximating the centroid of an arbitrary polytope within a distance $d^{\frac{1}{2}-\delta}$ can be used to obtain a fully polynomial approximation scheme for the centroid approximation problem and also an output sensitive polynomial algorithm for the Vertex Enumeration problem.

- There is no polynomial algorithm that approximates the vertex centroid of arbitrary $\mathcal{H}$-polyhedron within a distance $d^{\frac{1}{2}-\delta}$ for any fixed constant $\delta > 0$, unless $P = NP$.

We should remark that for the approximation of centroid, we only consider polytopes (and polyhedra) whose vertices lie inside a unit hypercube. To see how this assumption can easily be satisfied, notice that a halfspace h can be added to a polyhedron P such that $P \cap h$ is bounded and the vertices of P are preserved in $P \cap h$. Also, such a halfspace can be found in polynomial time from the inequalities defining P. Once we have a polytope in $\mathbb{R}^d$, solving $2d$ linear programs gives us the width along each coordinate axis. The polytope can be scaled by a factor depending on the width along each axis to obtain a polytope all whose vertices lie inside a unit hypercube. In case we started with a polyhedron P, the scaled counterpart of the halfspace h that was added can be thrown to get back a polyhedron that is a scaled version of P and all whose vertices lie inside the unit hypercube. In subsection 2.2 we provide further motivation for this assumption.

Since all the vertices of the polytope (or polyhedron) lie inside a unit hypercube, picking any arbitrary point from inside this hypercube yield a $d^{\frac{1}{2}}$-approximation of the vertex centroid. Thus, the last result should be contrasted to the fact that approximating the vertex centroid within a distance of $d^{\frac{1}{2}}$ is trivial. Also, even though we discuss only polytopes *i.e.* bounded polyhedra in subsections 2.1 and 2.2, the results and the proofs are valid for the unbounded case as well. We discuss the unbounded case explicitly only in subsection 2.3.

2

2 Results

2.1 Exact Computation of the Centroid

The most natural computational question regarding the centroid of a polytope is whether we can compute the centroid efficiently. The problem is trivial if the input polytope is presented by its vertices. So we will assume that the polytope is presented by its facets. Perhaps not surprisingly, computing the centroid of an $\mathcal{H}$-polytope turns out be #P-hard. We prove this by showing that computing the centroid of an $\mathcal{H}$-polytope amounts to counting the vertices of the same polytope, a problem known to be #P-hard.

Theorem 1 *Given an $\mathcal{H}$-polytope $\mathcal{P} \subset \mathbb{R}^d$, it is #P-hard to compute its centroid $c(\mathcal{P})$.*

Proof. Embed $\mathcal{P}$ in $\mathbb{R}^{d+1}$ by putting a copy of $\mathcal{P}$ in the hyperplane $x_{d+1} = 1$ and making a pyramid with the base $\mathcal{P}$ and apex at the origin. Call this new polytope $\mathcal{Q}$. Treating the direction of the positive x_{d+1}-axis as up, it is easy to see that the centroid of the new polytope lies at a height $1 - \frac{1}{n+1}$ iff the number of vertices of $\mathcal{P}$ is n. Thus any algorithm for computing the centroid can be run on $\mathcal{Q}$ and the number of vertices of $\mathcal{P}$ can be read off the $(d+1)$-st coordinate. $\qquad\square$

Suppose, instead, that one does not want to compute the centroid exactly but is just interested in knowing whether the centroid lies to the left or to the right of a given arbitrary hyperplane. This problem turns out to be hard too, and it is not difficult to see why.

Theorem 2 *Given an $\mathcal{H}$-polytope $\mathcal{P} \subset \mathbb{R}^d$ and a hyperplane $h = \{a \cdot x = b\}$, it is #P-hard to decide whether $a \cdot c(\mathcal{P}) \leq b$.*

Proof. Consider the embedding and the direction pointing upwards as used in the proof of Theorem 1. Given an oracle answering sidedness queries for the centroid and any arbitrary hyperplane, one can perform a binary search on the height of the centroid and locate the exact height. The number of queries needed is only logarithmic in the number of vertices of $\mathcal{P}$. $\qquad\square$

2.2 Approximation of the Centroid

As stated before, even though computing the gravitational centroid of a polytope exactly is #P-hard, it can be approximated to any precision by random sampling. Now we consider the problem of similarly approximating the vertex centroid of an $\mathcal{H}$-polytope. Let $dist(x, y)$ denote the Euclidean distance between two points $x, y \in \mathbb{R}^d$. We are interested in the following problem:

Input: $\mathcal{H}$-polytope $P \subset \mathbb{R}^d$ and a real number $\epsilon > 0$.
Output: $p \in \mathbb{R}^d$ such that $dist(c(P), p) \leq \epsilon$.

We would like an algorithm for this problem that runs in time polynomial in the number of facets of P, the dimension d and $\frac{1}{\epsilon}$. Clearly, such an algorithm would be very useful because if such an algorithm is found then it can be used to test whether a polytope described by m facets has more than n vertices, in time polynomial in m, n and the dimension d of the polytope by setting $\epsilon < \frac{1}{2}\left(\frac{1}{n} - \frac{1}{n+1}\right)$. This in turn would yield an algorithm that computes the number of vertices n of a d-dimensional polytope with m facets, in time polynomial in m, n and d. As stated before, a problem that is polynomially equivalent to the Vertex Enumeration problem is to decide if a given list of vertices of an $\mathcal{H}$-polytope is complete [1]. Clearly then, a polynomial

approximation scheme for the centroid problem would yield an output-sensitive polynomial algorithm for the Vertex Enumeration problem.

Also, the problem of approximating the centroid is not so interesting if we allow polytopes that contain an arbitrarily large ball, since this would allow one to use an algorithm for approximating the centroid with *any* guarantee to obtain another algorithm with an arbitrary guarantee by simply scaling the input polytope appropriately, running the given algorithm and scaling back. So we will assume that the polytope is contained in a unit hypercube in $\mathbb{R}^d$.

Now we prove that the problem of approximating the centroid is #P-easy. We do this by showing that given an algorithm that computes the number of vertices of an arbitrary polytope (a #P-complete problem), one can compute the centroid to any desired precision by making a polynomial (in $\frac{1}{\epsilon}$, the number of facets and the dimension of the polytope) number of calls to this oracle. Notice that in the approximation problem at hand, we are required to find a point within a d-ball centered at the centroid of the polytope and radius ϵ. We first modify the problem a bit by requiring to report a point that lies inside a hypercube, of side length 2ϵ, centered at the centroid of the polytope. (The hypercube has a clearly defined center of symmetry, namely its own vertex centroid.) To see why this does not essentially change the problem, note that the unit hypercube fits completely inside a d-ball with the same center and radius $\frac{\sqrt{d}}{2}$. We will call any point that is a valid output to this approximation problem, an ϵ-approximation of the centroid $c(P)$.

Given an $\mathcal{H}$-polytope P and a hyperplane $\{a \cdot x = b\}$ that intersects P in the relative interior and does not contain any vertex of P, define P_1 and P_2 as follows:

$$
\begin{aligned}
P_1 &= P \cap \{x | a \cdot x \leq b\} \\
P_2 &= P \cap \{x | a \cdot x \geq b\}
\end{aligned}
$$

Let V_1 be the common vertices of P_1 and P, and V_2 be common vertices of P_2 and P. The following lemma gives a way to obtain the ϵ-approximation of the centroid of P from the ϵ-approximations of the centroids of V_1 and V_2.

Lemma 1 *Given P, V_1, V_2 defined as above, let n_1 and n_2 be the number of vertices in V_1 and V_2 respectively. If c_1 and c_2 are ϵ-approximations of the centroids of V_1 and V_2 respectively, then $c = \frac{n_1 c_1 + n_2 c_2}{n_1 + n_2}$ is an ϵ-approximation of the centroid c^* of P.*

Proof. Let c_{ij} be the j-th coordinate of c_i for $i \in \{1, 2\}$. Also, let c_i^* be the actual centroid of V_i with c_{ij}^* denoting the j-th coordinate of c_i^*. Since c_i approximates c_i^* within a hypercube of side-length 2ϵ, for each $j \in \{1, \cdots, d\}$ we have

$$
c_{ij}^* - \epsilon \leq \quad c_{ij} \quad \leq c_{ij}^* + \epsilon
$$

Also, since c^* is the centroid of P,

$$
c^* = \frac{n_1 c_1^* + n_2 c_2^*}{n_1 + n_2}
$$

Hence, for each coordinate c_j^* of c^* we have

$$
\begin{aligned}
\frac{n_1(c_{1j} - \epsilon) + n_2(c_{2j} - \epsilon)}{n_1 + n_2} &\leq c_j^* \leq \frac{n_1(c_{1j} + \epsilon) + n_2(c_{2j} + \epsilon)}{n_1 + n_2} \\
\Rightarrow \quad \frac{n_1 c_{1j} + n_2 c_{2j}}{n_1 + n_2} - \epsilon &\leq c_j^* \leq \frac{n_1 c_{1j} + n_2 c_{2j}}{n_1 + n_2} + \epsilon \\
\Rightarrow \quad c_j - \epsilon &\leq c_j^* \leq c_j + \epsilon \\
\Rightarrow \quad c_j^* - \epsilon &\leq c_j \leq c_j^* + \epsilon
\end{aligned}
$$

$\square$

Now to obtain an approximation of the centroid, we first slice the input polytope P from left to right into $\frac{1}{\epsilon}$ slices each of thickness at most ϵ. Using standard perturbation techniques we can ensure that any vertex of the input polytope does not lie on the left or right boundary of any slice. For each slice any point in the interior gives us an ϵ-approximation of the vertices of P that are contained in that slice. We can compute the number of vertices of P lying in this slice by using the oracle for vertex computation and then using the previous Lemma we can obtain the centroid of P. Thus we have the following theorem:

Theorem 3 *Given a polytope P contained in the unit hypercube, the ϵ-approximation of the centroid of P can be computed by making a polynomial number of calls to an oracle for computing the number of vertices of a polytope.*

Now we present a bootstrapping theorem indicating that any "sufficiently" non-trivial approximation of the centroid can be used to obtain arbitrary approximations. For the notion of approximation let us revert back to the Euclidean distance function. Thus, any point x approximating the centroid c within a parameter ϵ satisfies $dist(x,c) \leq \epsilon$. As before we assume that the polytope $\mathcal{P}$ is contained in the unit hypercube. Since the polytope is thus contained in a hyperball with origin as its center and radius at most $\frac{\sqrt{d}}{2}$, any point inside $\mathcal{P}$ approximates the centroid within a factor $\sqrt{d}$. Before we make precise our notion of "sufficiently" non-trivial and present the bootstrapping theorem, some preliminaries are in order.

Lemma 2 *Suppose $(x,y),(u,u) \in \mathbb{R}^{2d}$, where $x,y,u \in \mathbb{R}^d$, then*

$$||u - \frac{x+y}{2}|| \leq \frac{||(u,u)-(x,y)||}{\sqrt{2}},$$

where $||\cdot||$ is the Euclidean norm.

The proof of the above lemma is easy and elementary, and hence we omit it here. Next, consider the product of two polytopes. Given d-dimensional polytopes $\mathcal{P}, \mathcal{Q}$ the product $\mathcal{P} \times \mathcal{Q}$ is defined as the set $\{(x,y)|x \in \mathcal{P}, y \in \mathcal{Q}\}$. The facet defining inequalities of the product of P,Q can be computed easily from the inequalities defining P and Q.

$$
\begin{aligned}
P &= \{x|A_1 x \leq b_1\}, \\
Q &= \{y|A_2 y \leq b_2\} \\
\Rightarrow P \times Q &= \{(x,y)|A_1 x \leq b_1, A_2 y \leq b_2\},
\end{aligned}
$$

where $A_1 \in \mathbb{R}^{m_1 \times d_1}, A_2 \in \mathbb{R}^{m_2 \times d_2}, x \in \mathbb{R}^{d_1}, y \in \mathbb{R}^{d_2}, b_1 \in \mathbb{R}^{m_1 \times 1}, b_2 \in \mathbb{R}^{m_2 \times 1}$.

It is easy to see that the number of vertices of $\mathcal{P} \times \mathcal{Q}$ is the product of the number of vertices of $\mathcal{P}$ and that of $\mathcal{Q}$, and the number of facets of $\mathcal{P} \times \mathcal{Q}$ is the sum of the number of facets of $\mathcal{P}$ and that of $\mathcal{Q}$. Moreover, the dimension of $\mathcal{P} \times \mathcal{Q}$ is the sum of the dimensions of $\mathcal{P}$ and that of $\mathcal{Q}$.

Observation 1 *If c is the centroid of a polytope P then (c,c) is the centroid of $P \times P$.*

Suppose we are given an algorithm for finding ϵ-approximation of an arbitrary polytope contained in the unit hypercube. For example, for the simple algorithm that returns an arbitrary point inside the polytope, the approximation guarantee is $\frac{\sqrt{d}}{2}$. We consider similar algorithms

whose approximation guarantee is a function of the ambient dimension of the polytope. Now suppose that for the given algorithm the approximation guarantee is $f(d)$. For some parameter k consider the k-fold product of P with itself $\overbrace{P \times \cdots \times P}^{k \ times}$, denoted by P^k. Using the given algorithm one can find the $f(2^k d)$ approximation of P^k and using Lemma 2 one can then find the $\frac{f(2^k d)}{\sqrt{2^k}}$-approximation of P. This gives us the following bootstrapping theorem:

Theorem 4 *Suppose we are given an algorithm that computes a $\frac{\sqrt{d}}{g(d)}$-approximation for any polytope contained in the unit hypercube in polynomial time, where $g(.)$ is an unbounded monotonically increasing function. Then, one can compute an ϵ-approximation in time polynomial in the size of the polytope and $g^{-1}(\frac{\sqrt{d}}{\epsilon})$.*

In particular, if we have an algorithm with $d^{\frac{1}{2}-\delta}$ approximation guarantee for finding the centroid of any polytope for some fixed constant $\delta > 0$, then this algorithm can be used to construct a fully polynomial approximation scheme for the general problem.

2.3 Approximating centroid of a polyhedron is hard

The reader should note that the analysis of subsections 2.1 and 2.2 remains valid even for the unbounded case (polyhedra). Even though we do not have any idea about the complexity of approximating the centroid of a polytope, now we show that for an arbitrary unbounded polyhedron the vertex centroid can not be $d^{\frac{1}{2}-\delta}$-approximated for any fixed constant $\delta > 0$. To show this we first prove that for an $\mathcal{H}$-polyhedron $P \subset \mathbb{R}^d$ the vertex centroid of P can not be $\frac{1}{d}$-approximated in polynomial time unless $P = NP$. This together with Theorem 4 completes the proof for hardness of $d^{\frac{1}{2}-\delta}$-approximation of the centroid of an $\mathcal{H}$-polyhedron.

The proof sketch is as follows: Given a boolean CNF formula ϕ, we construct a graph $G(\phi)$ such that $G(\phi)$ has a "long" negative cycle if and only if ϕ is satisfiable. For a given graph G we define a polyhedron $P(G)$ such that every negative cycle in G is a vertex of $P(G)$ and vice-versa. From the properties of the vertex centroid of this class of polyhedron, we then prove that for any formula ϕ, $\frac{1}{d}$-approximating the vertex centroid of $P(G(\phi))$ would reveal whether ϕ is satisfiable or not.

Before we proceed further, we would like to remark that for the graph corresponding to a given 3-CNF formula, we use the same construction that appears in [5] except for different edge weights. We nevertheless include the discussion here for completeness. Our main contribution is the negative-flow polyhedron associated with a directed graph and a set of edge weights and the analysis of the vertices of this polyhedron leading to Theorem 5.

2.3.1 Graph of a CNF formula

Given a 3-CNF $\phi(x_1, \ldots, x_N) = C_1 \wedge \ldots \wedge C_m$, we construct a weighted directed graph $G = (V, E)$, on $|V| = 5 \sum_{j=1}^{m} |C_j| + m - n + 1 = 16m - n + 1$ vertices and $|E| = 6 \sum_{j=1}^{m} |C_j| + 1 = 18m + 1$ arcs (where $|C_j|$ denotes the number of literals appearing in clause C_j, *i.e.* 3 for a 3-CNF), as follows. For each literal $\ell = \ell^j$ appearing in clause C_j, we introduce two paths of three arcs each: $\mathcal{P}(\ell) = (p(\ell), a(\ell), b(\ell), q(\ell))$, and $\mathcal{P}'(\ell) = (r(\ell), b'(\ell), a'(\ell), s(\ell))$. The weights of these arcs are set as follows:

$$w((p(\ell), a(\ell))) = \tfrac{1}{2}, \quad w((a(\ell), b(\ell))) = -\tfrac{1}{2}, \quad w((b(\ell), q(\ell))) = 0,$$

$$w((r(\ell), b'(\ell))) = 0, \quad w((b'(\ell), a'(\ell))) = -\tfrac{1}{2}, \quad w((b'(\ell), s(\ell))) = \tfrac{1}{2}.$$
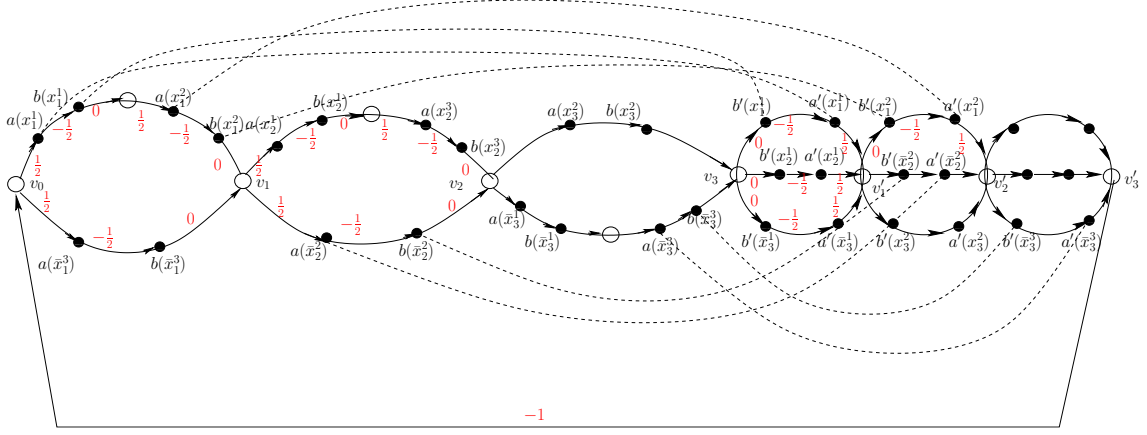
Figure 1: An example of the graph construction for CNF $C = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$.

These edges are connected in G as follows (see Figure 1 for an example):

$$G = v_0 \; \mathcal{G}_1 \; v_1 \; \mathcal{G}_2 \; v_2 \ldots \; v_{n-1} \; \mathcal{G}_n \; v_n \; \mathcal{G}'_1 \; v'_1 \; \mathcal{G}'_2 \; v'_2 \ldots \; v'_{m-1} \; \mathcal{G}'_m \; v'_m,$$

where $v_0, v_1, \ldots, v_n, v'_1, \ldots, v'_{m-1}, v'_m$ are distinct vertices, each $\mathcal{G}_i = \mathcal{Y}_i \vee \mathcal{Z}_i$, for $i = 1, \ldots, n$, consists of two parallel chains $\mathcal{Y}_i = \wedge_j \mathcal{P}(x_i^j)$ and $\mathcal{Z}_i = \wedge_j \mathcal{P}(\overline{x}_i^j)$ between v_{i-1} and v_i, and each $\mathcal{G}'_j = \vee_{i=1}^{|C_j|} \mathcal{P}'(\ell_i^j)$, for $j = 1, \ldots, m$, where $\ell_1^j, \ell_2^j, \ldots$ are the literals appearing in C_j.

Finally we add the arc (v'_m, v_0) with weight -1, and *identify* the pairs of nodes $\{a(\ell), a'(\ell)\}$ and $\{b(\ell), b'(\ell)\}$ for all ℓ, (i.e. $a(\ell) = a(\ell')$ and $b(\ell) = b(\ell')$ define the same nodes).

We will argue now that all negative cycles of G have weight -1. Clearly the arcs $(a(\ell), b(\ell))$ and $(b'(\ell), a'(\ell))$ form a directed cycle of total weight -1, for every literal occurrence ℓ. There are $\sum_{j=1}^m |C_j| = 3m$ such cycles, corresponding to a subset $\mathcal{X} \subseteq \mathcal{V}(P(G, w))$.

Call a cycle of G long if it contains the vertices $v_0, v_1, \ldots, v_n, v'_1, \ldots, v'_m$. Any long cycle has weight -1. The crucial observation is the following.

Claim 1 *Any negative cycle $C \in \mathcal{C}^-(G, w) \setminus \mathcal{X}$ must be long.*

Proof. Consider any cycle $C \notin \mathcal{X}$, and let us write the traces of the nodes visited on the cycle (dropping the literals, and considering a, a' and b, b' as different copies), without loss of generality as follows:

$$p \; a \; b \; p \; a \; b \; p \; \cdots \; a \; a' \; s \; b' \; a' \; s \; b' \; \cdots \; b' \; b \; p \; a \; b \; \cdots \; p.$$

Note that the sequences $a' \; a$ and $b \; b'$ are not allowed since otherwise C contains a cycle from $\mathcal{X}$.

Let us compute the distance (i.e., the total weight) of each node on this sequence starting from the initial p. Call the subsequences $a' \; a'$ and $b \; b'$, a- and b-jumps respectively. Then it is easy to verify that each a-jump causes the distance to eventually increase by 1 while each b-jump keeps the distance at its value. More precisely, the distance at a node x in the sequence is given by $d(x) = t + d_0 - \delta(x)$, where t is the number of a-jumps appearing upto x, and

$$d_0 = \begin{cases} 0 & \text{if } x \in \{p, s\}, \\ \frac{1}{2} & \text{if } x = a, \\ -\frac{1}{2} & \text{if } x = a', \\ 0 & \text{if } x = b = b', \end{cases} \qquad \delta(x) = \begin{cases} 1 & \text{if arc } (v'_m, v_0) \text{ appears on the path from } p \text{ to } x \\ 0 & \text{otherwise.} \end{cases}$$

7

One also observes that, if the sequence has a b-jump, then it must also contain an a-jump. Thus it follows from the definition of $d(x)$ that any cycle with a jump must be non-negative. So the only possible negative cycle not in $\mathcal{X}$ must be long. □

If we are interested only in the negative cycles then it is clear that long cycles are exactly those negative cycles that contain the edge (v'_m, v_0). In what follows this edge will be called the **marked** edge, the negative cycles containing this marked edge will be called **long** negative cycles and the other negative cycles will be called **short** negative cycles.

By Claim 1, checking of $\mathcal{V}(P(G, w)) = \mathcal{X}$ is equivalent to checking if G has a long cycle. It is easy to see that the latter condition is equivalent to the non-satisfiability of the input CNF formula ϕ (see [5] for more details).

To summarize this subsection, for a 3-CNF with m clauses we can obtain a graph G with $18m+1$ edges such that G has $3m$ short negative cycles and finding whether G has any negative cycle (called long negative cycle) containing one special edge is NP-complete. Furthermore, each of these negative cycles has total weight -1.

2.3.2 The polyhedron of negative-weight flows of a graph

Given a directed graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$ on its arcs, consider the following polyhedron:

$$
P(G, w) = \left\{ y \in \mathbb{R}^E \;\middle|\;
\begin{array}{ll}
(F) & \displaystyle\sum_{v:(u,v)\in E} y_{uv} - \sum_{v:(v,u)\in E} y_{vu} \;=\; 0 \quad \forall \; u \in V \\[2ex]
(N) & \displaystyle\sum_{(u,v)\in E} w_{uv} y_{uv} \;=\; -1 \\[2ex]
& y_{uv} \geq 0 \qquad\qquad\qquad\quad \forall \; (u,v) \in E
\end{array}
\right\}.
$$

If we think of $w_{u,v}$ as the cost/profit paid for edge (u, v) per unit of flow, then each point of $P(G, w)$ represents a *negative-weight circulation* in G, i.e., assigns a non-negative flow on the arcs, obeying the *conservation of flow* at each node of G, and such that total weight of the flow is strictly negative.

For a subset $X \subseteq E$, and a weight function $w : E \mapsto \mathbb{R}$, we denote by $w(X) = \sum_{e \in X} w_e$, the total weight of X. For $X \subseteq E$, we denote by $\chi(X) \in \{0, 1\}^E$ the characteristic vector of X: $\chi_e(X) = 1$ if and only if $e \in X$, for $e \in E$. The following theorem states that the vertex set $\mathcal{V}(P(G, w))$ of $P(G, w)$ is in one-to-one correspondence with the negative cycles of the graph G.

Theorem 5 *Let $G = (V, E)$ be a directed graph and $w : E \to \mathbb{R}$ be a real weight on the arcs. Then*

$$
\mathcal{V}(P(G, w)) \;=\; \left\{ \frac{-1}{w(C)} \chi(C) : \; C \in \mathcal{C}^-(G, w) \right\}. \tag{1}
$$

We prove this theorem by proving a series of claims about the graph G. Let $m = |E|$ and $n = |V|$. It is easy to verify that any element $y \in \mathbb{R}^E$ of the set on the right-hand side of (1) belongs to $P(G, w)$. Moreover, any such $x = -\chi(C)/w(C)$, for a cycle C, is a vertex of $P(G, w)$ since there are m linearly independent inequalities of $P(G, w)$ tight at x, namely: the conservation of flow equations (F) at $|C| - 1$ vertices of C, the equation $\sum_{e \in C} w_e y_e = -1$, and $m - |C|$ equations $y_e = 0$, for $e \in E \setminus C$.

To prove the opposite direction, let $y \in \mathbb{R}^E$ be a vertex of $P(G, w)$. Let $Y = \{e \in E : y_e > 0\}$. The proof follows from the following 3 claims.

Claim 2 *The graph (V, Y) is the disjoint union of strongly connected components.*

Proof. Consider an arbitrary strongly connected component X in this graph, and let X^- be the set of components reachable from X (including X). Summing the conservation of flow equations corresponding to all the nodes in X^- implies that all arcs going out of X^- have a flow of zero. $\square$

Claim 3 *There exists no cycle $C \in \mathcal{C}^0(G, w)$ such that $C \subseteq Y$.*

Proof. If such a C exists, we define two points y' and y'' as follows.

$$
y'_e = \begin{cases} y_e + \epsilon, & \text{if } e \in C \\ y_e, & \text{otherwise,} \end{cases} \qquad
y''_e = \begin{cases} y_e - \epsilon, & \text{if } e \in C \\ y_e, & \text{otherwise,} \end{cases}
$$

for some sufficiently small $\epsilon > 0$. Then y', y'' clearly satisfy (F). Moreover, (N) is satisfied by y' since

$$
\sum_{e \in E} w_e y'_e = \sum_{e \notin C} w_e y_e + \sum_{e \in C} w_e (y_e + \epsilon) = \sum_{e \in E} w_e y_e + w(C)\epsilon = -1.
$$

Similarly for y''. Thus $y', y'' \in P(G, w)$ and $y = (y' + y'')/2$ contradicting that y is a vertex. $\square$

Claim 4 *There exist no distinct cycles $C_1, C_2 \in \mathcal{C}^-(G, w) \cup \mathcal{C}^+(G, w)$ such that $C_1 \cup C_2 \subseteq Y$.*

Proof. If such C_1 and C_2 exist, we define two points y' and y'' as follows.

$$
y'_e = \begin{cases} y_e + \epsilon_1, & \text{if } e \in C_1 \setminus C_2 \\ y_e + \epsilon_2, & \text{if } e \in C_2 \setminus C_1 \\ y_e + \epsilon_1 + \epsilon_2, & \text{if } e \in C_1 \cap C_2 \\ y_e, & \text{otherwise,} \end{cases} \qquad
y'_e = \begin{cases} y_e - \epsilon_1, & \text{if } e \in C_1 \setminus C_2 \\ y_e - \epsilon_2, & \text{if } e \in C_2 \setminus C_1 \\ y_e - \epsilon_1 - \epsilon_2, & \text{if } e \in C_1 \cap C_2 \\ y_e, & \text{otherwise,} \end{cases}
$$

where $\epsilon_1 = -\frac{w(C_2)}{w(C_1)}\epsilon_2$, for some sufficiently small $\epsilon_2 > 0$ (in particular, to insure non-negativity of y', y'', ϵ_2 must be upper bounded by the minimum of $\min\{y_e : e \in C_2 \setminus C_1\}$, $\frac{|w(C_1)|}{|w(C_2)|}\min\{y_e : e \in C_1 \setminus C_2\}$, and $\frac{|w(C_1)|}{|w(C_1) - w(C_2)|}\min\{y_e : e \in C_1 \cap C_2\}$). Then it is easy to verify that y', y'' satisfy (F). Moreover, (N) is satisfied with y' since

$$
\begin{aligned}
\sum_{e \in E} w_e y'_e &= \sum_{e \notin C_1 \cup C_2} w_e y_e + \sum_{e \in C_1 \setminus C_2} w_e(y_e + \epsilon_1) + \sum_{e \in C_2 \setminus C_1} w_e(y_e + \epsilon_2) \\
&+ \sum_{e \in C_1 \cap C_2} w_e(y_e + \epsilon_1 + \epsilon_2) = \sum_{e \in E} w_e y_e + w(C_1)\epsilon_1 + w(C_2)\epsilon_2 = -1.
\end{aligned}
$$

Similarly for y''. Thus $y', y'' \in P(G, w)$ and $y = (y' + y'')/2$ contradicting that y is a vertex of $P(G, w)$. $\square$

The above 3 claims imply that the graph (V, Y) consists of a single cycle C and a set of isolated vertices $V \setminus V(C)$. Thus $y_e = 0$ for $e \notin C$. By (F) we get that y_e is the same for all $e \in C$, and by (N) we get that $y_e = -1/w(C)$ for all $e \in C$, and in particular that $C \in \mathcal{C}^-(G, w)$. This completes the proof of Theorem 5.

Since, in the graph G arising from a 3-CNF formula, every negative cycle has weight exactly -1, Theorem 5 implies that the vertices of $P(G, w)$ are exactly the characteristic vectors of the negative cycles of G. Recall that finding whether G has any long negative cycle, *i.e.* a negative

cycle containing a marked edge (say e), is NP-complete. Further recall that for a 3-CNF formula with m clauses the constructed graph G has $18m+1$ edges and $3m$ trivial short negative cycles. Consequently, the polyhedron $P(G, w)$ that is finally obtained has dimension $18m+1$ and $3m$ trivial vertices corresponding to the short negative cycles of G.

Now, if there are no long negative cycles then the vertex centroid of $P(G, w)$ has value 0 in the coordinate corresponding to the edge e. For simplicity, we will refer to this coordinate axis as x_e. On the other hand, if there are $K \geq 1$ long negative cycles in G then in the centroid $x_e = \frac{K}{K+3m} \geq \frac{1}{3m+1}$. This implies that having an ϵ-approximation for the centroid of $P(G, w)$ for $\epsilon < \frac{1}{2(3m+1)}$ would reveal whether or not $P(G, w)$ has a non-trivial vertex and hence whether or not G has a long negative cycle. Thus we have the following theorem:

Theorem 6 *There is no polynomial algorithm that computes a $\frac{1}{d}$-approximation of the vertex centroid of an arbitrary $\mathcal{H}$-polyhedron $P \subset \mathbb{R}^d$, unless $P = NP$.*

An immediate consequence of Theorem 4 and Theorem 6 is that there is no polynomial algorithm that computes any "sufficiently non-trivial" approximation of the vertex centroid of an arbitrary $\mathcal{H}$-polyhedron unless $P = NP$. More formally,

Corollary 1 *There is no polynomial algorithm that $d^{\frac{1}{2}-\delta}$-approximates the centroid of an arbitrary d-dimensional $\mathcal{H}$-polyhedron for any fixed constant $\delta > 0$ unless $P = NP$.*

3 Open Problems

Although we can show that for unbounded polyhedra almost any non-trivial approximation of the vertex centroid is hard, we can not make a similar statement for the bounded case (*i.e.* *polytopes*. One interesting variant of Theorem 2 would be to consider a ball of radius r instead of a halfspace. If containment of vertex centroid in a ball of radius r can be decided in time polynomial in the number of inequalities defining the polytope, the dimension and r then one can perform a sort of random walk inside the polytope and approximate the centroid in polynomial time. We leave out the details of this random walk since we do not have a method to check containment inside a ball.

References

[1] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comput. Geom.*, 7:265–301, 1997.

[2] M. E. Dyer. The complexity of vertex enumeration methods. *Mathematics of Operations Research*, 8(3):381–402, 1983.

[3] M. E. Dyer and A. M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17(5):967–974, 1988.

[4] R. Kannan, L. Lovász, and M. Simonovits. Random walks and an $o^*(n^5)$ volume algorithm for convex bodies. *Random Structures and Algorithms*, 11(1):1–50, December 1998.

[5] L. Khachiyan, E. Boros, K. Borys, K. M. Elbassioni, and V. Gurvich. Generating all vertices of a polyhedron is hard. In *SODA*, pages 758–765. ACM Press, 2006.

[6] N. Linial. Hard enumeration problems in geometry and combinatorics. *SIAM J. Algebraic Discrete Methods*, 7(2):331–335, 1986.

[7] L. Rademacher. Approximating the centroid is hard. In *Symposium on Computational Geometry*, pages 302–305, 2007.