Ellipsoid Algorithm

Siddharth Bhat
20161105

***Abstract*—We motivate the Ellipsoid algorithm, discuss its original theoretical importance, and remark on its practical efficiency.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The ellipsoid algorithm was discovered in Naum Z. Shor. Later, Leonid Genrikhovich Khachiyan proved that the algorithm runs in polynomial time. This was a breakthrough in the theory of linear programming, which proved that solving LP's is in `PTIME`.

II. High level description the algorithm: Checking for non-emptiness

The ellipsoid algorithm is an algorithm that allows us to check for the non-emptiness of a system of linear equations.

An *ellipsoid* in $\mathbb{R}^n$ is a set of points

$$E \equiv \left\{ (x_1, x_2, \dots x_n) \mid \sum_i \frac{x_i^2}{a_i^2} \leq 1 \right\}$$

for some constants $(a_1, a_2, \dots a_n)$.

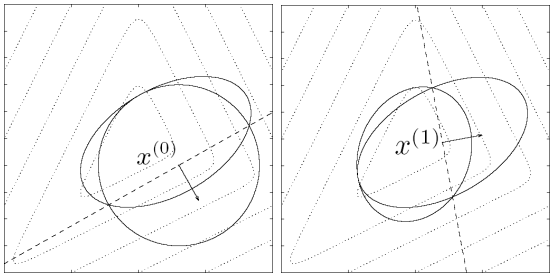
The algorithm works by first bounding the polyhedra $P \equiv \{x | Ax \leq b\}$ with a large initial bounding ellipsoid, such that $P \subset E_0$. We now pick a random point $p \in E_0$, and check if this point is in the polyhedra P. If it is, then we are done, as we have found a feasible point $p \in P$. On the other hand, if $p \notin P$, we shrink the ellipsoid E_0 to a smaller ellipsoid E_1 such that $p \notin E_1$, but $P \subseteq E_1$. That is, we shrik the ellipsoid such that the point p is no longer in the ellipsoid, but the polyhedra still is. We then repeat the process.

Now, of course, one can imagine this process going on forever, since perhaps our polyhedra P truly is empty, but we keep picking points in our ellipsoids, since our ellipsoid as described above can never shrink to the empty set.

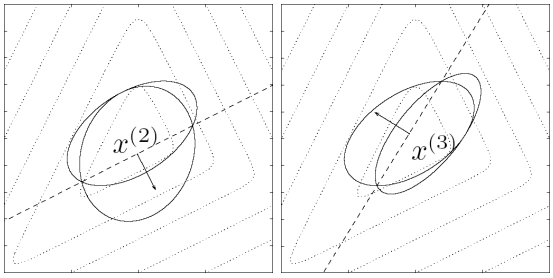
To prevent this case, we are also given a *lower bound* V_l on the volume of the polyhedra. Now, after some iteration, if the volume of the ellipsoid $Vol(E)$ goes below V_l, we can conclude that $P \not\subseteq E$. Thus, we can terminate with infeasibility.

Thus, for this algorithm to work, the main ingredients are:

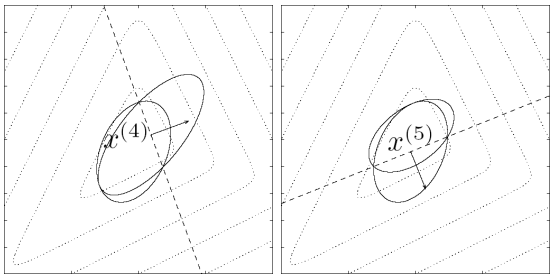
- The polyhedra $P \equiv \{x | Ax \leq b\}$ whose non-emptiness is to be tested.
- The initial ellipsoid E_0 that bounds P of volume V_u.
- The lower bound on the volume of P, V_l such that $0 < V_l \leq Vol(P)$.
- An algorithm which when given a point x and the polyhedra P, either certifies $x \in P$, or provides us information about how to separate x from P, so we can shrink our ellipsoid.



(a) iteration 0 (b) iteration 1



(c) iteration 2 (d) iteration 3



(e) iteration 4 (f) iteration 5

Fig. 1. Sequence of iteration of ellipsoid method

Given these ingredients, we show how to construct the algorithm. We show later in the report how this can be used to solve LP's in polynomial time. Here, we present a specific version of the Ellipsoid algorithm. The more general version works over convex sets, and requires a gadget known as a *separation oracle* which provides us the separation information of x from P. In the case of LP, this gadget reduces to the Farkas' Lemma. So, we shall proceed to discuss the algorithm for the LP case only.

III. Using non-emptiness to solve LP's

So far, all we can do using the ellipsoid algorithm is to check if some system of equations $Ax \leq b$ is *non-empty*. In other words, we can check the non-emptiness of a given polyhedron. Here, we will describe how to use this to solve *optimisation problems*.

Consider a linear program and its dual:

$$x, c \in \mathbb{R}^{n\times 1} \quad A \in \mathbb{R}^{m\times n} \quad b, y \in \mathbb{R}^{m\times 1}$$
$$P_{primal} \equiv \underset{x}{\text{maximise}}\ c^T x \text{ subject to } Ax = b$$
$$P_{dual} \equiv \underset{y}{\text{minimise}}\ b^T y \text{ subject to } A^T y \geq c$$

Let $x^\star$ be the optimal value of x for P_{primal}, and $y^\star$ be the optimal value of y for P_{dual}. From strong duality, we know that the value of $c^T x^\star = b^T y^\star$.

So, we can create a *combined* linear program, whose feasibility will force us to provide a point such that $c^T x = b^T y$. That is, we create a new polyhedra Q defined by the equations:

$$Ax \leq b \quad A^T y \geq c \quad c^T x = b^T y$$

Now, if a feasible point $(x_0, y_0) \in Q$, then it must be the case that $Ax_0 = b$, $A^T y_0 \geq c$, and $c^T x_0 = b^T y_0$. At this point, strong duality tells us that $(x_0, y_0) = (x^\star, y^\star)$.

Hence, we can find the optimal value of the linear program by evaluating $c^T x_0$.

Thus, the ellipsoid algorithm can be used to solve for the optimality of a linear program, by starting from a non-emptiness check! This is beautiful, and proves a deep result of LP's: A certificate of non-emptiness is as good as a ceritificate of optimality.

IV. Detailed description of the algorithm

We are given a polyhedra defined by a system of constraints $P \equiv \{x \mid Ax \leq b\}$. We are also given a bounding ellipsoid E_0 for P, and a lower bound V_l on the volume of P.

V. Proof that ellipsoid algorithm is polynomial in input size

Here, we sketch the proof that the ellipsoid algorithm is polynomial in its input size.

The minimum volume ellipsoid that surrounds a half-ellipsoid $E_i \cap H^+$ can be calculated in polynomial time, and:

$$V(E_{i+1}) \leq (1 - \frac{1}{2n})V(E_i)$$

This comes from a non-trivial lemma of convex geometry.

If the loop in the algorithm runs t times, then by the above lemma:

$$\left(1 - \frac{1}{2n}\right)^t \leq \frac{V_u}{V_l} \implies t = O(n \log \frac{V_u}{V_l})$$

VI. Code implementing the algorithm

The code is written using the SAGE mathematical system, which provides a rich library for polyhedra. This allows one to prototype geometric algorithms with very little code. The code can be run by executing it with `sage program.py`, and is presented in figure 2

A sample run is shown below:

```
Running ellipsoid on feasible polytope
* 0
trying point: (-128.66, 234.03)
polyhedra does not contain point. Shrinking.
  old radii: (500.00, 500.00)
  new radii: (128.66, 234.03)
* 1
trying point: (-0.52, 2.01)
polyhedra contains point: (-0.52, 2.01)



Running ellipsoid on empty polytope
* 0
trying point: (99.26, 105.70)
polyhedra does not contain point. Shrinking.
  old radii: (500.00, 500.00)
  new radii: (99.26, 105.70)
* 1
trying point: (-11.34, -47.03)
polyhedra does not contain point. Shrinking.
  old radii: (99.26, 105.70)
  new radii: (11.34, 47.03)
* 2
trying point: (1.27, 9.62)
polyhedra does not contain point. Shrinking.
  old radii: (11.34, 47.03)
  new radii: (1.27, 9.62)
* 3
trying point: (-0.82, -5.12)
polyhedra does not contain point. Shrinking.
  old radii: (1.27, 9.62)
  new radii: (0.82, 5.12)
* 4
trying point: (-0.44, -1.72)
polyhedra does not contain point. Shrinking.
  old radii: (0.82, 5.12)
  new radii: (0.44, 1.72)
* 5
trying point: (0.33, -0.42)
polyhedra does not contain point. Shrinking.
  old radii: (0.44, 1.72)
  new radii: (0.33, 0.42)
* 6
trying point: (-0.02, 0.39)
polyhedra does not contain point. Shrinking.
  old radii: (0.33, 0.42)
  new radii: (0.02, 0.39)
V bounding box(0.01) < min V (0.10)
```

```
def ellipsoid(poly, minvolume):

    # define 2D ellipse with radii (500, 500)
    # The initial radii is picked based on a notion of descriptive complexity
    # of the polyhedra.
    (ax, ay) = (500, 500)

    i = 0
    while True:
        # pick a random point on the unit disk (0, 0). This distribution is not
        # truly uniform, but it works in a pinch
        randr = random()
        randtheta = randrange(-100, 100) / 100.0 * 2.0 * pi

        # transform point to point in ellipse
        randx = randr * cos(randtheta) * ax
        randy = randr * sin(randtheta) * ay
        randp = (randx, randy)

        print("* %d " % (i, ))
        i += 1

        print("trying point: (%4.2f, %4.2f)" % (randx, randy))

        # now we have a random point in ellipse, check if the point is
        # in the polytope. Return the point if it does
        if poly.contains(randp):
            print("polyhedra contains point: (%4.2f, %4.2f)" % (randx, randy))
            return randp
        else:
            print("polyhedra does not contain point. Shrinking.")
            print("  old radii: (%4.2f, %4.2f)" % (ax, ay))
            # we do not have the point in the ellipse. Shorten ellipse
            # based on this info.
            ax = min(ax, abs(randx))
            ay = min(ay, abs(randy))
            print("  new radii: (%4.2f, %4.2f)" % (ax, ay))

            # compute overapproximation of volume (bounding box)
            # if this is less than the minimum volume, return. Otherwise,
            # continue
            if ax * ay < minvolume:
                print("V bounding box(%4.2f) < min V (%4.2f)" %
                      (ax * ay, minvolume))
                return None


# Run ellipsoid  on feasible polytope
print("Running ellipsoid on feasible polytope")
p = polytopes.regular_polygon(5) * 10
ellipsoid(p, 0.1)


# Run ellipsoid on infeasible (empty) polytope
print(" ")
print("Running ellipsoid on empty polytope")
p = Polyhedron()
ellipsoid(p, 0.1)
```

Fig. 2. Implementation of the ellipsoid algorithm in SAGE