Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Algorithms Lab HS16
Department of Computer Science
Prof. Dr. A. Steger, Prof. Dr. E. Welzl, Prof. Dr. P. Widmayer
www.cadmo.ethz.ch/education/lectures/HS16/algolab

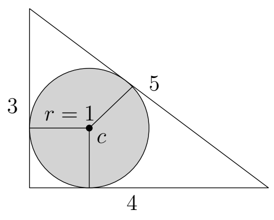
Solution — Inball

1 The problem in a nutshell

Given a cave, described by a set of linear inequalities in d dimensional space, find the maximum integral radius of a d dimensional ball that fits in the cave. A cave is described as:

$$C = \{\mathsf{x} \in \mathbb{R}^d \mid \mathsf{a}_i^T \mathsf{x} \leq b_i \,, i = 1, \dots, n\}.$$

$$-\mathbf{x} \leq 0$$
$$-\mathbf{y} \leq 0$$
$$3\mathbf{x} + 4\mathbf{y} \leq 12$$



The input to our problem is a set of linear inequalities, like on the left side of the figure above. We have to find what is the radius of a largest ball that can fit in this cave. On the right side we see a graphical description of the same triangular cave, with the lengths of the sides. The largest ball that fits in that cave has radius $r = 1$ and the center of the ball is at $c = (1, 1)$.

2 Modeling

This problem is clearly geometric and, thus, CGAL is a natural choice. But what exactly does it mean to compute the largest ball that can fit in a given cave? Or, maybe, we should first establish what it means that a ball can fit in the cave.

A cave is a set of linear inequalities. Consider an inequality $\mathsf{a}^T\mathsf{x} \leq b$ and the halfspace it defines $\{\mathsf{x} \mid \mathsf{a}^T\mathsf{x} \leq b\}$. Let H be the hyperplane that defines the halfspace at hand, i.e. $H = \{\mathsf{x} \mid \mathsf{a}^T\mathsf{x} = b\}$. The normal vector to this hyperplane is a and the normal unit vector is $\frac{\mathsf{a}}{\|\mathsf{a}\|_2}$, where $\|\mathsf{a}\|_2 = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$ is the Euclidean norm. Assume we have a ball of radius r and center c. When is it in the halfspace? Let us try to show this pictorially:

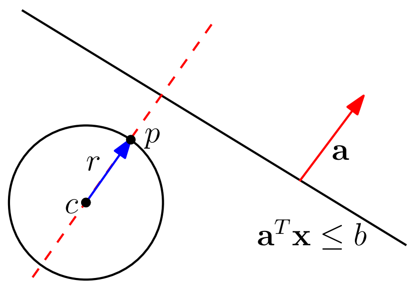


Figure 1: In this figure we have a 2-dimensional example. The line defines a halfplane. A vector a that is normal to this line is depicted in red. The ball has center c and radius r. The red dashed line goes through c and is parallel to the normal vector.

So, the point p is at distance r from c in a direction that is normal to the hyperplane H. Formally:

$$p = \mathbf{c} + r\frac{\mathbf{a}}{||\mathbf{a}||_2}.$$

The point p is contained in the halfspace $\{x | a^T x \leq b\}$ if and only if

$$\mathbf{a}^T(\mathbf{c} + r\frac{\mathbf{a}}{||\mathbf{a}||_2}) \leq b$$

which can be simplified as

$$\mathbf{a}^T p = \mathbf{a}^T \mathbf{c} + ||\mathbf{a}||_2 r \leq b \tag{1}$$

This is a linear inequality. A ball will fit in the cave if it satisfies every such inequality. At the same time we want to maximize the radius of the ball. It is clear now that the problem should be solved with Linear Programming.

3 Formulating the Linear Program

So, we have decided to go with Linear Programming. In addition, we have already expressed with (1) our linear constraints. We have one such inequality per wall of the cave. However, the center of the ball is not fixed; it is, actually, a variable in the problem we are trying to solve.

Therefore, we consider (1) but now instead of having $\mathbf{c}$ in the inequalities (which implies a fixed center) we write $\mathbf{x}$ to indicate that this is variable. Consider the following Linear Program:

$$\begin{array}{ll} \text{maximize} & r \\ \text{subject to} & \mathbf{a}_i^T \mathbf{x} + ||\mathbf{a}_i|| r \leq b_i, \quad i = 1, \dots, n \end{array}$$

What does it do? It maximizes r subject to the constraint that there is an $\mathbf{x}$ such that distance from $\mathbf{x}$ to each hyperplane is at least r, i.e. the ball with center $\mathbf{x}$ and radius r fits in the cave. There is three possible outcomes:

(i) The input constraints define a bounded polytope and thus there is an optimal value to be reported.

(ii) The input constraints define an unbounded polytope (i.e. the cave is open). In this case an arbitrarily large ball can be fit in the cave.

(iii) The input constraints define an empty set.

The CGAL LP solver can tell us which of the three outcomes we have.

Note that for this problem we do not have much choice for algorithm design. Also, by the problem description there is not really different sets of points awarded for different testsets. Thus, we are only asked here to set up the right Linear Program and feed it to the solver.

4 Implementation

Here are a few remarks about the implementation.

- By the problem description we have $|(\mathbf{a}_i)_j|, |b_i| \leq 2^{10}$, for every i, j. In addition, it is guaranteed that $||\mathbf{a}_i||_2$ is an integer, for every i. Then, all the coefficients of our LP inequalities can be described by the type `int`. This is because

$$||a_i||_2^2 = \sum_{j=1}^{d} (a_i)_j^2 \leq d \cdot (2^{10})^2 = 10 \cdot 2^{20} < 2^{24}$$

and we assume that the implementation of `int` on the judge is 32bit. Actually, note that only $||a_i||_2$ will be a coefficient (which, of course, is even smaller in bitsize). Therefore, we define the type of our LP as `typedef CGAL::Quadratic_program<int>`. Here, `double` would also work without any problem with the time limits.

- The constraint inequalities for the LP have the $\leq$ relationship. In addition, remember that the variables are the position of the center of the ball and the size of the radius. We do not want to impose any lower or upper bounds for the position of the center. Hence, we call the LP constructor as `Program lp(CGAL::SMALLER, false, 0, false, 0)`.

- A general reminder: The CGAL LP solver can only minimize. So, in order to maximize r we will actually minimize $-r$. In the solution we attach below r is the $(d+1)$th variable, i.e. it has index d. So, we set its coefficient in the objective function to be -1 with `lp.set_c(d, -1)`. In addition, we add a lower bound of 0 for r because any solution with $r < 0$ is irrelevant to us.

- We want to round down the radius to the closest integer. Let s be the variable that holds the solution that the solver returns. The function `s.objective_value()` returns an object of type `CGAL::Quotient<ET>` where `ET` is the exact type. You can experiment and see that the Quotient, even in the case of `Gmpz`, is not necessarily simplified. In order to round it down, we just have to divide the numerator by the denominator using integer division. We can directly output the exact type because it is integral. Please refer to line 47 of the complete solution in the next page.

5 A Complete Solution

```cpp
#include <iostream>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpz.h>

typedef int IT; //Input Type
typedef CGAL::Gmpz ET; //Exact Type
typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main () {
  std::cin.sync_with_stdio(false);

  int n;
  for (std::cin >> n; n > 0; std::cin >> n) {
    int d;
    std::cin >> d;
    Program lp(CGAL::SMALLER, false, 0, false, 0);
    for (int i = 0; i < n; ++i) {
      double norm2 = 0;
      for (int j = 0; j < d; ++j) {
        int ai;
        std::cin >> ai;
        norm2 += ai*ai;
        lp.set_a(j, i, ai);
      }

      //Check that the norm is indeed an integer
      double norm = std::floor(std::sqrt(norm2));
      if (norm2 != norm*norm)
        throw std::domain_error("Warning:␣norm2!=␣norm*norm.\n");

      lp.set_a(d, i, norm);
      int bi;
      std::cin >> bi;
      lp.set_b(i, bi);
    }
    lp.set_c(d, -1);
    lp.set_l(d, true, 0);

    Solution s = CGAL::solve_linear_program(lp, ET());
    if (s.is_infeasible())
      std::cout << "none\n";
    else if (s.is_unbounded())
      std::cout << "inf\n";
    else {
      ET out = -(s.objective_value().numerator()/s.objective_value().denominator());
      std::cout << out << std::endl;
    }
  }
  return 0;
}
```