САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Евменов Владимир Андреевич

Выпускная квалификационная работа

Алгоритмы оптимального покрытия выпуклых плоских областей изотетичными многоугольниками с учетом ограничений на расхождение площадей

Бакалавриат
Направление 01.03.01 Математика
Основная образовательная программа СВ.5000 Математика

Научный руководитель:
Вяткина Кира Вадимовна, к.ф-м.н.
зав. кафедрой биоинформатики и
математической биологии
СПбАУ РАН им. Ж.И. Алфёрова

Рецензент:
Степанов Евгений Олегович, д.ф.-м.н.,
с.н.с. ПОМИ им В.А. Стеклова РАН

Санкт-Петербург
2022

Содержание

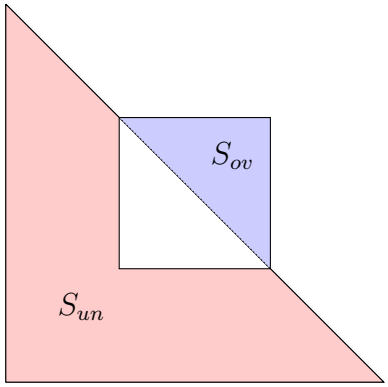
1 Введение

Задача о покрытии фигур привлекла внимание исследователей уже давно [1] и рассматривалась во множестве различных постановок. В данной работе будет рассмотрена очередная ее постановка, имеющая практическое значение с точки зрения промышленных приложений.

Вначале будет проанализирован частный ее случай, а именно задача об оптимальном покрытии плоской выпуклой фигуры прямоугольниками, имеющими одинаковую ориентацию. Далее будет представлен приближенный алгоритм поиска оптимального покрытия. В заключение будет представлена оценка сложности и обсуждены детали программной реализации предложенных подходов.

2 Постановка задачи

Определение 1. Функция ошибки покрытия $\Delta = \frac{S_{un}+S_{ov}}{S_{area}}$



S_{ov} - соотвествует красной зоне, т.е не покрытая область, S_{un} - покрытая область не принадлежащая покрываемой фигуре. S_{area} - площадь покрываемой фигуры

В этой работе будет решаться задача поиска оптимального покрытия при данной оценке ошибки для фиксированного числа прямоугольников.

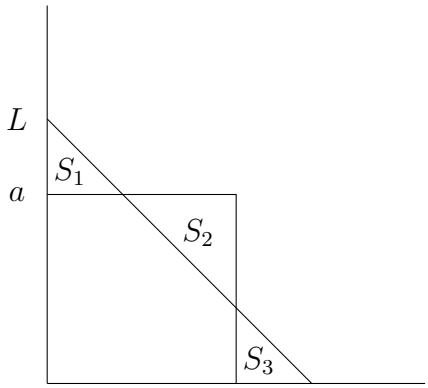
3 Обзор результатов

3.1 Случай покрытия треугольников

Сначала рассмотрим случай треугольника, катеты которого парралельны осям координат.

Определение 2. Обозначим через n число прямоугольников в покрытии.

- $n = 1$, равнобедренный прямоугольный треугольник.



Начнем с покрытия одним прямоугольником.

Левая нижняя вершина совпадает с вершиной треугольника. Если это не так, можно привести прямоугольник к такому виду без потери общности. Очевидно, прямоугольник - квадрат в силу симметрии. Таким образом, функция ошибки:

$$f_{err}(a) = \frac{S_1 + S_2 + S_3}{S_{area}} = \frac{(L-a)^2 + 0.5 \cdot (2a - L)^2}{\frac{1}{2}L^2} \longrightarrow min$$

Где L - длина стороны прямоугольника. А a - длина стороны квадрата. Данную функцию требуется минимизировать по a.

Несложно понять, что минимум данного выражение достигается в точке $a = \frac{2L}{3}$, а значение ошибки $\Delta = \frac{1}{3}$.

Заметим также, что значения выражений S_1, S_2, S_3 в точке минимума $S_1 = 0.5 \cdot (L - a)^2 = 0.5 \cdot \frac{1}{9}L^2 = 0.5 \cdot (\frac{4}{3}L - L)^2 = 0.5 \cdot (2a - L)^2 = S_2$ т.е $S_1 = S_2 = S_3$

- $n = 1$, прямоугольный треугольник.

Посмотрим, что происходит когда треугольник не равнобедренный.

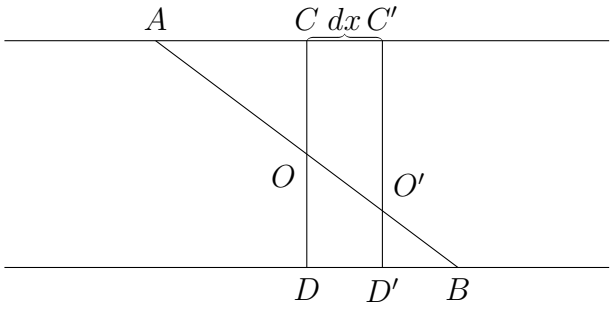
Формализуем идею из замечания к предыдущей подзадаче о равентсве площадей.

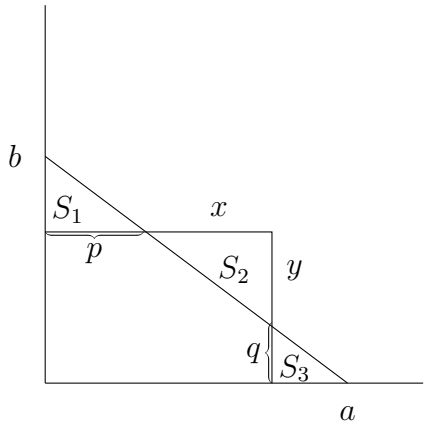
Лемма 1. *Имеется 2 парралельные прямые, и прямая их пересекающая. Обозначим точки пересечения через A и B. Точку O отрезка AB пересекает прямая перпендикулярная исходным прямым. Соответствующие прямые она пересекает в точках C и D. Тогда минимум площадей $S_{AOC} + S_{BOD}$ достигается тогда, когда O - середина AB.*

Доказательство. Расположим отрезок CD так, чтобы O была серединой AB. Теперь сдвинем отрезок налево или направо на вектор dx. В таком случае сумма площадей изменится на велечину

$$S_{OCC'O'} - S_{ODD'O'}$$

В силу того, что $|DO| > |DO'|$ эта разница положительна. □





Рассуждения аналогичны, но обозначения изменены. a и b теперь обозначают длины сторон. Причем, $a > b$. x и y - длины соответсвующих сторон прямоугольника.

$$f_{err}(a) = \frac{S_1 + S_2 + S_3}{S_{area}} \longrightarrow min$$

Таким образом, если зафиксировать x, тогда S_3 - константа. В таком случае надо минимизировать $S_1 + S_2$. Аналогично, зафиксировав y, и минимизировать $S_2 + S_3$. Надо проверить оба этих случая и выбрать минимальный их них. В силу леммы, оптимальное решение гарантирует равенство площадей S_1, S_2 или S_2, S_3.

При фиксированном y,

$$f_{err}(a) = \frac{(\frac{y}{2})^2 \frac{a}{b} + (\frac{y}{2})^2 \frac{a}{b} + (b-y)^2 \frac{a}{b}}{ab} \to min$$

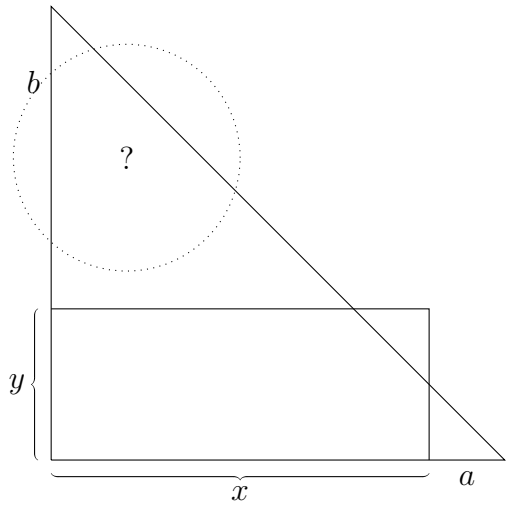
Данное выражение достигает минимума при значениях $y = \frac{2}{3}b$ и $\Delta = \frac{1}{3}$.

Теперь тоже самое сделаем и с x,

$$f_{err}(a) = \frac{(\frac{x}{2})^2 \frac{b}{a} + (\frac{x}{2})^2 \frac{b}{a} + (a-x)^2 \frac{b}{a}}{ab} \to min$$

Это и предыдущее выражение одинаковы с точностью до замене, следовательно и результат тот же. Из первого выражения получим $x = \frac{2}{3}a$, что совпадает с результатом оптимизации второго выражения Т.е мы получили $S_1 = S_2 = S_3$.

- $n \geq 2$, прямоугольный треугольник.



Давайте рассмотрим тот прямоугольник, который покрывает стык катетов треугольника. Тот треугольник, что остается сверху(или сбоку), это треугольник, который надо покрыть на 1 меньшим количеством прямоугольников.

В таком случае нам требуется найти сумму минимума ошибок от выбранного нами прямоугольника и того, что осталось покрыть.

Пусть Δ_n - оптимальная ошибка покрытия с n прямоугольниками. Тогда:

$$\Delta_n = \frac{S_1 + ... + S_{2n+1}}{S_{area}} = \frac{S_1 + ... + S_{2n-1}}{S_{area}} + \frac{S_{2n} + S_{2n+1}}{S_{area}} =$$
$$= \frac{S_1 + ... + S_{2n-1}}{S_{area}} \cdot \frac{S_{up}}{S_{up}} + \frac{S_{2n} + S_{2n+1}}{S_{area}} = \Delta_{n-1} \cdot \frac{S_{up}}{S_{area}} + \frac{S_{2n} + S_{2n+1}}{S_{area}}$$

Где S_{up} - площадь треугольника отсеченного прямоугольником. Из прядыдущей леммы ясно, как устроены площади S_{2n}, S_{2n+1}

Т.е

$$\Delta_n = \Delta_{n-1} \cdot \frac{(b-y)^2}{b^2} + \frac{2(\frac{y}{2})^2 \frac{a}{b}}{ab}$$

или

$$\Delta_n = \Delta_{n-1} \cdot \frac{(a-x)^2}{a^2} + \frac{2(\frac{x}{2})^2 \frac{b}{a}}{ab}$$

Будем выводить формулу и одновременно доказывать по индукции, что Δ_n - не зависит от a, b.

Мы уже доказали это для $n = 1$.

Теперь остается понять, чему равно:

$$\Delta_n = \Delta_{n-1} \cdot \frac{(b-y)^2}{b^2} + \frac{2(\frac{y}{2})^2 \frac{a}{b}}{ab} \to min$$

Несложно понять, что

$$\Delta_n = \frac{2\Delta_{n-1}}{\Delta_{n-1}+2}$$

$$y = \frac{b\Delta_{n-1}}{\Delta_{n-1}+2}$$

$$x = a - \frac{y}{2}\cdot\frac{a}{b}$$

Теперь оценим сходимоть Δ_n

$$\lim_{n\to\infty}\Delta_n = \lim_{n\to\infty}\frac{2\Delta_{n-1}}{\Delta_{n-1}+2}$$

Тогда, заменив Δ_{n-1} на Δ_n приходим к выражению

$$\Delta_n = \frac{2\Delta_n}{\Delta_n+2}$$

отсюда

$$\lim_{n\to\infty}\Delta_n = 0$$

Данная замена возможна, действительно в силу монотонного убывания и ограниченности снизу Δ_n по Т.Вейерштрасса существует предел. А также

$$\forall\epsilon > 0 \exists n_0 : \forall n > n_0 |\lim_{n\to\infty}\Delta_n - \Delta_n| < \epsilon$$

Так как $\Delta_n > 0$ и монотонно убывают то

$$\exists n_0, n_1 : |\Delta_{n_0} - \Delta_{n_1}| < \epsilon n_0 < n_1$$

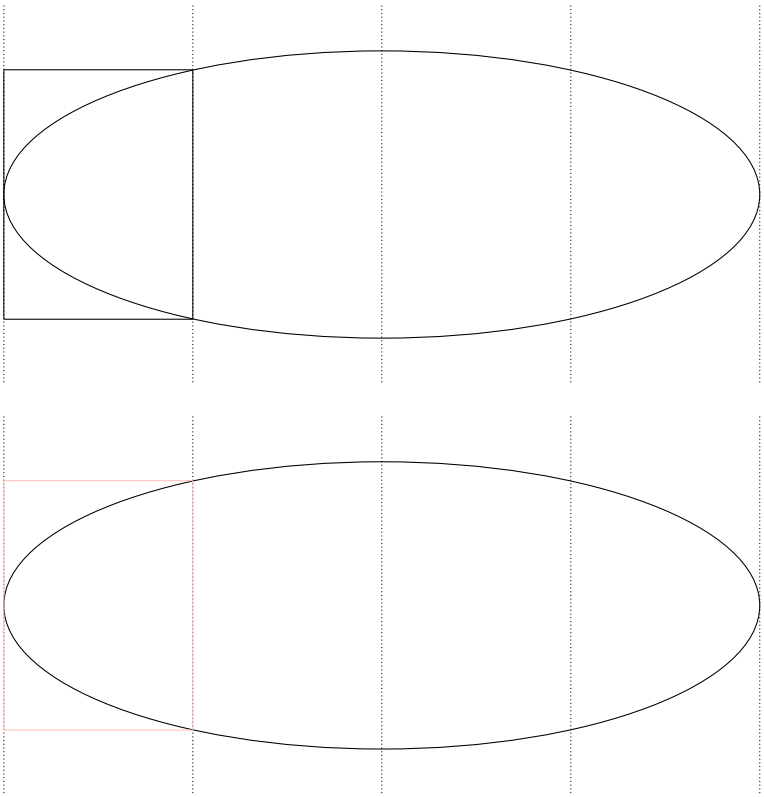
в следствие

$$|\Delta_{n_0} - \Delta_{n_1}| + |\lim_{n\to\infty}\Delta_n - \Delta_{n_1}| = |\lim_{n\to\infty}\Delta_n - \Delta_{n_0}| < \epsilon$$

4 Алгоритм и оценка

Алгоритм следующий. Взять самую левую и самую правые точки контура фигуры. x_l, x_r. Не умоляя общности можем считать, что высота фигуры больше, чем ее ширина. Равномерное разделим отрезок $[x_l, x_r]$ на n частей. Для каждого из отрезков возьмем самою нижнюю и самую верхнюю точку. Теперь проведем через самую левую, правую, нижнюю, верхнюю точки прямоугольник.

Теперь оценим ошибку. Достаточно рассмотреть нижний контур фигуры. Этот контур - выпуклая вниз функция. Значит у нее есть точка экстремума, причем ровно одна. Соответсвенно есть 2 случая:



Достаточно оценить S_{ov}, т.к в нашем случае покрытие - накрытие.Не умоляя общности можем считать, что $|[x_l, x_r]| = 1$. Таким образом отрезок делится на отрезки длины $\frac{1}{n}$. В первом случае:

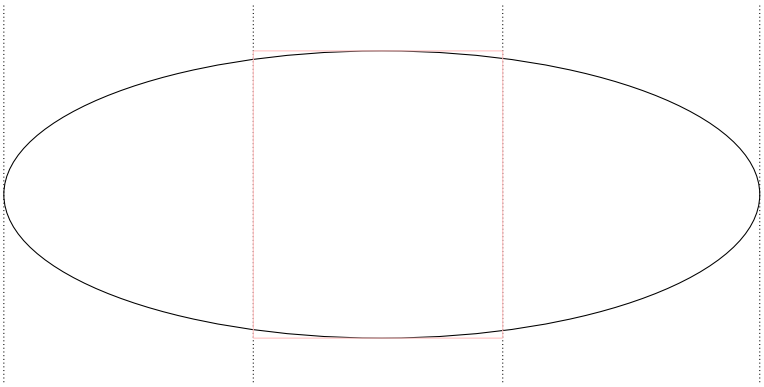
$$\delta \leq \frac{1}{n} \cdot r \cdot \frac{1}{2}$$

где r длина контура фигуры лежащего в прямоугольнике. Случай с точкой экстремума сводится делением отрезка по этой точке.

Таким образом,

$$\Delta \leq \sum_{i=0}^{n} \frac{1}{n} \cdot r_i \cdot \frac{1}{2}$$

Несложно понять, что $\sum_i^n r_i$ - периметр низа фигуры. Т.к фигура лежит в квадрате со стороной не более 1, и она выпукла, ее периметр не больше 4. Таким образом, мы получили оценки к 0 со скоростью $O(\frac{1}{n})$



5 Реализация

Входные данные

Выпуклая фигура описана множеством вершин. В первой строке записано число n, число вершин фигуры, далле идут n строк, $i-$ строка описывает координаты вершины (x_i, y_i). Вершины заданы в порядке обхода по часовой стрелке.

Выходные данные

n строк содержащих описание координат прямоугольников в нотации левой-нижней и правой-верхней вершины. В $n+1$ строке находится значение функции ошибки найденного покрытия.

Техническая реализация

```
// exe build command
// g++ -O2 -Wall -Wl,--stack=268435456 !.! -o !.exe

#include <fstream>
#include <sstream>
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <random>

using namespace std;

constexpr char* FILEPATH = "hex.in";

struct Point
{
double x;
```

```
double y;

Point(double x_n = 0, double y_n = 0)
{
x = x_n;
y = y_n;
}
};

struct Vector
{
double x;
double y;

Vector(Point a, Point b)
{
x = b.x - a.x;
y = b.y - a.y;
}

Vector(double x_n = 0, double y_n = 0)
{
x = x_n;
y = y_n;
}
};

double operator * (const Vector &a, const Vector &b)
{
return a.x * b.y - a.y * b.x;
}

double operator % (const Vector &a, const Vector &b)
{
return a.x * b.x + a.y * b.y;
}


inline bool cmp(const Point a, const Point b)
{
return a.x < b.x;
}
```

```
vector<Point> read_figure_from_file(const char* filepath = FILEPATH)
{
vector<Point> figure; // clockwise point order

ifstream in(filepath);
string s;
getline(in, s);

stringstream ns(s);

int n;
ns >> n;

for (int i = 0; i < n; i++)
{
getline(in, s);
stringstream ps(s);

double x, y;
ps >> x >> y;
figure.push_back(Point(x, y));
}

return figure;
}

double x_to_y(double x, const vector<Point> &v)
{
for (int i = 0; i < v.size() - 1; i++)
if (x >= v[i].x && x <= v[i + 1].x)
{
Point l = v[i], r = v[i + 1];
return min(l.y, r.y) + (x - l.x) / (r.x - l.x) * abs(r.y - l.y);
}
}

Point ternar_search(double l, double r, const vector<Point> &v, bool is_min)
{
double eps = 0.0005;
int c = (is_min ? +1 : -1);
```

```
while (r - l > eps)
{
double m_1 = l + (r - l) / 3;
double m_2 = r - (r - l) / 3;

if (c * x_to_y(m_1, v) < c * x_to_y(m_2, v))
r = m_2;
else
l = m_1;
}

return Point(l, x_to_y(l, v));
}

vector<Point> get_chain(const vector<Point> &v, bool is_lower = true)
{
int i;
int k = (is_lower ? +1 : -1);
vector<Point> chain;

Point l = v[0], r = v[0];
int l_id = 0, r_id = 0;

for (i = 0; i < v.size(); i++)
{
Point c = v[i];

if (l.x > c.x || (l.x == c.x && k * l.y > k * c.y))
{
l = c;
l_id = i;
}

if (r.x < c.x || (r.x == c.x && k * r.y > k * c.y))
{
r = c;
r_id = i;
}
}


i = r_id;
```

```
while (i != l_id)
{
chain.push_back(Point(v[i].x, k * v[i].y));
i = (i + k + v.size()) % v.size();
}
chain.push_back(Point(v[l_id].x, k * v[l_id].y));

sort(chain.begin(), chain.end(), cmp);

return chain;
}

double area(const vector<Point> &v)
{
int n = v.size();
Point c(0, 0);
double area = 0;

for (int i = 0; i < n; i++)
{
Vector a(c, v[i]);
Vector b(c, v[(i + 1) % n]);

area += (a * b) / 2;
}

return abs(area);
}

bool is_in(const vector<Point> &v, Point c)
{
int n = v.size();
double angle = 0;

for (int i = 0; i < n; i++)
{
Vector a(c, v[i]);
Vector b(c, v[(i + 1) % n]);

angle += atan2((a * b), (a % b));
}
```

```
return (abs(angle) > 2 ? true : false);
}

double monte_carlo(Point l, Point r, const vector<Point> &v)
{
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<> dist(0, 1000000);

int all = 0, cnt = 0;
for (int i = 0; i < 10000; i++)
{
double x = (dist(gen) * 1.0 / 1000000) * (r.x - l.x) + l.x;
double y = (dist(gen) * 1.0 / 1000000) * (r.y - l.y) + l.y;

cnt += (int)(is_in(v, Point(x, y)));
all += 1;
}

return 1.0 * cnt / all;
}

void algo(const vector<Point> &v, int n)
{
auto lower_chain = get_chain(v, true);
auto upper_chain = get_chain(v, false);



double l = lower_chain[0].x, r = lower_chain[lower_chain.size() - 1].x;
vector<double> cuts;
for (int i = 0; i < n + 1; i++)
{
cuts.push_back(l + (r - l) * i / n);
}

double error = 0;
for (int i = 0; i < n; i++)
{
l = cuts[i];
r = cuts[i + 1];
```

```
Point a(l, ternar_search(l, r, lower_chain, true).y);
Point b(r, -1 * ternar_search(l, r, upper_chain, true).y);

cout << "rectangular #" << i + 1 << " with coordinates : (" << a.x << ", " << a.y <<
    ") - (" << b.x << ", " << b.y << ")" << endl;

error += (1 - monte_carlo(a, b, v));
}

cout << "error function value is : " << error / area(v) << endl;
}

main()
{
vector<int> samples = {5};
auto v = read_figure_from_file();

for (int i = 0; i < samples.size(); i++)
{
algo(v, samples[i]);
}
}
```

Оценка сложности реализации

Сложность складывается из:

- Нахождение цепей - проход по массиву и сортировка - $O(n\log(n))$
- Нахождение точек покрывающих прямоугольников - проход по массиву и тернарный поиск - $O(c_1 n)$
- Вычисление ошибки - метод Монте-Карло и проверка на вхождение точки в фигуру - $O(c_2 n)$

Таким образом, итоговая сложность

$$O((c_1 + c_2)n^2)$$

где, c_1 - константа, отображающая скорость сходимости тернарного поиска. Ее можно задать метапараметром. c_2 - константа, являющаяся числом итераций метода Монте-Карло. На практике $c_1 << c_2$. Т.е c_2 влияет на время работы сильнее чем c_2.

Данную оценку можно улучшить до

$$O((c_1 + c_2)n\log(n))$$

Оптимизировав метод проверки на вхождение в выпуклую фигуру в функции monte-carlo до $O(\log(n))$, и тернарый поиск, в функции x-to-y заменив поиск точки, соответсвующей отрезку на бинарный поиск.

6 Заключение

В данной работе был подробно разобран и аналитически описан случай покрытия треугольника, катеты которого парралельны осям координат. А также, приведен универсальный алгоритм покрытия произвольной выпуклой фигуры, как и его техническая реализация.

Список литературы

[1] Paul Franklin, Optimal Rectangle Covers for Convex Rectilinear Polygons, B.Sc., Simon Fraser University, 1984.

[2] Nicholas Metropolis and S. Ulam. The Monte-Carlo method. Journal of the American Statistical Association Vol. 44, No. 247 (Sep., 1949), pp. 335-341

[3] Nitin Arora, Mamta Martolia Arora, Esha Arora. A Novel Ternary Search Algorithm. International Journal of Computer Applications (0975 – 8887) Volume 144 – No.11, June 2016