Прямокутник найбільшої площі вписаний в опуклу оболонку

Лабораторна робота
Студента 3 року навчання
Групи МІ-31
Гришечкіна Тихона Сергійовича

Київ, 2024

Зміст

Анотація

У лабораторній було досліджено проблему знаходження прямокутника найбільшої площі, вписаного в опуклу оболонку. Був наведений опис алгоритму, запропонований Бехрузі у 2019 [1], а також виконано приклад реалізації алгоритму мовою програмування C#.

Annotation

The problem of finding a rectangle of the largest area inscribed in a convex hull was studied in the lab. An algorithm proposed by Behrouzi in 2019 is described [1], and an example of the algorithm implementation in the C# programming language is given.

1 Вступ

Задача пошуку найбільшого вписаного прямокутника є важливою задачею у багатьох галузях, таких як комп'ютерна графіка, робототехніка та обробка зображень. Проблема знаходження найбільшого вписаного прямокутника (НВП) добре досліджена багатьма науковцями. В цілому зазвичай виділяють дві задачі знаходження НВП: задача знаходження НВП зі сторонами паралельними осям координат та загальна.

Наприклад, у праці Деніелса, Міленковича та Рота [2] досліджено методи та алгоритми для прямокутників зі сторонами паралельними осям координат для різних типів многокутників. А в праці Альта [3] описано алгоритм для цієї ж задачі у випадку опуклого многокутника «методом дотичних» зі складністю $O(\log(n))$. Це є найкращим результатом для випадку опуклого многокутника, який цікавить нас в рамках цієї лабораторної, що я знайшов у відкритому доступі.

В цій лабораторній буде описано і реалізовано алгоритм Бехрузі [1] для знаходження НВП зі сторонами паралельними осям координат у випадку опуклого многокутника, що має складність $O(n)$. Але важливо зазначити, що у статті Бехрузі показано, що можна перейти до алгоритму знаходження загального НВП у випадку опуклого многокутника (без обмеження на паралельність осям координат), зі складністю $O(\frac{n}{\epsilon})$, де ϵ - відносна похибка площі отриманого прямокутника від оптимального.

Важливо зазначити, що хоч і в рамках цієї лабораторної потрібне знаходження опуклої оболонки [4], алгоритми її знаходження розглядатися не будуть, а у реалізації буде використано «алгоритм Грехема».

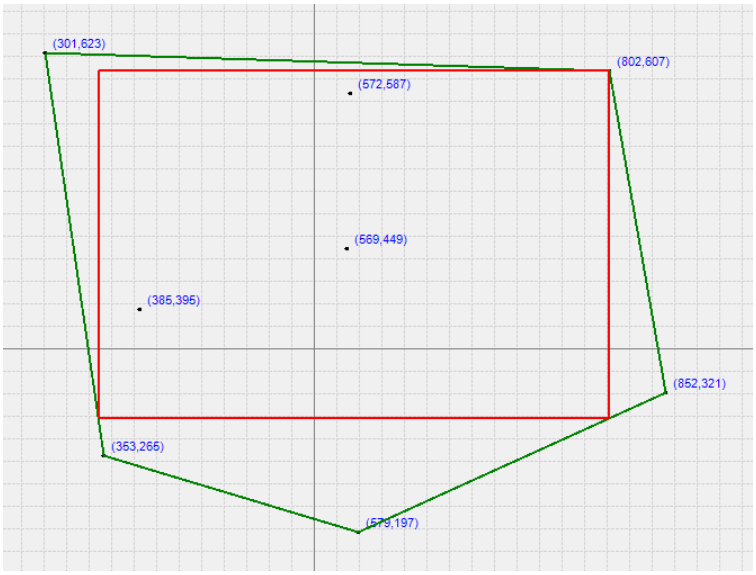

Рис. 1: Приклад опуклої оболонки множини точок, з вписаним прямокутником найбільшої площі

2 Основна частина

2.1 Загальні положення

Розглянемо довільний опуклий многокутник F розміру n. Тоді можемо сформувати критерій належності точки многокутнику F, за допомогою системи з n нерівностей. Для цього проведемо через кожні дві сусідні точки многокутника пряму, та обмежимо за допомогою рівняння цієї прямої точки, які попадають у многокутник. Для наочності рисунок[2].

Врешті-решт, будемо мати:

$$\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ \vdots & \vdots \\ p_{n1} & p_{n2} \end{pmatrix}$$

де P - матриця розміру $n \times 2$ (коефіцієнти обмежуючих прямих), та вектор

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

розміру n (обмежуючі коефіцієнти).

Тоді критерієм належності точки **x** многокутнику буде нерівність

$$Px \leq b.$$

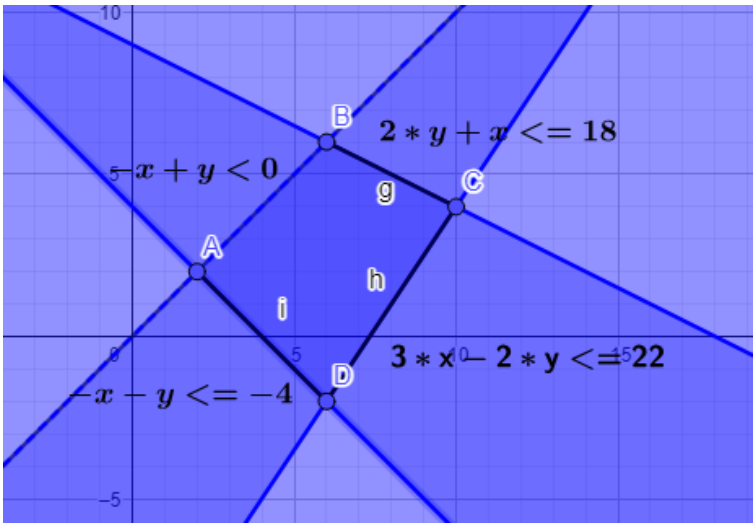

Рис. 2: Приклад многокутника $ABCD$, з обмежуючими його нерівностями

Тепер нехай для довільного багатокутника F, маємо довільні три точки x, y, z, та вектори $(\vec{u} = y - x, \vec{v} = z - x)$. Рисунок 3 для прикладу. Тоді рішенням знаходження найбільшого вписаного прямокутника, буде рішення наступної задачі оптимізації:

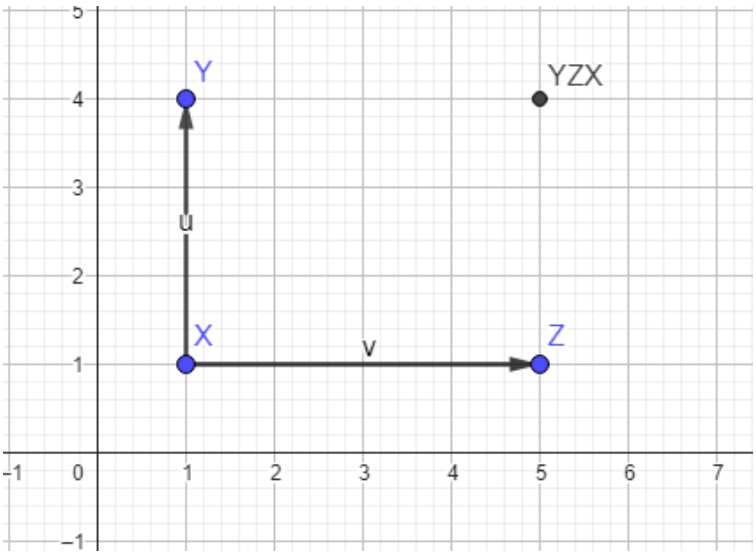

Рис. 3: Приклад точок x, y, z та векторів $(\vec{u}, \vec{v})$

$$\max |u_1 v_2 - u_2 v_1| \tag{1}$$

За умов:

$$u_1 v_1 + u_2 v_2 = 0 \tag{2}$$

$$(x, y, z, y + z - x) \subseteq F$$

Де формула 1 - максимізація площі шуканих прямокутників. Формула 2 - умова ортогональності векторів u та v.

2.2 Опис алгоритму

Виходячи з положень, що були описані в попередньому розділі, можемо сформулювати алгоритм для випадку прямокутників зі сторонами паралельними осям координат.

Припустимо F деякий опуклий многокутник. Тоді наша задача може бути описана як наступна:

$$\max \log u_1 + \log v_2 \tag{3}$$

За умов:

$$u_2 = 0$$

$$v_1 = 0$$

$$u - y + x = 0$$

$$v - z + x = 0$$

$$Px \leq b$$

$$Py \leq b$$

$$Pz \leq b$$
$$P(y + z - x) \leq b$$

де $P \in \mathbb{R}^{n \times 2}$ та $b \in \mathbb{R}^n$ є заданими характеристиками опуклого многокутника F.
Можемо переписати задачу оптимізації 3, в матричній формі.
Позначимо $s = (u^T, v^T, x^T, y^T, z^T)^T$, що собою являє вектор розміром (10×1).
Тоді маємо задачу оптимізації:

$$\min -(\log e_1^T s + \log e_4^T s)$$
$$As = 0$$
$$\tilde{P}s \leq b$$

Звідси e_i - це i-й стовпець 10×10 одиничної матриці.
$\tilde{b} = (b_1, b_1, b_1, b_1, b_2, b_2, b_2, b_2, ..., b_n, b_n, b_n, b_n)$, тобто вектор розміром $4n$, і A це

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix}$$

$$\tilde{P} = \begin{pmatrix} 0 & 0 & 0 & 0 & p_{11} & p_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{11} & p_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{11} & p_{12} \\ 0 & 0 & 0 & 0 & -p_{11} & -p_{12} & p_{11} & p_{12} & p_{11} & p_{12} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & p_{n1} & p_{n2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{n1} & p_{n2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{n1} & p_{n2} \\ 0 & 0 & 0 & 0 & -p_{n1} & -p_{n2} & p_{n1} & p_{n2} & p_{n1} & p_{n2} \end{pmatrix}$$

Де, матриця A має розміри 6×10, а матриця $\tilde{P}$ має розміри $4n \times 10$.

Тепер зазначимо, що наша цільова функція та функції нерівностей є опуклими та двічі дефірінційованими, а отже задача була зведена до задачі опуклої оптимізації, що вирішується так званим «log barrier» методом [5] за $O(1)$ ітерацій оптимізації, використовуючи «метод Ньютона при обмеженнях рівності», що описаний за посиланням [6].

2.3 Оцінка складності

Якщо використовувати «метод Ньютона», то кожна ітерація буде еквівалентна розв'язанню системи лінійних рівнянь $Hs = g$, де H є матрицею Гессіана, а g є вектором градієнта. Це обійдеться нам у $O(10^3) = O(1)$ операцій, плюс витрати на обчислення H та g.

Обчислення g та H вимагають $O(10 \times (4n+8)) = O(n)$ та $O(10^2 \times (4n+8)) = O(n)$ операцій відповідно. Таким чином загальна складність знаходження рішення нашої задачі оптимізації буде складати $O(n)$.

Зверніть увагу, що ця обчислювальна складність залежить лише від n, кількості нерівностей, які визначають опуклу множину C. Наприклад, для еліпсів це буде $O(1)$, оскільки нам потрібна лише одна нерівність для визначення еліпсоїдальної опуклої множини.

Варто зазначити, що сумарна складність алгоритму буде складати $O(n * \log n)$, адже ми також будуємо випуклу оболонку нашої множини точок «методом Грехема».

3 Практична частина

3.1 Особливості програмної реалізації

Програма була розроблена на мові програмування **C#** з використанням платформи **.NET** у середовищі розробки **Visual Studio**. Інтерфейс користувача реалізовано за допомогою технології **Windows Forms**, що забезпечує зручність і функціональність додатку.

Також варто зазначити, що задачу оптимізації було реалізовано з використанням бібліотеки **Python** *cvxpy*, що використовується для вирішення оптимізаційних задач. Але для повної коректності та оптимальності алгоритму треба реалізовувати «log barrier» метод, як було описано в основній частині 2.2.

3.2 Основні функції програмної реалізації

Нище наведено основну функцію програмної реалізації додатку, для обчислення опуклої оболонки. Також функцію перевірки належності точки опуклій оболонці, де обчислено коефіцієнти що потім будуть використані для вирішення нашої оптимізаційної задачі. Сама оптимізаційна задача вирішується за допомогою скрипту **Python** *cvx_optimization.py*.

```
public override PointF[] Run()
{
    Stack<PointF> stack = new Stack<PointF>();

    PointF p = GetBottomLeftMost();
    PointF[] sorted = SortPoints(p);

    for (int i = 0; i < sorted.Length; i++)
    {
        while (stack.Count > 1 && CCW(stack.ElementAt(1), stack.Peek(), sorted
            [i]) <= 0) {
            //Console.WriteLine("{0},{1},{2}",stack.ElementAt(1), stack.Peek()
                , sorted[i]);
            stack.Pop();
        }
        stack.Push(sorted[i]);
    }

    while (stack.Count > 1 && CCW(stack.ElementAt(1), stack.Peek(), p) <= 0)
    {
        stack.Pop();
    }

    stack.Push(p);
    return stack.ToArray();
}
```

Лістинг 1: Основний метод Run класу GrahamScan для знаходження точок випуклої оболочки

```
static public bool isInsideEquations(PointF[] CWHull, PointF point)
{
    for (int i = 0; i < CWHull.Length; ++i)
    {
        double p1 = (CWHull[i].Y - CWHull[(i + 1) % CWHull.Length].Y);
        double p2 = (CWHull[(i + 1) % CWHull.Length].X - CWHull[i].X);
        double b = CWHull[(i + 1) % CWHull.Length].X * CWHull[i].Y - CWHull[i
            ].X * CWHull[(i + 1) % CWHull.Length].Y;
        if (p1 * point.X + p2 * point.Y > b) return false;
    }
    return true;
}
```

Лістинг 2: Метод isInsideEquations для перевірки належності точки опуклій оболонці CWHull

Також наведемо деякі функції, що реалізують графічний інтерфейс програми, та обробку даних введених користувачем.

```
private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
{
    points.Add(AdjustPointCoordinates(new PointF(e.X, e.Y)));
    pictureBox1.Invalidate();
}
```

Лістинг 3: Функція обробки додавання нових точок

```
private void btnDrawRectangle_Click(object sender, EventArgs e)
{
    if (points.Count < 3)
    {
        MessageBox.Show("Please add at least three points.", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    grahamScan = new GrahamScan(points.ToArray());
    hull = grahamScan.Run();
    rectanglePoints = grahamScan.solveGetMaximumAxisAlignedRectangle();
    pictureBox1.Invalidate();
}
```

Лістинг 4: Функція обробки натискання на кнопку "Draw Rectangle"

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;

    // Draw coordinate system
    DrawCoordinateSystem(g);

    // Draw points
    foreach (var point in points)
    {
        PointF adjustedPoint = AdjustPointCoordinates(point);
        g.FillEllipse(Brushes.Black, adjustedPoint.X - 2, adjustedPoint.Y - 2,
            4, 4);
        g.DrawString($"({point.X},{point.Y})", new Font("Arial", 8), Brushes.
           Blue, new PointF(adjustedPoint.X + 5, adjustedPoint.Y - 15));
    }

    // Draw hull
    if (hull != null)
    {
        DrawHull(hull, g);
    }

    // Draw rectangle
    if (rectanglePoints != null)
    {
        DrawRectangle(rectanglePoints, g);
    }
}
```

Лістинг 5: Функція для оновлення графічного вікна

3.3 Характеризація вводу-виводу даних

Ввід точки виконується натисканням миші на полотно вводу, а також можливе додавання обраної кількості випадкових точок за допомогою кнопки *Random Points*. Виведення результатів виконується натисканням кнопки *Draw Rectangle*, або *Draw Hull* для виводу оболонки без прямокутника. Також можливе очищення вводу натисканням кнопки *Clear Points*. Для наочності наведемо зображення роботи програми.

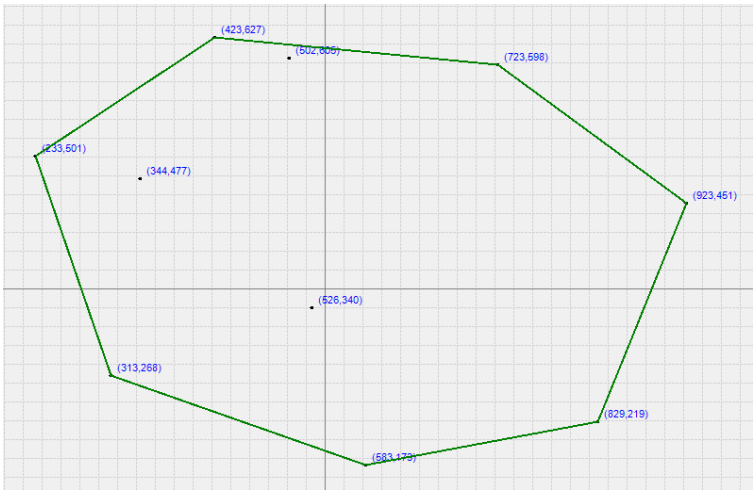

Рис. 4: Вичислена опукла оболонка заданої множини точок

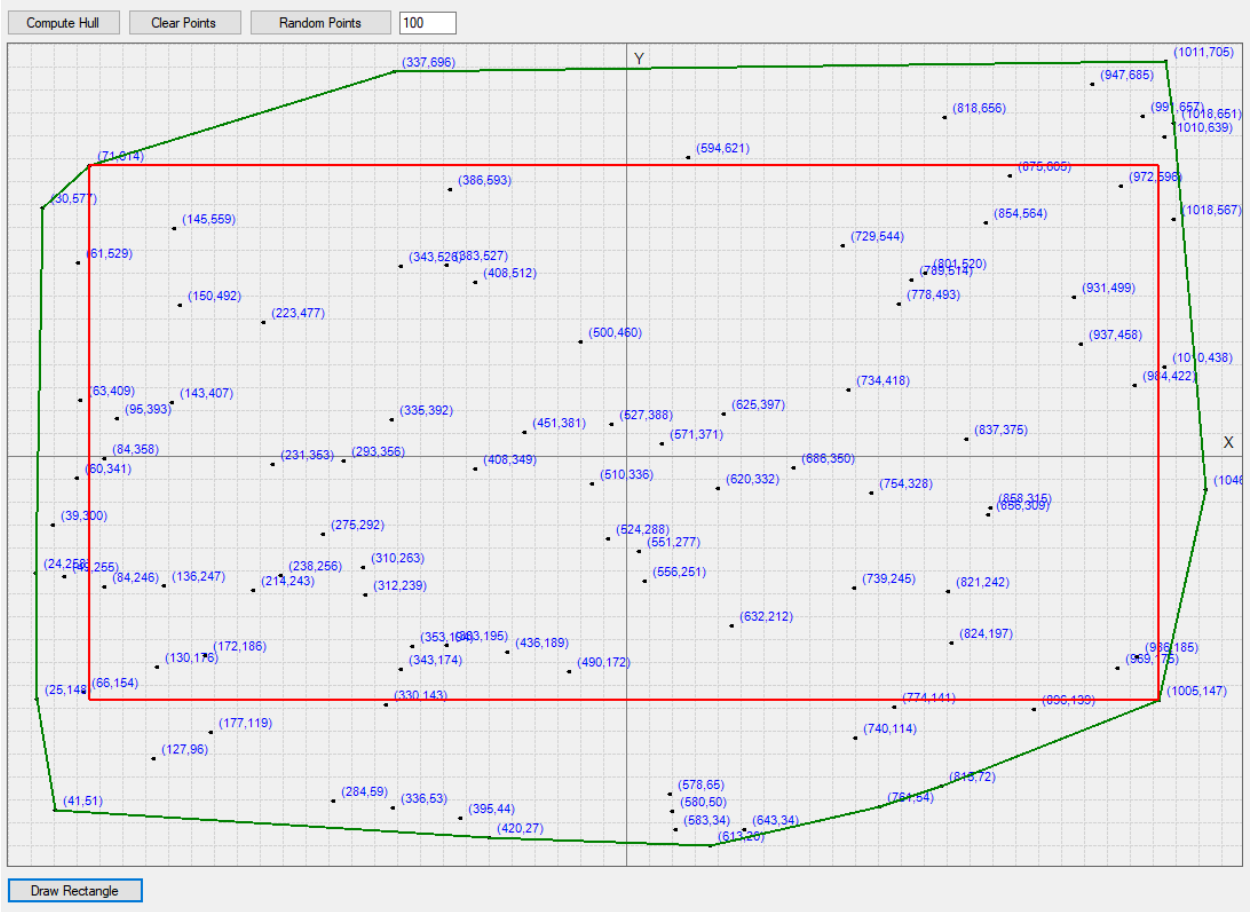

Рис. 5: Опукла оболонка з максимальним вписаним прямокутником з множини 100 випадково заданих точок

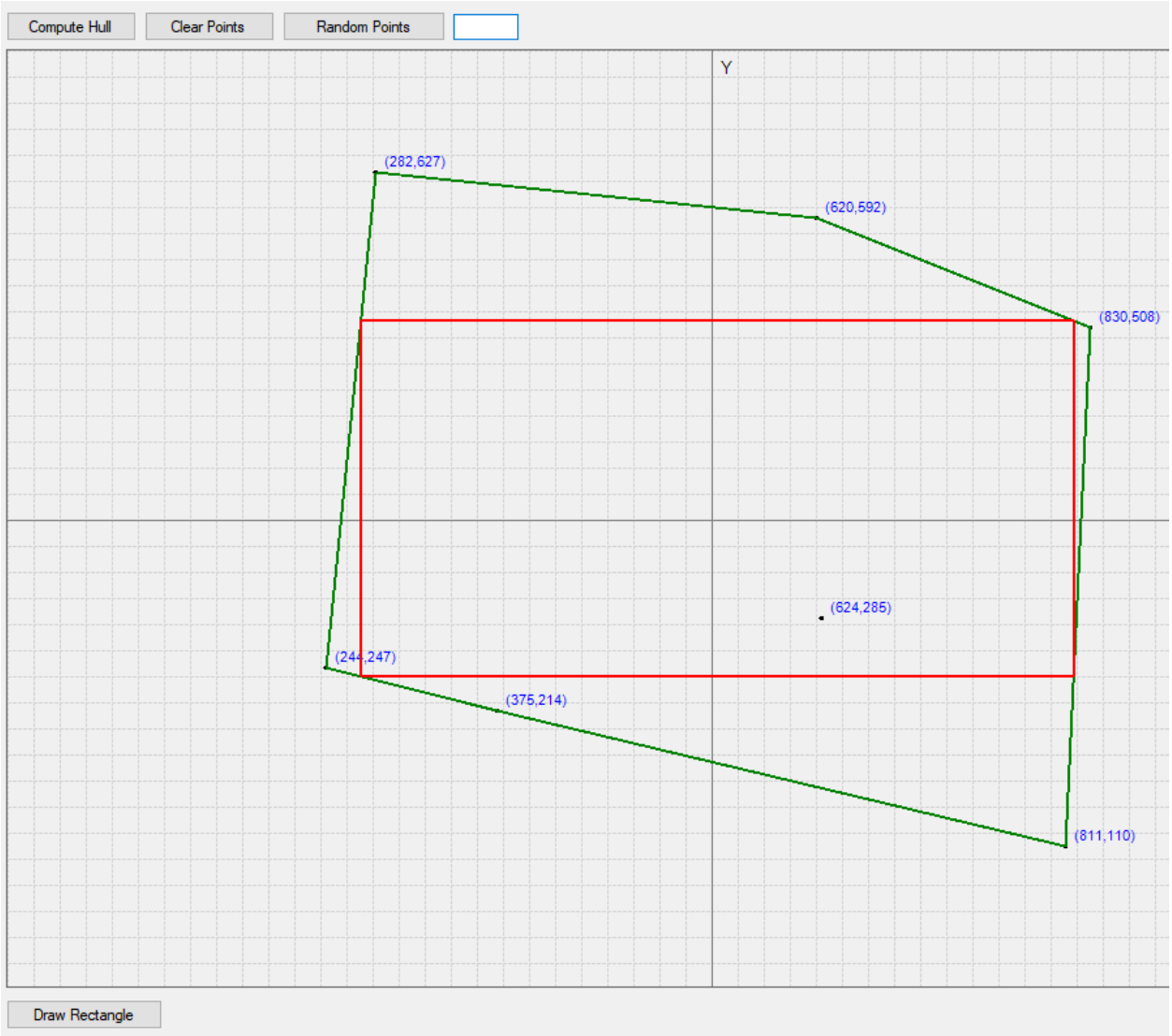

Рис. 6: Опукла оболонка з максимальним вписаним прямокутником

4 Висновки

Отже у даній лабораторній роботі було досліджено задачу знаходження максимального прямокутника вписаного в опуклу оболонку. Було проведено аналіз існуючих та сучасних алгоритмів та описано один з них, що був запропонований Бехрузі у його роботі. Також було написано демонстранційну програму мовою C#, що реалізує алгоритм.

5 Додатки

Посилання на github: *GitHub репозиторій*

```
public static double Distance(PointF p1, PointF p2)
{
    double dx = p2.X - p1.X;
    double dy = p2.Y - p1.Y;
    return Math.Sqrt(dx * dx + dy * dy);
}
```

Лістинг 6: Функція для визначення відстані між двома точками

```
public static float Cross(PointF p1, PointF p2, PointF p3)
{
    return (p1.X - p3.X) * (p2.Y - p3.Y) - (p1.Y - p3.Y) * (p2.X - p3.X);
}
```

Лістинг 7: Фукнція для визначення орієнтовної площі трикутника

```
private PointF[] SortPoints(PointF p)
{
    List<PointFAngle> s = new List<PointFAngle>();
    for (int i = 0; i < points.Length; i++)
    {
        s.Add(new PointFAngle()
        {
            Point = points[i],
            Angle = Math.Atan2(points[i].Y - p.Y, points[i].X - p.X)
        });
        if (s[i].Angle < 0) s[i].Angle += PI2;
        if (s[i].Angle < ANGLE_EPS) s[i].Angle += PI2;
    }
    s.Sort((p1, p2) => Math.Sign(p1.Angle - p2.Angle));

    List<PointF> res = new List<PointF>();
    for (int i = 0; i < s.Count; i++)
    {
        int cur_i = i;
        int best_i = i;
        while (cur_i<s.Count && Math.Abs(s[i].Angle - s[cur_i].Angle) <=
            ANGLE_EPS)
        {
            if (Distance(p, s[cur_i].Point)>Distance(p, s[best_i].Point))
            {
                best_i=cur_i;
            }
            ++cur_i;
        }
        res.Add(s[best_i].Point);
```

```
        i = cur_i - 1;
    }
    //for (int i = 0; i < s.Count; ++i)
    //{
    //    Console.WriteLine("Point: {0}, Angle {1}", s[i].Point, s[i].Angle);
    //}
    return res.ToArray();
}
```

Лістинг 8: Функція сортування масиву точок за кутом до заданої точки p

```
private void btnRandomPoints_Click(object sender, EventArgs e)
{
    int numberOfPoints;
    if (int.TryParse(txtNumberOfPoints.Text, out numberOfPoints))
    {
        Random random = new Random();
        points.Clear();
        for (int i = 0; i < numberOfPoints; i++)
        {
            float x = random.Next(10, pictureBox1.Width - 10);
            float y = random.Next(10, pictureBox1.Height - 10);
            points.Add(new PointF(x, y));
        }
        hull = null;
        rectanglePoints = null;
        pictureBox1.Invalidate();
    }
    else
    {
        MessageBox.Show("Please enter a valid number of points.", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Лістинг 9: Функція для генерації випадкових точок на формі

Література

[1] Behroozi, M. (2019). Largest inscribed rectangles in geometric convex sets. arXiv preprint arXiv:1905.13246.

[2] DANIELS, Karen; MILENKOVIC, Victor; ROTH, Dan. Finding the largest area axis-parallel rectangle in a polygon. Computational Geometry, 1997, 7.1-2: 125-148.

[3] ALT, Helmut; HSU, David; SNOEYINK, Jack. Computing the largest inscribed isothetic rectangle. In: CCCG. 1995. p. 67-72.

[4] CORMEN, Thomas H., et al. 33.3: Finding the convex hull. Introduction to Algorithms, 1990, 955-956.

[5] Lecture 15: Log Barrier Method Lecturer: Ryan Tibshirani 10-725/36-725:

Convex Optimization Spring 2015

[6] Lecture 12: October 4 Lecturer: Geoff Gordon/Ryan Tibshirani 10-725: Optimization Fall 2012

[7] Cabello, Sergio, et al. ”Finding largest rectangles in convex polygons.” Computational Geometry 51 (2016): 67-74.