

Algoritmos e Estrutura de Dados III

Primeiro Trabalho Prático - Hipercampos

Pablo Cecilio Oliveira
Alexander Cristian

1 Introdução

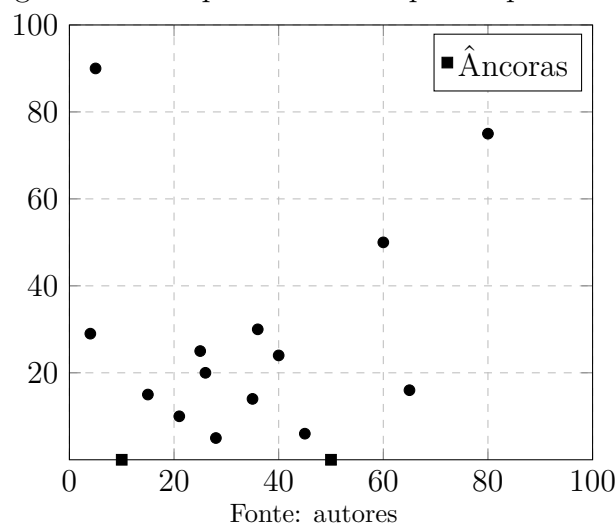
Na Ciência da Computação, o estudo de algoritmos para resolução de problemas geométricos é conhecido como Geometria Computacional. De forma geral, o objetivo deste ramo é resolver de maneira eficiente utilizando o menor número possível de operações sobre os elementos geométricos elementares.[6]

Dentre vários problemas geométricos temos o desafio conhecido como "Hipercampos", o qual pode ser visto em algumas maratonas de programação[2]. Neste trabalho é apresentado a solução para esse problema por meio de um algoritmo contido em um programa desenvolvido na linguagem em C.

1.1 Hipercampos, especificação do problema

No problema de Hipercampos, um plano cartesiano em \mathbb{R}^2 possui duas "âncoras", dois pontos A e B , onde o eixo Y das duas âncoras são iguais a zero, ou seja $A = (X_A, 0)$ e $B = (X_B, 0)$. Os valores do eixo X das âncoras variam de X_A até X_B , formando assim um segmento de reta horizontal, tal que $0 < X_A < X_B \leq 10^4$. (Figura 1)

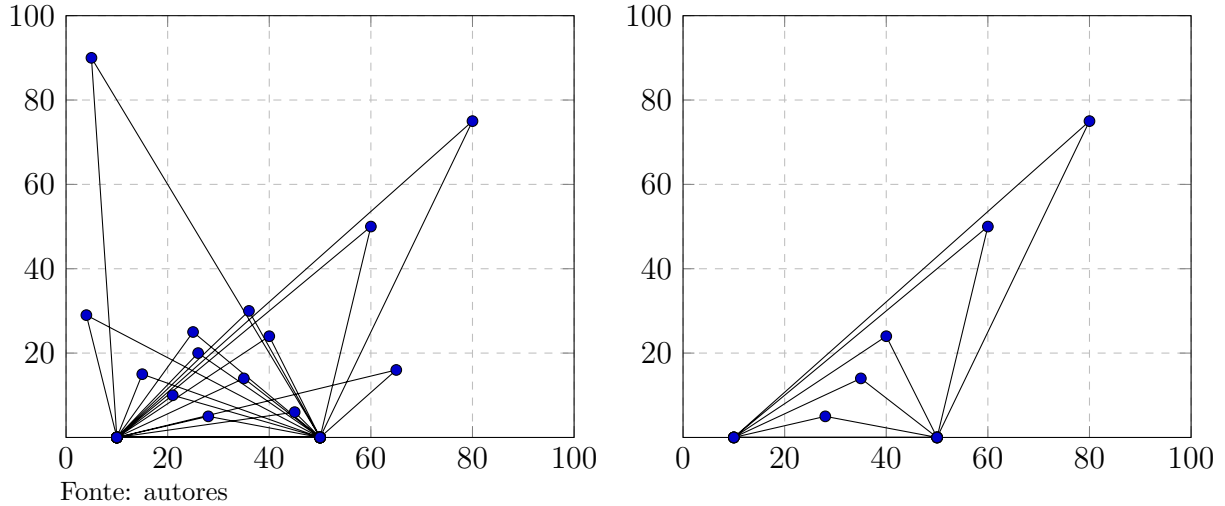
Figura 1: Exemplo de entrada para o problema.



Ao plano cartesiano também somam-se um conjunto P de N coordenadas (X_i, Y_i) , sendo que as N coordenadas do Conjunto P variam em seu total entre um e cem, $N(1 \leq N \leq 100)$. As coordenadas (X_i, Y_i) podem variar entre 0 até 10^4 , ou seja, $0 < X_i, Y_i \leq 10^4$.

O objetivo do problema de Hipercampos é ligar as coordenadas contidas em P às âncoras X_A e X_B , formando assim o máximo número de triângulos sem que esses se interceptem (Figura 2). E para esse propósito foi desenvolvido um algoritmo contido no programa apresentado neste trabalho.

Figura 2: Hipercampos, solucionando.



1.2 Visão geral sobre o funcionamento do programa

O programa desenvolvido recebe por parâmetro a entrada de um arquivo contendo em sua primeira linha um número N total de coordenadas, e o valor para o eixo X das âncoras A e B respectivamente. As linhas subsequentes a primeira correspondem às coordenadas das N tuplas do conjunto P a ser solucionado.

O algoritmo então processa esses dados retornando uma solução que pode ser verificada por meio de dois arquivos também gerados por um parâmetro: um contendo o número de triângulos possíveis e o outro em forma de uma imagem renderizada como "gráfico vetorial escalável" (Scalable Vector Graphics, ou ".svg")¹. Um terceiro arquivo em ".svg" é gerado contendo a entrada original dos dados para referência.

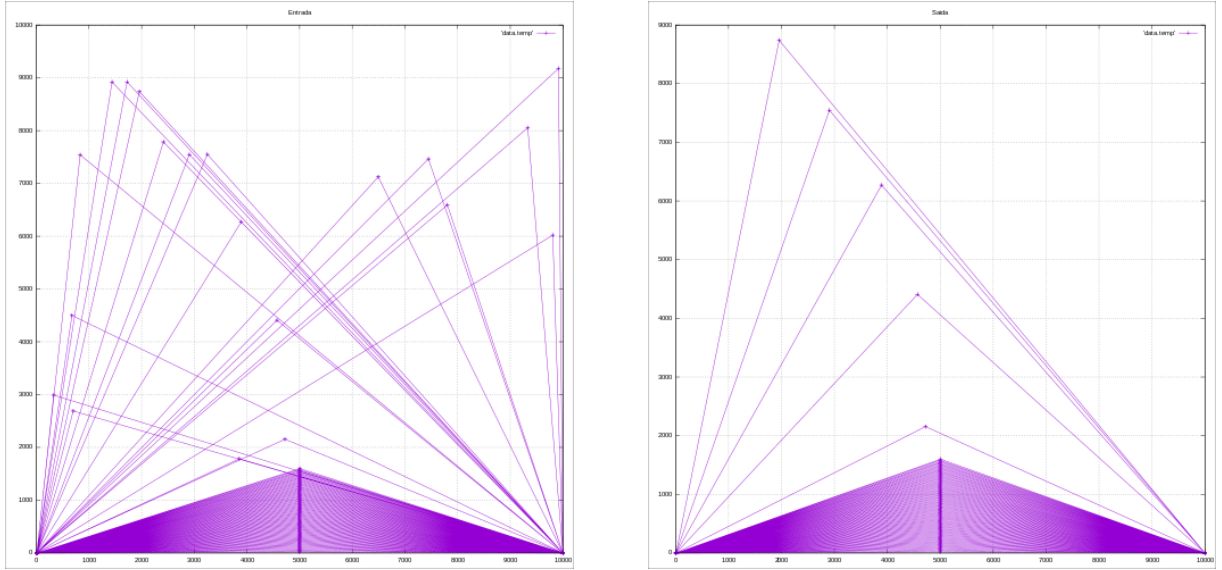
O programa é executado no prompt de comando e recebe as passagens de parâmetro dos arquivos de entrada e saída:

```
$ ./hcamp -i [input] -o [output]
```

A entrada e solução renderizada pode ser vista no exemplo da Figura 3.

¹Trata-se de uma linguagem XML para descrever de forma vetorial desenhos e gráficos bidimensionais.

Figura 3: Entrada e saída renderizadas.



Fonte: via gnuplot com dados de entrada obtidos em uDebug.[5]

2 Implementação

Inicialmente os dados contidos no arquivo de entrada são verificados e transferidos para uma lista simplesmente encadeada, esta limitada ao tamanho da memória principal disponível.

Após os dados estarem disponíveis na memória, uma função recursiva contendo um algoritmo para a solução determina entre todos os pontos do plano, qual ponto possui o maior numero de coordenadas dentro da área formada pelo ponto testado e suas duas âncoras. Este teste a principio tem como premissa o argumento em que o ponto com o maior número de coordenadas em sua área será aquele que terá o maior número de possibilidades de formações triangulares subsequentes.

O teste recursivo se repete sucessivamente para cada ponto interno em relação ao ponto inicialmente encontrado como sendo o de maior número de coordenadas (pontos X_i, Y_i), determinando o maior conjunto de elementos em relação ao conjunto anteriormente encontrado. O processo finaliza quando não existem mais coordenadas a serem encontradas.

A função que determina se uma coordenada está ou não dentro de uma área formada pelo ponto e suas duas ancoras é derivada do método do produto vetorial entre duas retas.[3] Este consiste em calcular a orientação do segmento de reta entre as âncoras e o ponto a ser testado com o ponto que forma um triangulo partindo das âncoras, determinando assim se o ponto testado está dentro da área formada pelo ponto de referência.

A equação que determina essa orientação é dada por:

$$(y2 - y1) * (x3 - x2) - (y3 - y2) * (x2 - x1)$$

Como o eixo Y das âncoras são iguais a zero, a equação pode ser simplificada como:

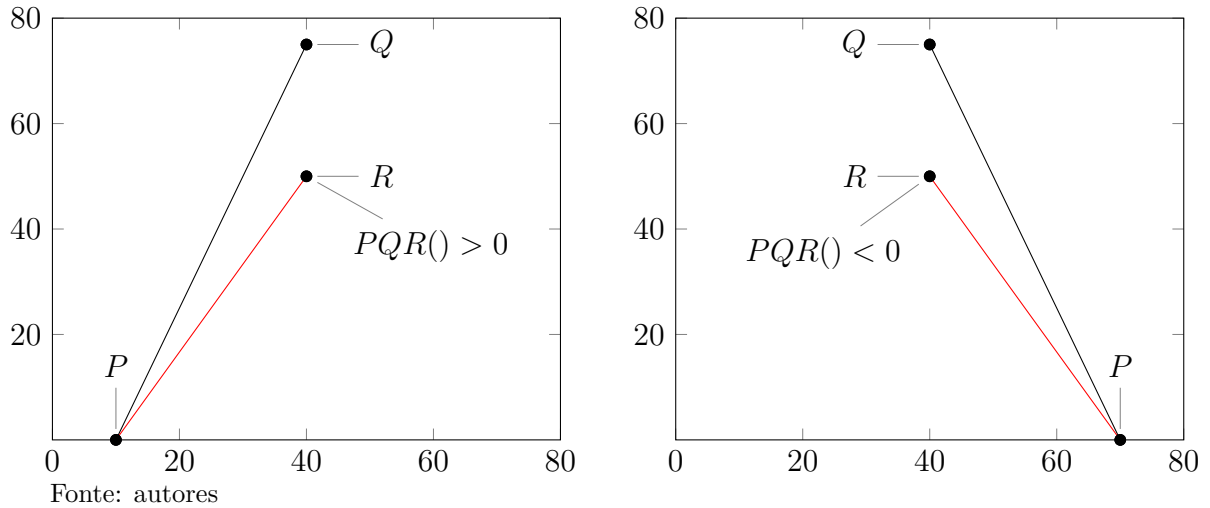
$$y2 * (x3 - x2) - (y3 - y2) * (x2 - x1)$$

Aplicando a equação entre os segmento de reta \overline{PQ} , sendo P a âncora e Q o ponto que forma o triângulo), com \overline{PR} (R , ponto sendo testado), resulta no valor que determina a orientação da reta \overline{PR} em relação a \overline{PQ} . No caso do resultado for maior que zero, a reta \overline{PR} está no sentido horário a reta \overline{PQ} , caso seja menor do que zero, está em sentido anti-horário à \overline{PQ} .

A relação entre os segmentos de reta \overline{PQ} e \overline{PR} com as coordenadas respectivas às âncoras em P , resultam que se um ponto estiver no sentido horário à $\overline{PQ_A}$ e anti-horário à $\overline{PQ_B}$, este está dentro da área do triângulo formado entre as duas âncoras e o ponto Q .

A Figura 5 demonstra esse o conceito.

Figura 4: Determinando o ponto interno ao triângulo



É por meio do método de força bruta com o teste recursivo entre todos os pontos que são encontrados os triângulos que possuem sempre o maior numero de coordenadas em sua área. Resultando desse numero de triângulos encontrados o valor para a solução.

A Tabela 1 contém a lista das principais funções utilizadas no programa, uma descrição sucinta de suas finalidades e sua complexidade por tempo.

Tabela 1: Funções do programa.

Função	Finalidade	Complexidade*
<i>debug()</i>	Função que verifica a condição para retorno de possíveis bugs no programa.	$O(1)$
<i>create()</i>	Inicializa a Lista encadeada.	$O(1)$
<i>insere()</i>	Insere os dados em uma lista encadeada.	$O(1)$
<i>printCJT()</i>	Imprime uma Lista encadeada.	$O(n)$
<i>sizeCJT()</i>	Retorna o tamanho da lista encadeada.	$O(n)$
<i>dump()</i>	Libera a memória alocada pela lista.	$O(n)$
<i>isEmpty()</i>	Verifica se uma lista encadeada está vazia.	$O(1)$
<i>openFILE()</i>	Abre o arquivo solicitado e transfere os dados para uma lista encadeada.	$O(n)$
<i>saveFILE()</i>	Salva a solução do problema em um arquivo.	$O(1)$
<i>chkFILE()</i>	Verifica por possíveis erros de entrada em um arquivo.	$O(n)$
<i>showerro()</i>	Retorna possíveis erros no arquivo de entrada.	$O(1)$
<i>ask()</i>	Solicita a confirmação do usuário caso erros de entrada sejam encontrados.	$O(1)$
<i>cpyCJT()</i>	Copia os dados de uma lista encadeada para outra lista encadeada.	$O(n)$
<i>PQR()</i>	Algoritmo de orientação do ponto em relação a reta da ancora.	$O(1)$
<i>findMAX()</i>	Função recursiva que determina o maior conjunto de pontos que se encontram dentro do triângulo formado pelas ancoras e um ponto (x, y) .	$O(n)$
<i>soluciona()</i> <i>solucao()</i>	Funções de chamada e retorno para a execução do algoritmo	$O(1)$
<i>plotGraph()</i>	PIPE para o gnuplot com a finalidade de renderizar os arquivos .svg contendo respectivamente, a entrada e saída da solução do problema.	$O(n)$

Fonte: autores

*Complexidade por tempo.

2.1 Análise de complexidade

Pergunta:

So I had to insert N elements in random order into a size- N array, but I am not sure about the time complexity of the program.

The program is basically:

```
for (i = 0 -> n-1) {  
    index = random (0, n); // (n is exclusive)  
    while (array[index] != null)  
        index = random (0, n);  
    array[index] = n  
}
```

Resposta:

First consider the inner loop. When do we expect to have our first success (find an open position) when there are i values already in the array? For this we use the geometric distribution:

$$Pr(X = k) = (1 - p)^{k-1}p$$

Where p is the probability of success for an attempt. Here p is the probability that the array index is not already filled. There are i filled positions so $p = (1 - (i/n)) = ((n - i)/n)$.

From the wiki, the expectation for the geometric distribution is $1/p = 1/((n - i)/n) = n/(n - i)$. Therefore, we should expect to make $(n/(n - i))$ attempts in the inner loop when there are i items in the array.

To fill the array, we insert a new value when the array has $i = 0..n - 1$ items in it. The amount of attempts we expect to make overall is the sum:

$$\begin{aligned}\sum_{i=0}^{n-1} \frac{n}{n-i} &= n * \sum_{i=0}^{n-1} \left(\frac{1}{(n-i)} \right) \\ &= n * (1/n + 1/(n-1) + \dots + 1/1) \\ &= n * (1/1 + \dots + 1/(n-1) + 1/n) \\ &= n * \sum_{i=1}^n \frac{1}{i}\end{aligned}$$

Which is n times the n th harmonic number and is approximately $\ln(n) + \gamma$, where γ is a constant. So overall, the number of attempts is approximately $n * (\ln(n) + \gamma)$, which is $O(n \log n)$. Remember that this

is only the expectation and there is no true upper bound since the inner loop is random; it may never find an open spot.

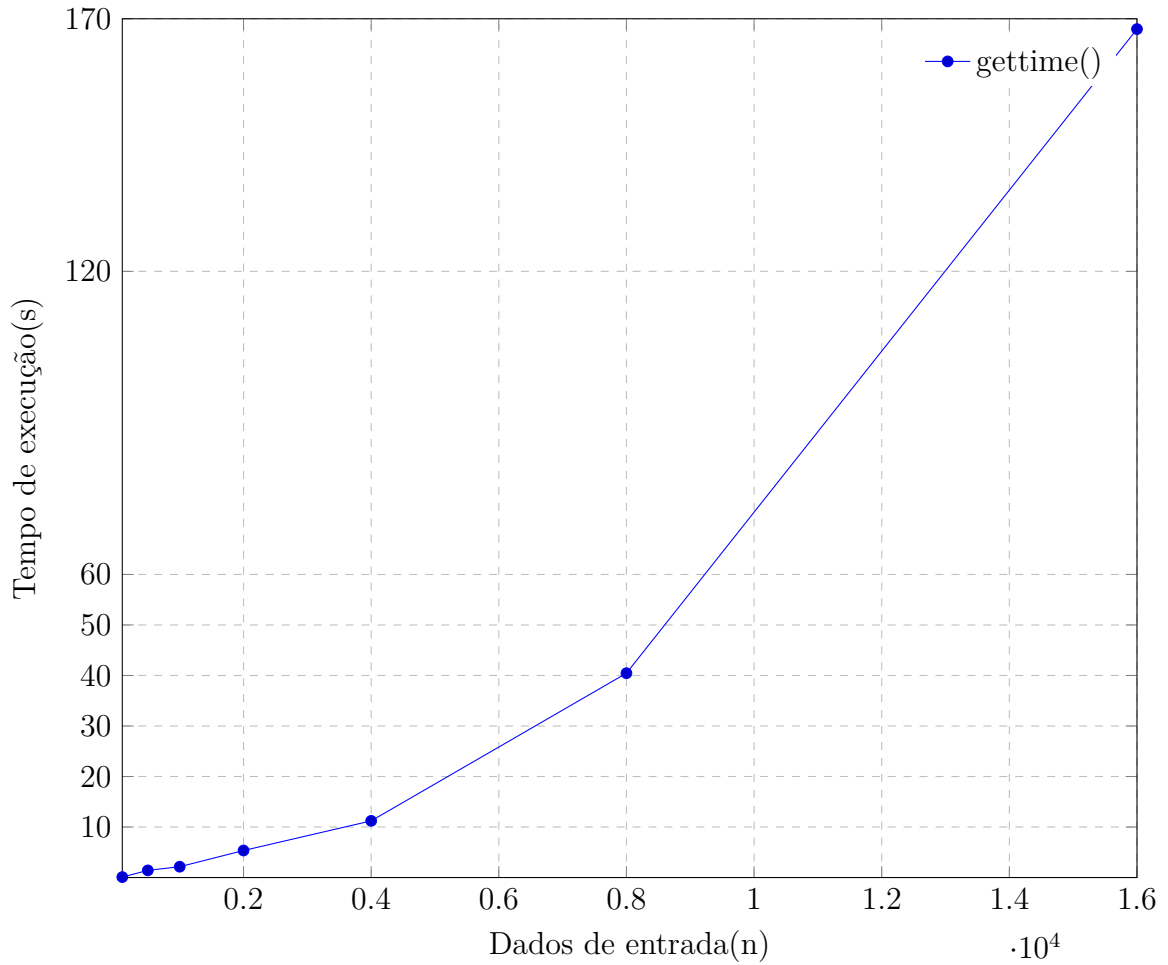
2.2 Tempos de computação

Tabela 2: Tempos de execução em relação a entrada n .

n	<i>gettime()</i>	<i>ru_utime()</i>	<i>ru_stime()</i>
100	91751 μ s	9847 μ s	0
500	1418852 μ s	154276 μ s	3435 μ s
1000	2144396 μ s	559727 μ s	3290 μ s
2000	5350661 μ s	2s 246933 μ s	3324 μ s
4000	11219083 μ s	9s 167495 μ s	3333 μ s
8000	40450777 μ s	38s 906846 μ s	3324 μ s
16000	167955974 μ s	166s 90082 μ s	6662 μ s

Fonte: autores

Figura 5: Tempos de execução em relação a entrada n .



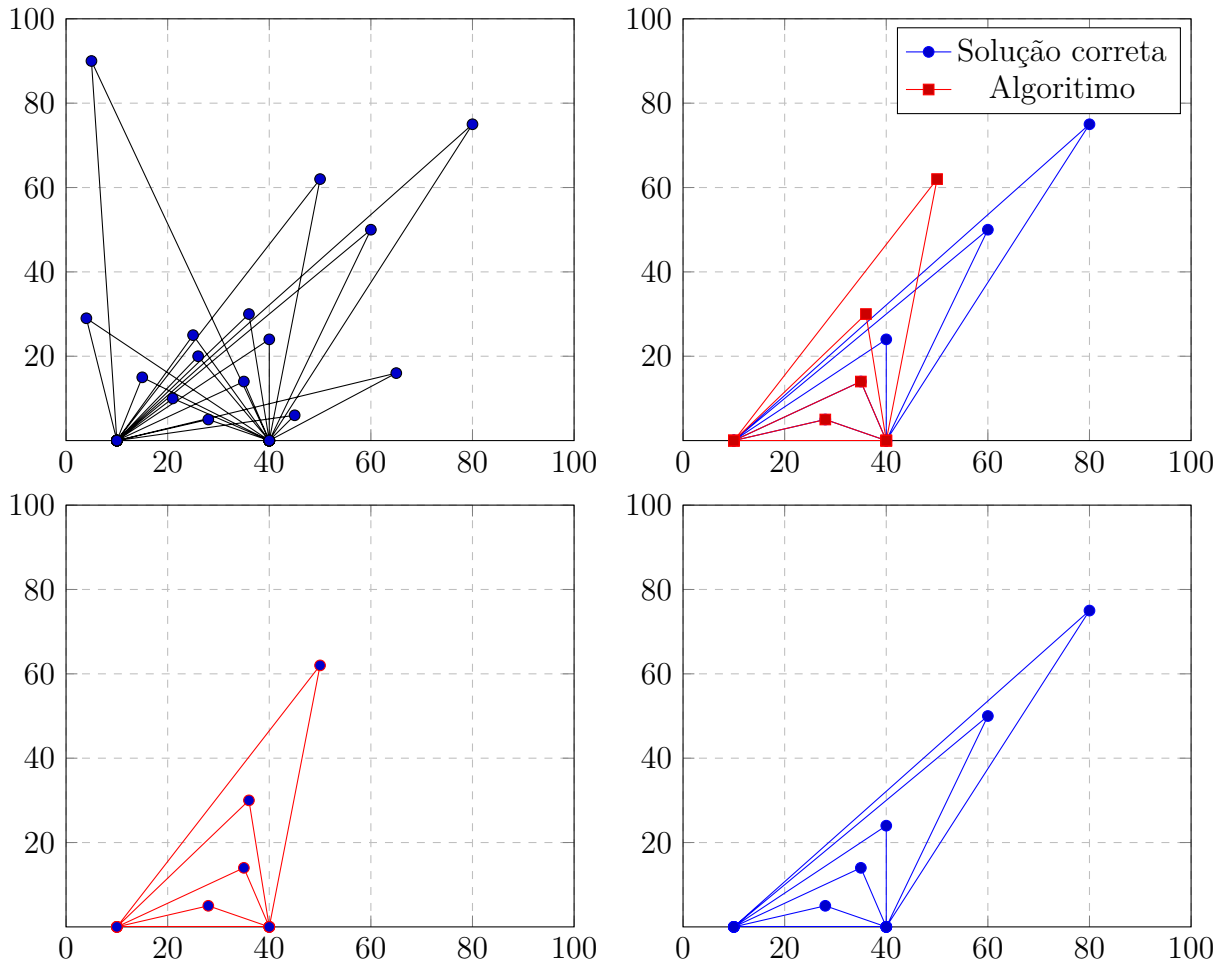
Fonte: autores

3 Considerações finais

Durante o desenvolvimento vários métodos foram testados com o objetivo de encontrar a maneira mais eficiente de encontrar a solução para o problema de Hipercampos. Foram considerados LCS², coordenadas baricêntricas³ e outros métodos envolvendo o cálculo da área formada pelos triângulos. Porém, não foi possível neste trabalho chegar a um resultado satisfatório quanto a aplicação desses métodos ao problema.

O uso de um algoritmo recursivo de força bruta foi o que mais se aproximou de um resultado preciso, mas mesmo esse método falhou em testes mais elaborados. A premissa de que o triângulo que contém com o maior número de coordenadas em sua área será aquele que terá o maior número de possibilidades de formações triangulares subsequentes, é incompleta, pois não considera a área dos triângulos formados. Esse erro pode ser verificado visualmente pela Figura 6.

Figura 6: Falha no algoritmo.



²Longest common subsequence ou máxima subsequência crescente consiste em encontrar um subseqüência de números, dada um seqüência, na qual seus elementos estão ordenados do menor para o maior, e a seqüência é a mais longa possível.

³As Coordenadas Baricêntricas definem uma forma de representação de um ponto no espaço em função de outros pontos.

O algoritmo sempre escolhe o triângulo com o maior número de coordenadas em sua área, não considerando que essas possam ser eliminadas em teste subsequentes. Quanto maior a área do triângulo testado, maior será a possibilidade de erros. E embora o programa cumpra a função de determinar um número de triângulos que não se interceptam, esse não é preciso quanto ao valor máximo que pode ser obtido.

Uma solução alternativa envolvendo o conceito de "dividir para conquistar" seria utilizar como entradas para um algoritmo, duas árvores binárias, cada uma delas conteria como raiz uma das âncoras e como filhos as coordenadas do arquivo de entrada. O algoritmo então testaria colisões, verificando se a reta formada a partir de um ponto até às âncoras é concorrente à alguma das retas encontradas em uma das duas árvores binárias. Porém não se chegou a um algoritmo eficaz que pudesse decidir entre um número de colisões iguais.

Referências

- [1] et al. Elin, Kisielewicz. How to determine if a point is in a 2d triangle? <https://stackoverflow.com/questions/2049582/how-to-determine-if-a-point-is-in-a-2d-triangle>. [Acesso em: 23-Agosto-2018].
- [2] URI Online Judge. Hipercampo. <https://www.urionlinejudge.com.br/judge/en/problems/view/2665>. [Acesso em: 3-Setembro-2018].
- [3] Cédric Jules. Accurate point in triangle test. <http://totologic.blogspot.com/2014/01/accurate-point-in-triangle-test.html>. [Acesso em: 23-Agosto-2018].
- [4] Patrick Prosser. Geometric algorithms. <http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>. [Acesso em: 23-Agosto-2018].
- [5] Vitor Vitela. Hipercampo. <https://www.udebug.com/URI/2665>. [Acesso em: 3-Setembro-2018].
- [6] Wikipedia contributors. Computational geometry — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Computational_geometry&oldid=841504892, 2018. [Acesso em: 3-Setembro-2018].

O histórico do desenvolvimento desse trabalho se encontra online em: <https://github.com/Durfan/ufs-j-aeds3-tp1>.