

# Algoritmos e Estrutura de Dados III

## Primeiro Trabalho Prático - Hipercampos

Pablo Cecilio Oliveira

Alexander Cristian

### 1 Introdução

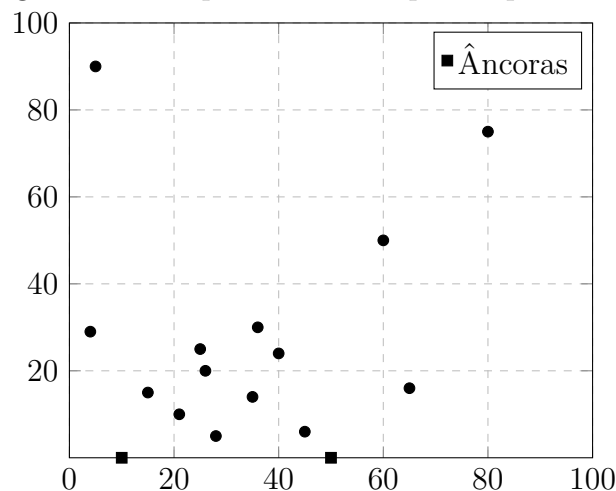
Na Ciência da Computação, o estudo de algoritmos para resolução de problemas geométricos é conhecido como Geometria Computacional. De forma geral, o objetivo deste ramo é resolver de maneira eficiente utilizando o menor número possível de operações sobre os elementos geométricos elementares.[6]

Dentre os problemas geométricos, temos um conhecido como "Hipercampos", o qual pode ser visto como desafio em 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  $0 < X_A < X_B \leq 10^4$ . (Fig. 1)

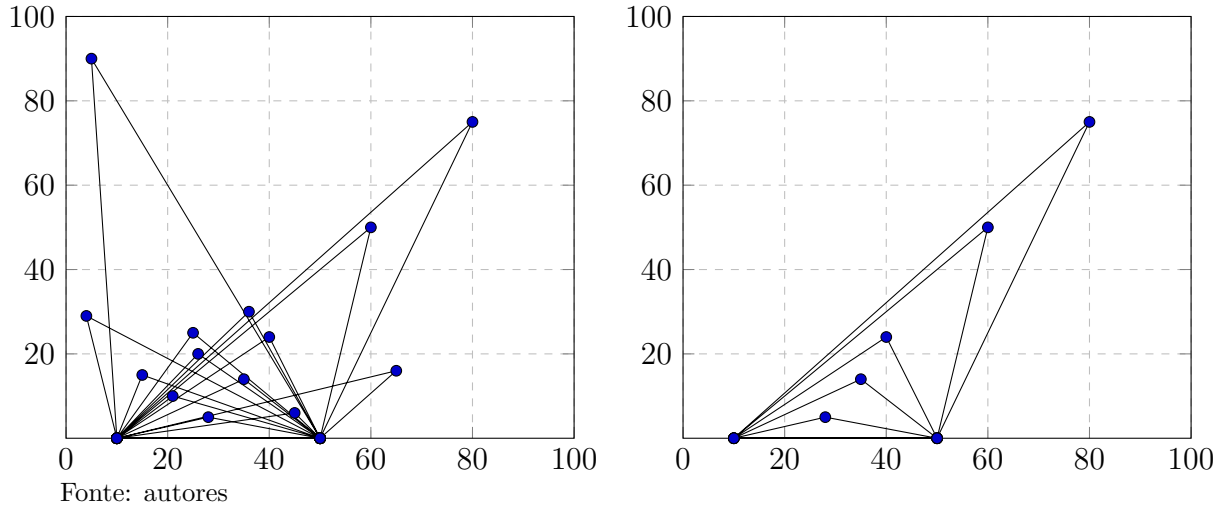
Figura 1: Exemplo de entrada para o problema.



Ao plano cartesiano também somam-se um conjunto  $P$  de  $N$  pontos  $(X_i, Y_i)$ , sendo que  $N(1 \leq N \leq 100)$ . Os pontos do conjunto  $P$  podem ter suas coordenadas variando entre 0 até  $10^4$ , ou seja,  $0 < X_i, Y_i \leq 10^4$ .

O objetivo do problema de Hipercampos é ligar os pontos contidos em  $P$  às âncoras  $X_A$  e  $X_B$ , formando assim o máximo número de triângulos sem que esses se interceptem (Fig. 2). E para esse proposito é apresentado 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  $N$  de pontos no plano  $\mathbb{R}^2$  e as coordenadas do eixo  $X$  das âncoras  $A$  e  $B$ , respectivamente. As linhas subsequentes a primeira correspondem às coordenadas dos  $N$  pontos 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")<sup>1</sup>. Um terceiro arquivo em ".svg" é gerado contendo a entrada original dos dados para referência.

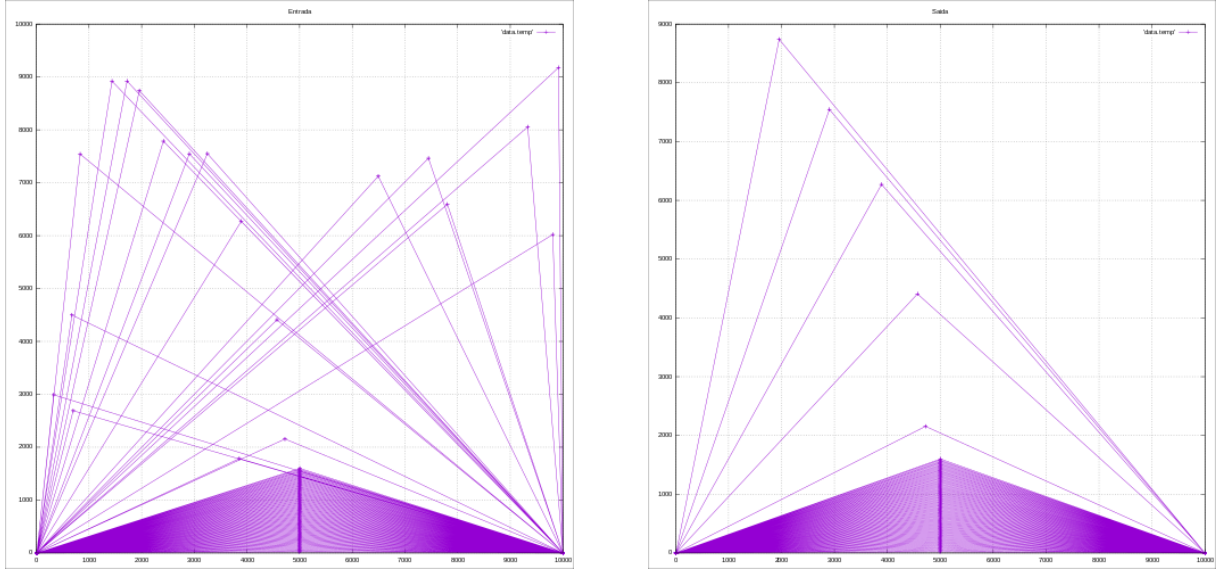
O programa é executado no prompt de comando e recebe as passagens de parâmetro dos arquivos de entrada e saída:

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

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

<sup>1</sup>Trata-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, a função que contém um algoritmo recursivo determina entre todos os pontos do plano, qual possui o maior número de coordenadas dentro da área formada pelo ponto testado e suas duas âncoras. Este teste a princípio 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 então 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 âncoras é derivada do método do produto vetorial entre duas retas.[3] Este consiste em calcular a orientação do segmento de reta formado entre as âncoras e o ponto a ser testado, em relação a orientação do segmento de reta formado com um ponto determinado como aquele a formar o triângulo.

A equação que determina essa orientação é dada por:

$$(y_2 - y_1) * (x_3 - x_2) - (y_3 - y_2) * (x_2 - x_1)$$

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

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

Snippet do código em C com o teste de orientação:

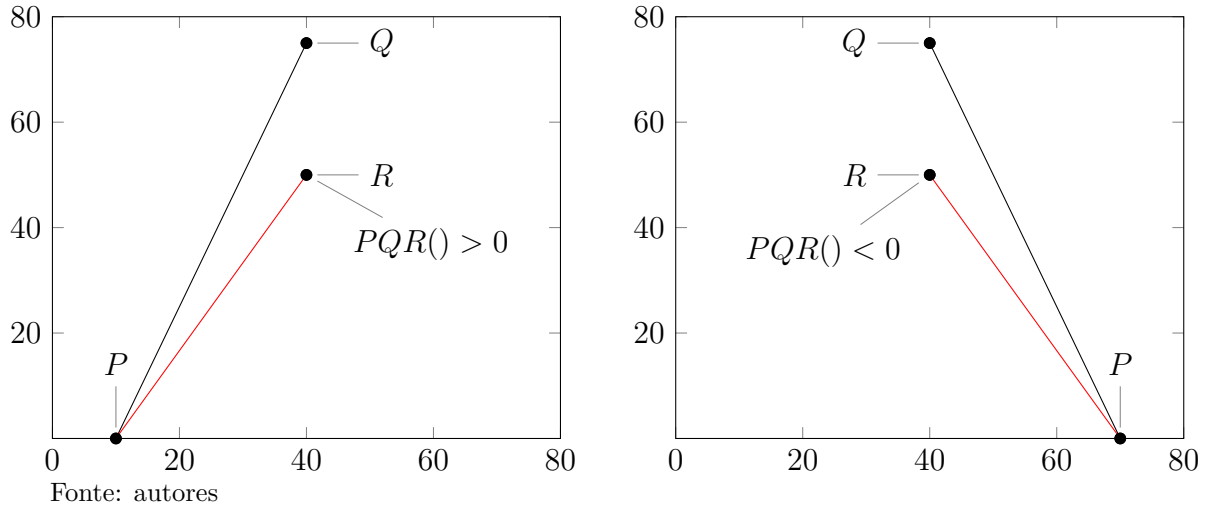
```
PaQR = Q->p.y*(R->p.x-Q->p.x)-(Q->p.x-P->Xa)*(R->p.y-Q->p.y);
PbQR = Q->p.y*(R->p.x-Q->p.x)-(Q->p.x-P->Xb)*(R->p.y-Q->p.y);
if (!PaQR || !PbQR) return 0; // colinear
PaQR = (PaQR > 0)? 1 : 2; // clock wise 1, counter clock 2
PbQR = (PbQR > 0)? 1 : 2; // clock wise 1, counter clock 2
if (PaQR == 1 && PbQR == 2) return 1; // inside
```

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 a  $\overline{PQ}$ .

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

Funções	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

---

**Algorithm 1** Ache o conjunto máximo de triângulos

---

```
1: procedure findMAX(Conjunto  $P$ )
2:   while  $n$  do
3:     while  $n$  do
4:       if  $n - 1$  está dentro do triângulo testado then
5:         insira em outro conjunto
6:       end if
7:     end while
8:   end while
9: end procedure
```

---

### 3 Análise de Complexidade

```
do {
    R = CJT->head;
    while (R != NULL) {
        if (PQR(CJT,Q,R)) {
            dots++;
            insere(R->p,AUX);
        }
        R = R->next;
    }
    if (dots >= moredots) {
        moredots = dots;
        win = Q;
        dump(MAX,0);
        cpyCJT(AUX,MAX);
    }
    dots = 0;
    dump(AUX,0);
    Q = Q->next;
} while (Q != NULL);

if (win) insere(win->p,plot);
dump(CJT,0);
cpyCJT(MAX,CJT);
dump(MAX,1);
dump(AUX,1);
CJT->total++;
return findMAX(CJT,plot);
```

$$f(n) = 11 + (n + 1) * \frac{n}{n - 1} + \frac{n}{2} * \frac{1}{2n} + n + \frac{n}{2} + 4n$$

$$f(n) = 11 + \frac{n^2 + n}{n - 1} + \frac{n}{4n} + \frac{n}{2} + 5n$$

$$f(n) = \frac{26n^2 + 27n - 45}{4(n - 1)}$$

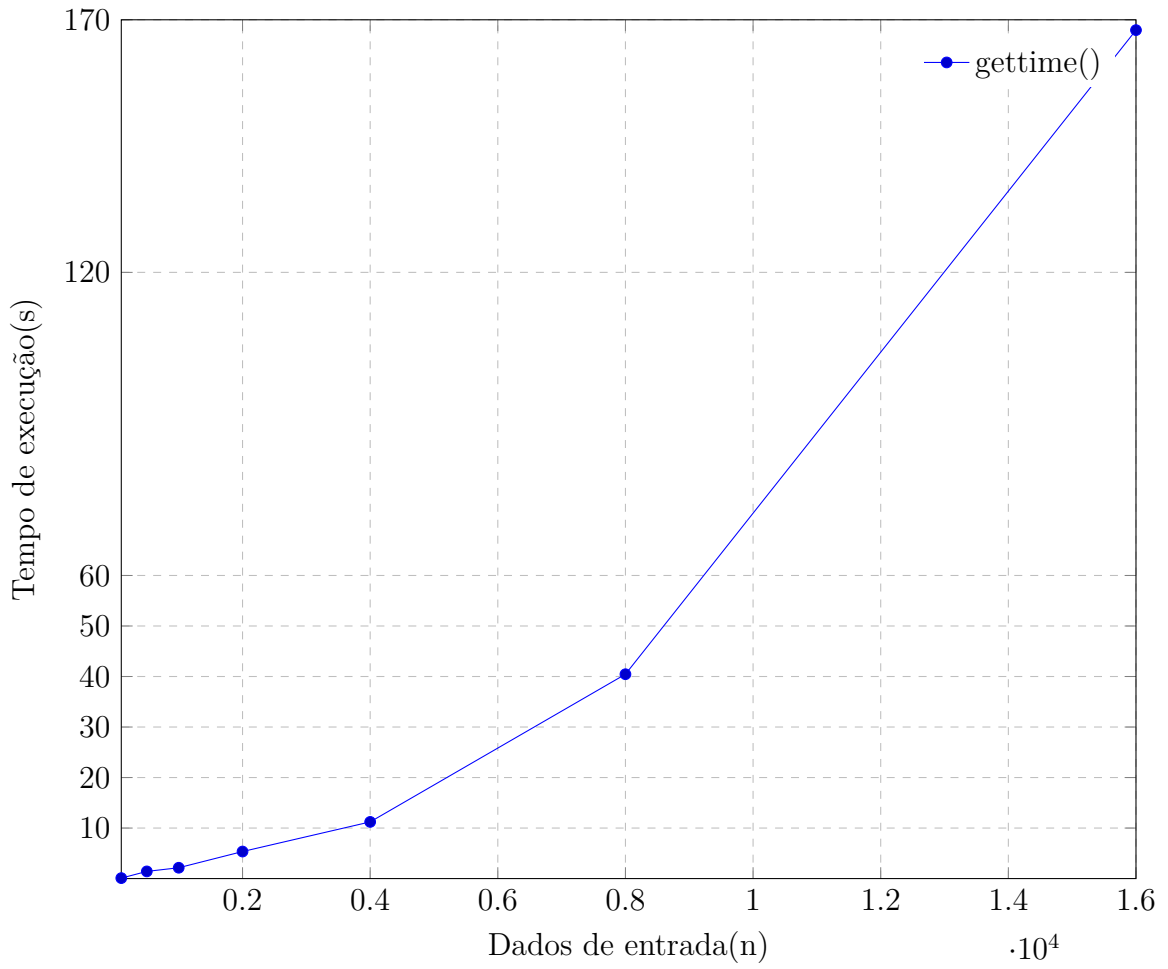
$$T(n) = aT(\frac{n}{b}) + f(n)$$

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

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

Fonte: autores

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



Fonte: autores

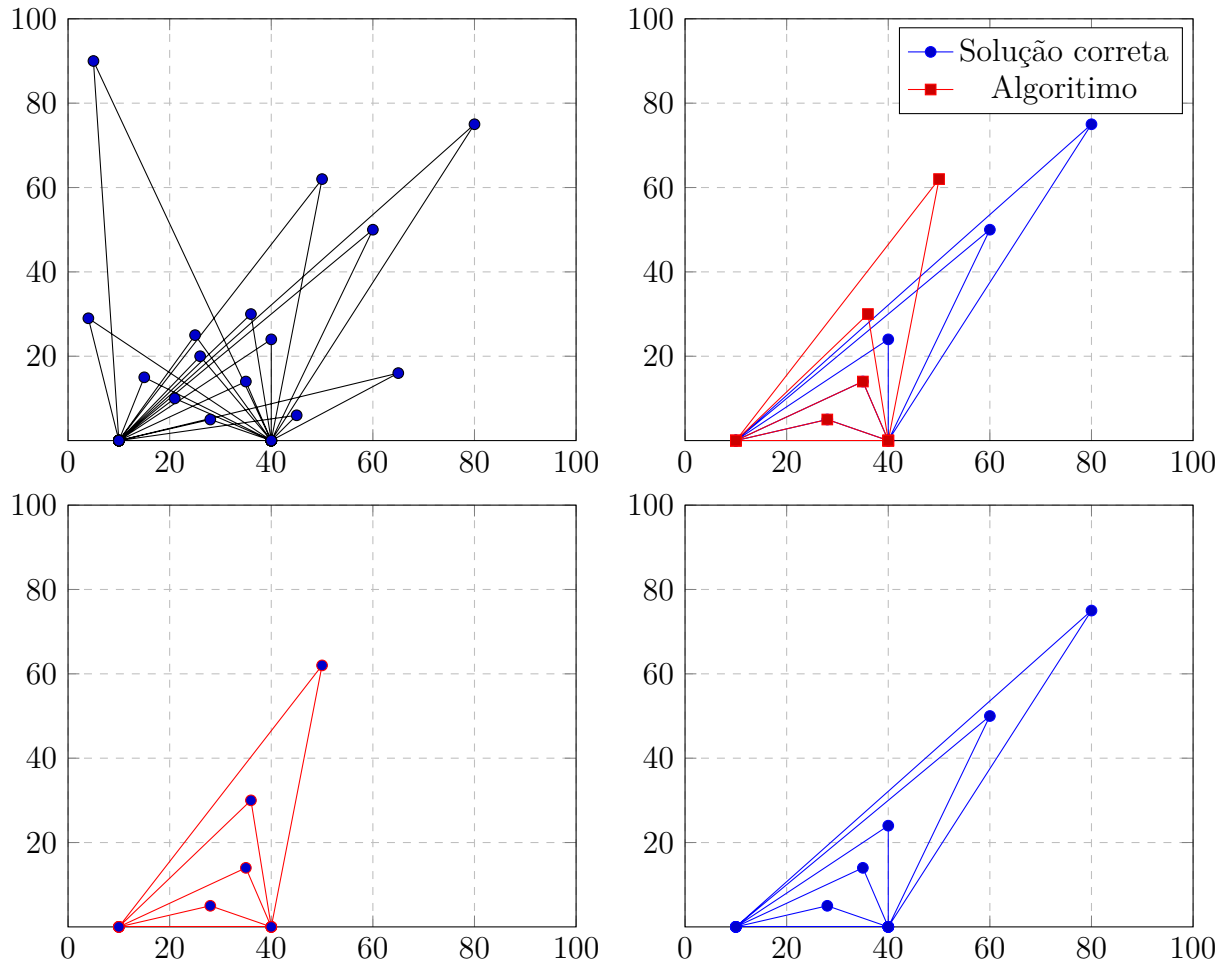


## 4 Considerações finais

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 LCS<sup>2</sup>, coordenadas baricênticas<sup>3</sup> 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.



<sup>2</sup>Longest 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.

<sup>3</sup>As Coordenadas Baricênticas 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.

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

## 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](https://en.wikipedia.org/w/index.php?title=Computational_geometry&oldid=841504892), 2018. [Acesso em: 3-Setembro-2018].