Apresentação Papermodel

Matheus H. P. Pacheco

Regras do modelo

- Utilizar Datasets em formato SVG (harry potter, avião e outros que encontrarmos)
- Inserir componentes em formato A4
- Distância mínima 5mm entre componentes
- Utilizar mínima quantidade de papel

O problema

- Um problema de natureza NP completo

- Conhecido também como nesting problem ou irregular packing problem.

O problema

- Existem diversas abordagens segundo a literatura;

- É muito comum trabalharmos com o método NFP(Not fit polygon), onde trabalhamos com algorítmos que verificam o conjunto de localizações onde polígonos não se sobreponham;

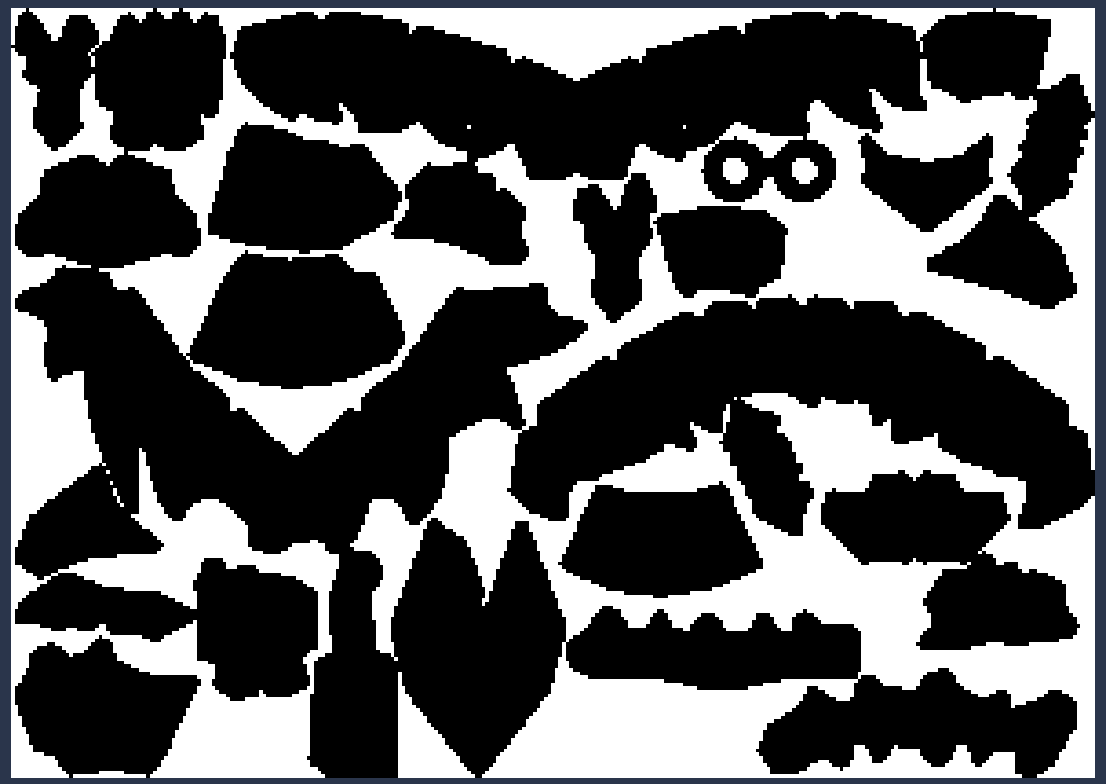
O problema

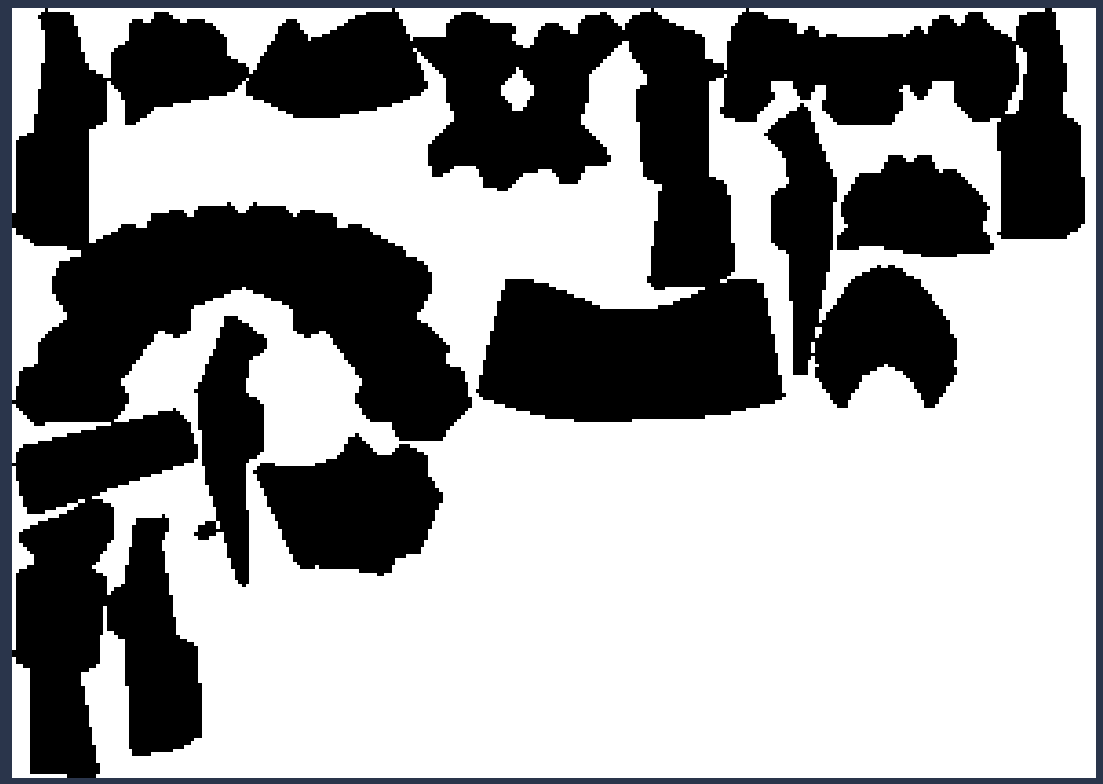
- Em aula o professor passou algumas soluções de outros alunos (empacotamento de retângulos, blob, time series, convex hull)

- Duas heurísticas me vieram a cabeça: A abordagem probabilística olhando para uma função de custo e séries temporais;

O Dataset

Melhor abordagem Anterior - Função de custo





Ideia

- Baseado na ideia do Williamson, transformar o SVG em vários polígonos e tratar eles como séries temporais.
- Utilizar o algorítmo DTW para comparar duas séries temporais.
- Testar a heurística com rotação, translação e verificação de colisões

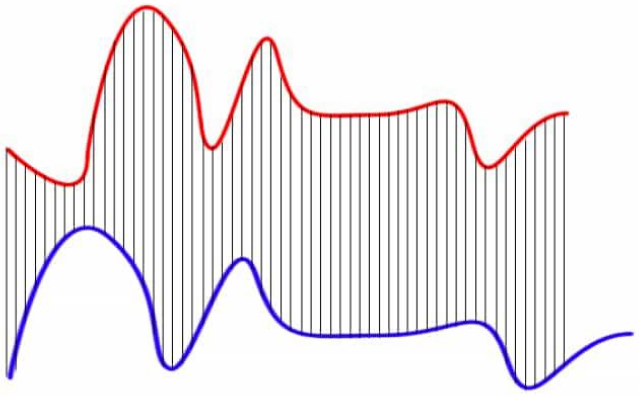
Ideia

- Como não havia o código de exemplo, foi feito a heurística do zero partindo de algumas bibliotecas do python;

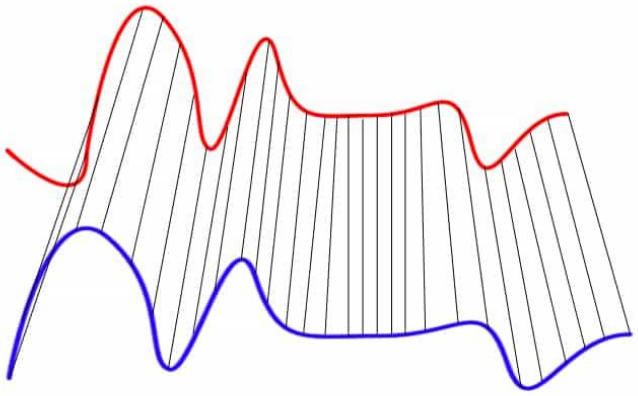
DTW – Dynamic Time Wraping

- **Dynamic time warping** (DTW) é um algoritmo para comparar e alinhar duas séries temporais. A DTW é utilizada para encontrar o alinhamento não-linear ótimo entre duas sequências de valores numéricos. Dessa maneira, é possível encontrar padrões entre medições de eventos com diferentes ritmos. Por exemplo, é possível casar a série temporal obtida por acelerômetros (ou outros sensores) de duas pessoas andando em diferentes velocidades.

DTW - Dynamic Time Wraping



Euclidean Matching



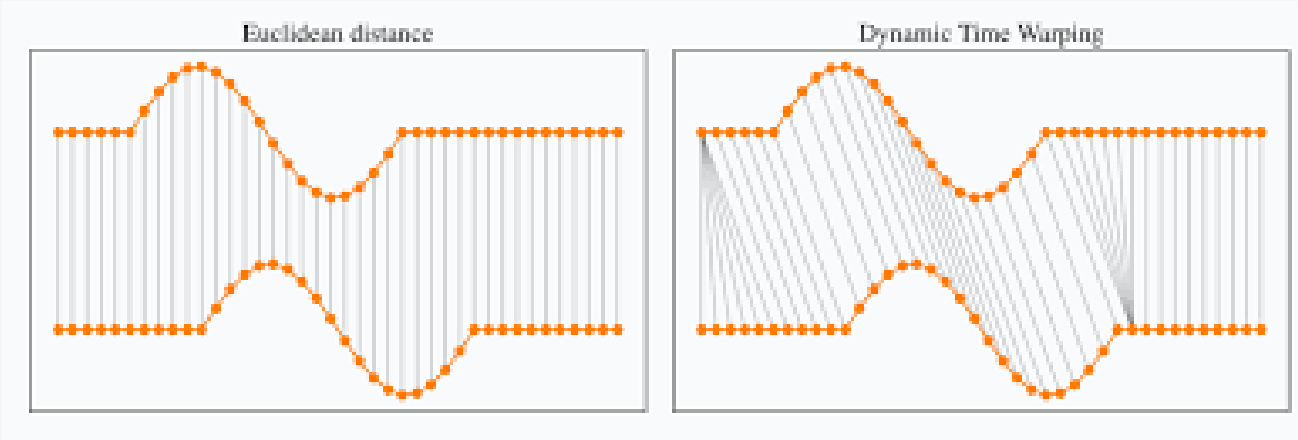
Dynamic Time Warping Matching

```
int DTWDistance(s: array [1..n], t: array [1..m]) {
    DTW := array [0..n, 0..m]

    for i := 1 to n
        DTW[i, 0] := infinity
    for i := 1 to m
        DTW[0, i] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost := d(s[i], t[j<nowiki>])</nowiki>
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i  , j-1],    // deletion
                                        DTW[i-1, j-1])    // match

    return DTW[n, m]
}
```

Algorithm 2: Pseudocode for forward DTW

Input : time series vectors $X = (x_1, x_2, ..., x_N), Y = (y_1, y_2, ..., y_N = M)$
Output: minimum distance

$d(x, y) = |x - y|$
for $i = 0$ **to** N **do**
  $DTW[i, 0] := \infty$
end
for $i = 0$ **to** M **do**
  $DTW[0, i] := \infty$
end
$DTW[0, 0] := 0$
for $i = 1$ **to** $N - 1$ **do**
  for $j = i + 1$ **to** M **do**
    $DTW[i][j] := d(X[i], Y[j]) + min(DTW[i-1][j-1], DTW[i-1][j])$
  end
end
return DTW[n, m]



Bibliotecas utilizadas

- Svgpathtools – Trabalhar com svg
- Shapely – transformar as imagens em polígonos
- Fastdtw – tranbalhar com o dtw
- Scipy – trabalhar com dist eucliadiana
- Numpy - manipulação de arrays
- Matplotlib – biblioteca grafica
- Os – lidar com diretorios
- Itertools – ferramenta de manipulação de estruturas de dados

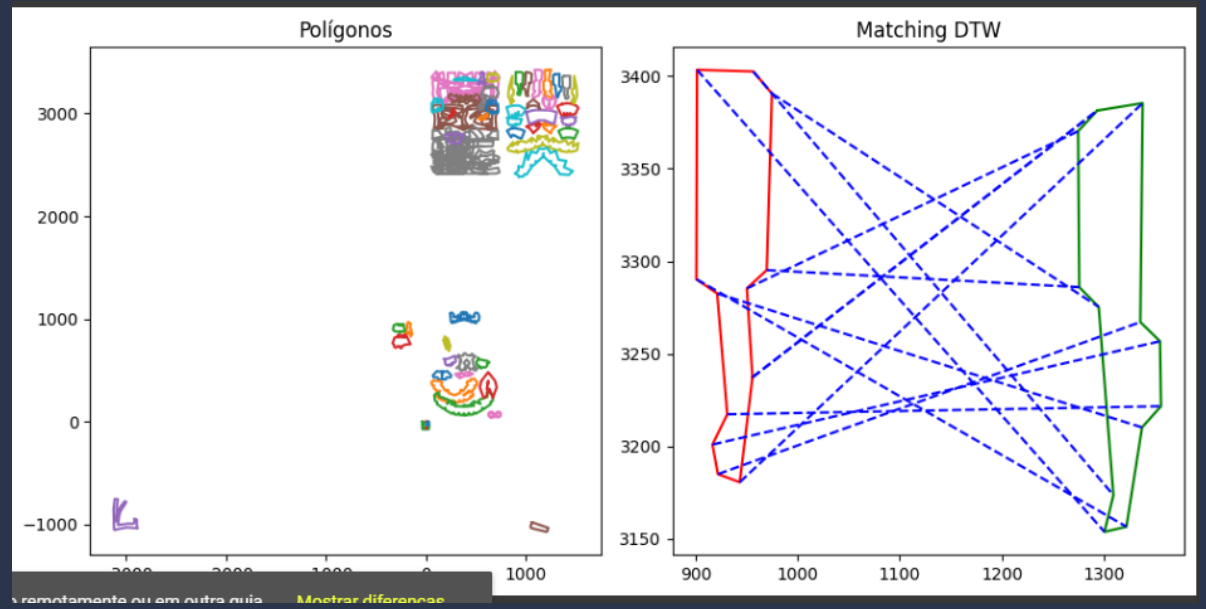
Bibliotecas utilizadas

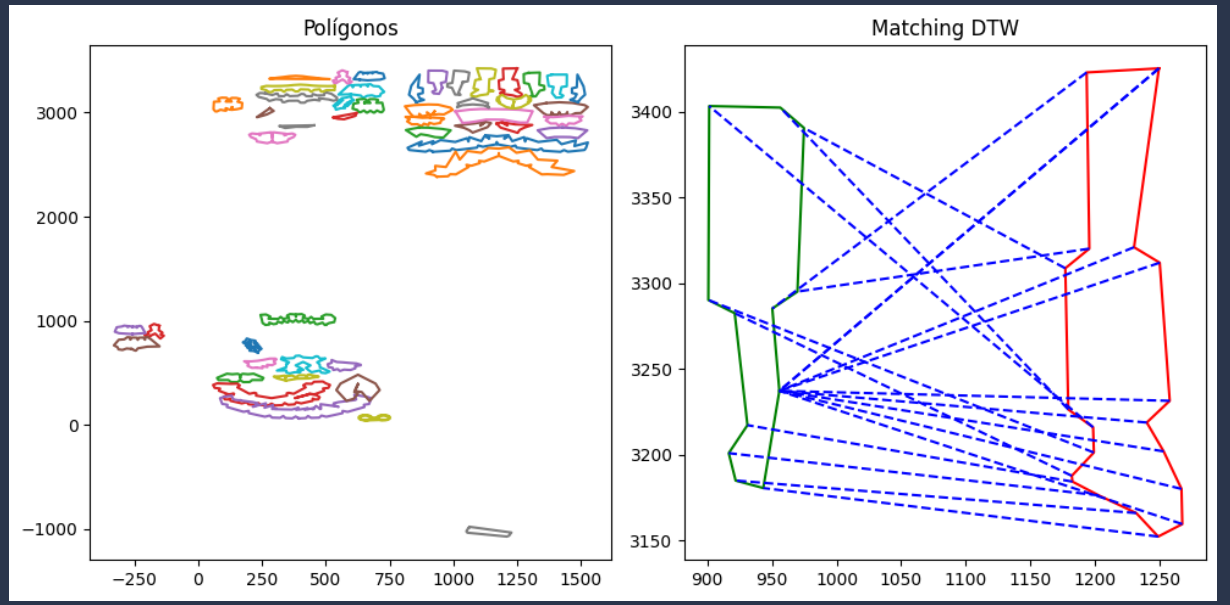
- Svgpathtools – Trabalhar com svg
- Shapely – transformar as imagens em polígonos
- Fastdtw – tranbalhar com o dtw
- Scipy – trabalhar com dist eucliadiana
- Numpy - manipulação de arrays
- Matplotlib – biblioteca grafica
- Os – lidar com diretorios
- Itertools – ferramenta de manipulação de estruturas de dados

Hand's On e Resultados

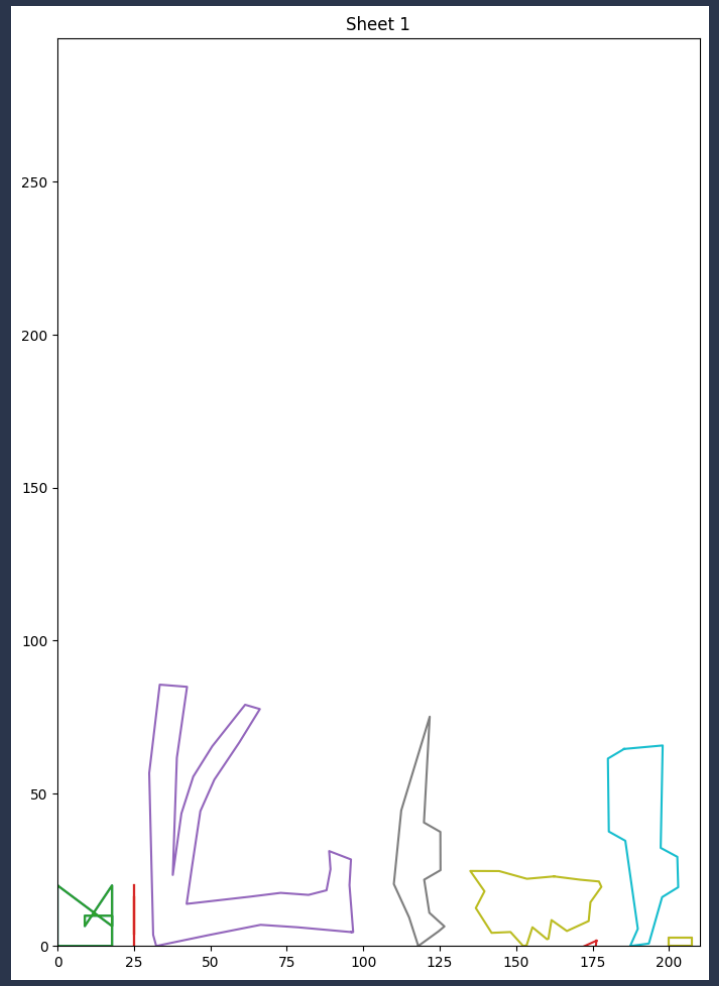
Github - papermodel_nest2d

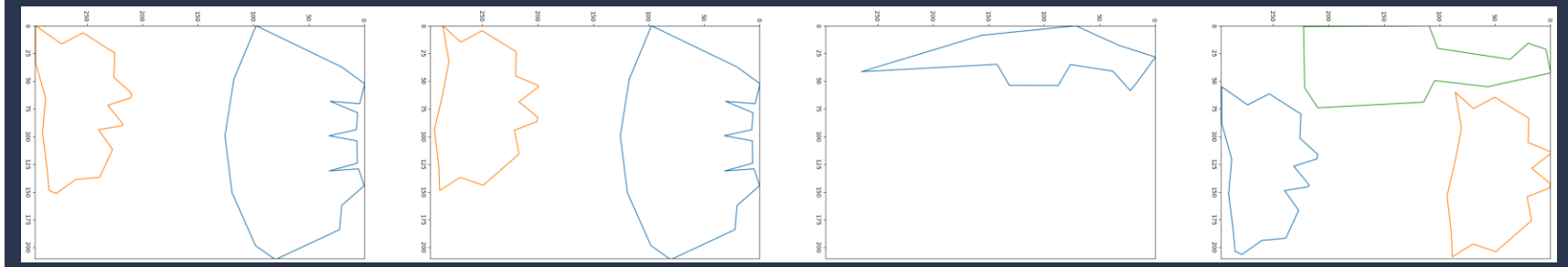
Implementação do DTW (Séries Temporais)



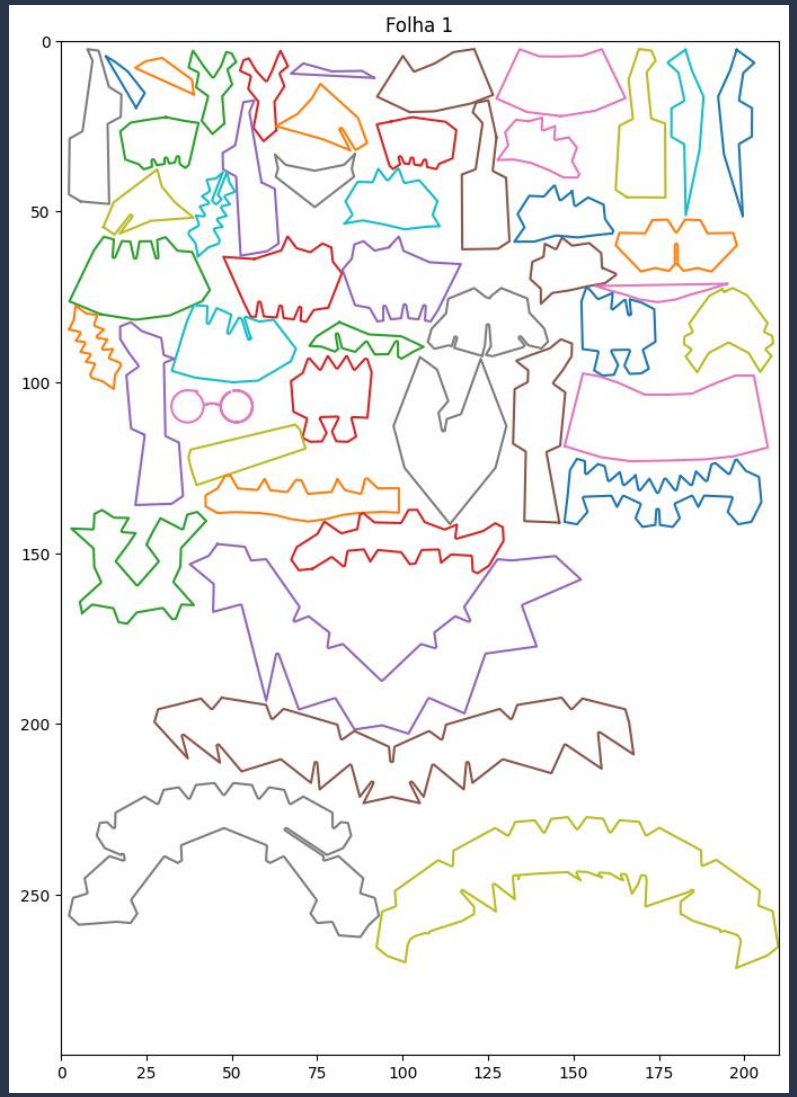


Implementação do DTW (Séries Temporais)

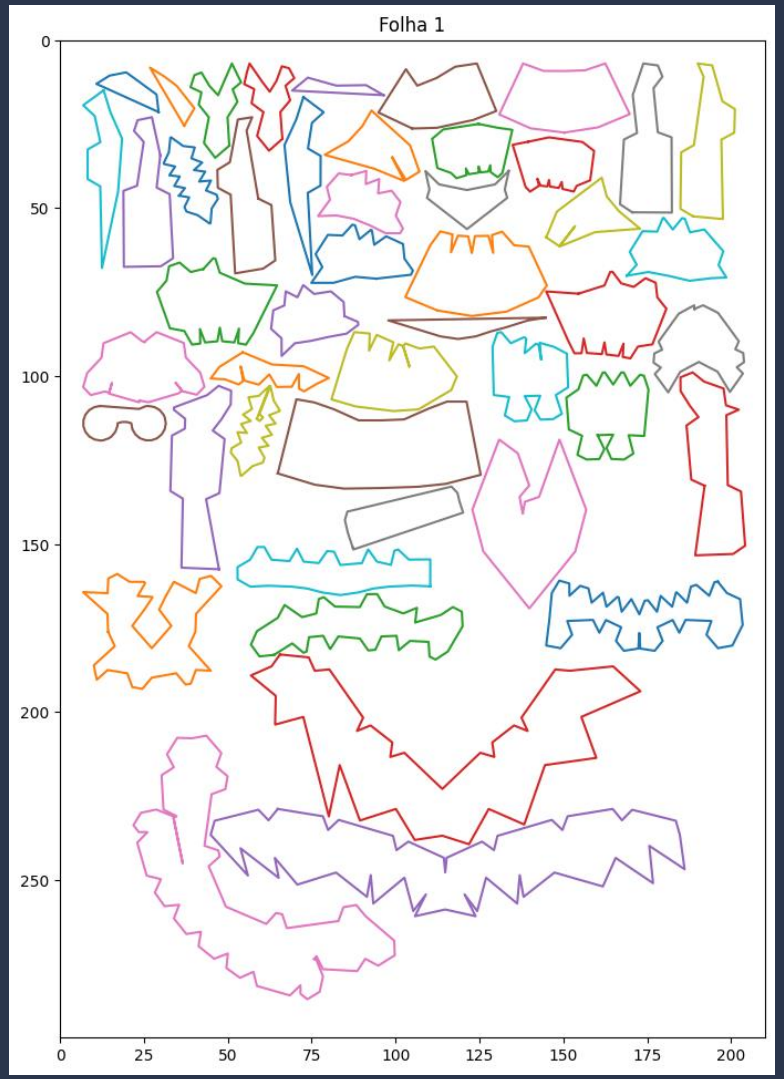


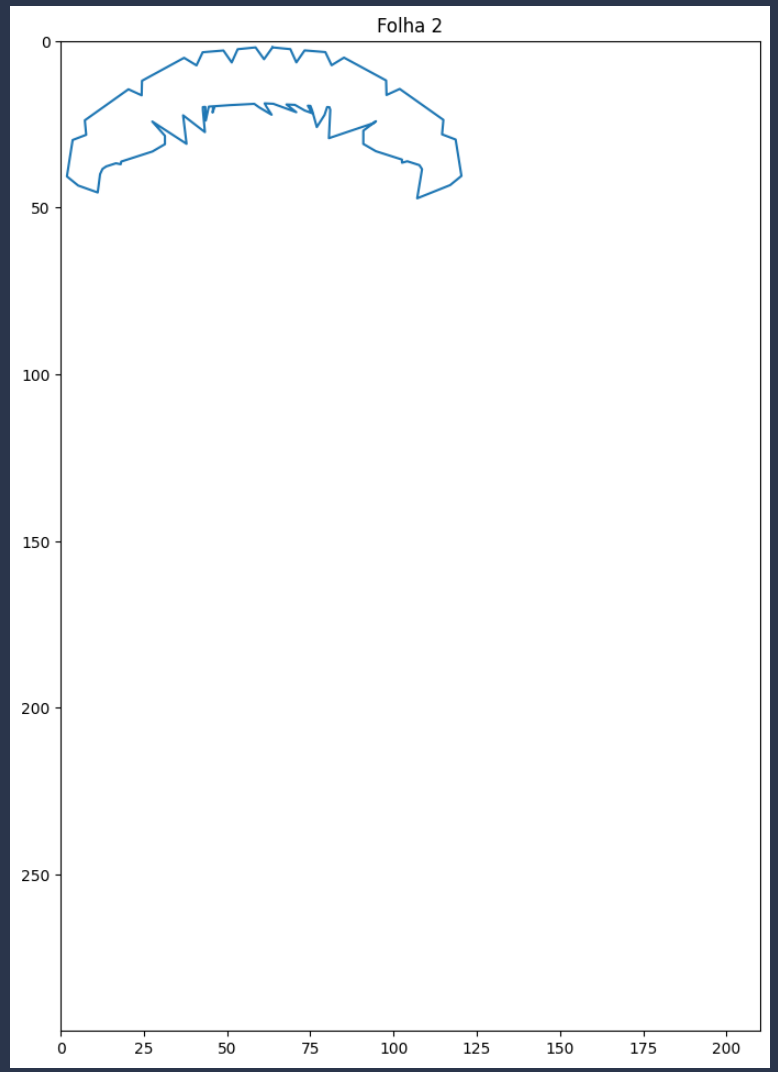


Principais resultados: dtw_1.py



Principais resultados: dtw_2.py





Próximos passos

- Utilizar outras bibliotecas e deep learning para comparar resultados;
- Utilizar variações do DTW;
- Melhorar a função da heurística find_position( ) para que ela suporte mais ajustes finos como: testar origens diferentes, testar mais blobs de peças.
- Alinhas com os professores e williamson para futuramente publicar a ideia.

Obrigado!