Reconnaissance des empreintes digitales

TIPE DE CARON RENAUD 2021 (N° SCEI : 5792)

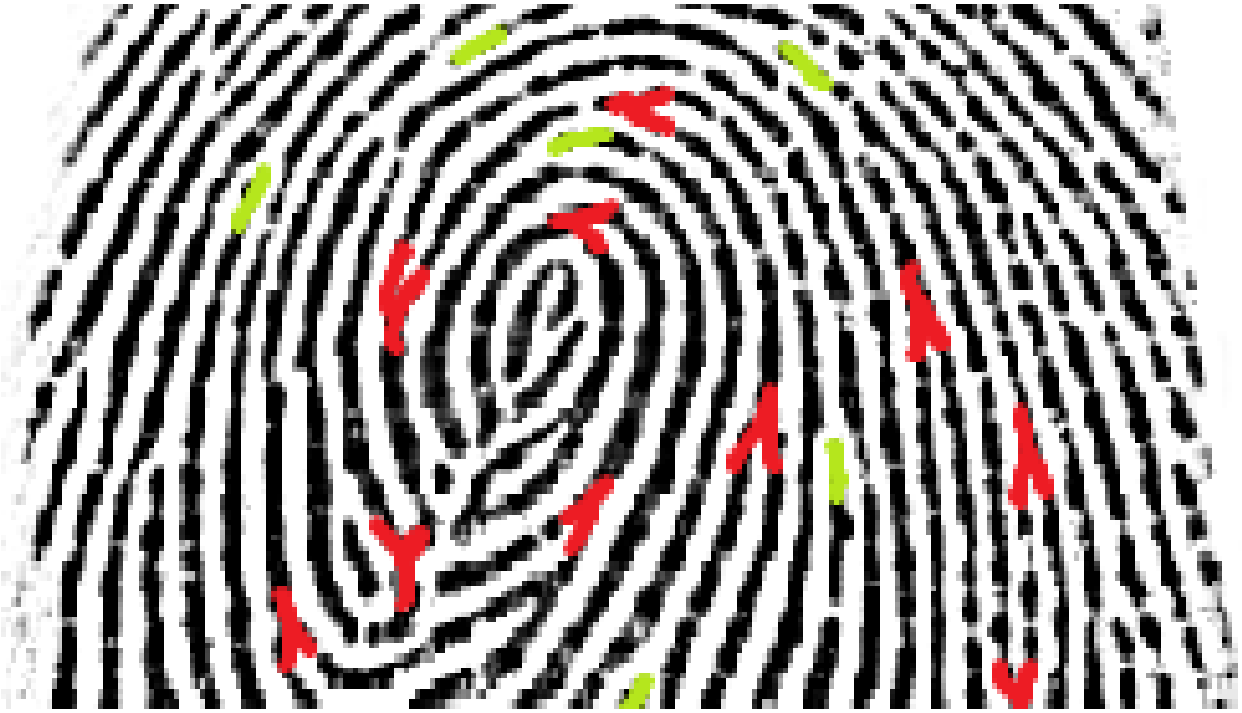
Sommaire

1) Extraction d’informations

2) Algorithme de comparaison

3) Résultats

Minuties

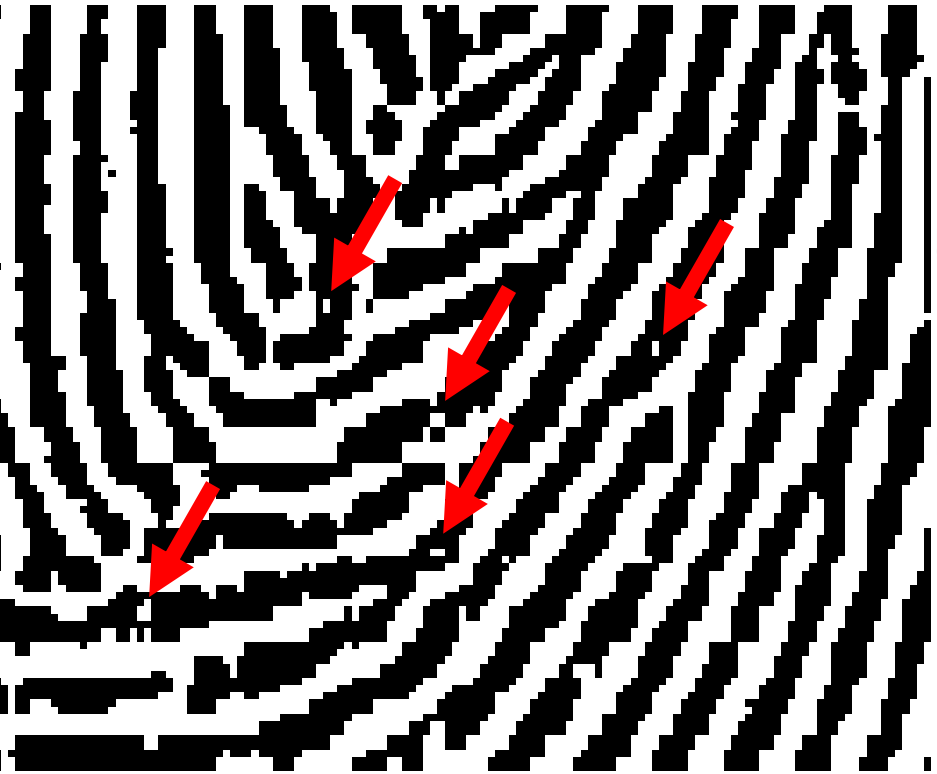


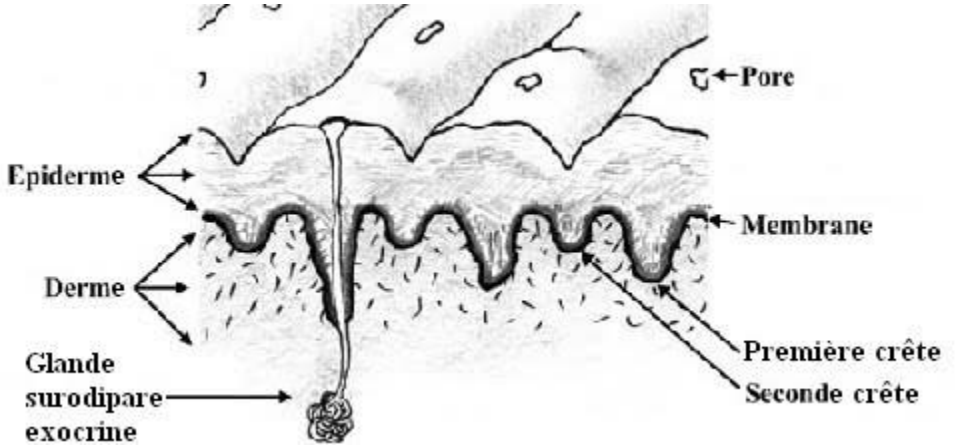
Intersection

Terminaison

Binarisation

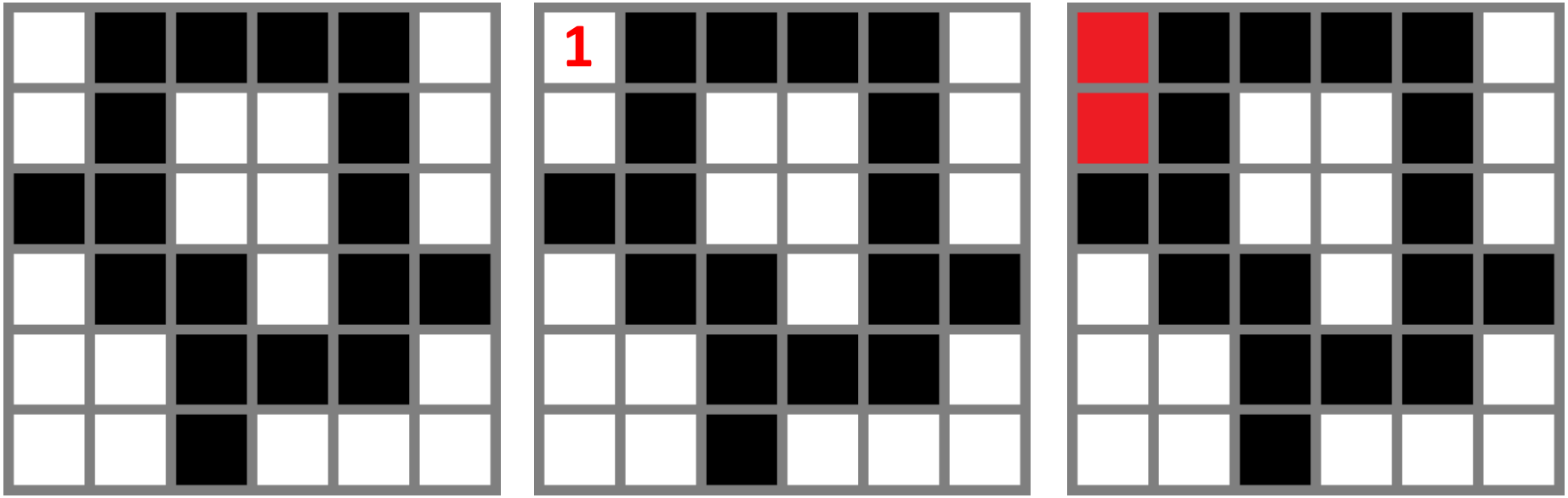
Pores





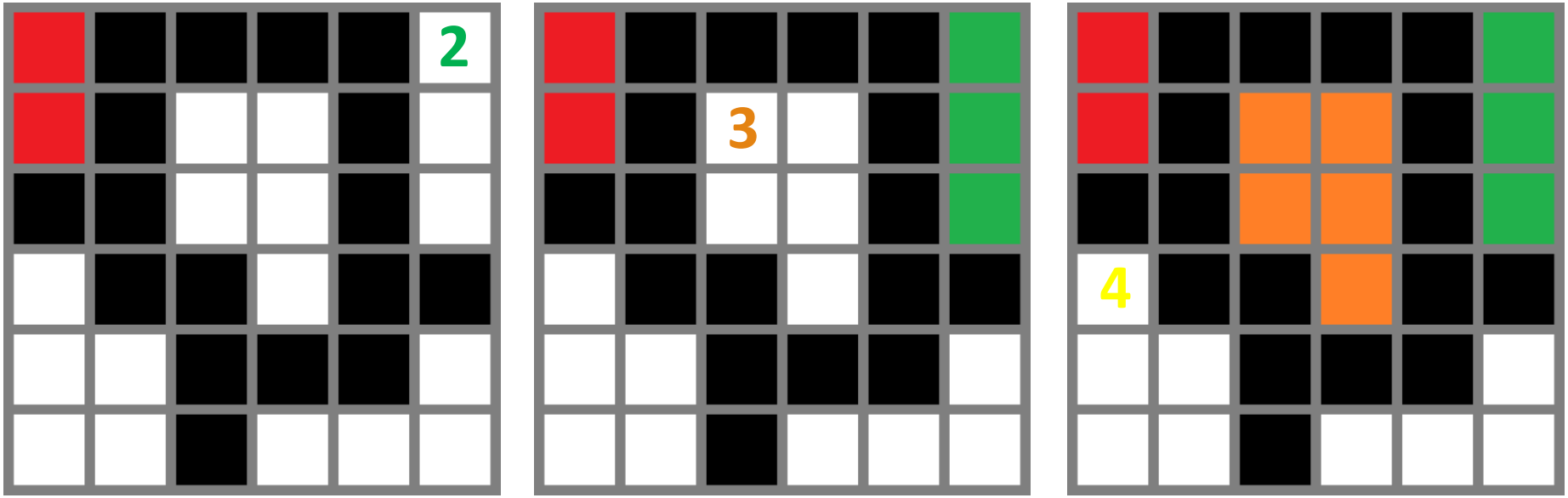
Coupe de la peau montrant les deux couches de dermatoglyphe - Vincent Fleury

Suppression des pores



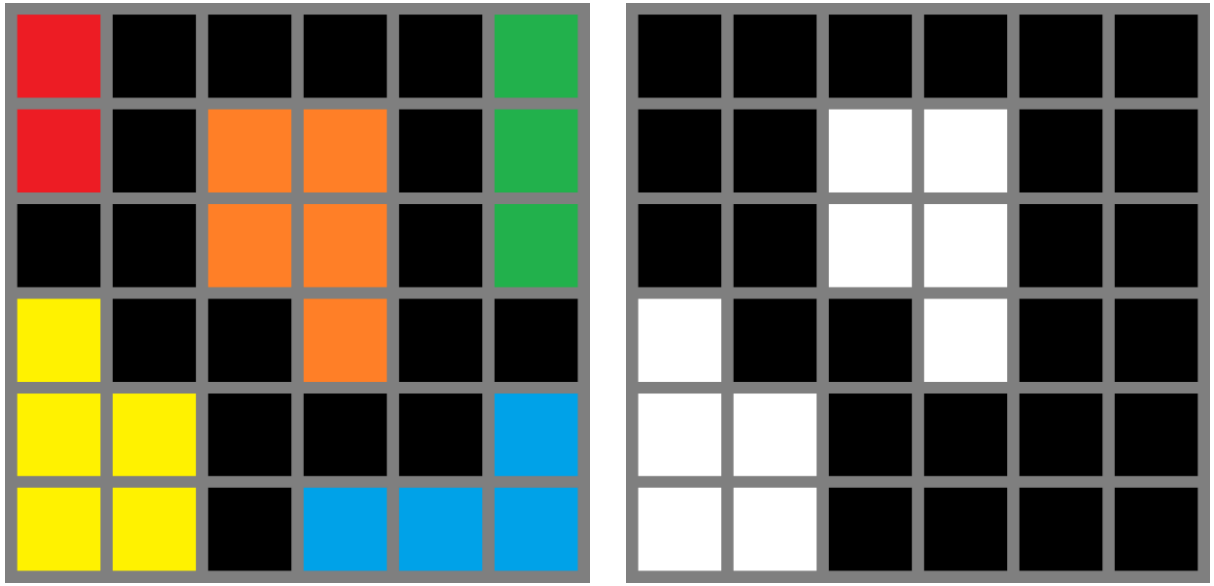
Rouge : {(0,0),(0,1)}

Suppression des pores



Vert : {(5,0),(5,1),(5,2)} …

Suppression des pores



Aire minimum = 4

Suppression des pores

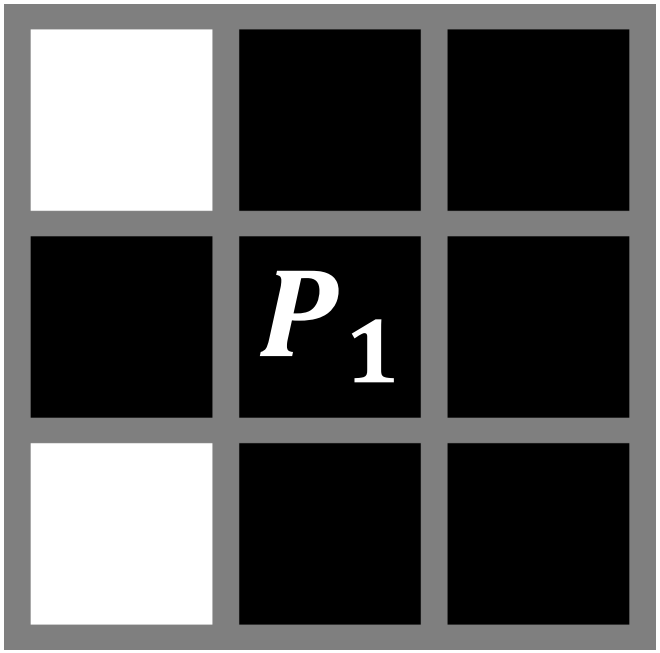
Amincissement : Zhang-Suen

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

A(P_1) = Nombre de transitions Blanc -> Noir dans (P_2,P_3),(P_3,P_4),…,(P_9,P_2)

B(P_1) = Nombre de pixels noirs dans (P_1,P_2,…,P_9)

Amincissement : Zhang-Suen



A(P_1) = 2

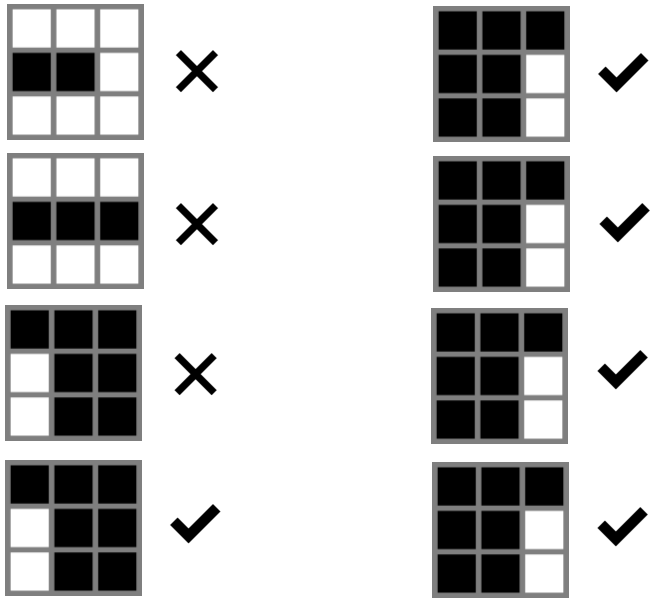
B(P_1) = 6

Amincissement : Zhang-Suen

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

Étape 1: $\forall P_1 \in$ Image

- P_1 est noir et a 8 voisins
- $2 <= B(P_1) <= 6$
- $A(P_1) = 1$
- P_2 ou P_4 ou P_6 est blanc
- P_4 ou P_6 ou P_8 est blanc

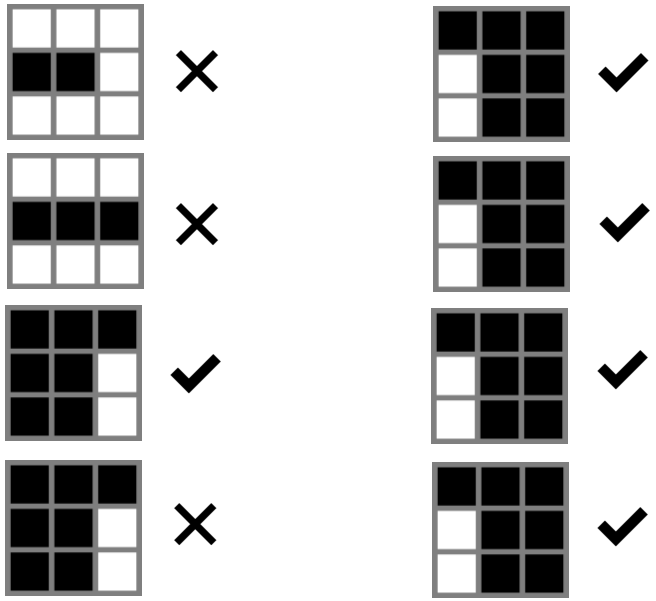


Amincissement : Zhang-Suen

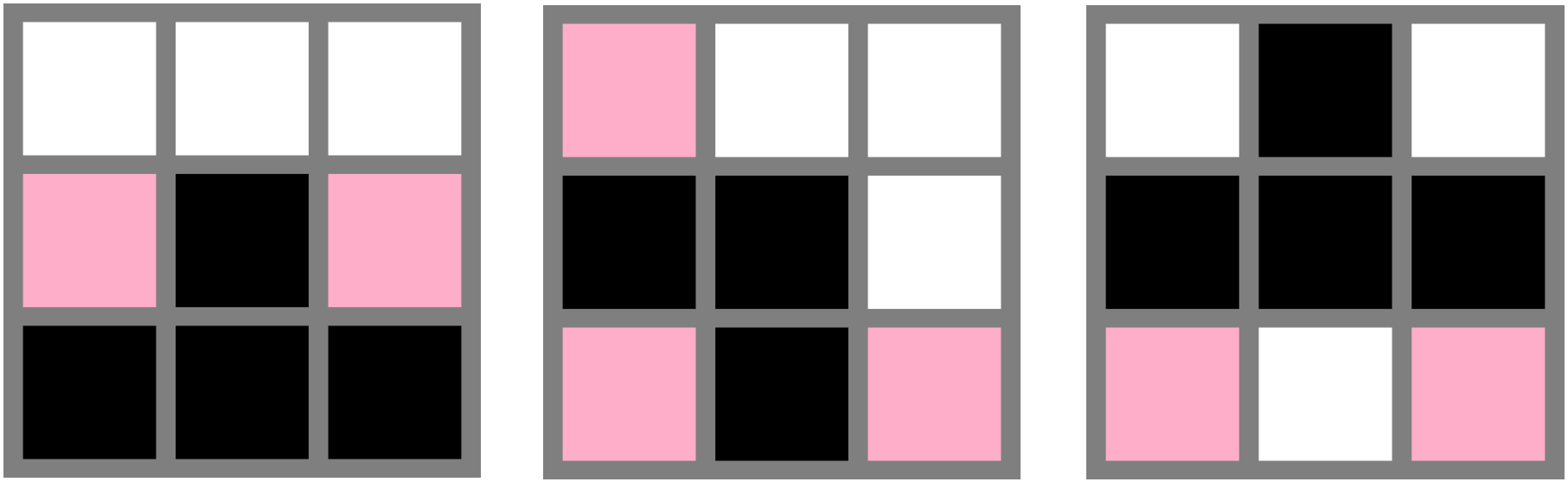
P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

Étape 2: $\forall P_1 \in$ Image

- P_1 est noir et a 8 voisins
- $2 <= B(P_1) <= 6$
- $A(P_1) = 1$
- P_2 ou P_4 ou **P_8** est blanc
- **P_2** ou P_6 ou P_8 est blanc

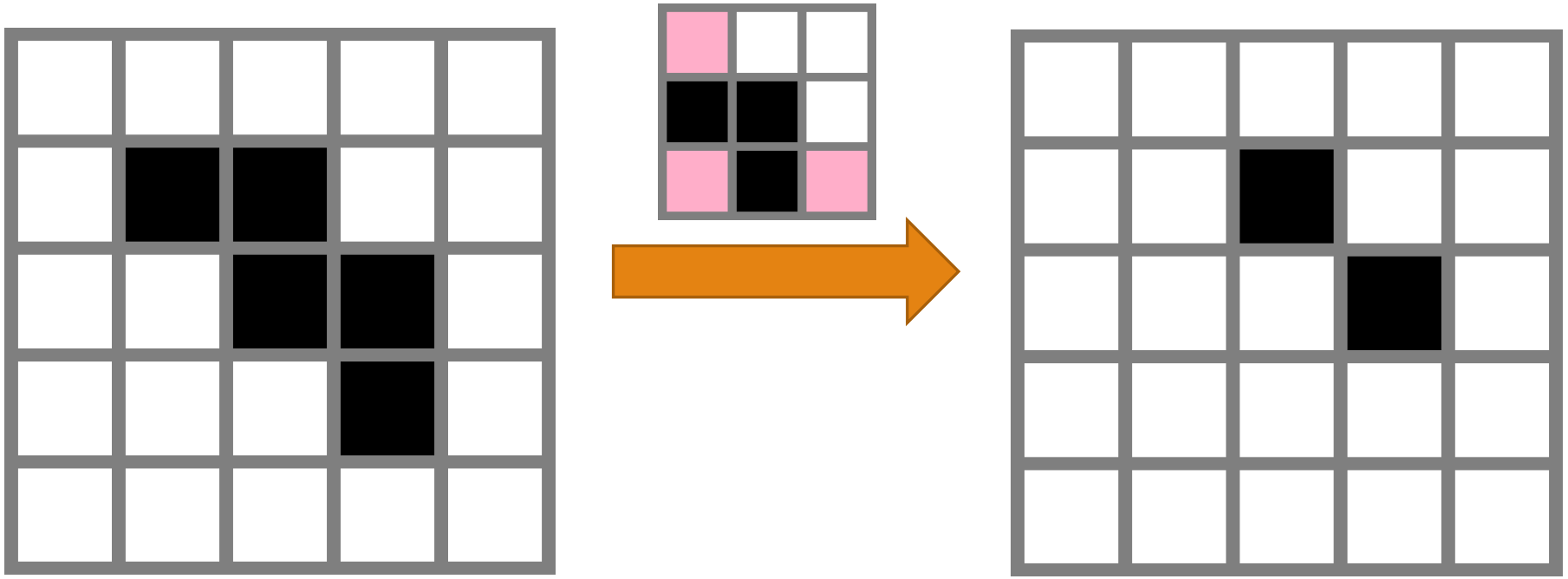


Amincissement : 2ème étape



Couleur quelconque

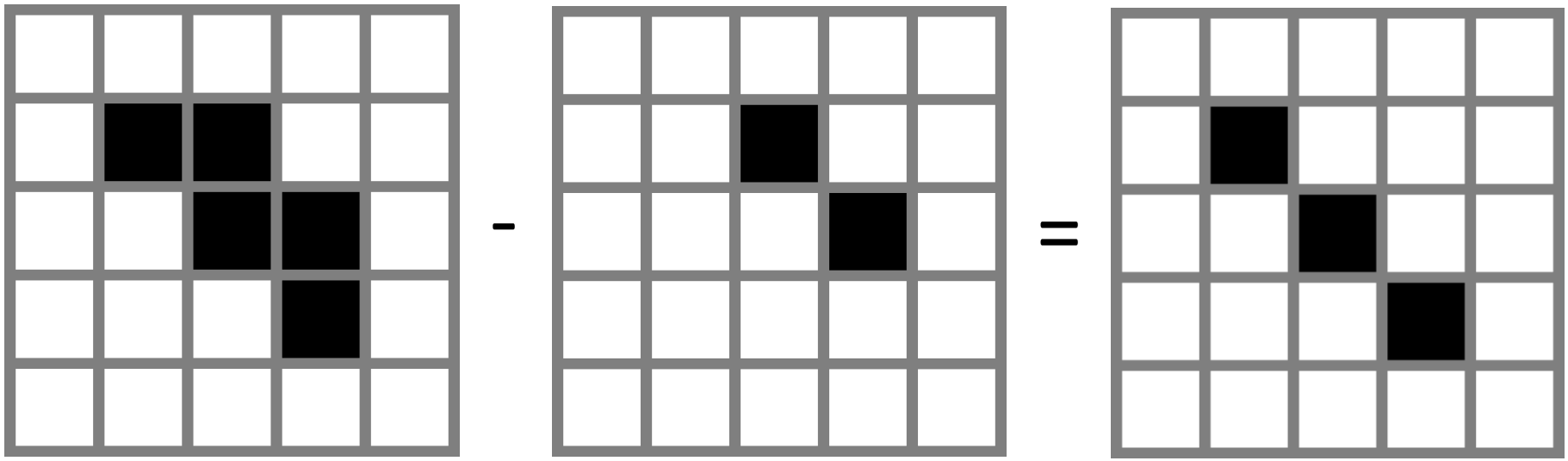
Amincissement : 2ème étape



Image

Correspondance du masque

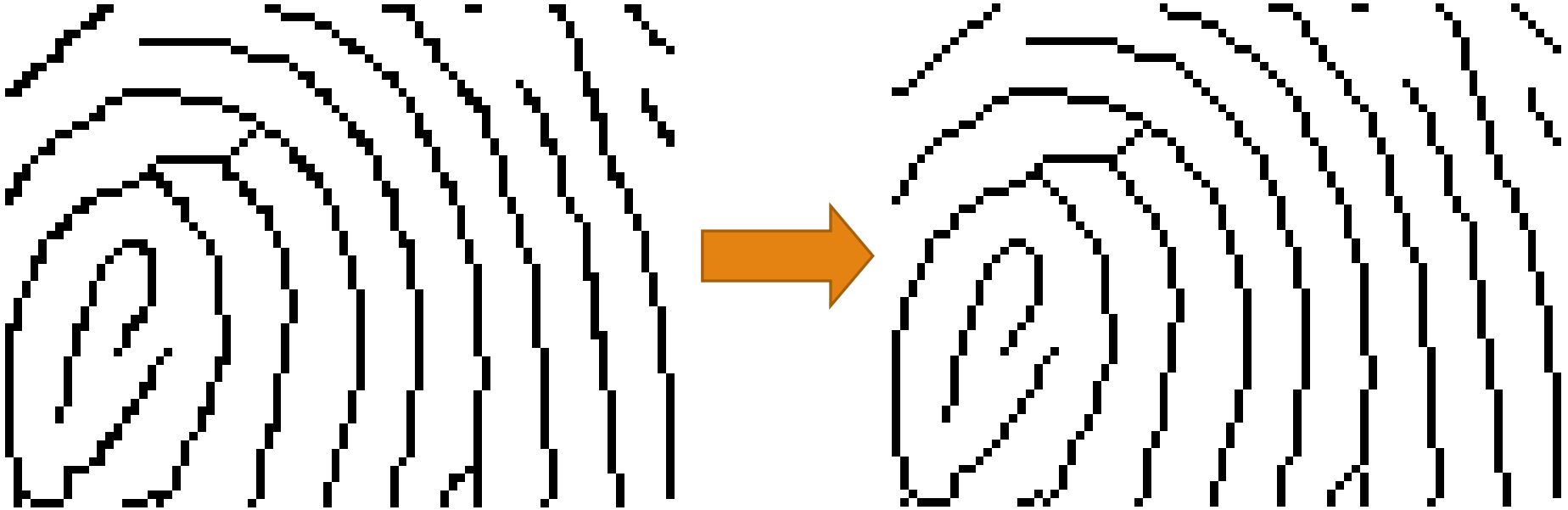
Amincissement : 2ème étape



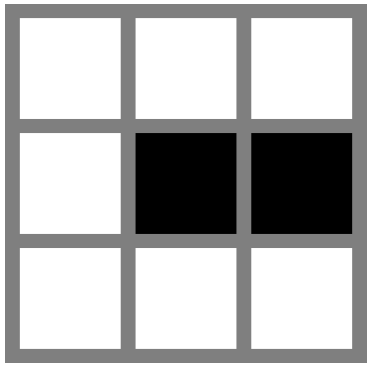
Nouvelle image

Amincissement : 2ème étape

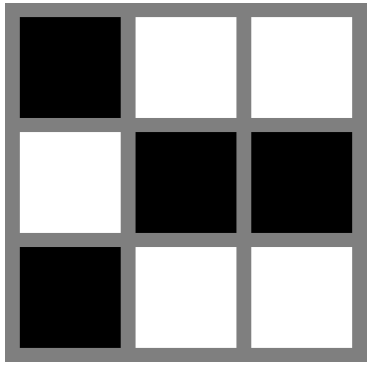
Amincissement : 2ème étape

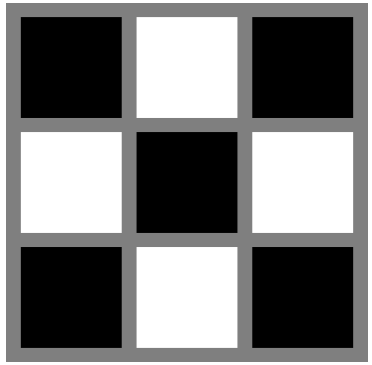


Caractérisation des minuties



Somme = 2 -> Terminaison



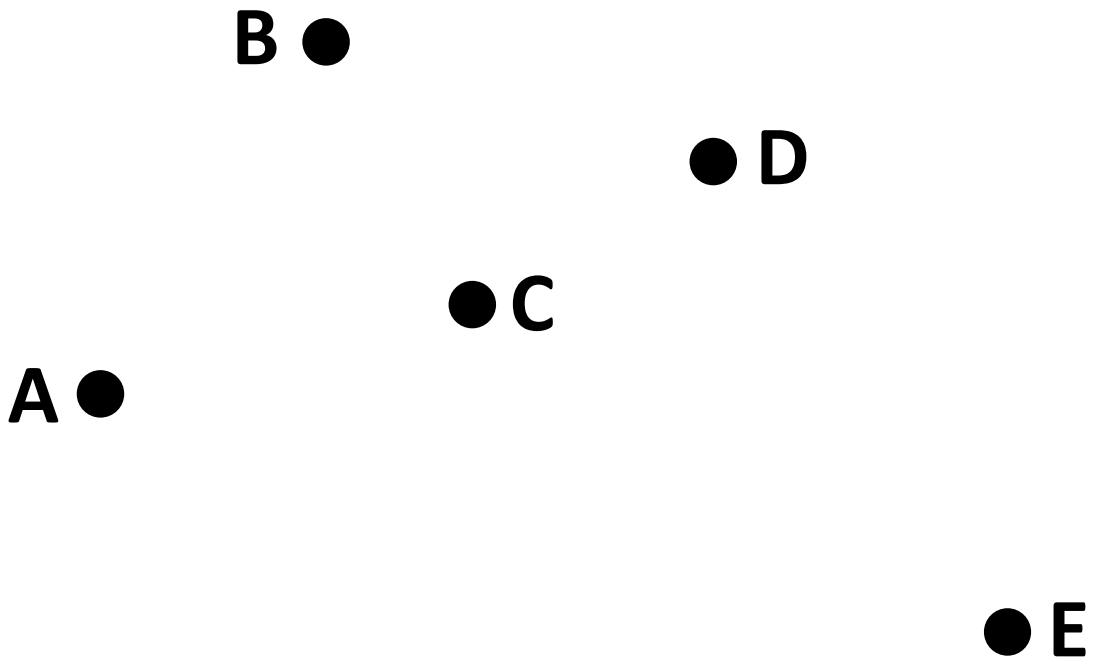


Somme ≥ 4 -> Intersection

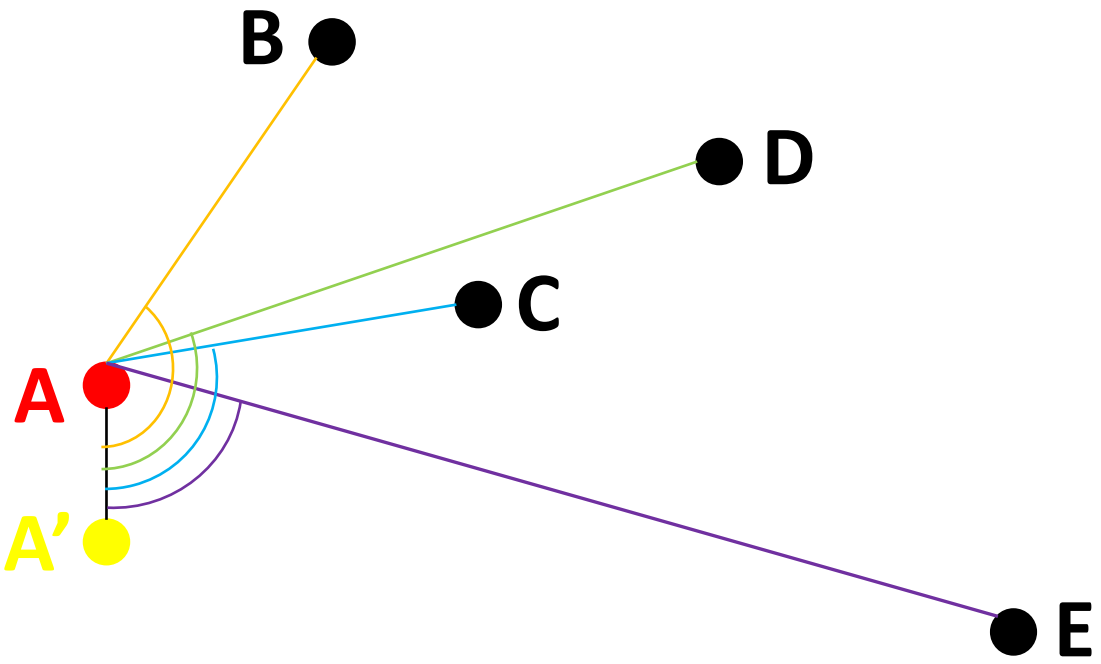
Problème : bord de l’image

Enveloppe convexe : candidats

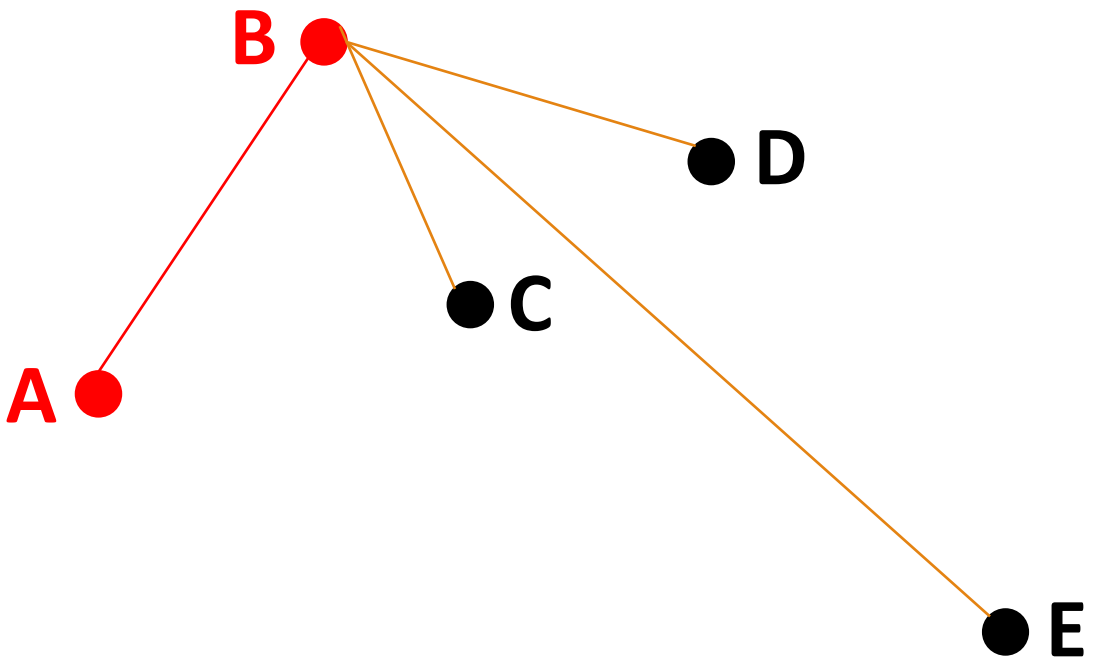
Marche de Jarvis



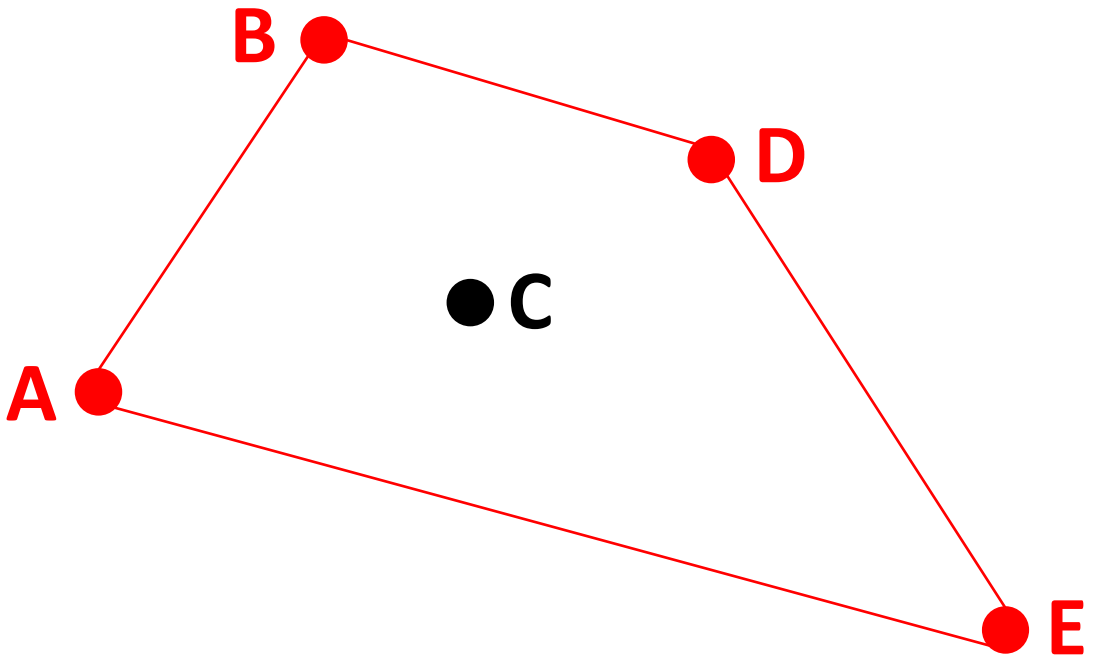
Marche de Jarvis



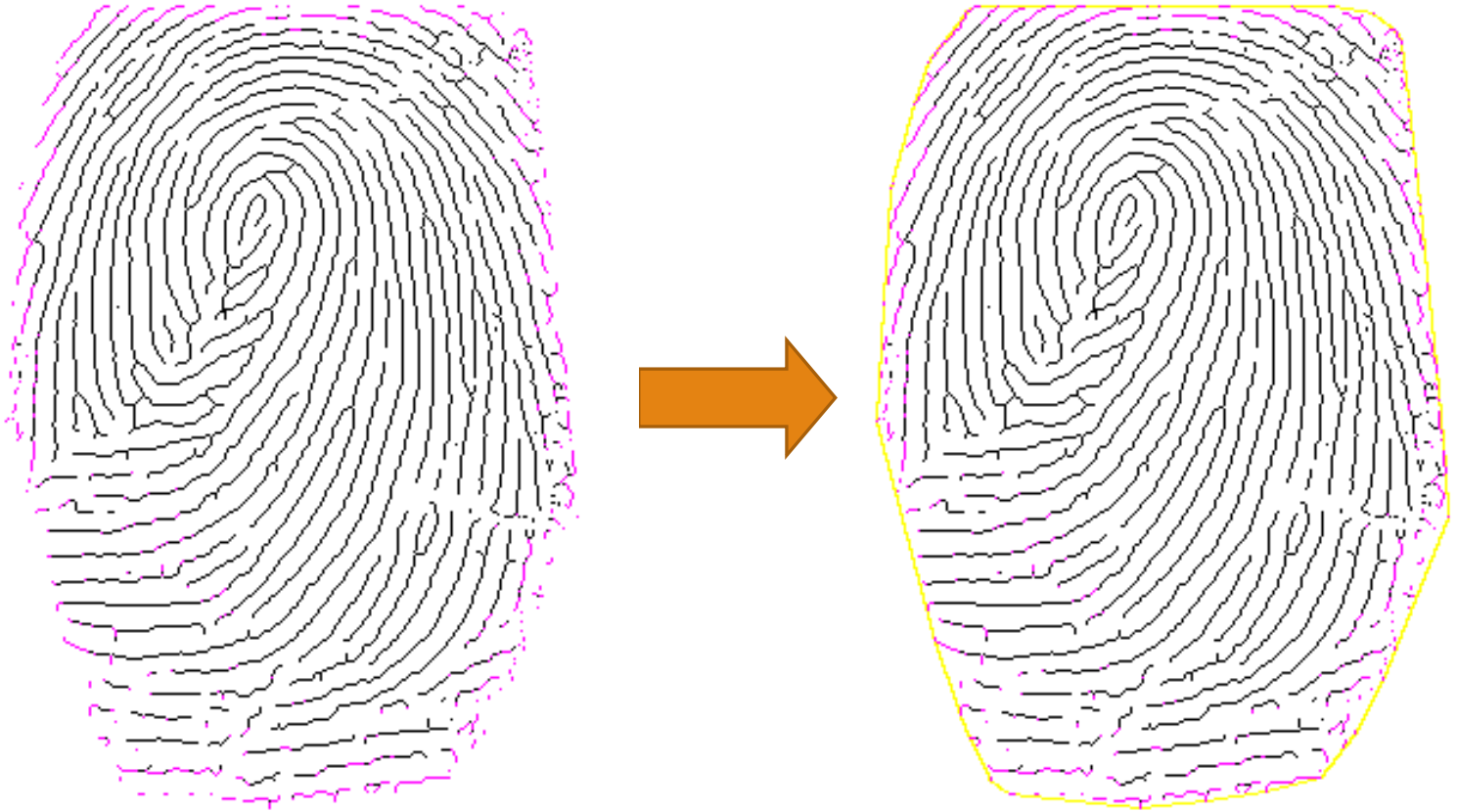
Marche de Jarvis



Marche de Jarvis



Marche de Jarvis



Barycentre d'un polygone

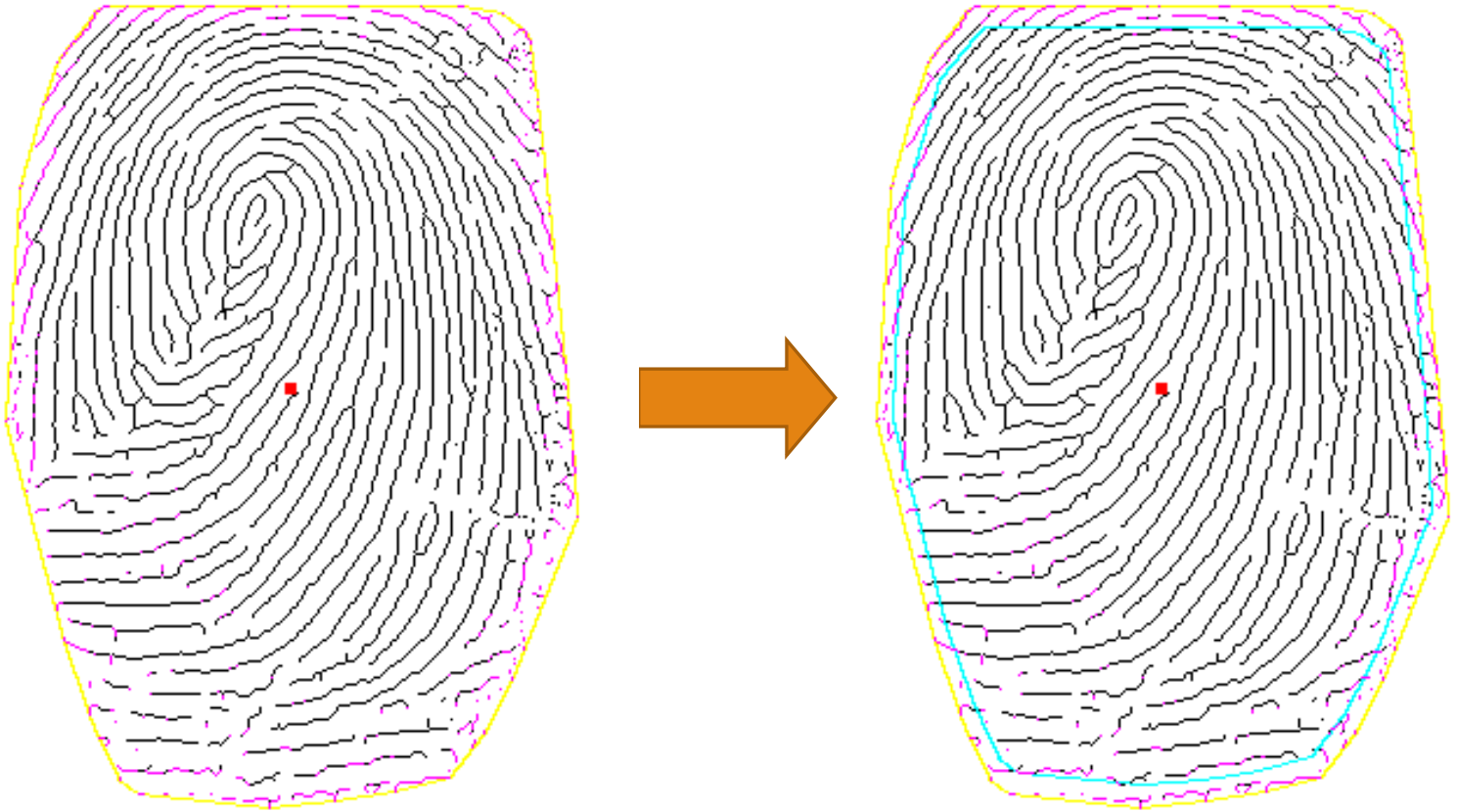
Soit P un polygone à n sommets (x_0, y_0), …, (x_{n-1},y_{n-1}), (C_x,C_y) son barycentre, et A son aire :

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i\, y_{i+1} - x_{i+1}\, y_i)$$

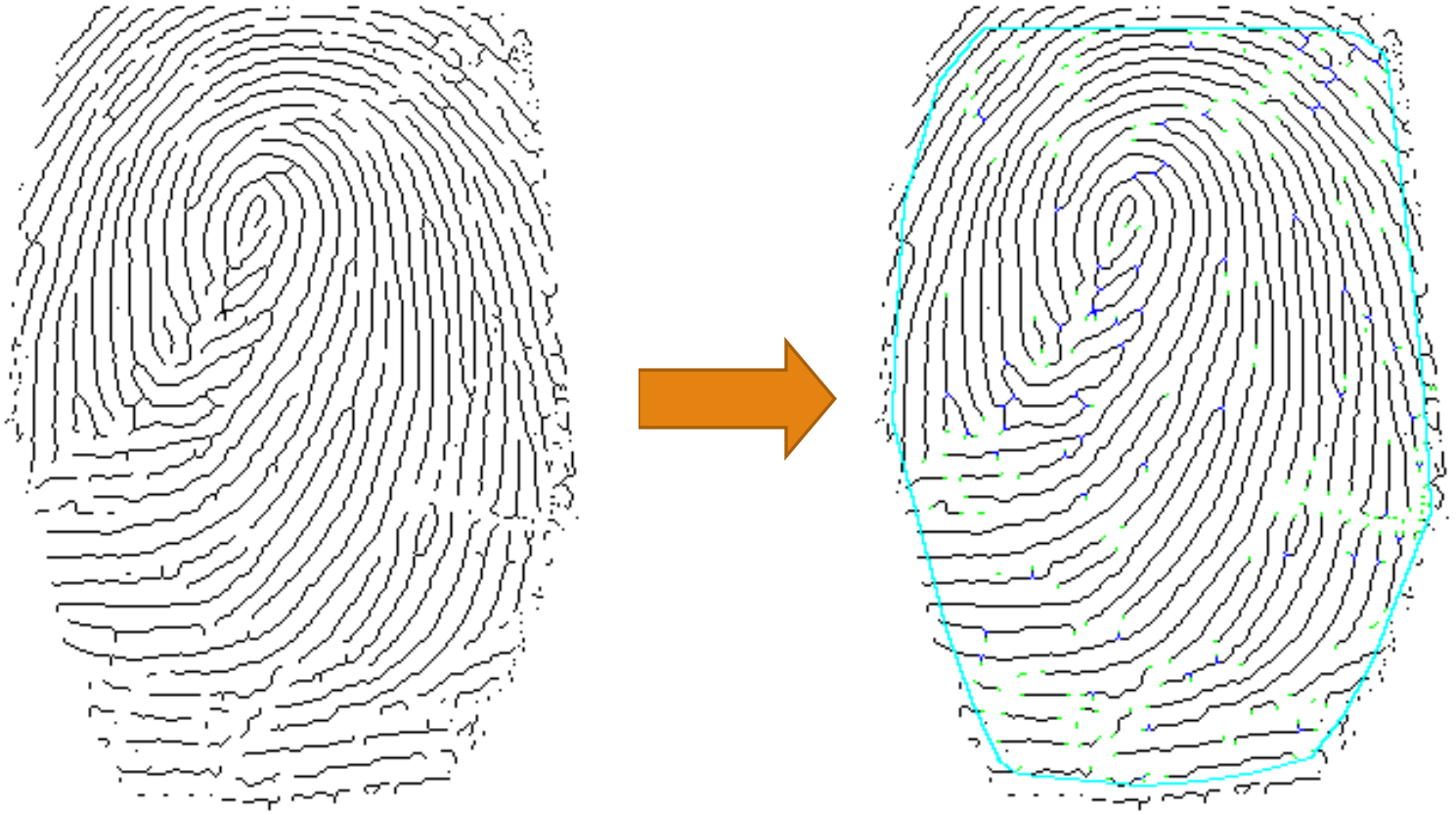
$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i\, x_{i+1})(x_i\, y_{i+1} - x_{i+1} y_i)$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i\, y_{i+1})(x_i\, y_{i+1} - x_{i+1} y_i)$$

Zone de recherche



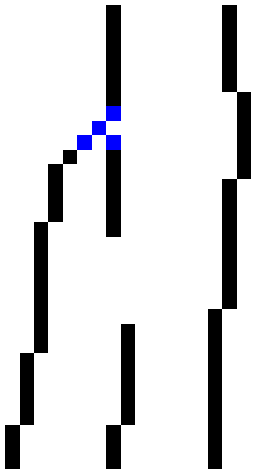
Extraction des minuties

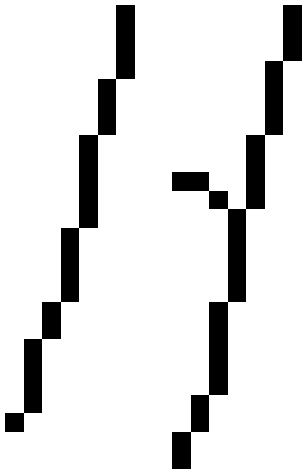


Tri supplémentaire

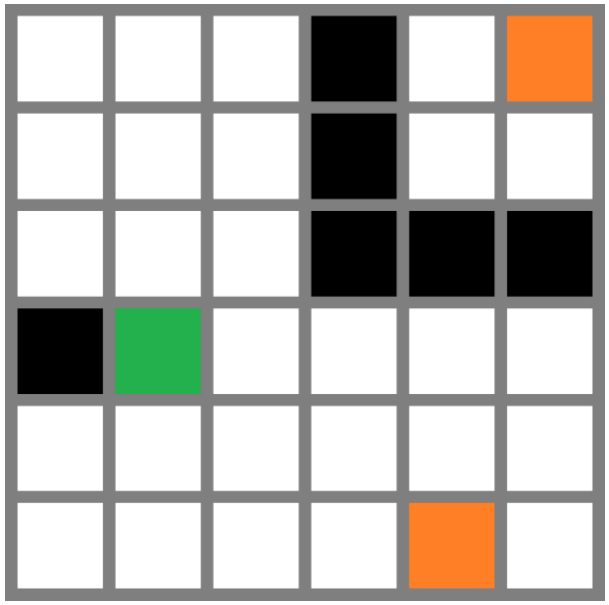
1) Suppression des intersections proches reliées

2) Suppression des intersections proches d'une terminaison non reliable à une autre terminaison





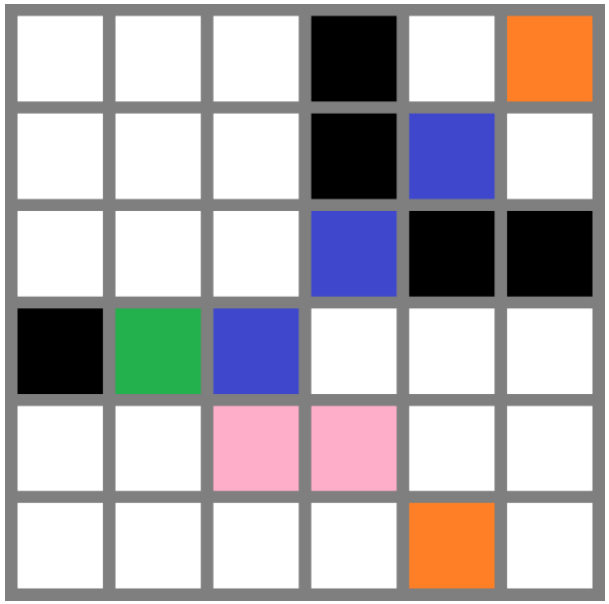
Caractère reliable



Terminaison à relier

Terminaisons proches

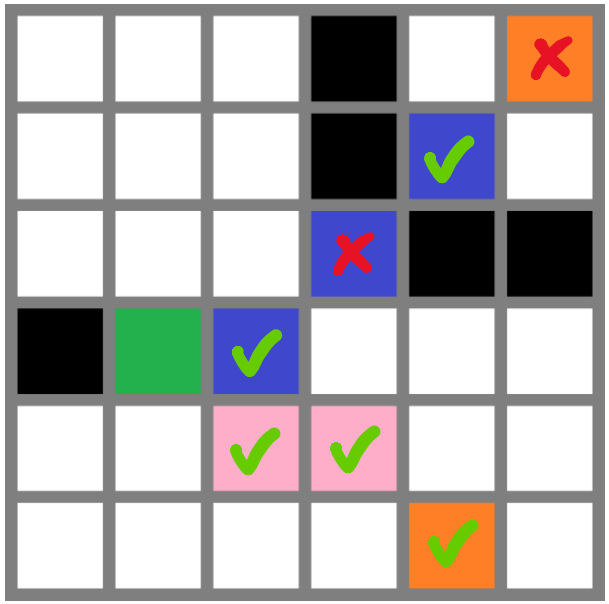
Caractère reliable



Chemin 1

Chemin 2

Caractère reliable

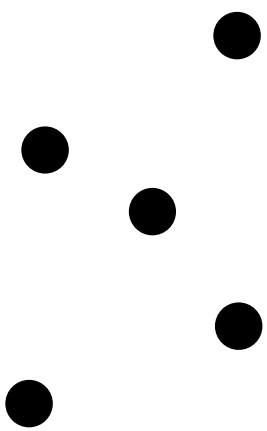


Chemin incorrect

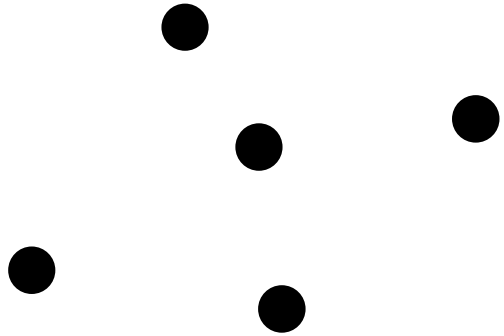
Chemin correct

Résultat de l’extraction

Algorithme de comparaison



Ensemble 1



Ensemble 2

Algorithme de comparaison

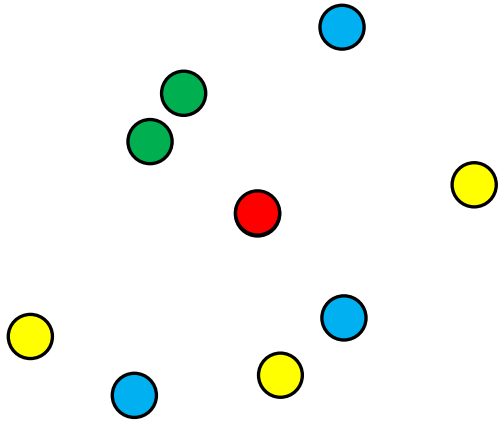
Distances très différentes

Algorithme de comparaison

Distances proches

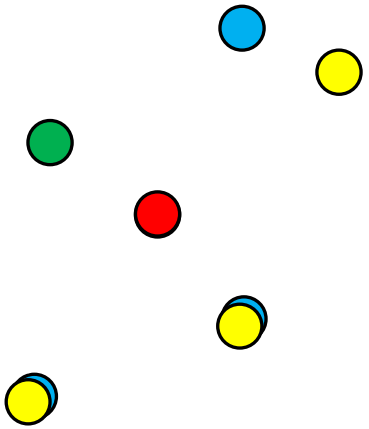
Algorithme de comparaison

Algorithme de comparaison



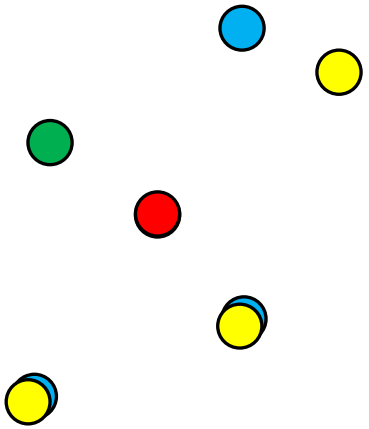
1 : Translation

Algorithme de comparaison



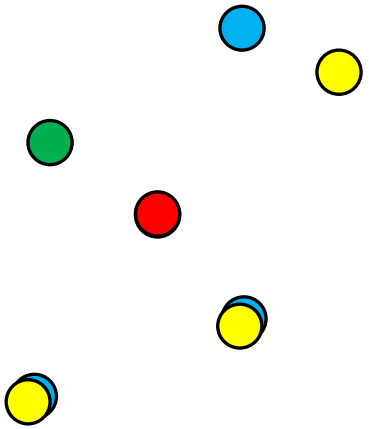
2 : Rotation et redimensionnement

Algorithme de comparaison



4 points correspondant dans les 2 ensembles

Algorithme de comparaison



4 points correspondant dans les 2 ensembles

Algorithme de comparaison

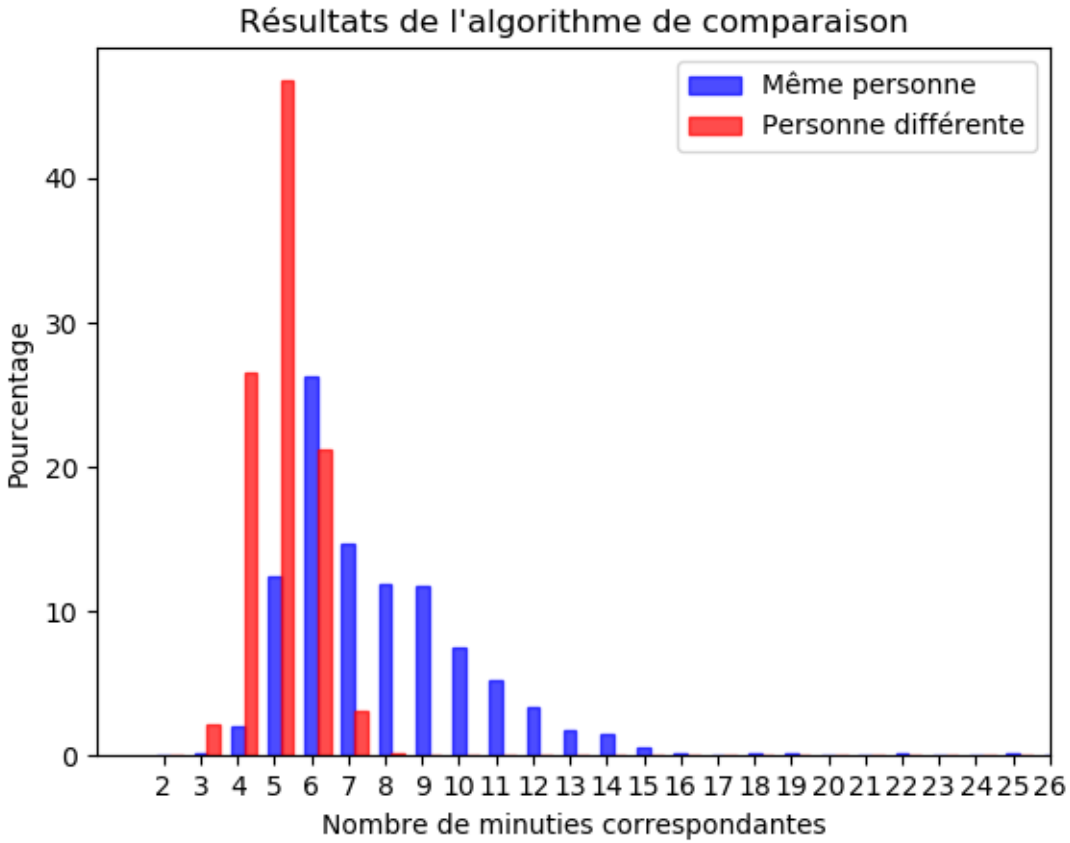
Soit n et m le nombre de points des ensembles

- Complexité : $O(m^2n^2)$

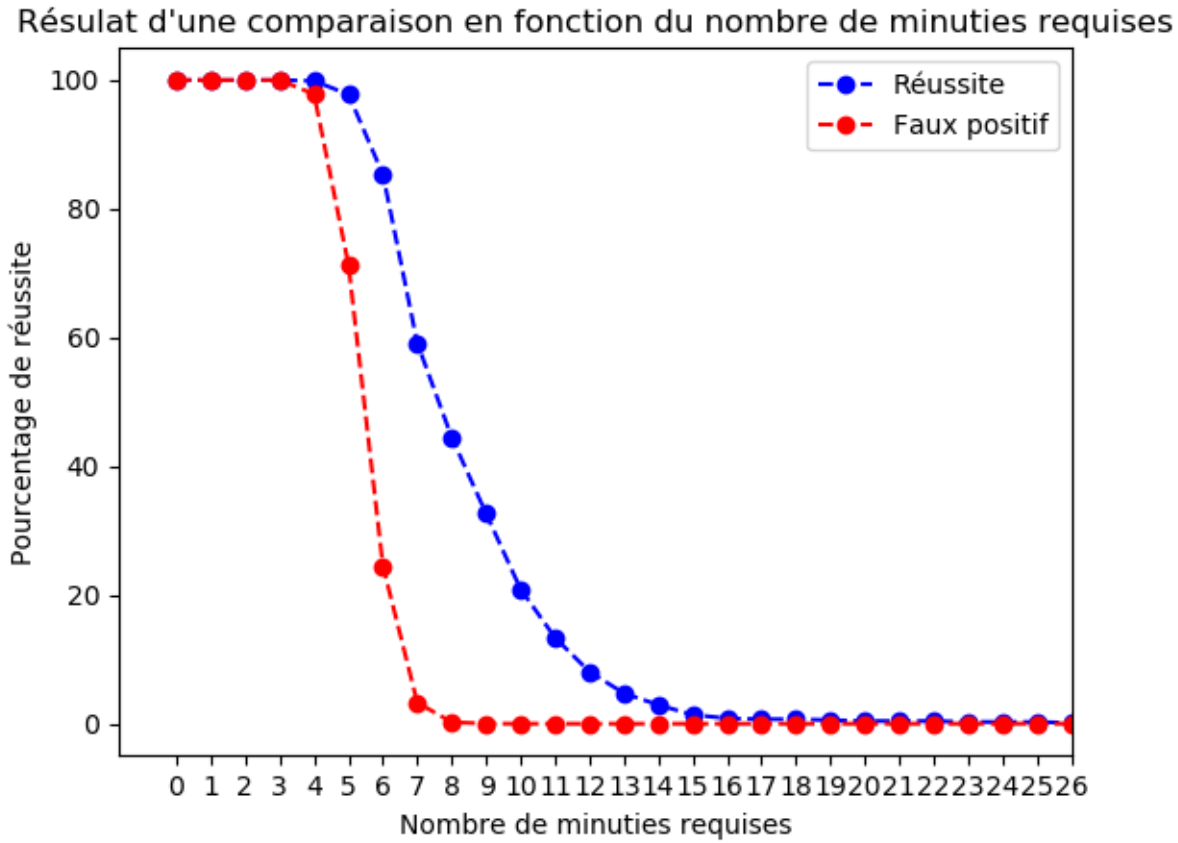
- $15 \leq n \leq 35$ et $15 \leq m \leq 35$

- Temps d'exécution moyen : 0.006s

Résultats (n=135)



Résultats (n=135)



Encre

Effet de Zhang-Suen

Liste des fichiers

main.py – diapositives 48-49

analysis.py – diapositives 50-55

image_processing.py – diapositives 56-69

matching.py – diapositives 70-73

minutiae.py – diapositives 74-80

polygons.py – diapositives 81-85

storage.py – diapositives 86-89

```
import analysis
import storage

if __name__ == "__main__":

    distance_tolerance = 4
    rescale_tolerance = 0.01

    db_name = "fingerprint database"
    fingerprints_folder = "hk"

    storage.create_fingerprints_database(db_name, fingerprints_folder)
    db = storage.load("db/" + db_name)

    good_quality_fingerprints = [0, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 20, 21, 24,
    25, 26, 27, 28, 29, 30, 31, 42, 47, 50, 51, 53, 54, 56, 57, 59, 61, 64, 67, 68, 69,
    70, 71, 73, 74, 77, 78, 81, 82, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 96, 97,
    100, 105, 106, 109, 110, 111, 116, 120, 121, 122, 123, 125, 131, 135, 138, 144, 145,
    147, 148, 149, 151, 156, 177, 179, 181, 182, 184, 188, 189, 190, 199, 216, 220, 221,
    222, 223, 234, 236, 241, 242, 244, 248, 250, 251, 266, 267, 269, 270, 271, 272, 273,
    276, 277, 278, 279, 280, 282, 284, 285, 287, 289, 290, 297, 298, 299, 300, 303, 311,
    312, 313, 314, 316, 317, 319, 321, 322, 323, 334]
```

```
db_q = []
for e in good_quality_fingerprints:
    db_q.append(db[e])

analysis.remove_fingerprints_number_minuaties(db_q, 15, 35)

same_person = analysis.stats_same_person(db_q, distance_tolerance, rescale_tolerance)
diffent_person = analysis.stats_diffent_person(db_q, distance_tolerance, rescale_tolerance)

analysis.bar_graph(same_person, diffent_person)
analysis.stats_one_to_one(same_person, diffent_person)
```

```
import matplotlib.pyplot as plt

import numpy as np

import matching

def stats_same_person(db, deltaD, deltaM):

    stats = np.zeros(40)
    db_size = len(db)

    for k in range(db_size):

        fingerprints = db[k]["fingerprints"]
        nb = len(fingerprints)

        for i in range(nb):
            for j in range(i+1,nb):

                set1 = np.array(fingerprints[i]).astype(float)
                set2 = np.array(fingerprints[j]).astype(float)

                stats[matching.cpu_match(set1,set2,float(deltaD),deltaM)] += 1

    return stats
```

```
def stats_diffent_person(db, deltaD, deltaM):

    stats = np.zeros(40)
    db_size = len(db)

    for k in range(db_size):
        for l in range(k+1, db_size):

            fingerprints_1 = db[k]["fingerprints"]
            nb1 = len(fingerprints_1)
            fingerprints_2 = db[l]["fingerprints"]
            nb2 = len(fingerprints_2)

            for i in range(nb1):
                for j in range(nb2):

                    set1 = np.array(fingerprints_1[i]).astype(float)
                    set2 = np.array(fingerprints_2[j]).astype(float)

                    stats[matching.cpu_match(set1,set2,float(deltaD),deltaM)] += 1

    return stats
```

```
def remove_empty_end(set1, set2):
    n = len(set1)
    repeat = True

    list1 = set1.tolist()
    list2 = set2.tolist()

    while repeat:
        n -= 1
        if set1[n] == 0 and set2[n] == 0:
            list1.pop(n)
            list2.pop(n)
        else:
            repeat = False

    return np.array(list1), np.array(list2)
```

```
def bar_graph(same, different):

    same_norm = (np.array(same)/np.sum(same))[2:]
    different_norm = (np.array(different)/np.sum(different))[2:]

    same_norm, different_norm = remove_empty_end(same_norm, different_norm)

    fig = plt.figure()
    ind = np.arange(len(same_norm))
    ax = fig.add_subplot(111)
    width = 0.35

    ax.bar(ind+2, same_norm*100, width, color='b', edgecolor='b', alpha=.7)
    ax.bar(ind+width+2, different_norm*100, width, color='r', edgecolor='r', alpha=.7)

    ax.set_xticks(ind+2)
    ax.set_xlabel('Nombre de minuties correspondantes')
    ax.set_ylabel('Pourcentage')
    ax.set_title("Résultats de l'algorithme de comparaison")
    ax.legend(['Même personne', 'Personne différente'])

    plt.show()
```

```python
def stats_one_to_one(same,different):
    same_norm = np.array(same)/np.sum(same)
    different_norm = np.array(different)/np.sum(different)

    same_norm, different_norm = remove_empty_end(same_norm, different_norm)

    sums_same = [same_norm[i:].sum()*100 for i in range(len(same_norm))]
    sums_different = [different_norm[i:].sum()*100 for i in range(len(different_norm))]
    ind = np.arange(len(sums_same))

    fig, axs = plt.plot(ind,sums_same, '--b', ind,sums_different, '--r', marker="o")
    plt.xticks(ind)
    plt.xlabel('Nombre de minuties requises')
    plt.ylabel('Pourcentage de réussite')
    plt.title("Résulat d'une comparaison en fonction du nombre de minuties requises")
    plt.legend(['Réussite', 'Faux positif'])
    plt.show()
```

```
def remove_fingerprints_number_minuaties(db,mintuiae_min,mintuiae_max):

    db_size = len(db)

    for k in range(db_size):

        fingerprints = db[k]["fingerprints"]
        nb = len(fingerprints)

        for i in range(nb-1, -1, -1):

            if not (mintuiae_min<=len(fingerprints[i])<=mintuiae_max):
                fingerprints.pop(i)
```

```
from PIL import Image
import tifffile

import numpy as np
from numba import njit

def path_to_greyscale_array(path):
    return tifffile.imread(path)

def get_thresold(image, thr):
    x_size, y_size = np.shape(image)
    liste = []
    for i in range(x_size):
        for j in range(y_size):
            if image[i][j] != 255:
                liste.append(image[i][j])
    return percentile(liste, thr)
```

```python
def percentile(liste, p):
    liste.sort()
    return liste[int(p*len(liste))]

def binarize(image, threshold):
    copy = image.copy()
    copy[copy <= threshold] = 1
    copy[copy > threshold] = 0

    return copy
```

```
@njit
def fill_holes(image, area_max):
    output = image.copy()
    x_max,y_max = image.shape
    zones = []

    for x in range(x_max):
        for y in range(y_max):
            if image[x,y]==0:
                i = x*x_max+y_max+2
                holes = spread(output,[(x,y)],x_max,y_max,i)
                zones.append((holes,i))

    for holes, i in zones:
        if len(holes) <= area_max:
            for x,y in holes:
                output[x,y] = 1
        else:
            for x,y in holes:
                output[x,y] = 0

    return output
```

```
@njit
def spread(image,stack,x_max,y_max,rep):
    holes = []
    while stack:
        x,y = stack.pop()
        if 0<=x<=(x_max-1) and 0<=y<=(y_max-1) and image[x,y]==0:
            image[x,y]=rep
            holes.append((x,y))
            stack += [(x+1,y),(x-1,y),(x,y+1),(x,y-1)]
    return holes
```

```
@njit
def zhang_suen(image):
    x_size, y_size = image.shape
    repeat = True

    while repeat:
        repeat = False
        marked = []
        for x in range(1,x_size-1):
            for y in range(1,y_size-1):
                if (image[x][y]==1) and step1(image,x,y) and (zhang_suen_a(image,x,y) == 1)
and (2 <= zhang_suen_b(image,x,y) <= 6):
                    marked.append((x,y))
        for (x,y) in marked:
            image[x][y] = 0
            repeat = True
        marked = []
        for x in range(1,x_size-1):
            for y in range(1,y_size-1):
                if (image[x][y]==1) and step2(image,x,y) and (zhang_suen_a(image,x,y) == 1)
and (2 <= zhang_suen_b(image,x,y) <= 6):
                    marked.append((x,y))
        for (x,y) in marked:
            image[x][y] = 0
            repeat = True

    return image
```

```
@njit
def zhang_suen_a(image,x,y):
    total = 0
    liste = [(x,y+1),(x+1,y+1),(x+1,y),(x+1,y-1),(x,y-1),(x-1,y-1),(x-1,y),(x-1,y+1),
(x,y+1)]

    for i in range(8):
        if image[liste[i]]==0 and image[liste[i+1]]==1:
            total += 1

    return total

@njit
def zhang_suen_b(image,x,y):
    total = 0
    for i in range(3):
        for j in range(3):
            if (i,j) != (1,1):
                total += image[x-1+i,y-1+j]
    return total
```

```
@njit
def step1(image,x,y):
    top = image[x,y+1]
    right = image[x+1,y]
    down = image[x,y-1]
    left = image[x-1,y]
    return (top==0 or right==0 or down==0) and (right==0 or down==0 or left==0)

@njit
def step2(image,x,y):
    top = image[x,y+1]
    right = image[x+1,y]
    down = image[x,y-1]
    left = image[x-1,y]
    return (top==0 or right==0 or left==0) and (top==0 or down==0 or left==0)
```

```
def thinning_2(image):
    filters = [np.array([[0,0,0],[2,1,2],[1,1,1]]),np.array([[2,0,0],[1,1,0],[2,1,2]]),
np.array([[0,1,0],[1,1,1],[2,0,2]])]
    filters = get_filters_list(filters)
    x_size, y_size = image.shape
    history = [image, np.zeros(image.shape)]

    while (history[0]!=history[1]).any():
        history[1] = history[0].copy()
        for filter in filters:
            history[0] = history[0] - hit_and_miss(history[0], filter, x_size, y_size)

    return history[0]

def hit_and_miss(image, filter, x_size, y_size):
    blank = np.zeros((x_size, y_size))
    for x in range(1,x_size-1):
        for y in range(1,y_size-1):
            if image[x][y] == 1:
                blank[x][y] = apply_filter(image[x-1:x+2,y-1:y+2].copy(), filter)

    return blank
```

```
def apply_filter(array, filter):
    for (x,y) in filter["tuples"]:
        array[x][y] = 2
    if (array != filter["array"]).any():
            return 0

    return 1

def get_filters_list(filters):
    filter_list = []
    for i in range(4):
        for f in filters:
            filter_rotated = np.rot90(f,i)
            filter_list.append({"array": filter_rotated,"tuples":
get_filter_tuples_not_checked(filter_rotated)})
    return filter_list

def get_filter_tuples_not_checked(filter):
    couple = []
    for (x,y) in [(0,0),(0,1),(0,2),(1,0),(1,2),(2,0),(2,1),(2,2)]:
        if filter[x][y] == 2:
            couple.append((x,y))
    return couple
```

```
def array_to_image(image, minutiae = [], points = [], vertices = []):
    x_size, y_size = image.shape
    rgb_array = np.zeros((x_size, y_size,3), dtype = np.uint8)

    for x in range(0,x_size):
        for y in range(0,y_size):
            rgb_array[x,y] = color_matching(image[x,y])

    if minutiae:
        for minutiae in minutiae:
            for i in range(9):
                if image[minutiae["x"] - 1 + i//3][minutiae["y"] - 1 + i%3] == 1:
                    rgb_array[minutiae["x"] - 1 + i//3][minutiae["y"] - 1 + i%3]
= color_matching(minutiae["type"])

    if vertices:
        for set_of_vertices in vertices:
            vertices = set_of_vertices[0]
            color = set_of_vertices[1]
            for i in range(-1,len(vertices)-1):
                pts = get_line((vertices[i][0], vertices[i][1]), (vertices[i+1][0],
vertices[i+1][1]))
                for p in pts:
                    rgb_array[p] = color
```

```
    if points:
        for set_of_point in points:
            points = set_of_point[0]
            color = set_of_point[1]
            thickness = set_of_point[2]
            for point in points:
                for i in range(thickness**2):
                    rgb_array[int(point[0] - thickness//2 + i//thickness)]
[int(point[1] - thickness//2 + i%thickness)] = color

    return Image.fromarray(rgb_array)

def color_matching(number):
    if number==0:                # Fond - blanc
        return [255,255,255]
    elif number==1:              # Empreinte - noir
        return [0,0,0]
    elif number==2:              # Terminaison - vert
        return [0,255,0]
    elif number>=4:              # Bifurcation - bleu
        return [0,0,255]
```

```
def add_minutiae(image, minutiae):
    x_size, y_size = image.shape
    rgb_array = np.zeros((x_size, y_size,3), dtype = np.uint8)

    for x in range(0,x_size):
        for y in range(0,y_size):
            c = image[x,y]
            rgb_array[x,y] = [c,c,c]

    for minutiae in minutiae:
        thickness = 5
        for i in range(thickness**2):
            rgb_array[int(minutiae[0] - thickness//2 + i//thickness)]
[int(minutiae[1] - thickness//2 + i%thickness)] = [255,0,0]

    return Image.fromarray(rgb_array)
```

```
def get_line(p1,p2):
    x1, y1 = p1
    x2, y2 = p2
    pl = []

    if x1==x2:
        for y in range(y1,y2+1):
            pl.append((x1,y))

    elif y1==y2:
        for x in range(x1,x2+1):
            pl.append((x,y1))

    else:
        dx = x2 - x1
        dy = y2 - y1
        err = 0

        if abs(dx) >= abs(dy):
            if x2<x1:
                x1,y1,x2,y2=(x2,y2,x1,y1)
                dx,dy = -dx,-dy
            deltaerr = abs(dy/dx)
            y = y1
```

```
            for x in range(x1,x2+1):
                pl.append((x,y))
                err = err + deltaerr
                if err >= 0.5:
                    if y!=y2:
                        pl.append((x+1,y))
                    y += np.sign(dy)
                    err -= 1
        else:
            if y2<y1:
                x1,y1,x2,y2=(x2,y2,x1,y1)
                dx,dy = -dx,-dy
            deltaerr = abs(dx/dy)
            x = x1

            for y in range(y1,y2+1):
                pl.append((x,y))
                err = err + deltaerr
                if err >= 0.5:
                    if x!=x2:
                        pl.append((x,y-1))
                    x += np.sign(dx)
                    err -= 1

    return pl
```

```
import numpy as np
from numba import njit, prange
import math


def cpu_match(set1, set2, deltaM, deltaD):
    m, n = len(set1), len(set2)
    if m>n:
        set1,set2 = set2,set1
        m, n = n, m

    res_mat = np.zeros((m,m,n,n)).astype(int)
    cpu_matched_matrix(set1, set2, deltaM, deltaD, res_mat)

    return np.amax(res_mat)
```

```
@njit
def cpu_match_coord(set1, set2, deltaM, deltaD, i, j, k, l):

    if i!=j and k!=l:

        set1_or, set2_or = set1[i], set2[k]
        set1_p, set2_p = set1[j], set2[l]
        norm1 = math.sqrt((set1_or[0] - set1_p[0])**2+(set1_or[1] - set1_p[1])**2)
        norm2 = math.sqrt((set2_or[0] - set2_p[0])**2+(set2_or[1] - set2_p[1])**2)

        if (1-deltaD)<norm1/norm2<(1+deltaD):

            ns = (set2 - set2_or)*norm1/norm2
            angle = math.atan2(set1_p[0]-set1_or[0],set1_p[1]-set1_or[1])
- math.atan2(set2_p[0]-set2_or[0],set2_p[1]-set2_or[1])
            cpu_rotate(ns, angle, ns)
            ns = ns + set1_or

            return cpu_close_enough(set1,ns,deltaM)

    return 0
```

```
@njit
def cpu_rotate(set, angle, ns):
    cos = math.cos(angle)
    sin = math.sin(angle)
    for i in range(set.shape[0]):
        ns[i,0] = cos*set[i,0] + sin*set[i,1]
        ns[i,1] = -sin*set[i,0] + cos*set[i,1]

@njit
def cpu_close_enough(set1,set2,delta):
    matchs = 0
    for i in range(set1.shape[0]):
        found = False
        for j in range(set2.shape[0]):
            if ((set1[i][0]-set2[j][0])**2+(set1[i][1]-set2[j][1])**2)<(delta**2):
                found = True
        if found:
            matchs += 1

    return matchs
```

```
@njit(parallel = True)
def cpu_matched_matrix(set1, set2, deltaM, deltaD, res_mat):

    m = set1.shape[0]
    n = set2.shape[0]

    for i in prange(m):
        for j in prange(m):
            for k in prange(n):
                for l in prange(n):
                    res_mat[i,j,k,l] = cpu_match_coord(set1, set2, deltaM, deltaD,
i, j, k, l)
```

```
from PIL import Image

import numpy as np

import image_processing
import polygons

def fingerprint_processing(path, threshold=0.87, ratio=0.95, saveImages = False):

    output = []

    greyscale_image = image_processing.path_to_greyscale_array(path)
    sensibility = image_processing.get_thresold(greyscale_image, threshold)
    binarized_image = image_processing.binarize(greyscale_image, sensibility)
    filled_image = image_processing.fill_holes(binarized_image, 15)



    thinned_image_1 = image_processing.zhang_suen(filled_image.copy())
    thinned_image = image_processing.thinning_2(thinned_image_1.copy())


    outer_pixels = polygons.get_outer_pixels(thinned_image)
    polygon_vertices = polygons.marche_de_jarvis(outer_pixels)
    polygon_centroid = polygons.get_centroid(polygon_vertices)
    shrinked_polygon_vertices = polygons.shrink(polygon_vertices, polygon_centroid, ratio)
```

```
    minutiae_list = minutiae_extraction(thinned_image, shrinked_polygon_vertices,
polygon_centroid)
    minutiae_list_filtered = intersections_extraction(thinned_image, minutiae_list)


    output += minutiae_list_filtered

    if saveImages:
        Image.fromarray(greyscale_image).convert("RGB").save('renders/0 - original.png')
        image_processing.array_to_image(binarized_image).save('renders/1 - binary.png')
        image_processing.array_to_image(filled_image).save('renders/2 - filled.png')
        image_processing.array_to_image(thinned_image_1).save('renders/3 - thinned.png')
        image_processing.array_to_image(thinned_image).save('renders/4 - thinned 2.png')
        image_processing.array_to_image(thinned_image,
points = [[[polygon_centroid], [255,0,0], 5],[outer_pixels, [255,0,255], 1]],
vertices = [[polygon_vertices, [255,255,0]],[shrinked_polygon_vertices, [0,255,255]]])
.save('renders/5 - borders.png')
        image_processing.array_to_image(thinned_image,
minutiae = minutiae_list,
vertices = [[shrinked_polygon_vertices,[0,255,255]]]).save('renders/6 - minutiae.png')
        image_processing.array_to_image(thinned_image, minutiae = minutiae_list_filtered)
        .save('renders/7 - intersections.png')

    intersection_list = [(e["x"],e["y"]) for e in output if e["type"]==4]

    return intersection_list,greyscale_image
```

```
def minutiae_extraction(image, polygon, polygon_centroid):
    minutiae = []
    x_size, y_size = image.shape
    for x in range(1,x_size-1):
        for y in range(1,y_size-1):
            if image[x][y]==1:
                type_of_minutiae = is_minutia(image,x,y)
                if type_of_minutiae:
                    temp = True
                    for i in range(36):
                        if {"x":x-1+i//3,"y":y-1+i%3,"type":type_of_minutiae} in minutiae:
                            temp = False
                    if temp and polygons.in_polygon(x,y, polygon, polygon_centroid):
                        minutiae.append({"x":x,"y":y,"type":type_of_minutiae})

    return minutiae
```

```
def is_minutia(image,x,y):
    sum = 0
    for i in range(3):
        for j in range(3):
            sum += image[x-1+i][y-1+j]
    if sum == 2:
        return 2
    elif sum >= 4:
        return 4

def intersections_extraction(image, minutiae):
    liste = []
    for e in minutiae:
        x,y = e["x"], e["y"]
        if e["type"]==4:
            if clean_minutae(image, minutiae, x, y, x, y, 10, 3, 7):
                liste.append(e)
    return liste
```

```
def clean_minutae(thinned, minutiae, x, y, xo, yo, range_look_int, range_look_ends
, range_find_ends, blocked_d=None):
    if range_look_int != 0:
        tbc = [0,1,2,3,5,6,7,8]
        if blocked_d in tbc: tbc.remove(blocked_d)

        for d in tbc:
            x_n = x - 1 + d//3
            y_n = y - 1 + d%3
            if thinned[x_n][y_n]==1:
                if not (x_n==xo and y_n==yo) and {"x":x_n, "y":y_n, "type":4} in minutiae:
                    return False
                if range_look_ends > 0:
                    if {"x":x_n, "y":y_n, "type":2} in minutiae and not clean_step_2(
thinned, minutiae, x_n, y_n, range_find_ends):
                        return False
                if not clean_minutae(thinned, minutiae, x_n, y_n, xo, yo,
range_look_int-1, range_look_ends-1, range_find_ends, blocked_d=abs(8-d)):
                    return False
    return True
```

```
def clean_step_2(thinned, minutiae, x, y, rf):
    for i in range(rf):
        for j in range(rf):
            x_n = x-rf//2+i
            y_n = y-rf//2+j
            if not (x_n == x and y_n == y) and {"x":x_n, "y":y_n, "type":2} in minutiae:
                if not collision(thinned,image_processing.get_line((x,y),(x_n,y_n))[1:-1]):
                    linked, _ = explore(thinned,(x,y),(x_n,y_n))
                    if not linked:
                        return True
    return False

def collision(array, points):
    for p in points:
        x,y = p
        if array[x][y] == 1:
            return True
    return False
```

```python
def explore(array,s,f,liste=[]):
    if s == f:
        return True,[]

    liste.append(s)
    x,y = s
    for i in [0,1,2,3,5,6,7,8,9]:
        x_n = x - 1 + i//3
        y_n = y - 1 + i%3
        if (x_n,y_n) not in liste and array[x_n][y_n]==1:

            liste.append((x_n,y_n))
            r,liste = explore(array,(x_n,y_n),f,liste)
            if r:
                return True,[]

    return False,liste
```

```
import numpy as np

import image_processing

def get_outer_pixels(image):
    x_size, y_size = image.shape
    points = []
    for x in range(0,x_size-1):
        for y in range(0,y_size-1):
            if image[x][y]==1:
                points.append([x, y])
                break
    for x in range(x_size-1,0,-1):
        for y in range(y_size-1,0,-1):
            if image[x][y]==1:
                points.append([x, y])
                break
    for y in range(0,y_size-1):
        for x in range(0,x_size-1):
            if image[x][y]==1:
                points.append([x, y])
                break
    for y in range(y_size-1,0,-1):
        for x in range(x_size-1,0,-1):
            if image[x][y]==1:
                points.append([x, y])
                break

    return points
```

```
def marche_de_jarvis(points):
    p_0 = points[0]
    for p in points:
        if p[0] < p_0[0]:
            p_0 = p

    p_fict = [p_0[0],p_0[1]-1]
    p_1 = next_point(points, p_fict, p_0)

    convex_hull = [p_0,p_1]
    i = 1

    while convex_hull[0] != convex_hull[i]:
        convex_hull.append(next_point(points, convex_hull[i-1],convex_hull[i]))
        i += 1

    convex_hull.pop(i)

    return convex_hull
```

```
def angle(a,b,c):
    if a == b or b == c:
        return 0

    ba = np.array(a) - np.array(b)
    bc = np.array(c) - np.array(b)

    cos_angle = max(-1,min(1,np.dot(ba, bc) / (np.linalg.norm(ba) * np.linalg.norm(bc))))

    return np.arccos(cos_angle)

def next_point(points, n_1, n):
    p_i = points[0]
    a_i = angle(n_1,n,p_i)
    for p in points:
        a_p = angle(n_1,n,p)
        if a_p > a_i:
            p_i = p
            a_i = a_p

    return p_i
```

```
def get_centroid(vertices):
    a = c_x = c_y = 0

    for i in range(-1,len(vertices)-1):
        a += vertices[i][0]*vertices[i+1][1]-vertices[i+1][0]*vertices[i][1]
        t = vertices[i][0]*vertices[i+1][1]-vertices[i+1][0]*vertices[i][1]
        c_x += (vertices[i][0]+vertices[i+1][0])*t
        c_y += (vertices[i][1]+vertices[i+1][1])*t

    return [int((1/(3*a))*c_x),int((1/(3*a))*c_y)]

def shrink(polygon, centroid, ratio):
    shrinked_vertices = []
    for point in polygon:
        r = np.sqrt((point[0]-centroid[0])**2+(point[1]-centroid[1])**2)
        theta = np.arctan2(point[1]-centroid[1],point[0]-centroid[0])
        r *= ratio
        shrinked_vertices.append([int(r*np.cos(theta))+centroid[0],
int(r*np.sin(theta))+centroid[1]])

    return shrinked_vertices
```

```
def in_polygon(x,y, polygon, polygon_centroid):
    l = len(polygon)

    for i in range(l-2):
        p1 = polygon[i]
        p2 = polygon[i+1]
        line1 = image_processing.get_line((x,y),polygon_centroid)
        line2 = image_processing.get_line(p1,p2)

        for e in line1:
            if e in line2:
                return False

    return True
```

```
from PIL import Image

from os import listdir
import pickle

import minutiae
import image_processing

def create_fingerprints_database(filename, folder, threshold=0.87, ratio=0.95):
    db_path = "db/"+filename
    create_db(db_path)
    db = load(db_path)

    persons = listdir(folder)
    persons.sort(key=int)

    for person in persons:
        path = folder+"/"+person
        print(path)
        print(" ")

        add_person(db, person, 2, "M")

        fingerprints = listdir(path)
        for fingerprint in fingerprints:

            path = folder+"/"+person+"/"+fingerprint
            print("--")
            print(path)
```

```
            minutiae_list,greyscale = minutiae.fingerprint_processing(path, threshold,
ratio)
            image_processing.add_minutiae(greyscale, minutiae_list).save("r " + path[:-4]
+ " - " + str(len(minutiae_list)) + ".png")
            add_array(db, len(db)-1, minutiae_list)
            save(db_path, db)

        print(" ")
        print("------------------")
        print(" ")

    save(db_path, db)
```

```
def create_db(path):
    save(path, [])

def load(path):
    with open(path, 'rb') as db_file:
        db = pickle.load(db_file)
    return db

def save(path, db):
    with open(path, 'wb') as db_file:
        pickle.dump(db, db_file)

def add_person(db, slide, age, gender):
    db.append({"id": len(db), "slide": slide, "age":age, "fingerprints": []})
```

```
def add_array(db, id, array):
    db[id]["fingerprints"].append(array)

def remove_person(db, id):
    db.pop(id)

def remove_array(db, id, number):
    db[id]["fingerprints"].pop(number)

def array_number(db, id):
    return len(db[id]["fingerprints"])
```