TIPE

Modélisation de réseaux vasculaires pour l'entraînement de nano-robots

Enzo Rodrigues 17181

Sommaire

Introduction

Introduction

Objectifs du TIPE

- Modéliser un réseau vasculaire réaliste pour la détection d'anomalies par algorithme.
  1. Le réseau doit être généré aléatoirement pour tester l'algorithme dans des cas variés
  2. Le réseau, ainsi que la répartition de la maladie doit être réaliste
- Créer des algorithmes et comparer leur efficacité

Introduction

Modélisation du réseau

Le réseau est modélisé par une structure de graphe qui relie des points placés aléatoirement par leur plus proche voisin.

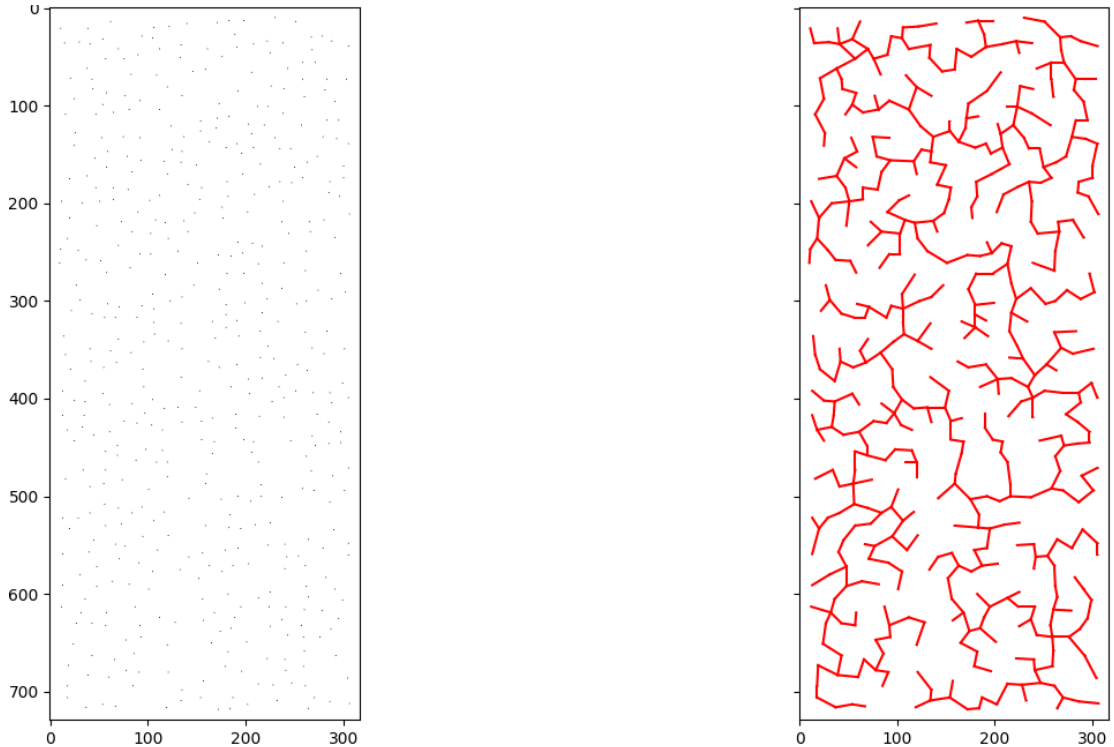


Figure 1: Modélisation d'un réseau

Introduction

Modélisation du réseau

- Pour arriver à ce résultat il y a plusieurs étapes:
  1. Création d'un graphe valué comprenant toutes les arrêtes possibles
  2. Création à partir du graphe valué d'un arbre couvrant minimal: Graphe acyclique connexe dont les arrêtes sont les distances minimales entre 2 points.

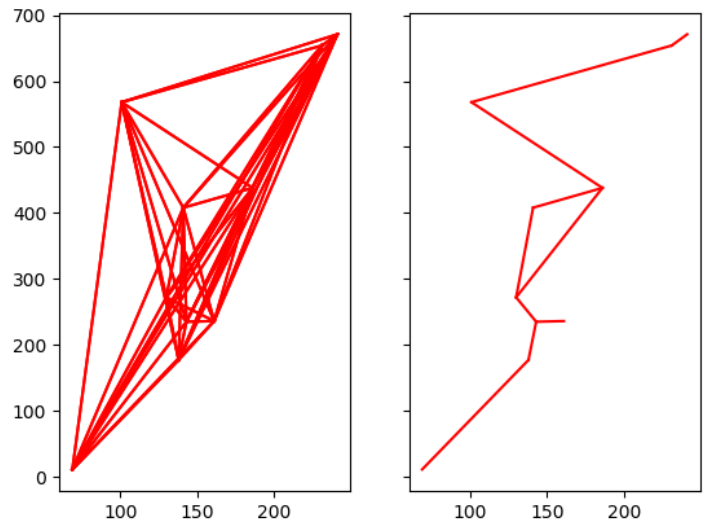


Figure 2: Modélisation d'un réseau

Modèle 2D

Modèle 2D

Répartition des points du réseau

- Pour obtenir un réseau réaliste, les points ne doivent pas être placés aléatoirement, mais selon une densité de probabilité.
- Les points ne peuvent pas être présents sur les pixels noirs

Figure 3: Radio d'un avant-bras

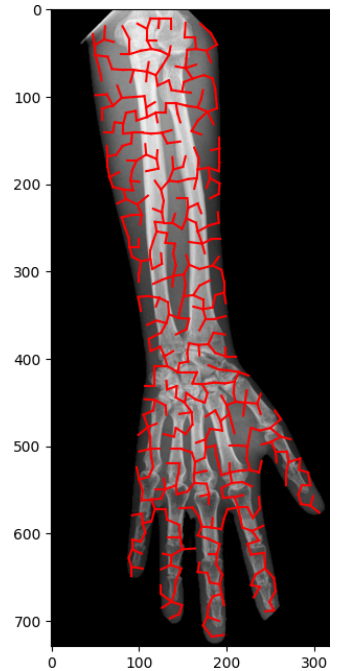


Figure 4: Réseau engendré

Modèle 2D

Proximité des points du réseau

- Pour éviter la présence de ramifications trop petites, l'image représentant la densité de probabilité est modifié à chaque nouveau point

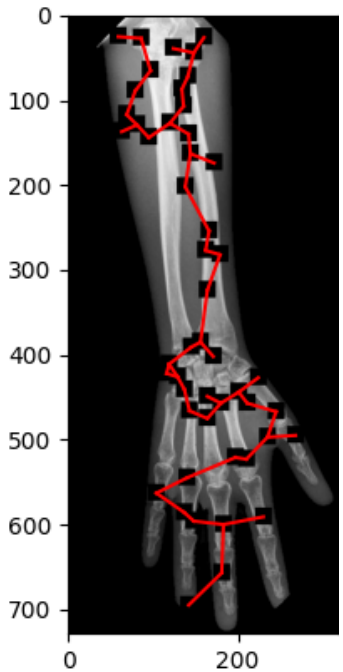


Figure 5: Modification de l'image après le positionnement de quelques points

Modèle 2D

Représentation de la maladie

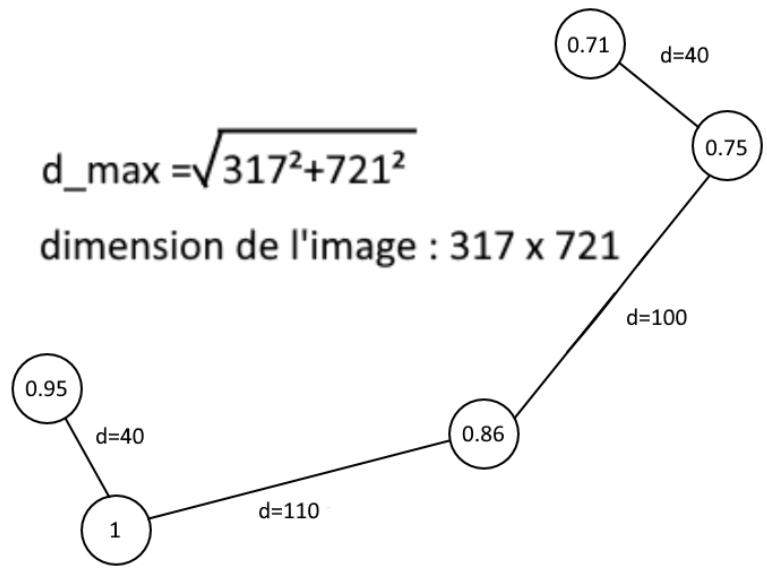
- Pour représenter la maladie j'ai décidé de mettre un poids sur mes points entre 0 et 1, plus le poids est élevé plus le point est touché par la maladie.

Modèle 2D

Représentation de la maladie

- Je choisi donc un point au hasard pour lui donner une valeur de 1 et défini le poids des autres points avec la récurrence suivante:

$$p_{i+1} = (1 - \frac{d(i,i+1)}{d_{max}}) \cdot p_i$$



- Où $d(x, y)$ donne le poids entre le point x et le point y, et d_{max} est la distance maximale possible

Modèle 2D

Représentation de la maladie

- J’ai ensuite affecté la valeur du poids à la couleur du mes segments, plus le segments est rouge clair, plus son poids est élevé

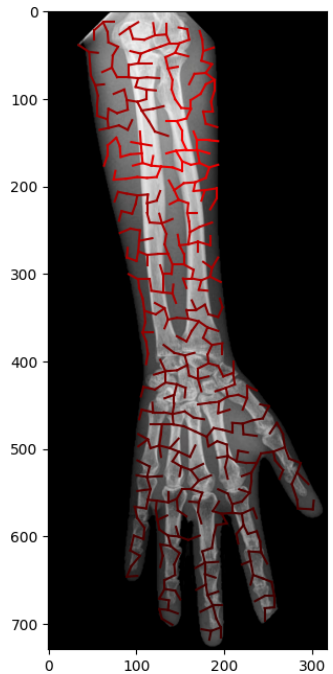


Figure 6: Visualisation de la maladie dans le réseau

Modèle 3D

Modèle 3D

Répartition des points

- Pour modéliser un avant bras dans le cadre de mon modèle j'ai choisi de représenter un cylindre de 5cm de rayon et de 30cm de hauteur pour y placer mes points.

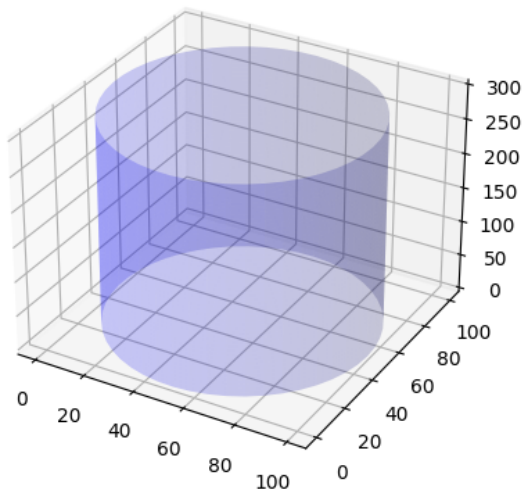


Figure 7: Représentation cylindrique

Modèle 3D

Modélisation du réseau

- Le réseau est modélisé de la même manière que celui en 2 dimensions, à la différence que chaque point possède une coordonnée en plus.

- Pour des questions de lisibilité, ici la largeur des arrêtes dépend également de leur poids jusqu'à un certains seuil

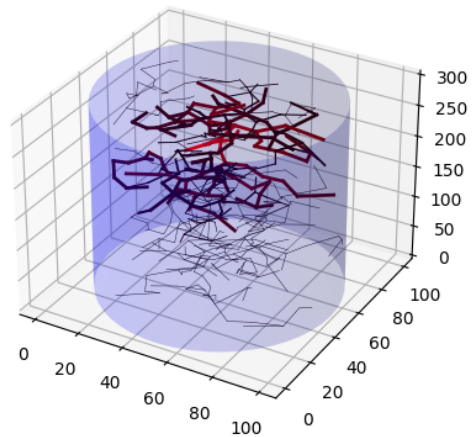


Figure 8: Représentation du réseau en 3 dimensions

Modèle 3D

Modélisation du réseau

- La largeur des traits varie de 0.3 à 2 selon la formule suivante

$$lw_{i,i+1} = max(0.3, min(2, (50 \cdot p_i)^2)$$

- Avec i et $i + 1$ les indices des extrémités du segment, et p_i le poids du point i

Algorithme

Algorithme
1er algorithme

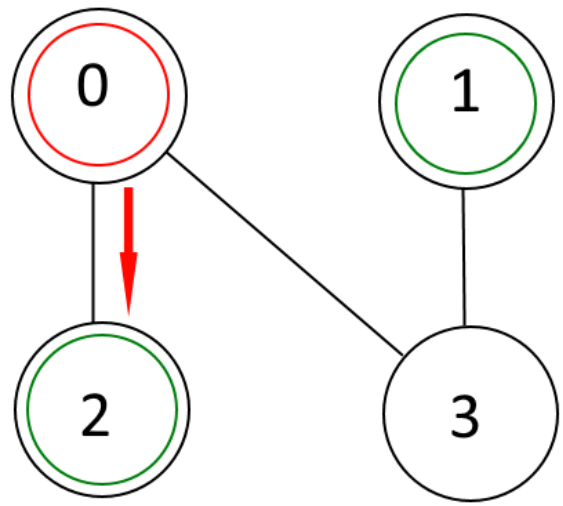
J'ai décidé d'utiliser la valuation de mon graphe afin de créer des algorithme plus efficace.
J'ai d'abord créer un algorithme avec mémoire:

1. On démarre d'un point au hasard et on ajoute tous ses voisins dans une file de priorité
2. On va au point suivant de la file et on réitère 1.
3. On fait ceci jusqu'à avoir visité tout les points ou avoir trouvé un point i tel que $p_i = 1$

Algorithme

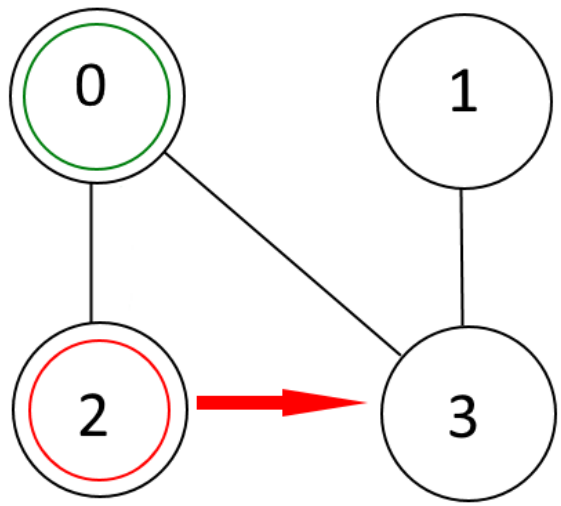
1er algorithme

1. Graphe: [[3,2],[3],[0],[0,1]]
   On commence au point 0
   Voisins(0) = [3,2]
   File = [(p_2,2),(p_3,3)]
   Avec $p_2 > p_3$
2. ...
3. ...



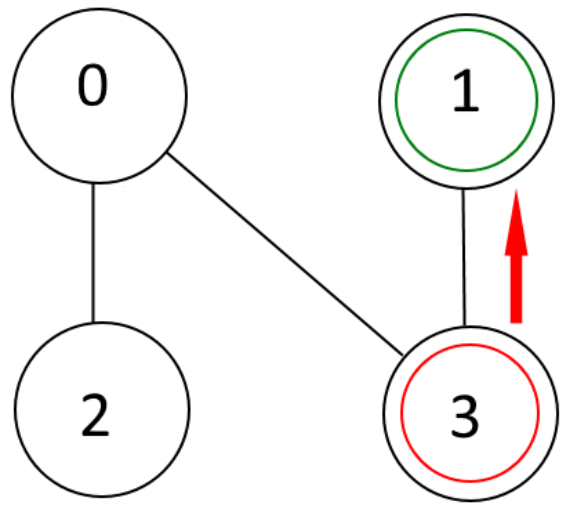
Algorithme
1er algorithme

1. ...
2. File = [(p_2,2),(p_3,3)]
   On va au premier point de la file
   Voisins(2) = [0]
   Or on a déjà vu 0
   On va donc au point 3
3. ...



Algorithme
1er algorithme

1. ...
2. ...
3. Si $p_2 = 1$ Alors on est sorti de l'algorithme à la première itération
   Sinon, on a parcouru tout le graphe en 3 itérations ici :
   $0 \to 2 \to 3 \to 1$



Algorithme

1er algorithme

J’ai voulu représenter les résultats de mon algorithme pas par pas pour en voir l’évolution:

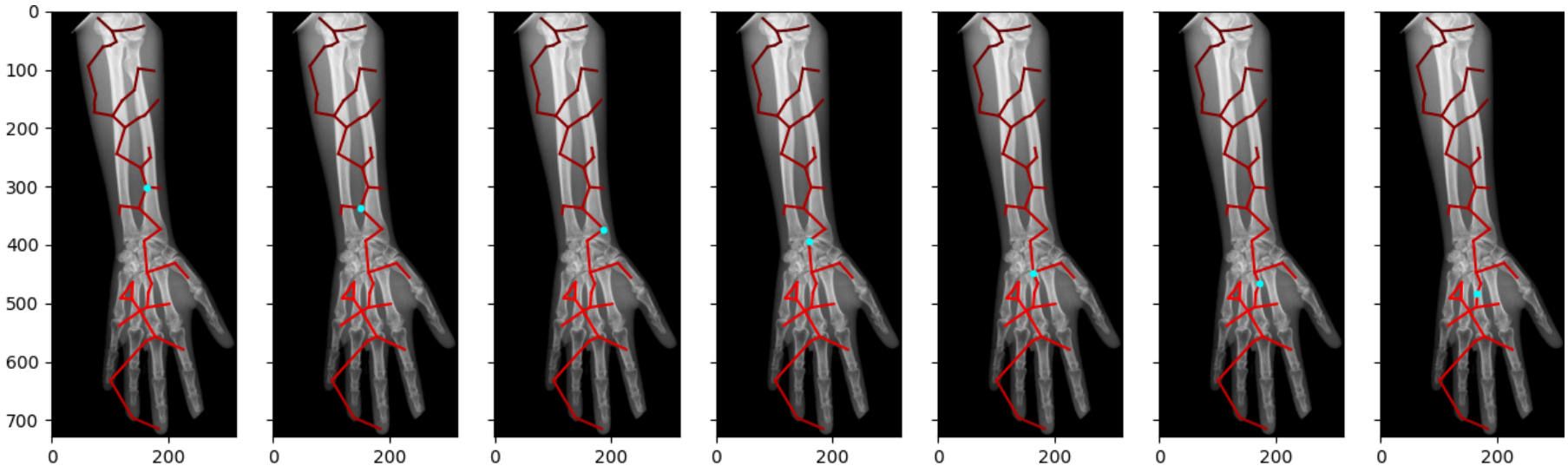


Figure 9: Évolution de l’algorithme 1

Algorithme
2nd Algorithme

Dans un soucis de réalisme, j'ai émis l'hypothèse que les nanos robots ne pouvaient pas disposer de mémoire.
Je me suis donc débarrasser de ma file et j'ai réaliser une 2nd algorithme:

1. On se place au hasard dans le réseau et on regarde les voisins du point de départ
2. On se place au point de poids maximum parmi les voisins et on réitère 1
3. On s'arrête lorsqu'on a trouvé un point de poids 1 ou lorsque qu'on a réaliser un nombre de boucle arbitraire

Algorithme

2nd Algorithme

En utilisant le même type de représentation que pour le premier algorithme j'ai représenté l'algorithme 2, ce dernier termine toujours avant 100 tours de boucle

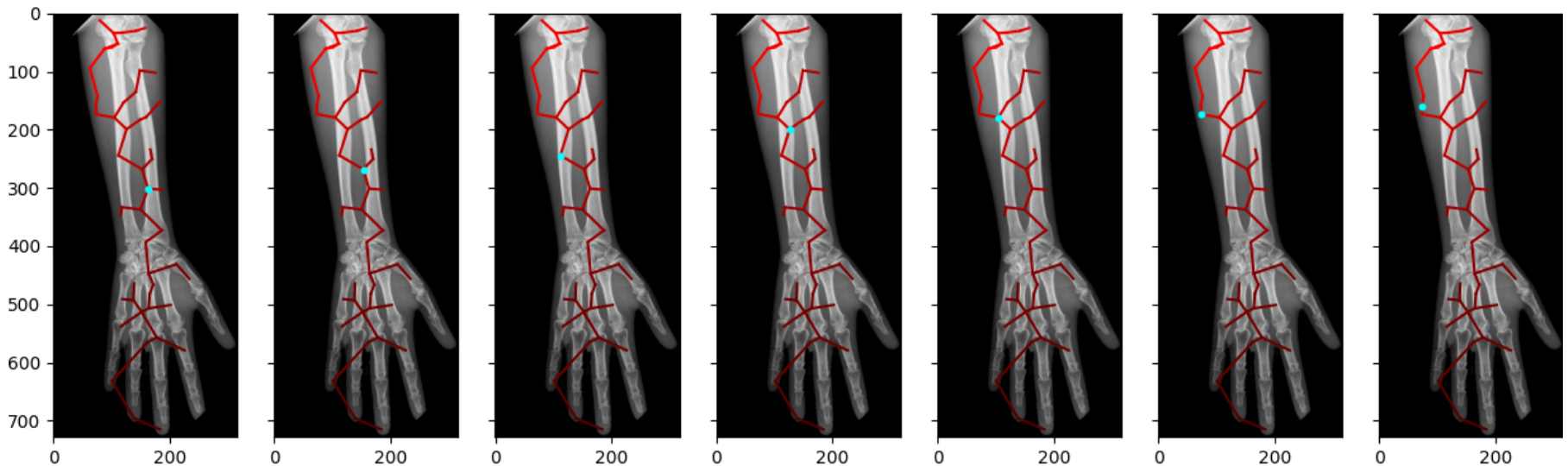


Figure 10: Évolution de l'algorithme 2

Algorithme

Comparaison des algorithmes

J’ai par la suite cherché à comparer ces deux algorithmes en commençant par comparer les sorties de ces deux algorithmes. En turquoise, l’algorithme 1 et en vert l’algorithme 2

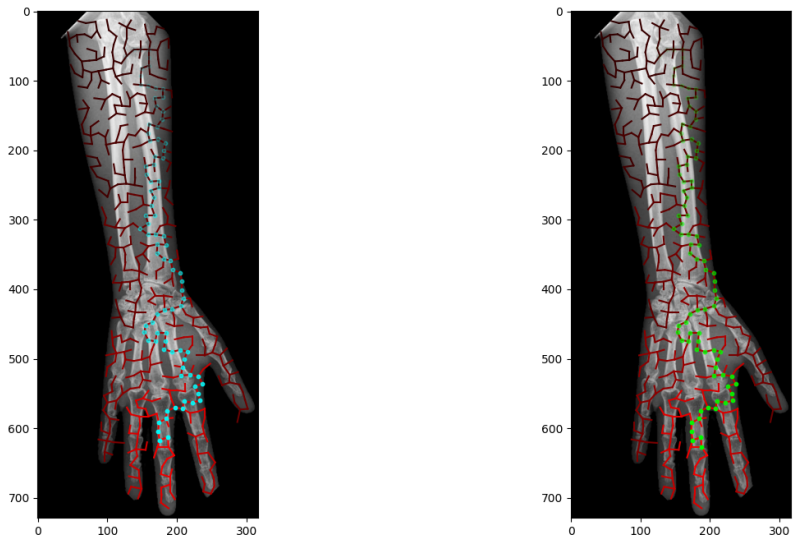


Figure 11: Comparaison des algorithmes

Algorithme

Comparaison des algorithmes

Les deux algorithmes réalise les mêmes étapes mais l'algorithme sans mémoire est plus simple, qu'en est il vis à vis de la complexité temporelle ?

	Algo 1 [s]	**Algo 2** [s]	**Parcours en largeur** [s]
2D 50 points	$1,8 \cdot 10^{-4}$	$4,6 \cdot 10^{-6}$	$2,3 \cdot 10^{-4}$
2D 500 points	$1,1 \cdot 10^{-2}$	$2,6 \cdot 10^{-4}$	$1,5 \cdot 10^{-2}$
3D 500 points	$8,9 \cdot 10^{-3}$	$9,8 \cdot 10^{-4}$	$4,6 \cdot 10^{-2}$
3D 3000 points	$1,1$	$7,2 \cdot 10^{-1}$	$5,8$

Table 1: Tableau de comparaison des algorithmes

Conclusion

Conclusion

Compte rendu

- Modélisation:
  Modélisation de réseau cohérente et adaptative dans le cas de la 2D
  Modélisation de la maladie avec une répartition dans le réseau vasculaire réaliste
- Algorithmes:
  Algorithmes très rapides et simple.

Conclusion

Améliorations possibles

- Modélisation:
  Réaliser une modélisation plus réaliste, avec des tailles de veines différentes et arborescentes.
- Algorithmes
  Ajout d'un autre type d'algorithme du type intelligence artificielle

Annexe

Annexe

Codes

Code de création des arbres et graphes servant à la modélisation

```python
def graphe_value(points):
    '''
    Entrée: Une liste de points représenté par des doublets de coordonnées
    Sortie: Un graphe valué sous la forme d'une liste donc les indices sont l
    les points et les éléments correspondent aux voisins des indices muni de leur poids
    '''
    graphe=[]
    for i in range(len(points)):
        voisins = []
        for j in range(len(points)):
            if i!=j:
                voisins.append((distance(points[i],points[j]),i,j))
        graphe.append(voisins)
    return graphe
```

```python
def graphe_arbre(arbre):
    '''Entrée : un arbre de la forme [segment] où segment est de la forme (a,b)
    Sortie: Un graphe sous la forme [voisins(i)]
    '''
    sortie = [[] for _ in range(len(arbre))]
    for segment in arbre:
        sortie[segment[0]].append(segment[1])
        sortie[segment[1]].append(segment[0])
    return sortie
```

```python
def arbre(graphe):
    '''
    Entrée : un graphe valué dont les valeurs sont p,o,d (poids, origine, destination)
    Sortie : Un arbre couvrant minimal sur la distance (Un arbre reliants les points
    par le point qui leur est le plus proche)
    '''
    n = len(graphe)
    vus = []
    sortie = []
    file = []
    v = voisins(graphe,random.randrange(0,n))
    for x in v:
        heappush(file, x)

    while len(sortie) != n:
        p,o,d = heappop(file)
        if d not in vus:
            vus.append(d)
            sortie.append((o,d))
            v = voisins(graphe,d)
            for x in v:
                if x != (p,d,o):
                    heappush(file,x)
    return sortie
```

Annexe

Codes

Code de création des modélisation 2D et 3D

```
'''Relie les points de l'arbre couvrant minimal'''
for segment in arbre1:
    X=[points[segment[0]][0],points[segment[1]][0]]
    Y=[points[segment[0]][1],points[segment[1]][1]]
    distance_XY=distance(points[segment[0]],points[segment[1]])
    poids[segment[1]]=max(poids[segment[0]]-distance_XY/d_max*poids[segment[0]],poids[segment[1]])
    axs[1].plot(Y,X, color=(poids[segment[0]],0,0),zorder=0)
    axs[0].plot(Y,X, color=(poids[segment[0]],0,0),zorder=0)
axs[1].imshow(image, cmap="gray",zorder=-1)
axs[0].imshow(image, cmap="gray",zorder=-1)
```

```
#Tracé du réseau
for segment in arbre1:
    X = [points[segment[1]][0],points[segment[0]][0]]
    Y = [points[segment[1]][1],points[segment[0]][1]]
    Z = [points[segment[1]][2],points[segment[0]][2]]
    distance_AB=distance(points[segment[0]],points[segment[1]])
    poids[segment[1]]=max(poids[segment[0]]-distance_AB/d_max*poids[segment[0]],poids[segment[1]])
    ax.plot(X,Y,Z,color=(poids[segment[0]],0,0),linewidth=max(0.3,min(2,50*poids[segment[0]]**2)))
plt.show()
```

Annexe
Codes

Code des algorithmes

```python
def parcours_value_avec_memoire(graphe,poids):
    valeur = poids[0]
    point = 0
    file = []
    max = valeur
    point_max = 0
    vus=[0]
    heappush(file,(poids[0],0))
    maximums=[point_max]
    while file:
        valeur, point = heappop(file)
        if valeur>max:
            max=valeur
            point_max = point
            maximums.append(point_max)
        if not(valeur<1):
            return maximums
        for x in graphe[point]:
            if x not in vus:
                heappush(file,(poids[x],x))
            vus.append(x)
    return maximums
```

```python
def parcours_largeur(graphe,poids):
    valeur = poids[0]
    point = 0
    file = []
    max = valeur
    point_max = 0
    passage=[point_max]
    vus=[0]
    heappush(file,(poids[0],0))
    while file:
        valeur, point = file.pop()
        if valeur>max:
            max=valeur
            point_max = point
            passage.append(point_max)
        if not(valeur<1):
            return passage
        for x in graphe[point]:
            if x not in vus:
                file.append((poids[x],x))
            vus.append(x)
    return passage
```

Annexe
Codes

Code des algorithmes

```
def parcours_value_sans_memoire(graphe,poids):
    valeur = poids[0]
    point = 0
    passage=[]
    k=0
    while valeur < 1 and k<3000:
        passage.append(point)
        point = graphe[point][indice_valeur_max([poids[x] for x in graphe[point]])]
        valeur = poids[point]
        k+=1
    return passage
```