

Network Force-Directed Drawing Algorithms

KIM TAEYEON (12204842)

1. Abstract

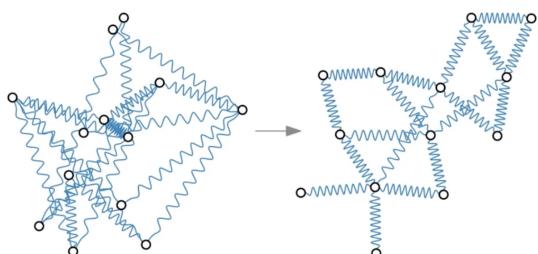
Ce projet s'est fait afin d'implémenter le *Network Force-Directed Drawing Algorithm*. Pour réaliser l'algorithme moi-même de bout en bout, J'ai fait des recherche sur plusieurs théories. Dans ce projet donc, je vais vous introduire 3 théories concernantes, puis je vais expliquer de l'algorithme que j'ai fait à l'aide d'une petite pseudo code. De plus, j'ai crée et utilisé un QuadTree ou un OctetTree pour avoir l'efficacité au niveau de '*computing cost*' et simplifier son calcul. Vous pouvez choisir ce que vous allez utiliser quand vous exécutez le programme comme une option. Enfin, je teste cette algorithme de manière différente pour l'évaluer et pour comparer tous les résultats, par exemple, en variant le type de Tree, le nombre de nœuds, et des façons pour calculer des forces, etc.

👉 Vous pouvez trouver tous les codes sur <https://github.com/citizenyves/SDA>

2. Les théories

2.1. Spring Embedder by Eades

Son algorithme est la base de tous les autres idées de '*Network force-directed drawing algorithms*'. On remplace les nœuds par des anneaux en acier et on remplace chaque arête



par un ressort pour former un système mécanique. Les nœuds sont placés dans une disposition initiale et relâchés de sorte que les forces du ressort sur les anneaux amènent le système à un état d'énergie minimal.

2.1.1 La force de repulsion

On utilise la distance carrée entre deux nœuds comme dénominateur.

$$f_{repulsion}(u, v) = \frac{c_{rep}}{\|p_v - p_u\|^2} \cdot \overrightarrow{p_v p_u}$$

2.1.2. La force de spring

On utilise une force de spring pour calculer une force attractive.

$$f_{spring}(u, v) = c_{spring} \cdot \log \frac{\|p_v - p_u\|}{l} \cdot \overrightarrow{p_u p_v}$$

- Si la distance = la longueur idéale(l), alors la force est à 0 *logarithme = 0
- Si la distance < la longueur idéale(l), alors deux nœuds se repoussent *logarithme < 0 (-)
- Si la distance > la longueur idéale(l), alors deux nœuds s'attirent *logarithme > 0 (+)

$\|p_u - p_v\|$: Euclidean distance entre u et v

$\overrightarrow{p_u p_v}$: unit vecteur pointant de u à v

l : une longueur idéale des arêtes.

c : une constante

2.1.3. La force d'attraction

$$f_{attraction}(u, v) = f_{spring}(u, v) - f_{rep}(u, v)$$

2.2. Variant by Fruchterman & Reingold

2.2.1. La force de repulsion

les nœuds ont envie d'avoir la distance idéale entre eux. si sa distance est inférieure à la longueur idéale la force de repulsion devient grande et vice-versa.

$$f_{repulsion}(u, v) = \frac{l^2}{\|p_v - p_u\|} \cdot \overrightarrow{p_v p_u}$$

2.2.2. La force d'attraction

Fruchterman & Reingold ont supprimé la force de spring puis ils essaient de calculer directement la force d'attraction.

$$f_{attraction}(u, v) = \frac{\|p_v - p_u\|^2}{l} \cdot \overrightarrow{p_u p_v}$$

2.3. Une autre option

2.3.1. La force de repulsion par degré du nœud

On ajoute une multiplication des (degré + 1) de chaque noeud sur le numérateur. Le degré signifie le nombre de noeud qui est connecté avec lui. Grâce à cette idée, des noeuds très peu connectés peuvent être plus proches à d'autres, d'un autre côté des noeuds qui ont plus d'arêtes auront plus grande force de repulsion.

$$f_{repulsion}(u, v) = c_{rep} \frac{(Deg(v)+1)*(Deg(u)+1)}{\|p_v - p_u\|^2} \cdot \overrightarrow{p_v p_u}$$

2.3.1. La force d'attraction (1)classique et de façon (2)LinLog

$$(1) f_{attraction}(u, v) = c_{attraction} \cdot \|p_v - p_u\| \cdot \overrightarrow{p_u p_v}$$

$$(2) f_{attraction}(u, v) = c_{attraction} \cdot \log(1 + \|p_v - p_u\|) \cdot \overrightarrow{p_u p_v}$$

3. L'algorithme

3.1. Pseudo-code

ForceDirected($G = (V, E)$, $p_{init} = (p_v) v \in V$, $\varepsilon > 0$, $K \in N$)

```

 $t \leftarrow 1$ 
while  $t < K$  and  $\max(v \in V) \|F_v(t)\| > \varepsilon$  do
    foreach  $u \in V$  do
         $F_v(t) \leftarrow \sum f_{repulsion}(u, v) + \sum f_{attraction}(u, v)$ 
    foreach  $u \in V$  do
         $p_u \leftarrow p_u + Accélération(F_u(t))$ 
     $t \leftarrow t + 1$ 
return  $p_{fin}$ 

```

Paramètres

V : Vertex (Nœud)

E : Edges

ε : threshold

K : max itérations

p_{init} : layout initial

p_{fin} : layout à la fin

On insère une graphe, des positions initiales de tous les nœuds, et un chiffre qui indique l'itération maximale dans l'algorithme. On peut également préciser un *threshold* qui décidera si on continue le calcul des forces et à les appliquer ou non. C'est-à-dire, quand la Force devient trop petite on n'a pas besoin de répéter le calcul des forces puisque il n'y aura pas beaucoup de mouvement des nœuds.

Normalement, on utilise deux boucles majeures. La première est utilisée dans le but de calculer la force repulsive et la force attractive entre deux nœuds. Ensuite, on applique l'accélération sur la somme de Force puis on l'additionne à ses positions précédentes **afin d'obtenir une nouvelle position**. Après des itérations suffisantes, on trouvera les dernières positions de chaque nœud dont l'état est '*stable*'.

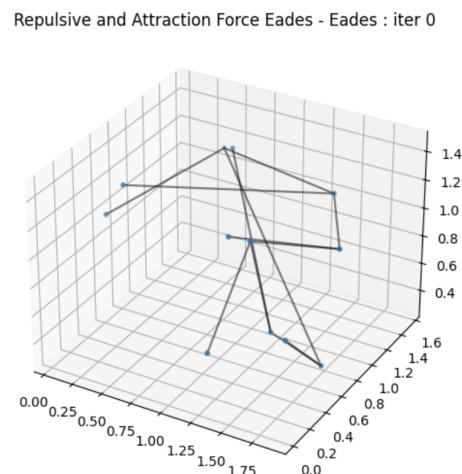
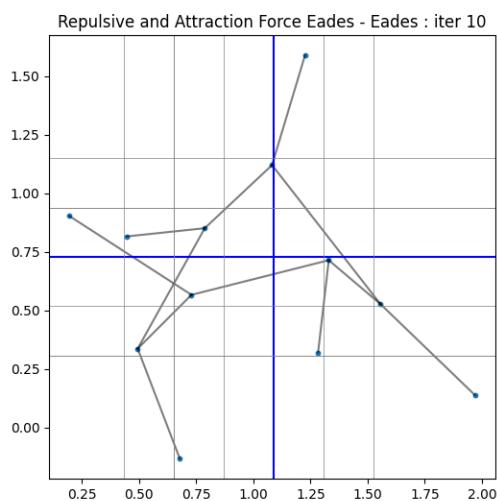
3.2. La structure des données

3.2.1. Quad Tree

- path : sda_project > quadtree.py
- Une Classe de **Region** contient centreX, centreY, width, height
- Une Classe de QuadTree contient
 - une Region
 - une sous-region(**4 parties**)
 - le profondeur d'arbre
 - le nombre de noeuds que chaque parent possède
- Une Classe de QuadTree n'a qu'une méthode 'insert'
- Le résultat de l'utilisation du QuadTree est une forme d'un plan

3.2.2. Octet Tree

- path : sda_project > octettree.py
- Une Classe de **Space** contient centreX, centreY, centreZ, width, length, height
- Une Classe de OctetTree contient
 - une Space
 - une sous-region(**8 parties**)
 - le profondeur d'arbre
 - le nombre de noeuds que chaque parent possède
- Une Classe de OctetTree n'a qu'une méthode 'insert'
- Le résultat de l'utilisation du OctetTree est une forme d'une espace en 3 dimension

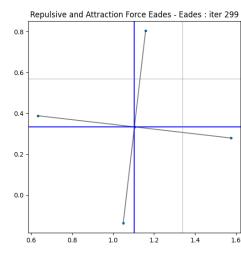
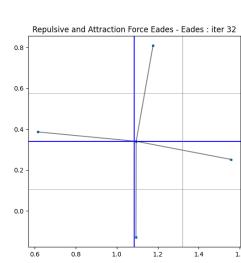
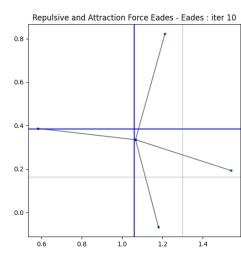
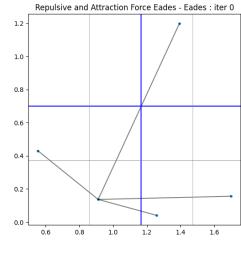


4. L'expérimentation et son résultat

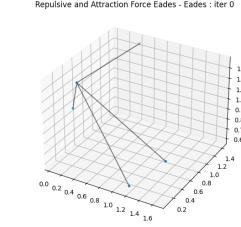
4.1. La qualité du résultat visuel (les nouvelles positions des noeuds)

4.1.1. Spring Embedder by Eades

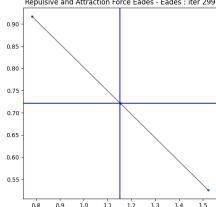
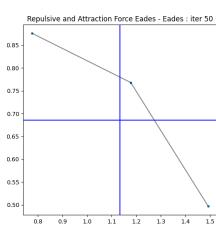
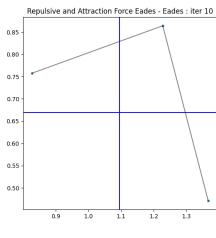
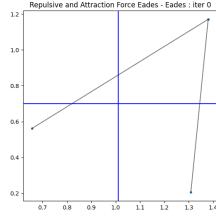
CH4



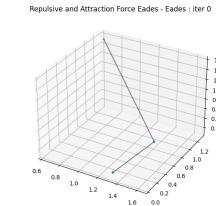
OctetTree(0,299 iter)



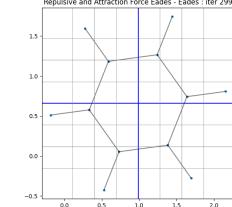
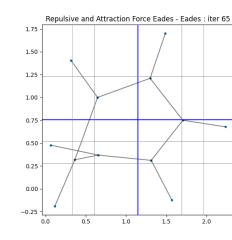
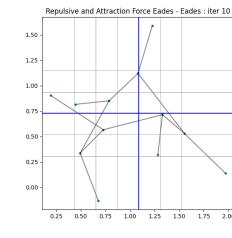
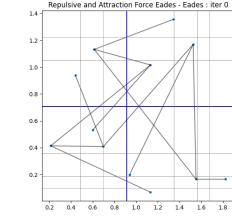
H2O



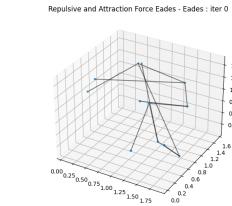
OctetTree(0,299 iter)



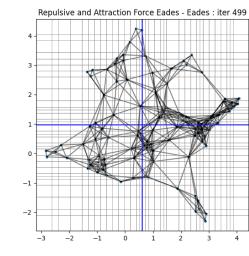
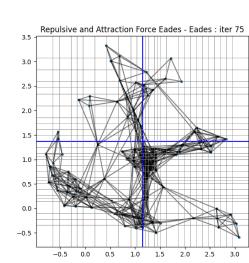
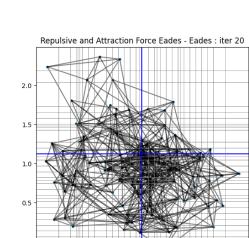
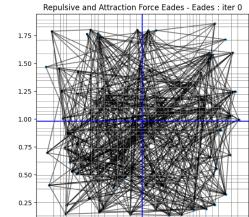
Benzène



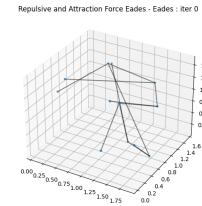
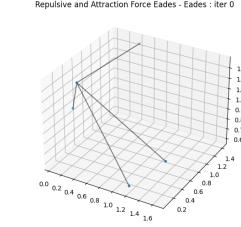
OctetTree(0,299 iter)



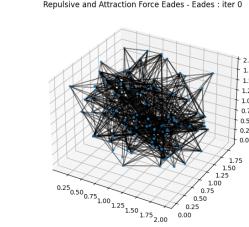
Node 100

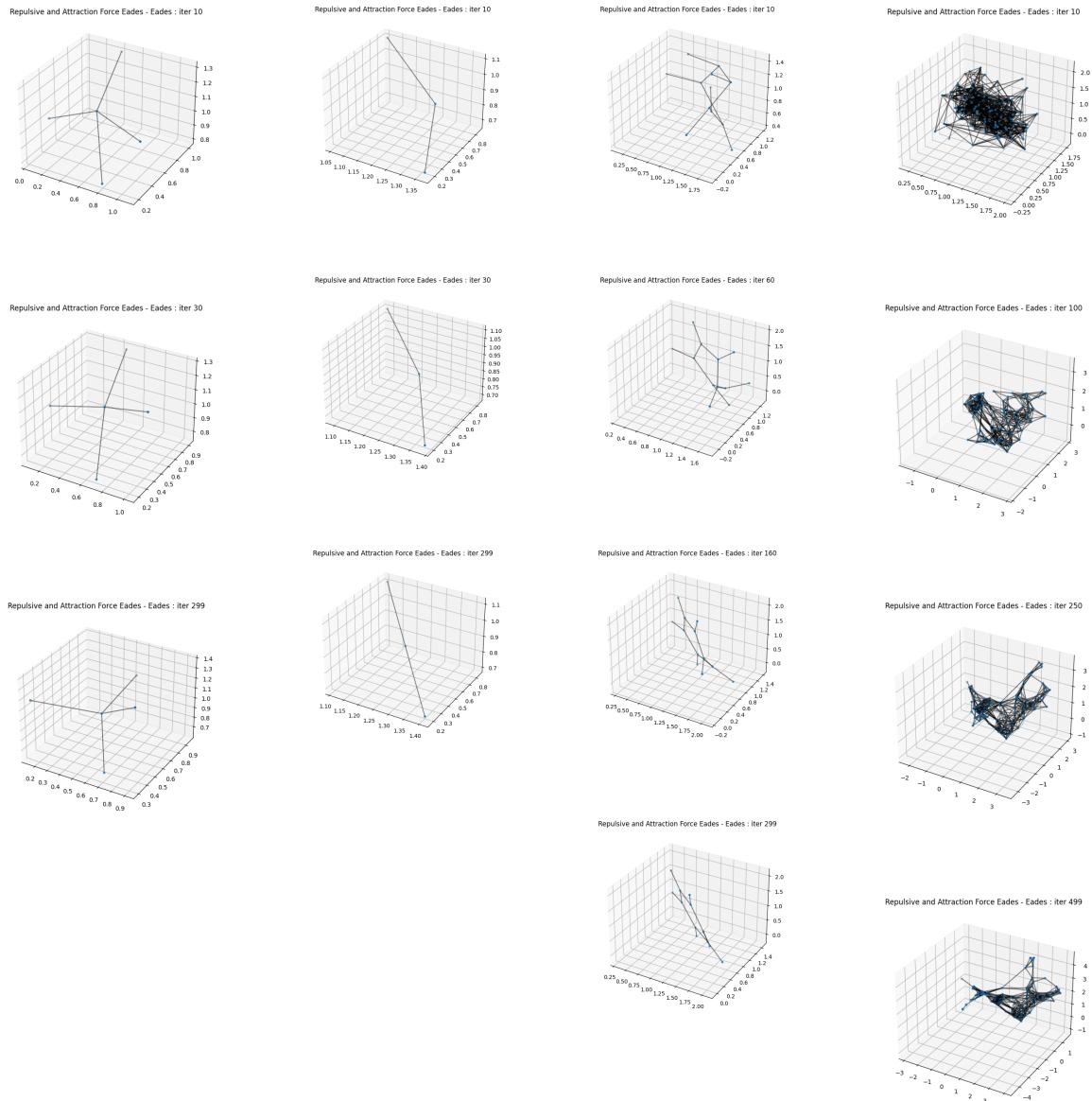


OctetTree(0,299 iter)



OctetTree(0,499 iter)





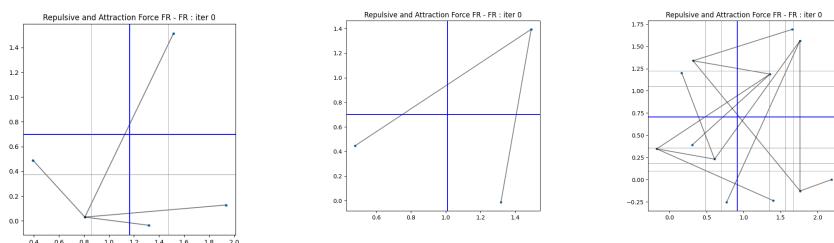
4.1.2. Variant by Fruchterman & Reingold

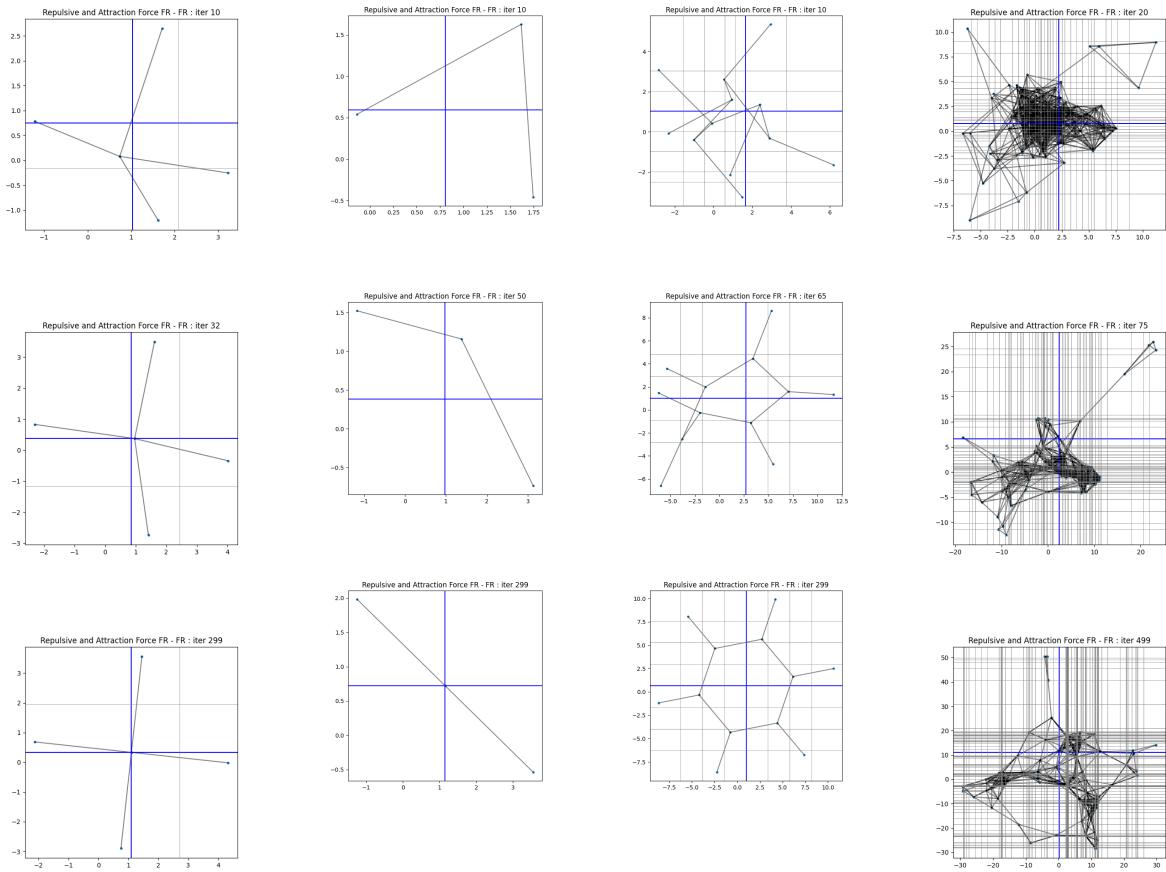
CH4

H2O

Benzène

Node 100



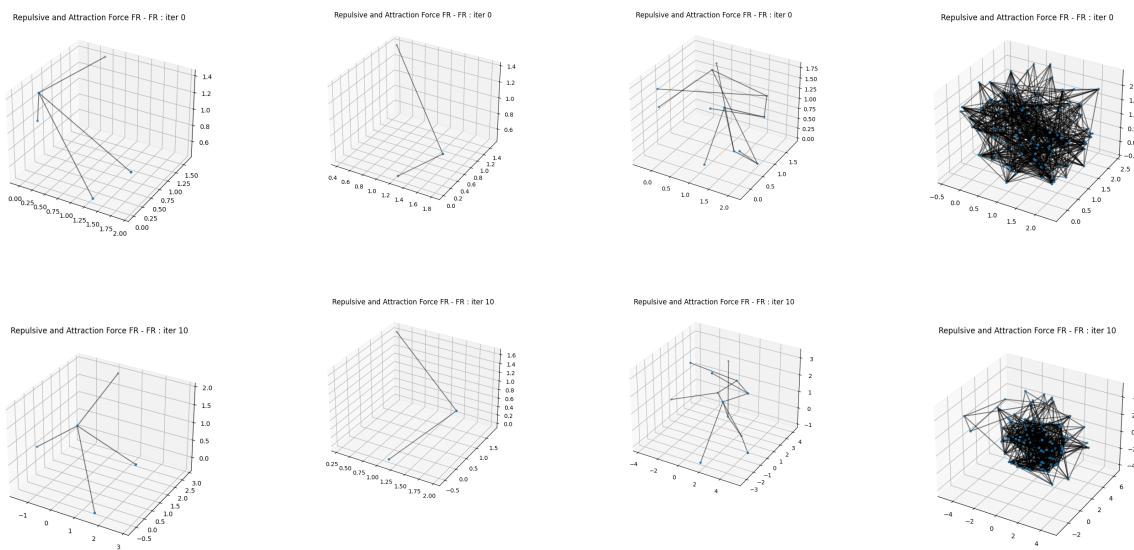


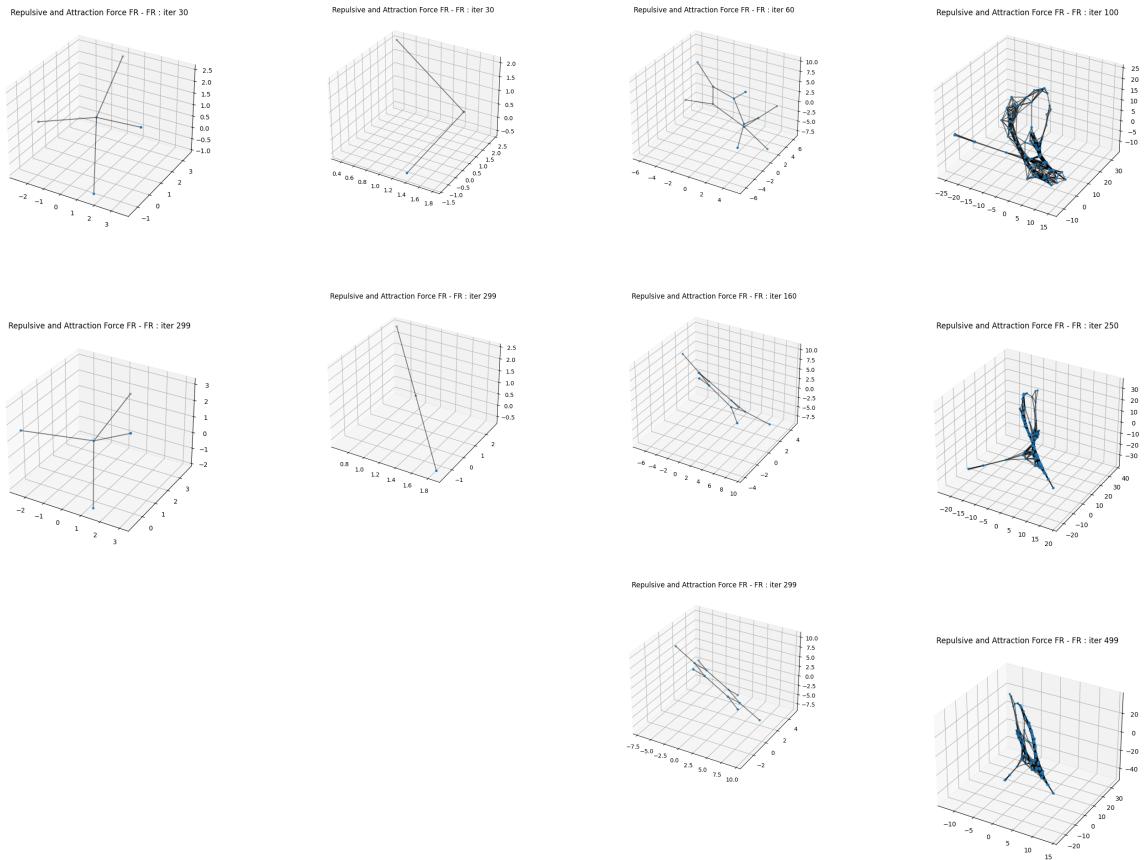
OctetTree(0,299 iter)

OctetTree(0,299 iter)

OctetTree(0,299 iter)

OctetTree(0,499 iter)





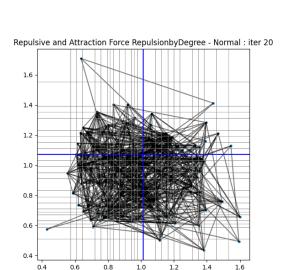
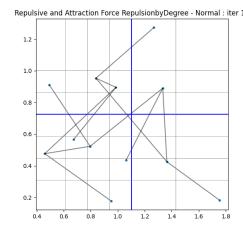
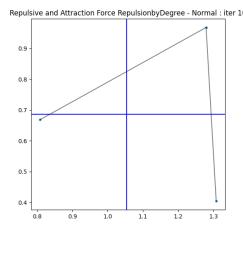
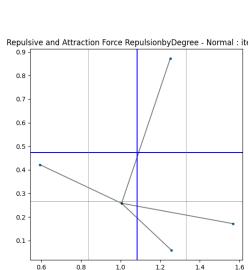
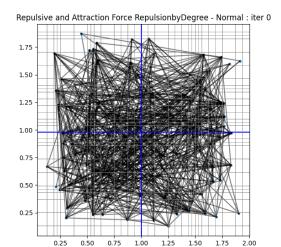
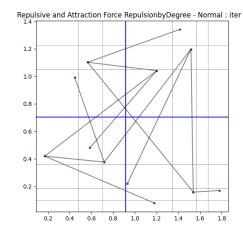
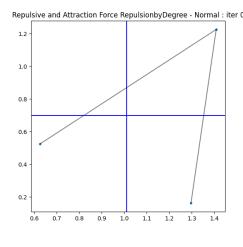
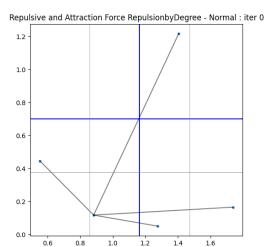
4.1.3. La force de repulsion par degré du nœud + La force d'attraction classique

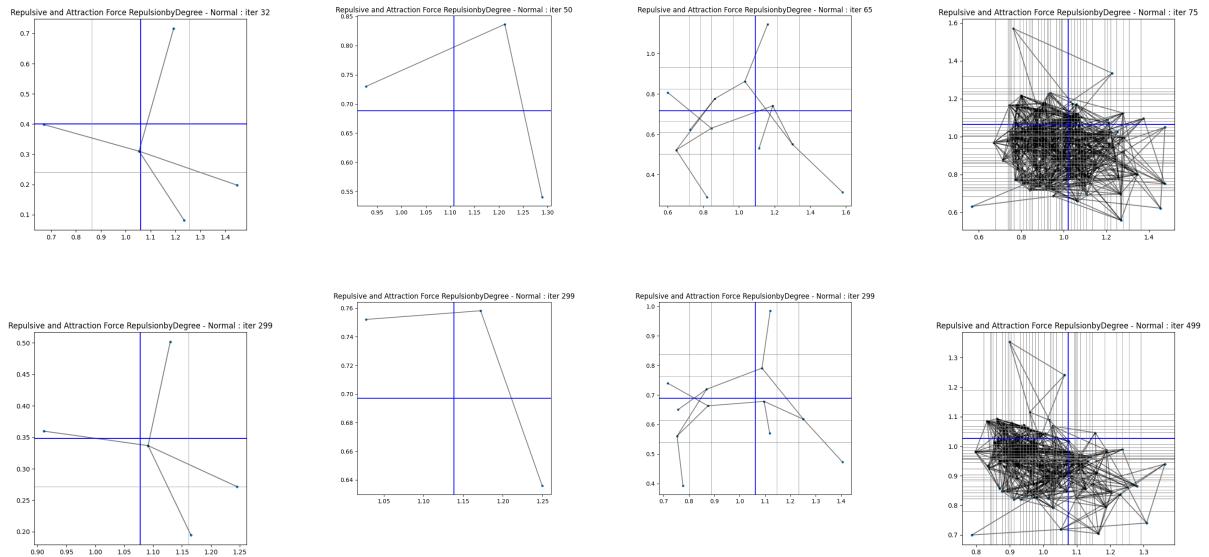
CH4

H₂O

Benzène

Node 100



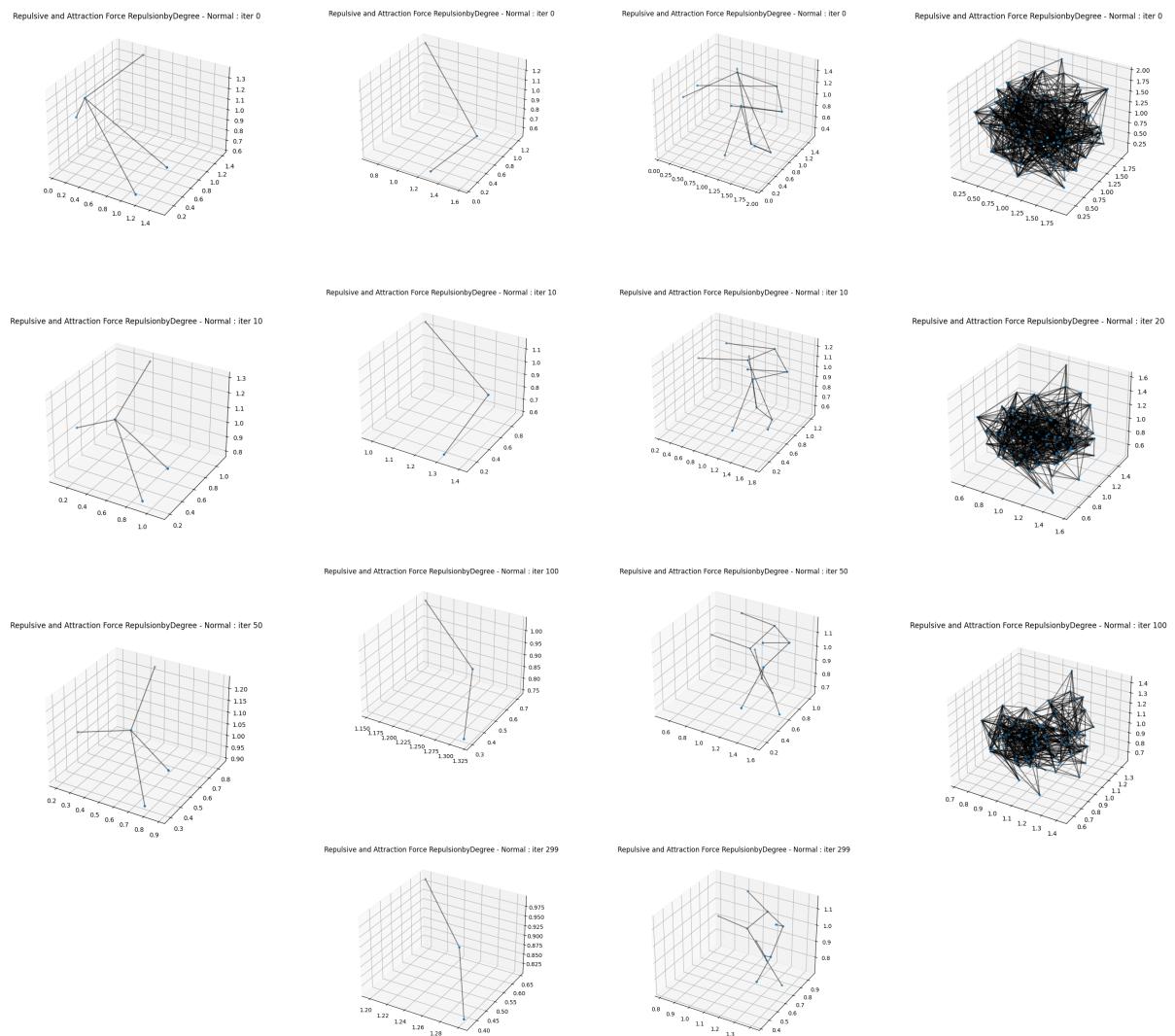


OctetTree(0,299 iter)

OctetTree(0,299 iter)

OctetTree(0,299 iter)

OctetTree(0,499 iter)





4.1. L'efficacité de l'algorithme

4.1.1. Time cost

- itération = 300
- octetree = True
- Graphe = une graphe avec 100 nœuds (`G = nx.random_geometric_graph(100)`)
- Les positions initiales des noeuds sont identiques (`random.seed(42)`)

Dans l'ordre efficace au niveau du temps,

- *repulsion par degré + La force d'attraction classique*
- *Fruchterman & Reingold*
- *Eades*

<u>No drawing</u>	Eades	Fruchterman & Reingold	repulsion par degré + La force d'attraction classique
Total cost	16,44962 s	3,92111 s	0,92185 s
Repulsion cost	15.54758 s	3,03029 s	0,05522 s
Attraction cost	0,33121 s	0,29930 s	0,31773 s

<u>Drawing</u>	Eades	Fruchterman & Reingold	repulsion par degré + La force d'attraction classique
Total cost	131,0333 s	124,2560 s	119,7646 s

<u>Drawing</u>	Eades	Fruchterman & Reingold	repulsion par degré + La force d'attraction classique
Repulsion cost	14,9590 s	2,7594 s	0,04665 s
Attraction cost	0,3012 s	0,2839 s	0,2886 s

4.1.2. L'utilisation de la mémoire

- itération = 300
- octetree = True
- Algorithme du calcul des forces = **Eades**

Mémoire utilisée	algo.run()	algo.run() la 1ère itération	Tree.insert() de la 1ère itération	Calcul de la Force de repulsion de la 1ère itération	Calcul de la Force d'attraction de la 1ère itération
Benzène.dot	75,26562 MB	9,78126 MB	trop petite	trop petite	trop petite
Node 50	67,28124 MB	11,28124 MB	0,01562 MB	0,07813 MB	trop petite
Node 100	58,15625 MB	18,68750 MB	0,03125 MB	0,034376 MB	trop petite