2. Partie Logique (Prolog) [8 points][8% de votre note finale]

*In this part of the comprehensive assignment, we will focus on the third step of the parallel DBSCAN algorithm introduced in the previous concurrent assignment. We will merge intersecting clusters from adjacent partitions.*

*The parallel DBSCAN algorithm extracts the clusters of a set by subdividing the region into a number of overlapping partitions. The fact that these partitions overlap with each other implies that some points (at the periphery of the partitions) might belong to more than one partition. Consequently, some clusters may contain the same point(s) and are then said to intersect. In this case, these clusters must be merged because they should in fact constitute one large cluster covering more than one partition. The merging can be simply done by changing the label of one of the clusters to the one of the second.*

*We will solve this problem by using a simple but not very efficient algorithm. We will apply it on a subset of the data generated in the previous step.*

Pour cette partie du projet intégrateur, nous nous concentrerons sur la troisième étape de l'algorithme DBSCAN parallèle, introduite dans la partie concurrente du projet. Nous allons donc fusionner les groupes provenant de partitions adjacentes.

L'algorithme parallèle du DBSCAN extrait les groupes d'un ensemble de points en subdivisant la région en un certain nombre de partitions qui se chevauchent. Le fait que ces partitions se superposent implique que certains points (à la périphérie des partitions) peuvent appartenir à plus d'une partition. Conséquemment, certains groupes peuvent avoir des points en commun, c'est-à-dire qu'ils s'intersectent. Lorsque cela se produit, ces groupes doivent être fusionnés car ils constituent, dans les faits, un seul grand groupe s'étendant sur plus d'une partition. Cette fusion se fait simplement en changeant l'étiquette d'un groupe pour l'étiquette du groupe avec lequel il se fusionne.

Nous vous demandons de résoudre ce problème en utilisant un algorithme simple mais pas très efficace. Vous appliquerez cet algorithme à un sous-ensemble des données obtenues à l'étape précédente.

**Algorithme**

*The input to the algorithm is a knowledge base describing the current clustering over the partitions. It is made of facts using the partition predicate:*

```
partition(PARTITION_ID, POINT_ID, X, Y, CLUSTER_ID).
```

*From this predicate, we want to verify if clusters in one partition intersect with clusters of adjacent partitions. This is accomplished by the simplified algorithm shown on the next page. The output will be a final list of (merged) clusters in the form:*

```
[[POINT_ID, X,Y,CLUSTER_ID], …]
```

L'entrée de cet algorithme sera une base de faits décrivant les groupes obtenus à travers les différentes partitions. Cette base est constituée de faits décrits à l'aide du prédicat suivant :

```
partition(PARTITION_ID, POINT_ID, X, Y, CLUSTER_ID).
```

A partir de ce prédicat, nous voulons vérifier si les groupes d'une partition intersectent avec les groupes des partitions adjacentes. Ceci est réalisé à partir de l'algorithme présenté ci-dessous. La sortie de cet algorithme sera une liste de groupes (fusionnés) représentée ainsi :

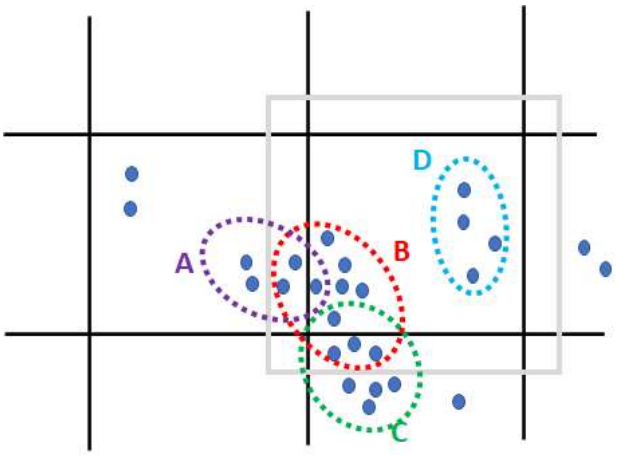
```
[[POINT_ID, X,Y,CLUSTER_ID], …]
```

```
ClusterList := 0     // list of clusters to be produced
for each partition P in knowledge base {
    for each cluster C in P {
       for each cluster C in P {
            I := C ∩ ClusterList // List of points in ClusterList intersecting with points in C
           for each label L in I {
              change label L in ClusterList to Label(C)
           }

          ClusterList := C ∪ ClusterList
       }
    }
}
```

*As an example, consider the figure below.*

Illustrons le fonctionnement de cet algorithme avec la figure suivante.




*Suppose we have the 5 points of cluster A and the 8 points of cluster C in current* `ClusterList`*. We are now processing the partition containing clusters B and D. We first consider cluster D; it has no intersection with the points in* `ClusterList` *so its 4 points will simply be inserted to the list. Now if we consider cluster B, it has 3 points intersecting with cluster A and 4 points intersecting with cluster C. These 7 points will constitute the intersection set I and the labels of I are A and C. Consequently, the cluster label of all points in* `ClusterList` *having label A will be changed to B and same for the points having label C. Finally, the points in B are inserted into the* `ClusterList`*.*

*Note that the proposed algorithm does not check if partitions are adjacent before computing cluster intersection. This is not efficient since clusters in non-adjacent partitions cannot have intersection but for simplicity, we will accept these useless computations and proceed as proposed in the algorithm. One could also initialize the* `ClusterList` *with all clusters from one first partition instead of starting with an empty list but again efficient is not a concern here. Note that for computing the intersection, use the POINT_ID to compare the points and not their X,Y coordinates.*

Supposons que nous avons les 5 points du groupe A et les 8 points du groupe C dans la liste courante `ClusterList`. Nous sommes à traiter la partition contenant les groupes B et D. Considérons d'abord le groupe D, celui-ci n'a pas d'intersection avec les points de `ClusterList`, alors ses 4 points sont simplement insérés dans cette liste. Maintenant si nous considérons le groupe B, celui-ci a 3 points en intersection avec le groupe A et 4 points en intersection avec le groupe C. Ces 7 points forment l'ensemble d'intersection I et les étiquette de I sont A et C. Il s'ensuit donc que tous les points dans `ClusterList` ayant l'étiquette A et C seront changés en B. Finalement les points restants de B sont insérés dans `ClusterList`.

Notons que cet algorithme ne vérifie si les partitions sont adjacentes avant de calculer les intersections. Ceci n'est pas très efficace car nous savons que les groupes provenant de partitions non-adjacentes ne peuvent pas avoir d'intersection. Toutefois, afin de simplifier notre problème, nous allons procéder ainsi et accepter ces quelques inefficacités. Il serait aussi possible d'initialiser la `ClusterList` avec tous les groupes provenant d'une partition initiale (au lieu de l'ensemble vide) mais encore, notre souci n'est pas l'efficacité. Notons enfin que pour le calcul des intersections, il est préférable d'utiliser le POINT_ID afin de comparer les points et non leur coordonnée X,Y.

**Programmation**

*We provide you with the clustering results of 7 partitions described in 7 csv files.*

*You first run the provided helper predicate* `import` *that creates the knowledge base using the predicate partition. Just make sure the csv files are in the working directory of your Prolog console. From this knowledge base write Prolog predicates that will implements the cluster merging algorithm.*

*Your solution must include a helper predicate called* `mergeClusters` *and that produces the list of all points with their cluster ID.*

Nous vous donnons les groupes provenant de 7 partitions et décrits dans 7 fichiers csv.

Vous devez d'abord lancer le prédicat assistant `import` qui va créer la base de faits des partitions. Veuillez simplement vous assurer que les fichiers se trouve dans votre répertoire de travail Prolog. A partir de cette base de faits, écrire les prédicats Prolog qui réaliseront l'algorithme de fusion des groupes.

Votre solution doit inclure un prédicat appelé `mergeClusters` et qui produit la liste de tous les points avec leur étiquette de groupe.

```
?- import.
partition(65, 1345, 40.750304, -73.952031, 65000001).
partition(65, 6017, 40.760146, -73.957873, 65000002).
partition(65, 17457, 40.760213, -73.955471, 65000003).
partition(65, 18582, 40.750299, -73.952027, 65000001).
partition(65, 20050, 40.750365, -73.952127, 65000001).
partition(65, 25351, 40.760153, -73.955467, 65000003).
partition(65, 34767, 40.758621, -73.957704, 65000004).
partition(65, 36487, 40.758621, -73.957704, 65000004).
…

?- mergeClusters(L),open('clusters.txt',write,F),write(F,L),close(F).
L=[…
```

*Finally, for each predicate include a test predicate that demonstrates the results obtained using it. You must also adequately comment each of your predicates. See example below.*

Finalement, pour chaque prédicat inclure un prédicat `test` démontrant les résultats obtenus en l'utilisant. Vous devez aussi commenter adéquatement chacun de vos prédicats. Par exemple :

```
% relabel/4
% relabels the points of cluster O with label R
% relabel(O,R,clusterListIn, clusterListOut)
relabel(…

?- test(relabel).
relabel(33, 77, [[1,2.2,3.1,33], [2,2.1,3.1,22], [3,2.5,3.1,33], [4,2.1,4.1,33], [5,4.1,3.1,30]],Result)
[[1,2.2,3.1,77],[2,2.1,3.1,22],[3,2.5,3.1,77],[4,2.1,4.1,77],[5,4.1,3.1,30]]
true .
```

*Where the definition of `test(relabel)` is as follows:*

Avec la définition suivante:

```
test(relabel) :- write('relabel(33, 77,
                    [[1,2.2,3.1,33], [2,2.1,3.1,22], [3,2.5,3.1,33], [4,2.1,4.1,33],[5,4.1,3.1,30]],Result)'),nl,
                  relabel(33, 77,
                    [[1,2.2,3.1,33], [2,2.1,3.1,22], [3,2.5,3.1,33], [4,2.1,4.1,33], [5,4.1,3.1,30]],Result),
                   write(Result).
```

Hint*: Since the main loops are across all partitions and all clusters, it is also possible to work from a global list of clusters, removing the partition information. This list can be obtained as below.*

*Indice*: Puisque la boucle principale se fait à travers toutes les partitions et tous les groupes, il est aussi possible de fonctionner à partir d'une liste globale de groupes (retirant l'information sur les partitions). Cette liste globale s'obtient ainsi :

```
?- findall([D,X,Y,C],partition(_,D,X,Y,C),L).
L = [[1345, 40.750304, -73.952031, 65000001], [6017, 40.760146, -73.957873, 65000002], [17457, 40.760213, -73.955471,
65000003], [18582, 40.750299, -73.952027, 65000001], [20050, 40.750365, -73.952127, 65000001], [25351, 40.760153, -
73.955467|...], [34767, 40.758621|...], [36487|...], [...|...]|...].
```