Compléments d'informatique

Projet 3 : clustering hiérarchique

25 novembre 2025

L'objectif concret de ce projet est d'implémenter un algorithme de clustering hiérarchique, populaire en science des données, et de l'appliquer ensuite à la construction d'un arbre phylogénétique. L'objectif pédagogique est de vous faire utiliser des structures de données de type arbre, liste, et dictionnaire, dans le cadre d'une application réelle. Ce projet est à réaliser **seul ou à deux**. La date limite de remise est précisée sur eCampus et sur Gradescope.

1 Clustering hiérarchique

Principe général. Soit un ensemble d'objets $\{o_1, \ldots, o_N\}$ et une mesure de distance $d(o, o') \in \mathbb{R}$ entre ces objets. L'objectif du clustering est de partitionner ces objets en sous-groupes, appelés "clusters", de sorte à ce que les objets dans un même cluster soient aussi proches que possible les uns des autres, selon d, et aussi éloignés que possible des objets appartenant aux autres clusters. Un algorithme de clustering populaire est le clustering (ou regroupement) hiérarchique [1], qui construit un ensemble de clusters organisés de manière hiérarchique. L'idée générale de cet algorithme pour obtenir k clusters est la suivante :

1. Chaque objet est assigné à son propre cluster.
2. Tant qu'il n'y a pas exactement k clusters, on cherche les deux clusters les plus proches et on les fusionne.

Cet algorithme demande de définir une mesure de distance entre clusters à partir de la distance d entre objets. Il existe plusieurs définitions possibles pour cette distance. Dans ce projet, on utilisera comme distance entre deux clusters la distance minimale entre un objet dans le premier cluster et un objet dans le second. On utilise généralement $k = 1$ pour obtenir une séquence complète de clustering allant de N à 1 cluster. Le processus de

1. https://fr.wikipedia.org/wiki/Regroupement_hiérarchique

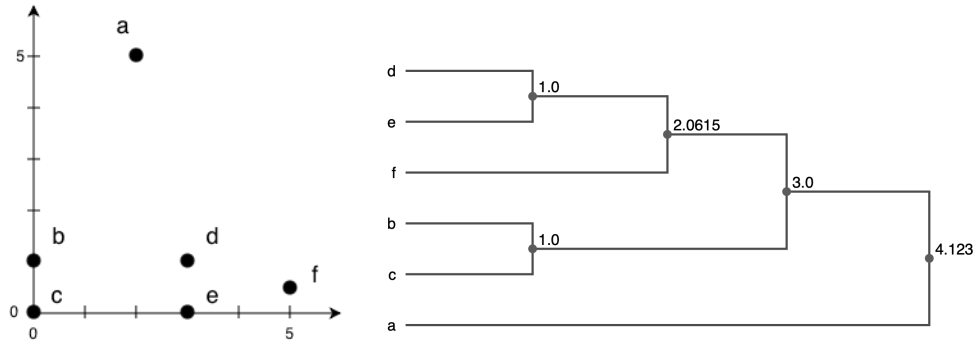

FIGURE 1 – A gauche 6 points dans le plan. A droite, le dendrogramme obtenu en utilisant la distance euclidienne. Les feuilles sont marquées par les noms des points et les nœuds intérieurs par les distance entre les clusters correspondant aux sous-arbres à droite et à gauche.

fusion de clusters est alors représenté par un **dendrogramme** tel que celui de la figure 1. Il s'agit d'un arbre binaire dont chaque feuille correspond à un objet et chaque nœud intérieur représente une opération de fusion lors de l'étape 2.1 de l'algorithme ci-dessous. Généralement, la position d'un nœud intérieur(sur l'abscisse dans la figure) est déterminée par la distance entre les clusters fils lors de la fusion.

Implémentation efficace. L'algorithme tel que décrit ci-dessus a une complexité $O(N^3)$ s'il est implémenté naïvement, ce qui le rendrait trop lent pour des valeurs de N même relativement faible. Il peut cependant être implémenté plus efficacement de la manière suivante :

1. Chaque objet est assigné à son propre cluster.
2. On trie les paires d'objets par ordre de distance croissante.
3. Pour chaque paire (o, o') prise par ordre croissant de distance et tant qu'il n'y a pas k clusters :
   (a) Si o et o' appartiennent au même cluster, on ne fait rien.
   (b) Sinon on fusionne les clusters de o et o'

Pour que cet algorithme soit efficace, il faut encore que les étapes 3.(a) et 3.(b) puissent être implémentées de manière efficace. C'est possible en procédant de la manière suivante. On utilise un dictionnaire de paires (o, S), où o est un objet et S est le cluster dans lequel l'objet o se trouve à l'étape courante (initialement le singleton $\{o\}$). A l'étape 3.(a), on peut alors vérifier facilement si o et o' sont dans le même cluster en utilisant la fonction `dictSearch` du dictionnaire et en comparant les clusters retrouvés. A l'étape 3.(b), soit (o, S) et (o, S') les deux objets et leurs clusters, à fusionner. On prend l'un des clusters, par exemple S, et on lui ajoute les objets du second, S', un par un. Lors de cet ajout, on

met également à jour l'information dans le dictionnaire que le cluster de ces objets passe de S' à S. Une manière d'optimiser ces opérations est de prendre comme ensemble S' dont on insère les objets dans S, le cluster le plus petit des deux clusters à fusionner.

Construction du dendrogramme. On vous demandera ci-dessous d'écrire une fonction construisant le dendrogramme complet ($k = 1$). Dans ce cas, on représentera lors de l'implémentation, un cluster S par le sous-arbre du dendrogramme correspondant à ce cluster, c'est-à-dire le sous-arbre dont les feuilles correspondent aux objets du cluster. Le dictionnaire contiendra alors des paires $(o, \mathcal{T})$, où $\mathcal{T}$ sera le sous-arbre dans lequel se trouve l'objet o à l'étape courante de l'algorithme. Soit $(o, \mathcal{T})$ et $(o', \mathcal{T}')$ les deux objets et leur sous-arbre à fusionner. L'étape 3.(b) de l'algorithme reviendra à modifier l'arbre $\mathcal{T}$, en lui ajoutant une nouvelle racine avec à sa gauche l'arbre $\mathcal{T}$ initial et à sa droite l'arbre $\mathcal{T}'$ et en parcourant ensuite les feuilles de $\mathcal{T}'$ pour modifier leur sous-arbre d'appartenance de $\mathcal{T}'$ à $\mathcal{T}$ dans le dictionnaire. Comme précédemment, il sera plus efficace de considérer comme arbre $\mathcal{T}'$ le plus petit des deux arbres à fusionner, de manière à minimiser les mises à jour dans le dictionnaire.

2 Arbre phylogénétique

Les applications du clustering hiérarchique sont nombreuses. On s'intéressera ici à son application à la reconstruction d'un arbre phylogénétique [2]. Un arbre phylogénétique est un arbre qui représente les liens de parenté entre des espèces, chaque nœud de l'arbre représentant l'ancêtre commun de ses descendants. Le clustering hiérarchique permet de reconstituer un arbre phylogénétique entre N espèces sur base d'une comparaison de leurs séquences d'ADN. Soit deux espèces o et o' et D et D' leurs séquences d'ADN respectives, c'est-à-dire une chaîne de caractères composée des lettres A, C, G, ou T, qu'on supposera de même longueur fixe n. On mesurera la distance entre o et o' par l'expression suivante [3] :

$$d(o, o') = -\frac{1}{2}\ln(1 - 2P - Q) - \frac{1}{4}\ln(1 - 2Q), \tag{1}$$

où P et Q sont respectivement les proportions de *transversions* et *transitions* entre les deux séquences d'ADN de o et o' (rapportée à la longueur n des séquences). Un transition est une transformation d'un A en un G ou d'un C en un T (ou inversement) à une position d'une séquence à l'autre. Toutes les autres transformations (par exemple d'un A en un T ou d'un G en un T) sont des transversions. Cette mesure de distance prend en compte le fait que les transitions sont plus fréquentes que les transversions.

2. https://fr.wikipedia.org/wiki/Arbre_phylogénétique
3. Voir par exemple ici : `https://www.megasoftware.net/mega1_manual/Distance.html`.

3 Implémentation

On vous fournit des implémentations d'arbre binaire générique (`BTree.c/.h`), de table de hachage (`Dict.c/.h`), et de liste liée (`LinkedList.c/.h`) telles que vues au cours. On vous demande d'ajouter deux fonctions dans `BTree.c` et d'implémenter complètement les deux modules suivants :

- `HierarchicalClustering.c/.h` implémentant le clustering hiérarchique générique
- `Phylogenetic.c/.h` permettant d'utiliser le clustering hiérarchique pour reconstruire un arbre phylogénétique

Par ailleurs, deux fichiers, `main-features.c` et `main-phylo.c`, vous sont fourni pour vos tests.

Ces différents fichiers sont décrits ci-dessous.

3.1 Fichiers `BTree.c` et `BTree.h`

L'implémentation est celle vue au cours et utilisée dans le TP3. Pour vous faciliter l'implémentation des fonctions suivantes, on vous demande d'ajouter ces deux fonctions dans `BTree.c` :

```
void btMapLeaves(BTree *tree, BTNode *n, void (*f)(void *data, void *fparams),
void *fparams);
```

Cette fonction applique la fonction `f` en argument aux données stockées à chacune des feuilles du sous-arbre de `tree` dont `n` est la racine. `fparams` sera donné comme second argument à `f` lors de chaque appel.

```
void btMergeTrees(BTree *lefttree, BTree *righttree, void *data)
```

Cette fonction modifie l'arbre `leftree` de manière à ce qu'il ait une nouvelle racine avec `data` comme données, l'arbre `lefttree` comme sous-arbre à gauche et l'arbre `righttree` comme sous-arbre à droite. L'arbre `righttree` est libéré de la mémoire et ne doit plus être utilisé.

3.2 Fichiers `HierarchicalClustering.c` et `HierarchicalClustering.h`

Ce fichier implémente le clustering hiérarchique. Le résultat est stocké dans une structure de données de type `Hclust` opaque que vous devrez définir par vous-même. Les fonctions suivantes sont à implémenter :

```
Hclust *hclustBuildTree(List *objects, double (*distFn)(const char *, const char *,
void *), void *distFnParams);
```

Fonction principale qui construit le dendrogramme complet selon la stratégie efficace décrite dans la section 1. La signification des arguments est la suivante : `objects` est une liste contenant les noms des objets (des chaînes de caractères) et `distFn` est une fonction prenant comme argument deux noms d'objet et l'argument `distFnParams` et renvoyant la distance

entre les deux objets. La structure renvoyée, de type `Hclust`, doit contenir suffisamment d'information pour pouvoir implémenter les fonctions ci-dessous.

`void hclustFree(Hclust *hc)` : libère l'espace mémoire pris par `hc`.

`int hclustDepth(Hclust *hc)` : renvoie la profondeur du dendrogramme.

`int hclustNbLeaves(Hclust *hc)` : renvoie le nombre de feuilles du dendrogramme.

`void hclustPrintTree(FILE *fp, Hclust *hc)` : écrit dans le fichier `fp` une représentation dans le format Newick du dendrogramme. Le format demandé sera décrit sur la page Ecampus du projet.

`List *hclustGetClustersDist(Hclust *hc, double distanceThreshold)` : renvoie une liste de listes contenant les noms des objets des clusters qui auraient été obtenus si on avait arrêté la fusion des clusters dès que la distance entre les clusters à fusionner devenait strictement supérieure à `distanceThreshold`.

`List *hclustGetClustersK(Hclust *hc, int k)` : renvoie une liste de listes contenant les noms des objets des clusters qui auraient été obtenus si on avait arrêté la fusion des clusters après avoir obtenu exactement k clusters. Cette fonction, comme la précédente, devrait renvoyer ce résultat le plus efficacement possible en se basant sur le dendrogramme complet construit par l'appel à `hclustBuildTree`.

`BTree *hclustGetTree(Hclust *hc)` : renvoie un arbre binaire encodant le dendrogramme. Cet arbre devra être tel que chaque feuille contienne le nom de l'objet correspondant et chaque nœud interne contienne un pointeur vers un double contenant la distance entre les clusters correspondant aux sous-arbres à gauche et à droite lors de la fusion. [4]

3.3 Fichiers `Phylogenetic.c` et `Phylogenetic.h`

Ce fichier implémente la construction d'un arbre phylogénétique à partir d'un fichier. Ce fichier est un fichier au format CSV. Chaque ligne correspond à une espèce et contient le nom de l'‘espèce et sa séquence d'ADN (en lettres capitales), séparés par une virgule. Les fonctions à implémenter sont les suivantes :

`double phyloDNADistance(char *dna1, char *dna2)` : calcule la distance $d(o, o')$ décrite dans l'équation (1) entre les deux séquences d'ADN données en argument. Si ces deux séquences n'ont pas la même longueur, on tronquera à droite la plus longue à la longueur de la plus courte.

4. Cette fonction n'est pas strictement nécessaire à l'implémentation mais elle est introduite, d'une part, pour vous forcer à utiliser cette représentation en interne de votre code (il devrait y avoir un pointeur vers cet arbre directement dans votre structure `HClust_t`) et, d'autre part, pour nous permettre de tester plus facilement votre code.

`Hclust *phyloTreeCreate(char *filename)` : renvoie le clustering hiérarchique obtenu à partir des espèces dans le fichier `filename`.

3.4 Fichiers `main-features.c` et `main-phylo.c`

Les fichiers `main-features.c` et `main-phylo.c` vous sont fournis pour tester votre implémentation.

Le fichier `main-features.c` génère un exécutable appelé `hcfeatures` [5] qui permet d'appeler le clustering hiérarchique sur un fichier contenant des objets décrits par des valeurs numériques et en utilisant la distance euclidienne pour les comparer. Vous pouvez regénérer l'exemple de la figure 1 à partir du fichier `features-simple.csv`. Le fichier `features-breastcancer.csv` reprend des données réelles et permet de montrer un intérêt pratique du clustering pour faire de la classification. Chaque ligne de ce fichier correspond à une tumeur, maligne ou bénigne, décrite par différentes mesures relatives à sa taille et forme [6]. La nature de la dernière tumeur, `unknown`, dans le fichier est inconnue. En observant les résultats du clustering hiérarchique, essayez de déterminer si elle est maligne ou bénigne. Un fichier supplémentaire, `features-zoo.csv` [7], contient 87 animaux appartenant à 5 grandes familles (mammifères, oiseaux, poissons, reptiles, et insectes) et décrits par différentes caractéristiques. En utilisant `hclustGetClustersK`, il est intéressant de vérifier si le clustering permet de retrouver les 5 grandes familles à partir de ces caractéristiques.

Le fichier `main-phylo.c` génère un exécutable appelé `hcphylo` qui permet de construire un arbre phylogénétique à partir d'un fichier de séquences d'ADN. Nous vous fournissons deux fichiers pour vos tests, `dna-species.csv` et `dna-virus.csv`. Le fichier `dna-species.csv` contient les séquences d'ADN réelles [8] de 69 espèces animales. Il est intéressant de regarder si l'arbre phylogénétique que vous obtenez permet de retrouver des relations de parenté attendues entre espèces (regardez en particulier où se trouve l'homme dans le dendrogramme).

4 Conseils d'implémentation

Commencez par compléter le fichier `BTree.c`. Les fonctions demandées ne font pas plus que quelques lignes.

Attaquez ensuite l'implémentation de `HierarchicalClustering.c`. Votre structure `Hclust_t` devrait contenir au minimum un pointeur vers l'arbre binaire contenant le dendrogramme. L'implémentation de la fonction `hclustBuildTree` doit suivre à la lettre la description faite

5. Tapez `./hcfeatures` dans le terminal pour voir les arguments acceptés.
6. Ces données viennent de cette source : `https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic`.
7. Obtenu de `https://archive.ics.uci.edu/dataset/111/zoo`.
8. Pour ceux qui veulent les détails, tous ces organismes sont des Eukaryotes et la séquence correspond à leur mitochondrie.

ci-dessus. La difficulté principale est de comprendre le fonctionnement des structures utilisées (liste, arbres, et dictionnaire). Relisez le cours pour les détails de ces structures et prenez connaissance des fonctions disponibles dans les fichiers d'entête. On s'attend à ce que la fonction `hclustBuildTree` construise directement le dendrogramme complet (pour $k = 1$) et le stocke dans la structure renvoyée. Pour représenter le dendrogramme, on s'attend également à ce que vous utilisiez un `BTree` avec en chaque nœud intérieur la distance entre les clusters à gauche et à droite lors de la fusion et en chaque feuille le nom de l'objet correspondant. La fonction `hclustGetTree` renverra alors simplement cette structure en $O(1)$.

Les fonctions `hclusPrintTree` et `hclusGetClustersDist` peuvent s'implémenter comme des parcours assez directs de l'arbre binaire. La fonction la plus compliquée du projet est la fonction `hclustGetClustersK`, à notre avis. Si vous voulez l'implémenter de façon efficace, c'est-à-dire sans devoir reconstruire le dendrogramme, vous devrez stocker de l'information supplémentaire dans la structure `Hclust_t`. Cette fonction ne comptera pas pour plus que 2 points sur 20, sur le total du projet.

L'implémentation de `Phylogenetic.c` vous demandera d'écrire le code pour lire le fichier contenant les séquences et de préparer l'appel à `hclustBuildTree`. Pour cette implémentation, inspirez vous et adaptez le fichier `main-features.c` qui vous est fourni.

5 Soumission

Vous devez soumettre (uniquement) les fichiers suivants sur Gradescope :

- `BTree.c`, `HierarchicalClustering.c`, et `Phylogenetic.c`.
- `rapport.txt`

Dans `rapport.txt` vous sont demandés les contributions de chacun au projet, une analyse de complexité de vos fonctions `hclustBuildTree` et `hclustGetKClusters`, ainsi que la type présenti de la dernière tumeur du fichier `features-breastcancer.csv`.

Par la soumission de ce fichier vous attestez également avoir respecté le règlement en matière de plagiat du cours.

Bon travail !