Sars-CoV-2-Challenge

IMPORTANT:

- Degut al nivell de programació dels membres del grup i el temps en el que s'han dut a terme les tasques, ens hem vist obligats a utilitzar solucions "poc ortodoxes" en certs parts de la pràctica. Aquestes solucions depenents del sistema fan que la pràctica funcioni correctament en un sistema Ubuntu 18.04, però no ho podem assegurar en altres SO.

Fitxers per a l'execució:

- sarscovhierarcy.py <directory> -> Fitxer principal. De moment fa k-medoids clustering.
- setup.py <ruta all_sequences.csv> -> Executat com a sudo, crida als altres fitxers necessaris per a la obtenció de mostres.
- libraries.py -> Executat com a sudo, instala unes llibreries que necessitarem per a descarregar les mostres FASTA.
- fasta_finder.py -> Utilitza all_sequences.csv per a descarregar les mostres FASTA, i guarda el que ha tardat a time.txt. Es crida dins de setup.py.
- fasta_fixer.py -> Ens servirà per a substituïr nucleòtids que s'utilitzen com a abreviació. S'executa dins de fasta_finder.py.
- all_sequences.csv -> Dades sobre totes les seqüències en format CSV, ordenades per llargada.
- failsafe.py -> Executat amb python2.6 ens serveix com a guàrdia al provar els algoritmes. Si un algoritme està a punt de fer que l'ordenador es pengi, l'atura.
- Analisi-Algoritmes.ods -> **DEPRECAT** Full de càlcul amb l'anàlisi dels algoritmes. Està deprecat ja que un cop finalitzat l'anàlisi es va poder millorar dràsticament la velocitat del millor algoritme (~13 minuts a ~10 segons).
- NWS_C -> És la millor versió de l'algoritme final, separat de la resta d'algoritmes per comoditat.
- saved_distances.csv -> Tot i que 10 segons és prou ràpid, amb el número de comparacions que necessitem s'allarga molt el temps d'execució. En aquest fitxer guardem les calculacions ja fetes per no haver-les de repetir.
- map.html -> Resultat de k-medoids, representat gràficament.
- time.txt -> Temps que hem tardat en descarregar les mostres.
- Carpeta Algoritmes_Acabats -> Tal com el seu nom indica, aquí hi han tots els algoritmes que hem programat i provat.
- Carpeta fitxers_vells -> Fitxers deprecats ja no necessaris, però que ens poden ser útils en algun moment. Aquests fitxers compilen amb errors i malament, no formen part de l'entrega.

Com executar

Obtenció de mostres:

```
./setup.py <ruta all_sequences.csv>
```

Clasificació de les mostres:

```
./sarscovhierarchy <directori amb les mostres>
```

Executar una comparació:

```
./<nom_algoritme> <fitxer_sequencia_1> <fitxer_sequencia_2>
```

Preprocessament:

Càlcul de medianes:

- Hem descarregat la informació en csv de totes les seqüències, ordenades per llargada ("all_sequences.csv").
- Hem filtrat les dades per país, sense diferenciar les regions.
- Hem calculat la mediana accedint a la posició directament, ja que les dades ja estaven ordenades.
- Hem accedit a la mostra i l'hem descarregat en format FASTA.

Alineament de seqüències:

- Sense RAM a partir de ~26000 caracters (25GB)
- Separar les seqüències no seria possible (error AAAC-AC)

Força bruta: Python compilat i interpretat

- Hem generat tots els alineaments possibles, per després mirar quina és la millor. No és fiable, ja que tarda massa i gasta molta memòria.

Mètode Needleman-Wunsch: Python compilat i interpretat, Haskell i C

- Aplicant programació dinàmica, hem creat una taula que ens permet resseguir el camí per alinear les seqüències (traceback matrix).
- També hem creat una taula que ens permet puntuar la similitud entre les dues seqüències.
- L'algoritme té 2 versions, una per trobar la puntuació i l'alineament i una altra que només troba la puntuació.

Mètode Needleman-Wunsch Simple: Python compilat i interpretat, Haskell i C

- És el mètode Needleman-Wunsch iteratiu però, enlloc de generar la matriu completa solament genera una fila utilitzant l'anterior. No ens serveix per a trobar l'alineament de les seqüències, però si per a trobar la seva puntuació.

> ***Aquest és el mètode que utilitzarem finalment, ja que és el més ràpid i eficient en memòria, com podem veure a l'anàlisi de costos (Analisi-Algoritmes.ods). Utilitzarem la versió en C, que tarda aproximadament 8 segons per comparació.***

Mètode de classificació

- Per a estalviar temps en les comparacions, creem un fitxer csv on guardem els resultats d'aquestes per a futura referència. El fitxer és saved_distances.csv.

Duració algoritmes de classificació

- Partint d'un fitxer saved_distances.csv buit, k-medoids tarda uns 35 minuts, i Hierarchical classification uns 40.

K-medoids:

- Escollir k-centres, els k primers països.
- Assignar cada país al seu centre més proper.
- Per tots els clusters, mirar quin país té la menor distància total entre tots els del cluster. Canviar el centre per aquest país.
- Com que han canviat els centres, cada país es torna a assignar al cluster més proper.
- Repetir fins que els centres no canvïin.

Hierarchical clustering

- Agrupar els països que estiguin més propers entre ells, creant una unió.
- Aquesta unió fusiona les distàncies dels 2 membres (països o arbres de països), calculant la mitjana de cada distància.
- Es repeteix el procés amb els grups de països enlloc de amb els països, fins que hem agrupat tots els països en un sol grup.