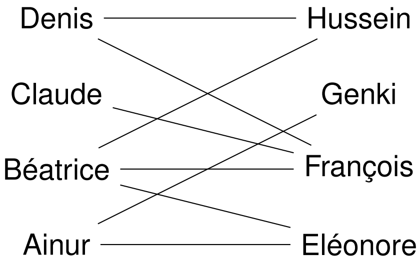
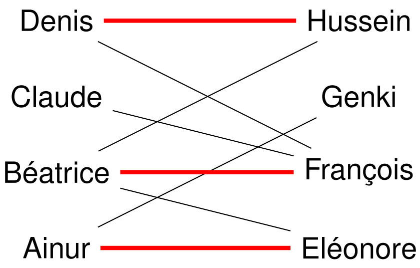
Ressources nécessaires

- Python 3.x (depuis le site officiel)
- un éditeur (notepad++,scribes,gedit,sublimetext)
- IDE (thonny,pycharm,pyzo)
- les données : `ouralou.fr/Resources/epitech.zip`
- papier + crayon
- présentation de graphes (graphviz)

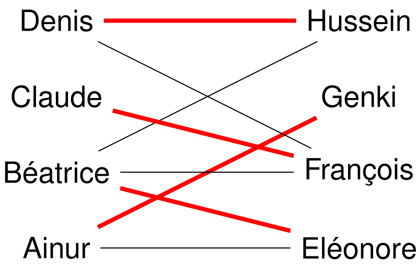
Un graphe de compatibilité



Une allocation sous-optimale

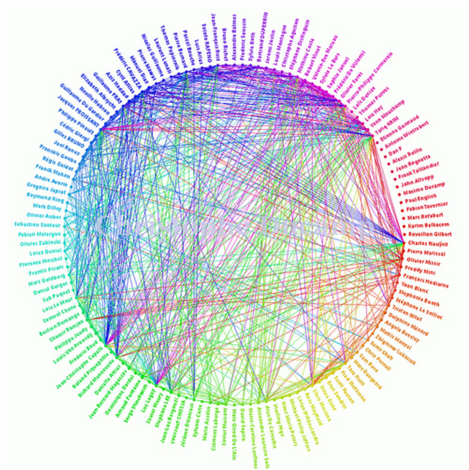


Une allocation optimale



EPITECH.
L’ECOLE DE L’INNOVATION ET DE L’EXPERTISE INFORMATIQUE

Pas toujours facile



Formalisation du problème

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

Formalisation du problème

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

un sous-ensemble d'arêtes $F \subset E$:

Formalisation du problème

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

un sous-ensemble d'arêtes $F \subset E$:

tel que deux arêtes ne soient pas incidentes

Formalisation du problème

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

un sous-ensemble d'arêtes $F \subset E$:

tel que deux arêtes ne soient pas incidentes

de taille maximale

À vous de jouer !

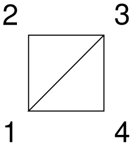
A l’aide du fichier `data_gen.py`, générez un graphe aléatoire à 20 sommets et 50 arêtes. Puis dessinez-le et cherchez graphiquement une allocation aussi grande que possible.

3 Possibilités

Un graphe peut être représenté par exemple comme :

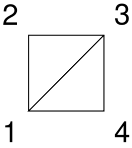
- une liste (voire un set) de sets de taille 2 (les arêtes)
- un dictionnaire de successeurs
- une classe spécifique (si vous connaissez la POO)

Liste de sets




```
g1 = [{1,2},{1,3},{2,3},{3,4},{1,4}]
```

Dictionnaire




```
g1 = { 1:{2,3,4}, 2:{1,3}, 3:{1,2,4}, 4:{1,3} }
```

Classe

```
class Graphe:
    def __init__(self,sommets,aretes):
        self.sommets = frozenset(sommets)
        self.aretes = set(aretes)
    def __contains__(self,e):
        return e in self.aretes
    def _set_aretes(self,aretes):
        self._aretes = set()
        while len(aretes) > 0:
            a = aretes.pop()
            if a.issubset(self.sommets):
                self._aretes.add(a)
    def _get_aretes(self):
        return self._aretes
    aretes = property(_get_aretes,_set_aretes)
```

À vous de jouer !

Sur l’exemple précédemment dessiné, concevez un algorithme glouton pour le problème du couplage maximum.

Algorithme Glouton

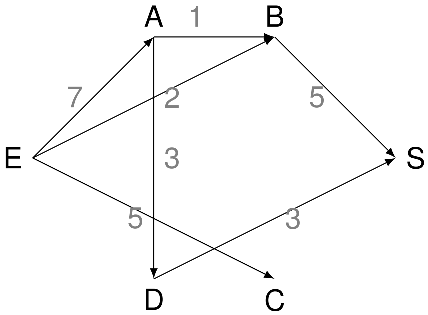
```
def deg(v,reste):
    return len([x for x in sommets if {v,x} in reste])
def glouton(couplage,sommets,aretes):
    sol,reste = [],aretes.copy()
    while(len(reste)>0):
        v = sorted([s for s in sommets
    if deg(s,reste)>0],key=lambda x:deg(x,reste))[0]
        w = [y for y in sommets if {v,y} in reste][0]
        sol.append({v,w})
        reste = [e for e in reste
            if v not in e and w not in e]
    return(sol)
```

Chaîne augmentante

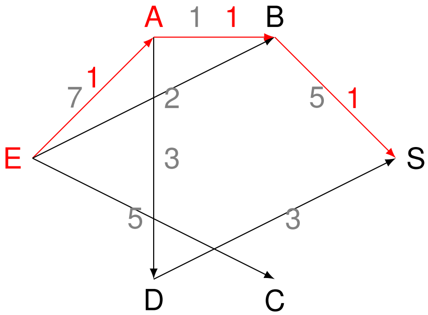
```
def chaine(couplage,sommets,aretes):
    isoles = {x for x in sommets
              if all(x not in e for e in couplage)}
    v = isoles.pop()
    reste,voisin,w,sol = sommets.copy(),None,v,[]
    while reste != set() and voisin not in isoles:
        voisin = {y for y in reste
                  if {y,w} in aretes}.pop()
        sol.append({w,voisin})
        reste.remove(voisin)
        if voisin not in isoles:
            w = {y for y in reste
                  if {y,voisin} in couplage}.pop()
            reste.remove(w)
    return(sol)
```

L'ECOLE DE L'INNOVATION ET DE
L'EXPERTISE INFORMATIQUE

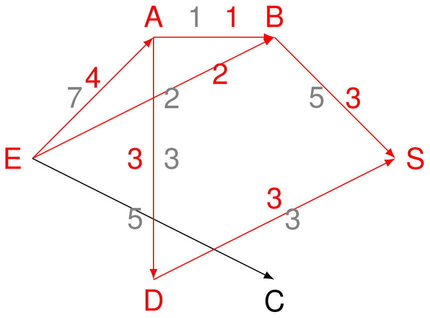
Un réseau avec capacités



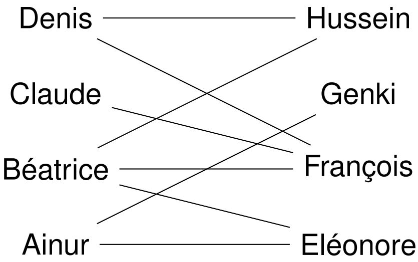
Un flot sous-optimal



Un flot optimal

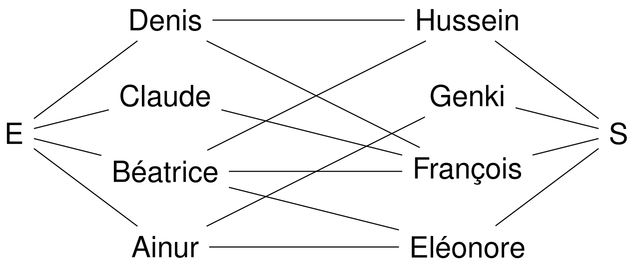


À vous de jouer !



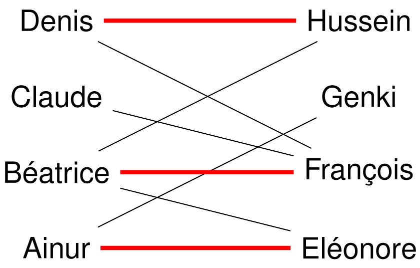
Comment ramener un problème de MATCHING dans un graphe biparti comme celui-ci à un problème de FLOT MAX ?

Solution

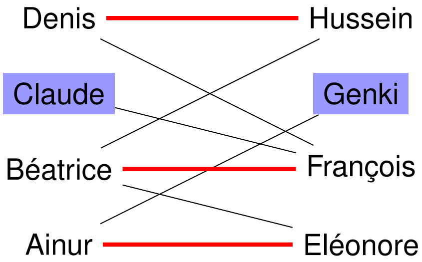


Toutes les arêtes ont une capacité 1.

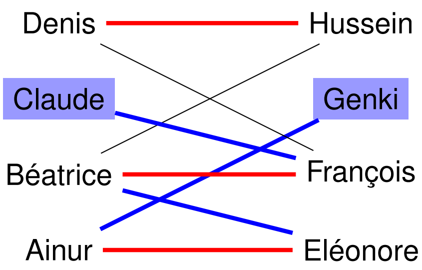
Chaîne augmentante



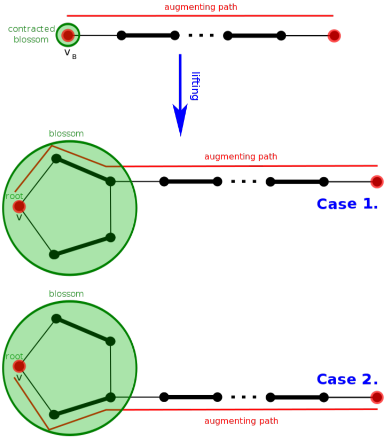
Deux sommets isolés



Une chaîne alternée



Le cas non-biparti



À vous de jouer !

1) Reprenez l'exemple graphique que vous avez construit précédemment et générez rapidement une solution quelconque (non optimale). Puis cherchez une sous-chaîne améliorante.

2) Récupérez le code complet de Ford-Fulkerson/Edmonds (par ex. sur wikipedia) et testez le sur un exemple simple.

Complexité

Les algorithmes de Ford-Fulkerson et Edmonds ont une complexité polynomiale (resp. $n \times m$ et $n^2 \times m$).

Ilpeuvent donc tourner sur des instances incomparablement plus grandes qu'un algorithme exponentiel.

Complexité

On va pouvoir travailler sur des graphes contenant des milliers de sommets...

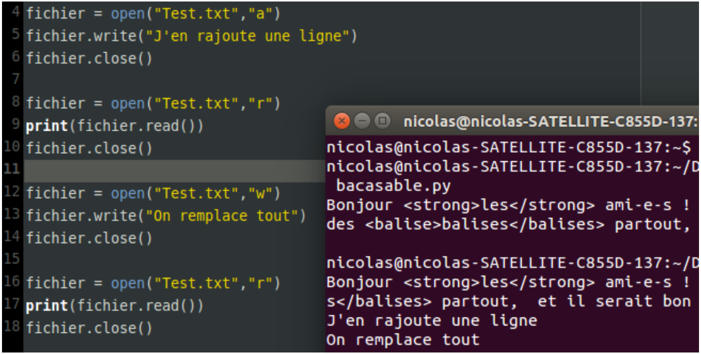
Naturellement, plus question de rentrer ces données manuellement.

Lire un fichier

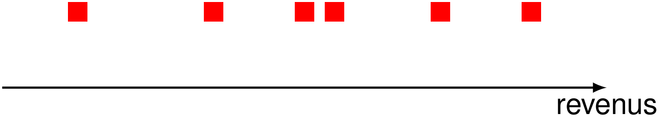
```python
#!/usr/bin/python3.4
# -*-coding:utf-8 -*

fichier = open("Test.txt","r")
contenu = fichier.read()
print(contenu)
fichier.close()
```

```
nicolas@nicolas-SATELLITE-C855D-137: ~/Documents/Enseignements/Programmes
nicolas@nicolas-SATELLITE-C855D-137:~$ cd Documents/Enseignements/Programmes/
nicolas@nicolas-SATELLITE-C855D-137:~/Documents/Enseignements/Programmes$ python
 bacasable.py
Bonjour <strong>les</strong> ami-e-s ! Ceci est une longue chaîne de texte avec
des <balise>balises</balises> partout,  et il serait bon de le couper un peu.
```

Écrire dans un fichier

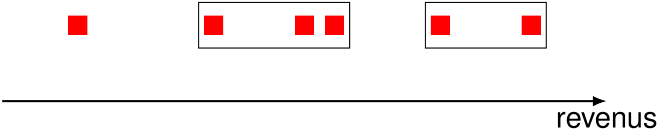


Données monovariées



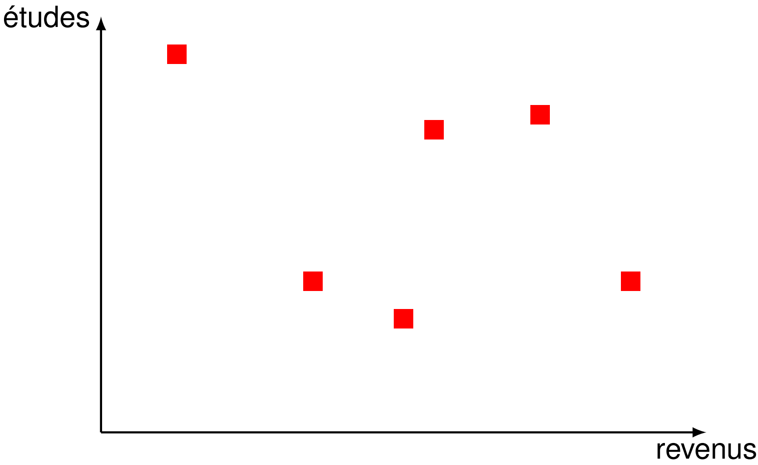
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Données monovariées

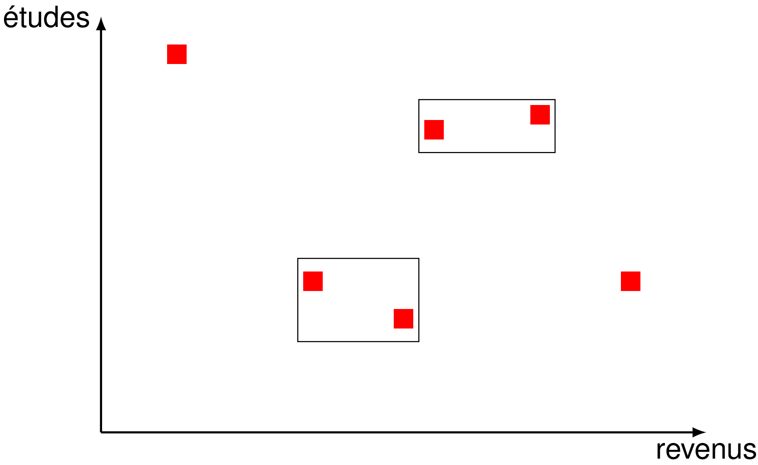


L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Données bivariées



Données bivariées

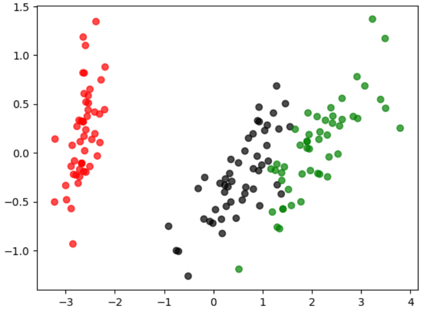


Données multivariées

Comment représenter sur un écran un classement selon des dizaines ou des milliers de critères ?

Comment déterminer des compatibilités entre des individus représentés par autant de variables ?

Réduction dimensionnelle



Réduction dimensionnelle

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA

iris = datasets.load_iris()
X,Y = iris.data, iris.target
colMap={0:"red",1:"green",2:"black"}
colors=list(map(lambda x:colMap.get(x),Y))
X_2ev = PCA(n_components=2).fit_transform(X)
plt.scatter(X_2ev[:,0],X_2ev[:,1],alpha=0.7,c=colors)

plt.show()
```

À vous de jouer !

Importez le fichier `data2.csv` et essayez de construire une représentation ou de modéliser un graphe de compatibilité.

Données numériques

Normalisation (exemple) :

$$x' = \frac{x - xmin}{xmax - xmin}$$

Agrégation (exemple) :

$$d(X, Y) = \sqrt{\sum (x_i - y_i)^2}$$

Données par modalités

Distance binaire :

$$d(X,Y) = \sharp\{x_i \neq y_i\} = \sum_{x_i \neq y_i} 1$$

Distance pondérée :

$$d(X,Y) = \sum_{x_i \neq y_i} \omega_i$$

Exemple

X : BLOND, BAC+5, MODEM, 43 ans
Y : BLOND, BAC+2, NPA, 36 ans

Exemple

L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

X : BLOND, BAC+5, MODEM, 43 ans
Y : BLOND, BAC+2, NPA, 36 ans

X' : BLOND, 0.6, MODEM, 0.7
Y' : BLOND, 0.3, NPA, 0.55

Exemple

X : BLOND, BAC+5, MODEM, 43 ans
Y : BLOND, BAC+2, NPA, 36 ans

X' : BLOND, 0.6, MODEM, 0.7
Y' : BLOND, 0.3, NPA, 0.55

$$d(X, Y) = \sqrt{0 + (0.6 - 0.3)^2 + 1 + (0.7 - 0.55)^2}$$

À vous de jouer !

Construisez une matrice de distances sur les données du fichier `data2.csv`.

Principe

On fixe un seuil, par exemple $S = N/4$, où N est le nombre de variables.

Principe

On fixe un seuil, par exemple $S = N/4$, où N est le nombre de variables.
On considère que deux sommets doivent être reliés si et seulement si leur distance est inférieure au seuil.
$(X, Y) \in G \iff d(X, Y) < S$

Exemple

X : BLOND, 0.6, MODEM, 0.7
Y : BLOND, 0.3, NPA, 0.55
Z : BRUN, 0.5, LR, 0.8
T : BRUN, 0.2, NPA, 0.2

Exemple

	X	Y	Z	T
X		1.45	2.30	2.90
Y			2.45	1.45
Z				1.90
T				

Exemple

	X	Y	Z	T
X		1.45	2.3	2.9
Y			2.45	1.45
Z				1.9
T				

À vous de jouer !

1) Fixez un seuil et utilisez la matrice de l'exercice précédent pour construire des proximités entre les individus.

2) Essayez de produire le graphe correspondant.

Rappel : le MATCHING

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

Rappel : le MATCHING

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

un sous-ensemble d'arêtes $F \subset E$:

Rappel : le MATCHING

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

un sous-ensemble d'arêtes $F \subset E$:

tel que deux arêtes ne soient pas incidentes

Rappel : le MATCHING

L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

un sous-ensemble d'arêtes $F \subset E$:

tel que deux arêtes ne soient pas incidentes

de taille maximale

Rappel : le CLUSTERING

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

Rappel : le CLUSTERING

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

Une division de V en sous-ensembles disjoints V_1, V_2, V_3...

Rappel : le CLUSTERING

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

Une division de V en sous-ensembles disjoints V_1, V_2, V_3...

avec un maximum d'arêtes à l'intérieur de chaque V_i

Rappel : le CLUSTERING

Soit un graphe G défini par un ensemble de sommets V et un ensemble d'arêtes E. On cherche

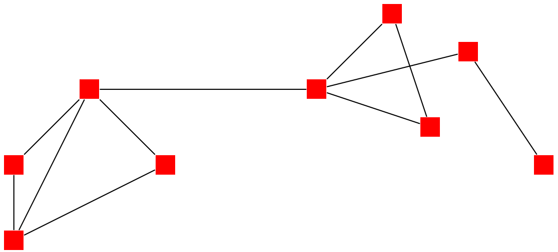
Une division de V en sous-ensembles disjoints V_1, V_2, V_3...

avec un maximum d'arêtes à l'intérieur de chaque V_i

et un minimum à l'extérieur, entre les différents V_i.

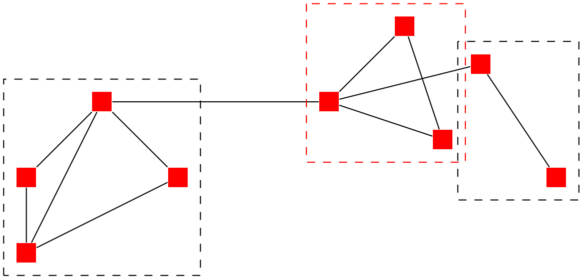
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Exemple



L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Exemple



Différents types d'objectifs

- Ne regrouper que des éléments tous deux à deux compatibles :

$$x \in V_i, y \in V_i \implies (x, y) \in G$$

Différents types d'objectifs

- Ne regrouper que des éléments tous deux à deux compatibles :

$$x \in V_i, y \in V_i \implies (x, y) \in G$$

- Ratio inter/intra minimal :

$$\min \frac{\sharp\{(x, y) \in G, x \in V_i, y \in V_j\}}{\sharp\{(x, y) \in G, x, y \in V_i\}}$$

À vous de jouer !

Trouvez un clustering pertinent sur l'exemple des exercices précédents.

Principe

On va procéder de façon itérative.

Principe

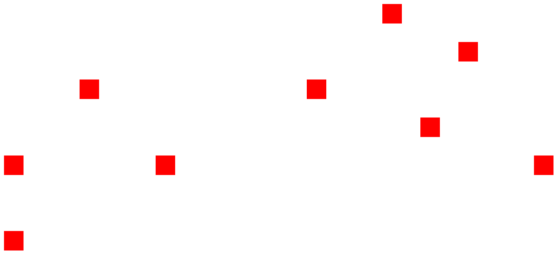
On va procéder de façon itérative.
A chaque étape on regroupe les deux éléments les plus proches.

L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

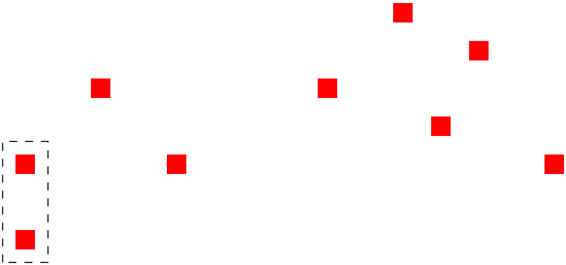
Principe

On va procéder de façon itérative.
A chaque étape on regroupe les deux éléments les plus proches.
Le groupement ainsi constitué est considéré comme un pseudo-élément positionné en son barycentre.

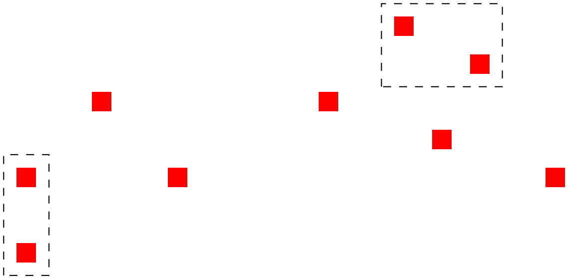
Exemple



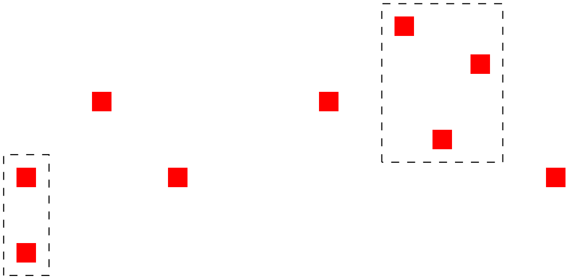
Exemple



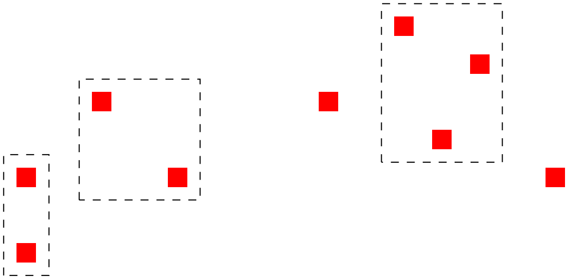
Exemple



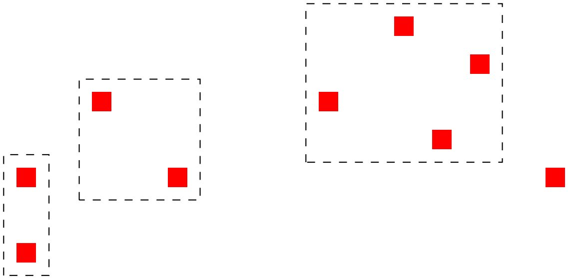
Exemple



Exemple



Exemple



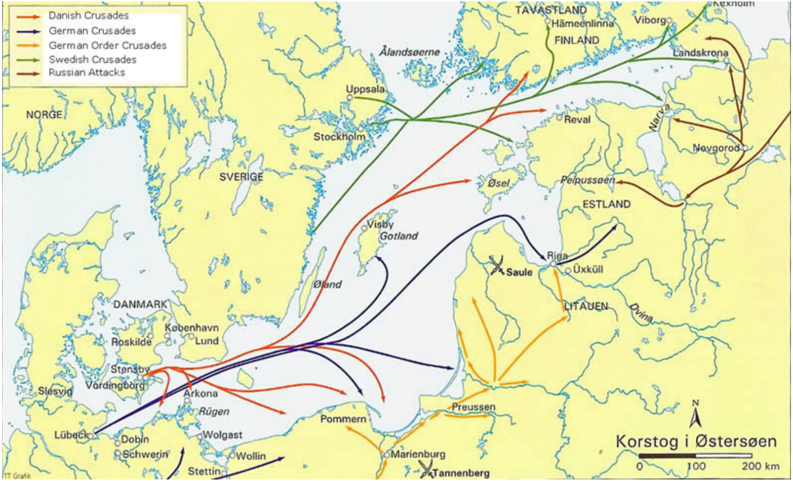
À vous de jouer !

Programmez un algorithme de classification hiérarchique ascendante. Testez-le sur l'exemple précédent (à partir de la table de distances).

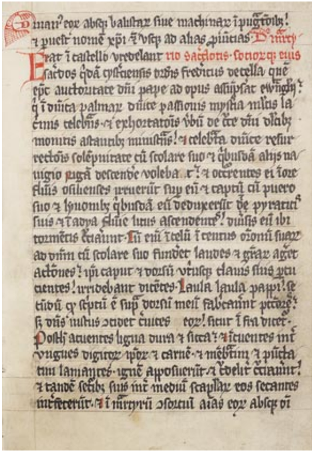
Qu’est-ce que c’est ?

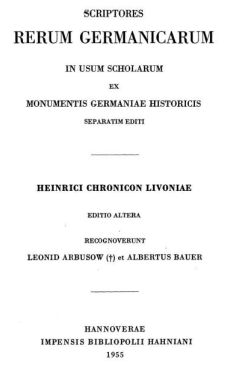
Quelle est son origine ?

La Livonie, XIIe-XIIIe siècles



Le texte d'Henri





SCRIPTORES

RERUM GERMANICARUM

IN USUM SCHOLARUM

EX

MONUMENTIS GERMANIAE HISTORICIS

SEPARATIM EDITI

HEINRICI CHRONICON LIVONIAE

EDITIO ALTERA

RECOGNOVERUNT

LEONID ARBUSOW (†) et ALBERTUS BAUER

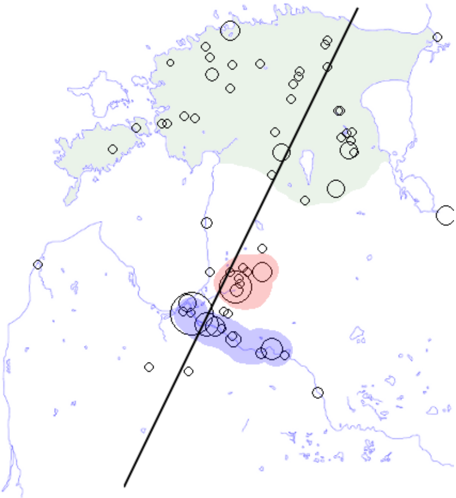
HANNOVERAE
IMPENSIS BIBLIOPOLII HAHNIANI
1955

Saisie

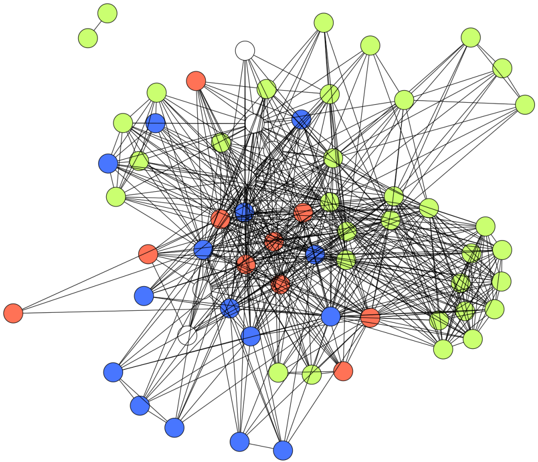
Dünamünde	Daugavgrīva	N 57° 3’	E 24° 2’	1186
Dünamünde	Daugavgrīva	N 57° 3’	E 24° 2’	1186
Dünamünde	Daugavgrīva	N 57° 3’	E 24° 2’	1191
Dorpat	Tartu	N 58° 23’	E 26° 43’	1214
Dorpat	Tartu	N 58° 23’	E 26° 43’	1217

L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

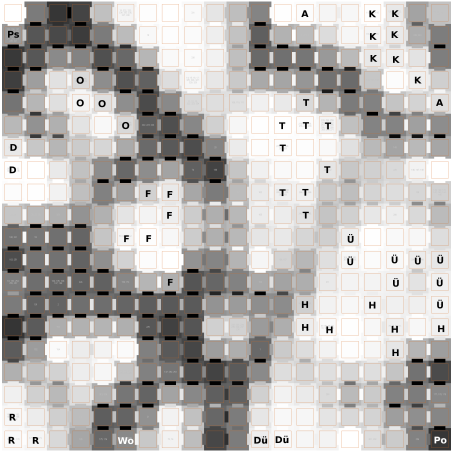
la Livonie d'Henri



Sous forme de graphe

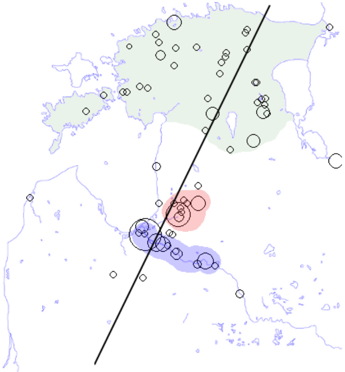


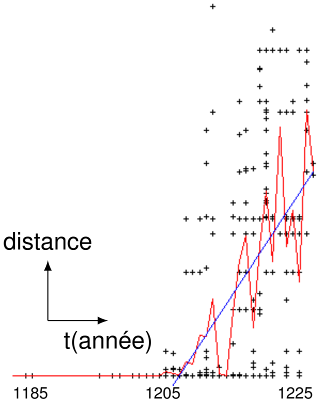
Carte de Kohonen



Distance à la rivière

L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

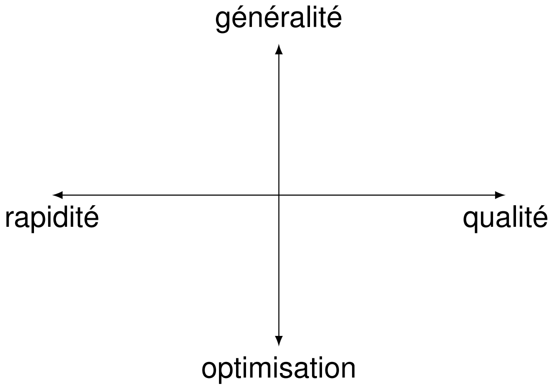




L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Notions contradictoires

Pour un problème donné, il faut souvent équilibrer plusieurs notions.



Arbitrages

- Les solutions exhaustives fonctionnent toujours, mais leur coût est prohibitif.

Arbitrages

- Les solutions exhaustives fonctionnent toujours, mais leur coût est prohibitif.
- Les algorithmes polynomiaux exacts comme FF n'existent que sur un nombre limité de problèmes.

Arbitrages

- Les solutions exhaustives fonctionnent toujours, mais leur coût est prohibitif.
- Les algorithmes polynomiaux exacts comme FF n'existent que sur un nombre limité de problèmes.
- Dans les autres cas (clustering par exemple), on se rabat sur des heuristiques et on essaie d'équilibrer rapidité et qualité.

Arbitrages

- Les solutions exhaustives fonctionnent toujours, mais leur coût est prohibitif.
- Les algorithmes polynomiaux exacts comme FF n'existent que sur un nombre limité de problèmes.
- Dans les autres cas (clustering par exemple), on se rabat sur des heuristiques et on essaie d'équilibrer rapidité et qualité.
- Et évaluer cette dernière est souvent en soi un problème difficile...

À vous de jouer !

Quelle est la complexité respective des algorithmes vus tout au long de ce cours ?