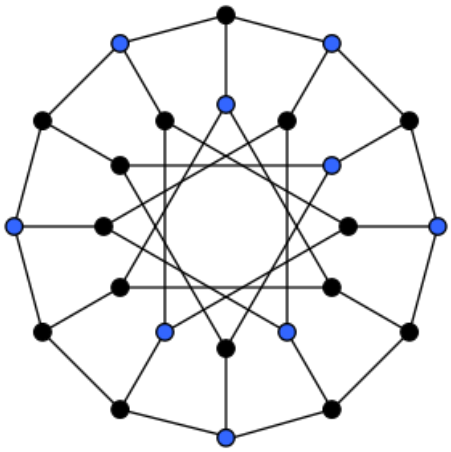
Faculté des Sciences
Aix*Marseille Université

Rapport de stage

Appariement multiple de graphes

Juliette Philibert



Tuteur : François-Xavier Dupé

2019

Sommaire

Notation

Dans ce rapport, on notera :

— $X = \{x_1, ..., x_m\}$ et $Y = \{y_1, ..., y_m\}$ 2 sets de données de taille m

— On notera K la matrice de Gram de X et L celle de Y.

— $\mathbb{1}_m = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ le vecteur colonne de 1 de taille m

— $\mathbb{1}_{m \times m} = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$ la matrice de 1 de taille $m \times m$

— $I_m = \begin{pmatrix} 0.27 & 0.43 & 0.28 \\ 0.33 & 0.51 & 0.17 \\ 0.44 & 0.1 & 0.49 \end{pmatrix}$ la matrice identité

— $||.||_F$ la norme de Frobenius

— $||.||_2$ la norme euclidienne

— $< A, B >$ le produit scalaire de A et B

— $tr(A)$ la trace de A

— $\overline{K} = H\overline{K}H$ où $H = (I - 1)/n$ la matrice centrée de K.

— A^T = la matrice transposée de A.

— $P_m = \{\pi | \pi \in \{0, 1\}^{m \times m}$ et $\pi 1_m = 1_m$ et $\pi^T 1_m = 1_m\}$.

— l'ensemble des permutations de taille m.

— $k : X \times X \to \mathbb{R}$ une fonction noyau.

— $\odot$ le produit d'Hadamard

1 Sujet de stage

Mon stage est le fruit d'une collaboration entre l'équipe Qarma et l'INT. Nous cherchons à appliquer une méthode pour trouver une solution à un problème de neurosciences ; à partir de résultats d'examen du cerveau de patients sains (et d'autres présentant des pathologies particulières), on cherche à mettre en évidence des correspondances entre les sulcal pits des patients.

Mais que sont les sulcal pits ?
Le cerveau se divise en plusieurs lobes cérébraux qui rassemblent un ensemble de plis délimités par des sillons. Les sulcal pits sont les points les plus profonds situés dans ces sillons. L'intérêt ici, est de trouver des sulcal pits "communs" à tous les individus et de trouver ou pas des anomalies liées à des pathologies particulières.

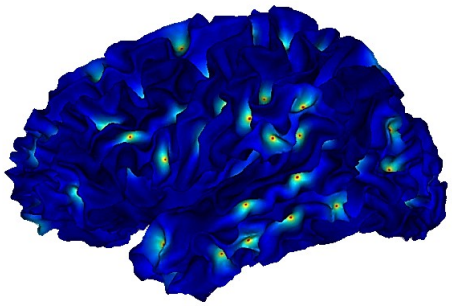


FIGURE 1 – Les sulcal pits du cerveau(représenté par les points rouges)

Il faut maintenant trouver une structure adaptée pour représenter un cerveau et ses sulcal pits. On utilise les graphes : on construit un graphe pour chaque personne où chaque sommet correspond à un sulcal pit (avec ses caractéristiques en étiquette : profondeur, coordonnées, aire, épaisseur, sph_coord) et où 2 sommets sont reliés par une arête s'ils sont à côté.

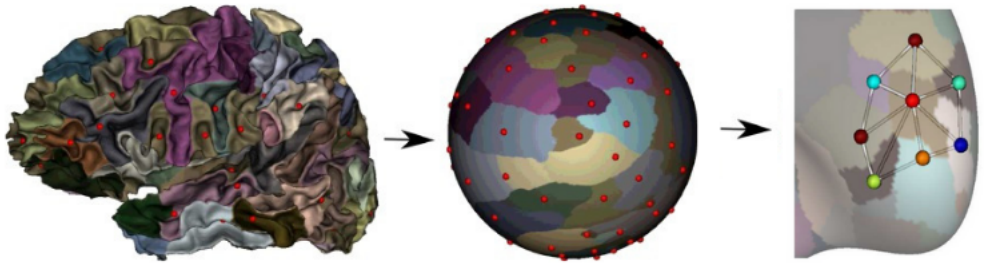


FIGURE 2 – Passage du cerveau au graphe

L'appariement de sulcal pits devient alors un problème d'appariement de graphes. La difficulté vient du fait qu'on a une multitude de graphes à appareiller et qu'ils peuvent être très différents vue la complexité du cerveau humain.

2 Travail réalisé

2.1 Compréhension du sujet

Mon stage a commencé par une semaine de lecture de différents articles scientifiques sur des algorithmes de comparaison d'objets. Le but était d'étudier différentes méthodes pour faire correspondre les objets ; on retiendra pour la suite *les méthodes à noyaux* .

Pourquoi les méthodes à noyaux ?
Lorsqu'un problème est trop compliqué à résoudre dans son espace, on le projette dans un autre espace où le problème devient linéaire et où on peut appliquer des méthodes de résolution connues.

De plus, il existe plusieurs fonctions noyaux pour les graphes qui permettent de mesurer la ressemblance entre les sommets de graphe, entre les structures, ou entre les chemins. On peut intégrer au noyau les particularité des noeuds (des sulcal pits dans notre cas) pour des résultats pertinents.

En un sens, le noyau correspond à une mesure de similarité entre une entrée x et y, donc pour nos graphes à des mesures de similarité de chemins , de structures ou de sommets selon le noyau. Plus précisément, si x et y sont des vecteurs un noyau a une valeur élevé si x est équivalent à y et faible sinon. Voici la définition mathématique d'une fonction noyau avec $x, y \in X$:

$$k : X \times X \to \mathbb{R} \text{ où } k(x, y) \geq 0 \text{ et } k(x, y) = k(y, x)$$

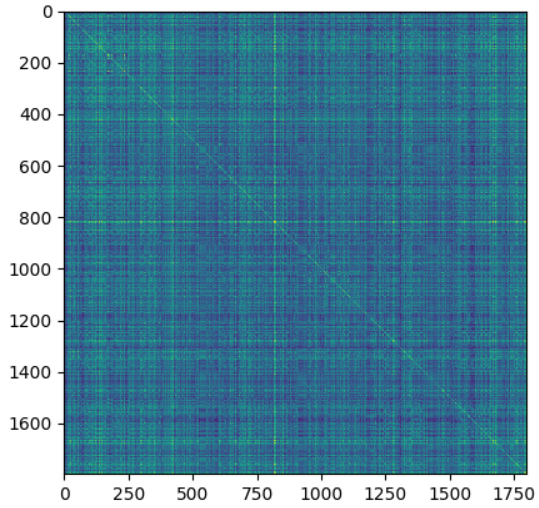


FIGURE 3 – Représentation du noyau linéaire sur des données de 1797 images

Par exemple, la définition du noyau linéaire est : $k(x, y) = x^T y$ où x et y sont des vecteurs colonnes. Dans le cadre de mon problème, j'utilise plusieurs fonctions noyau. Leurs formules sont données en annexes [3].

2.2 Implémentation du "Convex Kernelized Sorting"

2.2.1 Explication de l'algorithme

Le "Convex Kernelized Sorting" s'appuie sur le Kernelized Sorting et rend le problème convexe. On va s'intéresser à coder cet algorithme d'abord pour 2 objets puis on l'étendra plus tard à une multitude d'objets.

2.2.2 Kernelized Sorting

Le Kernelized Sorting [4] permet l'appariement de 2 objets (ou plus) de types différent n'ayant pas forcément la même structure. Cette méthode s'appuie sur le critère d'indépendance d'Hilbert Schmidt (HSIC) qui permet de mesurer la dépendance entre 2 sets de données X et Y de taille égale à m qui seront représentés par leurs matrices noyaux K et L. On cherche alors une permutation π qui permet de passer de K à L.

Soient 2 ensembles d'objets $X = \{x_1, ..., x_m\}$ et $Y = \{y_1, ..., y_m\}$ de taille m, k la fonction noyau appliqué à X et l celle appliquée à Y. On cherche à trouver une correspondances entre les objets de X et ceux de Y. Soient $KetL$ leurs matrices noyau aussi appelé matrice de Gram, où $K_{ij} = k(x_i, x_j)$ (resp. $L_{ij} = l(x_i, x_j)$) mesure la similarité en quelque sorte entre x_i et x_j.

On cherche en fait une permutation $\pi \in P_m$
où $P_m = \{\pi | \pi \in \{0, 1\}^{m \times m}$ and $\pi 1_m = 1_m$ and $\pi^T 1_m = 1_m\}$
et π_{ij}=1 si x_j correspond à y_i. Pour trouver cette permutation, le Kernelized Sorting utilise le critère d'Hilbert Schmidt (HSIC) qui mesure les dépendances de deux ensembles d'objets. L'algorithme veut maximiser les dépendances donc ce critère. Ce critère est la norme d'Hilbert Schmidt qui peut être estimé par :

$$\triangle^2 = m^{-2}.tr(\overline{K}.\overline{L})$$

On note $\overline{K}$ et $\overline{L}$ les matrices centrées de K et L avec $\overline{K} = HKH$ où $H = (I - 1)/m$.

Pour trouver les meilleurs correspondances, on cherche une permutation π_{opt} qui maximise cette norme et donc les dépendances entre nos matrices noyaux, on obtient le problème d'optimisation suivant :

$$\pi_{opt} = \arg\max_{\pi \in \pi_m} tr(\overline{K}\pi^T\overline{L}\pi)$$

Une version de calcul du HSIC sur plusieurs matrices existe et peut donc nous intéresser vu notre problème(voir 2.2.4).

2.2.3 Convexification

Le Convex Kernelized Sorting [1] se base sur le Kernelized Sorting mais le transforme en un problème d'optimisation convexe qui permet de trouver une solution (pas forcément optimale).

En remarquant que la valeur de $tr(\overline{K}.\overline{L})$ est maximisée si les lignes et les colonnes de $\overline{K}$ et $\overline{L}$ sont permutées de tel sorte que les lignes de $\overline{K}$ correspondent aux colonnes de $\overline{L}$ (à une constante

multiplicative près). Le problème revient alors à,

$$\underset{\pi \in \pi_m}{\text{minimize}} \|\overline{K}\pi^T - (\overline{L}\pi)^T\|_F^2 \quad \text{s.t.} \quad \begin{cases} \pi \in \{0,1\} \\ \pi 1_m = 1_m \\ \pi^T 1_m = 1_m \end{cases} \tag{1}$$

— $\overline{K}\pi^T$ échange les colonnes de $\overline{K}$.
— $(\overline{L}\pi)^T$ échange les lignes de $\overline{L}$.
— $\pi 1_m = 1_m$ est la contrainte que la somme des lignes est égale à 1.
— $\pi^T 1_m = 1_m$ est la contrainte que la somme des colonnes est égale à 1.

On simplifie le problème en remplaçant la contrainte $\pi \in \{0,1\}$ par $\pi \in [0,1]$. On obtient non plus une permutation mais une matrice avec des coefficients entre 0 et 1 (matrice stochastique). On peut alors interpréter π_{ij} comme une probabilité de correspondance entre x_j et y_i. On va aussi intégrer les contraintes que les sommes des valeurs des lignes (des colonnes aussi) doit être égal à 1. Le problème devient alors :

$$\underset{\pi \in \pi_m}{\text{minimize}} \|\overline{K}\pi^T - (\overline{L}\pi)^T\|_F^2 + c(\|\pi 1 - 1\|_2^2 + \|\pi^T 1 - 1\|_2^2) \quad \text{s.t.} \quad \pi \in [0,1] \tag{2}$$

— $||\pi 1 - 1||_2^2$ est la contrainte que la somme des lignes est égale à 1.
— $||\pi^T 1 - 1||_2^2)$ est la contrainte que la somme des colonnes est égale à 1.
— c est une constante à fixer selon le poids de la contrainte voulue.

Pour minimiser cette fonction, on utilise la descente du gradient projeté.

2.2.4 Multi HSIC

Une version pour comparer plusieurs ensembles du HSIC existe. Si on veut comparer n+1 ensembles $X_0, X_1 ... X_n$ de taille m où leurs matrices de gram respectives sont : $K_0, K_1 ... K_n$. On cherche alors $\pi_0, \pi_1 ... \pi_n \in P_m$ où $\pi_i[l,c] = 1$ si $X_0[c]$ correspond à $X_i[l]$. On a donc toujours $\pi_0 = I$ (on compare les objets de X_0 avec eux mêmes).

Le calcul du HSIC revient ici à :

$$HSIC(X_0 ... X_n) = \mathbb{1}_m^T (\odot_{i=0}^n \pi_i^T K_i \pi_i) \mathbb{1}_m + \prod_{i=0}^n \mathbb{1}_m \pi_i^T K_i \pi_i \mathbb{1}_m^T - 2.\mathbb{1}_m^T (\odot_{i=0}^n \pi_i^T K_i \pi_i \mathbb{1}_m) \tag{3}$$

Pour maximiser cette fonction, on utilise la descente du gradient projetée sur son inverse.

2.2.5 Implémentation

Plusieurs méthodes de résolutions de problème d'optimisation convexe existe, la méthode choisi est ici la descente du gradient projeté.

La descente du gradient vise à minimiser(ou maximiser) une fonction différentiable. On choisit cette méthode si la résolution par $f'(x) = 0$ n'est pas possible. C'est une approche itérative qui permet une approximation d'une solution locale.

Pour utiliser la descente du gradient, on doit calculer le gradient de notre fonction objectif qui est (pour 2 ensembles) :

$$f_{obj}(\pi) = \|K\pi^T - (L\pi)^T\|_F^2 + c(\|\pi \mathbb{1}_m - \mathbb{1}_m\|_2^2 + \|\pi^T \mathbb{1}_m - \mathbb{1}_m\|_2^2) \tag{4}$$

On obtient pour le gradient (voir annexe pour le calcul) :

$$\frac{\partial f_{obj}}{\partial \pi} = 2(\pi K^T - \pi^T L^T)K - 2L^T(K\pi^T - L\pi) + c(2(\pi + \pi^T - 2I_n)\mathbb{1}_{m\times m}) \tag{5}$$

Algorithm 1 Descente du gradient projeté

Entrée : $f_{obj}, \mu \in \mathbb{R}, \pi_0 \in, \mathcal{P}, it \in \mathbb{N}$
Sortie : π_{opt}
  Pour $i = 0$ to it **Faire**
    $\pi_{i+1} = \mathcal{P}(\pi_i - \mu \bigtriangledown f(\pi_i))$
  fin Pour
  Retourner π_{it}

J'utilise une descente du gradient qui à chaque pas fixe une nouvelle valeur pour μ pour favoriser une meilleure descente. Je dois donner en paramètre de ma fonction : $\boldsymbol{f_{obj}}(4)$ la fonction d'objectif, $\boldsymbol{f_{grad}}$ (5) son gradient calculé au préalable, $\boldsymbol{\mathcal{P}}$ le projecteur, **c** la constante à fixer qui modélise la prise en compte des contraintes, $\boldsymbol{\mu}$ le pas d'initialisation, $\boldsymbol{\mu_{min}}$ la valeur minimum du pas, $\boldsymbol{\pi_0}$ la matrice de permutation d'initialisation, **it** le nombre d'itération à faire, et bien sur mes 2 matrices noyaux centrées **K**et **L** à comparer.

Une fois la matrice "optimale" trouvée je la transforme en une matrice de permutation en cherchant le maximum, puis je le remplace par un 1, je met la ligne et la colonnes du maximum à 0 et je recommence jusqu'à avoir une permutation (n fois pour une matrice n×n) ou grâce à l'algorithme hongrois.

Pour la version multiple, notre fonction objectif est :

$$f_{objmulti}(\pi_0, ...\pi_n) = -(\mathbb{1}_m^T(\odot_{i=0}^n \pi_i^T K_i \pi_i)\mathbb{1}_m + \prod_{i=0}^{n} \mathbb{1}_m \pi_i^T K_i \pi_i \mathbb{1}_m^T - 2.\mathbb{1}_m^T(\odot_{i=0}^n \pi_i^T K_i \pi_i \mathbb{1}_m)) \tag{6}$$

Pour des soucis de calcul du gradient, on remplace dans la fonction objectif $\pi_i^T K_i \pi_i$ par $\pi_i K_i \pi_i^T$
.

Le gradient associé est :

$$\frac{\partial f_{objmulti}}{\partial \pi_i} = -(2A\pi_i K_i + 2b(\mathbb{1}_{m\times m}\pi_i K_i) - 2(c\mathbb{1}_m^T + \mathbb{1}_m c)\pi_i K_i \tag{7}$$

où $A = \odot_{j=0,j!=i}^n \pi_j^T K_j \pi_j$
$b = \prod_{j=0,j!=i}^n \mathbb{1}_m^T \pi_j K_j \pi_j^T \mathbb{1}_m$
$c = \odot_{j=0,j!=i}^n \pi_j^T K_j \pi_j \mathbb{1}_m$

Pour trouver les n+1 matrices de permutations $\pi_0 ... \pi_n$, on réalise la descente du gradient projeté n fois, une par rapport à chaque π_i. On peut même faire plusieurs tours pour des meilleurs résultats.

2.3 Autre méthode : branch and bound

Un algorithme branch and bound est une méthode générique de résolution de problèmes d'optimisation combinatoire, c'est à dire trouver le minimum d'une fonction d'un ensemble discret. Grâce à cette algorithme, on trouve la solution optimale (pas comme avec les algorithmes précédents).

Dans un ensemble discret énumérer toute les possibilités peut s'avérer difficile voir impossible. Comme par exemple, dans notre cas il y a $10! = 3628800$ matrices de permutations de taille 10 (ce qui est déjà énorme), mais pour les matrices de notre problème qui peuvent être de taille 100, il y en a plus de 10e157 ce qui est très problématique pour la résolution.

L'algorithme du branch and bound est basé sur l'idée d'utiliser les bornes supérieurs pour choisir quelles solutions éliminer en étant sûr qu'elle ne sont pas meilleur que la solution courante. On énumère intelligemment les solutions en éliminant celles qu'on sait ne pas être intéressante, et cela nous permet de parcourir seulement une petite portion des possibilités pour trouver la solution à notre problème.

On représente l'exécution de l'algorithme par un arbre. Dans notre cas, on parcourt l'ensemble des permutations et chaque noeud de notre arbre sera un noeud de contrainte. On appelle contrainte une liste de tuple (i,j) qui fixe dans la permutation p[i,j] = 1 et le reste de la ligne i à 0 (respectivement le reste de la colonne j à 0). Les fils de la racine représentera par exemple toute les possibilités de la position du 1 sur la première ligne.

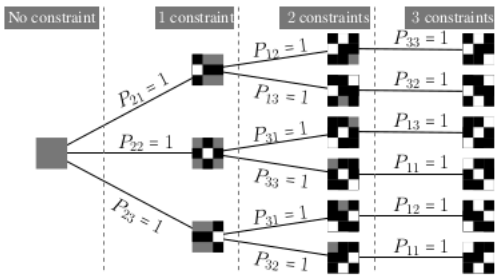


Fig. 1. Tree structure to explore $\mathcal{P}_3$ by adding constraints (white ones) on the permutation matrix. Some elements are consequently zeros (black), others are free (gray).

FIGURE 4 – Figure issu de [2]

On commence par considérer le problème de départ avec son ensemble de solutions. On calcule les bornes inférieure et supérieure de la racine, si elles sont égal c'est la solution optimale. Sinon on divise le problème en sous problème, on fixe le 1 sur la première ligne au différents endroits où il peut être (pour une matrice de taille n, ça nous fait n sous problèmes), on calcule les bornes inférieures/supérieures. On répète l'opération jusqu'à ce que la borne inférieure soit égal à la borne supérieure. L'opération étant de choisir une contrainte active, ajouter des nouvelles contraintes (si elle sont possibles), calculer leurs bornes inférieures / supérieures et éliminer celle où leurs bornes inférieures est supérieure à la plus petite borne supérieure. La recherche continue jusqu'à ce que tous les noeuds soit explorés ou éliminés.

Le papier [2] nous explique comment trouver facilement nos bornes inférieurs, supérieurs et comment choisir une bonne heuristique pour des problèmes impliquant l'ensemble des matrices de permutation comme notre problème.

Pour le calcul de la borne inférieur appelée l, on prend une relaxation du problème de base et on prend la valeur de la fonction à minimiser avec la matrice stochastique trouvée.

$$l = \min \|\overline{K}\pi^T - (\overline{L}\pi)^T\|_F^2 + c(\|\pi 1 - 1\|_2^2 + \|\pi^T 1 - 1\|_2^2) \quad \text{s.t.} \quad \pi \in [0,1] \tag{8}$$

Pour le calcul de la borne supérieure appelé u, on récupère la matrice stochastique π trouvée pour la borne inférieur, et on la transforme en une matrice de permutation. La borne supérieur est alors le résultat de la fonction objectif avec cette matrice de permutation.

Pour le choix de la prochaine contrainte à explorer, on prend celle qui a la plus petite borne inférieur.

Voici l'algorithme implémenté dans le fichier branch_and_bound.py pour la correspondance entre deux ensembles d'objets grâce à la recherche de la correspondances entre leurs matrices de gram.(8)

Algorithm 3 Proposed branch-and-bound algorithm

Require: $\mathbf{Y}$, $\mathbf{D}$.

Initialize list for active constraint sets $\mathcal{A} \leftarrow \{\{\varnothing\}\}$
repeat
  Select $\mathcal{C} \in \mathcal{A}$ with lowest lower bound, remove it from $\mathcal{A}$.
  Choose splitting point (i_0, j_0) as the location of the greatest element in the bistochastic matrix obtained from $l_{\mathbf{Y},\mathbf{D}}(\mathcal{C})$.
  Add $\mathcal{C} \cup \{(i_0, j)\}$ to $\mathcal{A}$ for each j such that $\mathcal{P}_{\mathbf{Y},\mathbf{D}}(\mathcal{C} \cup \{(i_0, j)\})$ is feasible.
  Compute the lower bound $l_{\mathbf{Y},\mathbf{D}}(\mathcal{C} \cup \{(i_0, j)\})$ and the upper bound $u_{\mathbf{Y},\mathbf{D}}(\mathcal{C} \cup \{(i_0, j)\})$ of each added constraint set.
  Update L (resp. U) as the lowest lower bound (resp. lowest upper bound) among all the elements of $\mathcal{A}$.
  Prune every $\mathcal{S} \in \mathcal{A}$ such that $l_{\mathcal{P}}(\mathcal{S}) > U$.
until $L = U$
return permutation matrix $\widehat{\mathbf{P}}$ and sparse matrix $\widehat{\mathbf{X}}$ related to the only element induced by the unique constraint set in $\mathcal{A} = \{\mathcal{C}\}$.

FIGURE 5 – Algorithme du Branch and bound issu de [2]

2.4 Résultats

2.4.1 Comparaison d'images

J'ai fait toutes sortes de tests. Une première série avec des petites matrices 3x3 ou 5x5 ou 10x10. C'était difficile dans ce cas de trouver des résultats qui aient du sens car derrière ces matrices il n'y avait pas d'objet.J'ai ensuite fait des tests avec des matrices symétriques.

J'ai maintenant un jeu de données grâce à sklear.datasets.load_digits(), ce jeu de donnée comporte 1797 images 8x8 qui représentent des nombres entre 0 et 9. Voir figure ci-dessous.

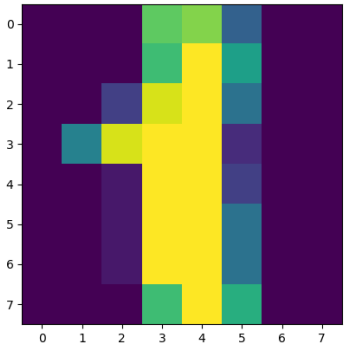


FIGURE 6 – Image 8x8 représentant un 1

J'ai d'abord redimensionné ces données pour passer d'une dimension 1797x8x8 à une dimension 1797x64 (les images sont maintenant représentées par un vecteur de taille 64). Par exemple, la ligne 1 représente l'image 1. J'applique ensuite 2 noyaux différents sur ce même jeu de données qui me donne 2 matrices K et L, je normalise mes données. Je les centre et j'applique une descente du gradient.

A partir de la matrice obtenue qui ne contient que des réels entre 0 et 1, je transforme le résultat obtenu pour avoir une vraie matrice de permutation. Pour cela, je cherche le maximum dans ma matrice je le met à 1 puis je met le reste de sa ligne et sa colonne à 0, je fais cela n fois jusqu'à avoir une matrice de permutation. On peut aussi utiliser l'algorithme hongrois pour faire cette transformation.

Je peux ensuite évaluer la pertinence de mes résultats car ce jeu de données propose une partie qui pour chaque image i nous donne le numéro auquel correspond l'image. J'obtiens donc des scores d'appariement tel que 1200/1797 qui m'indique que 1200 images sur 1797 ont trouvé une bonne correspondance.

Je ne met pas ici mes résultats car à cette étape de mon stage, j'interprétai mal mes résultats, ce n'est donc pas intéressant. J'ai ensuite continué mes tests sur la base de données cifar-10.

Étant donné que mes résultats précèdent était très peu faussé. J'ai voulu vraiment validé (ou pas) l'algorithme, pour cela j'ai commencé par comparer 2 ensembles d'images identiques où les images étaient juste mélangées. Ici on avait donc une correspondance parfaite entre les images. C'est à ce point que je me suis rendu compte que je regardai mes résultats à l'envers depuis le début. Je pensais que lorsque $\pi_{i,j} = 1$ c'était x_i qui correspondait à y_j alors que c'était le contraire (x_j correspond à y_i).

J'ai donc réalisé plusieurs expériences sur ces comparaisons de mêmes images pour voir si l'algorithme trouve à tous les coup : et bien non en moyenne l'algorithme trouve la solution 1 fois sur 2 (cela est dû sûrement à une mauvaise initialisation ou un mauvais jeu de paramètres).

Une initialisation avec les valeurs propres (fonction init_eig) nous permet dans certains cas de tomber directement sur la solution avec l'initialisation. (pourquoi ? je ne sais pas).

Je me suis ensuite attaqué à comparer plusieurs ensembles avec le HSIC multiple. J'ai commencé par comparer plusieurs ensembles contenant des images identiques. Avec des images identiques, l'initialisation avec les valeurs propres (voir init_eig dans util.py) donne souvent (environ 1 fois sur 2) le résultat optimal. Avec une initialisation aléatoire, j'ai remarqué que le résultat était l'initialisation, il ne se passe donc presque rien dans cette descente du gradient, il y a un problème.

J'ai eu des difficultés à trouver des bons paramètres de descente du gradient car le gradient devient vite énorme et on se heurte à des difficultés de dépassement mémoire.

2.4.2 Comparaison de graphes de pits

On rentre ici dans le vif du sujet après la comparaison d'images, on passe à la comparaison de graphes de pits. J'ai à disposition 134 graphes des hémisphères droit et gauche du cerveau. Je ne me suis intéressée à comparer que ceux de l'hémisphère gauche. A partir de ses graphes, Sylvain m'a sympathiquement préparé les matrices de Gram associé à ces graphes. Pour chaque graphe, on a 7 matrices de Gram différentes associés. Chacune de ces matrices comparent différents attributs des noeuds (pits).

Voici les numéros associés aux matrices de Gram selon les attributs pris en compte dans la construction de la matrice :

- 0 = structure, coordonnées, profondeur
- 1 = coordonnées, profondeur
- 2 = structure, profondeur
- 3 = structure, coordonnées
- 4 = structure
- 5 = coordonnées
- 6 = profondeur

On pourra donc ensuite comparer les résultats avec les différents noyaux utilisés. Pour le début, on utilisera seulement le noyau coordonnées qui devrait normalement faire correspondre les points les plus proches entre eux. Cela nous permettra de valider ou non visuellement les résultats.

Étant donnée que tous les graphes n'ont pas le même nombre de noeuds (les patients n'ont pas le même nombre de pits), et que pour appliquer notre algorithme il faut que les matrices soient de même taille, je prend le nombre maximum de noeud dans les graphes que je vais comparer et j'ajoute à ceux qui n'ont pas ce nombre maximum de noeuds "des pits fictifs" qui ne sont proches que d'eux même dans les matrices de Gram, c'est à dire des lignes de 0 avec des 1 sur la diagonale. Toutes les matrices de Gram font alors la même taille et on peut appliquer l'algorithme. (Il y a peut-être une meilleure méthode pour gérer cela.)

Pour la visualisation des résultats, on colore les pits matchés entre eux de la même couleur, et on les met sur une sphère représentant un hémisphère du cerveau "gonflé". On peut ensuite voir à l'oeil si le matching a du sens. Les pits matchés avec des pits fictifs ne sont pas représentés.

On peut aussi voir les correspondances d'une autre manière en regardant les graphes et non les pits sur la sphère (show_graph_for_2.py dans show_results_on_sphere). Je n'ai codé cette fonction que pour l'appariement de 2 graphes (et pas pour plusieurs). Je n'ai pas commenté ces résultats dans ce rapport, j'ai mis ce que j'ai obtenu en annexe. (voir Annexes - Résultats des matching de graphes)

De plus, j'ai implémenté une fonction qui peut faire office de métrique faisant la moyenne des distances géodésiques (sur la sphère) de chaque paires matchés ensemble. (fonction metric_geodesis dans util.py) Dans ce qui suit cette mesure s'appellera d.

2.4.2.a Comparaison entre 2 graphes

Le fichier matching_two_graph.py contient les méthodes pour l'appariement de 2 graphes entier. Et le fichier matching_graph_on_pt_of_interest.py permet de faire l'appariement entre deux graphes sur des zones d'intérêts (zone 1 centré sur le point [-43, 6, 89] avec un rayon de 80, et zone 2 centré sur [-48, 8, -87] avec un rayon de 60).

J'ai vite compris que les résultats de l'appariement entre deux graphes entiers étaient difficile à interpréter. (voir 7) Il y a des points de couleurs un peu partout c'est compliqué à analyser.

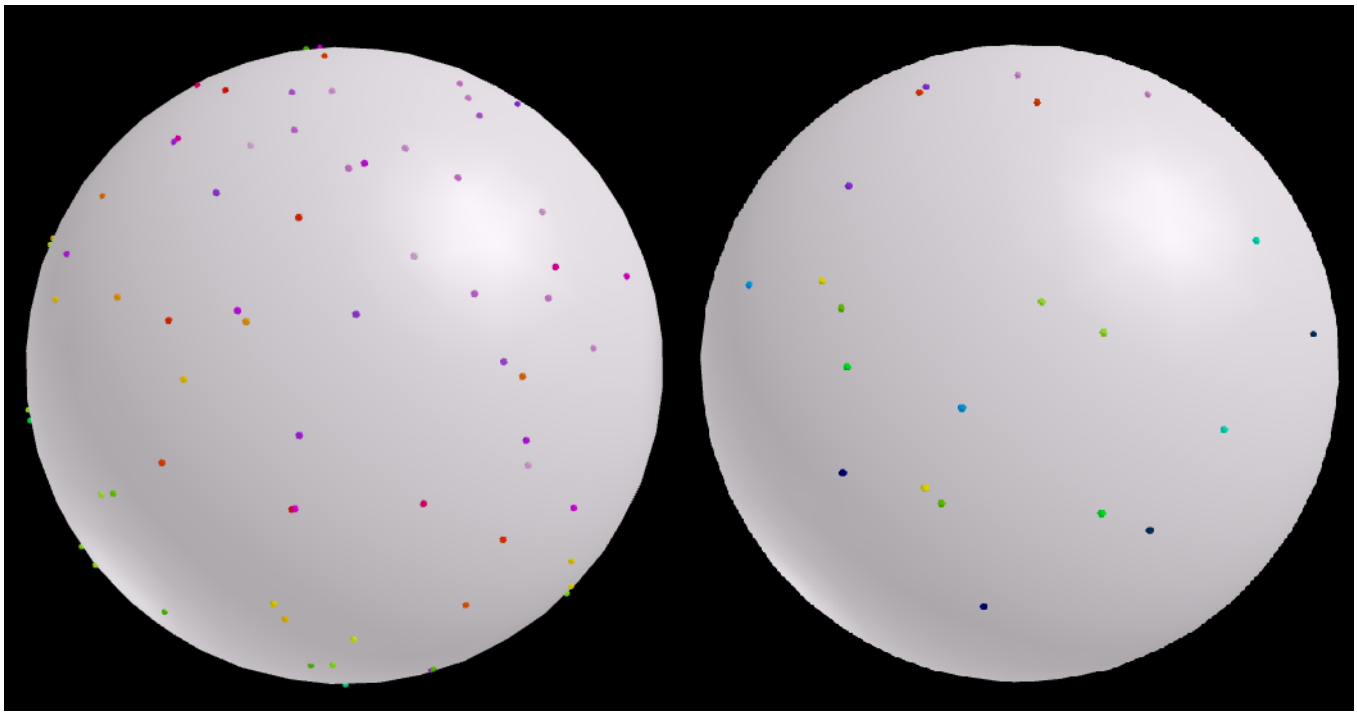


FIGURE 7 – Exemple d'un appariement entre 2 graphes complets (les 2 côtés de la sphère)

On va donc s'intéresser à un appariement dans certaines zones plutôt que sur la totalité de l'hémisphère.

Pour commencer, j'ai fait l'appariement de 2 graphes de même taille grâce au Convex Kernelized Sorting(CKS) avec un noyau type coordonnées qui devrait permettre de matcher les pits les plus proches entre eux. J'ai ensuite testé avec des graphes de différentes tailles. Comme le résultats dépend beaucoup de l'initialisation je réalise nb_tests fois la procédure avec une initialisation aléatoire, et je prend celle qui a la plus petite valeur pour la fonction objectif.

Quelques résultats ont du sens, mais certaines correspondances semblent être trop éloignées pour réellement être ensemble. (voir 8)

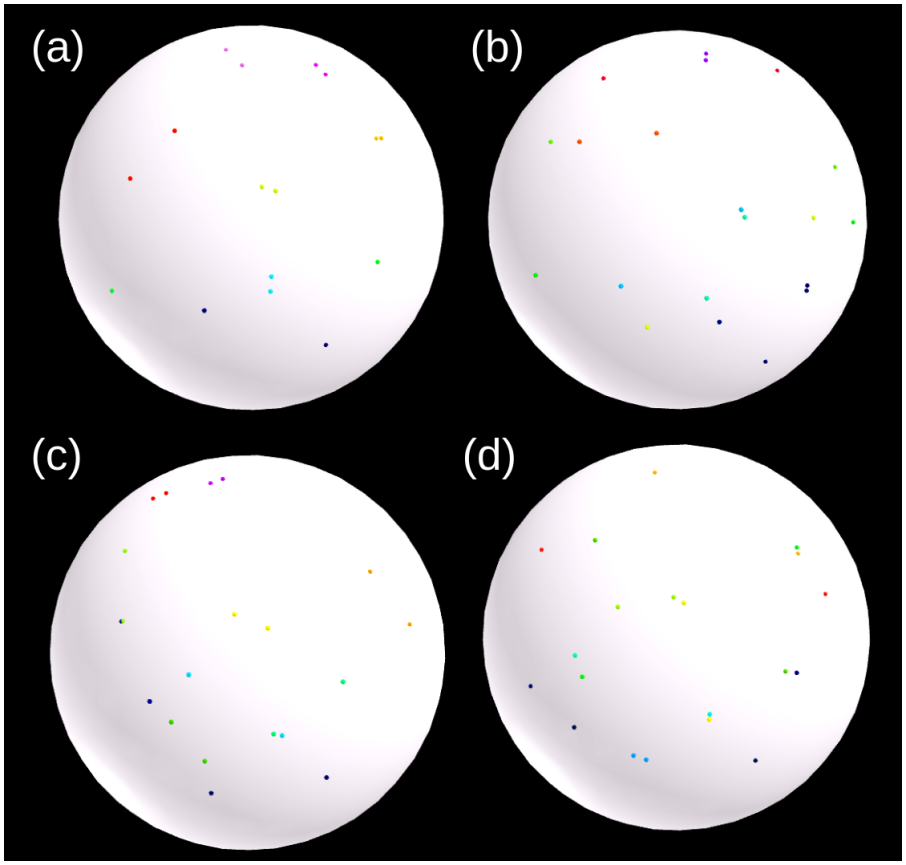


FIGURE 8 – Exemples de correspondances entre 2 graphes (mu=1 mu_min=1e-5 it=1000 c=1 nb_test=500)

(a) : Graphes 17 et 28 tous deux de taille 14
(b) : Graphe 30 de taille 8 et graphe 2 de taille 10
(c) : Graphe 36 de taille 11 et graphe 101 de taille 10
(d) : Graphe 6 de taille 10 et graphe 130 de taille 13

(voir annexe pour mieux voir ces graphes)

Après ces premiers tests, je me suis demandé si l'algorithme de CKS trouvait bien le résultat optimal. Pour cela, j'ai utilisé l'algorithme du branch and bound qui est beaucoup plus long mais qui nous permet de trouver la valeur optimale qu'on cherche soit :

$$\arg\min_{\pi \in \pi_m} \|\overline{K}\pi^T - (\overline{L}\pi)^T\|_F^2 \quad \text{s.t.} \quad \begin{cases} \pi \in \{0,1\} \\ \pi 1_m = 1_m \\ \pi^T 1_m = 1_m \end{cases} \tag{9}$$

En générale, le CKS ne trouve pas le résultat optimal.

De plus, j'ai remarqué que les résultats trouvés avec le branch and bound donc les résultats optimaux ne collent pas toujours avec ce qu'on espère : on remarque que certaines correspondances qui semblent triviales (les points pratiquement les uns sur les autres) ne sont pas toujours matchés ensembles (voir 9 à gauche).

Je me suis donc demandé si le branch and bound avait bien trouvé le minimum de la fonction. J'ai donc cherché à trouver bêtement le minimum de cette fonction pour ce graphe (qui était de taille 10 j'ai de la chance) en parcourant l'ensemble des permutations de taille 10. Le minimum est bien celui trouvé par le branch_and_bound.

J'ai ensuite fait un appareillage "à la main" (voir 9 à droite) qui me paraissait plus logique, pour connaître le résultat de la fonction objectif et comparer. On obtient 2.33 donc bien plus que le match optimal trouvé par l'algorithme (1.06).

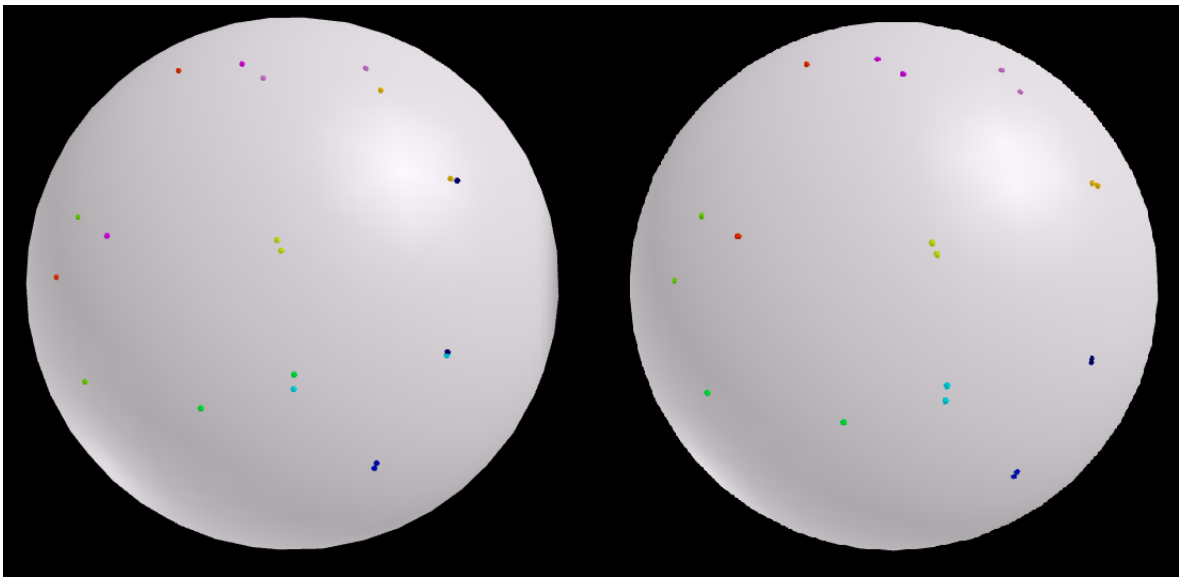


FIGURE 9 – Correspondances optimales trouvé par l'algorithme (d=40) vs correspondances instinctive (d=15)

On peut aussi comparer les résultats que nous donne le matching selon les différents noyaux. Les appariement optimaux trouvé avec le noyau structure + coordonnées, sont exactement les mêmes pour ces 2 graphes (2 et 3). Cependant, avec le noyau structure + coordonnées + profondeur on obtient un résultat différent. (de peu) (voir 10). Le choix du noyau a donc une influence sur le résultat.

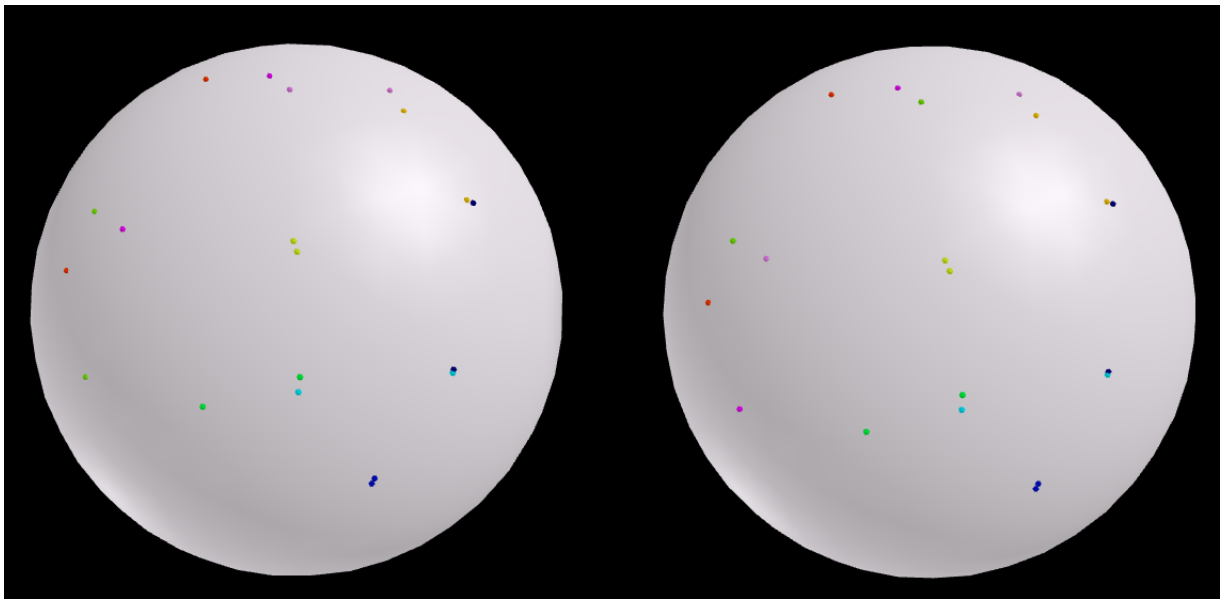


FIGURE 10 – Évolution du matching selon les noyaux (à gauche noyau coordonnées, à droite coordonnées + structure + profondeur)

2.4.2.b Comparaison multiple de graphes

Le fichier multiway_matching_graphs.py propose un appariement multiple grâce à la formule du HSIC multiple.

L'appareillage pour une multitude de graphes pose problème, il faut trouver un bon paramètre mu pour la descente du gradient que je cherche en réglant mu et mu_min "à taton", si mu est trop grand les calculs explosent et on a une erreur de "overflow encountered". Et si mu est trop petit il ne se passe rien, le résultat reste l'initialisation.

Il faudrait trouver une autre méthode de résolution de ce problème ou tester avec une descente du gradient où le pas mu se règle automatiquement pour le problème. Ou alors il y a une erreur dans le calcul du gradient.

En attendant, pour un matching multiple j'applique mon algorithme de comparaison de 2 graphes entre un graphe et tous les autres à la suite sur des zones d'intérêts. Je compare le graphe 1 avec le 2, puis avec le 3 ... jusqu'au 134. (voir fichier matching_on_pt_of_interest.py). Il faut donc choisir une sorte de graphe "modèle" qui va servir à être comparé avec tous les autres.

Dans un premier temps, je n'ai pas prêté attention au choix de ce modèle, et j'ai pris tout simplement le premier graphe de ma liste. Puis j'ai changé de modèle en prenant un des graphes qui a le nombre maximum de pits, à l'oeil nu c'est difficile de voir la différence entre les 2.

Quelques résultats en images : (c = 1, mu = 1, mu_min = 1e-4, it = 800, et nb_tests = 200)

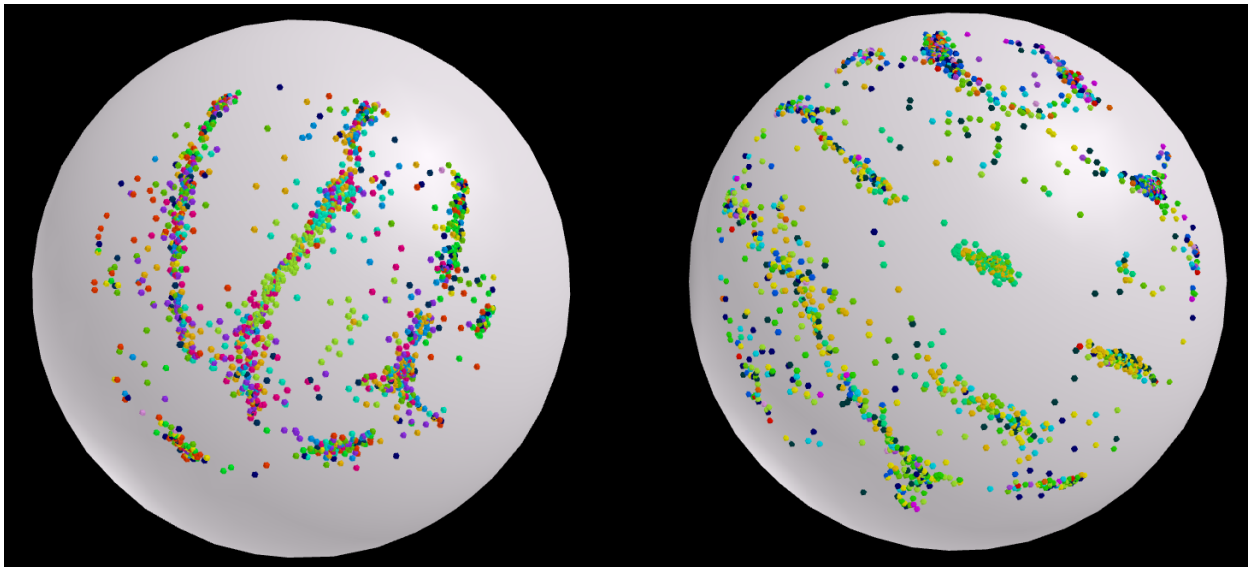


FIGURE 11 – Correspondances entre pits de 134 personnes sur 2 zone d'intérêt (noyau coordonnées)

à gauche : zone d'intérêt [-48, 8, -87] (d=45.07)
à droite : zone d'intérêt [-43, 6 , 89] (d=71.93)

On remarque qu'on a quand même des "paquets" de même couleur comme au milieu, mais on a aussi des endroits où c'est chaotique, ça ressemble à un festival de confettis et c'est difficile de conclure.

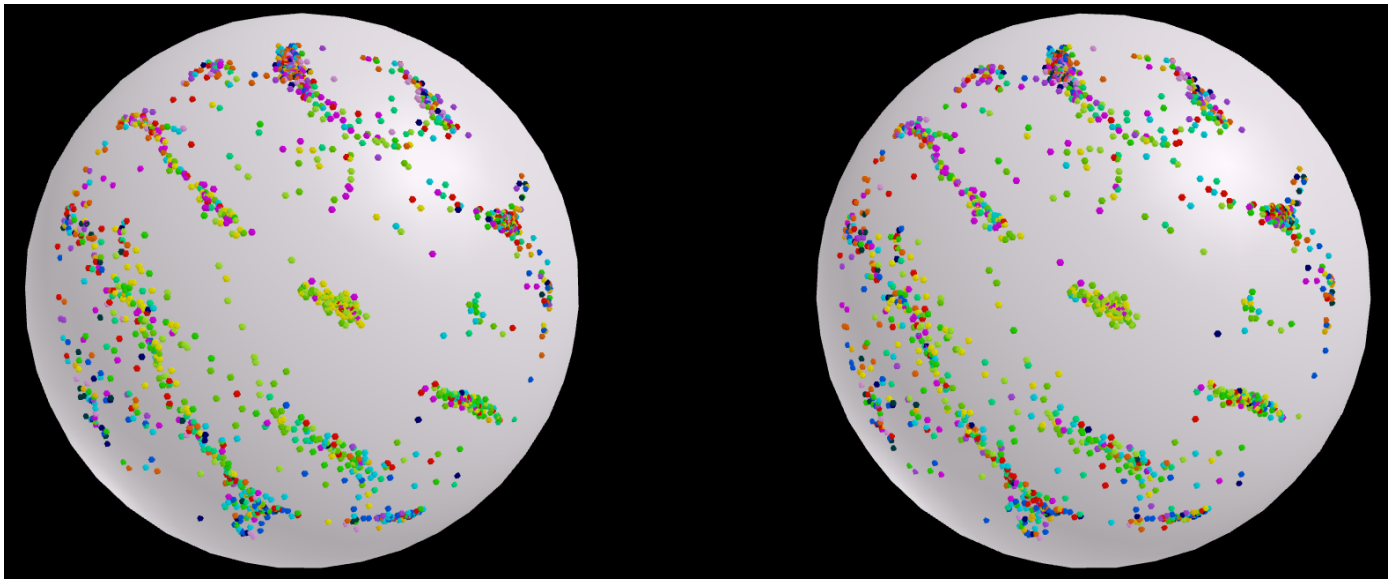


FIGURE 12 – Différences des noyaux

à gauche : noyau coordonnées (d=57.12)

à droite : noyau coordonnées + structure + profondeur (d=60.09)

Le problème de cette visualisation des résultats c'est que l'on ne peut pas voir les plis et donc si les correspondances ont vraiment du sens. Il faudrait une métrique qui nous permettrait de mieux interpréter ces résultats.

Je me demande si ces correspondances invraisemblables comme les points rouges entourés dans 13 ne sont pas dû à une sorte de symétrie des distances. Par exemple, tous les points sur la bande rouge sont à la même distance des points rouges de la zone 1 que des points rouges de la zone 2. Cela pourrait expliquer leur appariement.

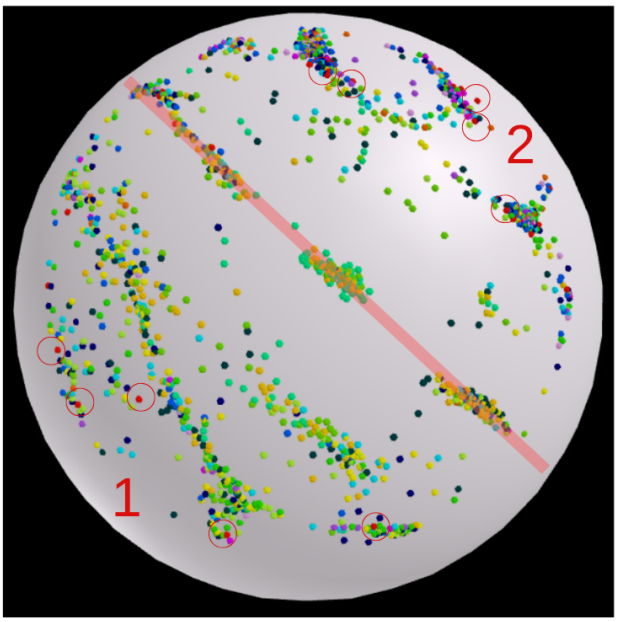


FIGURE 13 – Représentation de la symétrie des résultats

2.5 Améliorations

Dans un premier temps, il faudrait une solution pour la résolution du problème multiple qui permettrait un appareillage dans l'ensemble et non 2 à 2. Il faudrait pour cela une descente du gradient qui s'adapte vraiment au problème, ou alors changer complètement de méthode de résolution.

Comme l'interprétation des résultats est difficile avec ces graphes réels, une idée pour réellement quantifier les résultats serait de réaliser des faux graphes en prenant une sorte de modèle et en ajoutant du bruit sur l'emplacement des pits et sur la structure à la main de façon à connaître les correspondances.

L'algorithme du branch and bound nous offrant la solution optimale vaudrait peut-être le coup si certains pits trop éloignés par exemple, était dès le début impossible à matcher (établir une distance au delà de laquelle il est impossible que 2 pits se correspondent), avec la possibilité de paralléliser l'algorithme. On pourrait ainsi accélérer l'algorithme et cela nous permettrait peut-être de l'utiliser sur des plus grandes matrices.

Il faudrait régler le problème des pits qui ont des structure autour semblable qui match entre eux alors qu'ils sont à 2 côtés opposés.

On pourrait aussi changer de métrique pour l'évaluation des résultats, par exemple en ajoutant un critère de comparaison de la profondeur de nos pits matchés.

3 Annexes

3.1 Dérivée de la fonction objectif

Notre fonction objectif est :

$$\begin{aligned} f(\pi) =& ||K\pi^T - (L\pi)^T||_F^2 + c(||\pi \mathbb{1}_m - \mathbb{1}_m||_2^2 + ||\pi^T \mathbb{1}_m - \mathbb{1}_m||_2^2) \\ =& g(\pi) + c.h(\pi) \end{aligned}$$

$$\begin{aligned} g(\pi) =& \|K\pi^T - (L\pi)^T\|_F^2 \\ =& \|K\pi^T\|_F^2 + \|(L\pi)^T\|_F^2 - 2.tr(\pi K^T L\pi) \\ =& tr(\pi K^T K\pi^T) + tr(L\pi\pi^T L^T) - 2.tr(\pi K^T L\pi) \end{aligned}$$

$$\begin{aligned} h(\pi) =& \|\pi \mathbb{1}_m - \mathbb{1}_m\|_2^2 + \|\pi^T \mathbb{1}_m - \mathbb{1}_m\|_2^2 \\ =& \|\pi \mathbb{1}_m\|_2^2 + \|\mathbb{1}_m\|_2^2 - 2 < \pi \mathbb{1}_m, \mathbb{1}_m > + \|\pi^T 1\|_2^2 + \|\mathbb{1}_m\|_2^2 - 2 < \pi^T \mathbb{1}_m, \mathbb{1}_m > \\ =& \mathbb{1}_m^T \pi^T \pi \mathbb{1}_m + n^2 - 2(\mathbb{1}_m^T \pi 1) + \mathbb{1}_m^T \pi\pi^T \mathbb{1}_m + n^2 - 2(\mathbb{1}_m^T \pi^T \mathbb{1}_m) \end{aligned}$$

$$\begin{aligned} \frac{\partial g}{\partial \pi} =& 2\pi K^T K + 2L^T L\pi - 2(\pi^T L^T K + L^T K\pi^T) \\ =& 2(\pi K^T - \pi^T L^T)K - 2L^T(K\pi^T - L\pi) \end{aligned}$$

$$\begin{aligned} \frac{\partial h}{\partial \pi} =& \pi(\mathbb{1}_m \mathbb{1}_m^T + \mathbb{1}_m \mathbb{1}_m^T) - 2(\mathbb{1}_m \mathbb{1}_m^T) + \pi^T(\mathbb{1}_m \mathbb{1}_m^T + \mathbb{1}_m \mathbb{1}_m^T) - 2(\mathbb{1}_m \mathbb{1}_m^T) \\ =& 2\pi \mathbb{1}_{m\times m} + 2\pi^T \mathbb{1}_{m\times m} - 4\mathbb{1}_{m\times m} \\ =& 2(\pi + \pi^T - 2I_m)\mathbb{1}_{m\times m} \end{aligned}$$

Finalement, on obtient :

$$\frac{\partial f}{\partial \pi} = 2(\pi K^T - \pi^T L^T)K - 2L^T(K\pi^T - L\pi) + c(2(\pi + \pi^T - 2I_m)\mathbb{1}_{m\times m}) \tag{10}$$

Voir dans le Matrix Cookbook pour les dérivées.[3]

3.2 Quelques fonctions noyaux

Soient x et y des vecteurs.

Le noyau linéaire : $k(x, y) = x^T y$.

Le noyau polynomial : $k(x, y) = (\gamma x^T y + c_0)^d$ avec γ, c_0, et d à fixer.

Le noyau gaussien : $k(x, y) = exp(-\gamma||x - y||^2)$ avec $\gamma = -\sigma^{-2}$.

3.3 Récapitulatif

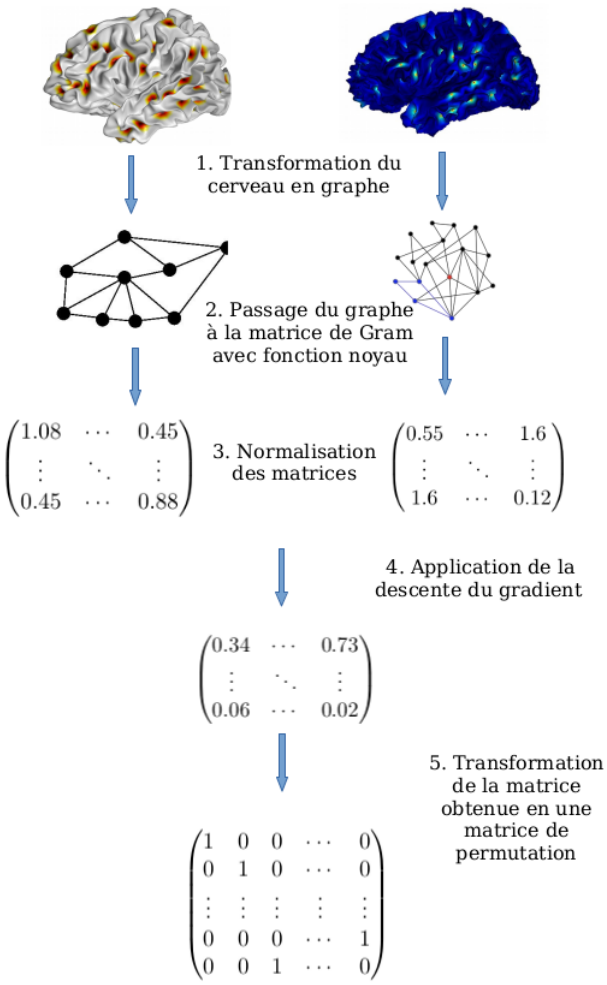


FIGURE 14 – Schéma récapitulatif de la méthode appliquée pour la comparaison de 2 cerveaux

Dans le point 3, après la normalisation des matrices, il faut les centrer.

3.4 Résultats des matching de graphes

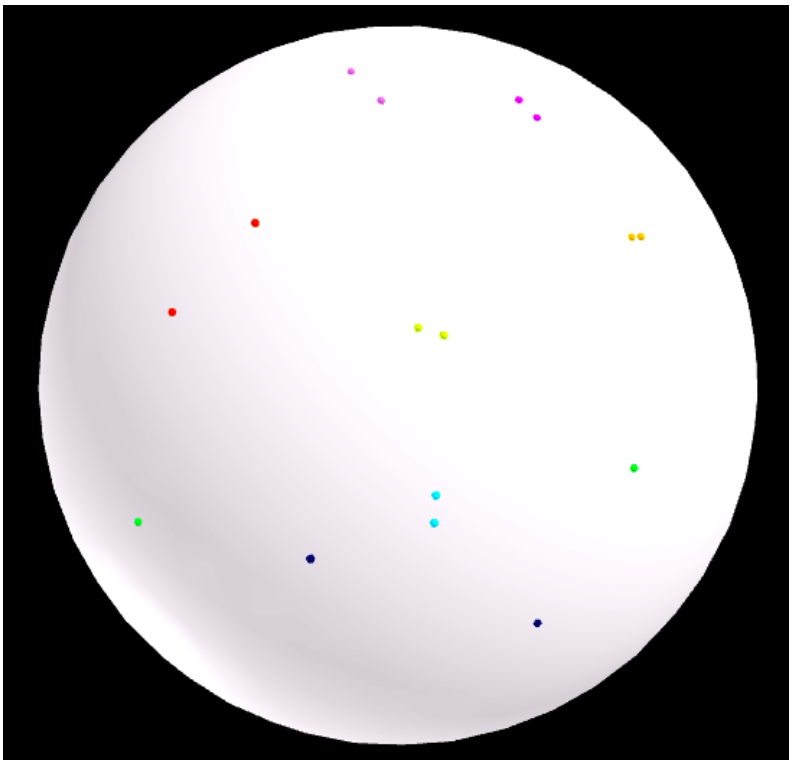


FIGURE 15 – Graphes (a) de la figure 8

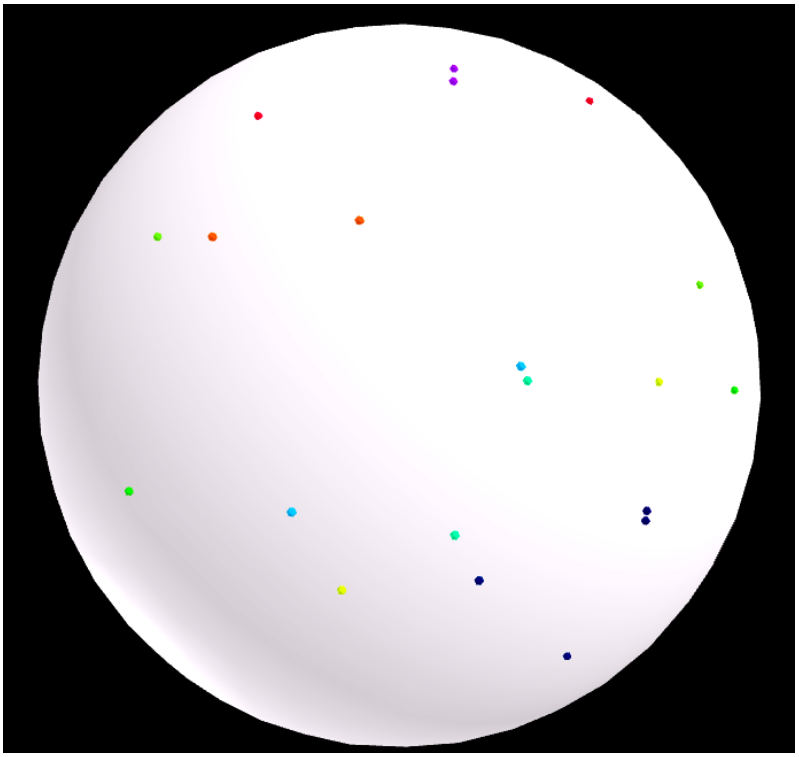


FIGURE 16 – Graphes (b) de la figure 8

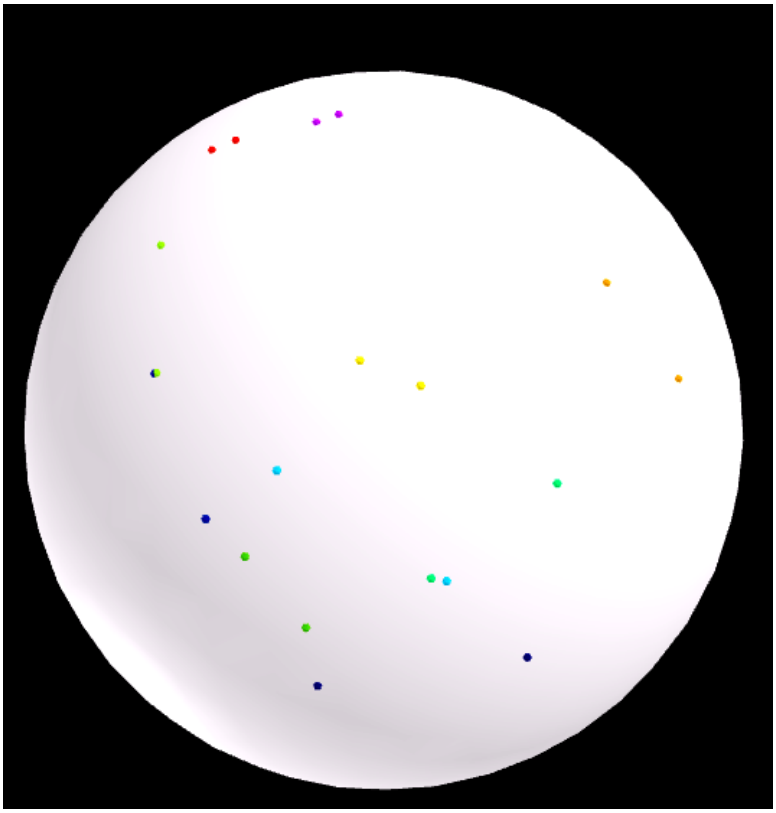


FIGURE 17 – Graphes (c) de la figure 8

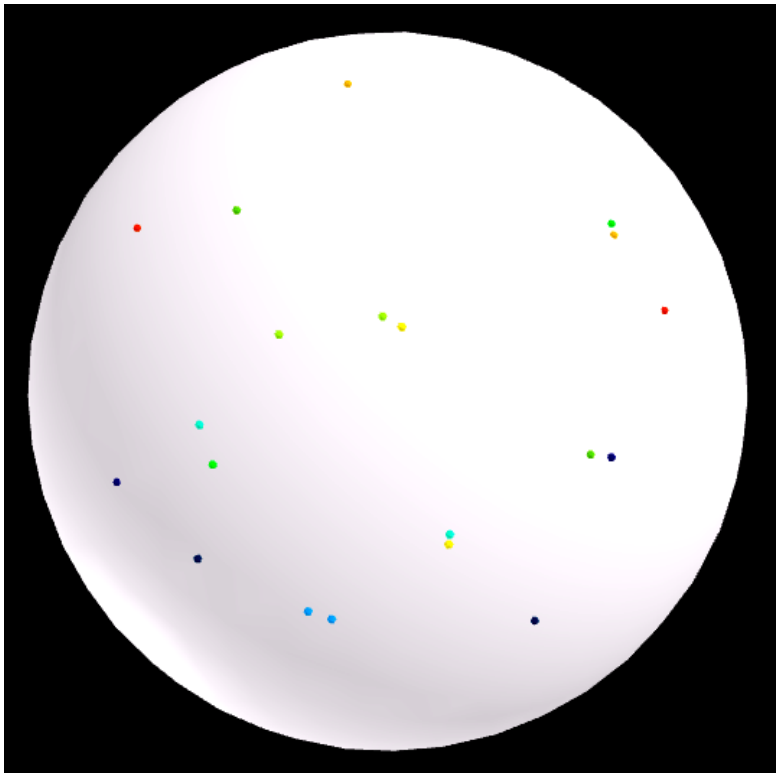


FIGURE 18 – Graphes (d) de la figure 8

Dans ce qui suit l'algorithme utilisé est celui du branch and bound, on a donc la solution optimale. Le noyau utilisé prend en compte la structure du graphe et les coordonnées des pits. Les paramètres étant : mu=1 mu_min=1e-5 it=1000 c=1.

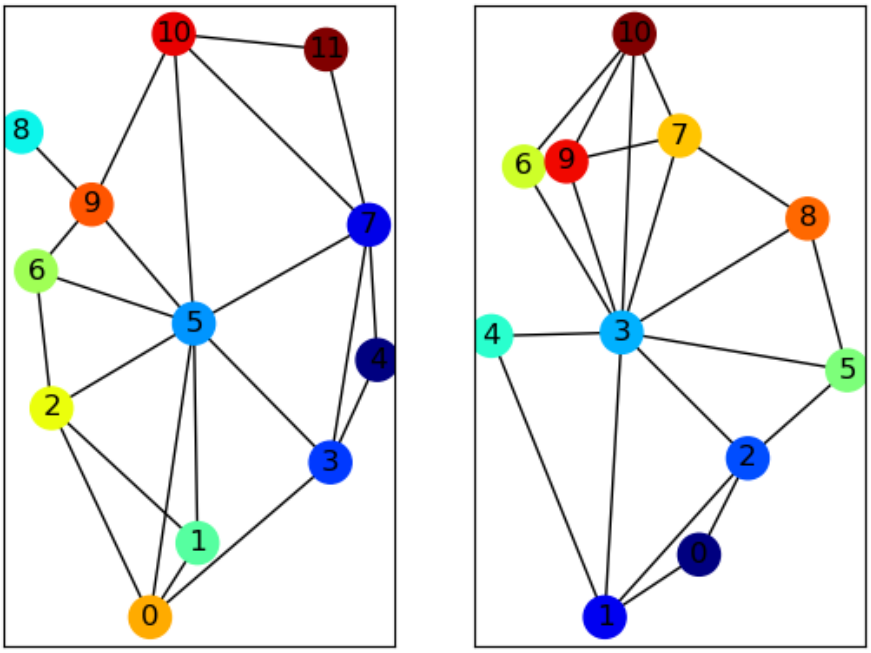


FIGURE 19 – Comparaison graphes 20 et 33 (d=48)

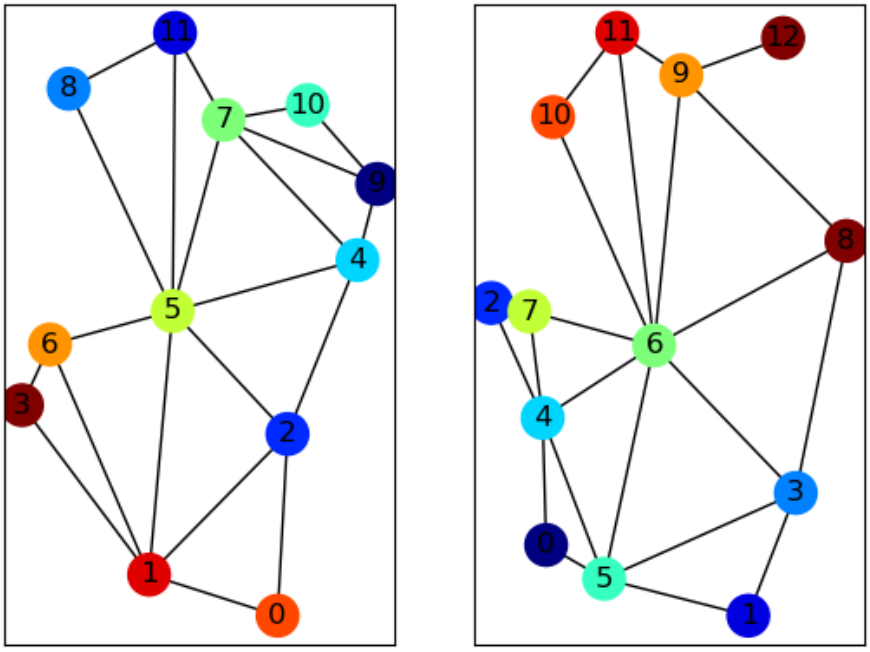


FIGURE 20 – Comparaison graphes 2 et 3 (d=40)

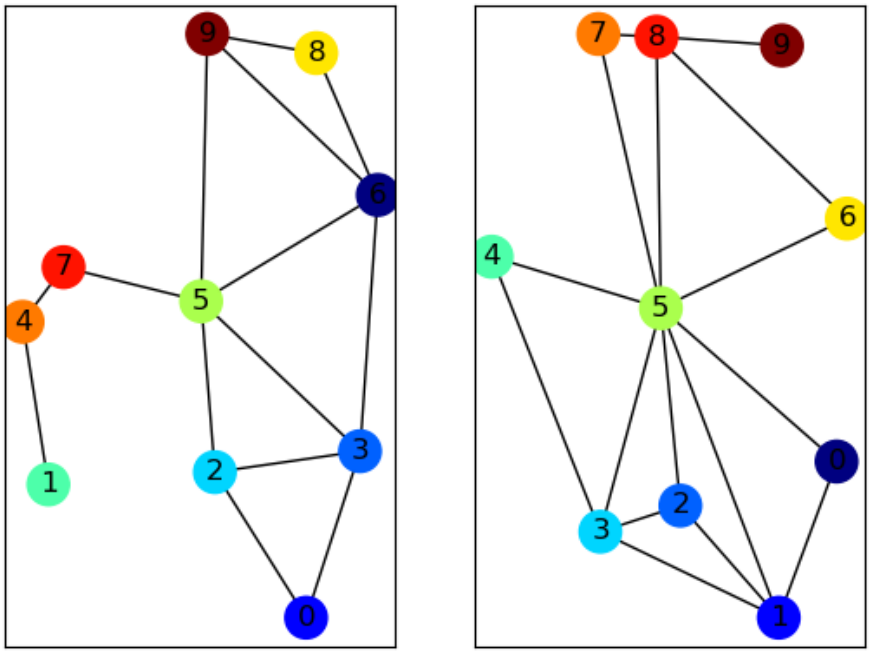


FIGURE 21 – Comparaison graphes 27 et 100 (d=94)

Pour les 3 graphes précedents, le matching a été fait dans la zone d'intérêt 1.

Bibliographie

[1] Nemanja Djuric, Mihajlo Grbovic et Slobodan Vucetic. « Convex Kernelized Sorting ». In : *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (2012), p. 893–899.

[2] Valentin Emiya et al. « Compressed sensing with unknown sensor permutation ». In : (2014).

[3] Kaare Brandt Petersen et Michael Syskind Pedersen. *The Matrix Cookbook*. 2012.

[4] Novi Quadrianto, Le Song et Alex J. Smola. « Kernelized Sorting ». In : *Advances in Neural Information Processing Systems 21* (2008), p. 1–9.

[5] Sylvain Takerkart, Guillaume Auzias et Lucile Brun. « Structural graph-based morphometry : A multiscale searchlight framework based on sulcal pits ». In : *Medical Image Analysis* (2017), p. 32–45.