TIPE : Interpolation d'images et Transport Optimal

Martin CUINGNET
Numéro d'inscription : 22268

Sommaire

La ville et le transport optimal

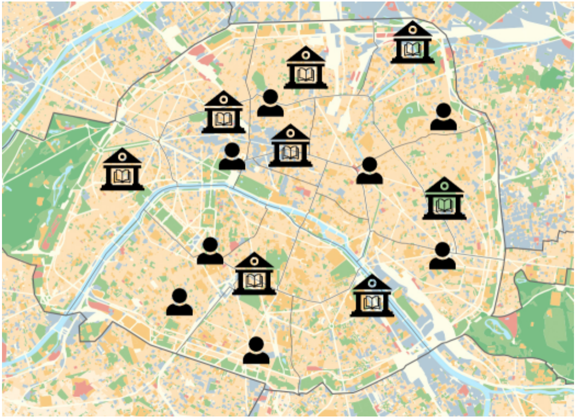


Figure – Carte des étudiants et des bibliothèques à apparier

La ville et le transport optimal

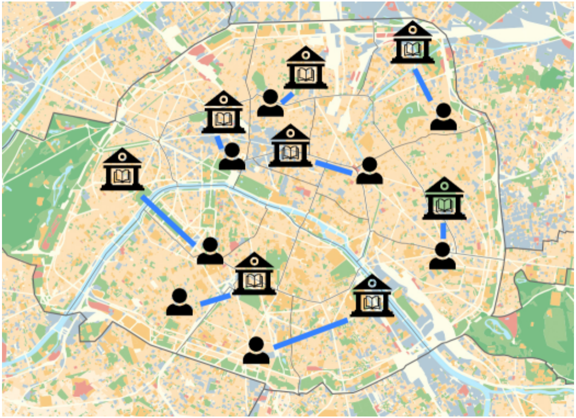


Figure – Plan d'appariement avec le transport optimal

Objectifs

Réaliser des interpolations entre plusieurs images.
Interpolations de nature :

- Géométrique
- Colorimétrique

Interpolation : Conserver les informations (colorimétriques et géométriques) contenues dans les images interpolées.

Méthode Naïve

Figure – Moyenne pondérée entre les images des logos de Batman et Capitain America

Monge et le transport optimal

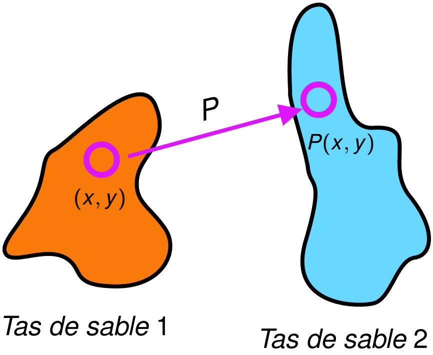
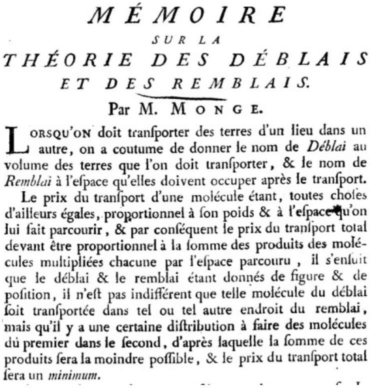


Figure – Plan de transport entre deux tas de sables



MÉMOIRE

SUR LA

THÉORIE DES DÉBLAIS ET DES REMBLAIS.

Par M. MONGE.

LORSQU'ON doit tranſporter des terres d'un lieu dans un autre, on a coutume de donner le nom de *Déblai* au volume des terres que l'on doit tranſporter, & le nom de *Remblai* à l'eſpace qu'elles doivent occuper après le tranſport.

Le prix du tranſport d'une molécule étant, toutes choſes d'ailleurs égales, proportionnel à ſon poids & à l'eſpace qu'on lui fait parcourir, & par conſéquent le prix du tranſport total devant être proportionnel à la ſomme des produits des molécules multipliées chacune par l'eſpace parcouru, il s'enſuit que le déblai & le remblai étant donnés de figure & de poſition, il n'eſt pas indifférent que telle molécule du déblai ſoit tranſportée dans tel ou tel autre endroit du remblai, mais qu'il y a une certaine diſtribution à faire des molécules du premier dans le ſecond, d'après laquelle la ſomme de ces produits ſera la moindre poſſible, & le prix du tranſport total ſera un *minimum*.

Figure – 1ere page de Théorie des Remblais et des Déblais de Gaspard Monge en 1784

Introduction au Transport Optimal

Plan de transport : application bijective $P : \quad \to$

Valeur d'un plan de transport :

$$V(P) = \iint_{(x,y)\in} ||P(x,y) - (x,y)|| dxdy$$

Plan de transport optimal : Plan de transport P^* minimisant la valeur Ainsi :

$$P^* = min \left\{ \iint_{(x,y)\in} ||P(x,y) - (x,y)|| dxdy \ , \ P : \quad \to \right\}$$

Formalisation mathématique discrète

Soit X et Y des ensembles finis de points de même cardinal.
Plans de transports $P : X \to Y$. Plan de transport optimal :

$$P^* = min\left\{ V(P) \,\middle|\, P : X \to Y, \text{bijective} \right\}$$

avec $V(P) = \sum_{x \in X} \|x - P(x)\|$, valeur plan de transport.
En pratique : solution exacte irréalisable $\Rightarrow$ solution approchée

Application du Transport Optimal à l'interpolation d'image

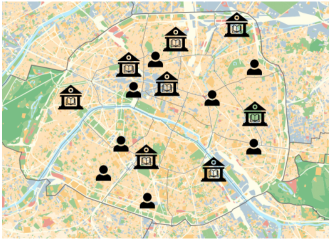


Figure – Image de départ

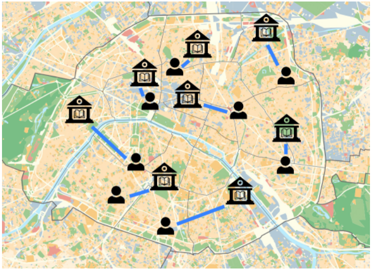
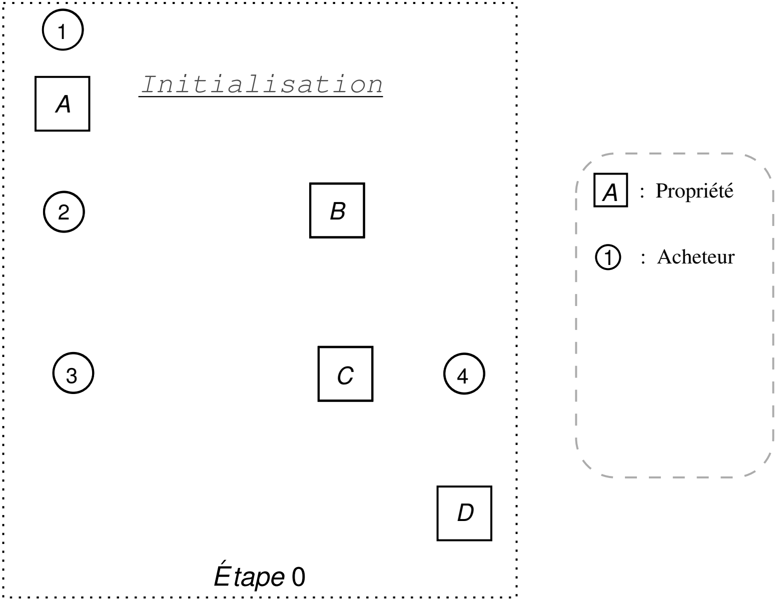
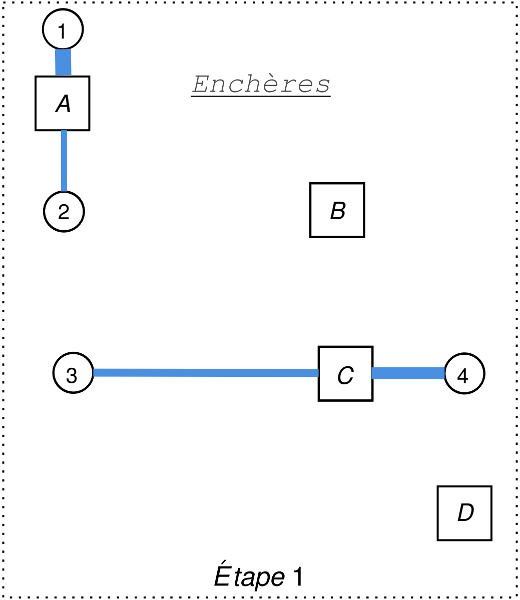


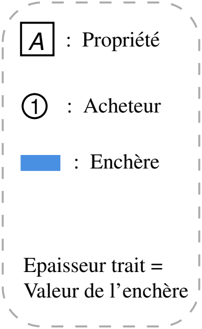
Figure – Image d'arrivée

Présentation de l'algorithme des enchères

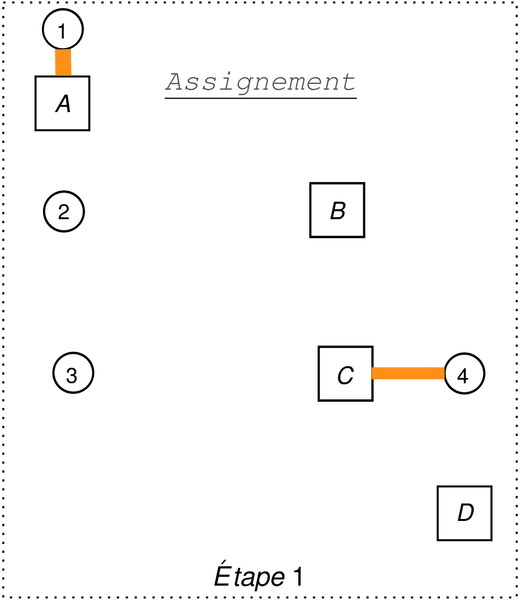


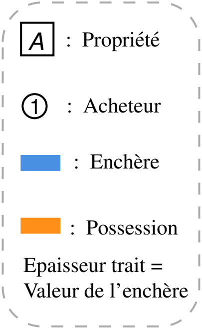
Présentation de l'algorithme des enchères



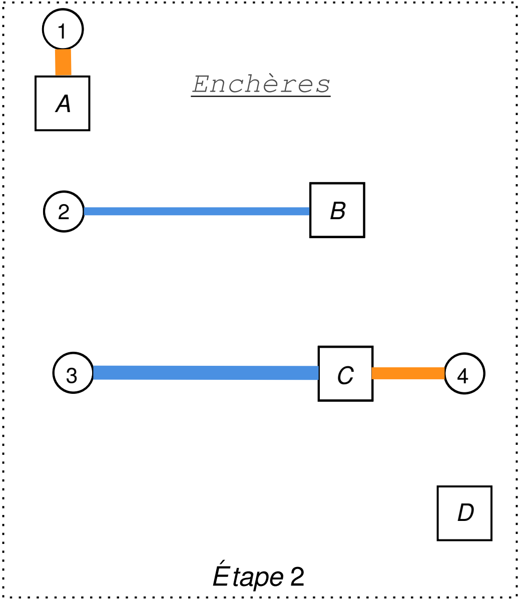


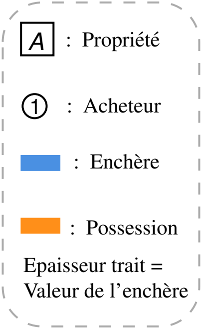
Présentation de l'algorithme des enchères



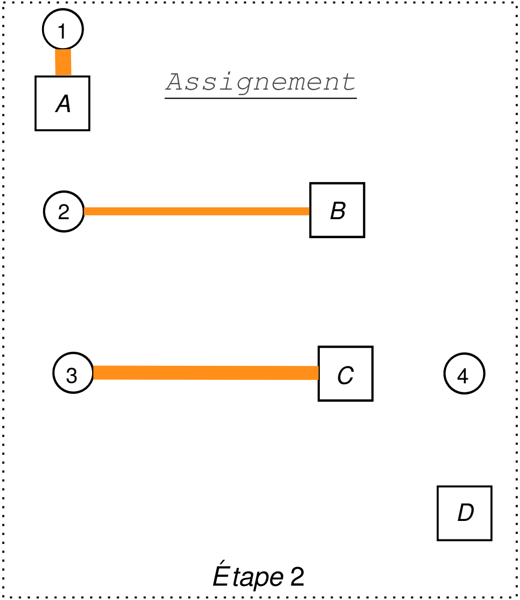


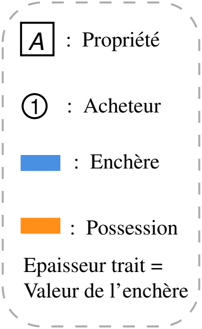
Présentation de l'algorithme des enchères



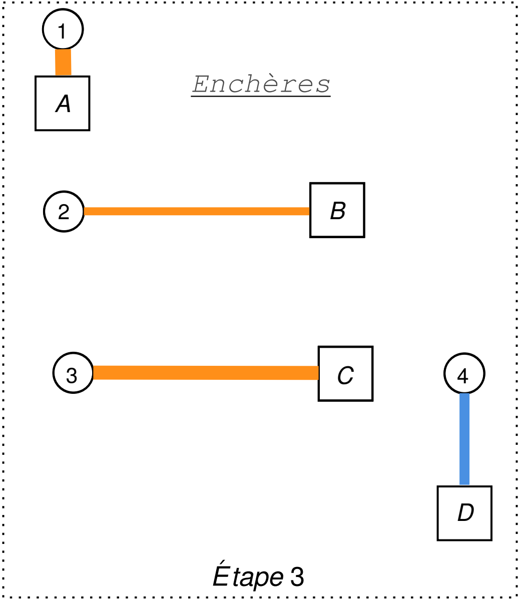


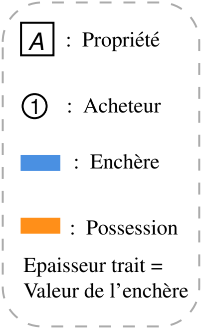
Présentation de l'algorithme des enchères



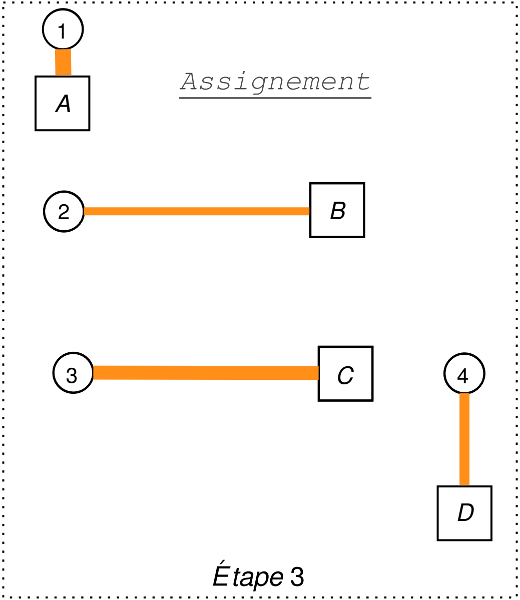


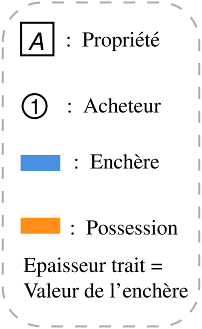
Présentation de l’algorithme des enchères





Présentation de l'algorithme des enchères





Pseudo-code de l'algorithme des enchères

```
Algorithme des enchères (ensemble A de taille n,
ensemble B de taille n, réel ε) :

gains  ←  obtention matrice de gain
prix  ←  liste nulle de taille n
enchères  ←  tableau nul de taille n×n
appariement  ←  liste vide

tant que tous les acheteurs ne sont pas appariés :

    pour chaque acheteur non apparié :
        bénéfices  ←  gains - prix
        max, max_id, deux_max  ←  bénéfice max, son index, ainsi que deuxième bénéfice max
        enchères.( index de l'acheteur, max_id)  ←  prix.(max_id) + (max - deux_max) + ε

    pour chaque objet sur lequel on a placé une enchère :
        max_ench, max_ench_id  ← obtention de l'enchère max et son index
         retirer à  appariement  le précédent détenteur de l'objet
         ajouter à  appariement  l'enchérisseur max comme détenteur de l'objet
        prix.(max_ench_id)  ←  max_ench

    enchères  ←  tableau nul de taille n × n

retourner appariement
```

Algorithme des enchères : Terminaison et Correction

L'algorithme des enchères est correct et termine : le plan de transport obtenu est ε-optimal.

Idée de la preuve

- Terminaison : par l'absurde
- Correction : invariant de boucle avec condition des écarts complémentaires

Algorithme des enchères : Premiers résultats

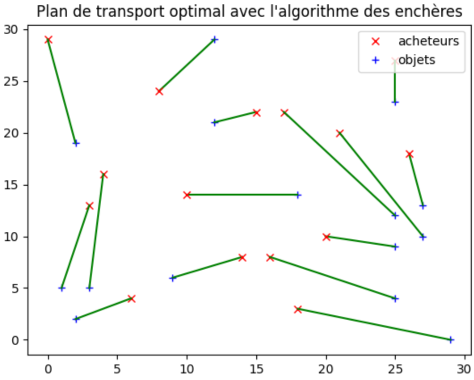


Figure – Exemple algorithme des enchères - $n = 15$, $\varepsilon = 0$

Algorithme des enchères : Premiers résultats

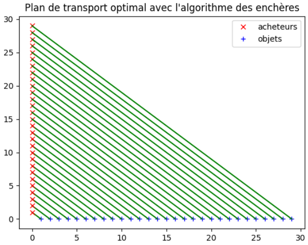


Figure – Exemple algorithme des enchères - $n = 29$, $\varepsilon = 0$

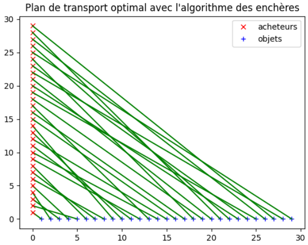


Figure – Exemple algorithme des enchères - $n = 29$, $\varepsilon = 0.5$

Algorithme des enchères : Analyse de la complexité

Étude complexité corps boucle `tant que`
Boucle `pour` enchère :

1. Vérifie si la personne est déjà appariée $\mathcal{O}(n)$
2. Créé la liste de bénéfices $\mathcal{O}(n)$
3. Trouve le premier et second bénéfice maximal. $\mathcal{O}(n)$
4. Enchérit sur l'objet au bénéfice maximal $\mathcal{O}(1)$

Complexité boucle `pour` enchère : $\mathcal{O}(n^2)$

Algorithme des enchères : Analyse de la complexité

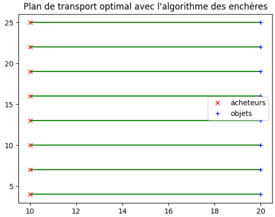
Boucle `pour` d'appariement :

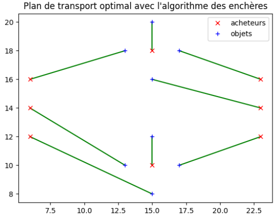
1. Trouve la personne ayant fait l'enchère maximale $\mathcal{O}(n)$
2. Vérifie si l'enchère maximale est non nulle $\mathcal{O}(1)$
3. Augmente le prix de l'objet $\mathcal{O}(1)$
4. Assigne l'enchérisseur maximal à l'objet. $\mathcal{O}(1)$

Complexité boucle `pour` appariement : $\mathcal{O}(n^2)$.
Complexité corps boucle `tant que` : $\mathcal{O}(n^2)$.

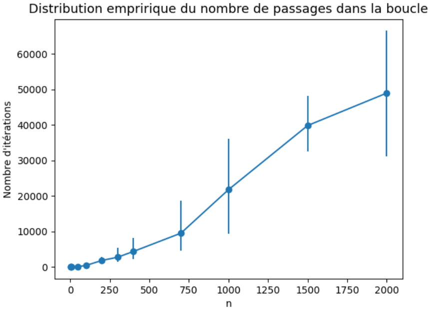
Algorithme des enchères : Analyse de la complexité

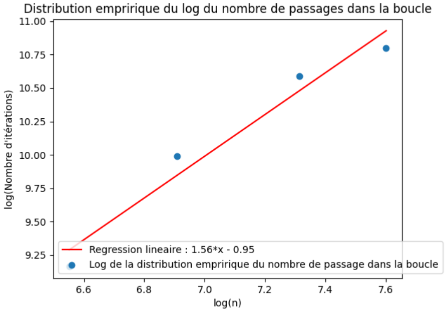
Nombre d'itérations de la boucle `tant que` : Difficile obtenir expression théorique. Avec $n = 8$, si k est le nombre d'itérations dans la boucle `tant que` :





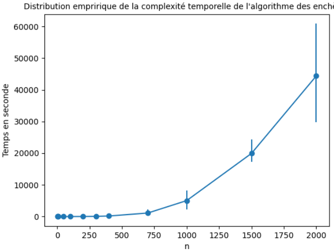
Algorithme des enchères : Analyse de la complexité

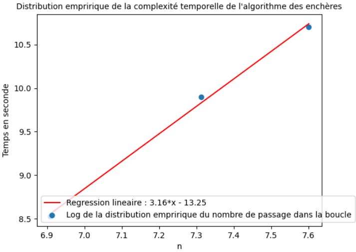




Nombre de passages dans la boucle `tant que` : $\mathcal{O}(n^{1.56})$.
Complexité totale théorique : $\mathcal{O}(n^{3.56})$.

Algorithme des enchères : Analyse de la complexité





Complexité totale empirique : $\mathcal{O}(n^{3.16})$, ce qui diffère légèrement de la complexité trouvée précédemment.

Résultats

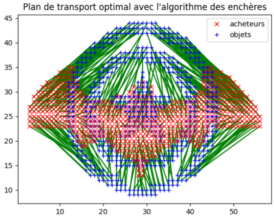


Figure – Plan de transport de Capitain America vers Batman

Figure – Capitain America vers Batman

Résultats

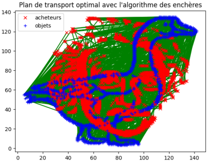


Figure – Plan de transport d'une oie vers Einstein

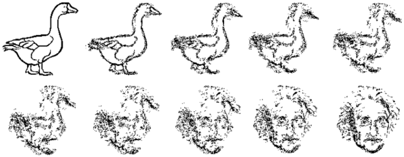


Figure – Une oie vers Einstein - $\varepsilon = 2$, $n = 2719$

Une nouvelle approche

Critique de l'approche précédente :

- Complexité trop importante en $\mathcal{O}(n^3)$
- Donne un plan de transport bijectif, ce qui n'est pas nécéssaire pour de la synthèse d'image

On propose donc une nouvelle approche : le barycentre de Wasserstein.

Definition mathématique du barycentre de Wasserstein

Barycentre de Wasserstein : généralisation du barycentre entre des ensembles

Le barycentre des points de $(A_i)_{1 \leq i \leq n}$ pondéré par les $(m_i)_{1 \leq i \leq n}$ s'écrit

$$\overrightarrow{OG} = \frac{\sum_{i=1}^{n} \overrightarrow{OA_i} m_i}{\sum_{i=1}^{n} m_i}$$

Définir barycentre d'ensembles d'ensembles $\Rightarrow$ définir une distance entre ensembles.

Definition mathématique du barycentre de Wasserstein

Distance choisie : **deuxième distance de Wasserstein**, notée $W_2(X, Y)$ pour deux ensembles de points X et Y de même cardinal, avec :

$$W_2(X, Y) = V(P^*)$$

avec P^* : plan de transport optimal entre X et Y et $V(p^*)$ la valeur de ce transport.

Ainsi pour des ensembles de points de $\mathbb{R}^d$, $(X_i)_{1 \leq i \leq n}$ et des scalaires $(\lambda_i)_{1 \leq i \leq n}$, le barycentre s'écrit :

$$Bar((X_i), (\lambda_i)) = B$$

tel que $B \in \mathbb{R}^d$ minimise $\dfrac{\sum\limits_{i=1}^{n} W_2(B,X_i)\lambda_i}{\sum\limits_{i=1}^{n} \lambda_i}$

Barycentre de Wasserstein en pratique

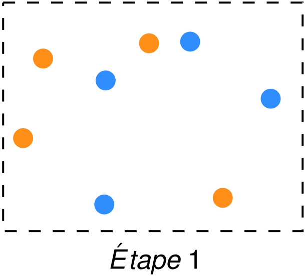
Une telle définition répond exactement aux objectifs :

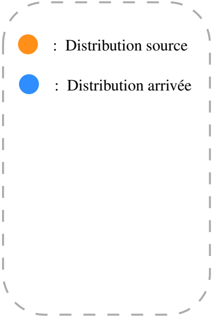
- Barycentre géométrique : points de $\mathbb{R}^2$ représentant les coordonnées des pixels
- Barycentre colorimétrique : points de $\mathbb{R}^3$ représentant les couleurs de chacun des pixels

Mais difficile à calculer : trouver un plan de transport optimal pour calculer la distance très coûteux.

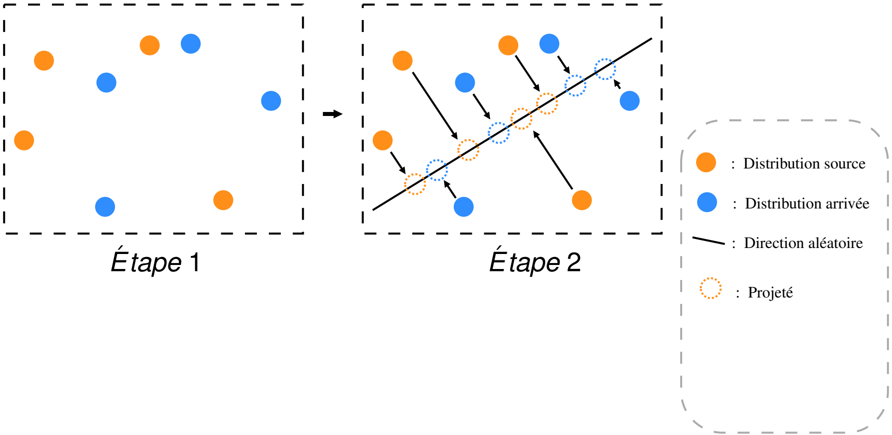
Méthode approchée de calcul du barycentre de Wasserstein : le barycentre de Wasserstein en tranches.

Barycentre de Wasserstein en tranches

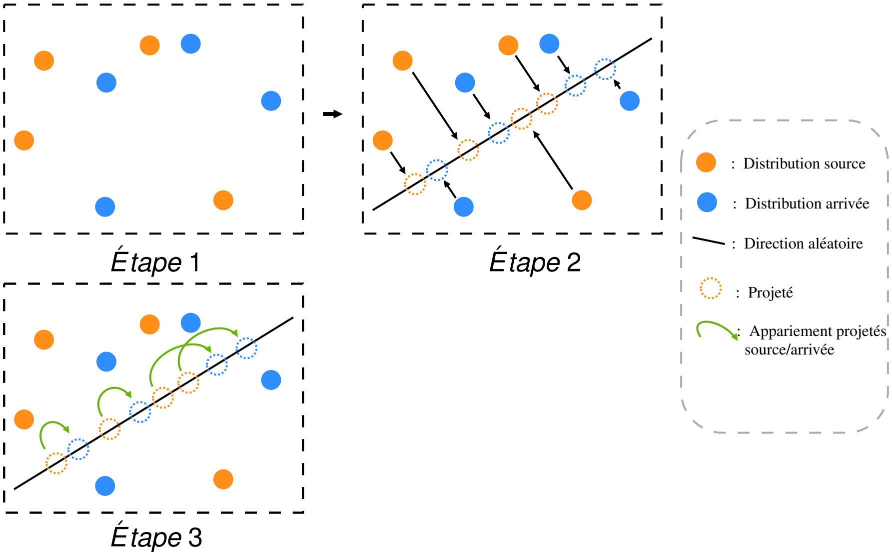




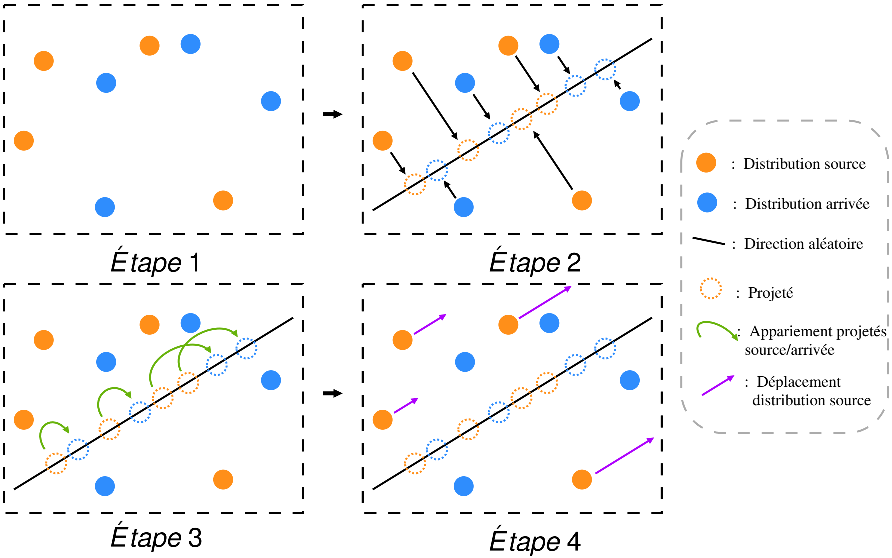
Barycentre de Wasserstein en tranches



Barycentre de Wasserstein en tranches



Barycentre de Wasserstein en tranches



Pseudo-code de l'algorithme du barycentre de Wasserstein en tranches

```
Barycentre de Wassertstein en tranches géométrique (ensemble A de taille n,
ensemble B de taille n, entier nombre_etapes) :

pour tous les entiers de 1 à nombre_etapes :
    advection_vec  ← liste de n vecteurs nuls de ℝ²

    dir_alea  ← vecteur aléatoire de la sphère unité de ℝ²

    projA  ← projections des points de A sur dir_alea
    projB  ← projections des points de B sur dir_alea

    trier projA
    trier projB

    distanceAB  ← distance entre les projections projA triée et projB tiée

    pour chaque point de A :
        advection_vec.(index du point)  ←  dir_alea × distanceAB.(index du point)

    A += advection_vec

retourner A
```

Résultats géométriques

Figure – Interpolation colorimétrique entre deux images

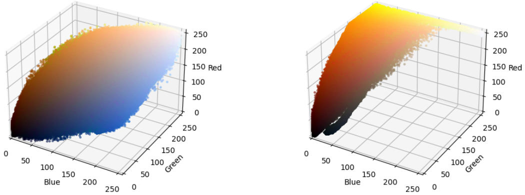


Figure – Représentation des couleurs des deux images dans l'espace RGB

Figure – Interpolation géométrique de Monge à Villani

Figure – Interpolation géométrique de Villani à Monge

Résultats géométriques

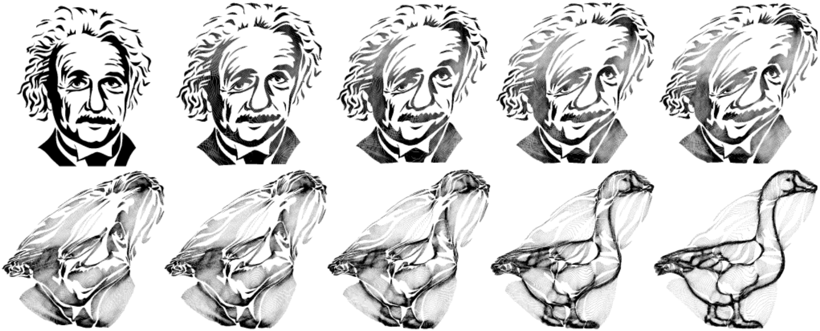


Figure – Interpolation géométrique entre deux images d'une oie et d'Einstein

Conclusion

Objectifs atteints :

- Interpolation géométrique : image conservant les caractéristiques des images sources avec l'algorithme des enchères et le barycentre de Wasserstein
- Interpolation colorimétrique : transfert de style en appliquant les couleurs d'une image à une autre avec le barycentre de Wasserstein

Annexe : Correction de l'algorithme des enchères I

Soit n le nombre d'acheteurs (et donc de propriétés). On appelle X_i les personnes et Y_i les objets à apparier, $\forall i \in [1, n]$. Soit $a_{i,j}$ le gain associé à un déplacement de X_i vers Y_j (par exemple l'opposé de la distance de X_i à Y_j).
À chacun des objets Y_j, on associe le prix p_j. Comme chacun des objets est d'importance égale

$$\forall j \in [1, n], p_j = 0$$

au début de l'algorithme.
On définit le bénéfice associé à un déplacement de X_i vers Y_j par

$$b_{i,j} = a_{i,j} - p_j$$

On appelle **assignement** S un ensemble de couples (i, j), représentant le fait que l'on associe l'objet Y_j à la personne X_i. De plus

- $\forall i \in [1, n]$, il existe au plus un j tel que $(i, j) \in S$
- $\forall j \in [1, n]$, il existe au plus un i tel que $(i, j) \in S$

C'est-à-dire qu'à une personne est associé un seul objet et vice-versa.

De plus, on pose S_i l'ensemble des i tels que $\exists j \in [1, n]$ avec $(i, j) \in S$ et S_j l'ensemble des j tels que $\exists i \in [1, n]$ avec $(i, j) \in S$
On pose ainsi

$$T_S : \left| \begin{array}{rcl} [1,n] & \to & [1,n] \\ i & \mapsto & j \text{ tel que } (i,j) \in S \end{array} \right.$$

On dit que cet assignement est **complet** si $card\ S = n$
On note δ_n l'ensemble des assignements complets.
A S on associe sa **valeur** : $V(S) = \sum_{i \in S_i} a_{i,T_S(i)}$
Ainsi, plus la valeur du transport est élevée, plus il est proche d'être optimal.

On appelle S^* l' **assignement complet optimal**, c'est-à-dire que

$$\begin{cases} card\ S^* = n \\ V(S^*) = \min_{S \in \delta_n} V(S) \end{cases}$$

On note $T^* = T_{S^*}$
On définit maintenant la **condition des écarts complémentaires (EC)**. Un assignement S respecte (EC) si :

$$(EC) : \max_{j \in [1,n]} \{a_{i,j} - p_j\} = a_{i,T_S(i)} - p_{T_S(i)} \ , \forall i \in S_i$$

De même, on définit la **condition des ε écarts complémentaires (ε-EC)**, ($\varepsilon > 0$) :

$$(\varepsilon - EC) : \max_{j \in [1,n]} \{a_{i,j} - p_j\} - \varepsilon \leq a_{i,T_S(i)} - p_{T_S(i)} \ , \forall i \in S_i$$

Pour montrer l'intérêt qu'un assignement respecte ($\varepsilon - EC$), on montre que pour S un assignement complet,

$$(S \text{ réalise } (\varepsilon - EC)) \implies (V(S) + n\varepsilon \geq V(S^*))$$

Preuve :
On pose $T = T_S$. Notons tous d'abord que

$$V(S) = \sum_{i=1}^{n} a_{i,T(i)} \leq \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \max_{j \in [1,n]} \{a_{i,j} - p_j\}$$

Annexe : Correction de l’algorithme des enchères III

En effet,

$$\sum_{i=1}^{n} \max_{j\in[1,n]} \{a_{i,j} - p_j\} \geq \sum_{i=1}^{n} \left[a_{i,T(i)} - p_{T(i)}\right]$$

et directement $\sum_{j=1}^{n} p_j \geq \sum_{j=1}^{n} p_{T(i)}$ car T est bijective. En additionnant les deux inégalités, on obtient l’inégalité annoncée :

$$V(S^*) \leq V(S) \leq \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \max_{j\in[1,n]} \{a_{i,j} - p_j\} := D$$

Comme on a (ε-EC) sur S :

$$(\varepsilon - EC) \quad : \quad \max_{j\in[1,n]} \{a_{i,j} - p_j\} + p_{T(i)} \leq a_{i,T(i)} + \varepsilon \ , \ \ \forall i \in [1,n]$$

En sommant la condition de (ε-EC) pour i variant de 1 à n, il vient :

$$D = \sum_{i=1}^{n} \left[\max_{j\in[1,n]} \{a_{i,j} - p_j\} + p_{T(i)}\right] \leq \sum_{i=1}^{n} \left[a_{i,T(i)} + \varepsilon\right] = V(S) + n\varepsilon$$

Ainsi, avec $V(S^*) \leq D$:

$$V(S^*) \leq V(S) + n\varepsilon$$

Montrons désormais que l’algorithme des enchères vérifie et préserve la condition (ε-EC). ***Preuve*** :

Annexe : Correction de l'algorithme des enchères IV

<u>Initialisation :</u>
Au début de l'algorithme, $S = \emptyset$ et $p_j = 0\ ,\ \forall j \in [1,n]$. Ainsi, l'inégalité de (ε-EC) est vérifiée.
<u>Hérédité :</u>
Supposons (ε-EC) vérifiée à l'itération précédente de l'algorithme. Soit $(p_j)_{j\in[1,n]}$ les prix de l'itération précédente et $(\widetilde{p_j})_{j\in[1,n]}$ les prix de l'itération actuelle. Clairement,

$$\forall j \in [1,n]\ ,\ \widetilde{p_j} \geq p_j$$

Ainsi, on a :

$$(\varepsilon - EC)\ :\ \max_{j\in[1,n]} \{a_{i,j} - p_j\} - \varepsilon \leq a_{i,T(i)} - p_{T(i)}\ \forall i \in [1,n] \in S_i$$

Soit $(i, T(i))$ un élément de S à la fin de l'itération actuelle.

<u>Cas 1 : $(i, T(i))$ vient d'être ajouté à S</u>

Ainsi, le nouveau prix associé à Y_j est

$$\begin{aligned} \widetilde{p_{T(i)}} &= \cancel{p_{T(i)}} + (a_{i,T(i)} - \cancel{p_{T(i)}}) - \omega_i + \varepsilon \\ &= a_{i,T(i)} - \omega_i + \varepsilon \end{aligned}$$

avec $\omega_i = \max_{\substack{j \neq T(i) \\ j \in S_j}} \left(\{a_{i,j} - p_j\}\right)$, le deuxième bénéfice maximum.

Annexe : Correction de l'algorithme des enchères V

Montrons que $(\varepsilon - EC)$ est préservée.
On a

$$\begin{aligned}
a_{i,T(i)} - \widetilde{p_{T(i)}} &= \cancel{a_{i,T(i)}} - \cancel{a_{i,T(i)}} + \omega_{i,T(i)} - \varepsilon \\
&= \max_{\substack{j \neq T(i) \\ j \in S_j}} \left(\{a_{i,j} - p_j\}\right) - \varepsilon \\
&\geq \max_{\substack{j \neq T(i) \\ j \in S_j}} \left(\{a_{i,j} - \widetilde{p_j}\}\right) - \varepsilon
\end{aligned}$$

Car $\forall j \in [1, n]$, $\widetilde{p_j} \geq p_j$.
Ainsi, on a $(\varepsilon - EC)$ $\forall j \in S_j \setminus \{T(i)\}$.
Pour $j = T(i)$ on a :

$$a_{i,T(i)} - \widetilde{p_{T(i)}} \geq a_{i,T(i)} - \widetilde{p_{T(i)}} - \varepsilon$$

Ainsi, on a bien la condition de $(\varepsilon - EC)$ à l'itération suivante.

<u>Cas 2 : $(i, T(i))$ étais déjà dans S à l'itération précédente</u>

Ainsi, $\widetilde{p_{T(i)}} = p_{T(i)}$. Or

$$\max_{j \in [1,n]} \{a_{i,j} - p_j\} - \varepsilon \leq a_{i,T(i)} - p_{T(i)} \; \forall i \in S_i$$

Donc

$$\max_{j \in [1,n]} \{a_{i,j} - \widetilde{p_j}\} - \varepsilon \leq a_{i,T(i)} - \widetilde{p_{T(i)}} \; \forall i \in S_i$$

Ainsi, on a bien la condition de $(\varepsilon - EC)$ à l'itération suivante.
<u>Conclusion :</u>

Annexe : Correction de l'algorithme des enchères VI

La condition ($\varepsilon - EC$) est respectée au début de l'algorithme et préservée à chaque itération, elle est donc vérifiée à la fin de l'algorithme. Ainsi, si S est l'assignement renvoyé par l'algorithme, on a

$$V(S^*) \leq V(S) + n\varepsilon$$

Annexe : Terminaison de l'algorithme des enchères I

Preuve :
On rappelle en premier lieu que l'algorithme se termine dès que l'assignement S qu'il construit est complet et que le cardinal de S ne peut pas baisser (on peut seulement ajouter ou modifier des couples, pas en retirer). Supposons que l'algorithme ne se termine jamais. Ainsi, si l'algorithme ne se termine pas, un objet n'est jamais apparié après un nombre quelconque d'itérations. Appelons Y_{j-} cet objet. Ainsi, aucun acheteur ne fait d'enchères sur Y_{j-}, c'est-à-dire que pour chaque acheteur, à chaque itération, il existe un objet Y_j tel que $b_{i,j} \geq b_{i,j-}$. Donc à chaque itération,

$$p_{j-} = 0$$

Ainsi, au moins un objet Y_{j+} recevra un nombre quelconque d'offres. Comme à chaque offre, p_{j+} augmente au moins de ε, $\varepsilon > 0$ et que p_{j+} reçoit un nombre quelconque d'offres,

$$p_{j+} \longrightarrow +\infty$$

Posons

$$A = \max_{\substack{i \in S_i \\ j,k \in S_j}} \left(\{a_{i,j} - a_{i,k}\} \right)$$

Comme $p_{j+} \longrightarrow +\infty$, il existe une itération à laquelle $p_{j+} > A$. Supposons que dans une itération suivante, il existe une personne X_i désirant faire une offre sur Y_{j+}. Ainsi,

$$b_{i,j+} = \max_{j \in S_j} \left(\{a_{i,j} - p_j\} \right)$$

Annexe : Terminaison de l'algorithme des enchères II

Calculons la différence entre $b_{i,j+}$ et $b_{i,j-}$ avec $b_{i,j}$ le bénéfice associé à (i,j) :

$$\begin{array}{rcl} b_{i,j+} - b_{i,j-} & = & (a_{i,j+} - p_{j+}) - (a_{i,j-} - p_{j-}) \\ & = & (a_{i,j+} - a_{i,j-}) - p_{j+} \\ & < & A - A = 0 \end{array}$$

Ainsi, personne ne fera jamais plus d'offres sur Y_{j+}, car son prix ne peut qu'augmenter et qu'il ne maximise le bénéfice d'aucun X_i avec $p_{j+} > A$. Or, on a supposé que Y_{j+} recevait un nombre quelconque d'offres, on a une contradiction.
On en déduit que l'algorithme se termine.

Annexe : Code python - L'algorithme des enchères I

```python
def max_snd_max(l):
        '''Renvoie le premier et second élément maximal d'une liste, ainsi que
        ↪  l'index de l'élément maximal'''
        mx = max(l[0],l[1])
        mxid = int(mx == l[1])
        sndmx = min(l[0],l[1])
        n = len(l)
        for i in range(2,n):
                if l[i] > mx:
                        sndmx = mx
                        mx = l[i]
                        mxid = i
                elif l[i] > sndmx and mx != l[i]:
                        sndmx = l[i]
        return (mx,sndmx, mxid)


def max_id(l):
        '''Renvoie l'élément maximal d'une liste ainsi que son index'''
        mx = max(l[0],l[1])
        mxid = int(mx == l[1])
        for i in range(len(l)):
                if l[i] > mx:
                        mx = l[i]
                        mxid = i
                        continue
        return (mx, mxid)


def matrice_gain(A, B):
```

Annexe : Code python - L'algorithme des enchères II

```python
    '''A et B sont des distributions de points du plan euclidien de même cardinal'''
    n = len(A)
    # Initialisation de la matrice de gain
    gmat = np.zeros((n,n))
    # Valeurs de coût maximal de chaque colonne
    max_cout_col = np.zeros(n)
    for i in range(n):
        for j in range(n):
            # Coût entre le point i de A et j de B : c'est l'opposé de la distance euclidienne
            gmat[i , j] = -np.sqrt(  (A[i][0] - B[j][0])**2 + (A[i][1] - B[j][1])**2 )
    return gmat


def enchere_algo(A, B, eps):
  '''Trouve un plan de transport optimal entre A et B en utilisant l'algorithme des enchères /
  A et B : ensembles de points de même cardinal ; A : les acheteurs, B les propriétés'''
  # Obtention de la matrice de bénéfice
  gmat = matrice_gain(A, B)
  counter = 0 # Compte le nombre d'objets appariés
  n = len(gmat)
  appariment = np.array([-1 for i in range(n)]) # Plan de transport
  prix = np.zeros(n) # Prix de chaque objet
  enchere = np.zeros((n,n)) # Encheres faites par la ième personne pour le jeme objet
  while counter < n: # On s'arrête quand tous les objets ont été appariés
```

Annexe : Code python - L'algorithme des enchères III

```
      for i in range(n): # Enchères :
        if i in appariment: # Une personne possédant déja un objet n'enchérit pas
          continue
        benefices = gmat[i] - prix
        m, sm, mid = max_snd_max(benefices) # Obtention des valeurs max et leurs index
        # L'agent i enchérit sur l'objet mid
        enchere[mid, i] = prix[mid] + (m - sm) + eps
      for j in range(n): # Assignement :
        mb, mbid = max_id(enchere[j]) # Obtention de l'enchère maximale
        if mb == 0: # On ne fait rien si personne n'a enchéri pour l'objet
          continue
        # On augmente le prix de l'objet j et on le donne au plus offrant
        prix[j] = mb
        if appariment[j] == -1:
          counter += 1
        appariment[j] = mbid
      # On remet les enchères à 0
      enchere = np.zeros((n,n))
    return appariment


def lerp(a,b,t):
        '''Renvoie le point c = a + (b-a)t'''
        return [b[0]*t + a[0]*(1-t) , b[1] * t + a[1] * (1-t)]


def melange_dist(A, B, app, t):
```

Annexe : Code python - L'algorithme des enchères IV

```
    '''Renvoie l' "ensemble moyen" de A et B /
    A et B les ensembles d'origine et d'arrivée app le plan de transport de A vers B t le coefficient d'interpolation entre A et B'''
    n = len(app)
    res = []
    for i in range(n):
        m = lerp(B[i], A[app[i]], t)
        res += [m]
    return res


def plot_transport(A, B, app=[], title=""):
    '''Affiche une représentation d'un assignement'''
    n = max(len(A),len(B))
    for i in range(len(app)):
        # if len(app) < i:
            X = [  B[i][0] , A[app[i]][0]  ]
            Y = [  B[i][1] , A[app[i]][1]  ]
            plt.plot(X,Y, 'g')

    for i in range(n):
        if i == 0:
            plt.plot(A[i][0], A[i][1], 'xr', label="acheteurs")
            plt.plot(B[i][0], B[i][1], '+b', label="objets")
        else:
            if i < len(A):
                plt.plot(A[i][0], A[i][1], 'xr')
```

Annexe : Code python - L'algorithme des enchères V

```python
            if i < len(B):
                plt.plot(B[i][0], B[i][1], '+b')
    plt.legend(loc=0)
    plt.title("Plan de transport optimal avec l'algorithme des enchères")


def equalize(A,B):
    '''Duplique des points au hasard de A et de B pour obtenir deux ensembles de même cardinal'''
    res = []
    if len(B) > len(A):
        indexs = [random.randint(0, len(A)-1) for k in range(len(B) - len(A))]
        for i in indexs:
            res += [[A[i][0], A[i][1]]]
        A = np.vstack((A, res))
    if len(A) > len(B):
        indexs = [random.randint(0, len(B)-1) for k in range(len(A) - len(B))]
        for i in indexs:
            res += [[B[i][0], B[i][1]]]
        B =  np.vstack((B, res))
    return A,B


def dist_to_img(A, L):
    '''Renvoie une image en noir et blanc à partir d'un ensemble de points'''
    tab = np.full((L,L,3), 255 )
    for p in A:
```

Annexe : Code python - L'algorithme des enchères VI

```
                tab[L - int(p[1]) - 1][int(p[0])] = [0, 0, 0]
        return tab


def create_anim(A, B, app, L, k=10, title="plt", reverse=False):
        '''Crée une séquence de k images décrivant le déplacement optimal de A vers
        ↪  B'''
        for i in range(0, k+1):
                t = i/k if i != 0 else 0
                im = dist_to_img(melange_dist(A, B, app, t), len(A))
                cv2.imwrite(f'{title}_{i}.png', im)
                if reverse:
                        cv2.imwrite(f'{title}_{2*k-i}.png', im)
```

Annexe : Code python - Complexité de l'algorithme des enchères I

```
def random_dist(n, L):
        ''' Crée un ensemble de points de R^2 disposés dans une grille de taille L selon une probabilité uniforme'''
        res = []
        for i in range(n):
                res += [(random.randint(0,L), random.randint(0,L))]
        return res


###

Création des données

###

def generate_complexity()
        ''' Écrit dans un fichier le temps d'exécution et le nombre de passages dans la boucle tant que de l'algorithme des enchères pour des ensembles de cardinal différents générés selon une probabilité uniforme'''
        t = 10
        values = [2, 5, 10, 50, 100, 200, 300, 400, 700, 1000,1500, 2000]
        DATA = [[] for i in values]

        for i in range(t):
                for n in values:
                        A,B = random_dist(n,n*2),  random_dist(n,n*2)
                        k = time.time()
```

Annexe : Code python - Complexité de l’algorithme des enchères II

```
            nb = enchere_algo(A, B, 2, debug=True)
            w = time.time() - k
            f = open("comp2.txt", 'a')
            f.write(f"\n{n}:{nb}")
            f.write(f"\n{n}-{w}")
            f.close()


###

Analyse de la complexité

###

avr = []
minerr = []
maxerr = []
values = [2, 5, 10, 50, 100, 200, 300, 400, 700, 1000,1500, 2000]
DATA = [[] for i in values]
f = open("comp2.txt", 'r')

lines = f.readlines()
f.close()

for  line in lines:
    if ':' in line:
            n,nb = int(line.split(":")[0]) , int(line.split(":")[1][:-1])
```

Annexe : Code python - Complexité de l’algorithme des enchères III

```python
    try:
        DATA[values.index(n)] += [nb]
    except:
        pass

for v in values:
    avr += [np.average(DATA[values.index(v)])]
    minerr += [ avr[-1] - min(DATA[values.index(v)]) ]
    maxerr += [ max(DATA[values.index(v)]) - avr[-1] ]



def plot_complexity_raw():
    ''' Affiche à l'aide de matplotlib le nombre de passages dans la boucle
    tant que avec des barres d'erreurs'''

    plt.title("Distribution empririque du nombre de passages dans la boucle",
    ↪  fontsize=13)
    plt.scatter(values, avr)
    plt.xlabel("n")
    plt.ylabel("Nombre d'itérations")
    plt.errorbar(values, avr, yerr=(minerr, maxerr))

    plt.show()
```

Annexe : Code python - Complexité de l'algorithme des enchères IV

```python

def plot_complexity_log():
        ''' Réalise une régression linéaire et affiche à l'aide de matplotlib le
        nombre de passages logarithmique dans la boucle tant que de l'algorithme des
        ↪   enchères'''
        k = 8
        plt.title("Distribution empririque du log du nombre de passages dans la
        ↪   boucle", fontsize=12)
        plt.scatter(np.log(values[k:]), np.log(avr[k:]), label="Log de la
        ↪   distribution empririque du nombre de passages dans la boucle")
        a,b = np.polyfit( np.log(values[k:]), np.log(avr[k:]), 1)
        plt.plot(np.log(values[k:]), np.log(values[k:]) * a + b, label=f"Regression
        ↪   lineaire : {round(a,2)}*x - {abs(round(b,2))}", color="red")
        plt.xlabel("log(n)")
        plt.ylabel("log(Nombre d'itérations)")
        plt.legend(loc="lower left")

        plt.show()


###

Analyse du temps d'exécution

###


```

Annexe : Code python - Complexité de l'algorithme des enchères V

```python
avr = []
minerr = []
maxerr = []
values = [2, 5, 10, 50, 100, 200, 300, 400, 700, 1000,1500, 2000]
DATA = [[] for i in values]
f = open("comp2.txt", 'r')

lines = f.readlines()
f.close()

for  line in lines:
        if '-' in line:
                n,nb = float(line.split("-")[0]) , float(line.split("-")[1][:-1])
        try:
                DATA[values.index(n)] += [nb]
        except:
                pass

for v in values:
        avr += [np.average(DATA[values.index(v)])]
        minerr += [ avr[-1] - min(DATA[values.index(v)]) ]
        maxerr += [ max(DATA[values.index(v)]) - avr[-1] ]


def plot_time_raw():
        ''' Affiche à l'aide de matplotlib le nombre de passages dans la boucle
        tant que avec des barres d'erreurs'''
```

Annexe : Code python - Complexité de l'algorithme des enchères VI

```python

        plt.title("Distribution empirique de la complexité temporelle de l'algorithme
        ↪ des enchères", fontsize=10)
        plt.scatter(values, avr)
        plt.errorbar(values, avr, yerr=(minerr, maxerr))
        plt.xlabel("n")
        plt.ylabel("Temps en seconde")

        plt.show()


def plot_time_log():
        ''' Réalise une régression linéaire et affiche à l'aide de matplotlib le
        temps d'exécution de l'algorithme des enchères'''
        k = 8
        plt.title("Distribution empirique du log de la complexité temporelle de
        ↪ l'algorithme des enchères", fontsize=10)
        plt.scatter(np.log(values[k:]), np.log(avr[k:]), label="Log de la
        ↪ distribution empirique du nombre de passages dans la boucle")
        a,b = np.polyfit( np.log(values[k:]), np.log(avr[k:]), 1)
        print(a)
        plt.plot(np.log(values[k:]), np.log(values[k:]) * a + b, label=f"Régression
        ↪ linéaire : {round(a,2)}*x - {abs(round(b,2))}" , color="red")
        plt.legend(loc="lower left")
        plt.xlabel("n")
        plt.ylabel("Temps en seconde")
```

Annexe : Code python - Complexité de l'algorithme des enchères VII

```
        plt.show()
```

Annexe : Code python - Barycentre de Wasserstein en tranches I

```python
def rnd_direction(c):
  ''' Renvoie un vecteur aléatoire de la sphere unité de R^c selon une loi normale'''
  d = np.random.normal(size=c)
  return d / np.linalg.norm(d)


def wasserstein_barycenter(src, trg, steps=10 , batch=3):
  ''' Fais converger les points de src vers les points de trg pour trouver une approximation du barycentre de Wasserstein /
  src et trg sont des ensembles de points de R^c de même cardinal'''
  h,w,c = src.shape
  nsrc = src.copy()

  for s in range(steps):
    # On crée le vecteur d'advection
    adv = np.zeros((h*w,c))

    for b in range(batch):
      # On choisit la direction de R^d sur laquelle opérer
      direction = rnd_direction(c)
      # On projette les points des deux ensembles sur la direction
      projs = np.sum(nsrc * direction , axis = -1).reshape(h*w)
      projt = np.sum(trg * direction , axis = -1).reshape(h*w)
      # On trie les points selon leurs projections
      ids = np.argsort(projs)
      idt = np.argsort(projt)
      # On calcule les déplacements
```

Annexe : Code python - Barycentre de Wasserstein en tranches II

```
        dis = projt[idt] - projs[ids]
        # On rajoute au vecteur d'advection les déplacements de chaque
        #  points selon la direction choisie précédemment
        for i in range(c):
          adv[ids,i] += dis * direction[i]

      # On déplace les points de l'ensemble
      # source selon le vecteur d'advection
      nsrc += adv.reshape((h,w,c)) / batch

    return nsrc


def vec_color_to_vec_pos(vec, rows, cols, targcol):
    ''' Convertit une image en couleurs en points de N^2 en retournant les coordonnées
    ↪  de tous les pixels de la couleur targcol'''
    length , d = vec.shape
    nvec = []
    for k in range(length):
      if tuple(vec[k]) == tuple(targcol):
        continue
      nvec.append((float(k % rows), float(k // rows)))
    return np.array(nvec)


def vec_pos_to_tab_color(vec, rows, cols, targcol):
    ''' Convertit un ensemble de points de N^2 en image en couleur en retournant une
    ↪  image dont chaque pixel aynt ses coordonnées dans vec a été coloré en la
    ↪  couleur targcol'''
```

Annexe : Code python - Barycentre de Wasserstein en tranches III

```python
    length , d = vec.shape
    tab = np.full((rows, cols, 3), 255)
    for k in range(length):
      w,h = int(vec[k][0]), int(vec[k][1])
      if w >= 0 and w < cols and h > 0 and h < cols:
        tab[int(vec[k][1]), int(vec[k][0])] = targcol
    return tab


  def export_points(src, dim, title):
    ''' Exporte en png un ensemble de points de N^2 représentant les coordonnées des
    ↪  pixels colorés en noir'''
    t = vec_pos_to_tab_color(src, dim[0], dim[1], [0, 0, 0])
    cv2.imwrite(title, t)
```

- Dimitri P. Bertsekas, *A distributed algorithm for the assignement problem*, `https://www.mit.edu/~dimitrib/Orig_Auction.pdf`, 1979.
- ———, *The auction algorithm for assignment and other network flow problems*, `https://dspace.mit.edu/bitstream/handle/1721.1/3154/P-1908-20783037.pdf?sequence=1`, 1989.
- Julie Delon, Gabriel Peyré, Julien Rabin, and Marc Bernot, *Wasserstein barycenter and its application to texture mixing*, `https://hal.archives-ouvertes.fr/hal-00476064/document`, 2011.

- Brittany Hamfeldt, *Optimal transport - discrete optimal transport*, `https://www.youtube.com/watch?v=7MZoLsXXyvA&t`, 2019.
- ———, *Optimal transport - the auction algorithm*, `https://www.youtube.com/watch?v=cONEfprs5l4`, 2019.
- Gaspard Monge, *Mémoire sur la théorie des déblais et des remblais*, 1784.
- Gabriel Peyré and Marco Cuturi, *Computational optimal transport*, `https://optimaltransport.github.io/pdf/ComputationalOT.pdf`, 2018.