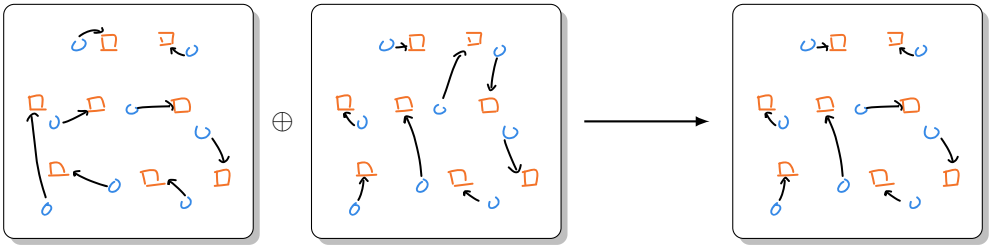
# BSPOT – cas non uniforme

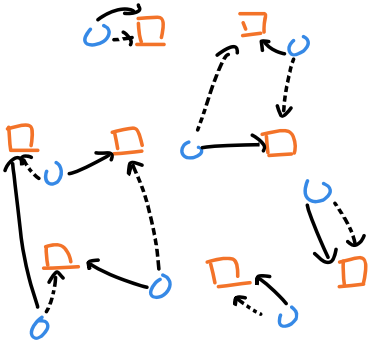
Baptiste Genest, Nicolas Bonneel, David Cœurjolly, Vincent Nivoliers

# Fusion de BSP

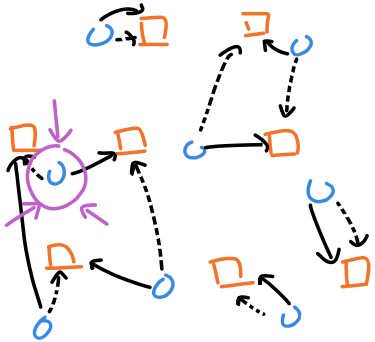


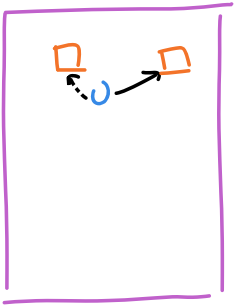
- améliorer le coût de transport
- masquer les artéfacts de séparation linéaire
- efficacité

# Échanges locaux

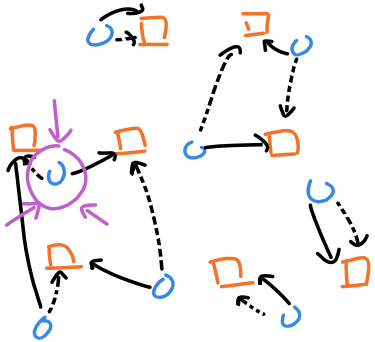


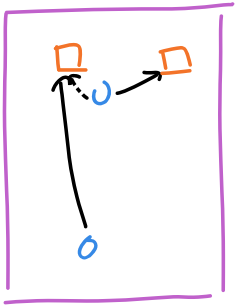
# Échanges locaux



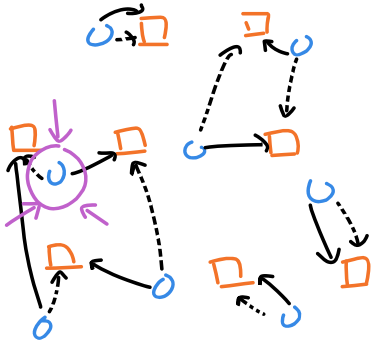


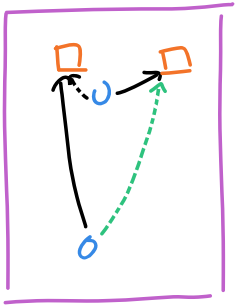
# Échanges locaux





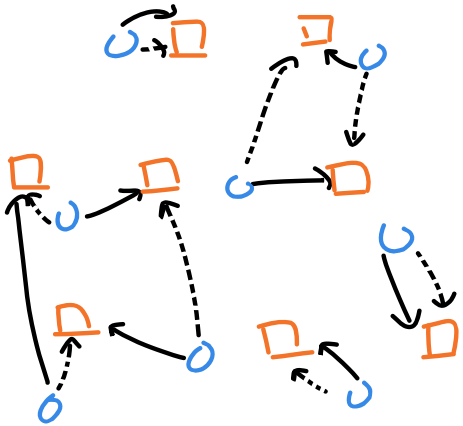
# Échanges locaux



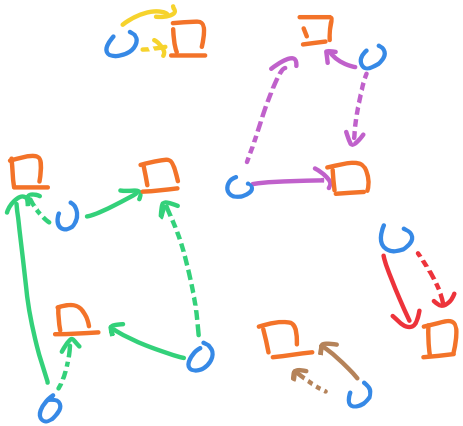


- simple et efficace
- coût décroissant
- arête inconnue

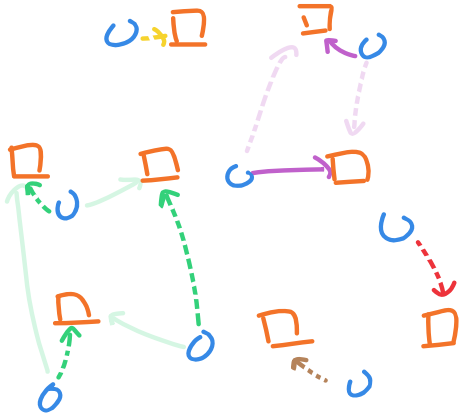
# Composantes connexes



# Composantes connexes

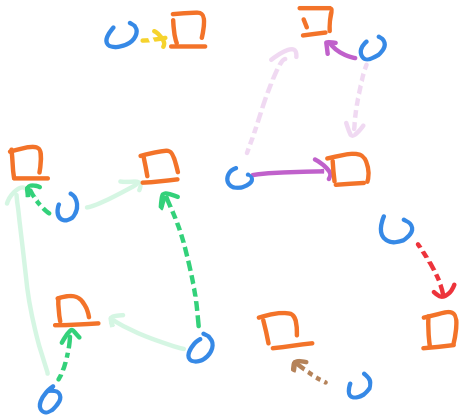


# Composantes connexes



- composantes par parcours en profondeur
- coût décroissant
- meilleur qu'échanges locaux
- conserve les artéfacts de séparation

# Composantes connexes

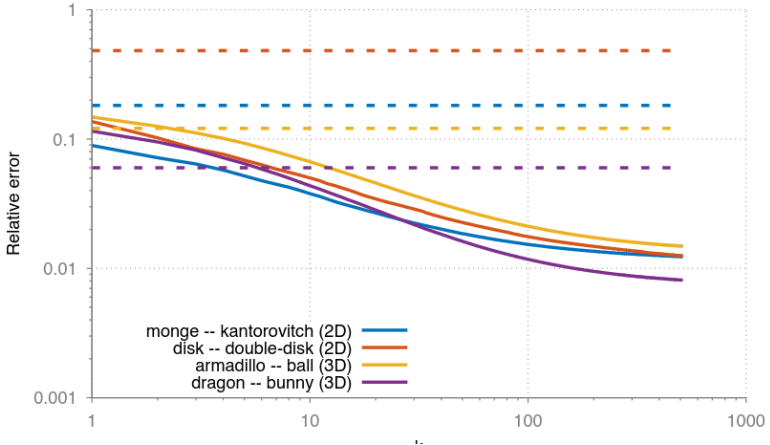


- composantes par parcours en profondeur
- coût décroissant
- meilleur qu'échanges locaux
- conserve les artéfacts de séparation

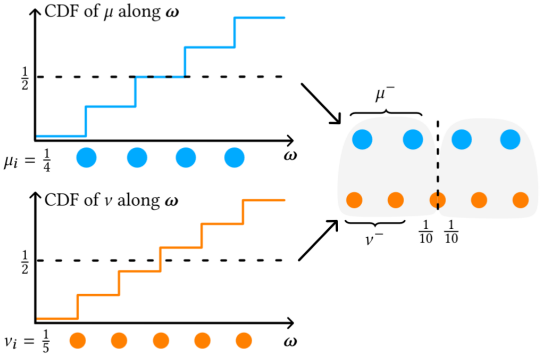
## Au final

- retenter des échanges après
- parallèle sur les composantes

# Impact des fusions



# BSP pour le cas non uniforme



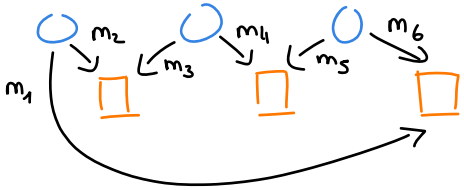

- plus une bijection
- les composantes sont grosses
- moins de parallélisme
- échanges locaux possibles

5.2 BSP merging

While we can merge couplings produced by Alg. 7 in a way similar to the previous settings, it is algorithmically much more involved and the complexity of the merge depends on the degree of each vertex, which leads to significantly worse run times. These algorithms are implemented in the code we provide in supplementary materials.

ACM Trans. Graph., Vol. 44, No. 6, Article . Publication date: December 2025.

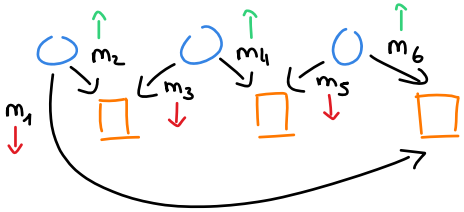
# Cas des cycles




Coût du transport

$$C(\{m_i\}) = \sum_i m_i c_i$$

# Cas des cycles



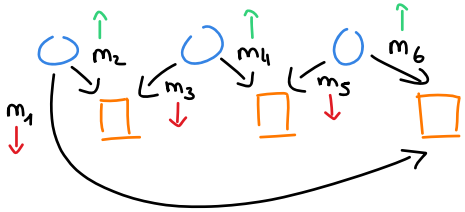

## Coût du transport

$$C(\{m_i\}) = \sum_i m_i c_i$$

## Ajustement $\varepsilon$

$$\sum_i (m_{2i} + \varepsilon) c_{2i} + \sum_i (m_{2i+1} - \varepsilon) c_{2i+1}$$

# Cas des cycles



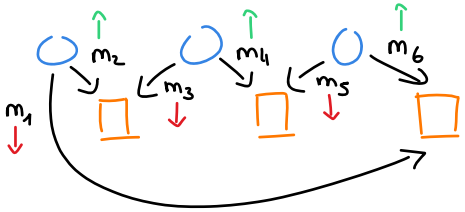

Coût du transport

$$C(\{m_i\}) = \sum_i m_i c_i$$

Ajustement $\varepsilon$

$$\sum_i (m_{2i} + \varepsilon) c_{2i} + \sum_i (m_{2i+1} - \varepsilon) c_{2i+1}$$
$$= C + \varepsilon \underbrace{\sum_i (c_{2i} - c_{2i+1})}_{\text{regarder le signe !}}$$

# Cas des cycles




On peut supprimer une arête via un ajustement en améliorant le coût. L'optimal n'a pas de cycle.

## Coût du transport

$$C(\{m_i\}) = \sum_i m_i c_i$$

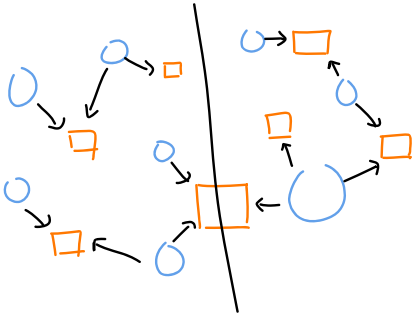
## Ajustement $\varepsilon$

$$\sum_i (m_{2i} + \varepsilon) c_{2i} + \sum_i (m_{2i+1} - \varepsilon) c_{2i+1}$$
$$= C + \varepsilon \underbrace{\sum_i (c_{2i} - c_{2i+1})}_{\text{regarder le signe !}}$$

# Un BSP n'a pas de cycle

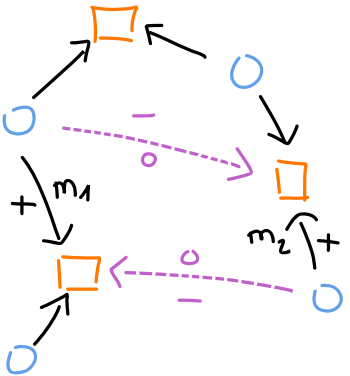
Cas d'arrêt

Cas général

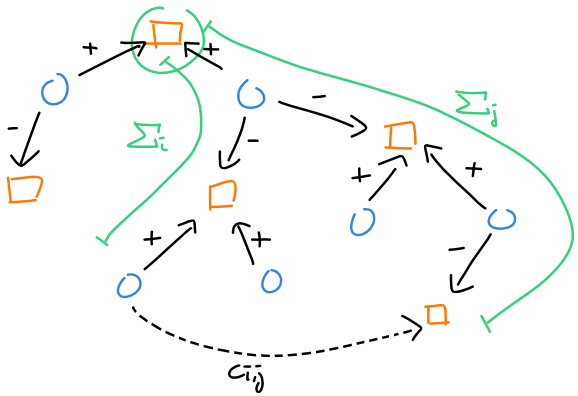


# Échange local non uniforme

- choix multiple de voisins
- crée deux boucles maximum
- n'en supprime qu'une de base
- augmente le nombre d'arêtes de 1



# Potentiel, network simplex




- enraciner l'arbre
- sommes alternées depuis la racine
- pour chaque paire $i, j$ comparer

$$\Sigma_i - c_{i,j} - \Sigma_j$$

- annulation de racine $\rightarrow$ ancêtre

Problème : arbre dynamique

Peyré et Cuturi : Computational optimal transport (2018)

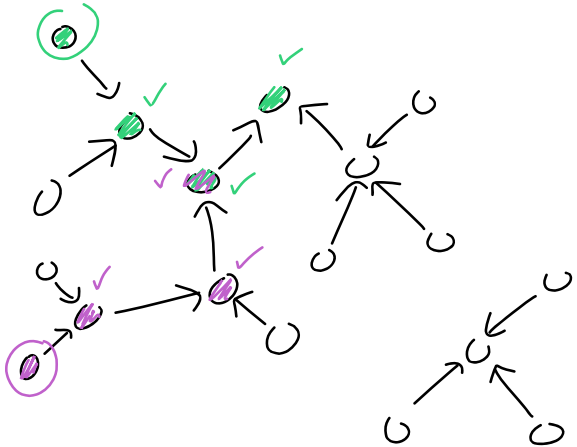
# Gestion de forêts

## Union Find

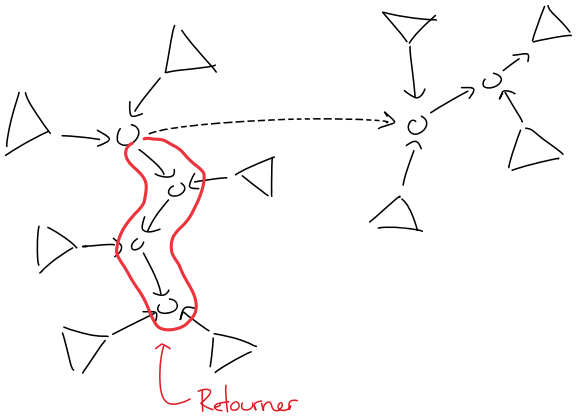
- un parent par nœud
- si ancêtre commun : boucle

## Chemins dans l'arbre

- pas de compression de chemin
- find par parcours en largeur
- union plus compliquée
- on peut couper une arête



# Joindre deux arbres




## Ajouter une arête

- trivial si un sommet est racine

## Changer de racine

- trouver le chemin vers la racine
- retourner ses arêtes
- maintenir potentiels compliqué

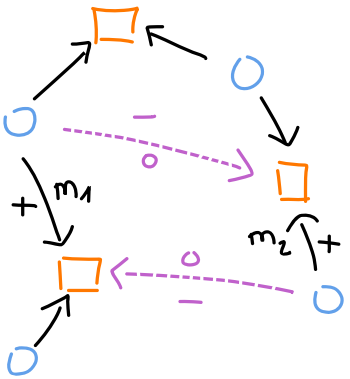
# Performance : échanges locaux

## Base

- pour chaque BSP, échanges locaux sur ses arêtes
- 100 arbres produits en 1.7s, fusion en 171s
- coût initial des BSP $\approx 0.08$
- coût du plan final : 0.0559 pour 209625 arêtes

## Retrait des cycles

- réinsérer itérativement les arêtes de l'arbre final
- temps supplémentaire : 0.17s
- coût du plan final : 0.0557 pour 75485 arêtes



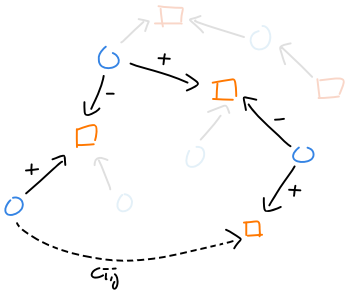
# Performance : insertion séquencielle

## Algo

- pour chaque BSP, proposer ses arêtes
- pour chaque arête, chercher le cycle
- supprimer le cycle

## Résultat

- 100 arbres produits en 1.7s, fusion en 90s
- coût du plan final : 0.0596 (pire)



# Performance : barycentre

## Algo

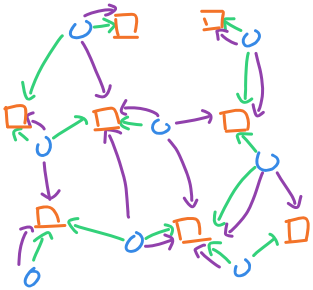
- fusionner les plans via un barycentre
- pour chaque arête, chercher le cycle
- supprimer le cycle

## Base

- 100 arbres produits en 1.7s, fusion en 12s
- coût du plan final : 0.551 (mieux)

## Échanges locaux

- 1.6s supplémentaires
- coût du plan final : 0.549



# Performance : barycentre

## Algo

- fusionner les plans via un barycentre
- pour chaque arête, chercher le cycle
- supprimer le cycle

ordre important !

## Base

- 100 arbres produits en 1.7s, fusion en 12s
- coût du plan final : 0.551 (mieux)

## Échanges locaux

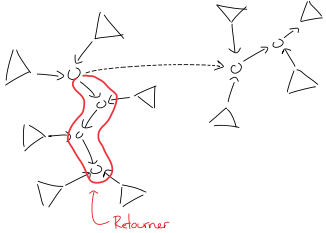
- 1.6s supplémentaires
- coût du plan final : 0.549

- pas de tri : rapide
- poids : lent
- sommet puis poids

# Perspectives

## Parallélisme

- travailler indépendamment sur des sous-ensembles




## Forêts

- chercher les boucles plus vite : skips ?
- bornes sur le potentiel