Algorithmique avancées Python

Projet

Simulation de recherche publicitaire sur graphe pondéré

A3MSI Groupe 1

Amaury Kamerer | Vincent Angoulvant | Gauthier Pagezy

Professeur encadrant :

Bilel Ben Romdhanne

Une image contenant symbole, capture d’écran, Police, Graphique

Le contenu généré par l’IA peut être incorrect.

2025-2026

Sommaire

[1 INTRODUCTION 4](#_Toc211505682)

[1.1 Contexte et problématique 4](#_Toc211505683)

[1.2 Structure des données 4](#_Toc211505684)

[2 CONSTRUCTION DU GRAPHE 4](#_Toc211505685)

[2.1 Algorithme K-NN 4](#_Toc211505686)

[2.2 Justification du graphe 4](#_Toc211505687)

[3 DISTANCE PONDÉRÉE 5](#_Toc211505688)

[3.1 Formule 5](#_Toc211505689)

[3.2 Impact sur la recherche 5](#_Toc211505690)

[4 ALGORITHMES DE RECHERCHE 5](#_Toc211505691)

[4.1 Stratégie NAIVE (baseline) 5](#_Toc211505692)

[4.2 Stratégie BFS (Breadth-First Search) 5](#_Toc211505693)

[4.3 Stratégie DIJKSTRA (priorité sur distance) 6](#_Toc211505694)

[4.4 Stratégie HYBRID (heuristique) 6](#_Toc211505695)

[5 COMPARAISON EXPÉRIMENTALE 6](#_Toc211505696)

[5.1 Métriques d'évaluation 6](#_Toc211505697)

[5.2 Hypothèses de test 6](#_Toc211505698)

[5.3 Résultats attendus 7](#_Toc211505699)

[6 ANALYSE ET DISCUSSION 7](#_Toc211505700)

[6.1 Trade-offs identifiés 7](#_Toc211505701)

[6.2 Cas d'usage recommandés 7](#_Toc211505702)

[6.3 Limites et améliorations 8](#_Toc211505703)

[7 CONCLUSION 8](#_Toc211505704)

[7.1 Synthèse 8](#_Toc211505705)

[7.2 Perspectives 8](#_Toc211505706)

# INTRODUCTION

## Contexte et problématique

* Objectif : Rechercher efficacement des nœuds (utilisateurs) dans un rayon pondéré D autour d'un point de départ, en utilisant un vecteur de pondération Y\_vector (préférences publicitaires)
* Enjeu : Optimiser la recherche dans un graphe de grande taille (scalabilité)
* Application : Ciblage publicitaire personnalisé

## Structure des données

* Nodes : Utilisateurs avec features (caractéristiques comportementales)
* Ads : Annonces avec A\_\_vector (caractéristiques), Y\_vector (pondérations des features) et D (rayon de recherche)
* Graphe : Construction K-NN (K plus proches voisins) pour capturer la similarité entre utilisateurs

# CONSTRUCTION DU GRAPHE

## Algorithme K-NN

* Principe : Connecter chaque nœud à ses K voisins les plus proches (distance euclidienne non pondérée)
* Connecter les ads aux nœuds dont la distance est inférieur à D.
* Implémentation : sklearn.neighbors.NearestNeighbors
* Complexité :
* Construction : O(N log N) avec KD-Tree
* Espace : O(N \* K) arêtes
* Paramètre K : Trade-off densité/performance (K=10 par défaut)

## Justification du graphe

* Réduction d'espace de recherche : Exploiter la structure locale (voisinage)
* Hypothèse : Les nœuds similaires (proches en features) sont connectés
* Avantage : Éviter la recherche exhaustive (O(N) → O(E) ou mieux)

# DISTANCE PONDÉRÉE

## Formule

* + Y\_vector : Pondérations des features (importance pour l'ad)
  + Exemple : Si Y = [1, 0, 0], seule la feature 0 compte

## Impact sur la recherche

* Personnalisation : Chaque ad a sa propre métrique de distance
* Déformation de l'espace : Les arêtes K-NN (non pondérées) ne garantissent plus la proximité pondérée
* Conséquence : Nécessité de comparer plusieurs algorithmes de recherche

# ALGORITHMES DE RECHERCHE

## Stratégie NAIVE (baseline)

* Principe : Parcourir TOUS les nœuds réguliers du graphe
* Complexité : O(N) (N = nombre de nœuds)
* Avantages : Exhaustif, résultats garantis
* Inconvénients : Inefficace, n'exploite pas le graphe

## Stratégie BFS (Breadth-First Search)

* Principe : Explorer le graphe par couches depuis le nœud de départ
* Complexité : O(E) (E = arêtes explorées, E << N si rayon D petit)
* Avantages : Exploite la structure du graphe, plus rapide si résultats concentrés localement
* Inconvénients : Peut manquer des nœuds proches en distance pondérée mais éloignés en graphe (à cause de Y\_vector)

## Stratégie DIJKSTRA (priorité sur distance)

* Principe : Explorer le graphe avec file de priorité (nœuds les plus proches d'abord)
* Complexité : O(E log V) (V = nœuds visités)
* Avantages : Priorise les nœuds proches, coupe l'exploration tôt
* Inconvénients : Overhead de la heap, peut explorer inutilement si graphe mal structuré

## Stratégie HYBRID (heuristique)

* Principe : BFS pour explorer localement, puis élargir progressivement
* Complexité : Variable (O(E) à O(N) selon cas)
* Avantages : Adaptatif, équilibre vitesse/exhaustivité
* Inconvénients : Paramétrage délicat (seuils, profondeur)

# COMPARAISON EXPÉRIMENTALE

## Métriques d'évaluation

* Temps d'exécution : Mesurer le temps moyen par recherche
* Nœuds vérifiés : Compteur de calculs de distance
* Précision : % de nœuds trouvés vs naive (recall)
* Scalabilité : Comportement avec N croissant (100, 1000, 10000 nœuds)

## Hypothèses de test

* Cas favorable BFS : D petit, graphe bien connecté (K élevé), Y\_vector uniforme
* Cas défavorable BFS : D grand, Y\_vector non uniforme (déformation forte)
* Cas favorable Dijkstra : D moyen, résultats dispersés

## Résultats attendus

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithme | Temps | Nœuds vérifiés | Précision |
| Naive | 100% | 100% (N) | 100% |
| BFS | 20-50% | 10-30% (E) | 80-95% |
| Dijkstra | 30-60% | 15-40% | 90-98% |
| Hybrid | 25-55% | 12-35% | 85-98% |

(Valeurs indicatives, à mesurer expérimentalement)

# ANALYSE ET DISCUSSION

## Trade-offs identifiés

* Vitesse vs Précision : BFS rapide mais incomplet, Naive lent mais exact
* Impact de Y\_vector : Forte pondération → graphe K-NN moins pertinent → BFS moins efficace
* Impact de K : K élevé → graphe dense → BFS explore plus, mais plus de voisins pertinents

## Cas d'usage recommandés

* BFS : Recherches locales (D < 10% du diamètre du graphe), Y\_vector uniforme
* Dijkstra : Recherches moyennes, Y\_vector déformant
* Hybrid : Usage général, adaptatif entre BFS et Dijkstra selon taille du graph
* Naive : Benchmark ou petits graphes (N < 1000)

## Limites et améliorations

* Limites :
* Graphe K-NN fixe (pas de réévaluation avec Y\_vector)
* Pas de prétraitement des ads (index spatial)
* Améliorations possibles :
* Construire un graphe pondéré par ad (coûteux)
* Utiliser des structures spatiales (R-Tree, VP-Tree)
* Algorithme A\* avec heuristique (distance euclidienne comme borne inférieure)

# CONCLUSION

## Synthèse

* Les algorithmes par graphe (BFS, Dijkstra) sont plus rapides que Naive dans la majorité des cas
* Le choix dépend du contexte (taille du graphe, rayon D, Y\_vector)
* Hybrid offre un bon compromis pour un usage général

## Perspectives

* Tester sur des graphes réels (réseaux sociaux, recommandations)
* Étendre à des métriques non euclidiennes (cosinus, Manhattan)
* Parallélisation des recherches (multi-threading)