# Estimation de la Densité 2D par Noyau dans l'Espace Urbain Piéton

Mortada Bouzaffour, Antoine Halter-Mingaud
*ENSG Géomatique, 2ème année du cycle ingénieur*

Janvier 2025

Pour avoir accès au code complet du projet et au rapport détaillé, se rendre sur : Code GitHub

## Résumé

L'estimation de densité par noyau (KDE) est une méthode statistique couramment utilisée pour estimer la densité de probabilité d'une variable aléatoire sur un espace donné. Cette étude explore l'application de la KDE dans un environnement urbain contraint, en prenant en compte les distances perçues par des piétons, qui ne sont pas euclidiennes. Nous comparons trois algorithmes de recherche de chemin : Dijkstra [3], A* [5] et Jump Point Search (JPS) [6], pour adapter la KDE aux contraintes urbaines. Les résultats montrent que l'algorithme JPS puis A*, offrent le meilleur temps de calcul et qu'en pratique, l'algorithme Djikstra n'est pas efficace dans ce type d'environnement.

## Abstract

Kernel density estimation (KDE) is a commonly used statistical method for estimating the probability density of a random variable over a given space. This study explores the application of KDE in a constrained urban environment, taking into account perceived pedestrian distances, which are non-Euclidean. We compare three pathfinding algorithms: Dijkstra [3], A* [5], and Jump Point Search (JPS) [6], to adapt KDE to urban constraints. The results show that JPS, followed by A*, offers the best computation time. In practice, the Dijkstra algorithm proves inefficient in this type of environment.

## 1 Introduction

La recherche de chemin optimal dans des environnements complexes est un sujet fondamental en géomatique et en intelligence artificielle. Le pathfinding 2D [2] représente l'un des problèmes les plus étudiés dans ces domaines, en particulier lorsqu'il s'agit de modéliser des environnements urbains piétonniers. Les algorithmes classiques de recherche de chemin, tels que Dijkstra et A*, utilisent généralement des mesures de distance euclidienne. Toutefois, ces méthodes s'avèrent limitées dans des environnements urbains où les déplacements sont souvent restreints par des obstacles tels que des bâtiments, des murs ou des zones interdites, et où les distances ne sont pas nécessairement euclidiennes. Dans ce contexte, la mise en œuvre d'un algorithme de recherche de chemin adapté à ces spécificités devient essentielle pour améliorer la modélisation de la mobilité piétonne.

## 1.1 Contexte

Les environnements urbains se distinguent par leur complexité géométrique, où les piétons se déplacent le long de rues, de trottoirs et d'autres infrastructures, souvent dans des espaces très contraints. Ces contraintes imposent une réflexion sur la manière de mesurer la distance entre deux points, en particulier en prenant en compte les obstacles physiques et les caractéristiques spécifiques des chemins urbains. En outre, la modélisation de ces déplacements peut avoir des applications variées, allant de la simulation de la mobilité dans les systèmes de transport urbains à la planification de l'aménagement du territoire.

## 1.2 Objectifs

Les objectifs de cette étude sont les suivants :

1. Faire un état de l'art sur le pathfinding 2D
2. Comparer plusieurs algorithmes de recherche de chemin pour identifier les plus efficaces dans ce contexte.
3. Sélectionner 3 algorithmes pour les étudier et faire un benchmarking pour en choisir le meilleur .
4. Adapter la KDE pour intégrer des distances "marchables" prenant en compte les obstacles urbains.

# 2 Estimation par Densité Noyau (KDE)

L'estimation par densité noyau, ou *Kernel Density Estimation* (KDE), est une méthode non paramétrique permettant d'estimer la densité de probabilité d'une variable aléatoire à partir d'un échantillon de données. Elle repose sur l'hypothèse que la densité peut être approximée par une somme pondérée de fonctions *noyau*, chacune centrée sur une donnée de l'échantillon.

## 2.1 Définition formelle

Soit $\{x_1, x_2, \ldots, x_n\}$ un échantillon de taille $n$ extrait d'une variable aléatoire continue $X$ avec une densité de probabilité inconnue $f(x)$. L'estimation par densité noyau de $f(x)$ est donnée par l'expression :

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right),$$

où :

- $K(\cdot)$ est la fonction noyau, qui est une fonction symétrique intégrant à 1, souvent choisie comme une densité de probabilité (exemple : gaussienne, uniforme, etc.),Elle intègre la distance utilisée.
- $h > 0$ est le paramètre de lissage, appelé *largeur de bande* (*bandwidth* en anglais), qui contrôle la régularisation de l'estimation.

Pour plus de propriétés sur le noyau K et le choix de la largeur h, voir le Chapitre 1.1 sur le rapport annexe sur le repository

## 2.2 Applications

Dans notre sujet, la KDE est utilisée en analyse spatiale pour :

- La visualisation de la distribution d'un ensemble de données
- La détection d'anomalies
- Identification des zones d'activité piétonne

# 3 État de l'art sur le pathfinding 2D

Le pathfinding en 2D (ou recherche de chemin en 2D) est un problème

d'intelligence artificielle consistant à déterminer le chemin le plus court ou le plus efficace entre deux points dans un environnement bidimensionnel, tel qu'un jeu, une simulation, ou un système d'information géographique. Un tel algorithme de recherche de chemin vise à trouver un itinéraire qui évite les obstacles, minimise la distance ou le temps, et satisfait des contraintes spécifiques.

### 3.1 Quelques principes et approches clés sur le pathfinding 2D

- Algorithmes de base : Les algorithmes de base pour le pathfinding 2D incluent Dijkstra, A* (A-star), et Floyd-Warshall.
- Heuristiques : en informatique et en mathématique une heuristique est une méthode ou une stratégie de calcul approximative qui permettant de résoudre un problème de manière rapide et efficace, sans garantir une solution optimale ou parfaite. L'objectif principal est de réduire le temps de calcul et la complexité d'un problème en donnant une solution « acceptable » dans des cas où une solution exacte serait trop coûteuse en temps ou en ressources. Dans le contexte du pathfinding 2D ,et dans certains algorithmes, les heuristiques sont utilisées pour les guider vers le bon chemin.
- Graphe et réseau : Un problème de pathfinding 2D est souvent représenté à l'aide d'un graphe ou d'un réseau, où les nœuds représentent les positions dans l'environnement et les arêtes représentent les mouvements possibles entre ces positions.

### 3.2 Défis et limitations

- Complexité de l'environnement : Les obstacles irréguliers ou mobiles nécessitent des recalculs en temps réel, rendant le cheminement plus difficile.
- Coût de calcul élevé dans les environnements étendus : Les problèmes de recherche de chemin deviennent plus gourmands en ressources (temps de calcul et mémoire). Des optimisations, comme des heuristiques efficaces, sont nécessaires pour maintenir des performances acceptables.
- Limites des algorithmes heuristiques : Dans les environnements complexes ou non structurés, les heuristiques peuvent ignorer des chemins valables ou privilégier des solutions locales inadéquates. Un compromis entre précision et efficacité est essentiel.

## 4 Méthodologie

La méthodologie suivie dans cette étude repose sur l'application des algorithmes classiques de recherche de chemin, tout en tenant compte des particularités des environnements urbains. Cette section décrit les différents algorithmes testés, les données utilisées pour l'étude, et les étapes de prétraitement mises en place pour assurer des résultats fiables.

### 4.1 Algorithmes de recherche de chemin

**Dijkstra** est un algorithme de recherche de chemin optimal basé sur une exploration exhaustive de tous les nœuds à partir du point de départ. Bien que très précis, cet algorithme est relativement lent dans les grandes grilles, ce qui peut poser un problème dans des environnements urbains complexes. (Pour plus de précisions sur l'algorithme Dijkstra, voir le rapport annexe Chapitre 3.1)**A*** améliore Dijkstra en utilisant une heuristique [1] qui guide l'exploration du graphe, réduisant ainsi le temps de calcul tout en conservant une

bonne précision (Pour plus de précisions sur l'algorithme A*, voir le rapport annexe Chapitre 3.5). Enfin, **Jump Point Search (JPS)** représente une amélioration d'A*, en supprimant les nœuds intermédiaires non pertinents et en sautant directement aux points de décision significatifs, ce qui permet de réduire davantage le nombre de calculs dans les grilles denses (Pour plus de précisions sur l'algorithme JPS, voir le rapport annexe Chapitre 3.10).

## 4.2 Données et prétraitement

Les données utilisées dans cette étude sont deux rasters représentant des zones urbaines piétonnes, chacun ayant une résolution différente (301x250 et 151x125 pixels). Ces rasters ont été générés à partir de données géospatiales de la ville de référence, en identifiant les zones accessibles aux piétons (rues, trottoirs, etc.) et en excluant les zones inaccessibles (bâtiments, parcs, etc.). Le prétraitement des données a impliqué l'identification des pixels marchables et la génération d'une liste d'adjacence, ce qui a permis de construire un graphe représentant les connexions possibles entre les différents points de l'espace urbain. Des ajustements ont également été effectués pour assurer une cohérence des tests sur plusieurs itérations.

## 4.3 Implémentation et Tests

Les trois algorithmes ont été testés sur un total de 25 couples de points de départ et d'arrivée, générés aléatoirement dans les zones marchables des rasters. Pour chaque test, les performances des algorithmes ont été mesurées en termes de temps d'exécution et de précision des chemins calculés. Les temps d'exécution ont été mesurés pour chaque raster, et des visualisations graphiques ont été produites pour illustrer les différences entre les chemins calculés par chaque algorithme.

# 5 Résultats

Les résultats de cette étude montrent des différences significatives entre les performances des trois algorithmes dans le cadre de la recherche de chemin dans des environnements urbains contraints. Nous détaillons ici les performances mesurées, les résultats qualitatifs des chemins générés, et des exemples visuels des résultats obtenus.

## 5.1 Performance des algorithmes

Les résultats des tests de performance montrent des variations marquées entre les algorithmes, en particulier en ce qui concerne le temps d'exécution moyen. Le tableau suivant résume les temps d'exécution observés pour les grille 2D utilisées pour le test.

| **Algorithme** | **Temps (Zone 1 Test)** |
|---|---|
| Dijkstra | 4.3 s |
| A* | 1.2 s |
| JPS | 0.6 s |

Table 1: Temps d'exécution moyen des algorithmes sur la zone de test

## 5.2 Analyse qualitative

Les résultats qualitatifs révêlent des différences notables entre les algorithmes. A* et JPS sont en effet bien plus performants grâce à l'utilisation de l'heuristique et grâce au pruning. Ces deux optimisations permettent ainsi de visiter beaucoup moins de noeuds que Dijkstra et donc d'améliorer le temps de calcul.

Pour le pruning et le saut, il est nécessaire de calculer les distances du noeud d'où l'on vient vers le noeud où on veut aller, ce qui implique de calculer jusqu'à huit distances supplémentaires.

Donc cela rajoute beaucoup d'exécutions supplémentaires. Notre conjecture était que le saut, même avec

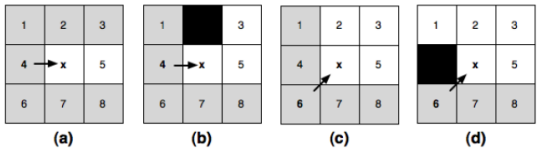


Figure 1: Schéma du processus de l'élagage avec JPS [4]

des sous exécutions de A* supplémentaires, permettrait au final sur l'ensemble de la grille, d'avoir moins de noeuds à visiter.

Les résultats permettent de confirmer cette conjecture.


Figure 2: Exemple de chemin calculé par A* sur la zone 1 de test.

# 6 Discussion

L'utilisation des algorithmes de pathfinding pour le traitement des rasters représente une méthode puissante pour calculer des dsitances plus réalistes dans un environnement contraint telqu'un espace urbain. Contrairement à la distance euclidienne, la distance réaliste ( "marchable") permet d'intégrer les spécificités des infrastructures urbain ( batiments, zones interdites ... ).

## 6.1 Application sur le raster de la zone d'étude

Notres zone d'étude est le centre ville de nantes , nous disposons de données vecteurs représentant la zone marchable, les batiments , les point d'intérêt. Pour utiliser les algorithmes qu'on a implémenté , la première étape est la conversion de la couche vecteur de notre zone "marchable" en raster.

- Le calcul montre bien la grande différence entre ; distance euclidienne (figure 3) et distance calculée par les algoritmes de pathfinding (figure 4)

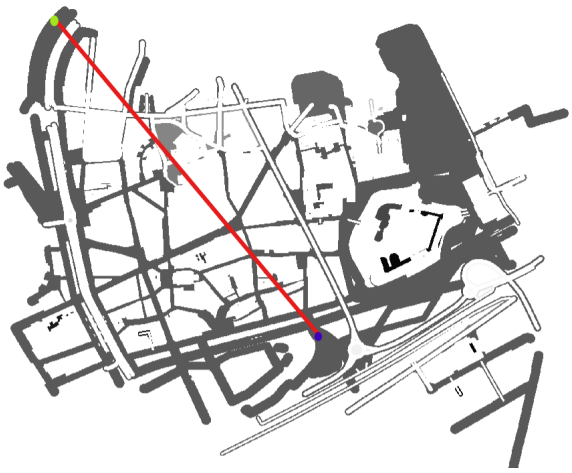

Figure 3: Exemple de chemin basé sur la distance euclidienne.

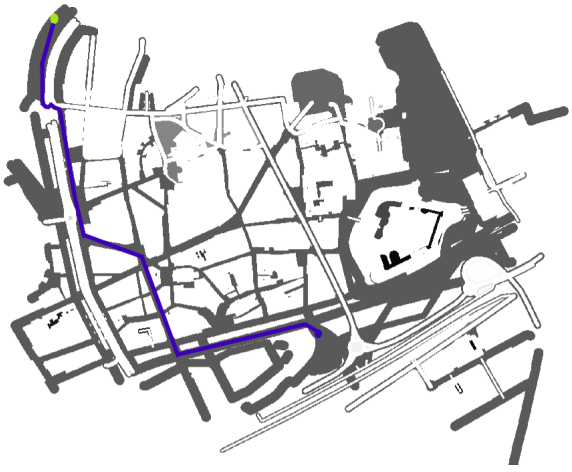

Figure 4: Exemple de chemin (approximatif) résultat de l'utilisation des algorithmes .

### 6.2 Un facteur important : le seuil de simplification du raster

Lors de la conversion des données de la zone marchable en un fichier raster, il est crucial de prendre en compte la résolution du raster. Il s'agit de trouver un seuil de simplification satisfaisant, permettant d'obtenir un compromis entre le temps d'exécution, la précision des résultats et la connexité de la zone marchable.

- Une résolution élevée permet de décrire les distances réalistes avec une grande précision, mais au prix d'un temps d'exécution plus important.
- À l'inverse, une résolution plus basse (mauvaise qualité) réduit la précision des résultats, mais diminue considérablement le temps d'exécution des algorithmes et peux donner lieu à des interruptions de la zone marchable.

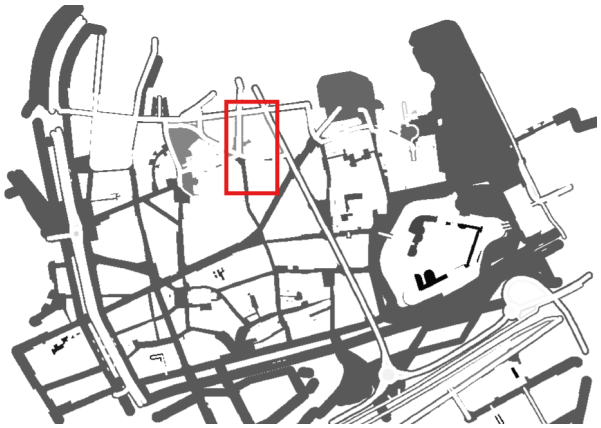

Figure 5: Exemple de zone où on peut avoir des interruptions de connexité de la zone "marchable".

## 7 Conclusion

Cette étude met en évidence l'importance d'adapter les algorithmes de recherche de chemin aux contraintes spécifiques des environnements urbains. Au delà du type d'algorithme utilisé, ainsi que la méthodologie adoptée pour appliquer ces algorithmes, influencent significativement le temps d'exécution. L'algorithme JPS s'est révélé être le plus performant en offrant un excellent compromis entre déterminisme et rapidité d'exécution.

## References


[1] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[2] Dorian Essamir. Le pathfinding a* et l'optimisation pour des grilles 2d, mémoire de master. `https://www.guillaumelevieux.com/siteperso/contents/recherchem1/essamir/pathfinding.pdf`. Disponible en ligne.

[3] ETUDESTECH.com. Guide complet de l'algorithme de dijkstra. `https://etudestech.com/decryptage/algorithme-dijkstra-calculer-chemin-plus-cou` Disponible en ligne.

[4] Daniel Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. volume 2, 01 2011.

[5] Amit Patel. Introduction to a*. `https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html`. Disponible en ligne.

[6] Nathan Witmer. A visual explanation of jump point search. `https://zerowidth.com/2013/a-visual-explanation-of-jump-point-search/`, 2013. Disponible en ligne.