WCITCA 2015: Paper ID. WCITCA2015-1570145581-77

The 2nd IEEE World Symposium on Web Applications and Networking (WSWAN'2015) , 21-23 March 2015 | Hammamet, Tunisia

# Optimisation de l'utilisation de l'algorithme de Dijkstra pour un simulateur multi-agents spatialisé

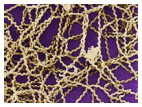
**Pape Adama MBOUP[1,2], Mamadou Lamine MBOUP[3], Karim KONTE[1], Pascal HANDSCHUMACHER4 [4] and Jean Le Fur[5]**

[1]Département de mathématiques et d'informatique, UCAD, Dakar, Sénégal
[2]IRD, UMR 022 BIOPASS, Dakar, Sénégal
[3]Département Génie Informatique, ESP, UCAD, Dakar, Sénégal
[4]*IRD, UMR 912 SESSTIM INSERM - IRD - U2, Faculté de géographie UdS, 3, rue de l'Argonne F-67000,* Strasbourg , France
[5]IRD, UMR 022 CBGP, Campus international de Baillarguet, CS 30016, 34988, Montferrier, France
pamboup@hotmail.fr...

## Plan

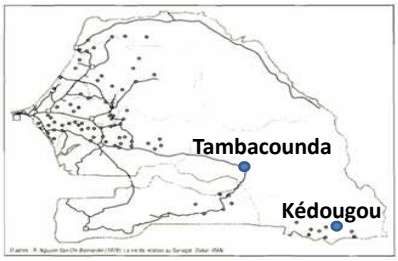
- ❑ **Contexte et Motivation**
- ❑ **Problématique**
- ❑ **Contribution**
- ❑ **Conclusion**

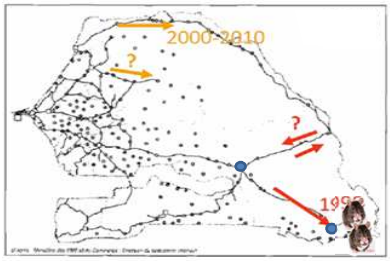
# Contexte et Motivation



**Bactéries**

**Maladie : ex. leptospirose**




- **Question : diffusion des rats par le transport humain ?**
- **Pour répondre :** simulateur transport humain sur l'ensemble du pays et sur 1 siècle

# Problématique

**Simulater** les **transports humains** sur l'**ensemble du pays** et sur **1 siècle**

**Nécessite :**

- Beaucoup de voies de transport (graphes)
- Beaucoup de transporteurs (agents)
- Temps considérable (siècle)

**Méthode :**

- ✓ Graphe non pondérés non orienté
- ✓ algorithme de Dijkstra (plus cours chemin)

**Problème :**

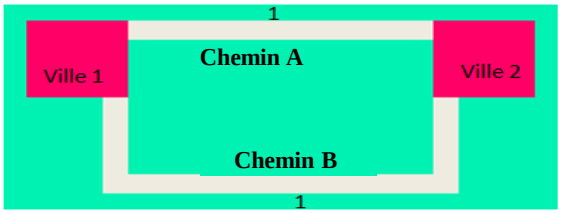
- algorithme de Dijkstra : lourd

**Solution :**

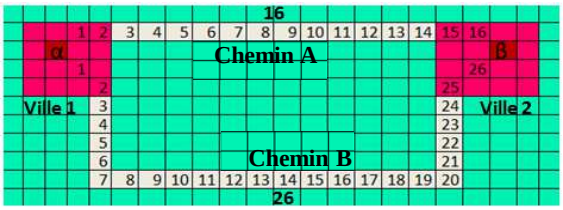
- o Autre algorithme : **A***
- ✓ Optimiser l'utilisation de Dijktra

# Remarque

## Graphe non pondérés et distance




**Chemin A = 1 arête**
**Chemin B = 1 arête**
**→ Chemin A = Chemin B**




**Chemin A = 16 arêtes**
**Chemin B = 26 arêtes**
**→ Chemin A plus court que Chemin B**

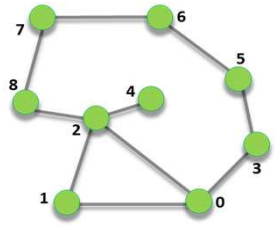
# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijkstra

# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijkstra

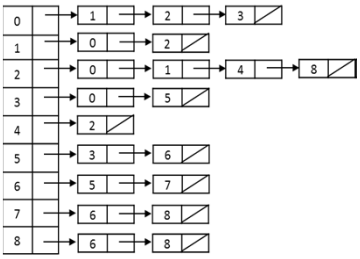
### 1. Optimisation de la représentation informatique d'un graphe




Représentations classiques

Représentation optimisée

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 8 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |




| | |
|---|---|
| 0 | {1, 2, 3} |
| 1 | {0,2} |
| 2 | {0,1,4,8} |
| 3 | {0,5} |
| 4 | {2} |
| 5 | {3,6} |
| 6 | {5,7} |
| 7 | {6,8} |
| 8 | {2,7} |

# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijkstra

### 1. Optimisation de la représentation informatique d'un graphe

#### Résultat :

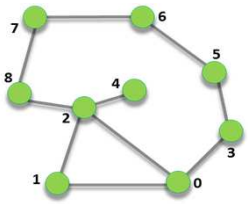
| | graphe 1 | graphe 2 | graphe 3 | graphe 4 | graphe 5 | Graphe 6 |
|---|---|---|---|---|---|---|
| **Nombre nœud** | 112 | 67 | 40 | 151 | 685 | 4677 |
| **E.M.A.C.** | 12544 | 4489 | 1600 | 22801 | 469225 | 21874329 |
| **E.M.A.O.** | 548 | 362 | 198 | 686 | 3920 | 9352 |
| **E. libéré** | 11996 | 4127 | 1402 | 22115 | 465305 | 21864977 |
| **E. libéré (%)** | 95,63 | 91,94 | 87,63 | 96,99 | 99,16 | 99,96 |

E = Espace
M.A. = matrice d'adjacence
C. = classique
O. = optimisée

# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijkstra

### 2. Algorithme de Dijkstra :




Agent 1:
Chemin(0 et 7)

**Dijkstra (0)**

**Tableau des précédents pour 0**

| nœuds | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| nœuds précédents | 0 | 0 | 0 | 0 | 2 | 3 | 5 | 8 | 2 |

**Chemin entre 0 et 7**

| | |
|---|---|
| Chemin entre 0 et 7 | {0, 2, 8, 7} |

| | |
|---|---|
| **0** | {1, 2, 3} |
| **1** | {0,2} |
| **2** | {0,1,4,8} |
| **3** | {0,5} |
| **4** | {2} |
| **5** | {3,6} |
| **6** | {5,7} |
| **7** | {6,8} |
| **8** | {2,7} |

Agent 1 ou Agent 2
Chemin(**0** et 7) ou
Chemin(**0** et 6)

Dijkstra (0)

Tableau des précédents pour 0 **(réinventer la roue)**

| nœuds | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| nœuds précédents | 0 | 0 | 0 | 0 | 2 | 3 | 5 | 8 | 2 |

Chemin entre 0 et 7 **(réinventer la roue)**

| | |
|---|---|
| Chemin entre 0 et 7 | {0, 2, 8, 7} |

**Chemin entre 0 et 6**

| | |
|---|---|
| Chemin entre 0 et 6 | {0, 3, 5, 6} |

# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijkstra

### 3. Stockage des infos déjà construits

**2 Principes :**

- ✓ Tableau des précédents ou chemin jamais utilisé → jamais construit
- ✓ Tableau des précédents ou chemin 1 fois construit l'est pour tous

**Matrice des précédents**

| | nœuds | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| nœud précédent | **0** | 0 | 0 | 0 | 0 | 2 | 3 | 5 | 8 | 2 |
| | **1** | 1 | 1 | 1 | 0 | 2 | 3 | 5 | 8 | 2 |
| | **2** | 2 | 2 | 2 | 0 | 2 | 3 | 7 | 8 | 2 |
| | **3** | 3 | 0 | 0 | 3 | 2 | 3 | 5 | 6 | 2 |
| | **4** | 2 | 2 | 4 | 0 | 4 | 3 | 7 | 8 | 2 |
| | **5** | 3 | 0 | 0 | 5 | 2 | 5 | 5 | 6 | 7 |
| | **6** | 3 | 0 | 8 | 5 | 2 | 6 | 6 | 6 | 7 |
| | **7** | 2 | 2 | 8 | 5 | 2 | 6 | 7 | 7 | 7 |
| | **8** | 2 | 2 | 8 | 0 | 2 | 6 | 7 | 8 | 8 |

**Matrice des chemins (optimisés)**

| | | |
|---|---|---|
| **0** | **1** | {0, 1} |
| | **2** | {0, 2} |
| | **3** | {0, 3} |
| | **4** | {0, 2, 4} |
| | **5** | {0, 3, 5} |
| | **6** | {0, 3, 5, 6} |
| | **7** | {0, 2, 8, 7} |
| | **8** | {0, 2, 8} |
| **1** | **2** | {1, 2} |
| | **3** | {1, 0, 3} |
| | **4** | {1, 2, 4} |
| | **5** | {1, 0, 3, 5} |
| | **6** | {1, 0, 3, 5, 6} |
| | **7** | {1, 2, 8, 7} |
| | **8** | {1, 2, 8} |

…

# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijkstra

### 3. Stockage des infos déjà construits

#### Résultat :

**Graphe :**

**Nombre de nœud : 4677**
**nombre d'arêtes : 9352**

**Durée construction d'un plus court chemin :**

| (1) Dijkstra | (2) Matrice des précédents | (3) Matrice des chemins |
|---|---|---|
| 2,40 ms | 0,01 ms | 0,00018 ms |

**Comparaison :**

**(3)** est **55,56** fois plus rapide **(2)**
**(2)** est **240,0** fois plus rapide **(1)**

# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijkstra

### 4. Construction ou récupération d'un plus court chemin

Algorithme : Faire Chemin (*entree*: entier, *sortie*: entier)
    Si dans matrice des chemins il existe *chemin* entre *entree* et *sortie* alors
        renvoie *chemin*
    Sinon si matrice des précédents existe pour *entree* alors
        en élabore chemin
        stocker chemin dans matrice des chemins
        renvoie chemin
    Sinon
        matrice des précédents pour entree ← Dijkstra(entree)
        faire chemin (entree, sortie)
    Fin si
Fin algorithme

# Contribution

## Optimisation de l'utilisation de l'algorithme de Dijktra

### 4. Construction ou récupération d'un plus court chemin

### Résultat :

**Simulation :**

**Nombre** agents transporteurs : **400**
**nombre ticks (pas de temps) : 30 000**
**Nombre de chemins construits : 21 676**

**Nombre d'utilisation :**

| (1) Dijkstra | (2) Matrice des précédents | (3) Matrice des chemins |
|---|---|---|
| 54 | 20 399 | 1 277 |

**Comparaison :**

**Dijkstra n'est utilisé que 54 fois au lieu de 21 676 fois.**

# Conclusion

Dans cette étude, nous avons montré comment nous pouvons passer de l'optimisation spatiale de la représentation informatique d'un graphe pour stocker de manière optimale les tableaux des précédents et les plus courts chemins construits par l'algorithme de Dijkstra. Ce qui permet d'éviter aux agents transporteurs de se reconstruire des chemins déjà construits auparavant et donc diminue considérablement le temps de calcul dû à la construction de plus court chemin. ~~Ainsi nous sommes arrivés à une optimisation spatiale et temporelle de l'utilisation de l'algorithme de Dijkstra dans un simulateur Multi-agents spatialisé.~~

# Merci de votre attention