420-203-RE
DÉVELOPPEMENT D'APPLICATIONS DANS UN ENVIRONNEMENT GRAPHIQUE

RECHERCHE DE CHEMINS (*PATHFINDING*)

Concepts de base

1

INTRODUCTION (1)

Le concept de recherche de chemins est utile dans de nombreux domaines. Certains sont évidents :

- GPS et recherche d'itinéraires
- Robotique
- Jeux vidéo

2

2

INTRODUCTION (2)

Cependant, cela va beaucoup plus loin. Le domaine de la recherche de chemin est en fait, celui de la **recherche de solutions**.

3

3

INTRODUCTION (3)

Les **algorithmes d'exploration** liés à la recherche de chemins peuvent être utilisés dans de nombreux problèmes où les différentes possibilités peuvent être représentées sous la forme d'un graphe. Par exemple, le jeu de Go ou les échecs.

4

4

INTRODUCTION (4)

C'est pourquoi le domaine de la recherche de chemins est associé, en intelligence artificielle, au domaine plus général de la **planification**.

5

5

INTRODUCTION (5)

Dans le domaine du jeu vidéo, ces algorithmes ont de nombreux usages, les principaux étant le déplacement des NPC et les *planners* permettant de déterminer une séquence d'actions optimale pour un agent intelligent.

6

6

UN GRAPHE ??? (1)

Le graphe est une structure mathématique qui est habituellement représentée graphiquement par un ensemble de nœuds (points, sommets) reliés par des arcs (liens, arêtes).

7

7

UN GRAPHE ??? (2)

La théorie des graphes représente un des principaux sous-domaines des mathématiques discrètes.

8

8

UN GRAPHE ??? (3)

En planification, le graphe permet de représenter l'ordonnancement des différentes étapes d'un problème, menant à sa solution.

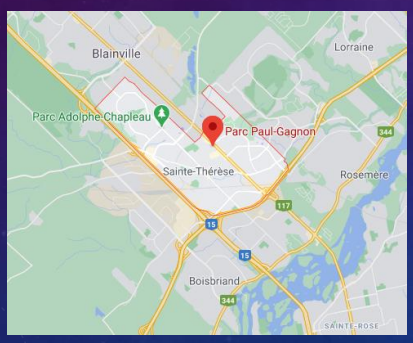
9

9

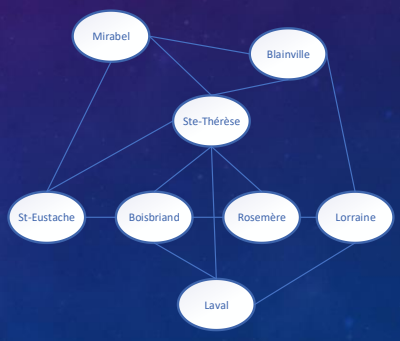
UN GRAPHE ??? (4)

On l'utilise fréquemment comme un modèle simplifié de la réalité.

Par exemple, en cartographie…



Réalité



Graphe (modèle)

10

10

UN GRAPHE ??? (5)

La représentation graphique n'est pas toujours très pratique à utiliser, surtout si l'on doit la manipuler avec un algorithme.

On peut alors utiliser une **matrice adjacente**. Dans cette matrice la présence d'un arc entre deux nœuds est symbolisée par la valeur **1** et l'absence d'un arc par la valeur **0**.

11

11

UN GRAPHE ??? (6)

	Blainville	BoisBriand	Laval	Lorraine	Mirabel	Rosemère	Ste-Thérèse	St-Eustache
Blainville	0	0	0	1	1	0	1	0
BoisBriand	0	0	1	0	0	1	1	1
Laval	0	1	0	1	0	0	1	1
Lorraine	1	0	1	0	0	1	0	0
Mirabel	1	0	0	0	0	0	1	1
Rosemère	0	1	0	1	0	0	1	0
Ste-Thérèse	1	1	1	0	1	1	0	1
St-Eustache	0	1	1	0	1	0	1	0

Représentation graphique

Matrice adjacente

12

12

COÛT DU CHEMIN (1)

La représentation graphique du graphe et la matrice adjacente nous permettent de voir s'il existe un chemin possible entre deux nœuds, mais ne nous indique pas s'il s'agit du plus court chemin, car pour l'instant, ces deux représentations ne tiennent pas compte de la longueur des arcs.

13

13

COÛT DU CHEMIN (2)

Pour trouver le plus court chemin (le moins couteux), il faut ajouter des informations que l'on nomme habituellement des longueurs ou des poids. Ces informations peuvent représenter, selon le contexte, des minutes, des mètres ou des dollars par exemple.

14

14

COÛT DU CHEMIN (3)

On ajoute les longueurs (ou les poids) sur la représentation graphique de notre graphe. Dans notre exemple, c'est le temps nécessaire, en minutes pour aller d'une ville à l'autre.

15

15

COÛT DU CHEMIN (4)

Pour obtenir la **matrice de longueur** :

1. On remplace les « 1 » de la matrice adjacente par la longueur des arcs.
2. Les « 0 », qui ne sont pas sur la diagonale de la matrice, sont remplacés par ∞.

Les **0** indiquent que le temps de déplacement est nul (coût = 0), tandis que les **∞** indiquent que le déplacement est impossible.

	Blainville	BoisBriand	Laval	Lorraine	Mirabel	Rosemère	Ste-Thérèse	St-Eustache
Blainville	0	∞	∞	15	15	∞	5	∞
BoisBriand	∞	0	15	∞	∞	5	8	10
Laval	∞	15	0	15	∞	∞	20	15
Lorraine	15	∞	15	0	∞	5	∞	∞
Mirabel	15	∞	∞	∞	0	∞	10	20
Rosemère	∞	5	∞	5	∞	0	10	∞
Ste-Thérèse	5	8	20	∞	10	10	0	15
St-Eustache	∞	10	15	∞	20	∞	15	0

16

16

ALGORITHME D'EXPLORATION (1)

Il existe plusieurs algorithmes d'exploration et nous devons bien comprendre leurs caractéristiques pour être capables de déterminer lequel utiliser en fonction de la nature du problème à résoudre. Aucun algorithme n'est systématiquement meilleur que tous les autres.

17

17

ALGORITHME D'EXPLORATION (2)

On mesure la performance de ces algorithmes en fonction de 4 critères :

1. Complétude
2. Optimalité
3. Complexité en temps
4. Complexité en espace (en mémoire)

18

18

ALGORITHME D'EXPLORATION (3)

Mesure de la complexité, notation grand O.

- O pour ordre => la vitesse de croissance d'un fonction
- b : le nombre maximum de successeurs pour un état
- d : profondeur du meilleur nœud de la solution
- m : profondeur maximum (*peut être infinie*)
- n

19

19

ALGORITHME D'EXPLORATION (4)

On catégorise fréquemment les algorithmes d'exploration en fonction de deux caractéristiques :

1. Sont-ils « **naïfs** » ou « **intelligents** » ?
2. Sont-ils « **non informés** » ou « **informés** » ?

20

20

« NAÏFS » OU « INTELLIGENTS » (1)

Un algorithme « naïf » va avoir tendance à tester tous les chemins possibles pour déterminer s'il existe un chemin entre deux nœuds. De plus, il n'est pas garanti que le chemin trouvé par cet algorithme soit réellement le plus court.

21

21

« NAÏFS » OU « INTELLIGENTS » (2)

Un algorithme « intelligent » va habituellement permettre de trouver le plus court chemin, mais il n'est pas le plus optimisé. Il utilise l'information produite par l'algorithme pour éviter d'avoir à tester tous les chemins possibles.

22

22

« NON INFORMÉS » OU « INFORMÉS » (1)

Un algorithme « non informé » ou « aveugle » ne dispose pas d'information préalable sur le problème à résoudre. Par exemple, sur une carte, il ne connaît pas la position du but à atteindre. Il sait ce qu'il cherche, mais ne sait pas où cela se trouve.

23

23

« NON INFORMÉS » OU « INFORMÉS » (2)

Un algorithme « informé » ou « d'exploration heuristique » dispose d'informations qui lui permettent d'approximer la direction à suivre pour atteindre le but.

24

24

ALGORITHME DE PARCOURS EN PROFONDEUR (1)
DEPTH-FIRST SEARCH (*DFS*)

- Créateur : Charles Pierre Trémaux (avant 1882)
- Utilisation : Trouver un chemin dans un labyrinthe.
- Catégorie : Naïf et non informé
- Variante : Exploration en profondeur limitée

  Exploration itérative en profondeur

25

25

ALGORITHME DE PARCOURS EN PROFONDEUR (2)
DEPTH-FIRST SEARCH (*DFS*)

- Complétude : Complète, si le graphe est fini,

  non-complète dans le cas contraire.
- Optimalité : Non optimale.
- Complexité en temps : $O(b^m)$ => mortelle si m >>>d
- Complexité en espace : O(b * m) ou O(m) si linéaire

26

26

ALGORITHME DE PARCOURS EN LARGEUR (1)
BREADTH FIRST SEARCH (*BFS*)

- Créateur : Konrad Zuse (1945)
- Utilisation : Trouver un chemin dans un labyrinthe.
- Catégorie : Naïf et non informé
- Variante : Exploration à coût uniforme

27

27

ALGORITHME DE PARCOURS EN LARGEUR (2)
BREADTH FIRST SEARCH (*BFS*)

- Complétude : Complète
- Optimalité : Optimale si le coût d'une action = 1,
  non optimale en générale
- Complexité en temps : $O(b^d)$
- Complexité en espace : $O(b^d)$

28

28

ALGORITHME DE DIJKSTRA (1)

DIJKSTRA'S SHORTEST PATH FIRST ALGORITHM (*SPF*)

- Créateur : Edsger Dijkstra (1956)
- Utilisation : Calculer le plus court chemin entre une source et une destination.
- Catégorie : Intelligent et non informé
- Variante : Algorithme de Bellman-Ford

29

29

ALGORITHME DE DIJKSTRA (2)

DIJKSTRA'S SHORTEST PATH FIRST ALGORITHM (*SPF*)

- Complétude : Complète
- Optimalité : Optimale
- Complexité en temps : O(a + s log(s)) où a est le nombre d'arcs et s le nombre de sommets.
- Complexité en espace : O(s)

30

30

ALGORITHME « MEILLEUR D'ABORD » (1)
BEST FIRST SEARCH

- Utilisation : Calculer le plus court chemin entre une source et une destination.
- Catégorie : Intelligent et informé

31

31

ALGORITHME « MEILLEUR D'ABORD » (2)
BEST FIRST SEARCH

Principe (*algorithme gourmand ou glouton*)

1. On associe à chaque nœud une mesure d'utilité basée sur une heuristique, par exemple la distance de Manhattan entre le nœud et le but.
2. Lors du parcours, on choisit systématiquement d'aller vers le nœud qui a la valeur d'heuristique la plus basse.

32

32

ALGORITHME « MEILLEUR D'ABORD » (3)

BEST FIRST SEARCH

- Complétude : Complète, si le graphe est fini,
  non-complète dans le cas contraire.
- Optimalité : Non optimale
- Complexité en temps : $O(b^m)$
- Complexité en espace : $O(b^m)$

33

33

ALGORITHME « A*» (1)

- Utilisation : L'algorithme de recherche de chemins le plus utilisé en jeu vidéo. Calculer le plus court chemin entre une source et une destination.
- Catégorie : Intelligent et informé
- Variante du « Meilleur d'abord »
- Créateur : Nils Nilsson (1968)

34

34

ALGORITHME « A*» (2)

Principe

1. On associe à chaque nœud une mesure d'utilité basée sur une heuristique, par exemple la distance de Manhattan entre le nœud et le but.
2. Lors du parcours, on choisit systématiquement d'aller vers le nœud qui a la valeur d'heuristique la plus basse **en tenant compte du coût nécessaire pour y parvenir**.

35

35

ALGORITHME « A*» (3)

- Complétude : Complète
- Optimalité : Optimale
- Complexité en temps :
  - Pire cas : Exponentiel, donc $O(b^m)$
  - Cas moyen : Logarithmique O(log h(source))
- Complexité en espace :
  - Pire cas : Exponentiel, donc $O(b^m)$
  - Cas moyen : O(b)

36

36