Algorithmes de recherche de chemin

Afin que notre robot puisse aller d'un point A à un point B, nous avons décidé d'utiliser un algorithme de recherche de chemin basé dans un repère. Voici un état de l'art de tous les algorithmes existants.

L'algorithme de parcours en largeur.

Cet algorithme permet le parcours récursif d'un graphe ou d'un arbre à partir d'un nœud source. Le principe du parcours en largeur est le suivant : on explore en premier le nœud source, puis ses successeurs, puis les successeurs de ses successeurs, etc. Cet algorithme visite tous les nœuds du graphe atteignable à partir du nœud source. Il peut être utilisé afin de calculer les plus courts chemins entre le sommet source et tous les sommets du graphe. Sa complexité temporelle dans le pire cas est O(s+a) avec s le nombre de sommets et a le nombre d'arcs.

L'algorithme de parcours en profondeur.

Le parcours en profondeur est un algorithme récursif de parcours de graphe ou d'arbre. A partir d'un sommet source, il explore un chemin dans le graphe jusqu'à arriver à une impasse ou à un nœud déjà visité. Il revient alors au dernier sommet pouvant mener à un autre chemin et explore celui-ci. L'algorithme s'arrête lorsque tous les sommets atteignables à partir du sommet source ont été visités. Il est utilisé afin de trouver l'ensemble des sommets accessibles depuis un sommet source. Sa complexité dans le pire des cas est O(s+a) avec s le nombre de sommets et a le nombre d'arcs.

L'algorithme de Dijkstra.

Cet algorithme est un algorithme de recherche du plus court chemin dans un graphe pondéré positivement. Il est fondé sur un parcours en largeur. A partir d'un point de départ, il recherche le plus court chemin menant à la destination de façon itérative. L'algorithme se base sur le principe que tous les sous-chemins C-D du chemin le plus court entre les points A et B sont aussi le plus court chemin entre les points C et D. La complexité temporelle de l'algorithme de Dijkstra varie en fonction de la manière dont il est implémenté : avec un tas binaire, elle est de O((a+s) x log(s)) ; avec un tas de Fibonacci, elle est de O(a+s x log(s)).

L'algorithme A*.

L'algorithme A* est un algorithme de recherche du plus court chemin d'un point à un autre dans un graphe. C'est est une version améliorée de l'algorithme de Dijkstra qui utilise des heuristiques pour orienter sa recherche du plus court chemin. Les heuristiques estiment la distance qui sépare le point considéré du point d'arrivée. Pour que l'algorithme soit admissible, c'est-à-dire pour qu'il garantisse de trouver le plus court chemin, l'heuristique choisie ne doit pas surestimer la distance au point d'arrivée. La complexité de l'algorithme dépend de la qualité de l'heuristique utilisée. L'un des avantages de A* est sa simplicité de calculs. Toutefois, celui-ci ne peut pas adapter le chemin trouvé si l'environnement (le graphe) change.

L'algorithme D* ou l'algorithme A* dynamique.

Cet algorithme se comporte comme A* avec la différence que l'algorithme permet une replanification rapide de la trajectoire si un obstacle inattendu est détecté sur la route. L'algorithme commence à étudier les nœuds à partir du nœud d'arrivée. Chaque nœud possède un "backpointer" qui représente le nœud étudié précédemment (donc le

prochain nœud de la trajectoire finale), et le coût pour aller d'un nœud au nœud d'arrivée est connu. L'algorithme se termine lorsque le nœud de départ est rencontré. Le chemin à suivre est alors indiqué par les "backpointers".

LPA* ou Lifelong Planning A*.

Cet algorithme est une version incrémental de A* qui réutilise des informations calculées lors de précédentes recherches. Sa première recherche est l'algorithme A*. L'algorithme trouve de manière répétée les chemins les plus courts d'un nœud de départ à un nœud d'arrivée dans un graphe, à mesure que des arcs ou des nœuds sont ajoutés ou supprimés ou que les coûts des arêtes sont modifiés. LPA* permet de réduire fortement la durée de recherche lorsque des changements mineurs surviennent dans le graphe. En effet, l'algorithme ne réévalue que les noeuds affectés par le changement, contrairement à A*.

D* Lite.

Cet algorithme est un algorithme de recherche heuristique incrémentale qui a le même comportement que l'algorithme D* mais qui se base sur le LPA*. Contrairement au LPA* qui commence sa recherche à partir du noeud de départ, D* Lite commence à chercher à partir du nœud d'arrivée. Cet algorithme, plus simple à implémenter que D* et a une performance égale voire supérieure à celui-ci.

Dans l'étude de notre robot, A* peut représenter un bon choix en raison de sa simplicité de calcul. Toutefois, l'algorithme présente un inconvénient : il est incable de s'ajuster lorsque des obstacles inattendus apparaissent sur le chemin. Il n'est donc pas adapté à notre situation. Parmi les algorithmes de recherche incrémentale, D*, LPA* et D* Lite, nous avons décidé d'utiliser le l'algorithme D* Lite, de par sa simplicité d'implémentation et son efficacité.