Projet interdisciplinaire ou de recherche

Comparateur d'algorithmes de plus court chemin

Vincent Albert
Nicolas Bédrine

Année 2015–2016

Projet réalisé pour l'équipe Maia du laboratoire Loria

Encadrant : M. Olivier Buffet

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Albert, Vincent

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 1205033068

Année universitaire : 2015–2016

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Benchmarking d'algorithmes de plus court chemins sur grilles générées aléatoirement

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 16 mai 2016

Signature :

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Bédrine, Nicolas

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) :

Année universitaire : 2015–2016

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Benchmarking d'algorithmes de plus court chemins sur grilles générées aléatoirement

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 16 mai 2016

Signature :

Projet interdisciplinaire ou de recherche

Comparateur d'algorithmes de plus court chemin

Vincent Albert
Nicolas Bédrine

Année 2015–2016

Projet réalisé pour l'équipe Maia du laboratoire Loria

Vincent Albert
Nicolas Bédrine
vincent.albert@telecomnancy.eu
nicolas.bedrine@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

Loria
615, Rue du Jardin botanique
54506, Vandœuvre-lès-Nancy
03 83 59 20 00

Encadrant : M. Olivier Buffet

Remerciements

Nous tenons à remercier toutes les personnes nous ayant accompagnés durant la réalisation de ce projet, et en particulier de M. Buffet pour nous avoir encadré durant toute la durée du projet et pour nous avoir prodigué ses précieux conseils.
Nous remercions aussi M. Jean-François Scheid qui a supervisé l'organisation des PIDRs et de Mme Isabelle Chenet, secrétaire des PIDRs.
Enfin, nous remercions toute l'équipe pédagogique et administrative de TELECOM Nancy pour nous fournir un cadre d'études adéquat à la réalisation de ce projet.

Avant-propos

Ce rapport est le résultat d'un projet de découverte de la recherche de quatre mois effectué dans le cadre du sujet fourni par M. Olivier Buffet, chercheur au Loria.
Le projet a été réalise pour l'équipe Maia (MAchines Intelligentes Autonomes), groupe de recherches centré sur les comportements décisionnels intelligents. Ses principaux champs de recherches sont les systèmes multi-agents et complexes ainsi que les systèmes décisionnels incertains.
N'ayant que peu d'expérience dans le domaine de l'Intelligence Artificielle mais étant très intéressé par cette dernière, nous avons choisi ce sujet afin d'en approfondir nos connaissances. Étant donné notre méconnaissance de la plupart des algorithmes de plus courts chemin, nous nous sommes demandés quelles en sont les principales différences.
Notre étude portant sur l'implémentation des algorithmes ainsi que de leurs comparaison au travers de batteries de tests était donc l'occasion d'en apprendre plus sur leur fonctionnement.

Table des matières

1 Introduction

1.1 État actuel du domaine de recherches et légitimité du sujet

Les algorithmes de plus court chemin appartiennent à la théorie des graphes et constituent un vaste champ de recherches de l'Intelligence Artificielle. Il existe de nombreux algorithmes qui ont été développés depuis les débuts de la Recherche Opérationnelle au début des années 1940, les plus connus étant Dijkstra et A*.

Cependant, bien que ces algorithmes soient connus et maîtrisés, il est difficile de déterminer *a priori* leur comportement sur différents types d'environnements, et donc de savoir lequel sera le plus efficace.

1.2 Présentation du fonctionnement global et de l'utilisation du logiciel

Le projet se propose donc de permettre de réaliser des séquences de tests de différents algorithmes sélectionnés par l'utilisateur sur des environnements différents générés aléatoirement.

1.2.1 Fonctionnement du projet

1.2.2 Les environnements

Les environnements correspondent aux graphes sur lesquels les algorithmes sont appliqués. Il existe deux types d'algorithmes : les algorithmes générés sous forme de grilles à N dimensions avec N > 1 (hypercubes) et des environnements générés de manière totalement aléatoires.

1.2.3 Les algorithmes

Plusieurs algorithmes ont été implémentés afin de tester leur efficacité sur différents types d'environnements de différentes tailles. Leur point de départ et d'arrivée sont définis aléatoirement. Les algorithmes parcourent le graphe en explorant ses noeuds par accès directs.

1.2.4 Lancer le projet

Afin de répondre à tout type de besoins, le jar exécutable du logiciel prend en paramètre plusieurs options permettant différents modes d'expérimentation.

En vue graphique

En lançant le logiciel avec la commande : *java -jar AlgoComparator.jar –view*, celui-ci se lance en mode graphique. Cependant cette option a été implémentée au début du développement, avant que nous nous rendions compte qu'elle ne nous était pas utile. Elle a donc été délaissée et n'est plus fonctionnelle.

En console avec dialogue utilisateur

En lançant le logiciel avec la commande : *java -jar AlgoComparator.jar –console*, celui-ci se lance en mode console et initie un dialogue avec l'utilisateur. Il lui permettra de créer un environnement, de sauvegarder la graine, de lancer une expérience et d'enregistrer les résultats dans un fichier de logs.

En console en mode de benchmarking

En lançant le logiciel avec la commande : *java -jar AlgoComparator.jar –bench*, celui-ci se lance en mode benchmarking. Il va charger la configuration dans le fichier bench.conf afin de déterminer la taille et le nombre de dimensions des environnements à générer. Un seul dialogue se lance avec l'utilisateur en début d'exécution pour lui demander les algorithmes qu'il veut lancer. Tous les résultats sont stockés dans le fichier "logs.txt".
Note : l'option –bench all permet de lancer un benchmarking sur tous les algorithmes existants et supprime la séquence de dialogue.

Tester la génération d'environnements

Etant donné que nous avons rencontré des difficultés lors de la génération des environnements durant une certaine période du développement, nous avons aussi ajouté une option permettant de lancer un benchmarking sur la création d'environnements de 10 à plusieurs millions de points. Cette option est accessible avec la commande *java -jar AlgoComparator.jar –test consenv*. Il est aussi possible d'avoir le test de l'affichage graphique d'un environnement en remplaçant l'option CONSENV par l'option GRAPHENV. Ceci peut-être pratique pour vérifier que la création des liens entre les points se fait correctement.

2 Matériel et Méthode

Nous allons maintenant voir les procédés mis en place pour le bon déroulement du projet et les outils développés afin de parvenir au résultat final.

2.1 Déroulement du projet

2.1.1 Son organisation

Afin d'avoir le meilleur suivi possible de l'avancement du projet, une réunion a été organisée avec l'encadrant toutes les deux semaines durant toute le période de développement du projet. En dehors de ces réunions, nous nous rencontrions afin de développer le projet en commun. Le reste du temps, nous avons programmé à distance grâce à l'utilitaire git.

2.1.2 Les technologies utilisées

Le projet a été réalisé en Java sous Eclipse.
Afin de travailler au mieux en équipe et de pouvoir gérer les versions de notre projet, nous avons utilisé le gestionnaire de version git.

2.2 Outils développés pour tester les algorithmes

Afin de tester les algorithmes, nous avons codé des outils nous permettant de tester l'efficacité des algorithmes que nous allons présenter ci-dessous.

2.2.1 Les environnements

Le premier et plus important de ces outils est le générateur d'environnements permettant de tester un algorithme sous plusieurs configurations possibles.

Présentation générale de la conception des environnements : listes d'adjacence

Tous les environnements sont conçus de la même façon. Ce sont des ensembles de places (représentées par N coordonnées pour N dimensions) reliées entre elles par des liens pondérés. Afin de représenter ces liens, nous possédons une liste d'adjacence associant à chacun des points la liste de ses successeurs. Chacune des dimensions possède une borne supérieure et inférieure.

Les environnements aléatoires

Au début du projet, nous voulions générer aléatoirement des graphes orientés. Cependant nos premiers échecs furent infructueux ; nos graphes n'étaient en effet jamais suffisamment connexes pour qu'une solution existe. Nous avons alors recherché un moyen de générer un environnement aléatoire, orienté et fortement connexes, mais ils semblerait que cela soit encore du domaine de la recherche. Ceci n'étant pas l'axe principal de notre projet, nous avons décidé d'abandonner ce type de graphe afin de nous concentrer sur lagénération de grilles.

Les grilles et hypercubes

Le principal type d'environnement utilisé sont donc les grilles à N dimensions. A partir des bornes données par l'utilisateur et le nombre de places appartenant à chaque dimension du graphe, les places sont générées de manière régulière sur chaque dimension. En parallèle de la création des points, chacun d'entre eux est relié à son voisin. On sait que dans un hypercube, tous les points ont pour voisins directs les points ayant exactement un coordonnées différente. A partir de là il est facile de générer le lien entre la place courante et tous les points qui ont été précédemment générés.
Les liens sont pondérés pseudo-aléatoirement à partir d'un RNG alimenté par une graine spécifique (dans notre cas, le timestamp).

Génération d'environnement par graine : l'objet Seed

Afin d'enregistrer facilement les données relatives à un graphe sans enregistrer la totalité des points (ce qui pourrait être une perte considérable de place dans le cas d'un environnement à plusieurs millions de points), nous enregistrons toutes les données nécessaires à la génération d'un environnement dans un fichier texte, ce qui nous permettra par la suite de recréer exactement le même environnement sans sauvegarder tous les points.

Les données enregistrées sont la graine (le timestamp lors de la première création de l'environnement), le nombre de points pour chacune des dimensions (nous sommes dans le cas des grilles et hypercubes) ainsi que la borne min et la borne inf des points (toutes les bornes sont considérées comme identiques, les hypercubes sont donc réguliers).

L'objet Seed de notre programme permet, à partir d'un fichier texte formaté, de reproduire à l'identique le graphe d'une environnement.

Les refactorisations du modèle des environnements

Étant donné qu'au début nous désirions créer des environnements aléatoires, l'ensemble du code de nos Environnements se basait sur les positions des places dans le graphe. Or de cette manière, des parcours supplémentaires étaient obligatoires afin de rechercher des places par leurs coordonnées, ce qui ralentissait considérablement notre code.

Cependant une fois les graphes aléatoires abandonnés, nous nous sommes rendus qu'il était bien plus facile et rapide de générer les graphes en grille à partir de leurs indices. Nous avons donc développé une seconde version plus efficace (étant donné que le nombre de dimensions et de points par dimension est connu à l'avance).

Cependant en effectuant des recherches, nous avons vu le principe de génération d'hypercubes de dimension N récursivement par fusion d'hypercubes de dimension N-1. Nous avons donc décidé d'adapter ce principe à nos grilles en N dimensions et à M points.

Voici les images exposant ce principe et dont nous nous sommes inspirés :

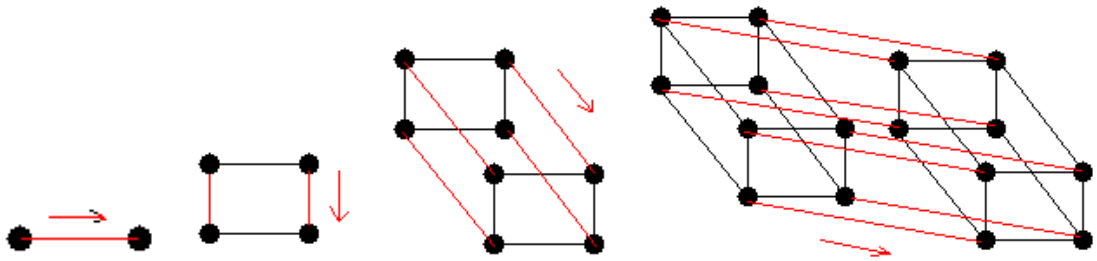

FIGURE 2.1 – Génération d'un hypercube en 4 dimensions

Nous avons donc refactorisé le code des Environnements afin d'appliquer cette modification, et la différence fut flagrante. Alors qu'il était impossible de générer un environnement d'un million de points dans la première version (même après plusieurs heures), le nouvel algorithme était capable de le faire en moins d'une dizaine de secondes.

Voici l'algorithme que nous avons implémenté :

Data: M : tableau qui associe à chaque dimension i son nombre de points, N : nombre de dimensions du graphe, borne_inf : borne inférieure de toutes les dimensions, borne_sup : borne supérieure de toutes les dimensions, i : compteur correspondant à la dimension courante, j : compteur correspondant au nombre de points de la dimension courante, graph : graph actuel que l'on fait évoluer
Result: Hypercube de N dimensions à M[i] points chacune
graph = new Point(nb_dim = N, coordonnées = {borne_min, ...}) //On crée le point extremum
i = 0
while *i < N* **do**
  //On calcule la distance entre les points de la dimension courante
  distance = calculerDist(M[i], borne_inf, borne_sup)
  //On duplique le graphe le nombre de fois qu'il faut
  for *j=0 ; j<M[i] ; j++* **do**
    tmp = dupliquer(graph)
    //On décale les coordonnées de la copie de la distance qui existe entre les points
    glisser_coordonnées(tmp, distance)
    copies_graph.push(tmp)
  end
  graph = fusion(graph, copies_graph) //On fusionne toutes les copies ensembles
end

Algorithm 1: Algorithme de génération d'une grille à N dimensions

Voici un tableau comparatif des différentes versions de l'algorithme de création de cartes à N dimensions :

Version	**10^2**			**10^4**			**10^6**			**10^7**		
	2D	**3D**	**4D**	**2D**	**3D**	**4D**	**2D**	**3D**	**4D**	**2D**	**3D**	**4D**
$1^{ère}$ version	5	-	-	10^7	10^8	10^8	$+\infty$			$+\infty$		
$2^{ème}$ version	2	-	-	34	36	41	16 864	24 247	32 589	53 789	74 252	83 736
$3^{ème}$ version	2	-	-	13	16	23	368	512	746	14 713	21 697	37 250

Table 2.1 – Tableau récapitulatif des benchmarkings

2.2.2 Les différents algorithmes implémentés

2.2.3 Objet Evaluation et fichiers de log

Le second outil nous permettant de comparer les algorithmes entre eux sont les objets Evaluation récoltant les données relatives à l'exécution de l'algorithme. Ces données sont les suivantes :

- Le nombre de nœuds "visités" avant de trouver la première solution
- Le nombre total de nœuds "visités".
- Le nombre de nœuds "explorés".
- Le nombre de solutions trouvées par l'algorithme.
- La liste du coût pour toutes les solutions trouvées.

— Le nombre de nœuds appartenant au chemin de chacune des solutions.
— La liste du nombre de nœuds "visités" pour chaque solution.

Ce sont sur elles que se baseront les comparaisons entre les algorithmes.

2.2.4 Vue graphique

Lors de la première phase de développement, nous avons réalisé une interface graphique permettant à l'utilisateur de modifier toutes les options possibles et de lancer le déroulement d'un algorithme avec une interface graphique. Cependant cela n'était que fonctionnel pour les environnements en deux dimensions et ne correspondaient pas à nos besoins réels, nous avons donc abandonné le développement de cette fonctionnalité qui est désormais obsolète.

```
void CEquation::IniParser()
{
  if (!pP){ //if not already initialized...
    pP = new mu::Parser;

    pP->DefineOprt("%", CEquation::Mod, 6); //deprecated
    pP->DefineFun("mod", &CEquation::Mod, false);
    pP->DefineOprt("&", AND, 1); //DEPRECATED
    pP->DefineOprt("and", AND, 1);
    pP->DefineOprt("|", OR, 1); //DEPRECATED
    pP->DefineOprt("or", OR, 1);
    pP->DefineOprt("xor", XOR, 1);
    pP->DefineInfixOprt("!", NOT);
    pP->DefineFun("floor", &CEquation::Floor, false);
    pP->DefineFun("ceil", &CEquation::Ceil, false);
    pP->DefineFun("abs", &CEquation::Abs, false);
    pP->DefineFun("rand", &CEquation::Rand, false);
    pP->DefineFun("tex", &CEquation::Tex, false);

    pP->DefineVar("x", &XVar);
    pP->DefineVar("y", &YVar);
    pP->DefineVar("z", &ZVar);
  }
}
```

Listing 2.1 – Premier Exemple

Il est également possible d'afficher du code directement depuis un fichier source, le résultat de cette opération est visible dans le listing 2.2

```
void CEquation::IniParser()
{
  if (!pP){ //if not already initialized...
    pP = new mu::Parser;

    pP->DefineOprt("%", CEquation::Mod, 6); //deprecated
    pP->DefineFun("mod", &CEquation::Mod, false);
    pP->DefineOprt("&", AND, 1); //DEPRECATED
    pP->DefineOprt("and", AND, 1);
    pP->DefineOprt("|", OR, 1); //DEPRECATED
    pP->DefineOprt("or", OR, 1);
    pP->DefineOprt("xor", XOR, 1);
    pP->DefineInfixOprt("!", NOT);
    pP->DefineFun("floor", &CEquation::Floor, false);
```

```
15      pP->DefineFun("ceil", &CEquation::Ceil, false);
16      pP->DefineFun("abs", &CEquation::Abs, false);
17      pP->DefineFun("rand", &CEquation::Rand, false);
18      pP->DefineFun("tex", &CEquation::Tex, false);
19
20      pP->DefineVar("x", &XVar);
21      pP->DefineVar("y", &YVar);
22      pP->DefineVar("z", &ZVar);
23    }
24  }
```

Listing 2.2 – Affichage depuis le fichier source

3 Résultats et interprétation

3.1 Exécution des algorithmes sur les environnements

3.1.1 Présentation des résultats et comparaison

3.1.2 Interprétation des résultats

4 Discussion globale

Liste des illustrations

Liste des tableaux

Listings

Glossaire

Annexes

A Première Annexe

B Seconde Annexe

Résumé

Le sujet en bref. Le sujet qui a été choisi porte sur la réalisation d'un logiciel de benchmarking permettant de tester et de comparer différents algorithmes de recherche de plus court chemin sur des environnements diversifiés générés aléatoirement sur des séquences de tests paramétrées dans un fichier de configuration.

Les fonctionnalités implémentées.

Les résultats.

Interprétation. **Mots-clés :** algorithmes de plus court chemin, théorie des graphes, environnements aléatoires, benchmarking, interface utilisateur, graines.