Nantes Université — UFR Sciences et Techniques
Licence informatique parcours “mathématiques-informatique”
Année académique 2022-2023

X32I130
PROJET D'INFORMATIQUE SCIENTIFIQUE

Prof. Dr. Hab. Xavier GANDIBLEUX

Présentation du sujet d'étude,
consignes pour conduire vos travaux,
livrables attendus

Mots-clef : optimisation – recherche opérationnelle – intelligence artificielle – code open-source – langage Julia – ingénierie des performances – initiation à la recherche – expérimentation numérique.

DESCRIPTIF DE L'ENSEIGNEMENT 1

1.1 MOTIVATIONS

Cet enseignement s'adresse aux étudiants qui seront amenés à travailler dans un environnement où le développement logiciel, bien que n'étant pas nécessairement la finalité du métier, est omniprésent. Cela concerne par exemple les activités de recherche et développement dans les grandes entreprises et laboratoires publics, entreprises spécialisées dans le domaine de l'informatique et des nouvelles technologies, startup spécialisées, tous actifs dans de très nombreux domaines scientifiques : ingénierie, transport, production, santé, énergie, développement durable, robotique, data science, services, etc.

L'utilisation avancée et le développement des logiciels s'inscrivant dans un contexte scientifique requiert à l'évidence des bases solides dans les disciplines concernées mais également un haut niveau de maîtrise en programmation car il s'agit soit de développer une solution logicielle durable, soit d'intégrer de nouvelles fonctionnalités dans un code existant reposant sur des concepts informatiques de nature scientifique et très souvent situés à la pointe des nouvelles connaissances.

Les logiciels orientés scientifiques se différencient des logiciels par exemple orientés gestion par des exigences de performance et une complexité des méthodes mises en oeuvre. Ce que l'on attend d'un langage pour le calcul scientifique est assez précis et ne correspond pas à une utilisation généraliste. En effet, on veut écrire des mathématiques simplement, mais aussi avoir du code extrêmement efficace à l'exécution et la possibilité de le paralléliser (sur une machine ou un superordinateur), le tout avec les besoins usuels de la programmation moderne.

C'est particulièrement vrai dans des domaines de la recherche opérationnelle et de l'intelligence artificielle, domaines se déclinant dans de nombreuses activités du quotidien (allant de la gestion de la production d'électricité, qui vient avec des masses de données très volumineuses, à la conduite de véhicules autonomes, qui est au contact de systèmes automatisés complexes, en passant par la planification du personnel hospitalier en service d'urgence, qui vient avec son volet d'événements difficilement prévisibles). Pour ces applications souvent conséquentes, en plus du critère de performances, on attend du langage qu'il soit stable, normalisé, présente des garanties de pérennité afin qu'un code soit toujours exécutable dans 20 ans.

Par le passé ils ont été développés en Fortran, en C, ou encore en assembleur. Ils sont aujourd'hui développés et mis en production dans les environnements industriels en C++ et Java afin de bénéficier de fonctionnalités avancées comme une couche objet plus riche et apportant des éléments de sécurité au logiciel. Ces bénéfices viennent avec une rigueur faisant que le développement logiciel sous ces deux langages peut apparaître rigide et astreignant notamment dans des phases de prototypage de nouveaux algorithmes. On aimerait disposer de l'expressivité du langage Python, ses facilités de mise en oeuvre et son cycle de développement rapide sans perdre les performances et la pérennité des codes du langage C++, deux qualités inhérentes du langage de programmation Julia (`https://julialang.org/`).

Distribué en open-source sous licence MIT depuis 2012, Julia est un langage de programmation répondant aux besoins du développement d'applications scientifiques. On peut constater son essor dans des communautés telles que l'intelligence artificielle, l'apprentissage automatique, l'optimisation, la recherche opérationnelle, le calcul scientifique, les mathématiques appliquées. Déjà bien installée dans les pays anglo-saxons, sa diffusion progresse en France notamment par des enseignements dans plusieurs universités, sa présence dans des organismes publics de recherche (comme le CEA, Commissariat à l'énergie atomique et aux énergies alternatives). Aussi des entreprises proposent des moteurs d'optimisation en Julia (exemple du framework de "branch-and-price-and-cut" chez `https://atoptima.fr/`). Des offres de stage et d'emploi avec une expérience en Julia commencent à circuler.

Enfin, étant dûment inscrit dans le courant de l'open-source, tant pour des ressources pédagogiques[1], de collections de données destinées à évaluer de nouveaux algorithmes[2], que d'applications informatiques[3], j'ai souhaité y inscrire également l'esprit de cet enseignement. Déjà avec le choix du langage de programmation pris comme support, mais également en l'appuyant sur des cours de qualités disponibles en open-access et la réalisation des productions suivant les codes de conduite de la communauté scientifique.

[1] depuis 2021, nous proposons sur le site JULIANANTES que nous maintenons (`https://julialang.univ-nantes.fr/`) un cours sur ce langage de programmation.

[2] depuis 1998, nous maintenons en ligne VOPTLIB (`https://github.com/vOptSolver/vOptLib`), une librairie d'instances numériques servant à l'évaluation d'algorithmes d'optimisation multi-objectif.

[3] depuis 2015, nous développons VOPTSOLVER (`https://github.com/vOptSolver`), un écosystème offrant des fonctionnalités de modélisation et de résolution de problèmes d'optimisation multi-objectif.

1.2 ATTENDUS

Le projet d'informatique scientifique consiste en une activité d'étude, de conception et réalisation d'une solution logicielle open-source alliant mathématiques et informatique. Supervisé par un enseignant, il est réalisé individuellement durant le second quadrimestre de la licence 3 sur une thématique qui aborde un domaine de l'informatique scientifique. Chaque étudiant réalisera une solution logicielle documentée, qu'il défendra à l'occasion d'un exposé final.

Ce projet suppose acquises les notions d'algorithmique, de programmation, de mathématiques visées par les cours faisant partie du programme de licence informatique parcours maths-informatique. Son objectif est double :

1. conduire l'étudiant à mobiliser un ensemble de connaissances dans les divers enseignements de licence (algèbre, analyse, probabilités, statistiques, analyse numérique, optimisation, recherche opérationnelle, algorithmes, programmation, logique, etc.);
2. apprendre à l'étudiant à piloter un projet d'une certaine envergure sur plusieurs mois, surmonter les contraintes, faire preuve d'autonomie, de curiosité scientifique propre à la recherche et innovation, s'attaquer à du concret.

En fonction du sujet du projet (qui varie d'une année à l'autre), le travail couvrira plusieurs activités, comme la revue des fondements théoriques (énoncé et démonstration de propriétés, théorèmes, etc., mis en œuvre), la spécification, la conception, l'implémentation, le testing, la rédaction d'une documentation. On s'attendra à voir la mobilisation de différentes compétences comme la formalisation et la modélisation, les choix algorithmiques et de structures de données, le développement et l'analyse des résultats numériques.

A l'image d'un donneur d'ordre qui exprime un besoin et des moyens attendus pour y répondre, vous serez guidés vers des choix algorithmiques mais ils ne seront pas complètement décrits ici. Vous les trouverez dans vos cours ou ouvrages scientifiques. Il est attendu que vous soyez force de proposition sur les choix de mise en oeuvre les plus appropriés et les plus efficaces compte tenu des situations à traiter.

Les codes informatiques seront écrits dans le langage de programmation Julia en se donnant la possibilité d'utiliser lorsque c'est justifié des codes open-source existants (comme des librairies spécifiques aux calculs, à la visualisation, à la simulation, etc.) écrits tant en Julia que dans d'autres langages de programmation. Tout code repris comportera la mention de son origine et il sera de votre responsabilité d'apporter toutes les réponses aux questions qui pourraient vous être posées quant à ce code.

L'efficacité de la solution logicielle produite et la pertinence de l'analyse des résultats seront mis en avant. L'objectif clairement affiché dans ce projet est de produire un **travail personnel** de qualité, réfléchi, aux arguments pesés. C'est un **travail d'étude, de recherche et de développement**. Il ne s'agit pas de produire juste une réponse "qui marche" au cahier des charges. Les problèmes étudiés sont classiques, les implémentations demandées existent et nous les connaissons. Il n'y aucun intérêt à rapporter comme réponse des codes existants qui délivrent les solutions attendues.

Face à cela on appréciera un travail peut-être moins parfait, moins abouti, mais dans dans lequel on mesure une prise de position personnelle et assumée, une progression de votre aisance sur les sujets. Vous serez questionnable sur la totalité de vos productions, des réponses argumentées seront attendues. Le projet représente un travail étudiant de 3ECTS/75h.

1.3 ORGANISATION

L'enseignement est organisé sous forme de travaux dirigés (3 créneaux de 1h20 ventilés en 2 séances, une de lancement, une à mi-parcours), un encadrement du projet à raison de 1h par étudiant sous la forme d'un tutorat proposé sur un créneau annoncé, une défense finale qui déterminera la note.

1.3.1 Jalons

• Les trois séances fixées

1. Première séance :
   Présentation de l'enseignement, du sujet, des consignes, de l'organisation;
2. Mi-parcours :
   Relevé des productions d'étapes et présentation de la seconde partie du travail;
3. Fin quadrimestre :
   Remise des livrables et défense durant laquelle vous aurez à présenter votre travail et répondre aux questions.

• Les séances tutorées

- Examen de questions qui se présentent à vous et tentative d'y apporter des éléments de réponse, des directions à explorer.

1.3.2 Les lots de travaux à mener

Partie 1 (temps estimé : 40h)

- Prise en main du langage Julia
- Regard fouillé sur des questions relatives à la performance d'un code Julia
- Conception de votre réponse à la problématique de l'étude
- Mise en oeuvre de votre réponse sous la forme d'une solution logicielle
- Validation technique appuyée par l'élaboration de tests
- Usage impératif d'un dépôt Github hébergeant codes, données, tests, doc., etc.
- Rendu sous Github à mi-parcours qui déverrouille l'accès à la seconde partie du travail et qui sera évalué sur une instance test "mystère".

  Attention, un travail non rendu, non conforme ou non opérationnel conduira à une note définitive de 00/20 au module.

Partie 2 (temps estimé : 35h)

- Adaptation de votre production au regard de l'évolution qui sera dévoilée à mi-parcours

- Recueil des données dans la collection indiquée pour vos expérimentations numériques
- Campagne rigoureuse d'expérimentations numériques des algorithmes mis en œuvre
- Analyse et discussions des différents approches et versions d'algorithmes développés et évalués au regard des expérimentations numériques
- Elaboration du matériel pour la défense (modalités exposées à mi-parcours)

1.3.3 Consignes communes aux livrables

- Solution logicielle réalisée en Julia
- Documentation préparée avec LaTeX, markdown, etc.
- Mise à disposition de l'ensemble sur un dépôt github
- Organisation structurée en répertoires sur github (src, dat, res, doc), un readme indiquant les éléments essentiels

PREMIÈRE PARTIE DES TRAVAUX

2

2.1 LANGAGE DE PROGRAMMATION JULIA

Pour un public de niveau licence 3, venir à la programmation en Julia ne présente aucune difficulté. Pour vous accompagner dans cet apprentissage, vous avez a votre disposition un cours qui est destiné à un public débutant dans l'implémentation d'un code informatique. Même si la plupart des notions présentées vous sont familières dans un autre langage de programmation, ce cours a l'avantage de visiter les notions de base de Julia mais aussi de JuMP, langage de modélisation algébrique qui sera utilisé dans le cours de "recherche opérationnelle" durant ce quadrimestre. Le travail que vous fournirez pour vous former à Julia vous sera utile à plusieurs égards donc[1].

Le cours mis à votre disposition est disponible à l'URL suivante : `https://github.com/xgandibleux/Linz2021-2022` Il est composé de :

- 9 leçons présentées sous la forme de slides;
  elles sont toutes à travailler
- 8 séries d'exercices sous la forme de notebooks;
  elles sont facultatives.

Le reste du matériel mis à disposition sur cette URL (une vidéo décrivant l'installation du solveur MIP Gurobi, cas d'étude, projet, code, instances) est hors propos pour les besoins de ce projet. Les premières leçons vous accompagnent dans les ressources à connaître, l'installation en local, etc. Ce cours ne fait qu'effleurer les structures de données, il vous sera utile donc de compléter votre prise en main du langage avec d'autres ressources qui sont nombreuses sur le web. Un point de départ incontournable est `https://julialang.org/`.

TEMPS ESTIMÉ : 10h

[1] A en croire plusieurs experts qui suivent l'évolution des langages de programmation, c'est le moment de se former à Julia maintenant car, tant sa montée en puissance dans la communauté de data science, que son utilisation en interne dans des enseignes influentes de l'informatique (par exemple chez Amazon, Google, IBM, Apple, Microsoft, etc.) fait qu'il est attendu pour être incontournable au cours des 10 années à venir.

2.2 INGÉNIERIE DES PERFORMANCES

Le second volet de prise en main de Julia a pour objectif d'examiner différentes questions liées à la mise en oeuvre d'un code efficace avec un langage de haut niveau.

L'efficacité d'une solution logicielle passe d'abord par le choix judicieux des traitements à réaliser et des structures de données mises en œuvre, l'ensemble se mesurant par les complexités temporelles et spatiales qui en découlent (cela vous est connu, voir vos cours et travaux en ASD2 et ASD3).

Un langage de haut-niveau tel que Julia facilite grandement l'activité de codage notamment sur des aspects comme le typage ou encore la gestion de la mémoire, permettant ainsi de réaliser rapidement un prototype. Cependant le resultat d'un prototypage rapide peut s'avérer antinomique avec la recherche de la performance. Il convient d'être conscient des choix mis en œuvre pour de pas dévier vers un code aux performances pénalisées. C'est ici qu'intervient l'expertise du concepteur.

Il n'y a pas de miracle; programmer efficacement avec un langage de programmation de haut-niveau dont la totalité des mécanismes n'est pas à charge du concepteur (par opposition à C/C++ par exemple) va de pair avec une connaissance profonde du dit langage, afin d'éviter par exemple des opérations de transtypage cachées, des allocations de mémoire masquées, des déclenchements abusifs du garbage collector, etc.

A cette fin plusieurs sujets sont discutés dans les ressources suivantes mises à votre disposition :

- un cours donné en 2021 dans les séminaires "DataIA" du CEA, reprend la présentation du langage Julia mais en apportant différents focus sur des questions de performance consécutives aux choix d'implémentation;
  https://indico.in2p3.fr/event/17858/page/2706-julia
- un cours "Performance Engineering of Software Systems" donné en 2018 au Massachusetts Institute of Technology (MIT) met l'accent sur l'ingénierie des performances et en particulier, le dernier chapitre aborde précisément ces aspects avec Julia comparativement à Python et C.
  https://ocw.mit.edu/courses/6-172-performance-engineering-of-software-systems-fall-2018/resources/lecture-23-high-performance-in-dynamic-languages/

Les liens vers ces deux ressources (la totalité du cours du CEA et le dernier chapitre du cours du MIT) à parcourir sont repris sur madoc.

Pour tirer pleinement partie de ces ressources, il est indiqué de les aborder dans cet ordre après avoir acquis un premier niveau d'aisance sur les bases du langage. Il sera intéressant de cerner des différences marquantes du langage Julia notamment vis-à-vis des langages qui vous sont familiers, notamment C++ et java.

TEMPS ESTIMÉ : 10h

2.3 THÈME DE L'ÉTUDE ET SES APPLICATIONS

Le thème de cette année aborde le problème de *la recherche d'un chemin* dans un environnement contraint, appelé *pathfind* en anglais. Il fera appel pour l'essentiel à des notions vues en ASD2, ASD3 et RO. Il sera regardé sous le prisme de l'optimisation d'un critère (distance la plus courte, trajet le plus rapide, nombre d'opérations réalisé le plus bas, moindre espace mémoire requis pour résoudre la tâche, etc.).

Ce problème est central tant en recherche opérationnelle qu'en intelligence artificielle[2]. Les aspects abordés dans cet étude seront approfondis en master dans les enseignements comme "graphes et réseaux", "optimisation discrète et combinatoire", "métaheuristiques", "programmation par contraintes", notamment autour de l'algorithme de "branch-and-bound" qui est fondamental en optimisation discrète (voir fin du cours "recherche opérationnelle" pour un premier contact). Les applications réelles de ce problème sont vastes; citons par exemple les situations suivantes rencontrées dans :

- **les véhicules autonomes** (Figure 2.1 haut-gauche) :
  La mission qui consiste à planifier le chemin d'un véhicule autonome (drone, voiture, robot, etc.) vise à déterminer comment le véhicule peut naviguer dans un espace contenant des obstacles connus et inconnus en minimisant les collisions, le temps moyen pour accomplir le chemin ou encore maximiser la distance de sécurité entre véhicules quand plusieurs unités mobiles sont considérées (voir MAPF). Cette première situation touche des domaines comme la logistique, le transport, la robotique.

- **les jeux vidéos** (Figure 2.1 haut-droite) :
  Des stratégies basées sur pathfinding sont habituellement utilisées au coeur des mouvements de personnages en temps réel dans les jeux vidéos. Par exemple, comment éviter des obstacles habilement et comment trouver le chemin le plus efficace sur différents terrains.

- **la logistique d'entrepôt** (Figure 2.1 bas-gauche) :
  Le *Multi-agent path finding (MAPF)* est le problème dans lequel on déplace un ensemble d'agents depuis leur point de départ individuel a leur point d'arrivée sans collision. Ce problème se rencontre dans les grands entrepôts de biens (style méga dépôt Amazon) pour la préparation de commande. On associe un agent à un drone terrestre autonome chargé d'aller chercher un objet dans un rayonnage.

- **la circulation de trains** (Figure 2.1 bas-droite) :
  En associant un train à un agent qui occupe plusieurs ressources d'infrastructure (la tête de train sur la ressource courante, le reste du train jusqu'à sa queue sur les k précédentes ressources), le problème *Multi-Train Path Finding (MTPF)* est un cas particulier du MAPF où cherche à établir un plan de circulation de trains sans collision. Ce problème trouve tout son intérêt en situation d'incident dans lequel le plan de circulation initial doit être modifié en temps réel alors que le trafic est dense et le réseau ferroviaire complexe.

[2] A la naissance de l'intelligence artificielle, les techniques de résolution de problèmes par *recherche heuristique dans des graphes d'états* apparaissent comme une des branches maîtresses de la jeune discipline, comme en témoigne l'ouvrage de référence de Nilsson en 1971.

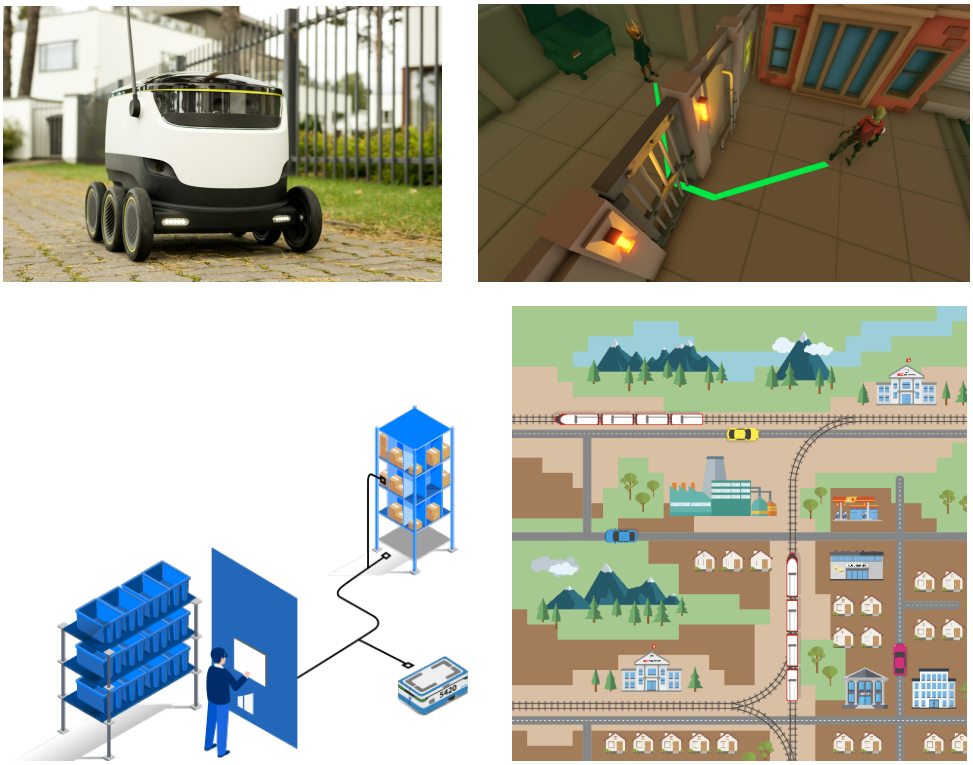

Figure 2.1: Illustration d'applications du pathfind dans quatre situations. Haut-gauche : mission en cours d'un drone livreur (photo : Starship); haut-droite : déplacement jusqu'à une cible dans les jeux vidéo; bas-gauche : illustration du système de stockage dynamique dans un entrepot (photo : scallog); bas-droite : illustration du simulateur de circulation de trains "fatland" (photo : SBB-DB-SNCF).

2.4 APPROCHES ET ALGORITHMES DE RÉSOLUTION

Dans une première approche la recherche de plus court chemin sur une carte sera considérée dans une situation de référence simple. Elle sera composée (voir figure 2.2-gauche) :

- d'une carte composant la zone de recherche;
- d'une décomposition de la carte en m lignes et n colonnes formant ainsi des cases élémentaires;
- de cases correspondant à des lieux visitables représentés par les carrés blancs, vert, rouge et oranges;
- d'un point de départ donné représenté par le carré vert;
- d'un point d'arrivée donné représenté par le carré rouge;
- d'obstacles infranchissables représentés par des carrés gris;
- de zones dont le franchissement est pénalisé, représenté par des carrés oranges;

et quatre déplacements élémentaires, Nord/Sud/Est/Ouest, sont considérés (figure 2.2 droite), un déplacement sur une diagonale étant non autorisé.

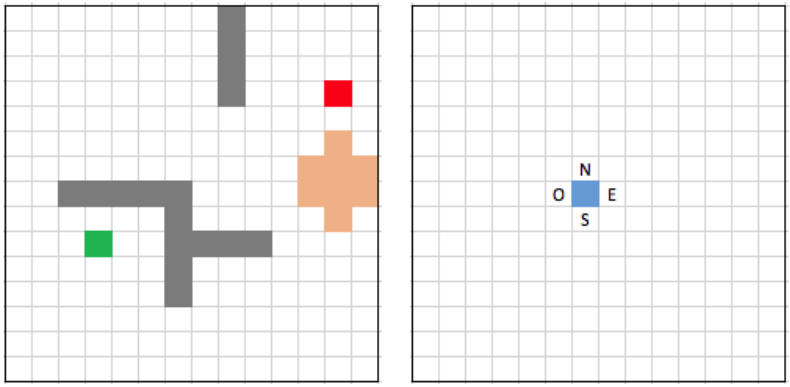

Figure 2.2: Gauche : illustration d'une situation type de recherche de plus court chemin sur une carte. Le carré vert est le départ, le carré rouge est l'arrivée, les carrés gris des obstacles, les carrés oranges sont des zones pénalisantes, les carrés blanc des lieux non-encore visités. Droite : illustration des quatre déplacements, Nord/Sud/Est/Ouest, autorisés depuis une case donnée représentée par le carré bleu.

Trois approches sont envisageables pour apporter une solution au pathfinding.

2.4.1 Approche “recherche aveugle”

Une *recherche en largeur d'abord* est une réponse possible avec par exemple l'application de *l'algorithme flood fill*.

Considérant les quatre déplacements élémentaires d'une case donnée à sa case adjacente, l'algorithme inonde la carte itérativement jusqu'à atteindre la case correspondant à l'arrivée, si celle-ci est atteignable depuis la case de départ compte-tenu des obstacles. Partant d'un exemple inspiré de [1] représenté par la figure 2.3 (gauche, situation de départ; droite, chemin trouvé), l'application de l'algorithme flood fill sur l'exemple est illustré sur la figure 2.4.

La distance de Manhattan (distance associée à la norme L_1) permet de calculer la longueur du chemin obtenu et, en ayant pris le soin de garder trace des déplacements, un *backtracking* (traduit en français par “retour sur trace” ou “retour arrière”) permet d'extraire le chemin obtenu. Pour cet exemple, le chemin est de longueur 4; il est optimal et il correspond aux cases visualisées en jaune sur la figure 2.3 droite.

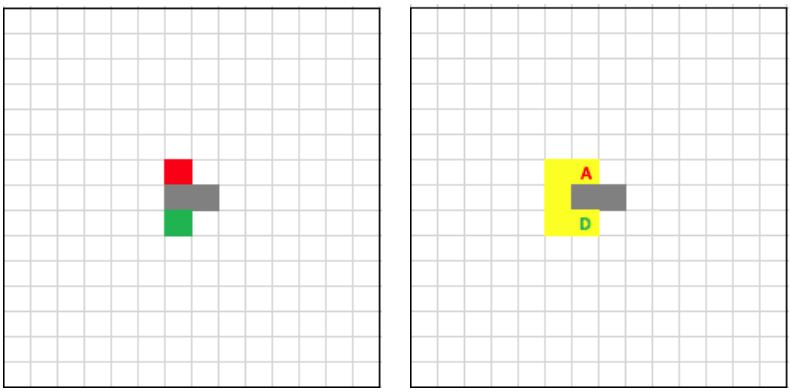

Figure 2.3: Gauche : illustration d'une situation initiale de recherche de plus court chemin sur une carte (exemple inspiré de [1]). Le carré vert est le départ, le carré rouge est l'arrivée, les carrés gris des obstacles, les carrés blanc des lieux non-encore visités. Droite : Le chemin optimal est ici de longueur 4 et il correspond aux cases visualisées en jaune.

Cet algorithme ne permet pas d'exploiter une situation dans laquelle la valeur associée à la transition d'une case i à une case j n'est pas unique. Cela peut correspondre à (a) une zône plus lente à traverser (boue, sable) dans un jeu vidéo, (b) une zône plus encombrée par d'autres drones en mouvement qu'il ne faut pas collisionner en robotique, ou encore (c) par des parties de l'infrastructure ferroviaire présentant une probabilité d'aléas fréquents et donc pouvant engendrer des retards en transport ferroviaire.

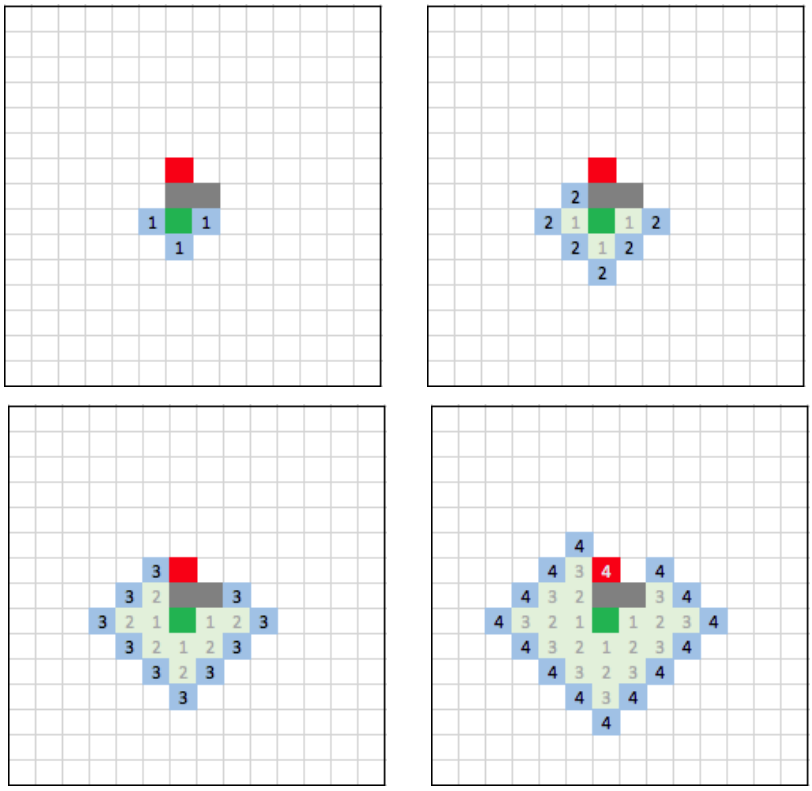

Figure 2.4: Approche “recherche aveugle” du pathfind avec l’algorithme flood fill. Les numéros indiquent les itérations de l’algorithme, les cases bleu-pâle sont les dernières cases découvertes, les cases vert-pâle sont les cases visitées lors des précédentes itérations. Le nombre d’opérations réalisées est $1 \times 4 + 3 \times 4 + 6 \times 4 + \max(10 \times 4) \leq 80$

2.4.2 Approche “recherche informée optimale”

Pour cette seconde approche, la carte de la figure 2.3 est reprise, à laquelle une zone pénalisante représententée par des cases oranges est ajoutée (figure 2.5). Les données du problème sont les mêmes que précédemment, avec en plus la connaissance du coût de transition entre 2 cases visitables adjacentes.

Lorsque tous les coûts sont non-négatifs, *l'algorithme de Dijkstra* est une réponse possible pour calculer le chemin optimal de longueur minimale depuis la case de départ jusqu'à la case d'arrivée, si celui-ci existe.

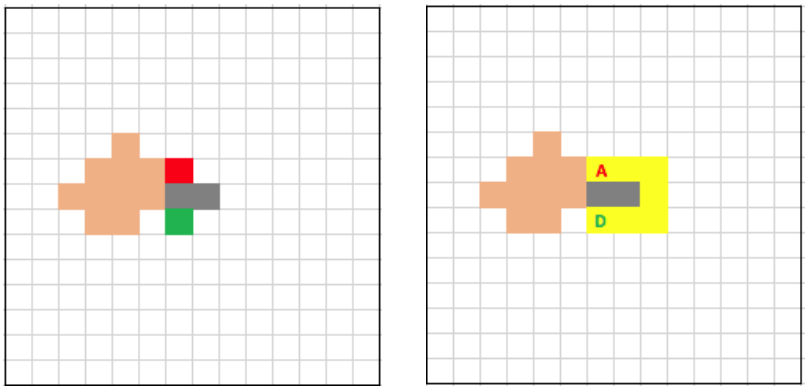

Figure 2.5: Gauche : illustration d'une situation initiale de recherche de plus court chemin sur une carte présentant une zone pénalisante (représentée en orange). Droite : Le chemin optimal obtenu avec l'algorithme de Dijstra est de longueur 6 et il correspond aux cases visualisées en jaune.

Il fonctionne de la manière suivante. La carte sera vue comme un graphe pondéré où chaque sommet correspond à une case visitable, une arête correspond à une transition N/S/E/O valuée par un poids non-négatif (figure 2.6). Partant de la case de départ, l'algorithme calcule le plus court chemin vers tous les sommets atteignables (figure 2.7). S'agissant du pathfind, l'algorithme de Dijstra peut être arrêté lorsque un chemin optimal est connu entre la case départ et la case d'arrivée.

L'exécution de l'algorithme de Dijstra peut être représenté sous la forme d'un arbre. Le principe de l'algorithme est de développer la position qui a le plus petit chemin déjà parcouru. Associons à chaque position la longueur du chemin déjà parcouru que nous appelons g.

Prenons une carte très simple où tous les coûts de transition valent 1 (figure 2.8) et déroulons l'algorithme. Pour la position de départ D, $g = 0$. L'algorithme marque la position de départ et développe vers les cases adjacentes visitables. Les fils de la position de départ valent $g = 1$. L'algorithme se poursuit en visitant la position non encore développée qui a un g minimal. Sur cet exemple toutes les positions aux feuilles (cad non encore développées) présentent $g = 1$. On choisit la première feuille (de gauche à droite) et on la développe, sans visiter des positions déjà visitées. L'algorithme continue ainsi jusqu'à visiter la case d'arrivée et à s'assurer que toutes les feuilles de l'arbre ont un g supérieur ou égal au g de la case d'arrivée. La figure 2.9 illustre l'état de l'arbre à un moment donné. A titre d'exercice, vous pouvez continuer le développement et vérifier qu'il trouve bien le plus court chemin.

L'algorithme de Dijstra ne permet pas d'exploiter une information telle que la connaissance a priori de la localisation de la case recherchée (coordonnées x,y sur la carte;

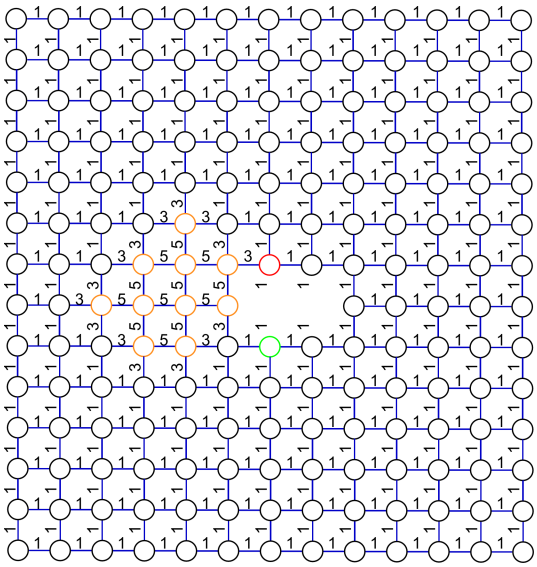

Figure 2.6: Une représentation du graphe correspondant à l'exemple de la figure 2.5. Les sommets correspondent aux cases visitables, les numéros sur les arêtes indiquent une valuation possible pour cet exemple, exprimant le "coût" de la transition entre 2 sommets.

coordonnées GPS; etc.). Or une telle information peut être avantageusement utilisée pour guider la recherche préférablement en direction de la case recherchée.

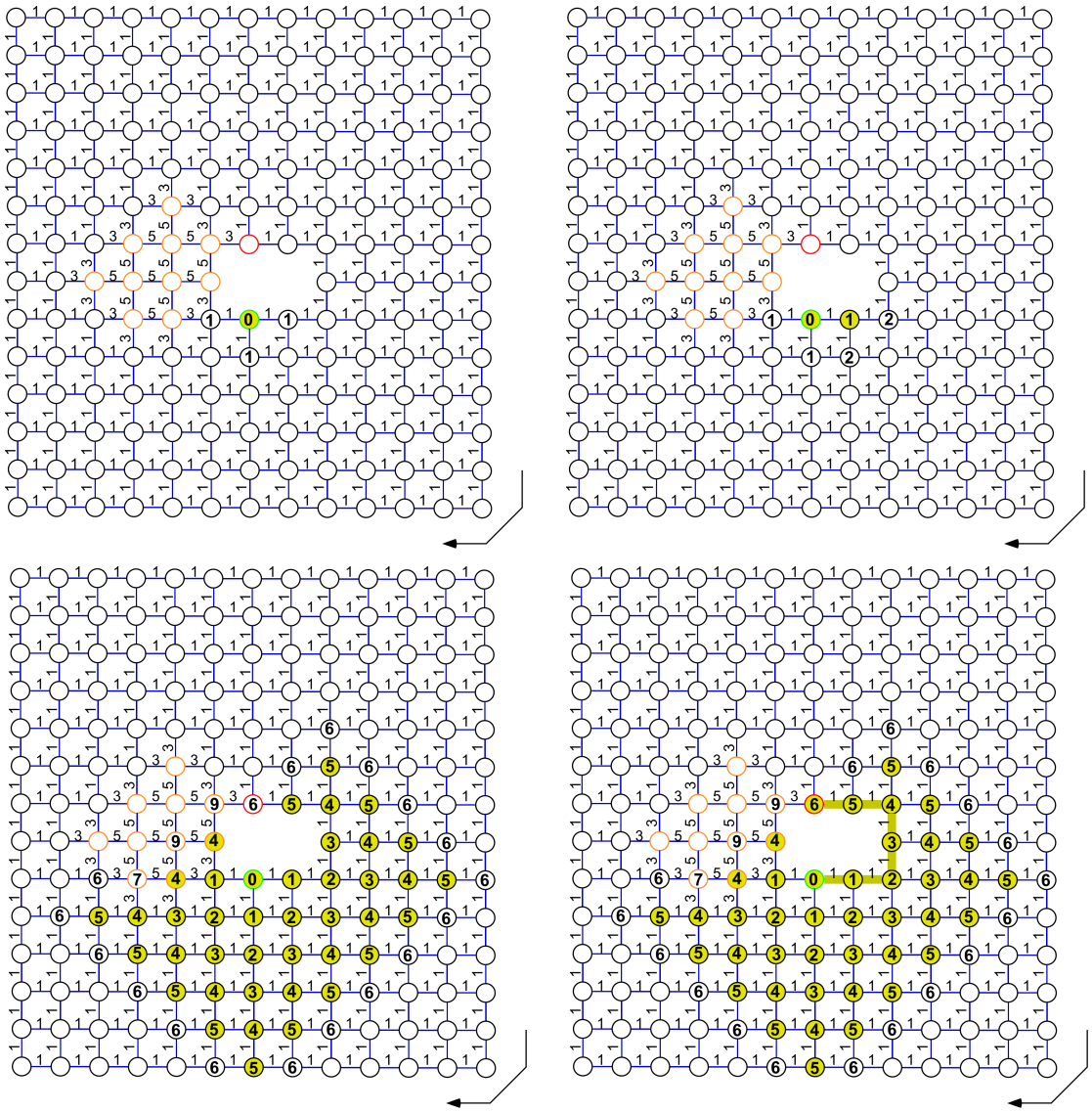

Figure 2.7: Approche “recherche informée optimale” du pathfind avec l’algorithme de Dijstra. Haut-gauche : le sommet de départ est sélectionné (marqué en jaune fluo), le coût du chemin optimal pour l’atteindre vaut zéro (chiffre en gras). Les coûts pour atteindre les sommets adjacents sont calculés (chiffre en gras), ils valent 1. Haut-droite : le sommet suivant est choisi, et quand plusieurs sommets sont candidats, ils sont pris dans cet exemple en suivant le sens de rotation horaire. Un nouveau sommet (de valeur 1) est marqué (jaune fluo), il correspond à un chemin optimal depuis la case départ jusqu’à lui. Les coûts pour les sommets adjacents sont calculés (chiffre en gras), et valent 2 au plus. Bas-gauche : plusieurs itérations de l’algorithme ont été réalisées. Le sommet d’arrivée a reçu une première valuation (valeur 6) mais il n’est pas encore marqué. Cette valeur est encore susceptible d’évoluer. Bas-droite : le sommet d’arrivée est sélectionné et est donc marqué (jaune fluo). Un premier chemin optimal (représenté en jaune fluo) a été trouvé, de valeur 6. L’algorithme peut donc être stoppé.

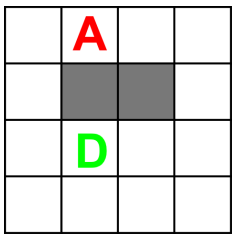

Figure 2.8: Une carte 4×4 très simple.

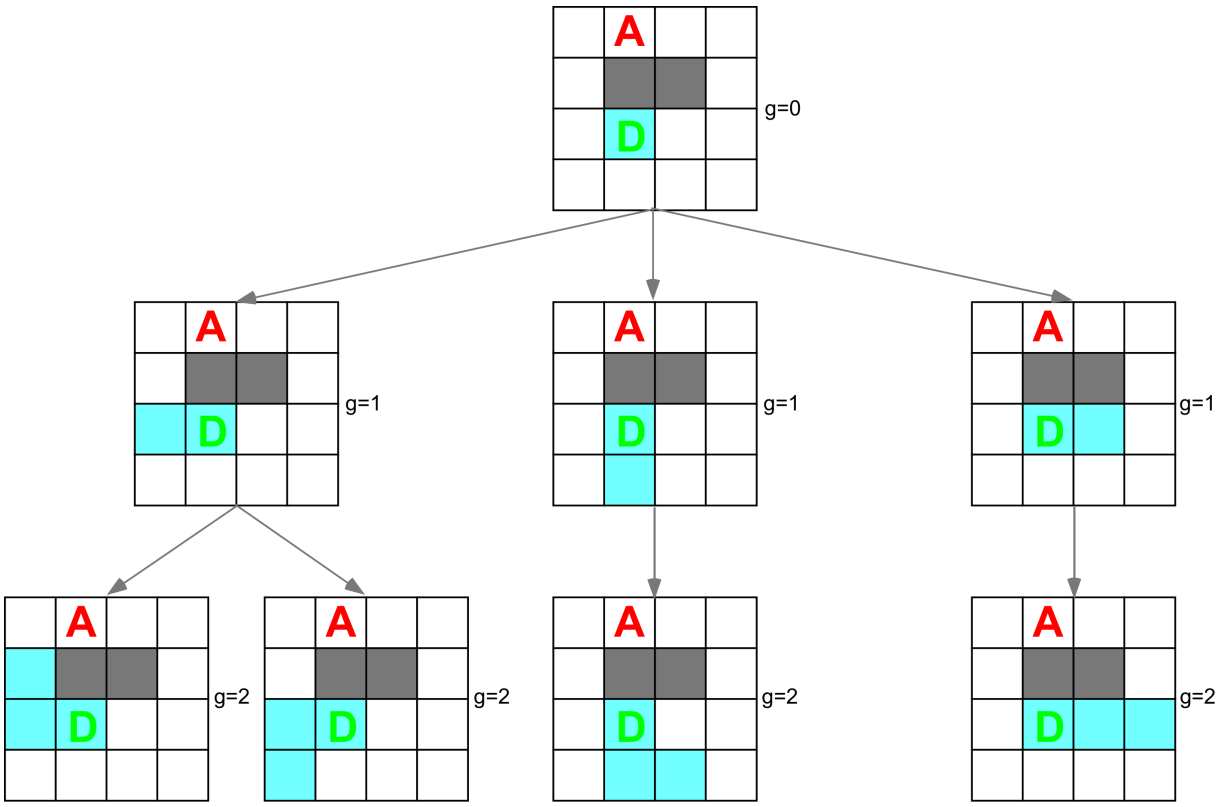

Figure 2.9: Vue de l'arbre après le 4ieme développement.

2.4.3 Approche “recherche informée heuristique”

L'algorithme A* est une réponse possible à cette troisième approche pour calculer le chemin de longueur minimale. En plus de mémoriser pour chaque case le coût du chemin jusqu'à cette case (la variable g), il va utiliser une *heuristique* (critère ou méthode fondée sur le bon sens permettant de déterminer parmi plusieurs attitudes celle qui promet d'être la plus efficace pour atteindre un but) pour évaluer de manière optimiste la longueur du chemin qui reste à parcourir. On disposera ainsi une évaluation de la longueur totale du chemin.

Dès lors, plutôt que développer la feuille qui a le plus petit g comme dans l'algorithme de Dijstra, c'est la feuille qui a le plus petit chemin *estimé* qui sera choisi pour être développé. Pour s'assurer que A* trouve le plus court chemin, il est nécessaire que l'heuristique soit admissible, c'est-à-dire que l'évaluation heuristique soit optimiste et qu'elle donne dans tous les cas une longueur de chemin inférieure ou égale à la longueur du chemin restant.

Une heuristique admissible est l'heuristique de Manhattan. Elle consiste à considérer le chemin vers le but sans obstacle. Facile à calculer, elle donne rapidement un minorant du chemin restant. Notons h l'heuristique admissible. La longueur estimée du chemin passant par une position est notée

$$f = g + h$$

et additionne le chemin déjà parcouru avec le chemin minimum qui reste à parcourir.

Le déroulement de l'algorithme A* est similaire à Dijstra sauf que l'on développe la feuille présentant le f le plus petit, comme l'illustre la figure 2.10.

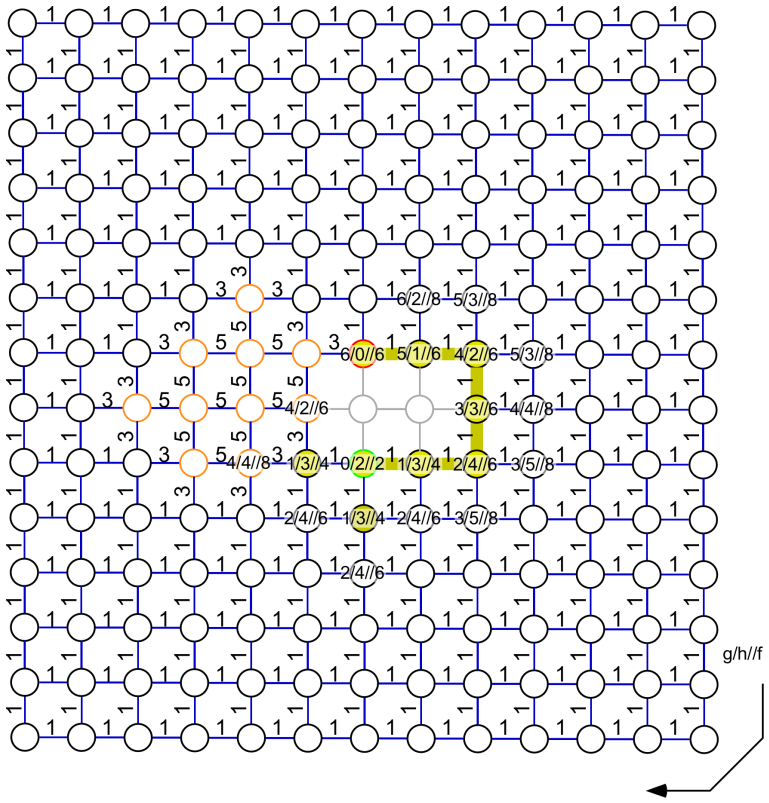

Figure 2.10: Résultat de l'exécution de l'algorithme A* sur l'exemple représenté par la figure 2.5 gauche.

L'exécution de l'algorithme A* peut également être représenté sous la forme d'un arbre. En reprenant la carte très simple représentée en figure 2.8, l'exécution de l'algorithme conduit à l'arbre de la figure 2.11.

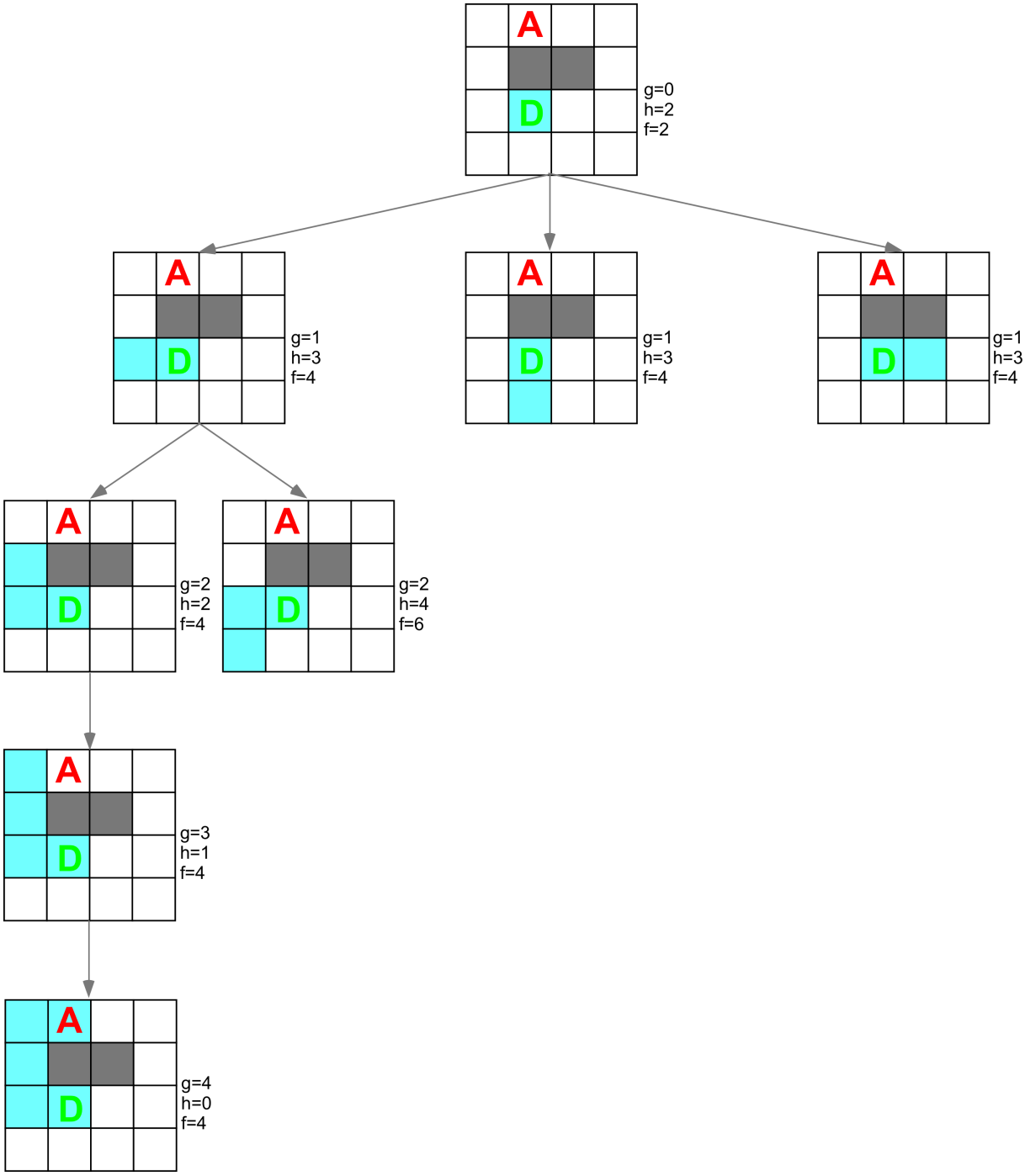

Figure 2.11: Arbre final obtenu sur l'exemple représenté en figure 2.8.

L'algorithme A* est prouvé pour être *monotone* (si f ne décroit jamais), *complet* (trouve un chemin quand il en existe un), *optimal* (trouve le meilleurs chemin quand il en existe un), et *efficace* (étend le nombre minimum nécessaire de noeuds). Il connait différentes extensions et variantes comme (1) A* with expansion distance, (2) bidirectional A* with expansion distance, (3) EBS-A* algorithm, etc.

2.4.4 Instances numériques

La collection *2D Pathfinding Benchmarks* [6] est un ensemble d'instances numériques de référence mise à disposition pour évaluer ce type d'algorithme. Elle est disponible à l'adresse `https://movingai.com/benchmarks/grids.html`.

Elle propose des données et résultats pour des instances issues de jeux vidéos (Dragon Age 2, Warcraft III, Starcraft, etc.), de villes réelles (Paris, NewYork, Milan, etc., ou encore des situations crées artificiellement (labyrinthe, terrain, etc.).

Par exemple, la carte suivante représente un quartier de Paris (`https://movingai.com/benchmarks/street/index.html`) :

Figure 2.12: Illustration des données correspondant à la carte `Paris_2_256.map`. Le fichier `Paris_2_256.scen` rapporte la longueur optimale entre un point de départ et un point d'arrivée. Pour cette instance, la plus grande longueur vaut `422.8843`.

Le format utilisé pour ces deux fichiers `map` et `scen` est décrit à `https://movingai.com/benchmarks/formats.html`.

2.4.5 Productions

1. Faire une lecture complète et approfondie de la section 2.3 du document qui vous a été remis, prendre possession des ressources qui vous ont été fournies.
2. Revenir sur chacun des trois algorithmes discutés en vous documentant à l'aide d'ouvrages de référence (dont vous aurez à donner la référence) de manière à vous donner une compréhension approfondie de ceux-ci.
3. Se donner une carte et une scène (point de départ, point d'arrivée) raisonnables et pour cette carte dérouler les trois algorithmes à la main en effectuant l'ensemble des opérations jusqu'à la condition d'arrêt de l'algorithme.
4. Préparer le codage des algorithmes de Dijkstra et A* (calculant le plus court chemin et retournant le chemin obtenu et sa valeur) en veillant à bien choisir les structures de données.
5. Implémenter les deux algorithmes en Julia en prenant soin à la qualité de l'implémentation.
6. Choisir au moins 1 instance numérique dans la collection qui a été indiquée et valider numériquement les trois implémentations à l'aide de l'instance choisie.

   Attention : vos implémentations et résultats pourront être améliorés au delà de cette étape. Il est donc attendu une version fonctionnelle mais pas nécessairement finale.

Consignes :

- vous êtes libre de vous inspirer des nombreux packages relatifs aux graphes (et dans ce cas vous aurez à en faire référence), mais vous ne pouvez pas les utiliser pour mettre en place vos algorithmes (autrement dit, les algorithmes sont à implémenter par vous même).
- vous êtes libre d'utiliser n'importe quel autre package disponible dans la librairie standard Julia, mais sans en abuser. Eviter donc d'invoquer un package pour une fonctionnalité mineure et éviter les packages/fonctionnalités exotiques.
- soyez force de proposition sur au moins un aspect de votre choix qui vient en complément des objectifs et consignes déjà exposés (exemple, une visualisation graphique, une tentative de paralléliser des traitements, etc.).
- Vos codes doivent être prêt pour être testés. Un utilisateur
  1. prendra connaissance de vos indications sur le readme du dépôt github;
  2. téléchargera en local votre dépôt github;
  3. suivra les indications du readme afin d'exécuter vos algorithmes sur un ordinateur sous linux/macOS depuis le REPL de Julia 1.9
- les précisions concernant la seconde partie vous seront données à la séance de mi-parcours.

TEMPS ESTIMÉ : 20h

PRINCIPALES RÉFÉRENCES

[1] Tristan Cazenave. *Intelligence artificielle-une approche ludique*. Ellypses, 2011.

[2] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction à l'algorithmique : cours et exercices*. Dunod (2eme édition), 2002.

[3] Michel Gondran and Michel Minoux. *Graphes et Algorithmes*. Lavoisier (4ème édition), 2009.

[4] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[5] Pierre Marquis, Odile Papini, and Henri Prade. *Algorithmes pour l'intelligence artificielle : Panorama de l'intelligence artificielle - Ses bases méthodologiques, ses développements*, volume 2, chapter "1. Recherche heuristiquement ordonnée dans les graphes d'états (Henri Farreny)". 2014.

[6] N. Sturtevant. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144 – 148, 2012.