Sujet Projet Informatique:
Résolution de labyrinthes par chemin géodésique

2011

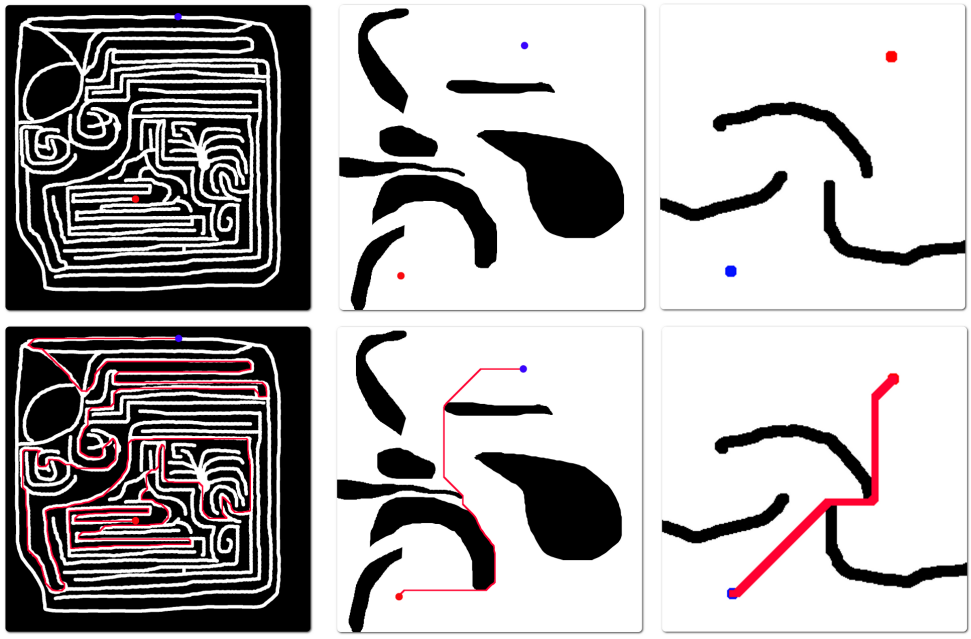

FIGURE 1 – Différents chemins minimaux trouvés automatiquement sur des images générées manuellement. Haut : image d'entrée (rouge=point de départ, bleu=point d'arrivé). Bas : Affichage du plus court chemin reliant les deux points.

1 But

L'objectif de ce sujet est de trouver et tracer automatiquement le chemin le plus court reliant un point A à un point B sur une carte/image bitmap dont les obstacles sont potentiellement définis manuellement tel que présenté en fig. 1.

Le sujet se décompose en deux parties :

1. Calcul d'une propagation de distance géodésique
2. Extraction du plus court chemin à partir de la carte de distances

On peut aisément appliquer ce type d'approche pour la résolution automatique de labyrinthe. On trouvera cependant que cet algorithme peut servir de bases dans de nombreux domaines :

- Intelligence artificielle : déplacement de robots, jeux vidéos, ...
- Imagerie médicale : parcours et caractérisation d'artères, bronches, ...
- Parcours de cartes : trajets de voitures, réseaux, ...

2 Définition discrète des structures

2.1 Entrée

On considère en entrée une image bitmap (ensemble de pixels colorés disposés sur une grille) possédant :

1. Des zones blanches correspondant à des régions vides.
2. Des zones noires correspondant à des obstacles infranchissables.
3. Un ensemble connexe et convexe
   - de pixels rouges correspondant au lieu de départ A
   - de pixels bleus correspondant au lieu d'arrivé B

Un exemple est présenté en fig. 2-gauche.

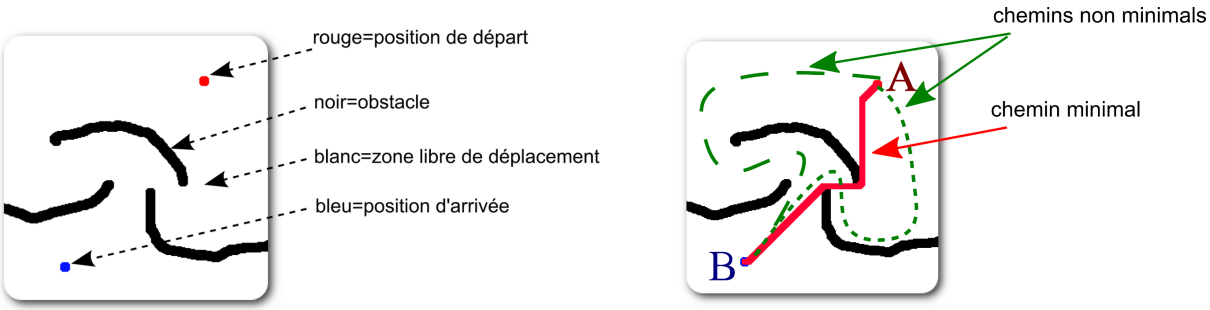

FIGURE 2 – Gauche : Caractéristiques de l'image d'entrée. Droite : Tracés de chemins reliant deux positions.

On considèrera qu'il existe au moins un *chemin* reliant A à B. Un *chemin* étant ici défini par un ensemble connexe de pixels.

2.2 Sortie

La sortie demandée consiste à définir le *chemin minimal* reliant A à B comme présenté en fig. 2-droite.

2.3 Formalisation discrète

Soit $p = (x, y)$ la position d'un pixel. Un chemin c reliant les positions p_A à p_B est donné par la suite de positions

$$(p_i)_{i\in[\![0,N-1]\!]} \quad \text{avec} \quad \left\{ \begin{array}{l} p_0 = p_A \text{ et } p_{N-1} = p_B \\ \forall i \in [\![1, N-1]\!] \, , \quad \|p_i - p_{i-1}\| \leq r \, , \end{array} \right. \tag{1}$$

avec $r = \sqrt{2}$ dans le cas d'une connexité de type $v8$.

On note E^{AB}_{chemin}, l'ensemble des chemins possibles sur une image reliant le point A au point B. La longueur d'un chemin $c \in E^{AB}_{\text{chemin}}$ est donné par $L(c)$, avec

$$L : \left\{ \begin{array}{ccc} E^{AB}_{\text{chemin}} & \to & \mathbb{R} \\ c = (p_i)_{i \in [\![0, N-1]\!]} & \mapsto & \sum_{i=1}^{N-1} \|p_i - p_{i-1}\| \end{array} \right. , \tag{2}$$

Un chemin $c^{p_A p_B}_{\text{min}} \in E^{AB}_{\text{chemin}}$ est dit minimal si il n'existe aucun autre chemin $c^{p_A p_B} \neq c^{p_A p_B}_{\text{min}}$ tel que $L(c^{p_A p_B}) < L(c^{p_A p_B}_{\text{min}})$. En d'autres termes, il s'agit de trouver le chemin le plus court reliant A à B.

On notera que les images avec au moins un obstacle associé à cette norme ne définissent pas un espace Euclidien. Il peut ainsi exister plusieurs chemins minimaux entre deux positions.

2.4 Carte de distance géodésique

Une carte de distances géodésique $\mathcal{D}_A$ par rapport à un (ou plusieurs) pixel(s) A peut se concevoir comme une image à niveau de gris dont la valeur de chaque pixel correspond à la distance minimale entre le pixel courant et A. Une carte de distances est illustrée en fig. 3.

Supposons p_A la position du pixel origine - c'est à dire de distance 0-, on alors peut définir

$$\mathcal{D} : \left\{ \begin{array}{l} \mathbb{R}^2 \to \mathbb{R} \\ p_k \mapsto L(c^{p_A p_k}_{\text{min}}) \end{array} \right. \tag{3}$$

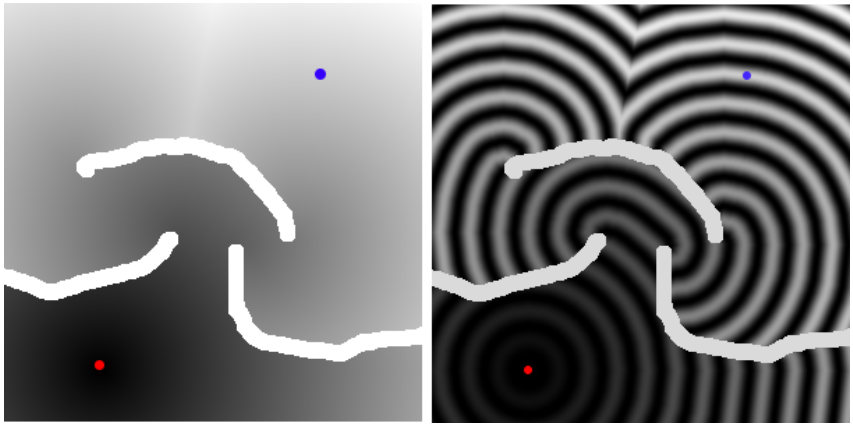

FIGURE 3 – Gauche : Carte de distance dont l'origine est indiquée par le point rouge. Une coloration noire indique une distance proche de 0, alors que la couleur blanche indique une distance maximale. Droite : Carte de distance dont les isocontours sont rendus visibles. Les points situés sur les cercles de même rayons sont à une distance identique du centre.

3 Étapes du projet et algorithmique

3.1 Analyse de l'entrée

La première étape consiste à charger et analyser une image couleur pour en extraire la position de départ et d'arrivée.

3.1.1 Chargement et manipulation d'une image

L'image de base pourra être générée à la main à l'aide d'un outil externe tel que *gimp*. Elle sera sauvegardée dans un format standard. On pourra dans un premier temps privilégier le

format d'écriture *ppm* ascii qui peut être lue et écrit trivialement sans faire appels à des librairies externe.

Note : Le chargement d'images *png* ou *jpg* est également possible en utilisant les librairies *libpng* et *libjpg*.

La structure de donnée de manipulation d'une image est la suivante :

Chaque pixel contient une couleur (R, G, B) où chaque entrée est un entier compris entre 0 et 255. On définit ainsi une classe

```
class color
{
 unsigned char r,g,b;
};
```

Un image de taille $N_x \times N_y$ est ainsi un vecteur de *color*

```
class image
{
 std::vector<color> data;
};
```

Le pixel de position $p = (p_x, p_y)$ contient la couleur définie dans

```
data[px + Nx*py];
```

Une classe de stockage, chargement et écriture d'image vous est fournie pour ce projet.

3.1.2 Extraction du point d'arrivée et de départ

On cherche à définir une unique position d'arrivée et de départ à partir d'un ensemble potentiel de pixels rouges et bleus. Une solution possible est de considérer le point de départ (/d'arrivé) comme le barycentre de l'ensemble des pixels rouges (/bleus).

L'algorithme est donc le suivant :

```
vector position_rouge;
vector position_bleus;
Pour k variant sur tout les pixels
  pk <- position du pixel courant
  c <- couleur de l'image data[pk]

  Si c==rouge
    position_rouge=[position_rouge,pk]
  Si c==bleu
    position_bleu=[position_bleu,pk]

Fin pour

depart=Barycentre(position_rouge);
arrive=Barycentre(position_bleu);
```

Pour manipuler des positions à deux dimensions spatiales entières, on pourra utiliser une classe

```
class p2d
{
 int kx;
 int ky;
};
```

dont les opérateurs de sommation et de division seront redéfinis.

3.2 Calcul de la carte de distance

Le calcul d'une carte de distance géodésique se réalise par propagation de proche en proche de la distance minimale à partir de la position de départ.

Soit $\mathcal{D}$ la carte de distance dont les valeurs sont initialisées à l'infini (par exemple $N_x \times N_y$). On initialise également le (les) point(s) de départ à la distance 0.

- On part de la position p du pixel de départ.
- On considère les voisins p_v de p (au sens d'un disque de rayon R).
- On met à jour la distance des voisins. La nouvelle distance étant donnée par

$$\min\left(\mathcal{D}(p_v), \mathcal{D}(p) + \|p_v - p\|\right) \ .$$

  C'est à dire, le minimum entre la valeur actuelle et celle obtenue en propagant la distance à partir de p.

Ensuite, on recommence itérativement l'opération de mise à jour des voisins pour tout les pixels voisins modifiés.

Pour éviter de parcourir chaque pixel à de multiples reprises, on ordonnera la liste de pixels à traiter suivant leur distance respective croissante. Ainsi la distance minimal se propagera sous forme de front. On appel cet algorithme *propagation de distance* pour une image. Sa généralisation est triviale dans le cas d'un graphe, et est connu sous la dénomination d'**algorithme de Dijkstra**.

Algorithme associé :

Soit Q une file de priorité contenant la paire d'information (distance,position) ordonnée suivant l'information de distance.

```
pA <- position de départ
Q <- (0,A)

Carte de distance D <- infini;
D[A]=0;

Tant que Q non vide
  (d,p)=PremierElement(Q);
  Q.pop();
  D[p]=d;

  Pour tous les voisins de p
    p2 <- position du voisin
    Si p2!=obstacle && D[p2]>d+||p2-p||
       Q.inserer(d+||p2-p||,p2);
    Fin si
  Fin pour
Fin tant que
```

On rappel que la STL fournie des structures de données efficace et générique. La file de priorité pourra etre implémentée sous la forme d'une classe particulière possédant la signature suivante :

```
//stocke une file de prioritee contenant des paires (distance,position)
// la file est triee suivant la distance
// il peut y avoir plusieurs elements pour une meme distance
// la complexite en ajout et en suppression doit etre au pire logarithmique
//   en fonction du nombre de distances sotckees
class PixelQueue
{
 public:

  //returne data.empty();
  bool empty() const;

  //retourne un element de distance minimale et le supprime des données
  // complexite doit etre en O(log(data.size()))
  std::pair<double,p2d> pop();

  //insere un nouvel element (distance,position) dans la structure
  // complexite doit etre en O(log(data.size()))
  void insert(double distance,const p2d& position);


 private:

  //structure de donnee de stockage
  std::map<double,std::list<p2d>,LessDouble > data;
};
```

On préfèrera redéfinir le foncteur LessDouble plutôt que d'utiliser directement *std : :less* de manière à ne pas différencier d et $d + \epsilon$.

```
class LessDouble
{
 public:
  LessDouble():epsilon(1e-5){}
  bool operator()(double a,double b) const {return (a+epsilon<b);}
 private:
  const double epsilon;
};
```

Un cas illustré de propagation de distances sur une image de taille réduite est proposé en fig. 4, et le cas d'exemple de la structure de données est illustré en fig. 5.

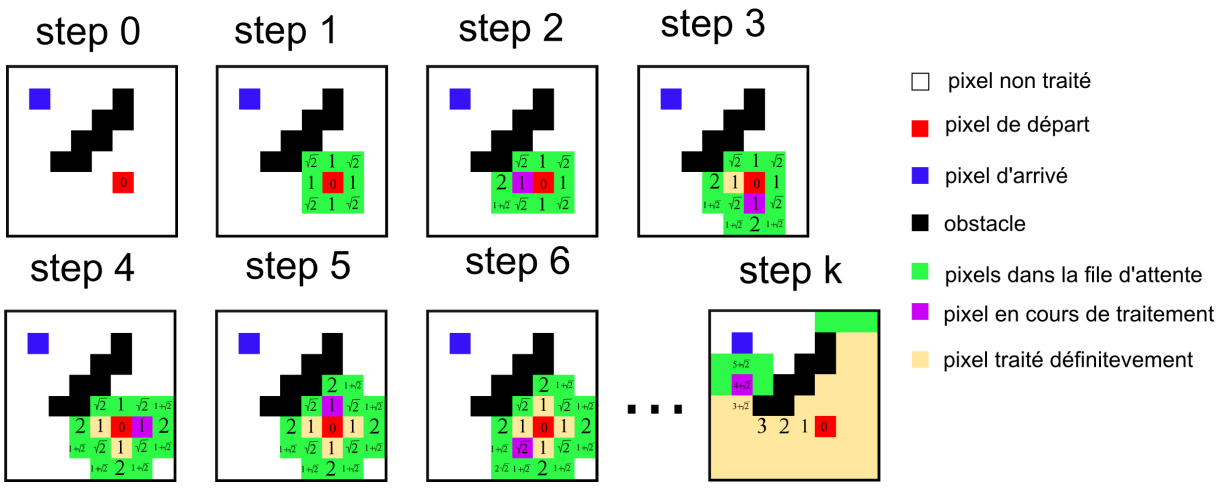

FIGURE 4 – Premières étapes de propagation de distance sur une image de taille 8×8 et pour un voisinage de type $v8$. La distance minimale entre le point bleu et le point rouge est donc de $6 + \sqrt{2}$.

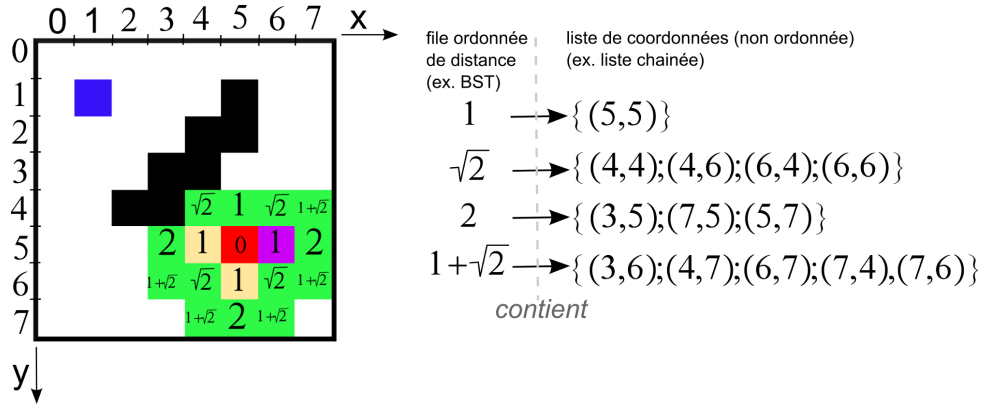

FIGURE 5 – Structure de données détaillée pour l'étape 4 de la figure. 4.

3.3 Extraction du chemin minimal

Une fois la carte de distance définie sur l'ensemble de l'image, il est possible d'extraire le chemin minimal reliant le point d'origine à celui d'arrivée.

Le principe est le suivant :
- On se place sur le pixel d'arrivée.
- On cherche dans son voisinage de type $v8$ le pixel de distance géodésique la plus faible.
- On itère sur le nouveau pixel jusqu'à tomber sur la position initiale de distance 0.

En marquant le passage sur chaqun des pixels, on obtient ainsi une courbe discrète orientée définissant le chemin le plus court à suivre pour se rendre du point A au point B.

Note : Une fois le pré-calcul global de la carte de distance réalisé, l'extraction du chemin minimal reliant le point d'origine à tout autre point de l'image est très efficace et rapide.

3.4 Travail demandé

Implémentez les algorithmes décrits et mettez les en oeuvres pour réaliser le tracé de chemins minimaux sur des images diverses.

Vous décrirez avec précision votre algorithme de propagation de distance.
Pour l'implémenter, vous procéderez par étapes dans des cas simples. Vous observerez et commenterez les caractéristiques de cette image (ex. Quelle relation existe-il entre les chemins minimaux et les isolignes de la fonction de distance, ...).
Vous illustrerez votre projet à l'aide de sorties visuelles pertinentes.
Enfin, vous justifierez de la complexité algorithmique de votre programme.

Important : Vous prendrez un soin tout particulier
- À réaliser et commenter des jeux de tests permettant de valider l'avancement de votre projet.
- À vous assurer que votre programme est robuste [1]
  - Chemin inexistant entre A et B
  - Positions de départ et d'arrivée inexistantes ou non connexes
  - Obstacles ou chemins possibles de faibles largeurs.
  - ...

Il vous appartient d'interpréter et de trouver des réponses pertinentes aux différents cas de figures que l'on pourra rencontrer.

Il n'est pas obligatoire de vous coller aux structures de classes proposées. Il est par contre nécessaire de justifier, commenter, et tester de manière pertinente l'ensemble de vos choix.

4 Généralisation possible

Une fois l'algorithme mis en place dans ce cas d'étude, il est possible d'implémenter différentes améliorations.

Note : Il est inutile de tenter des améliorations tant que l'algorithme de base n'est pas complet et testé convenablement.

1. On rappel que l'image de départ est potentiellement fournie par un utilisateur. Le bon déroulement de l'algorithme dans un cas simple n'est aucunement une justification d'un comportement correct sur l'ensemble des entrée possibles.

4.1 Voisinage adaptatif

On note que
- Un petit voisinage (type v4, v8) permet de tenir compte des obstacles de manière précise. La distance associée est par contre fortement éloignée de la distance de type $\|.\|_2$ continue.
- Un masque de voisinage plus large se rapproche d'une norme continue, mais risque de *passer au dessus* des obstacles rendant l'algorithme moins robuste à tout type d'images.

L'idée est de définir un masque de voisinage de taille adaptative permettant de
- Considérer un voisinage important (boule unitée de rayon 5 ou 6 pixels) lorsqu'aucun obstacle n'est proche.
- Considérer un voisinage plus petit lorsqu'un obstacle est détecté localement.

4.2 Distance pondérée

Dans des applications réelles (reseaux, terrains, ...) les chemins élémentaires entre deux pixels voisins peuvent être plus ou moins couteux. On peut alors pondérer cette distance unitaire de déplacement en fonction
- De la position (localitée),
- De l'orientation (anisotropie).

On pourra alors modifier l'algorithme de mise à jour des distances lors de la génération de la carte de distance pour prendre en compte ce cout variable.

En particulier, la valeur du niveau de gris de l'image d'origine pourra par exemple indiquer la difficultée de traverser telle ou telle zone.

4.3 Cas de déplacement de solides

Dans le cas d'un parcours d'un robot au milieu d'obstacle, la taille de celui-ci est également à prendre en compte.

On pourra implémenter un algorithme prenant en compte ce critère de taille minimale d'objet dans le cas simple d'un disque englobant isotrope invalidant certains passages trop étroits.

4.4 Objets 3D

Des données issues de scanners médicaux peuvent fournir des structures volumiques de type tubulaires (ex. artères). Les données de départ sont alors données par une image volumique (l'élément de base étant alors désigné par *voxel* à la place de *pixel*).

Le calcul de carte de distance et de chemin minimal peut se réaliser de manière identique en dimension quelconque trivialement. Seule la boule du voisinage considéré est fonction de la dimension.

On pourra implémenter le tracé d'un chemin minimal dans le cas d'une structure volumique et utiliser Matlab pour la visualisation de telles données.