RAIRO. RECHERCHE OPÉRATIONNELLE

Y. KERGALL

Algorithmes polynomiaux pour la recherche d'un chemin optimal dans une scène planifiée

RAIRO. Recherche opérationnelle, tome 16, n° 2 (1982), p. 131-154

<http://www.numdam.org/item?id=RO_1982__16_2_131_0>

R.A.I.R.O. Recherche opérationnelle Operations Research
(vol. 16, n° 2, mai 1982, p. 131 à 154).

ALGORITHMES POLYNOMIAUX POUR LA RECHERCHE D'UN CHEMIN OPTIMAL DANS UNE SCÈNE PLANIFÉE (*)

par Y. KERGALL (1)

Résumé. — *Nous proposons des algorithmes polynomiaux pour résoudre le problème de la recherche d'un chemin de coût minimal dans une scène planifiée lorsqu'on ne considère qu'un seul critère, lorsqu'on considère une hiérarchie de critères ou encore le problème muticritère.*

Mots-clés : algorithme, chemin optimal, graphe, robotique.

Abstract. — *We propose polynomial algorithms for solving the problem of the search of a minimum cost path in a planified area. We may consider either only one criterion or a hierarchy of criteria, or the problem with several criteria.*

Keywords: Algorithms, optimal path, graph, robotics.

INTRODUCTION

Un article récent [1] présenté au 2nd Congrès de Reconnaissance des formes et Intelligence artificielle de Toulouse (septembre 1979), propose un algorithme de type « énumératif » pour résoudre le problème de la recherche d'un chemin optimal par le biais d'une hiérarchie de critères, dans une scène planifiée.

Il existe de nombreux algorithmes polynomiaux classiques (2) en théorie des graphes et recherche opérationnelle pour résoudre le problème du plus court chemin : Ford, Ford amélioré, Bellman, Moore-Dijkstra ou algorithmes matriciels. Tous ces algorithmes opèrent sur des graphes valués. Or, dès que nous travaillons avec le critère de changement de direction, il n'est plus possible de valuer le graphe représentatif de la relation de voisinage entre les cases. Nous verrons de plus sur un exemple au paragraphe 6 que les chemins construits ici ne vérifient plus le théorème d'optimalité de Bellman, très général en recherche opérationnelle, puisque le coût d'un chemin ne dépend pas seulement des arêtes parcourues ou des sommets traversés mais de la façon dont se suivent les arêtes sur le chemin.

(*) Reçu juin 1981.
(1) Université de Tours, Laboratoire d'Informatique appliquée, parc Grandmont, 37200 Tours et depuis le 1/10/81 : Centre universitaire, informatique, 33, rue Louis-Pasteur, 84000 Avignon.
(2) Pour toutes les notions classiques sur graphes et algorithmes (*voir* Gondran et Minoux [2]).

Néanmoins, si ces algorithmes classiques ne peuvent être directement utilisés, nous proposons de les adapter à notre problème en leur conservant leur caractère polynomial.

En effet tous ces algorithmes reviennent à décomposer un graphe en niveaux et nous proposons ici de décomposer la scène en niveaux à partir de la case départ, sans construire de graphe représentatif de la scène qui est une perte de temps et de place mémoire.

L'utilisation de procédures énumératives [2], [6], ne nous semble pas s'imposer lorsque la scène est entièrement connue ou qu'elle est de taille modeste (jusqu'à 100×100) ce qui est bien souvent suffisant dans les applications.

Dans d'autres cas relevant de l'intelligence artificielle (univers partiellement inconnu au départ et appréhendé à l'aide de capteurs de différents types, univers de très grande dimension), on aura recours à d'autres algorithmes [3, 4, 5, 8], basés sur l'utilisation d'heuristiques.

Lorsque dans un problème interviennent un grand nombre de variables, trois types de méthodes permettent de réduire ce nombre de variables :

(1) réduire la taille de la structure de données (généralement une arborescence) qui est la représentation interne du problème : méthode $\alpha - \beta$, Graph Traverser de Doran et Michie [9], Flow-down method [10], etc.;

(2) partitionner le problème en k sous-problèmes [6];

(3) regrouper les variables en méta-variables ce que nous faisons ici en définissant et construisant des « blocs » horizontaux ou verticaux, notion qui nous semble assez bien représenter le travail effectué par un opérateur humain confronté à un labyrinthe lorsqu'apparait le critère de changement de direction.

Cette notion permet bien entendu de réduire le nombre de variables de départ (les cases) mais sans trop compliquer les relations sur les « méta-variables » obtenues. En particulier la notion de pavé maximal (au sens de l'inclusion) comme rectangle formé de cases autorisés, notion analogue à celle de cellule dans [4] semble mal adaptée ici vu la difficulté à retrouver la suite des cases à emprunter une fois la suite des pavés trouvée.

Remarquons enfin que si dans [1] l'exploration de la scène en profondeur d'abord conduit à essayer d'améliorer pas à pas un chemin initial non optimal mais néanmoins le meilleur possible (à l'aide d'une heuristique à rapprocher de celle utilisée par Hart [3] dans l'algorithme classique A^*), les parcours en largeur d'abord utilisés ici permettent de compléter un début de chemin optimal.

Après avoir défini le problème, présenté l'article [1] et proposé des simplifications préliminaires sur la scène, nous proposons deux algorithmes pour le problème de hiérarchie de critères et le problème multicritère. Leur complexité

est ensuite étudiée et nous donnons quelques temps d'exécution en fonction de divers paramètres pour des exemples tirés au hasard. Les programmes sont en annexe.

1. POSITION DU PROBLÈME

Définition d'une scène planifiée

Nous appelons scène planifiée, une aire de déplacement rectangulaire de M cases sur N cases, que l'on pourrait représenter par une matrice de type $M \times N$. Nous avons préféré, pour des raisons d'efficacité de programmation, associer à chaque case (i, j) de la scène, avec $1 \leqq i \leqq M$ et $1 \leqq j \leqq N$, une valeur $x=(i-1)N+j$. Ainsi on définit la scène à l'aide d'un *vecteur*, noté LAB, de la façon suivante : pour $1 \leqq x \leqq MN$: $\mathrm{LAB}(x)=0$ si la case x est interdite (obstacle); $\mathrm{LAB}(x)=1$ si la case x est autorisée.

Chaque case, non située sur un bord de la scène, possède 4 cases voisines. Un chemin est un ensemble de cases autorisées voisines. Le problème est la recherche d'un chemin « optimal » entre une case de départ notée D et une case d'arrivée notée A.

Critères d'optimalité

Nous nous sommes intéressés aux deux critères suivants : C1, nombre de cases traversées; C2, nombre de changements de direction.

Problème 1 : On considère les deux critères séparément et on recherche un chemin optimal pour C1 ou C2 c'est-à-dire comprenant un nombre minimal de cases traversées ou un nombre minimal de changements de direction.

Problème 2 : Hiérarchie de critères : un chemin optimal sera parmi les chemins optimaux pour C2, un chemin optimal pour C1 [problème considéré dans (1)].

Problème 3 : Plus généralement soit b le coût représenté par le fait de traverser une case, a le coût d'un changement de direction. Soit n_1 le nombre de changements de direction et n_2 le nombre de cases traversées pour un chemin allant de D à une case x. La valuation de cette case x s'écrit $v(x)=n_1 a+n_2 b$.

On se propose de chercher $v^*(x)=\min v(x)$ et de déterminer un chemin correspondant à cet optimum.

2. MÉTHODE DELANNOY-LEROI-BOURTON [1]

Cette méthode comprend 3 parties :

(1) réduction du nombre de cases autorisées;

(2) recherche d'un premier chemin entre D et A;

(3) amélioration de ce chemin.

2.1. Réduction du nombre de cases autorisées

En notant d^+ le nombre de cases voisines d'une case donnée, on élimine au départ toutes les cases – sauf A et D – telles que $d^+ = 1$, et que l'on appelle « cul-de-sac » dans la suite. On améliorera cette réduction au paragraphe 3.

2.2. Recherche d'un premier chemin entre D et A

Ce chemin va servir d'initialisation à cette méthode qui est une méthode de type « énumératif ». Ce premier chemin est construit avec l'heuristique suivante : en cours de construction du chemin, la suivante d'une case est sa voisine qui minimise la distance euclidienne à A.

2.3. Amélioration de ce chemin

Pour cela on essaie d'améliorer tout sous -chemin compris entre 2 sommets vérifiant $d^+ > 2$ (points de branchement). On ne sera assuré de l'optimalité que si l'on énumère tous les sous-chemins possibles. On connait les limites de telles procédures, même lorsqu'on leur applique certaines règles réduisant le nombre d'opérations envisagées (6). La règle de préclusion (sur les poids des chemins) est par exemple appliquée ici, qui abandonne tout début de chemin dont le poids atteint celui du meilleur chemin trouvé jusqu'alors.

3. SIMPLIFICATIONS PRÉLIMINAIRES DE LA SCÈNE

Les algorithmes présentés ici seront d'autant plus performants que la scène possède moins de cases autorisées, aussi va-t'on essayer de réduire *a priori* le nombre de cases permises.

Exemple (*fig.* 1) :

Figure 1

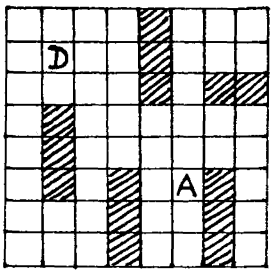
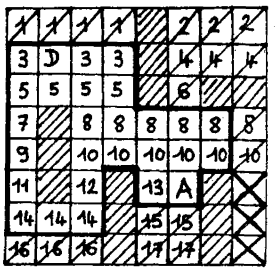


Figure 2



On élimine d'abord les portions en culs-de-sac (marquées X sur la figure 2); pour cela on peut, dans un premier passage, éliminer tous les culs-de-sac de la scène. Puis on rebalaye la scène en éliminant les nouveaux culs-de-sac, etc., jusqu'à ce que l'on ne puisse plus rien éliminer. Si λ est la longueur maximale d'une portion en cul-de-sac, la complexité est en $O[(\lambda+1)MN]$.

Nous avons préféré programmer la méthode suivante, présentée ici sous une forme recursive :

```
pour i=1 à MN
| Examiner la i-ième case :
| | si LAB (i) n'a qu'un seul voisin, LAB (j), alors :
| | éliminer LAB (i);
| | examiner LAB (j);
| fin
fin
```

On génère un « backtrack » (retour en arrière) de profondeur maximale λ. On examine ici au maximum $MN+\Sigma\lambda_k$ cases, où $\Sigma\lambda_k$ représente le nombre total des cases appartenant aux portions en culs-de-sac, d'où une complexité en $O(MN)$.

On définit ensuite des blocs horizontaux comme des ensembles maximaux (au sens de l'inclusion) de cases autorisées voisines situées sur une même ligne. Sur la figure 2 on a affecté du même numéro les cases appartenant au même bloc horizontal.

Définissons le graphe G_H de voisinage des blocs horizontaux par : soit 2 blocs horizontaux i et j alors $(i,j) \in G_H \Leftrightarrow$ il existe au moins une case du bloc i et une case du bloc j qui sont voisines. On supprimera de la scène toutes les cases appartenant à un bloc horizontal qui est une feuille de G_H et qui ne contient pas D ou A (une feuille supprimée est notée ●—//—● sur la figure 3). Ce graphe G_H pourrait être réutilisé pour trouver un chemin présentant un nombre minimal de changements de direction (§ 4.2). En définissant les blocs verticaux et G_V graphe de voisinage des blocs verticaux, on supprime aussi les feuilles de G_V.

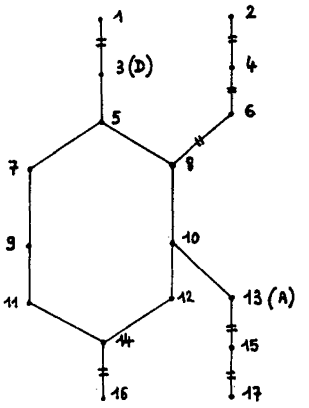


Figure 3

Pour l'exemple ci-dessus, les cases marquées ☑ sur la figure 2 sont éliminées. Cette simplification préliminaire permet de supprimer toutes les cases situées, récursivement, sur les « bords » de la scène, en particulier des cases ayant au départ un nombre quelconque de voisins.

4. PROBLÈME 1

4.1. Nombre minimal de cases traversées

Nous notons ici $l_1(x)$, la longueur c'est-à-dire le nombre de cases traversées par un chemin allant de D à la case x. La détermination de $l_1^*(A)$ qui est la longueur minimale d'un chemin de D à A ainsi que le chemin correspondant est très simple puisqu'il suffit de décomposer le scène en niveaux N_i à partir de D en posant :

$$N_0 = \{D\},$$

$N_i = \{$ ensemble des cases x voisines d'une case de N_{i-1} et non encore atteintes $\}$.

Il est évident que l'on peut écrire :

$$N_i = \{\text{cases } x / l_1^*(x) = i\}.$$

On arête la décomposition en niveaux dès qu'il existe j tel que $N_j = \emptyset$ (dans ce cas le problème n'a pas de solution) ou dès que A est atteinte; dans ce cas en notant $\lambda = l_1^*(A)$, un chemin solution est une suite de cases voisines partant de A et de valuations λ, λ_{-1}, λ_{-2}, ..., 2, 1, 0.

Remarque : Il est clair que ce que nous faisons ici est la décomposition en niveaux du graphe G_R représentatif de la scène. Ce graphe pourrait être défini par :

$$G_R=(X, \Gamma).$$

X : ensemble des cases autorisées.

Γ : pour deux cases i et j, $(i,j)\in\Gamma \Leftrightarrow i$ et j sont voisines. Il est bien sûr inutile de construire explicitement G_R ni même de se le donner sous la forme d'une case et de la liste de ses voisines (1) qui occupe encore un nombre de mémoires égal à 4 fois le nombre de cases libres ce qui peut être considérable sur une scène 100×100. En effet nous regarderons *en cours d'algorithme* quelles sont les cases libres voisines d'une case rencontrée, puisque chaque case ne sera considérée qu'une seule fois, sauf au paragraphe 6.

Exemple : Sur la figure 4.

4.2. Nombre minimal de changements de direction

Nous notons $l_2(x)$ la longueur, c'est-à-dire le nombre de changements de direction pour un chemin allant de D à x. Nous cherchons $l_2^*(A)$ et le chemin correspondant. Nous proposons une nouvelle décomposition en niveaux en notant :

$N_0=D\cup\{$cases que l'on peut atteindre
sans changer de direction (directement) à partir de $D\}$.

$N_i=\{$cases non atteintes
et que l'on peut atteindre directement à partir de $N_{i-1}\}$.

On s'arrête dès qu'il existe j tel que $N_j=\emptyset$ ou dès que l'on atteint A. Si A appartient à N_k alors $l_2^*(A)=k$.

En effet si A appartient à N_k alors $l_2(A)=k$.

De plus $k=\min l_2(A)=l_2^*(A)$ puisqu'on s'arrête dès que l'on atteint A, c'est-à-dire qu'il n'existe pas de niveau N_l, $l<k$, et $A\in N_l$.

Pour retrouver un chemin minimal, il suffit au cours de l'algorithme de noter chaque fois que l'on atteint une case x de N_i, à partir de quelle case y de N_{i-1} on a pu atteindre x. On posera $\mathrm{PRED}(x)=y$ qui nous donnera ainsi la liste des prédecesseurs de chaque sommet. En d'autres termes PRED contient la liste des sommets où l'on a changé de direction entre D et A; entre 2 de ces sommets, le chemin en ligne droite est évident à reconstituer.

Exemple :

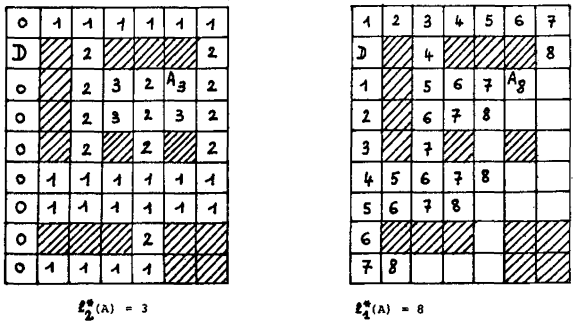
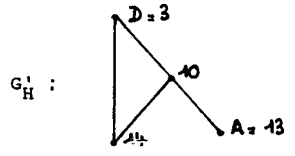


Figure 4

4.3. Utilisation de G_H (§ 3, *fig.* 3)

Dans le graphe G_H nous avons représenté les blocs horizontaux et les liaisons entre eux. Ces arêtes entre 2 blocs horizontaux représentent des liaisons *verticales* que l'on peut songer à prolonger de façon à relier tout bloc horizontal i à tout bloc horizontal j s'il existe un bloc vertical entre i et j (fermeture transitive de la relation de voisinage entre 2 blocs horizontaux)



On obtient un graphe G'_H : A tout chemin $x_0 x_1 \ldots x_n$ de G'_H ($D \in x_0$, $A \in x_n$) on fait correspondre simplement un chemin dans la scène puisque l'on se déplace horizontalement dans tout bloc x_i et verticalement entre tout couple (x_i, x_{i+1}), $0 \leqq i \leqq n-1$. Le nombre de changements de direction est :

$$l_2(A) = 2(n-1) + \varepsilon + \varepsilon',$$

avec $\varepsilon = 0$ si on peut atteindre directement, dans la scène, x_1 à partir de D, 1 sinon et $\varepsilon' = 0$ si on peut atteindre directement, dans la scène x_{n-1} à partir de A, 1 sinon.

Pour le chemin (3, 10, 13) de G'_H on trouve $l_2(A) = 2+1+0 = 3$ et pour (3, 14, 10, 13), $l_2(A) = 4+1+0 = 5$.

On se ramène donc, à $\varepsilon + \varepsilon'$ près, à la recherche d'un chemin comportant le moins d'arêtes dans un graphe très réduit.

5. HIÉRARCHIE DE CRITÈRES DE CHOIX

Disposant de 2 critères de choix, on commence par minimiser le critère « le plus coûteux ». Pour un robot, le changement de direction est l'un des critères les plus coûteux en temps donc en énergie. Nous résolvons alors le problème : parmi les chemins qui comportent un minimum de changements de direction, quels sont ceux qui traversent un nombre minimal de cases [problème étudié dans (1)].

Nous réappliquons l'algorithme vu en 4.2, mais pour chaque niveau N_i, on associe de plus à toute case x de N_i sa valuation $l_1(x)$ obtenue de la façon suivante [où $d(x, y)$ représente le nombre de cases traversées pour aller (en ligne droite) de x à y] :

5.1. Algorithme

On rappelle (*voir* 4.2) que si une case x est directement accessible à partir d'une case y, c'est que l'on peut aller de y à x sans changer de direction, c'est-à-dire en suivant une horizontale ou une verticale de la scène :

Début *Initialisation* : pour toute case autorisée de la scène :
$l_2(x) = -1$
$i = -1$; $N_{-1} = \{D\}$; $l_1(D) = 0$
Répéter :
Pour chaque case y dans N_i, et pour toute case x directement accessible à partir de y :
faire :
si $l_2(x) = -1$: mettre x dans N_{i+1}
faire $l_2(x) = i+1$
et $l_1(x) = l_1(y) + d(x, y)$
PRED $(x) = y$
si $l_2(x) = i+1$:
$l_1(x) = \min(l_1(x), l_1(y) + d(x, y))$
si $l_1(x)$ est modifié : PRED $(x) = y$
$i = i+1$
Tant que ($N_i \neq \emptyset$ et $l_2(A) = -1$).
Imprimer le chemin trouvé s'il existe.
Fin

Preuve : Cet algorithme s'arrête si $N_i = \emptyset$ (il n'y a pas de solution) ou si $l_2(A) \neq -1$, le chemin étant alors facile à reconstituer à l'aide du tableau d'indice PRED qui permet de remonter jusqu'à D.

Pour tout $i \geqq 0$ on a $N_i = \{\text{cases } x / l_2^*(x) = i\}$ puisque la construction est identique à celle du paragraphe 3.2.

D'autre part il ne peut exister un chemin de D à A ayant $l_2^*(A)$ changements de direction et induisant une valeur $l_1(A)$ inférieure à celle trouvée par le chemin déduit du tableau PRED. En effet soit $\lambda = l_2^*(A)$. S'il existe deux prédecesseurs y à A dans $N_{\lambda-1}$ on mémorise dans PRED celui qui minimise $l_1(y) + d(A, y)$. On

est donc assuré d'obtenir ainsi parmi les chemins possédant un nombre minimal de changements de direction, un chemin traversant un nombre minimal de cases de D à A.

Notons que cet algorithme revient à construire pour chaque niveau N_i et pour toute case x de N_i, le bloc horizontal (ou vertical) maximal qui contient x (§ 3).

Remarque : Si une case x est atteinte horizontalement (resp. verticalement) alors il suffit de considérer à partir de x les cases que l'on peut atteindre verticalement (resp. horizontalement). Cette simple remarque a permis de diviser les temps d'exécution pratiquement par 2, mais a conduit à rajouter un tableau de taille MN.

Structure des données

La scène est représentée en mémoire par un vecteur, LAB (x), $1 \leqq x \leqq MN$, de façon à réduire dans les programmes le nombre de calculs d'indices. Chaque case x autorisée de la scène, contient en partie gauche $l_2(x)$ et $l_1(x)$ en partie droite. Cette association dans le même vecteur de 3 informations (définition de la scène, l_1 et l_2) a permis de réduire le temps d'exécution *et* la place mémoire utilisée.

Notons que cet algorithme revient à parcourir la scène en *largeur d'abord* (breadth-first-search) et que nous avons seulement besoin de 2 niveaux N_i et N_{i-1} simultanément en mémoire, lesquels sont gérés comme des files (si le parcours avait été de type profondeur d'abord, on aurait utilisé une pile [2], [11]).

5.2. Complexité

Nous notons τ le taux de cases interdites dans la scène. Nous avons choisi $0{,}10 \leqq \tau \leqq 0{,}35$, au-delà de 0,35 la scène ne présentant plus, en pratique, de chemin reliant le départ D à l'arrivée A; on prend arbitrairement $D=1$ et $A=M.N$ pour avoir des ordres de grandeur maximale pour une scène de taille donnée $M \times N$.

Le nombre de cases autorisées y, considérées dans l'algorithme est inférieur à $(1-\tau)\,MN$. Le nombre « moyen » de cases interdites sur une ligne est τN (espérance mathématique d'une loi binomiale de paramètre τ). La distance « moyenne » entre 2 cases interdites est $N/\tau N = 1/\tau$ d'où le nombre moyen de cases directement accessibles, horizontalement *ou* verticalement (*voir* remarque du paragraphe 5) à partir d'une case y est $1/\tau$. Ainsi le corps de la boucle, dans lequel le nombre d'instructions exécutées est indépendant de la taille de la scène, est traité $(1-\tau)\,MN.1/\tau$ fois, d'où une complexité en $O(MN)$. Un seul niveau N_{i-1}, est nécessaire pour déterminer N_i. On aura donc seulement deux tableaux simultanément en mémoire avec la scène elle-même. La taille de ces tableaux, qui

est égale au nombre de cases atteintes à un niveau, décroît avec τ. Nous avons obtenu les valeurs moyennes suivantes pour $M = N = 50$:

TABLEAU I

Taux.	0,10	0,15	0,20	0,25	0,30	0,35	0,40
Taille.	830	500	360	230	175	85	15

La complexité spatiale totale a toujours été inférieure à $4.MN$ dans tous les exemples traités.

Le tableau suivant donne les temps moyens d'exécution (en secondes) obtenus en fonction du taux et de la taille $= M = N$ de la scène (CII 10070 du C.I.C.R.C. d'Orléans-Tours).

TABLEAU II

Taille	Taux					
	0,10	0,15	0,20	0,25	0,30	0,35
20 × 20.	2	1	1	1	0	0
30 × 30.	2	2	2	2	2	1
40 × 40.	5	4	4	3	3	–
50 × 50.	7	6	6	5	4	3
60 × 60.	11	9	7	6	6	6
70 × 70.	13	9	7	6	6	4
80 × 80.	15	12	9	7	7	6
90 × 90.	20	15	13	10	8	–
100 × 100.	26	19	15	12	–	–

5.3. Commentaires

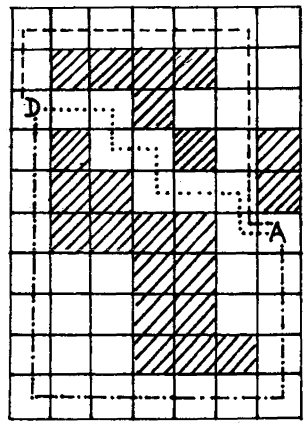
Nous avons pu borner le nombre maximal de cases considérées à partir de chaque case autorisée, par une « constante », $1/\tau$, indépendante de M et N, d'où un temps en $O(MN)$. Dans le pire des cas (worst case), quand il n'y a aucun obstacle dans l'aire, τ est nulle et $1/\tau = N$ c'est-à-dire que le nombre maximal de cases atteintes directement à partir d'une case donnée est borné par N (ou M), et $\lim_{\tau \to 0} (1-\tau) MN \cdot 1/\tau = MN^2$. La complexité maximale de l'algorithme HIERARCHIE est en $O(MN^2)$.

6. ALGORITHME MULTICRITÈRE

Soit n_1 le nombre de changements de direction et n_2 le nombre de cases traversées pour un chemin allant de D à une case x, a le coût d'un changement de

direction et b le coût dû au fait de traverser une case. La valuation associée par ce chemin à x est alors $v(x)=n_1 a+n_2 b$. On notera $v^*(x)=\min v(x)$, calculé sur l'ensemble des chemins allant de D à x. On cherche $v^*(A)$ et un chemin optimal correspondant.

Il est clair que les critères séparés ou une hiérarchie de critères ne construisent pas un chemin optimal pour v, comme le montre l'exemple ci-dessous dans lequel $a=2$ et $b=1$.

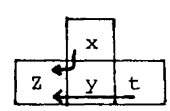


	n_1	n_2	v
chemin ----	3	13	19
chemin	6	9	21
chemin -.-.-	2	17	21

Exemple traité en annexe

Pour traiter ce problème, nous construisons l'arborescence des chemins allant de D à A. Soit $CH=\{D, x_1, x_2, \ldots, x_{i-1}, x_i\}$ un début de chemin allant de D à la case x_i. Supposons que $v(x_i)=k$. Soit $y \notin CH$ une case voisine de x_i. Si x_{i-1}, x_i, y est en ligne droite on pose $V(y)=k+b$ sinon on pose $v(y)=k+a+b$ puisqu'on a en plus un changement de direction pour passer de x_i à y en venant de x_{i-1}. On peut là aussi parler de décomposition en niveaux en posant : $y \in N_k \Leftrightarrow v(y)=k$. Une case y peut apparaitre dans plusieurs niveaux contrairement à l'algorithme précédent.

Exemple ($a=3$, $b=1$) : Considérons les 4 cases suivantes extraites d'une scène :



Supposons que l'on ait $v(x)=15$; alors $v(y)=16$ et $v(z)=20$. Supposons que l'on obtienne $v(t)=17$; alors $v(y)=18$. Cette valeur supérieure à $v(y)=16$ obtenue précédemment doit néanmoins être conservée puisqu'elle implique $v(z)=19$ meilleure que la valuation précédente de $v(z)$.

Ainsi, en posant $N_0 = \{D\}$, la détermination de $N_k (k \geqq a+b)$ se fait à partir des niveaux N_{k-b} et N_{k-b-a}. On doit donc conserver $(a+b+1)$ niveaux simultanément en mémoire, ce qui n'est pas une contrainte trop lourde car chaque niveau ne contient que peu de sommets et notablement moins qu'à l'algorithme précédent; en effet, dans l'algorithme précédent on pouvait atteindre directement à partir de chaque case $(\neq D)$ jusqu'à $\text{Max}(M, N)-1$ cases, contre au plus 3 ici.

On essaie de préciser et réduire le nombre d'éléments de chaque niveau à l'aide des remarques et corollaires suivants.

Remarque 1 : Soit une case y, $y \in N_k$. Alors toute occurence de y dans un niveau N_l avec $l-k \geqq a$ est inutile.

En effet toute case x obtenue à partir de l'occurence de y dans N_l possédera une valeur $v(x)$ supérieure ou égale à celle obtenue à partir de l'occurence de y dans N_k (règle classique de preclusion).

Remarque 2 : Si 2 cases x et y se suivent dans le même ordre sur plusieurs chemins à partir de D, on peut ne garder que l'occurrence de y qui présente la valuation $v(y)$ minimale.

En effet toute case z obtenue à partir de l'une quelconque des occurrences de y, l'est de la même façon (c'est-à-dire avec ou sans changement de direction), et l'ordre sur les valuations de y se conserve sur les valuations de tous les successeurs de y.

Cette remarque permet de fusionner plusieurs débuts de chemins, technique bien connue pour réduire la taille des arborescences (branch merging) [6].

COROLLAIRE 1 : *Toute case x possédant k cases voisines apparait au plus k fois dans l'arborescence.*

Les occurrences de x doivent en effet d'après la remarque précédente être suivies de cases différentes.

COROLLAIRE 2 - *Le nombre d'occurrences de sommets dans l'arborescence est inférieure à* 4 *fois le nombre de cases autorisées du labyrinthe.*

L'emplacement maximal est donc linéaire en MN et en fait très proche de MN grâce à la suppression de toute occurrence d'une case dans un niveau N_l dès que cette case apparaît dans le niveau N_k, avec $l-k \geqq a$. Notons que ce nombre d'occurrences est d'autant plus proche de MN que a est faible. Si $a=1$, pour toute case x on peut ne garder qu'une occurrence, celle pour laquelle $v(x)$ est minimale.

PROPOSITION : *Soit* $i = \min \{k / x \in N_k\}$; *alors* $i = v^*(x)$. *Dire que x appartient à* N_k *c'est dire qu'il existe un chemin de D à x induisant une valuation* $v(x)=k$. *Donc* $i = \min (v(x)) = v^*(x)$.

Algorithme :

Début

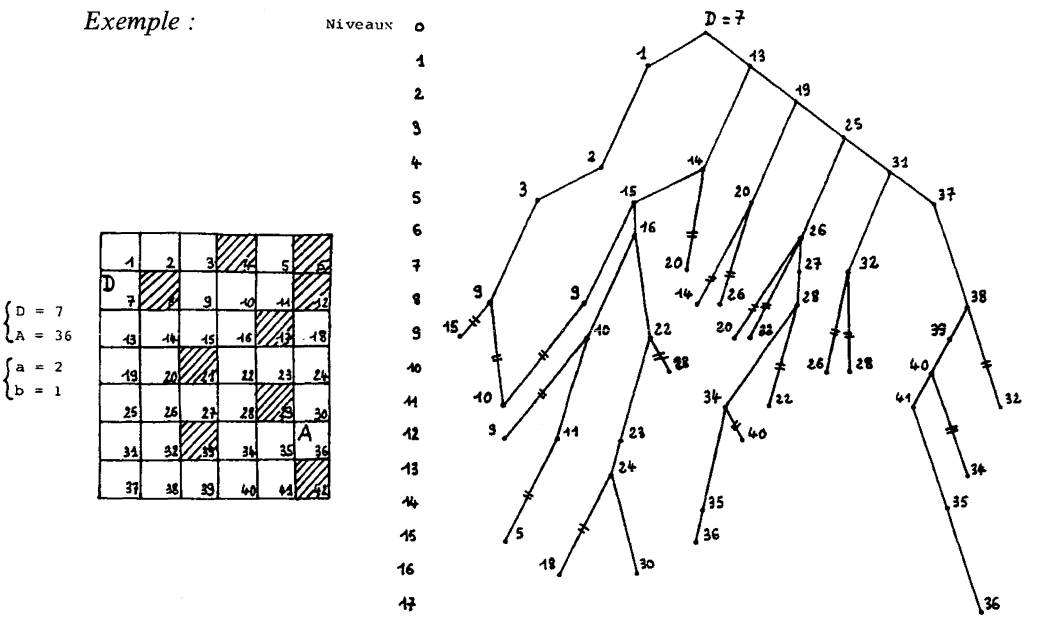
Initialisation Pour toute case x autorisée $\mathrm{LAB}(x) = -1$
sinon $\mathrm{LAB}(x) = -2$; $\mathrm{LAB}(A) = 1\,000$;
$N_0 = \{D\}$; $i = -1$

α : *Répéter* $i = i+1$
pour toute case y de N_i et pour toute case
x voisine de y et différente de PRED (y)
(prédecesseur de y)
faire : $l = i+a$ *si* (PRED(y), y, x) sont alignés
$l = i+a+b$ *sinon*
Si $\mathrm{LAB}(x) = -1$ placer x dans N_l; $\mathrm{LAB}(x) = l$;
$\mathrm{PRED}(x) = y$;
Sinon on a $\mathrm{LAB}(x) = k_0$;
Si $l \geqq k_0$: pour tout $k \in [k_0, l]$ s'il
existe une case x de N_k telle que
$\mathrm{PRED}(x) = y$ aller en α;
Si pour toute case x de N_k, $\mathrm{PRED}(x) \neq y$,
mettre x dans N_l;
Si $l < k_0$ mémoriser x dans N_l;
$\mathrm{LAB}(x) = l$; $\mathrm{PRED}(x) = y$;
Tant que $i < \mathrm{LAB}(A) - b$
Imprimer $\mathrm{LAB}(A)$ et le chemin correspondant.

Fin :

Remarque : Le problème n'a pas de solution s'il existe $a+b-1$ niveaux consécutifs vides, ce qui ne pose aucun problème au niveau de l'agorithme et du programme.

Exemple :

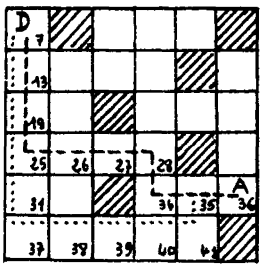


L'examen de N_{14} implique $v^*(A) = 14+b = 15$.

Le chemin optimal est $\{7, 13, 19, 25, 26, 27, 28, 34, 35, 36\}$.

Même exemple avec $a=3$, $b=1$:

Sur cet exemple, le principe d'optimalité de Bellmann n'est pas vérifié; le plus court chemin pour atteindre la case 35 (noté . . . . .) n'est pas la portion entre D et 35 du plus court chemin pour atteindre la case 36 en passant par 35 (noté — — — —).



En effet le chemin en pointillé traverse 12 cases et présente trois changements de direction d'où une longueur de 21 entre D et A et une longueur de 17 entre D et la case 35. Le chemin avec des tirés comporte 10 cases et 3 changements de direction soit une longueur de 19 entre D et A, et une longueur de 18 entre D et la case 35.

6.1. Complexité

Chacune des $(1-\tau)MN$ cases autorisées est mémorisée au plus 4 fois dans l'arborescence et pour chacune de leurs 3 voisines on peut être amené à considérer au plus a niveaux dont la taille a toujours été très largement bornée par N dans nos exemples. Au pire, si $\tau=0$, le nombre maximal de cases atteintes à un niveau pourrait être de l'ordre de $N\sqrt{2}$ comme le montre la figure ci-contre dans laquelle $a=2$, $b=1$ et la valeur de chaque case représente le niveau dans lequel appartient chaque case, c'est-à-dire la longueur minimale pour un chemin allant de D à cette case. Si τ croît, le nombre de niveaux augmente, et le nombre de cases par niveau décroît.

D	1	2	3	4	5	6	7	8	
1	4	5	6	7	8	9	10		
2	5	6	7	8	9	10			
3	6	7	8	9	10				
4	7	8	9	10					
5	8	9	10						
6	9	10							
7	10					etc...			
8									

En acceptant cette majoration en $O(N)$ on est conduit à un temps en $O(MN^2)$. La place mémoire utilisée n'a jamais excédé $2MN$. Le tableau suivant donne les temps moyens d'exécution (en secondes) en fonction de $M \times N$. Il est à noter que ces temps sont pratiquement indépendants du taux τ.

TABLEAU III

Taille.	20×20	30×30	40×40	50×50	60×60	70×70	80×80	90×90	100×100
Durée.	1	2	4	5	6	7	8	11	13

6.2. Commentaires

Il peut paraître paradoxal qu'un algorithme en $O(MN^2)$ donne certains temps moyens meilleurs que ceux d'un algorithme en $O(MN)$. En effet HIÉRARCHIE est plus lent que MULTICRITÈRE, mais seulement lorsque τ est inférieur à 0,20-0,25. On sait, d'autre part, que lorsque τ décroît, la rapidité de HIÉRARCHIE décroît aussi, puisqu'il y a, pour un nombre de niveaux constant, de plus en plus de cases à examiner, pour atteindre avec $\tau \sim 0$, le pire des cas. Avec MULTICRITÈRE, lorsque τ croît, il y a plus de niveaux à examiner mais moins de cases dans chaque niveau, et réciproquement lorsque τ décroît, ce qui explique l'indépendance des temps de MULTICRITÈRE par rapport à τ. Dans l'algorithme Hiérarchie, nous avons pu borner le nombre maximal (notons le γ) de cases considérées à partir de chaque case autorisée par une « constante » $1/\tau$, indépendante de N, d'où un temps en $O(MN)$. Au contraire, dans l'algorithme multicritère, γ peut être *au pire* de l'ordre de N, mais en pratique lorsque τ est faible, $1/\tau$ est supérieur à γ ce qui explique la différence des temps. $O(MN^2)$ représente la *complexité maximale* pour les 2 algorithmes car $\lim_{\tau \to 0} 1/\tau = N$ (c'est-à-dire que le nombre maximal de cases atteintes directement à partir d'une case donnée est aussi borné par N dans l'algorithme multicritère, si $\tau = 0$ (aucun obstacle dans l'aire). On connaît néanmoins les limites de la notation en $O(\ )$ (*voir* par exemple [12]).

7. PROGRAMMES

A l'image de J. Arsac, nous pensons qu'il est impossible de dissocier algorithmique et programmation. Nous donnons en annexe les 2 programmes documentés. Nous avons essayé d'affiner au mieux la programmation : 3 versions successives ont été testées pour l'algorithme Hiérarchie dont bien sûr

seule la plus rapide a été conservée. Notons par exemple que le fait d'associer dans la même mémoire l_1 et l_2 relatives à la même case x, a permis en supprimant une structure répétitive de diviser par 4 les temps d'exécution.

Notons enfin une heuristique utilisée dans l'algorithme multicritère pour retrouver plus rapidement le chemin de la case arrivée à la case départ à l'aide des valuations : une case x étant trouvée, le prédécesseur essayé est d'abord la case située au-dessus de x plutôt qu'au dessous, à gauche de x plutôt qu'à droite, vues les dispositions des cases départ et arrivée.

Remarque : Des mots de 32 bits sont nécessaires pour Hiérarchie de façon à avoir 8 chiffres significatifs par mémoire (4 pour l_1, 4 pour l_2).

CONCLUSION

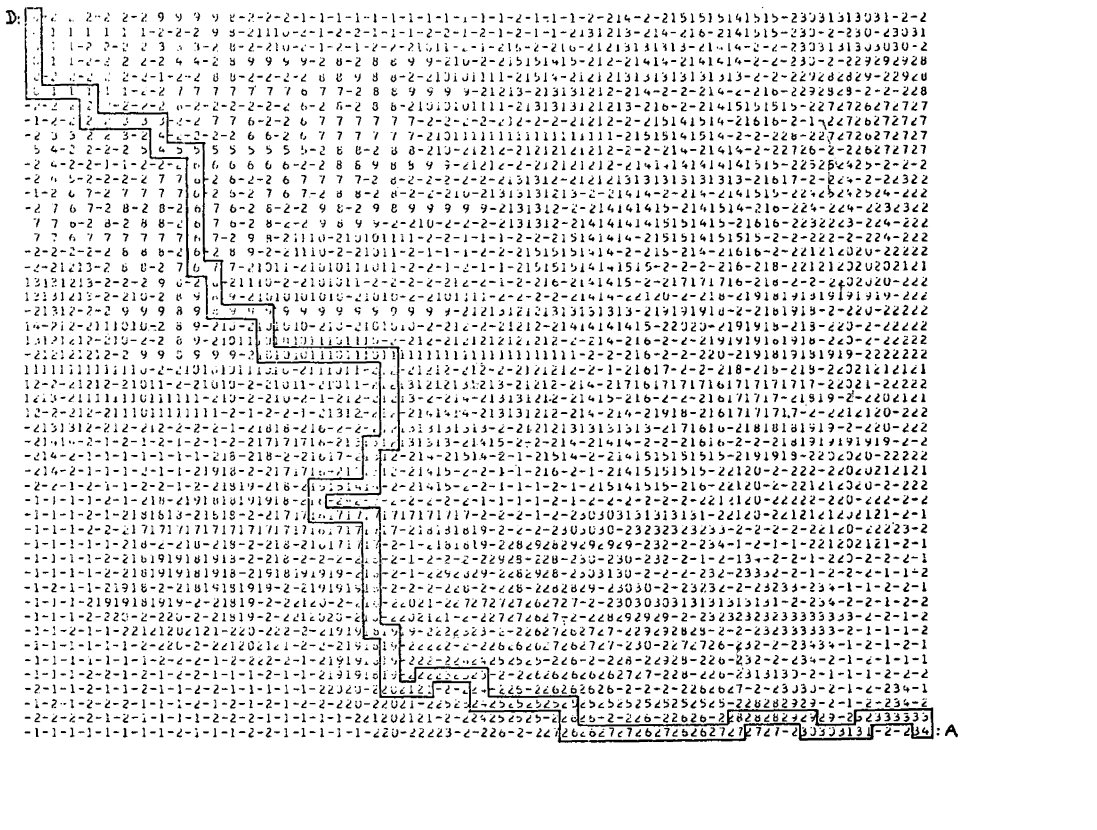
Nous avons traité ici les problèmes de parcours dans une aire comme des problèmes de décomposition d'un graphe en niveaux, la décomposition variant selon les critères d'optimalité choisis. Les algorithmes polynomiaux développés ici ont conduit à des temps très faibles pour des scènes dont nous avons fait varier la taille jusqu'à 100×100 (moins de 30 secondes sur un CII 10070). Ces algorithmes sont utilisables dès que la scène est entièrement déterminée ce qui peut se faire dans une première étape par des procédures énumératives (avec heuristiques) du domaine de l'intelligence artificielle.

ANNEXE

Nous donnons en annexe les programmes correspondants à ces deux algorithmes ainsi que des exemples de scènes tirées au hasard avec les résultats fournis par ces algorithmes.

Hiérarchie de critères : Un exemple de résultats obtenus pour $M = N = 50$ et un taux de 0,35.

Si la valeur d'une case $= -2$: case interdite; $= -1$: case non atteinte; $= i \geqq 0$: case atteinte en i changements de direction à partir de D.



PARCOURS

1 251 254 354 358 458 460 850 861 1011 1014 1164 1171 1471 1478 1620 1617 1717 1720 2120
2121 2321 2323 2273 2276 2376 2381 2481 2490 2440 2444 2494 2497 2447 2450 2500

NOMBRE DE CASES TRAVERSEES = 116
NOMBRE DE CHANGEMENTS DE DIRECTION = 34
DUREE= 5 (SECONDES)

Algorithme multicritère : On reprend ici les deux exemples simples du paragraphe 6. Dans chaque niveau, chaque case x est mémorisée sous la forme d'un entier avec 8 chiffres significatifs, les 4 de gauche indiquent le numéro de la case x, les 4 de droite indiquent le numéro de la case qui a permis d'atteindre x.

```
        LABYRINTHE DE DEPART( 7, 6):

DEPART=   7
ARRIVEE=  36
                 1   2   3   0   5   0
                (7)  0   9  10  11   0
                13  14  15  16   0  18
                19  20   0  22  23  24
                25  26  27  28   0  30
                31  32   0  34  35 (36)
                37  38  39  40  41   0
NIVEAU:   0       7.C000
NIVEAU:   1      13.C007   1.0007
NIVEAU:   2      19.C013
NIVEAU:   3      25.C019
NIVEAU:   4      14.C013   2.0001  31.C025
NIVEAU:   5      20.C019  15.0014   3.C002  37.C031
NIVEAU:   6      26.C025  16.0015
NIVEAU:   7      32.C031  27.0026
NIVEAU:   8       9.C015   9.0003  38.0037  28.C027
NIVEAU:   9      22.C016  10.0016  39.0038
NIVEAU:  10      40.C039
NIVEAU:  11      34.C028  41.0040
NIVEAU:  12      23.C022  11.0010
NIVEAU:  13      24.C023
NIVEAU:  14      35.C034  35.0041
NIVEAU:  15       5.C011 (36).0035

V(C)=0   V(A)=   15
              LABYRINTHE FINAL
              1   4   5  -2  15  -2
              0  -2   8   9  12  -2
              1   4   5   6  -2  16
              2   5  -2   9  12  13
              3   6   7   8  -2  16
              4   7  -2  11  14  15
              5   8   9  10  11  -2

NOMBRE DE CASES AUTORISEES CANDIDATES
A ENTRER DANS L ARBORESCENCE:  54
NOMBRE DE CASES DANS L ARBORESCENCE:  34
```

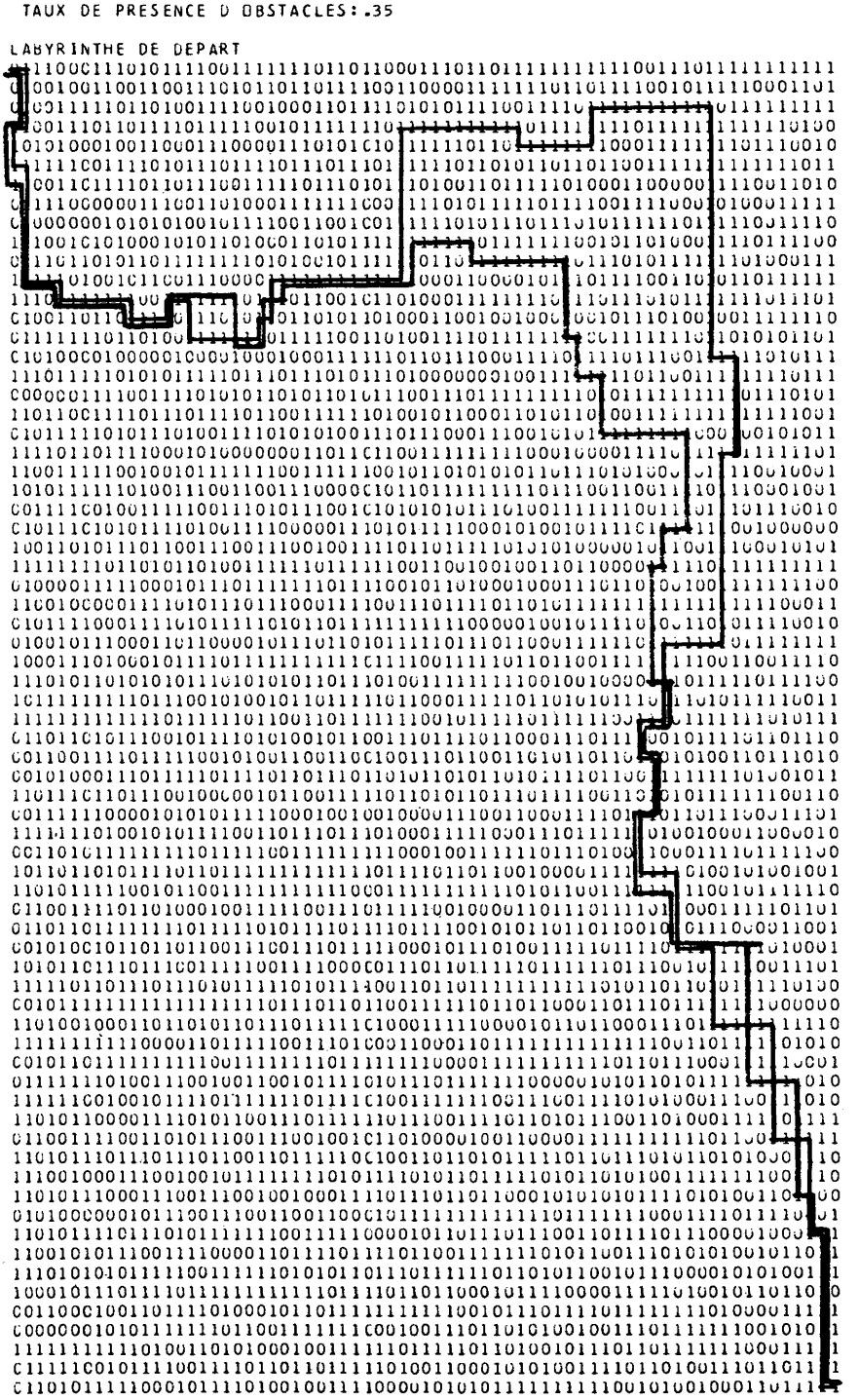
```
        LABYRINTHE DE DEPART(10, 7):

DEPART=  15
ARRIVEE=  42
                 1   2   3   4   5   6   7
                 8   0   0   0   0  13  14
                (15) 16  17   0  19  20  21
                22   0  24  25   0  27   0
                29   0   0  32  33  34   0
                36   0   0   0   0  41 (42)
                43  44  45   0   0  48  49
                50  51  52   0   0  55  56
                57  58  59   0   0   0  63
                64  65  66  67  68  69  70
NIVEAU:   0      15.C000
NIVEAU:   1      16.C015  22.0015   8.0015
NIVEAU:   2      17.C016  29.0022   1.C008
NIVEAU:   3      30.0029
NIVEAU:   4      23.C036
NIVEAU:   5      24.C017   2.0001  50.C043
NIVEAU:   6       3.C002  57.0050
NIVEAU:   7      44.C043   4.0003  64.C057
NIVEAU:   8      25.C024  51.0050  45.0044   5.C004
NIVEAU:   9      58.C057  52.0051   6.C005
NIVEAU:  10      65.C064  59.005B   7.C006
NIVEAU:  11      32.C025  66.0065
NIVEAU:  12      13.C006  67.0066
NIVEAU:  13      14.0007  20.0013  68.0067
NIVEAU:  14      33.C032  21.0014  27.C020  69.C068
NIVEAU:  15      34.C033  34.0027  70.C069
NIVEAU:  16      15.C020  41.0034
NIVEAU:  17      48.C041
NIVEAU:  18      31.C034  63.0070  55.0048
NIVEAU:  19     (42).C041  56.0063

V(C)=0   V(A)=   15
              LABYRINTHE FINAL
              2   5   6   7   8   9  10
              1  -2  -2  -2  -2  12  13
              0   1   2  -2  16  13  14
              1  -2   5   8  -2  14  -2
              2  -2  -2  11  14  15  -2
              3  -2  -2  -2  -2  16  19
              4   7   8  -2  -2  17  20
              5   8   9  -2  -2  18  19
              6   9  10  -2  -2  -2  18
              7  10  11  12  13  14  15

NOMBRE DE CASES AUTORISEES CANDIDATES
A ENTRER DANS L ARBORESCENCE:  70
NOMBRE DE CASES DANS L ARBORESCENCE:  49
```

Exemple (70×70) :



Les résultats des deux algorithmes sur cet exemple sont donnés page suivante, où les cases sont numérotées de 1 à 4900.

Multicritère :

```
TAUX DE PRESENCE D OBSTACLES=.35

V(D)=0   V(A)=  268
                LABYRINTHE FINAL

NOMBRE DE CASES AUTORISEES CANDIDATES
A ENTRER  DANS L ARBORESCENCE: 51953
NOMBRE DE CASES DANS L ARBORESCENCE: 24261
PARCOURS: 4900 4849 4829 4759 4639 4619 4549 4479 4409 4339 4336 4268 4198 4197 4127 4057 3987 3917 3847 3777
          3776 3775 3774 3773 3703 3633 3563 3493 3423 3353 3283 3282 3281 3280 3279 3278 3277 3207 3137 3067
          3066 3065 3064 2994 2924 2854 2784 2765 2715 2645 2575 2574 2504 2434 2435 2436 2366 2296 2295 2225
          2155 2085 2015 1945 1875 1876 1806 1736 1737 1738 1668 1598 1528 1458 1388 1387 1386 1385 1384 1383
          1382 1381 1311 1241 1171 1170 1169 1099 1029 1028  958  888  818  748  747  746  745  744  743  742
           741  740  670  669  668  667  666  665  735  805  804  803  802  801  800  799  798  797  796  795
           794  864  863  862  932 1002 1001 1000  930  860  859  858  857  856  855  854  924  923  922  921
           851  850  849  848  847  846  845  775  774  773  772  702  632  562  492  422  421  351  281  211
           212  142   72    2    1
NOMBRE DE CHANGEMENTS DE DIRECTION:   52
NOMBRE DE CASES TRAVERSEES:   165
DUREE=     4
```

Hiérarchie :

```
---------- LABYRINTHE FINAL ----------

PARCOURS
    1    2  212  211  351  352  772  775  845  851  921  924  854  856  996 1002  932  933  793  804
  204  254  324  330  190  200 1110 1112 1462 1461 2161 2156 2436 2434 2574 2575 2785 2784 2994 2997
 3277 3280 3560 3565 3985 3988 4338 4339 4899 4900
NOMBRE DE CASES TRAVERSEES =  187
NOMBRE DE CHANGEMENTS DE DIRECTION =  48
```

Programmes :

HIERARCHIE.3

```
C------------------MARK(I) REPRESENTE COMMENT ON ATTEINT LA CASE I
C------------------LAB REPRESENTE LE SCENE,DE DIMENSION M*NN
C------------------N REPRESENTE LES 2 NIVEAUX NECESSAIRES DANS L ALGORITHME
C------------------PRED(I) EST LE PREDECESSEUR DE LA CASE I
C                  LE CHEMIN SOLUTION EST MEMORISE DANS IPARC
      DIMENSION MARK(2500)
      DIMENSION LAB(2500),N(2,2000),PRED(2500),IOP(4)
      DIMENSION IPARC(300),ITEMP(5),IMPRIM(10)
      INTEGER CHIF(9)
      LOGICAL TEST(4)
      INTEGER A,D,TAUX
      INTEGER DISTIX,DISTIY,DIMIL
      DATA CHIF/'1','2','3','4','5','6','7','8','9'/
      DIMIL=10000
      LIM=1500
      READ(7,5)M,NN
5     FORMAT(2I2)
      MN=M*NN
      MAX=MAXO(M,NN)
      READ(7,7)IMPRIM
7     FORMAT(10A1)
      ICHIF=NN/10
      IMPRIM(6)=CHIF(ICHIF)
      IOP(1)=1
      IOP(2)=-1
      IOP(3)=NN
      IOP(4)=-NN
      DO 1789 TAUX=10,40,5
      CALL TIMER(IVA,1,0)
      WRITE(8,10)TAUX/100.
10    FORMAT(1H1,20X,'TAUX DE PRESENCE D OBSTACLES:',F3.2)
      CALL TIME(ITEMP,IX)
      IX=123579
      OUTPUT(8)IX
      RMN=FLOAT(MN)
      DO 15 I=1,MN
      LAB(I)=0
      CALL HASARD(IX,LX,R,AN,0.,10.)
      IX=LX
      TO=TAUX/10.
      IF(AN.GT.TO)LAB(I)=1
15    CONTINUE
      WRITE(8,20)
20    FORMAT(20X'LABYRINTHE DE DEPART')
21    FORMAT(20X,70I1)
      CALL HASARD(IX,LX,R,AN,1.,RMN)
      IX=LX
      D=AN
      D=1
      CALL HASARD(IX,LX,R,AN,1.,RMN)
      A=AN
      A=MN
      DO 25 I=1,MN
25    LAB(I)=LAB(I)-2
      LAB(A)=-1
      IF(NN.LE.60)WRITE(8,IMPRIM)(LAB(I7),I7=1,MN)
      WRITE(8,30)D,A
30    FORMAT(5X,'CASE DE DEPART:',I4,' CASE D ARRIVEE :',I4)
```

```
      LAB(D)=0
      N(1,1)=D
      NVIR=-1
      KO=1
      PRED(D)=0
40    IALF=KO
      KO=0
      NVIR=NVIR+1
       WRITE(8,41)NVIR
41    FORMAT(30X,'ETAT DU LABYRINTHE AU NIVEAU:',I4)
      J=1
      IF(NVIR-NVIR/2*2.NE.0)J=2
      DO 95 K=1,IALF
      IY=N(J,K)
      DO 90 ITEST=1,4
      IF(IY.EQ.D) GOTO 42
      IF(ITEST.LE.2.AND.MARK(IY).LE.2) GOTO 90
      IF(ITEST.GT.2.AND.MARK(IY).GT.2) GOTO 90
42    DO 85 IND=1,MAX
      IX=IY+IOP(ITEST)*IND
      TEST(1)=IX.GT.NN*((IY-1)/NN+1)
      TEST(2)=IX.LT.NN*((IY-1)/NN)+1
      TEST(3)=IX.GT.MN
      TEST(4)=IX.LE.0
      IF(TEST(ITEST))GOTO90
      IF(LAB(IX)+1)90,45,50
C  EN 45 ON A LAB(IX)=-1
45    KO=KO+1
      IF(KO.GT.LIM) GOTO 1600
      N(3-J,KO)=IX
      LAB(IX)=NVIR*DIMIL
      DISTIY=LAB(IY)-LAB(IY)/DIMIL*DIMIL
      LAB(IX)=LAB(IX)+DISTIY+IND
      PRED(IX)=IY
      MARK(IX)=ITEST
      GOTO 85
50    IF(LAB(IX)/DIMIL-NVIR+1)70,85,52
C  EN 52 LAB(IX)=NVIR
52    DISTIX=LAB(IX)-LAB(IX)/DIMIL*DIMIL
      DISTIY=LAB(IY)-LAB(IY)/DIMIL*DIMIL
      IF(DISTIX.LE.DISTIY+IND)GOTO 85
      LAB(IX)=NVIR*DIMIL+DISTIY+IND
      PRED(IX)=IY
      MARK(IX)=ITEST
      GOTO 85
70    IF(LAB(IX)/DIMIL.EQ.NVIR-2) GOTO 90
75    WRITE(8,80)IX,LAB(IX),NVIR,IY,IND,ITEST,K
80    FORMAT(20X'ERREUR LAB(',I4,')=',I4,'NVIR= ',I3,4I6)
85    CONTINUE
C---------------------------- ON CONTINUE MEME DIRECTION
90    CONTINUE
C---------------------------- ON CHANGE DE DIRECTION
95    CONTINUE
C---------------------------- ON PASSE AU POINT SUIVANT
      WRITE(8,100)NVIR,KO
100   FORMAT(10X,'NBRE D ELEMENTS DU NIVEAU ',I4,' = ',I6)
C......... EST-ON ARRIVE ????
      IF(LAB(A).NE.-1)GOTO 110

C......... PEUT ON PARVENIR A L'ARRIVEE
       IF(KO.NE.0)GOTO 40
       WRITE(8,105)D,A
105   FORMAT(20X,'PAS DE SOLUTION  ENTRE',I4,'ET',I4)
      GOTO 1789
110   IPRED=A
      LONG=LAB(A)-LAB(A)/DIMIL*DIMIL
      WRITE(8,115)
115   FORMAT(40X,'---------- LABYRINTHE FINAL ----------')
      DO 120 I7=1,MN
120   LAB(I7)=LAB(I7)/DIMIL
      IF(NN.LE.60)WRITE(8,IMPRIM)(LAB(I7),I7=1,MN)
```

```
C..... CONSTRUCTION DU VECTEUR  <== PARCOURS
      K=0
125   IF(IPRED.EQ.0)GOTO 130
      K=K+1
      IPARC(K)=IPRED
      IPRED=PRED(IPRED)
      GOTO 125
130   K=K+1
      IPARC(K)=0
      WRITE(8,135)
135   FORMAT(///,10X,'PARCOURS')
      WRITE(8,140)(IPARC(I),I=K,1,-1)
140   FORMAT(10X,20I5)
      WRITE(8,145)LONG+1
145   FORMAT(10X'NOMBRE DE CASES TRAVERSEES =',I5)
      WRITE(8,150)NVIR
150   FORMAT(10X'NOMBRE DE CHANGEMENTS DE DIRECTION =',I5)
      CALL TIMER(IVU,1,0)
      JTEMP=IVA-IVU
      WRITE(8,165)JTEMP
165   FORMAT(10X,'DUREE=',I6)
      GOTO 1789
1800  WRITE(8,1805)
1805  FORMAT(10X'PAS ASSEZ DE PLACE DANS LE TABLEAU N...')
1789  CONTINUE
      STOP
      END
```

BIBLIOGRAPHIE

1. M. DELANNOY, M. LEROI et M. BOURTON, *Recherche d'un chemin optimal dans une scène planifiée*, 2e congrès A.F.C.E.T.-I.R.I.A.; *Reconnaissance des formes et intelligence artificielle*, septembre 1979, Toulouse.
2. GONDRAN et MINOUX, *Graphes et algorithmes*, Eyrolles, 1979.
3. HART, NILSSON et RAPHAEL, *A formal Basis for the Heuristic Determination of Minimum Cost Paths*, I.E.E.E., vol. SSC-4, n° 2, 1968.
4. GIRALT, SOBEK et CHATILA, *A Multi-Level Planning and Navigation System for a Mobil Robot: a First Approach to Hilare*, Sixth International Joint Conference on Artificial Intelligence, 20-24/8/1979, Tokyo.
5. THOMPSON, *The Navigation system of the J.P.L. Robot*, Proceedings of the I.J.C.A.I., août 1977, p. 749-757.
6. BITNER et REINGOLD, *Backtrack programming techniques*, Comm. of the A.C.M., vol. 18, n° 11, novembre 1975, p. 651-656.
7. J. R. CAYROL, *Conception de la simulation d'un robot*, Thèse 3e cycle, 1978, Toulouse.
8. POHL, *Heuristic Search Viewed as Path Finding in a Graph*, Artificial Intelligence, vol. 1, 1970, p. 193-204.
9. DORAN et MICHIE, *Experiements With the Graph Traverser Program*, Proc. Roy. Soc., vol. A, n° 294, 1966, p. 235.
10. M. SHIMURA, *Heuristic Problem Solving by Tree Search*, Systems Computers Controls, vol. 8, n° 4, 1977.
11. D. E. KNUTH, *The Art of Computer Programming*, vol. 1. *Fundamentals Algorithms*, Addison-Wesley, 1973.
12. MEYER et BAUDOIN, *Méthodes de programmation*, Eyrolles, 1978.