

Planification et contrôle

Cours ENSTA Paris - ROB316 / 03

David FILLIAT

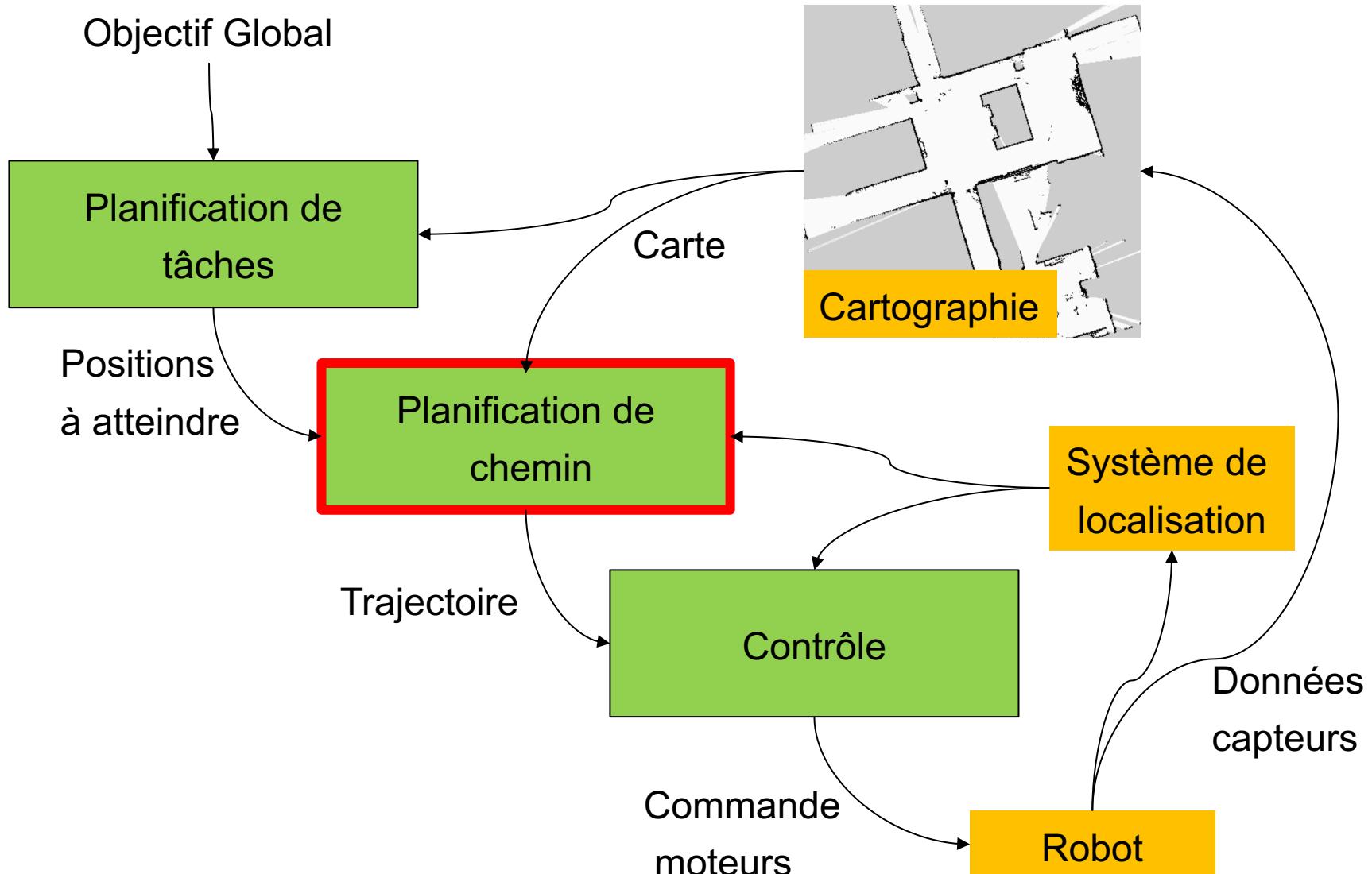
david.filliat@ensta-paris.fr

2019-2020



ENSTA

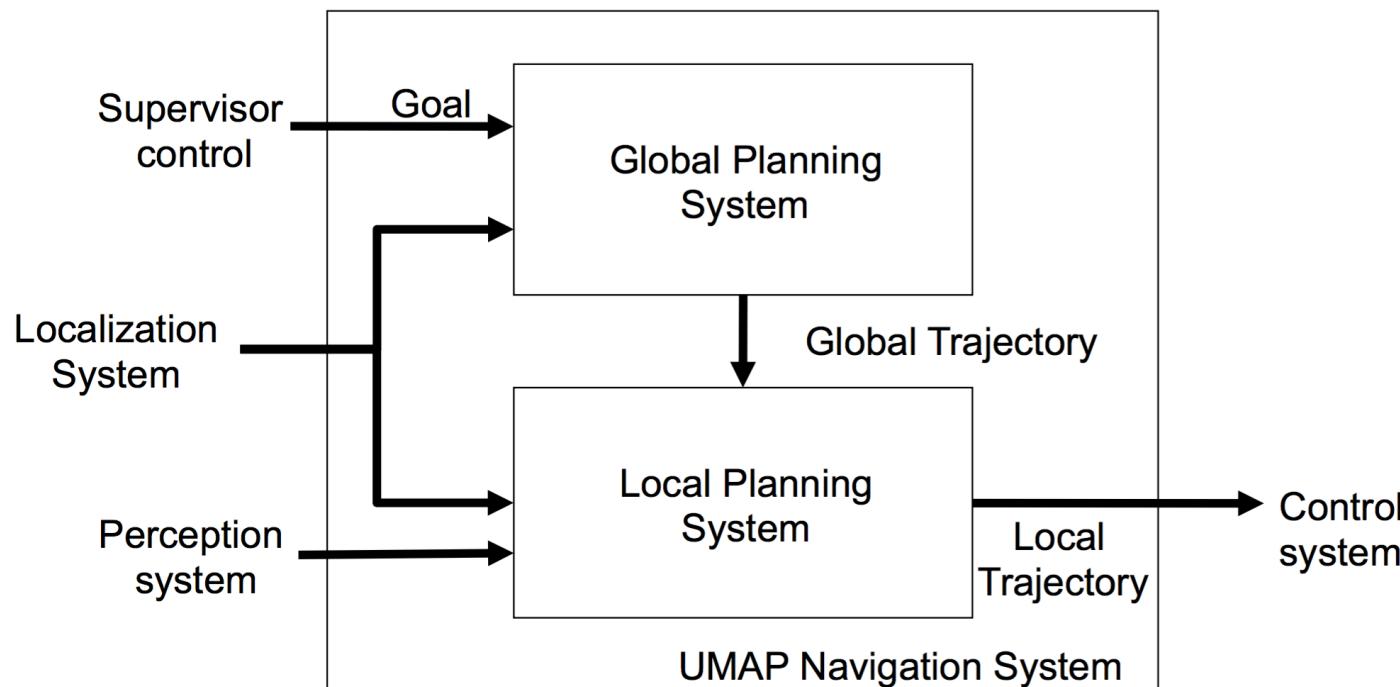




Planification de trajectoire

- Planification dans une carte
- Discrétisation des espaces de recherche
- Recherche de chemin dans un graphe
- Application pratique

Deux niveaux de planification (cf architecture hybrides)



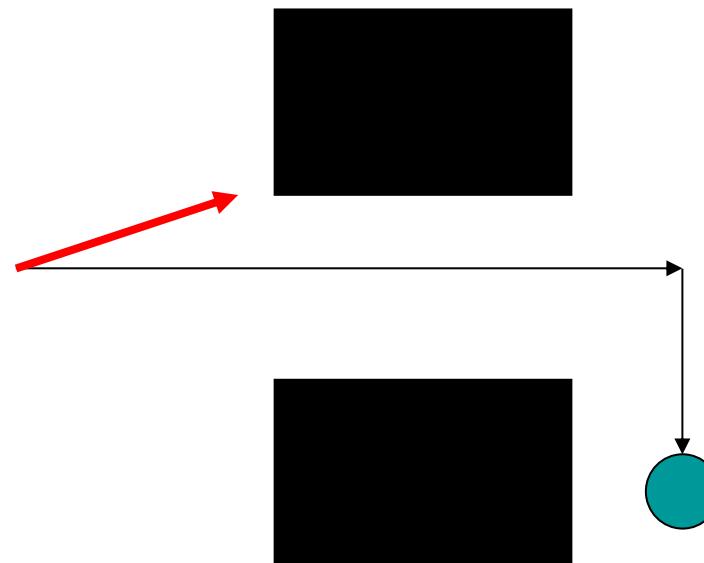
Deux types de plans haut-niveau

- Chemin
- Politique

cf Apprentissage par renforcement, champs de potentiels ...

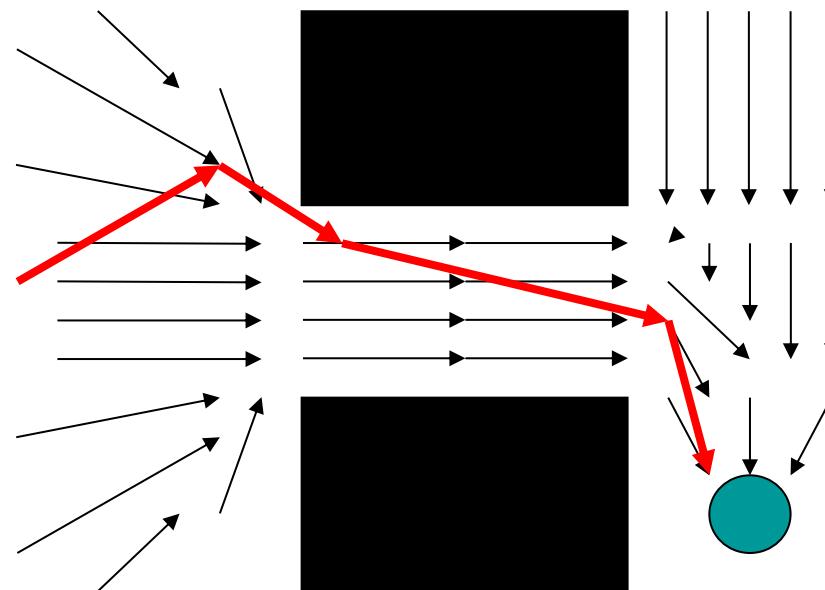
Plan = suite de nœuds et de liens ou trajectoire

- Succession prédefinie d'actions
- Problèmes en cas d'exécution imprécise -> replanification
- Problèmes notamment dans les architectures hybrides



Plan = action associée à chaque nœud ou à chaque position (discrétisée)

- Calcul initial plus complexe que pour un chemin (pas d'heuristiques)
- Action : localisation puis action associée à la position
- Pas de replanification en cas d'erreur d'exécution (sauf changement de carte)

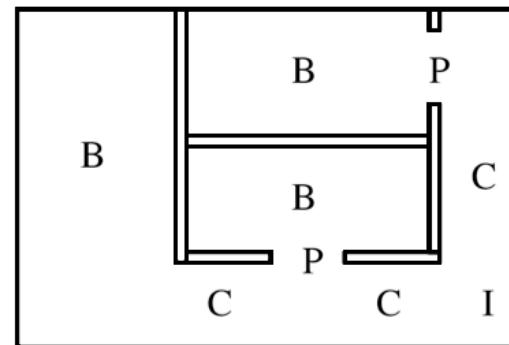


Discrétisation des espaces de recherche



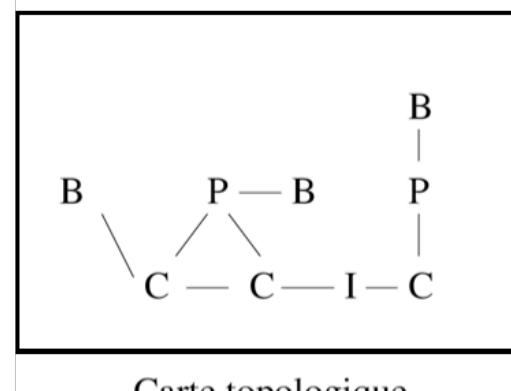
Cartes topologiques

- Graphe de lieux et de transitions entre lieux
- perception sans modèle métrique



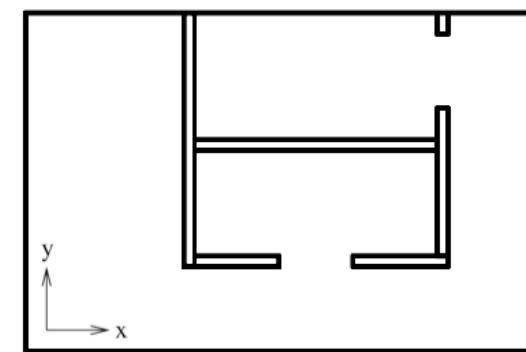
Cartes métriques

- Ensemble d'objets dans un espace commun
- perception avec modèle métrique



Carte topologique

Environnement réel



Carte métrique

Planification : Recherche d'un chemin entre la position courante et un but

- Carte supposée connue
- Position supposée connue (au moins de manière approximative)

Discrétisation de l'espace

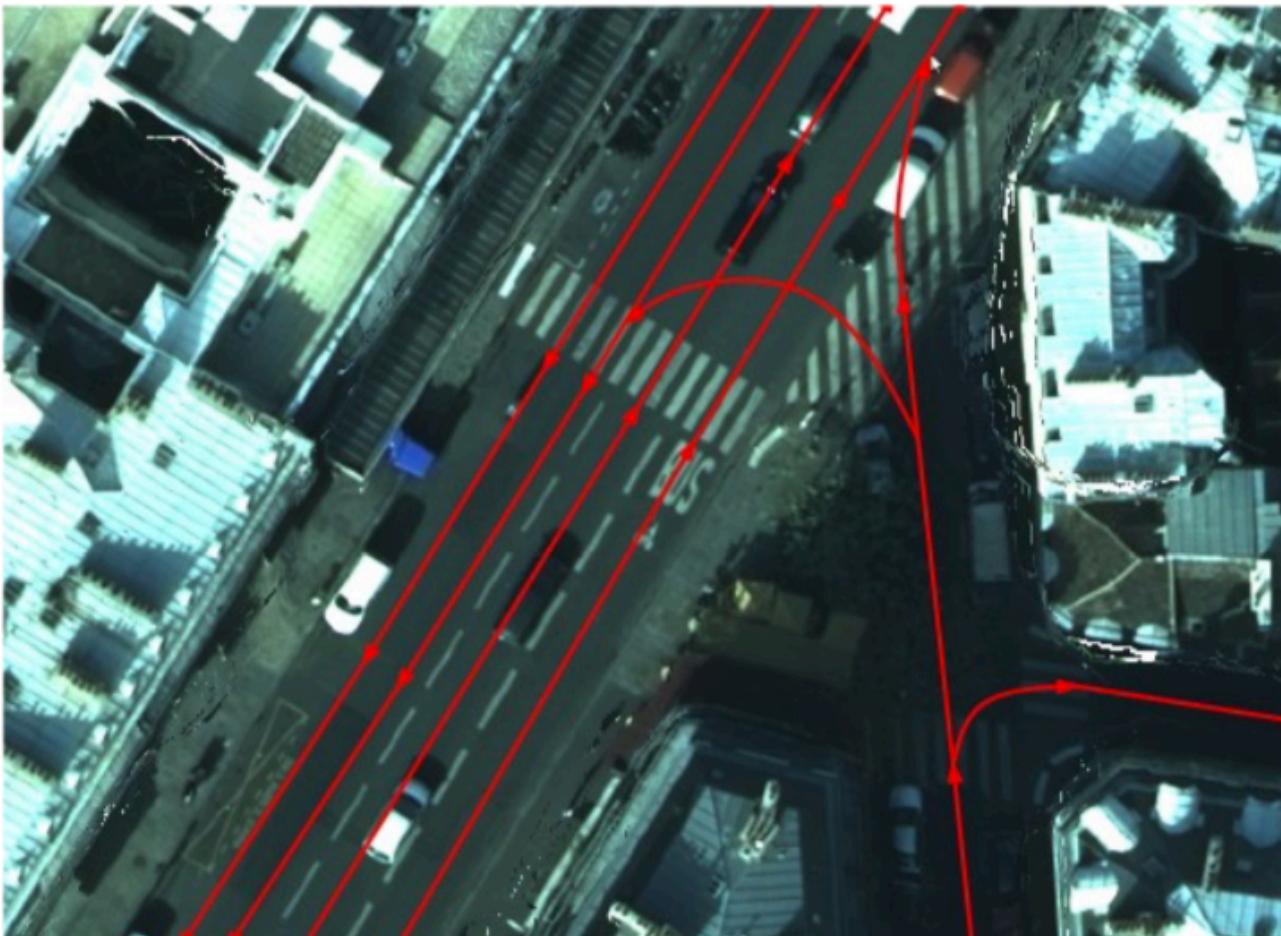
- Représentation du problème sous forme de graphe
- Planification = recherche de chemin dans un graphe
- En général, chemin minimisant un certain coût
 - algorithmes classiques

Carte topologique

- Graphe discret -> représentation déjà adaptée
- En général, nombre de nœuds assez faible -> calcul rapide
- Ne passe que par des chemins connus

Réseau routier

- Discrétisation déjà disponible
- Contient tous les chemins ‘canoniques’



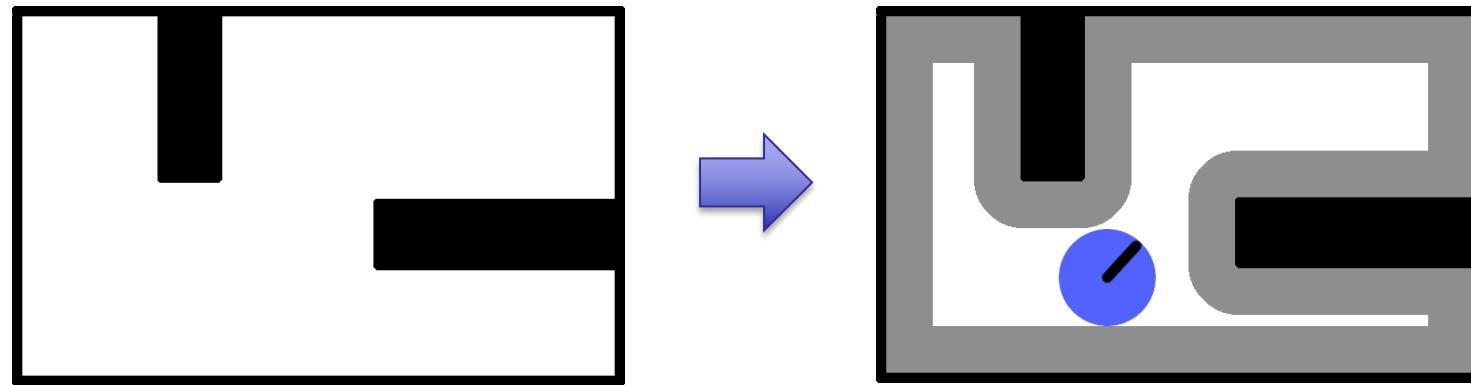
Espace continu -> discrétisation pour planifier dans le graphe correspondant

- Discrétisation en cellules
- Discrétisation en chemins

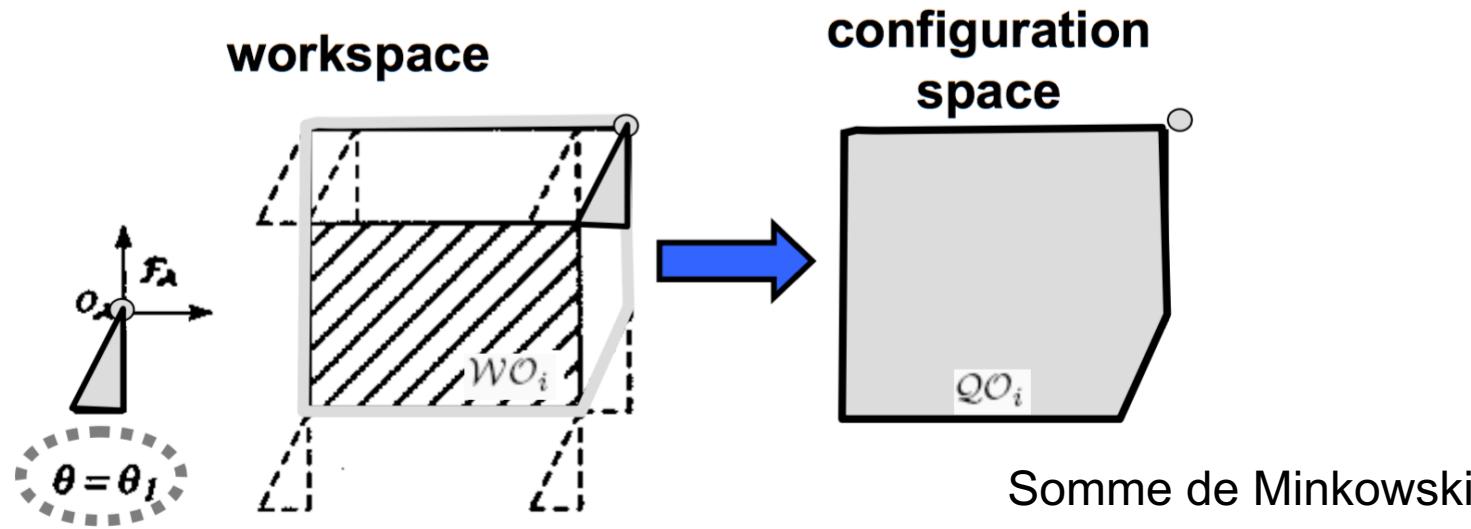
Espace des configurations

- Espace des positions du robot
- Obstacle physique \Leftrightarrow obstacle dans l'espace de config
- Trajectoire d'un point de l'espace de configuration
 \Leftrightarrow trajectoire du robot

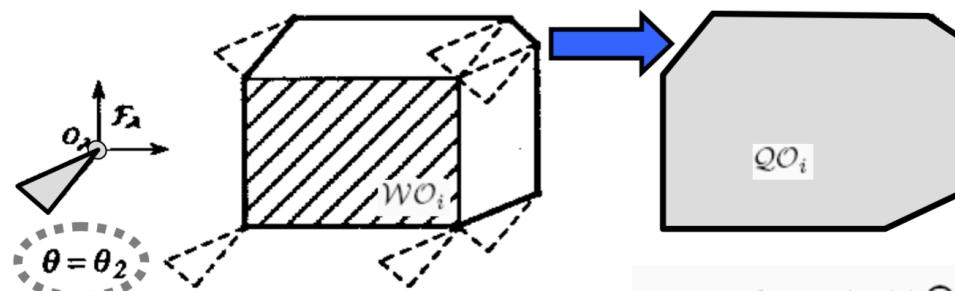
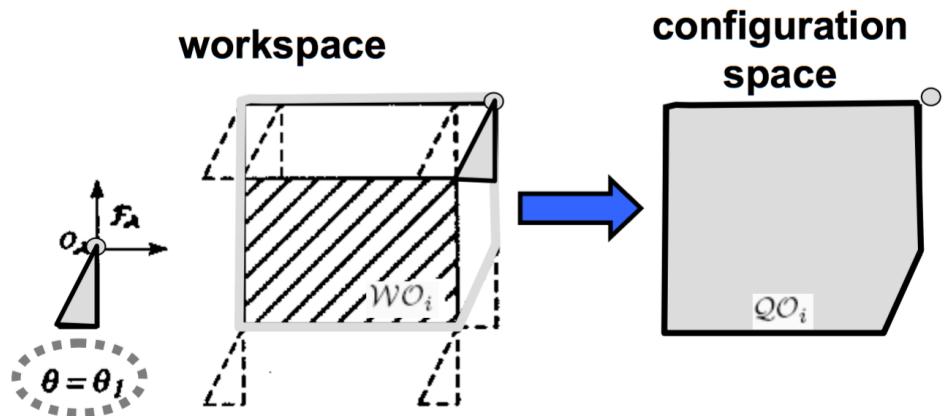
Cas d'un robot circulaire



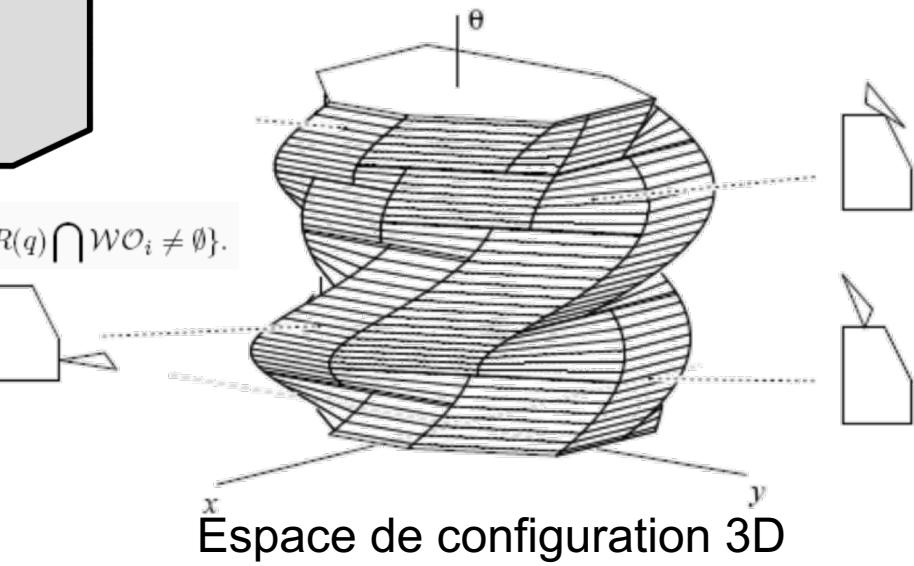
Cas d'un robot polygonal en translation



Cas d'un robot en translation/rotation



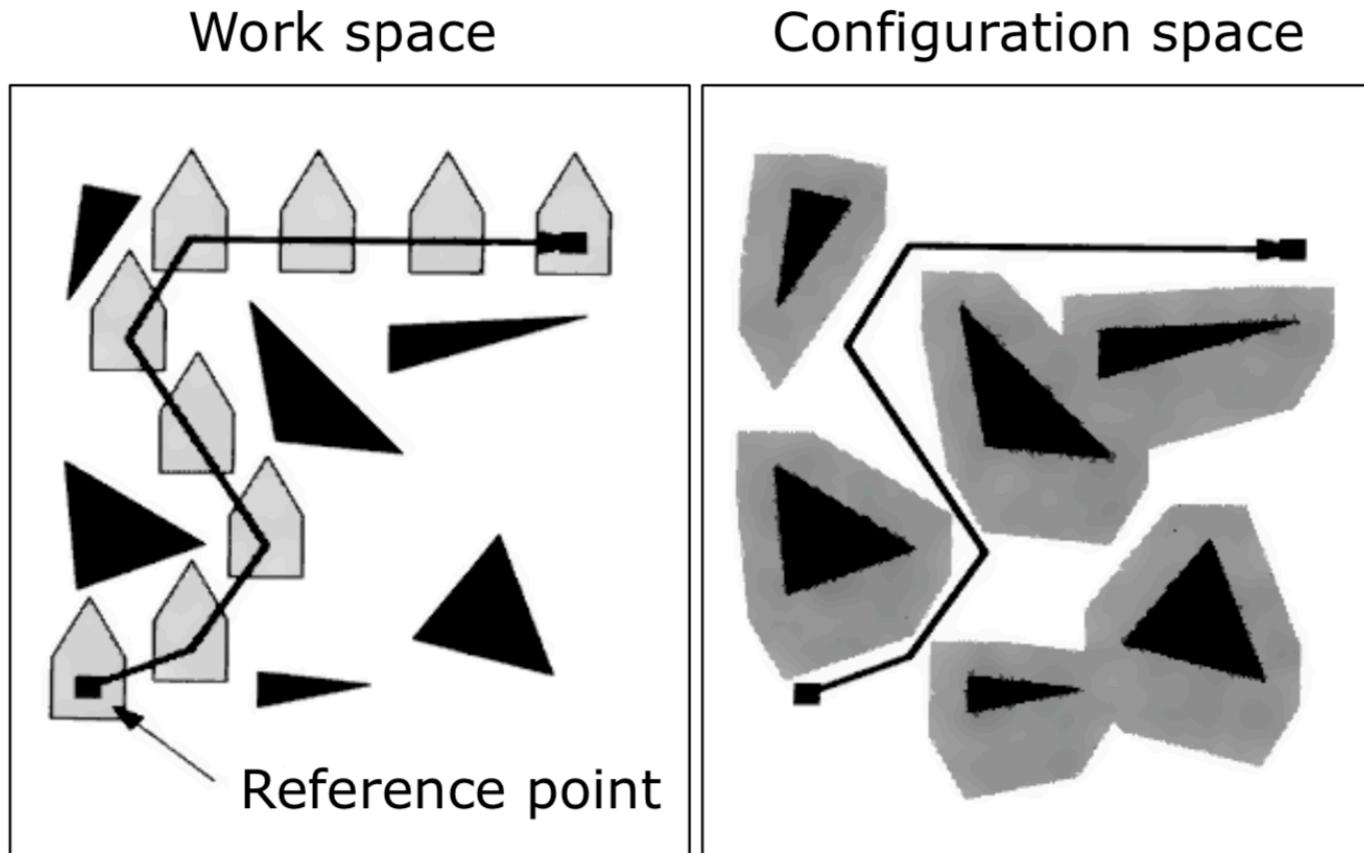
$$QO_i = \{q \in Q \mid R(q) \cap WO_i \neq \emptyset\}.$$



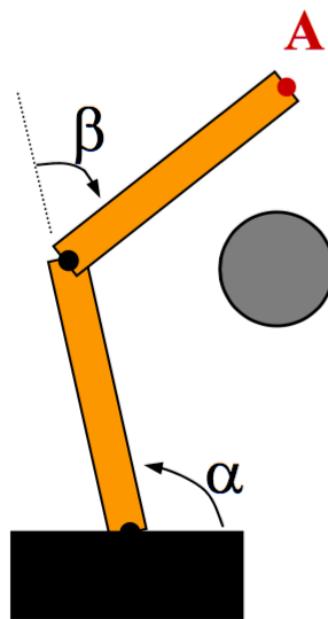
Espace de configuration 3D

Planification dans l'espace de configuration

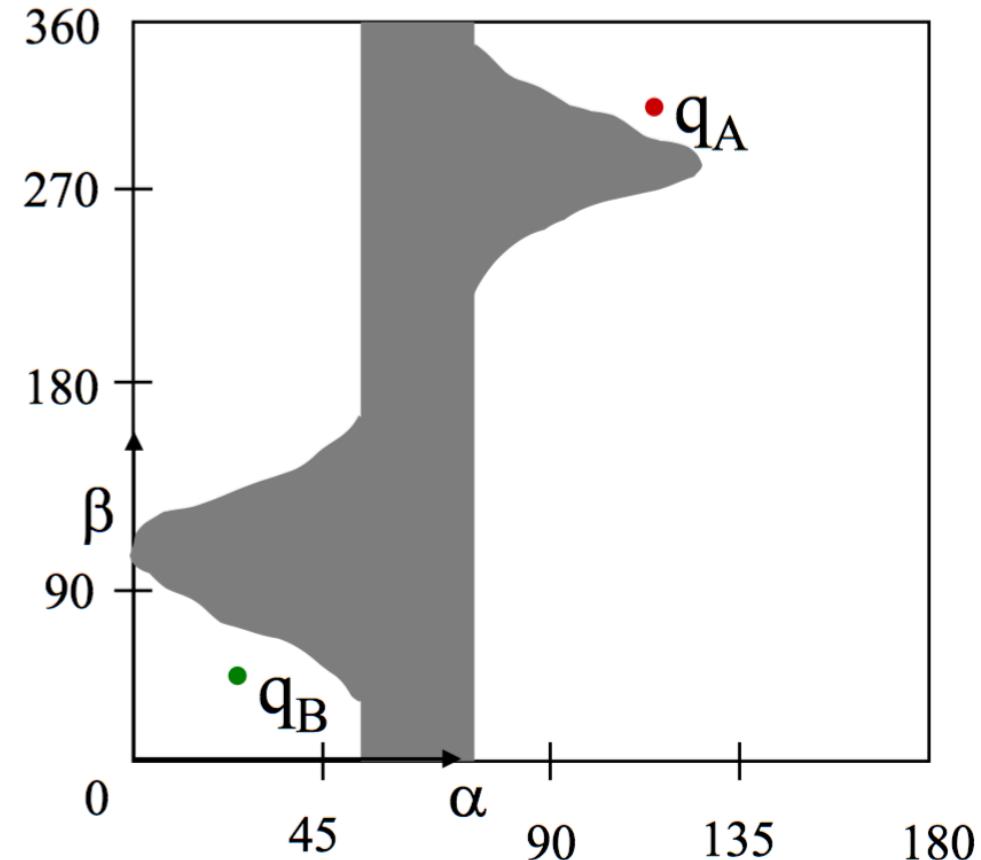
- Ramène à la trajectoire d'un point



Applicable à tous les mécanismes



B



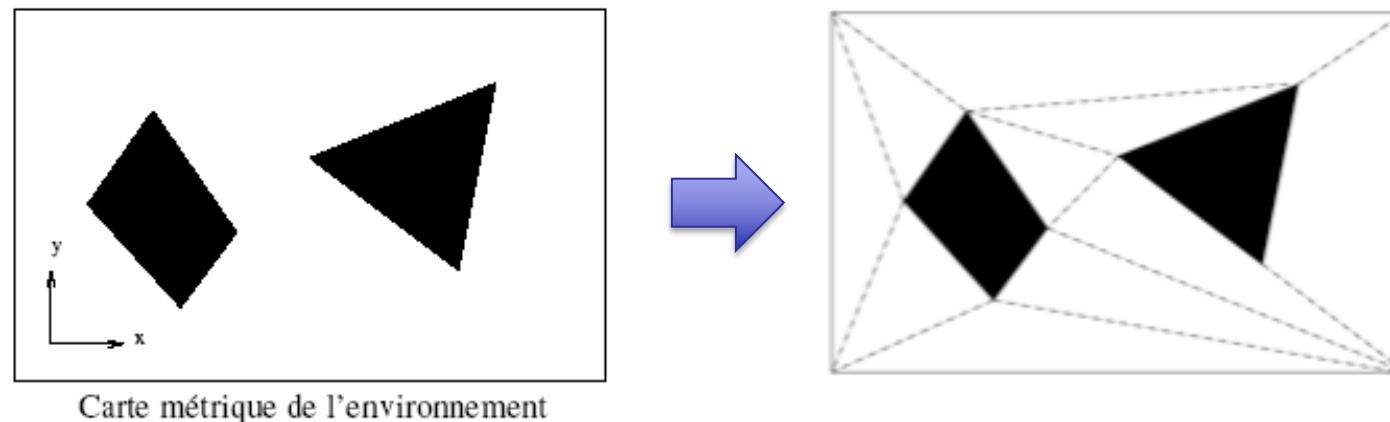
Chemin entre A et B ?

Discrétisation de l'espace de configuration

- Découpage de l'espace libre en zones contiguës
- Création d'un graphe des zones voisines

Décomposition exacte

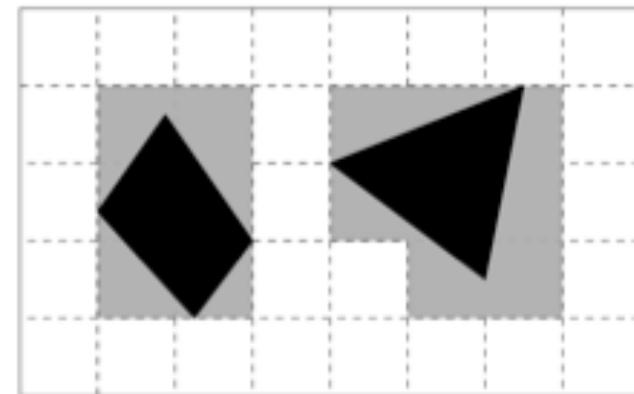
- Possible avec une carte polygonale



- Pas très adapté aux cartes produites par SLAM
- Utile dans des environnements simples/statiques

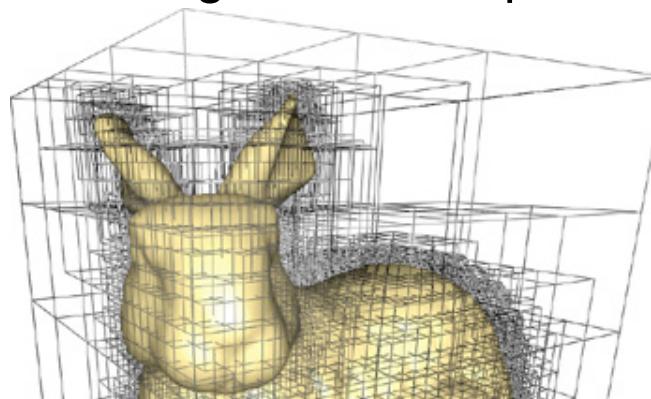
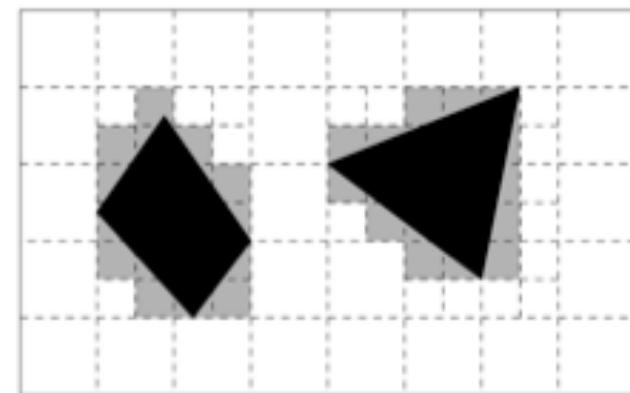
Décomposition en cellules régulières

- Directement disponible avec les grilles d'occupation
- Grande consommation mémoire pour les grands environnements



Décomposition en quad-trees / octrees

- Réduction de l'espace mémoire
- Réduction de l'espace de recherche
- Simplification de grilles d'occupations

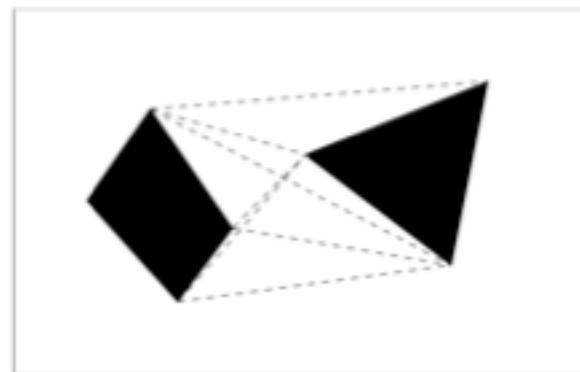


Discrétisation de l'espace de configuration

- Découpage de l'espace libre en chemins libres
- Création d'un graphe des chemins connectés

Graphe de visibilité

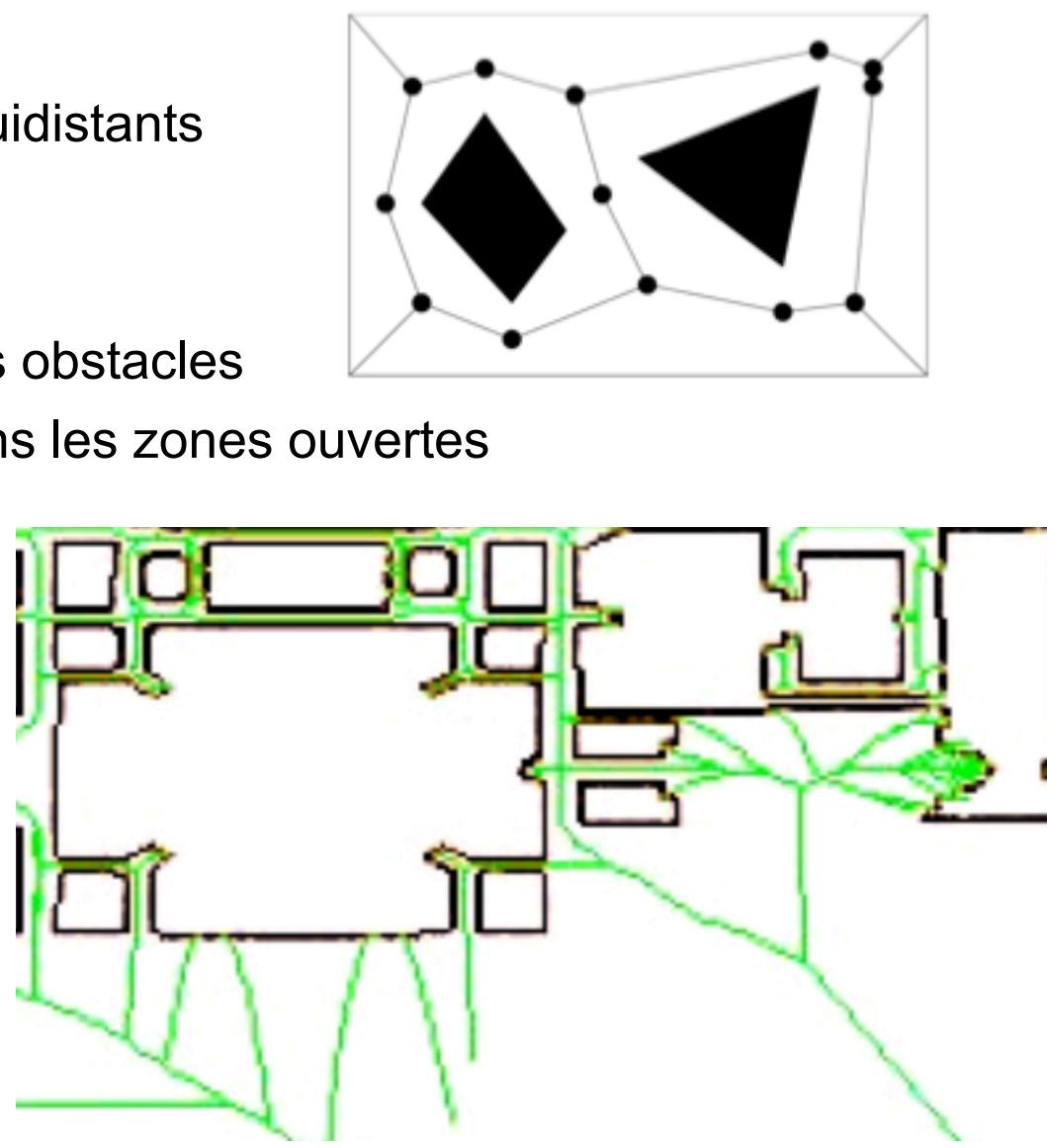
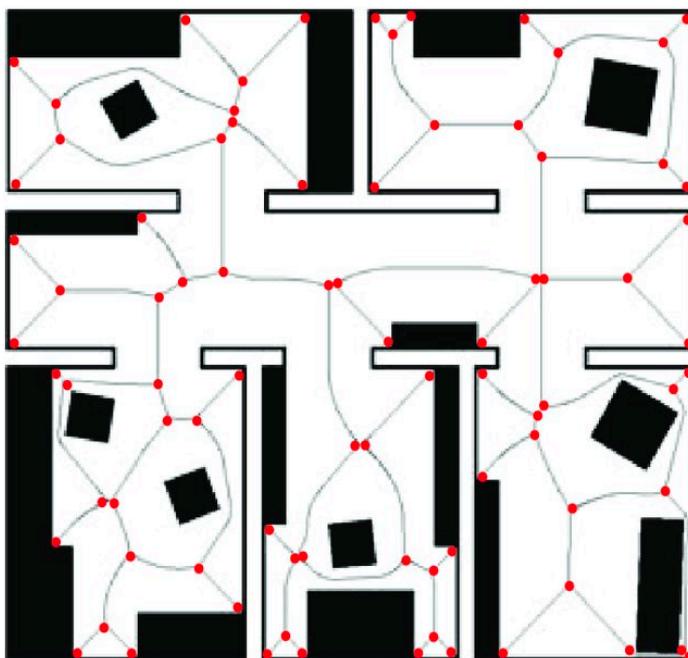
- Relie les coins des obstacles visibles entre eux



- Historiquement l'une des premières approches
- Adapté aux cartes polygonales
- Chemins optimaux, mais passent près des obstacles

Diagramme de Voronoi

- Ensemble des points équidistants de + de 2 obstacles
- Algos de calcul rapides
- Permet de rester loin des obstacles
- Pb de comportement dans les zones ouvertes



Discrétisation à l'aide de la route

- On peut discréteriser l'espace libre sous forme de treillis
- En général, on suit l'axe de la route
- Sinon, on se décale pour éviter un obstacle
- Création d'un treillis aligné sur la route selon cette logique

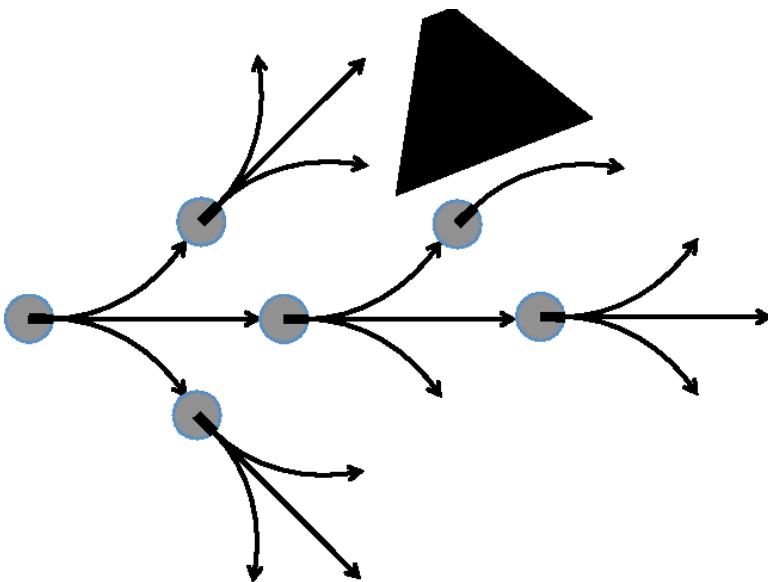
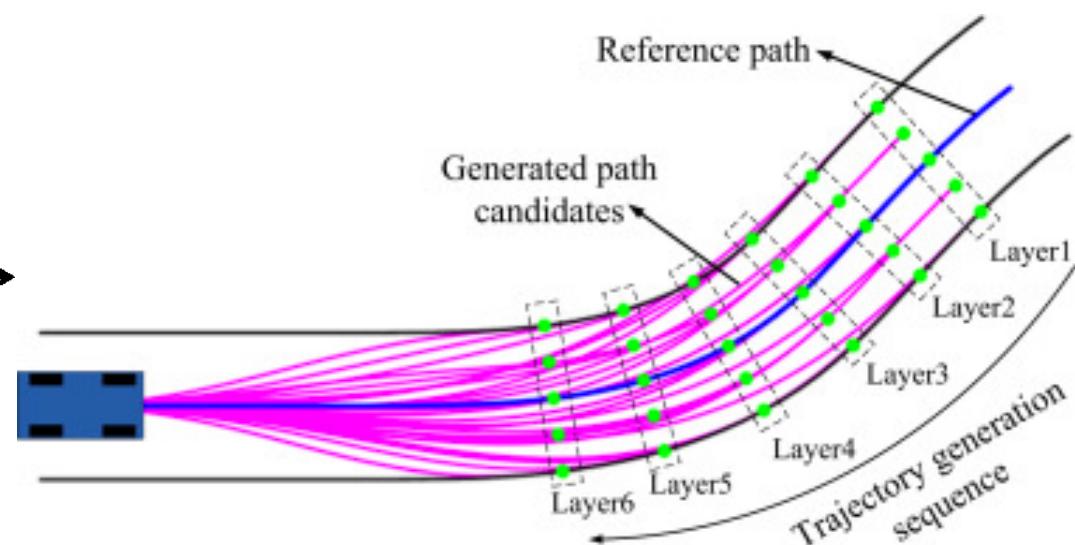
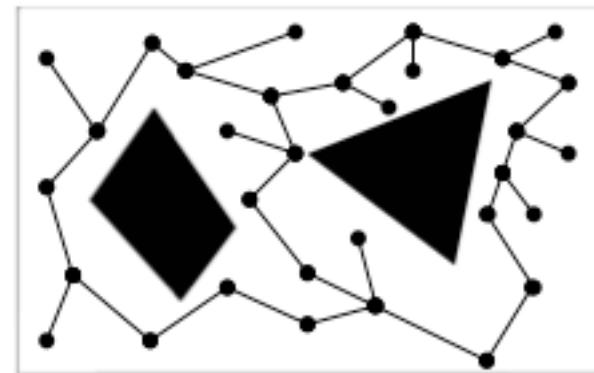


Fig. 10. An example of a lattice type graph



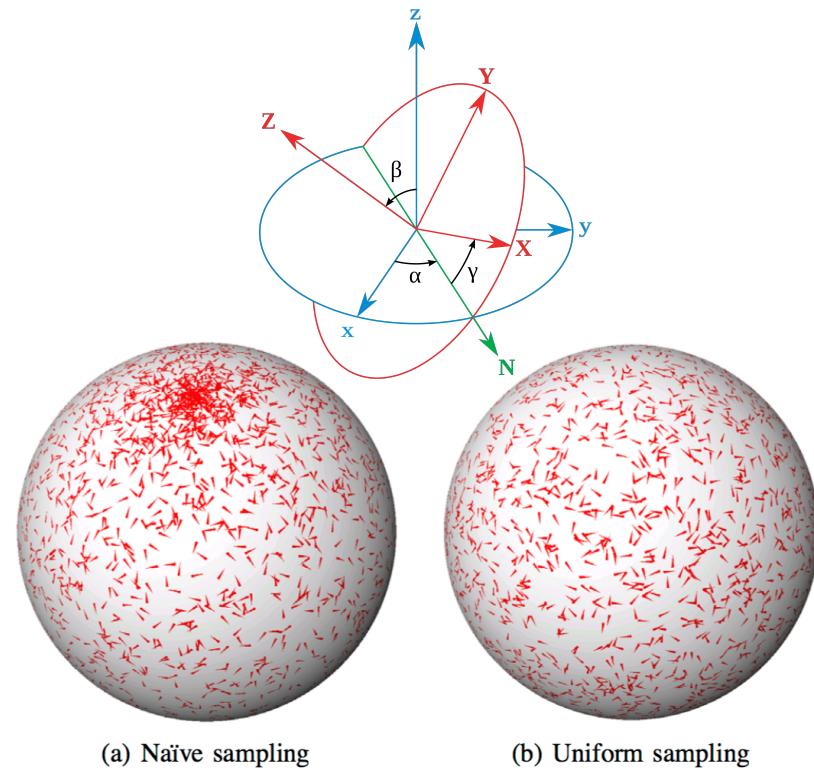
Echantillonnage aléatoire

- Permet de représenter les grands espaces de manière minimale
- Nécessite simplement une procédure de détection de collision
- Différentes méthodes dont
 - Probabilistic Road Maps (PRM)



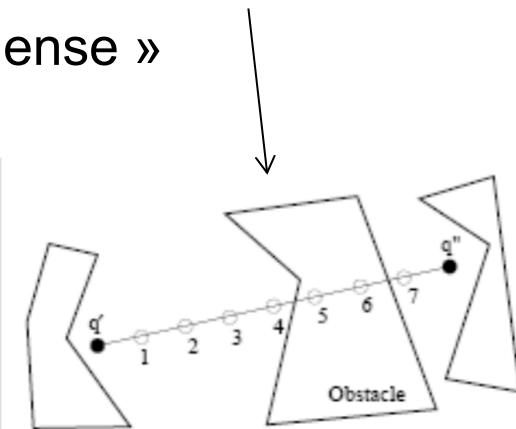
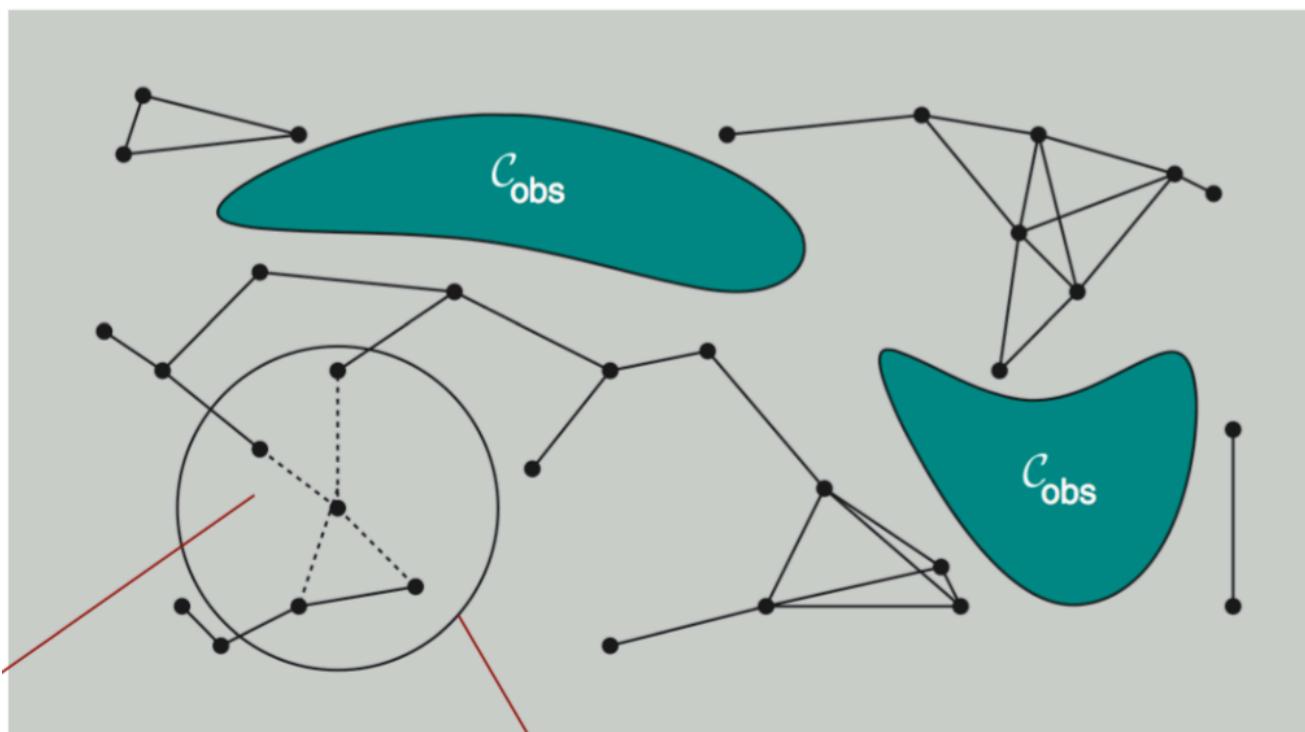
Echantillonnage des angles

- Echantillonnage uniforme en 3D
- Attention méthode naïve avec Euler
 -> quaternions



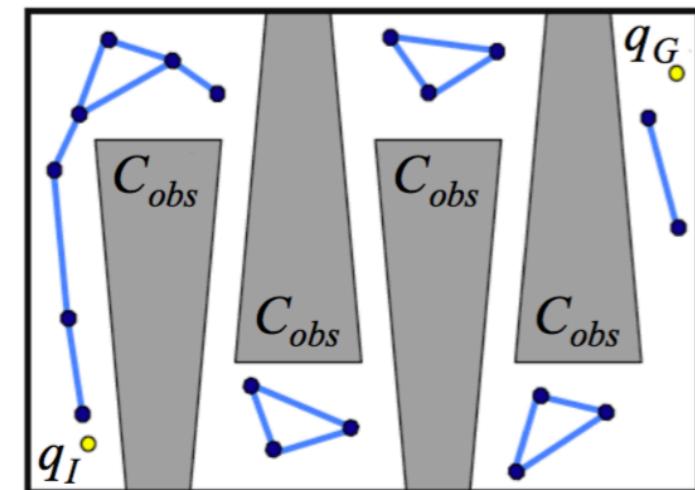
Probabilistic Road Maps (PRM, 92)

- Choisir des points aléatoirement dans l'espace libre
- Relier les points proches par un planificateur local (simple et rapide)
- Continuer jusqu'à ce que le graphe soit « assez dense »



Probabilistic Road Maps (PRM, 92)

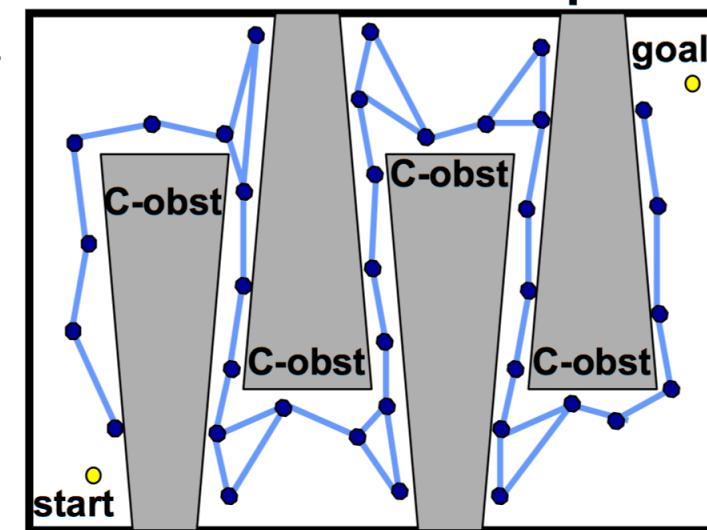
- Complet d'un pt de vue probabiliste
- Mais pas complet/optimal
- Applicable en grandes dimensions
- Pas d'espace de config. explicite
- Problèmes dans les passages étroits



NOMBREUSES EXTENSIONS

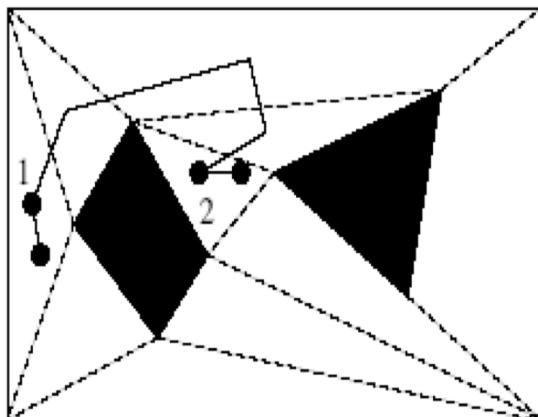
- Echantillonnage dans les zones réduites
- Echantillonnage près des obstacles
- Gestion des objets déformables
- ...

OBPRM Roadmap

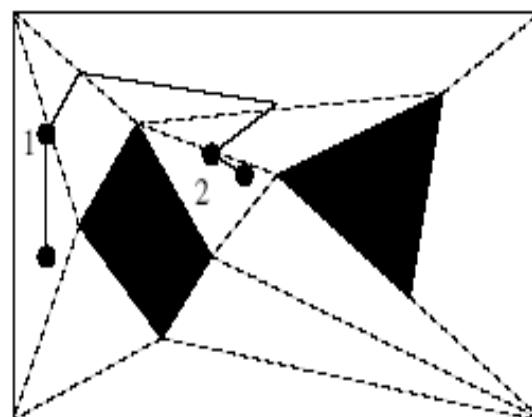


Calcul de chemin

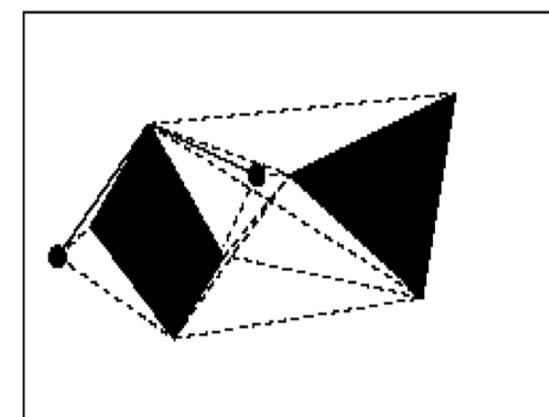
- Calculer chemin départ ->graphe / graphe ->but
- Calculer le chemin le plus court dans le graphe



Planification entre les centres des cellules

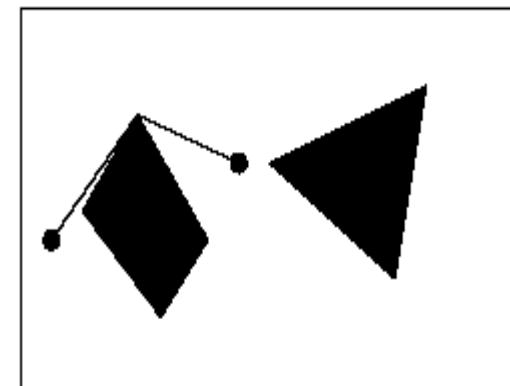


Planification entre les cotés des cellules



Planification en utilisant des chemins

- Éventuellement : optimisation du chemin par relaxation



Chemin optimisé

Recherche de chemin dans un graphe



Recherche de chemin dans un graphe

- Nombreux algorithmes pour estimer le « cout » du chemin le plus court entre chaque nœud et le but
 - (cf $V(s)$ et $Q(s,a)$ en Apprentissage par Renforcement)
- Algorithmes avec / sans information

Information utilisable

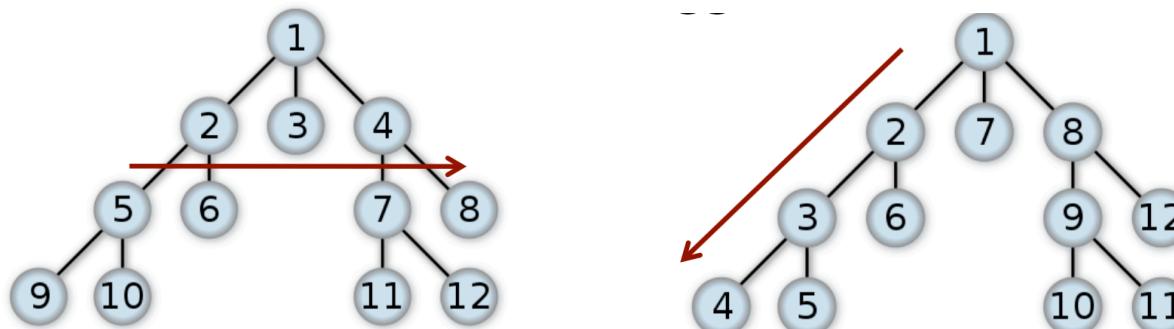
- Heuristique donnant la distance au but
- Par exemple : distance en ligne droite
- Doit être « admissible » : < longueur chemin le plus court

Coût

- Associé aux nœuds (zones dangereuses, zones à éviter...)
- Associé aux liens (distance, difficulté de traversée...)

Cas graphe sans poids

- distance = nombre de liens
- Algo non informés : recherche en largeur d'abord/profondeur d'abord

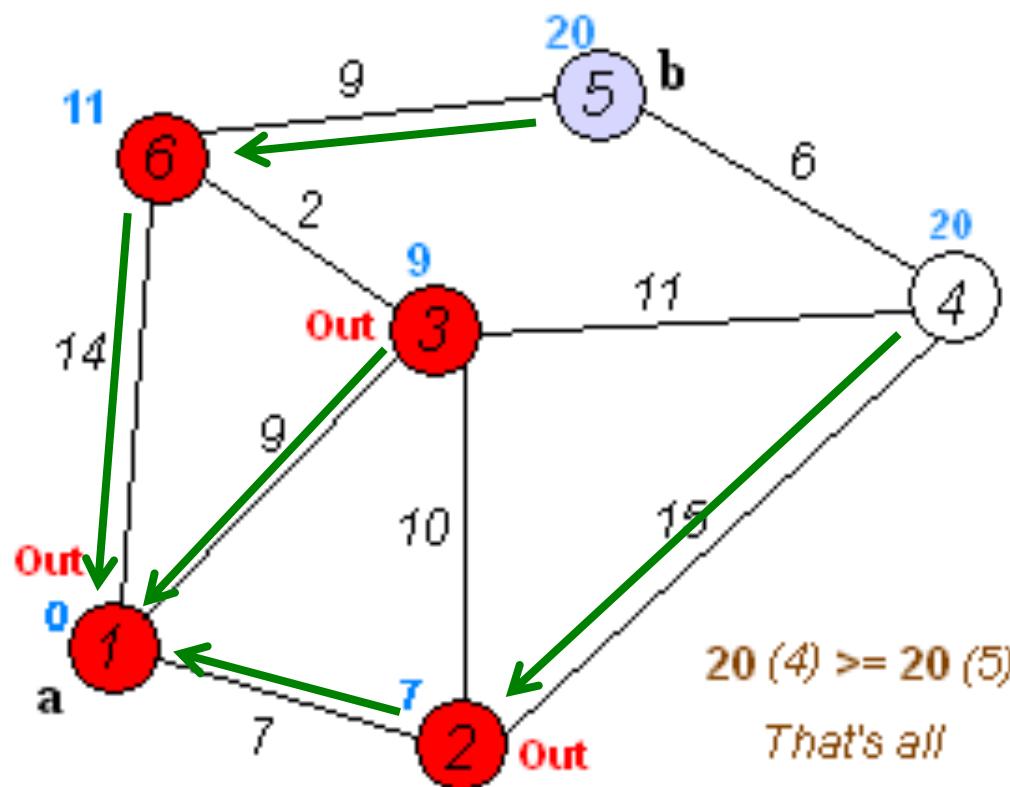


Cas de poids tous positifs : algorithme de dijkstra

- $S :=$ ensemble vide; $Q :=$ ensemble de tous les nœuds; $d = 0$ pour le but; $d = \infty$ pour les autres noeuds
- tant que Q n'est pas l'ensemble vide faire
 - $u := \text{Extract-Min-}d(Q)$
 - $S := S \cup \{u\}$
 - pour chaque nœud v voisin de u faire
 - » si $d[v] > d[u] + w(u,v)$ alors $d[v] := d[u] + w(u,v)$; $\text{previous}[v] := u$

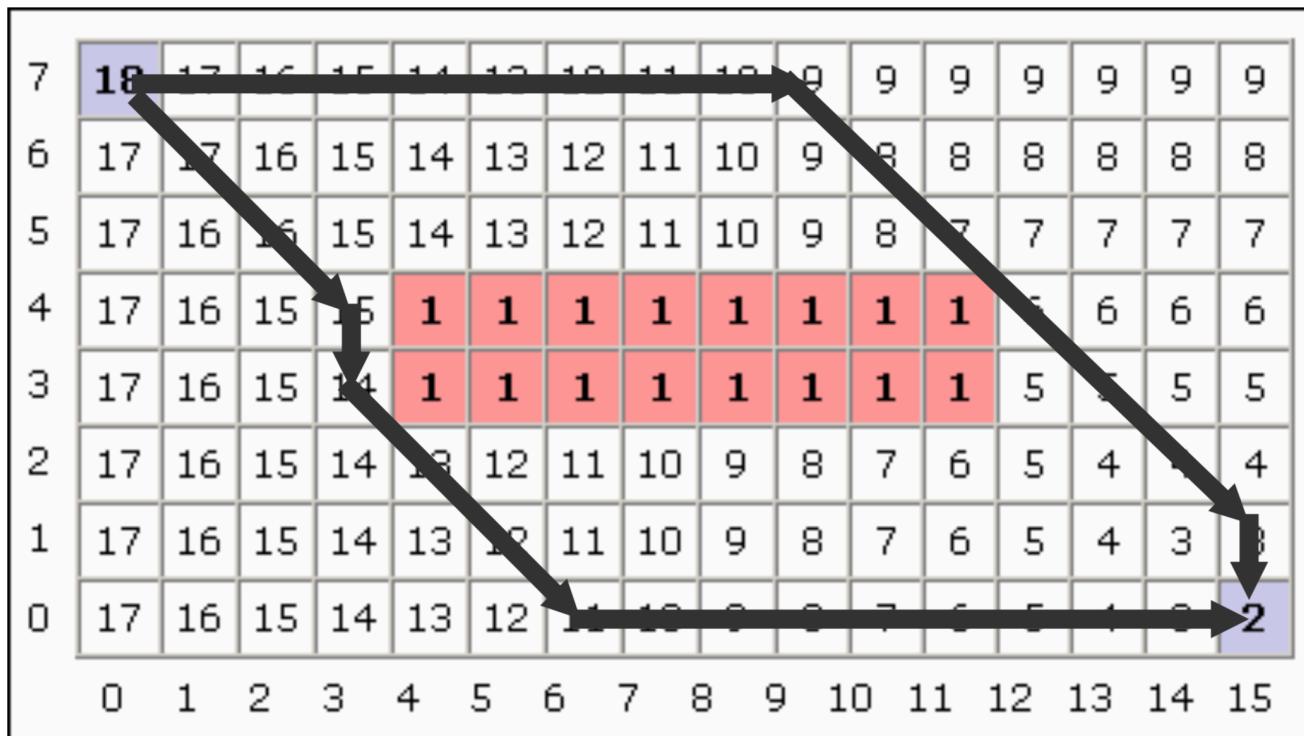
Choix des actions:

- descente de gradient



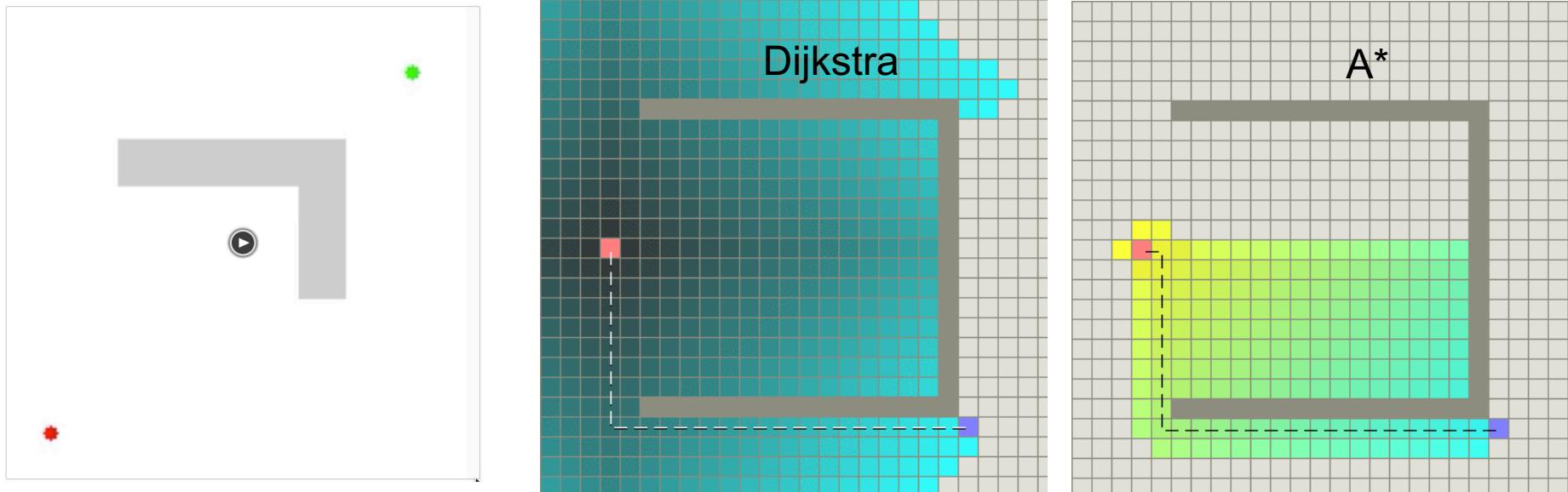
Wavefront planner

- Variante de Dijkstra lorsque les poids des liens sont similaires (e.g. tous == 1)
- Traitement de tous les nœuds « ouverts » en même temps
- Possibilité de paralléliser



Pour calculer un seul chemin : algorithme A*

- Similaire à l'algorithme de Dijkstra
- Algo informé : ajout d'une heuristique estimant la distance d'un point au but (ex : distance euclidienne)
- Traitement en priorité des points ayant la plus faible valeur
 - distance (départ) + μ^* heuristique
 - μ règle l'influence de l'heuristique
- Permet de ne parcourir qu'une partie des états



Replanification en environnement dynamique: D* Lite

- Inspiré de A*, pour permettre de replanifier rapidement en cas de changement de l'environnement
- Maintient 2 valeurs :
 - g =distance au but
 - rhs = distance après 1 déplacement + cout du déplacement
- Si $g \neq rhs \rightarrow$ replanifier
- Mettre les nœuds inconsistants « ouvert » et mettre à jour leurs voisins

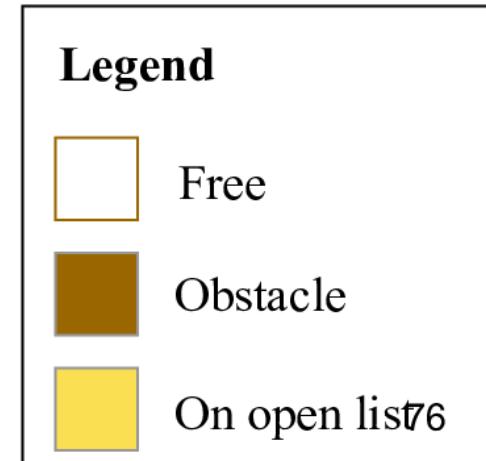
Replanification en environnement dynamique: D* Lite

g: ∞ rhs: ∞				
g: ∞ rhs: ∞				
g: ∞ rhs: ∞				
g: ∞ rhs: ∞				
g: ∞ rhs: 0	g: ∞ rhs: ∞	g: ∞ rhs: ∞	g: ∞ rhs: ∞	g: ∞ rhs: ∞

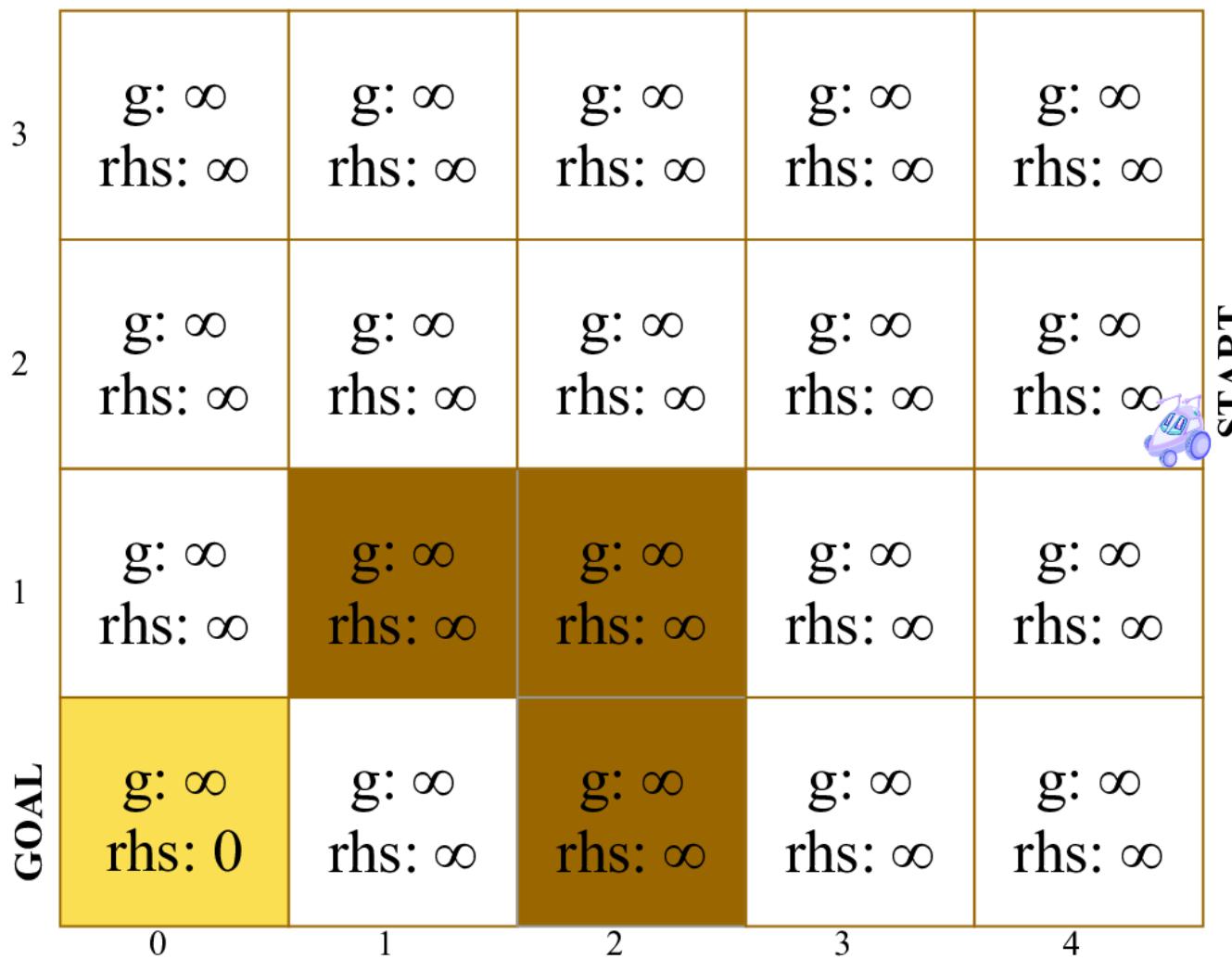
Initialization

Set the *rhs* value of the goal to 0 and all other *rhs* and *g* values to ∞ .

START



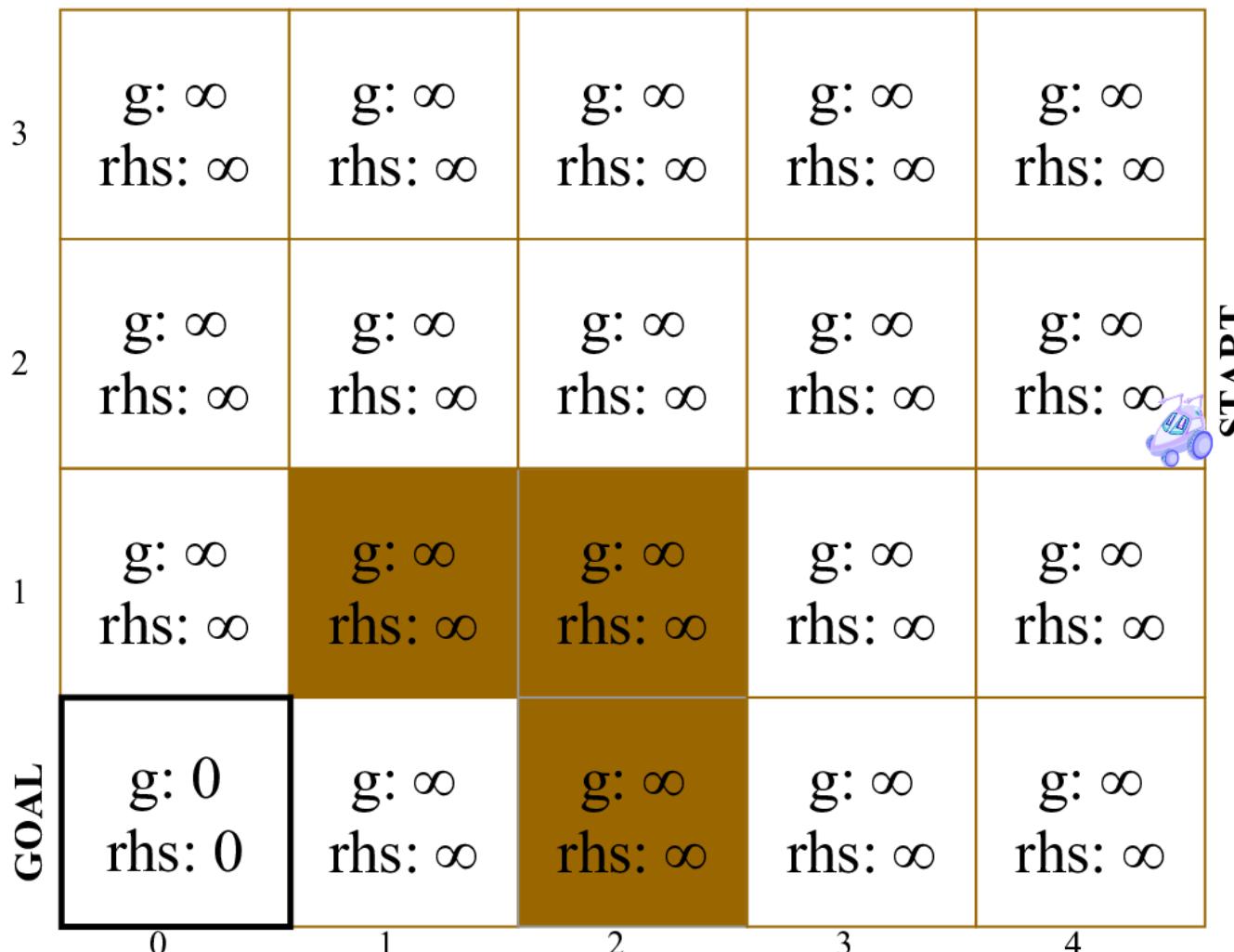
Replanification en environnement dynamique: D* Lite



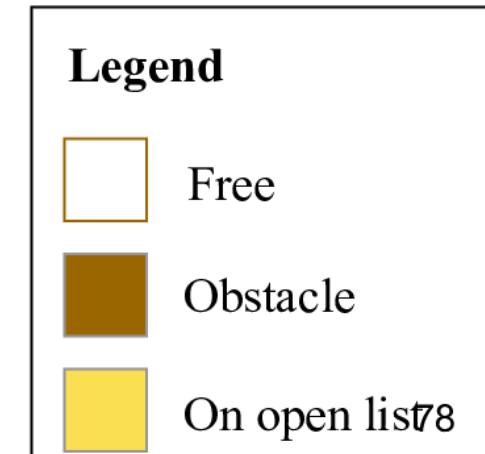
Initialization

Put the goal on the open list because it is inconsistent.

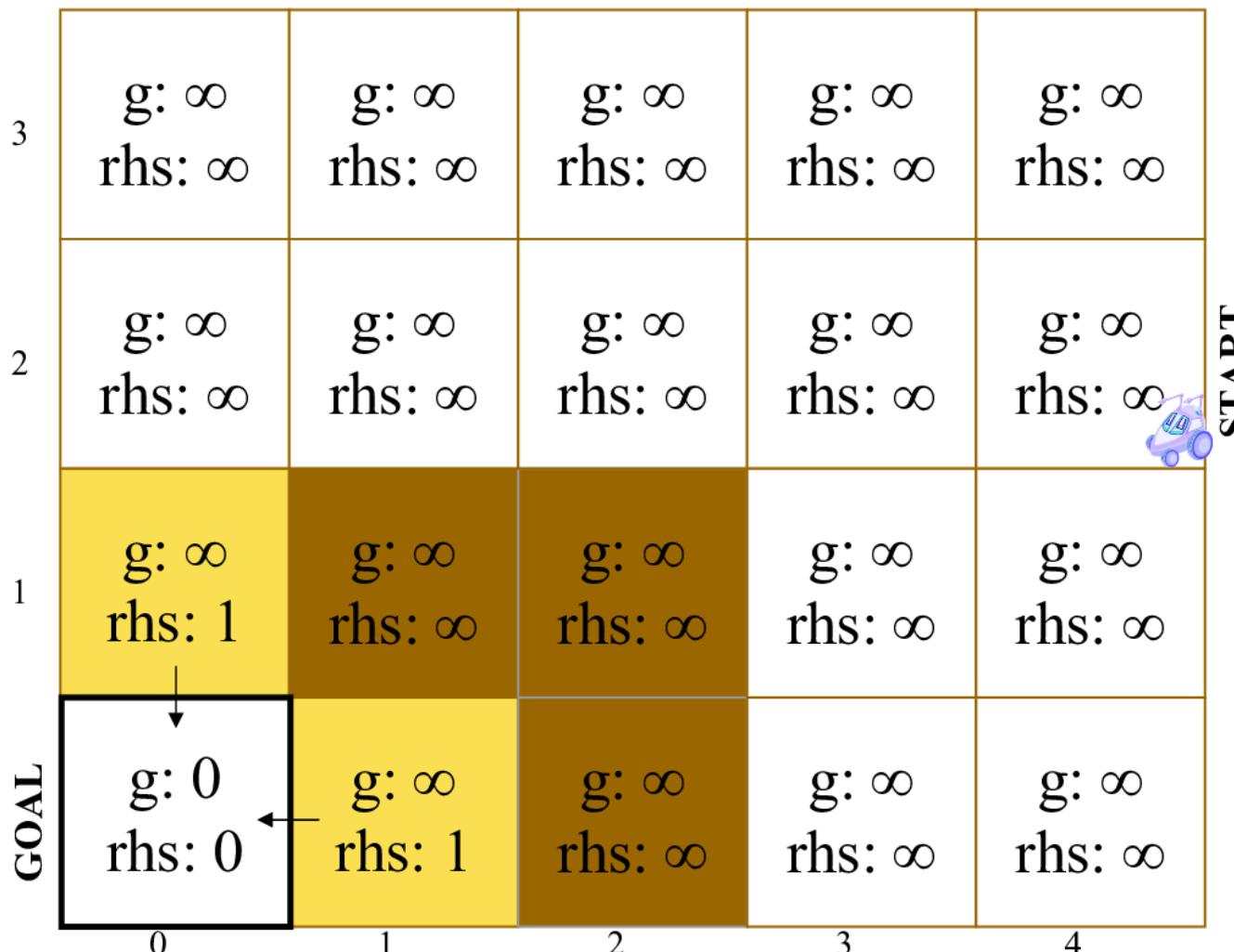
Replanification en environnement dynamique: D* Lite



ComputeShortestPath
 Pop minimum item off open list (goal)
 It's over-consistent ($g > \text{rhs}$) so set $g = \text{rhs}$.



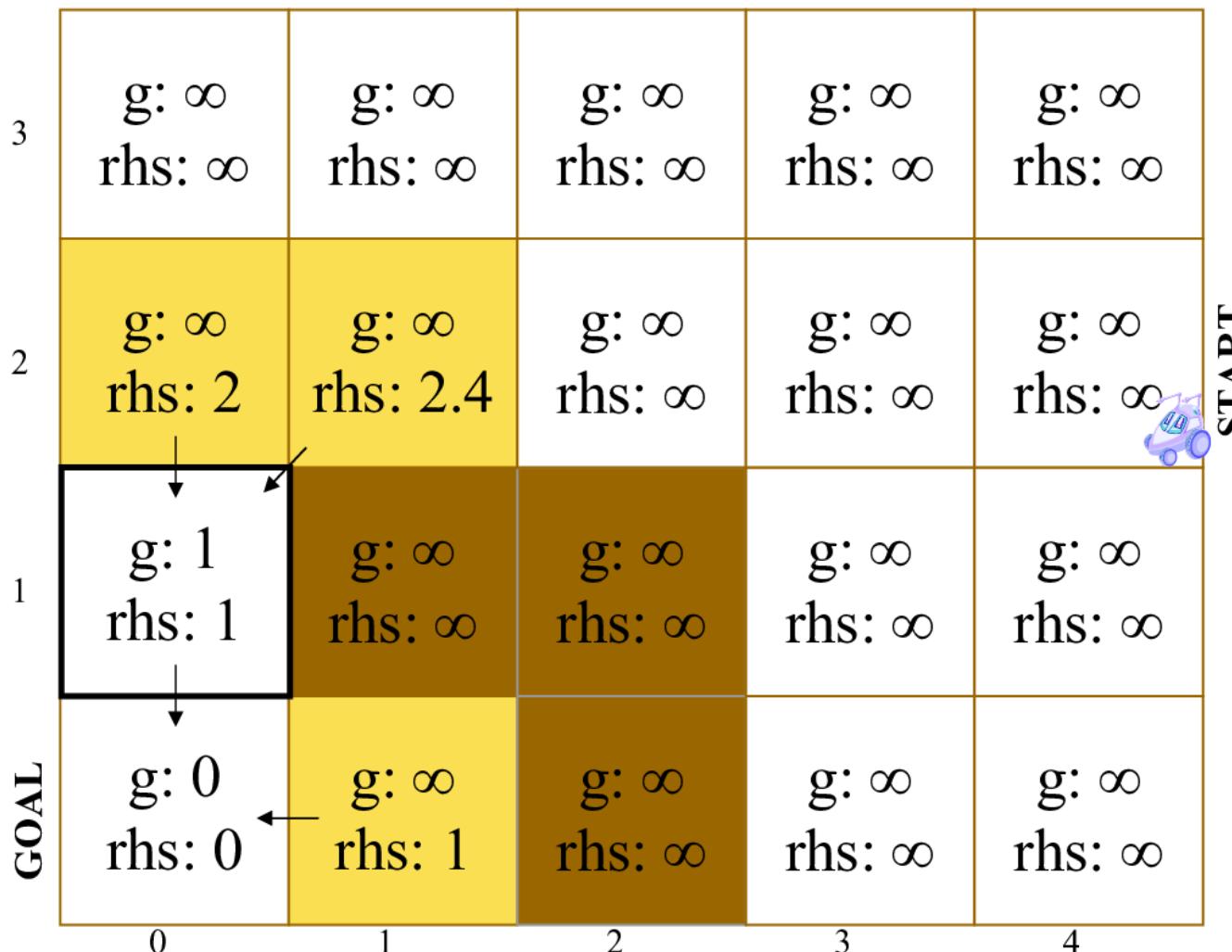
Replanification en environnement dynamique: D* Lite



ComputeShortestPath
 Expand the popped node i.e, call *UpdateVertex()* on all its predecessors in the graph. This computes *rhs* values for the predecessors and puts them on the open list if they become inconsistent.

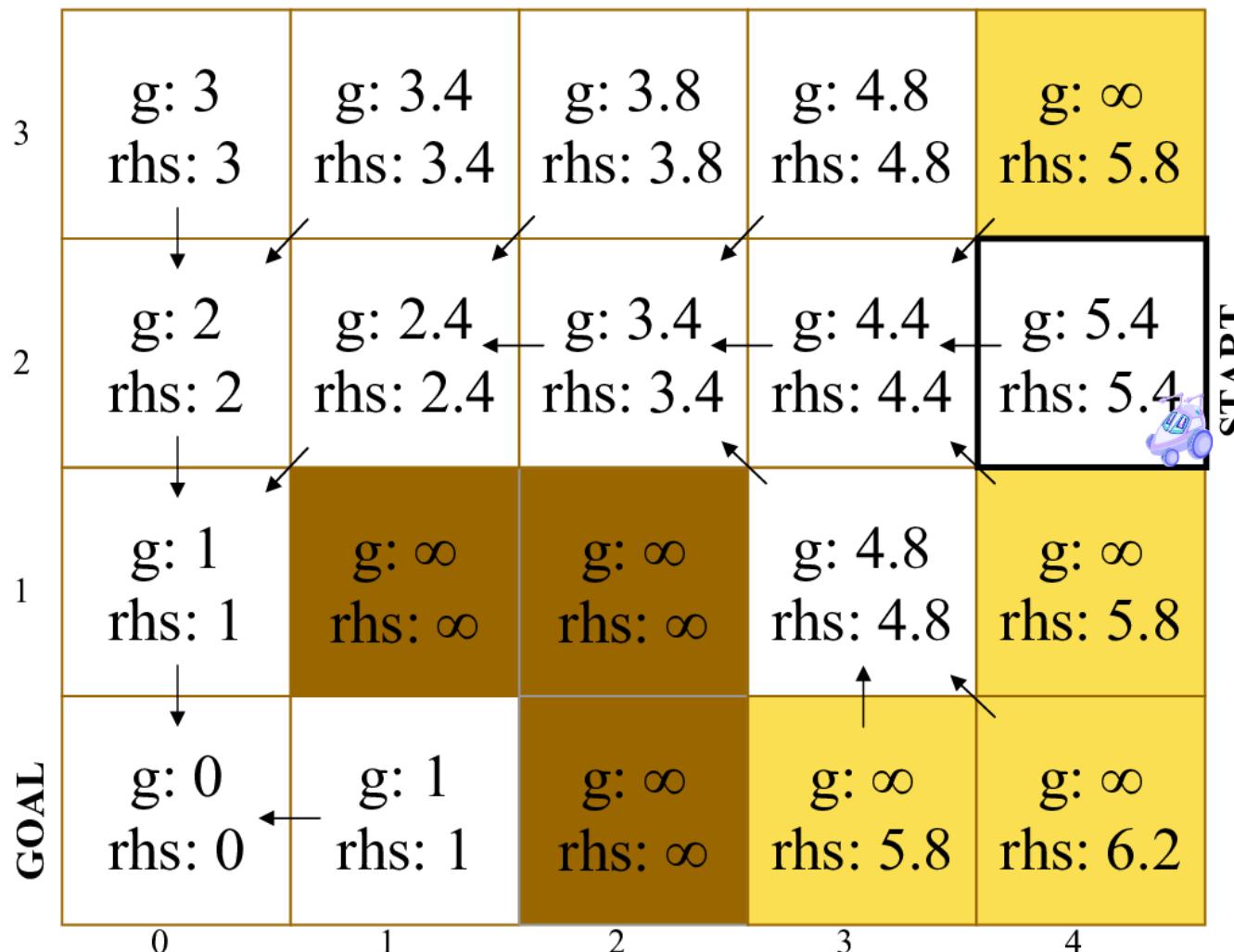
Legend	
	Free
	Obstacle
	On open list

Replanification en environnement dynamique: D* Lite

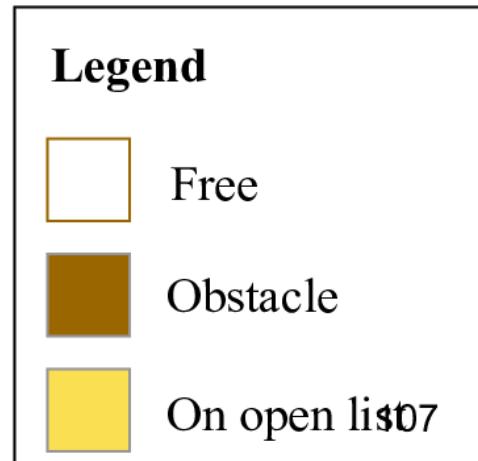


ComputeShortestPath
 Expand the popped node – call *UpdateVertex()* on all predecessors in the graph. This computes *rhs* values for the predecessors and puts them on the open list if they become inconsistent.

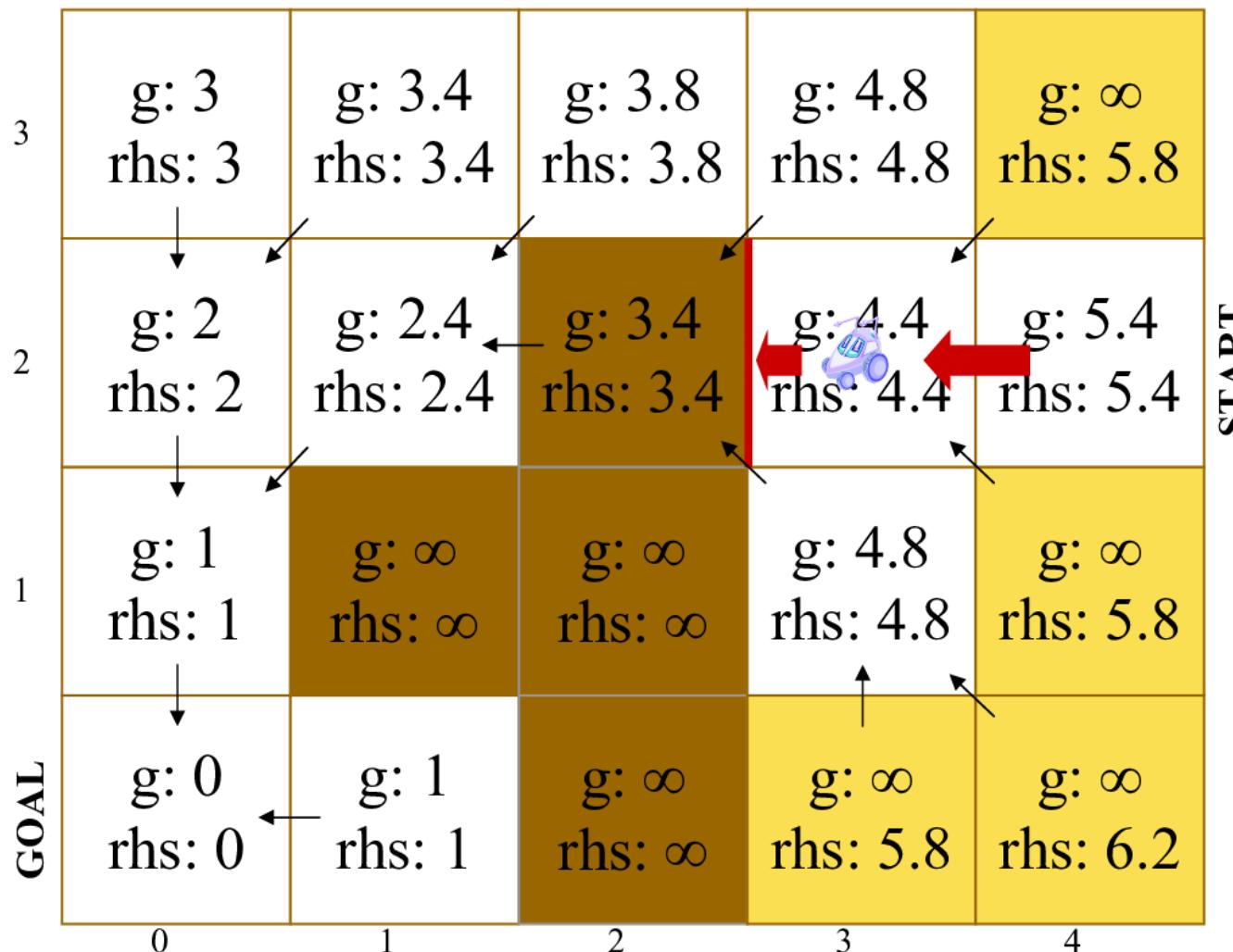
Replanification en environnement dynamique: D* Lite



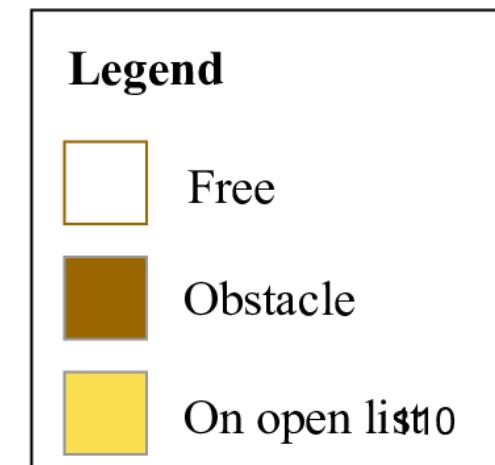
Note that there are still nodes on the open list – we leave them there. Also, for a larger map, there might be some nodes that are not touched at all but we still break out of the loop because we have an optimal path to the start.



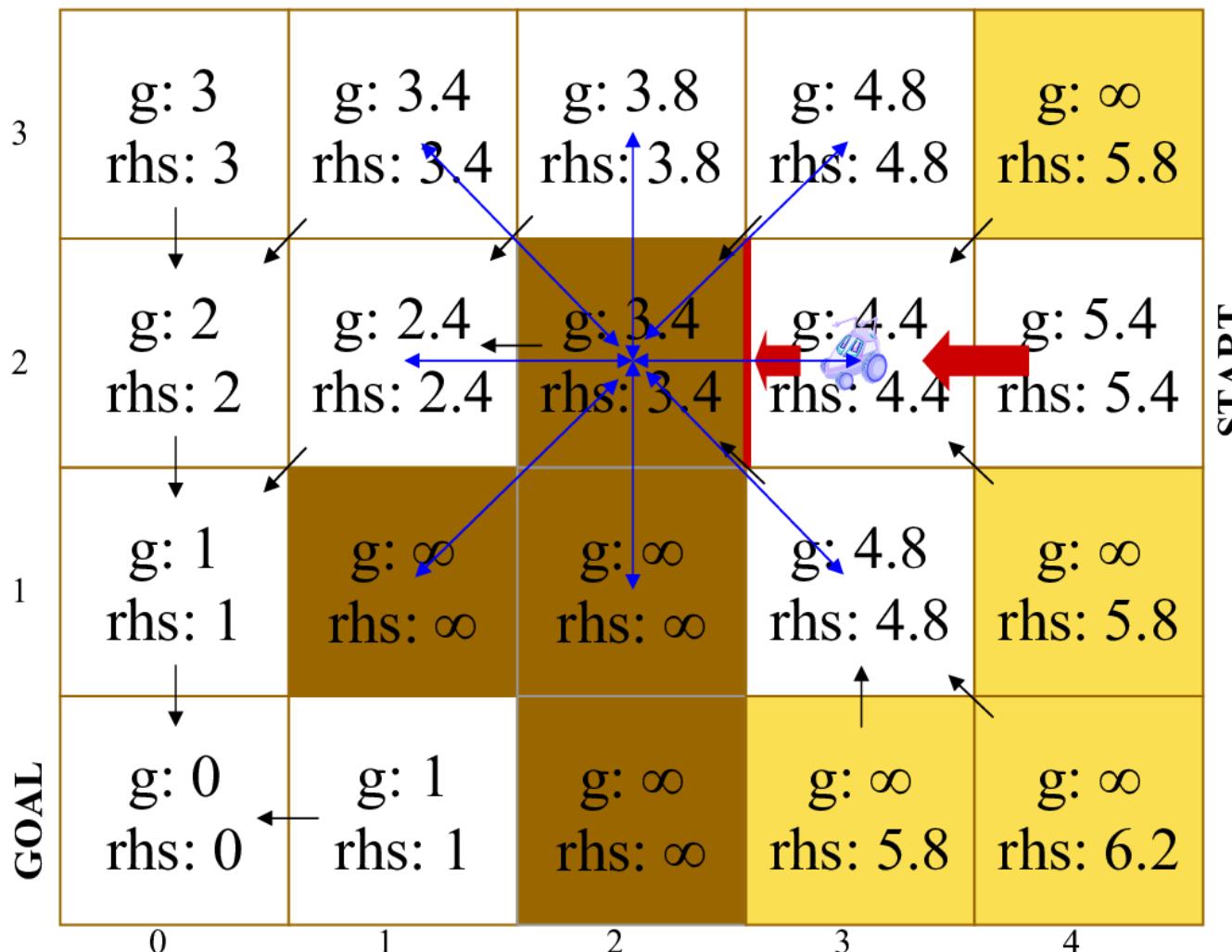
Replanification en environnement dynamique: D* Lite



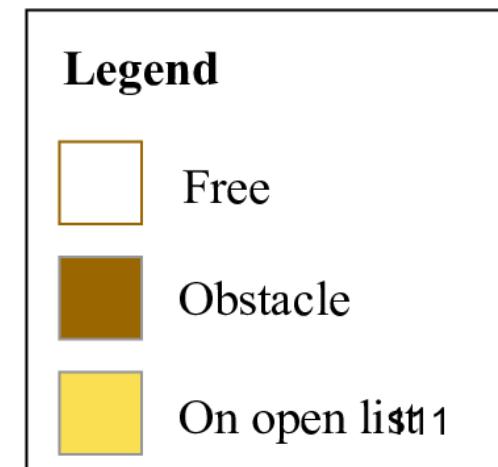
Suppose that, in trying to move from (3,2) to (2,2), the robot discovers that (2,2) is actually an obstacle. Now, **replanning** is required!



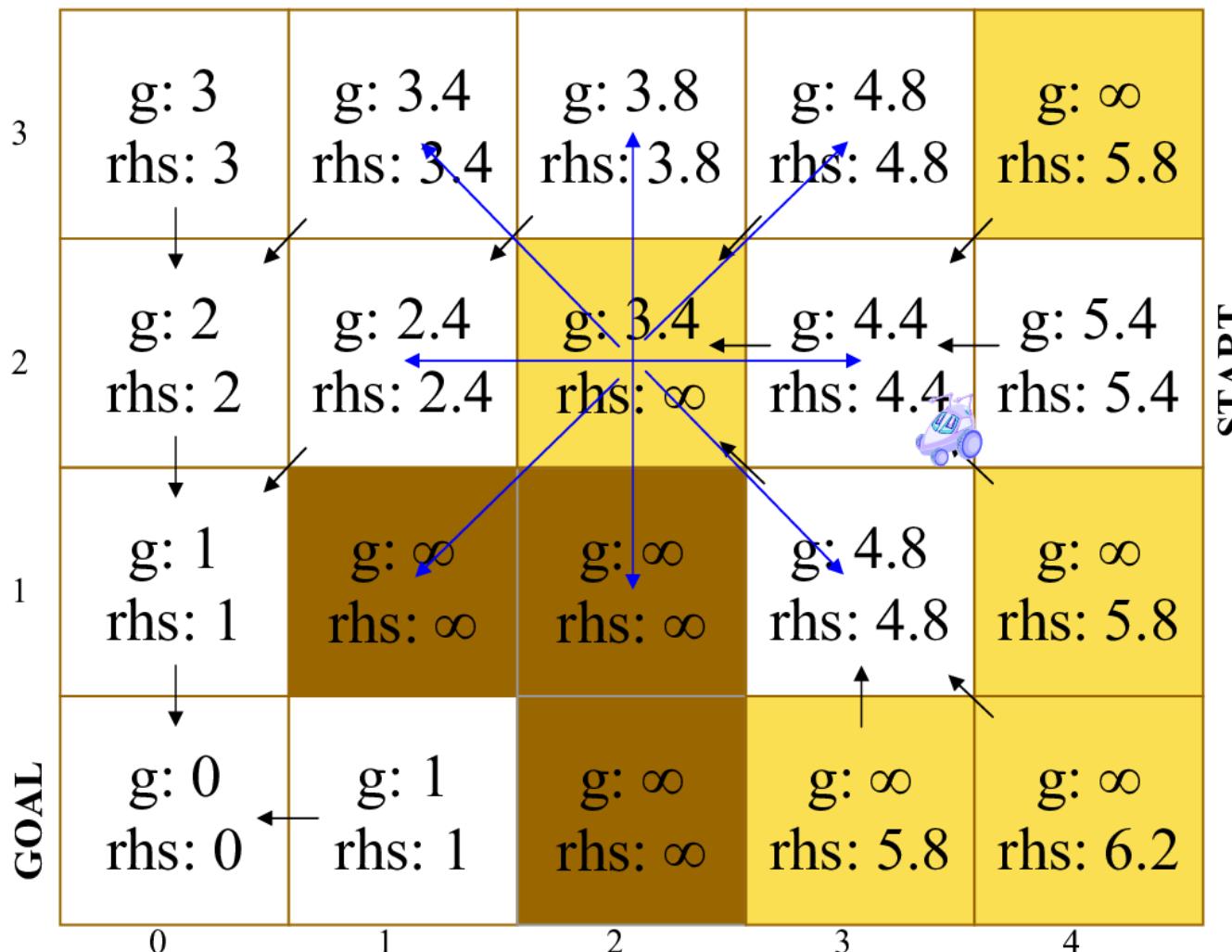
Replanification en environnement dynamique: D* Lite



The algorithm dictates that for all directed edges (u, v) with changed edge costs, we should call $UpdateVertex(u)$. Since the edges in our graph are bidirectional and all edges into or out of (2,2) are affected, we need to consider 16 different edges!



Replanification en environnement dynamique: D* Lite



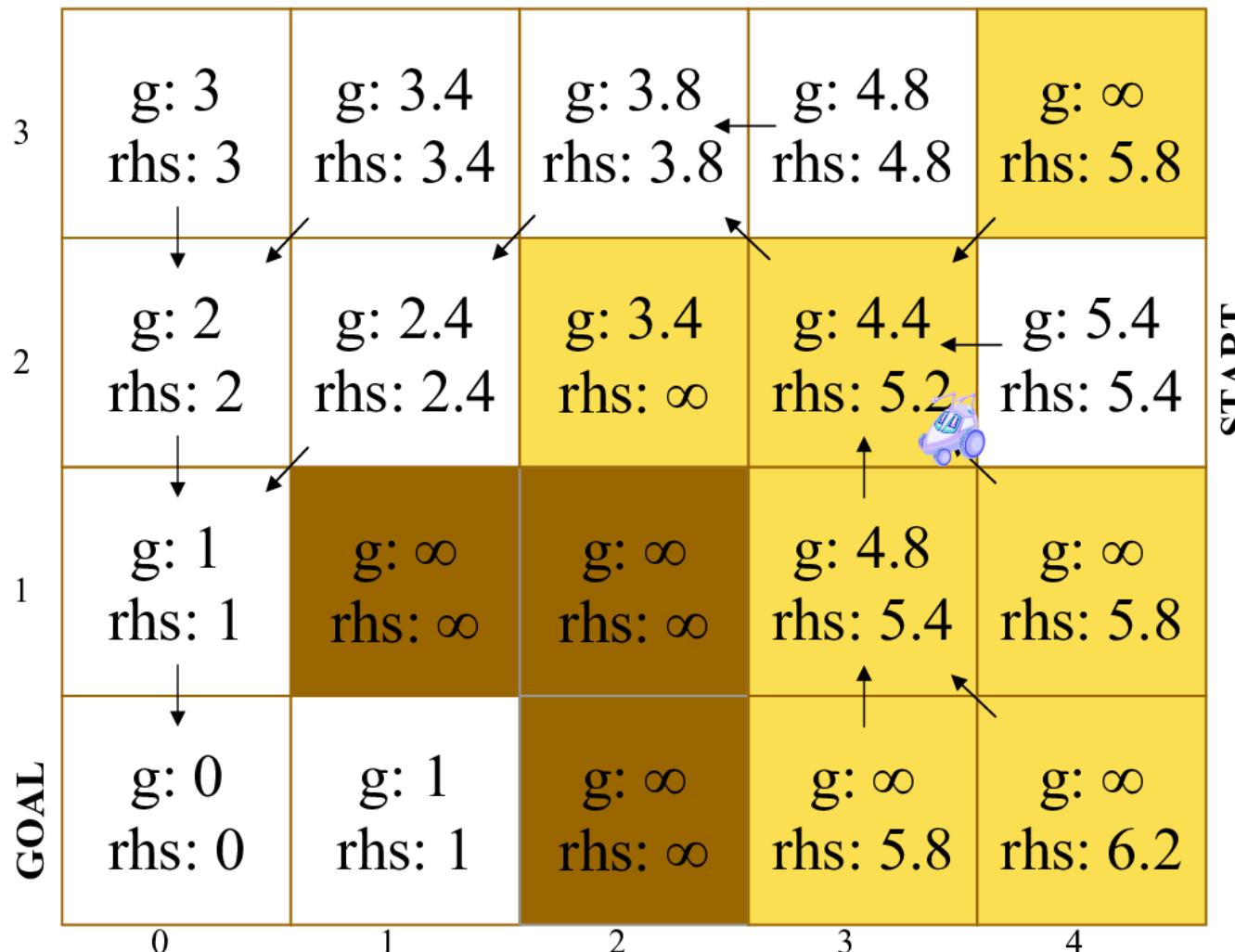
UpdateVertex

First, consider the outgoing edges from (2,2) to its 8 neighbors. For each of these we call *UpdateVertex()* on (2,2). Because the transition costs are now ∞ , the *rhs* value of (2,2) is raised, making it inconsistent. It is put on the open list .

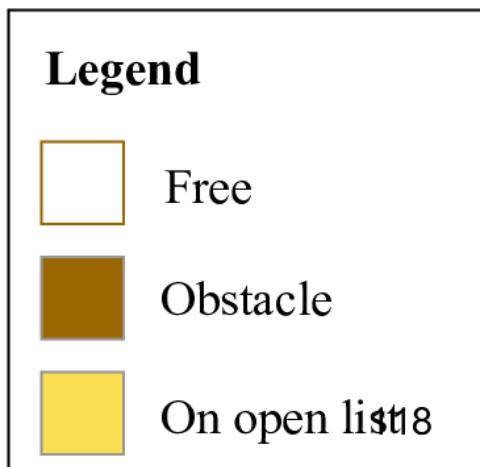
Legend

	Free
	Obstacle
	On open list

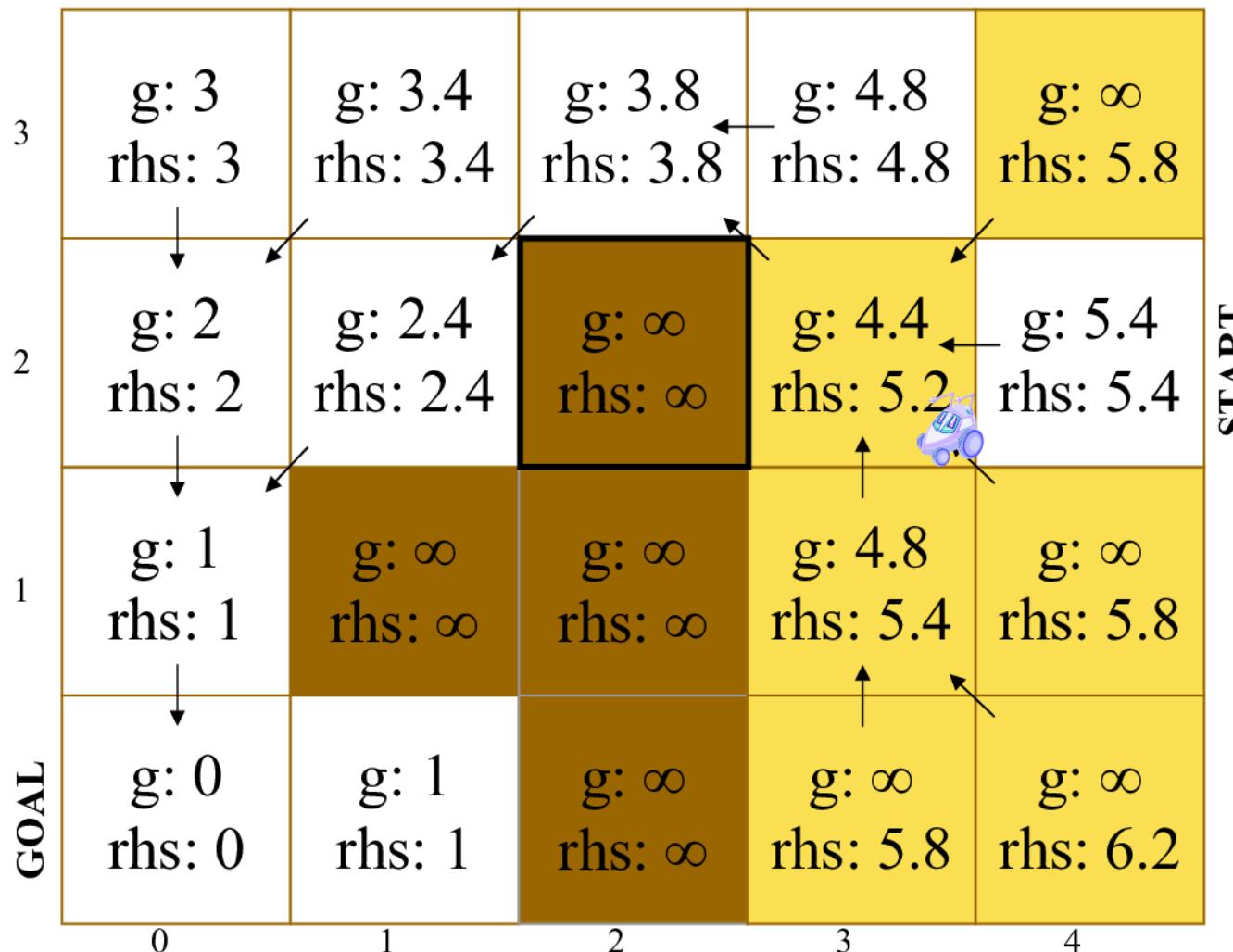
Replanification en environnement dynamique: D* Lite



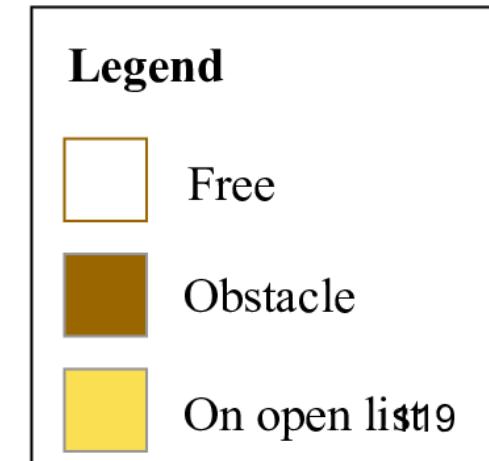
Now, we go back to calling *CompteShortestPath()* until we have an optimal path. The node corresponding to the robot's current position is inconsistent and its key is greater than the minimum key on the open list, so we know that we do not yet have an optimal path..



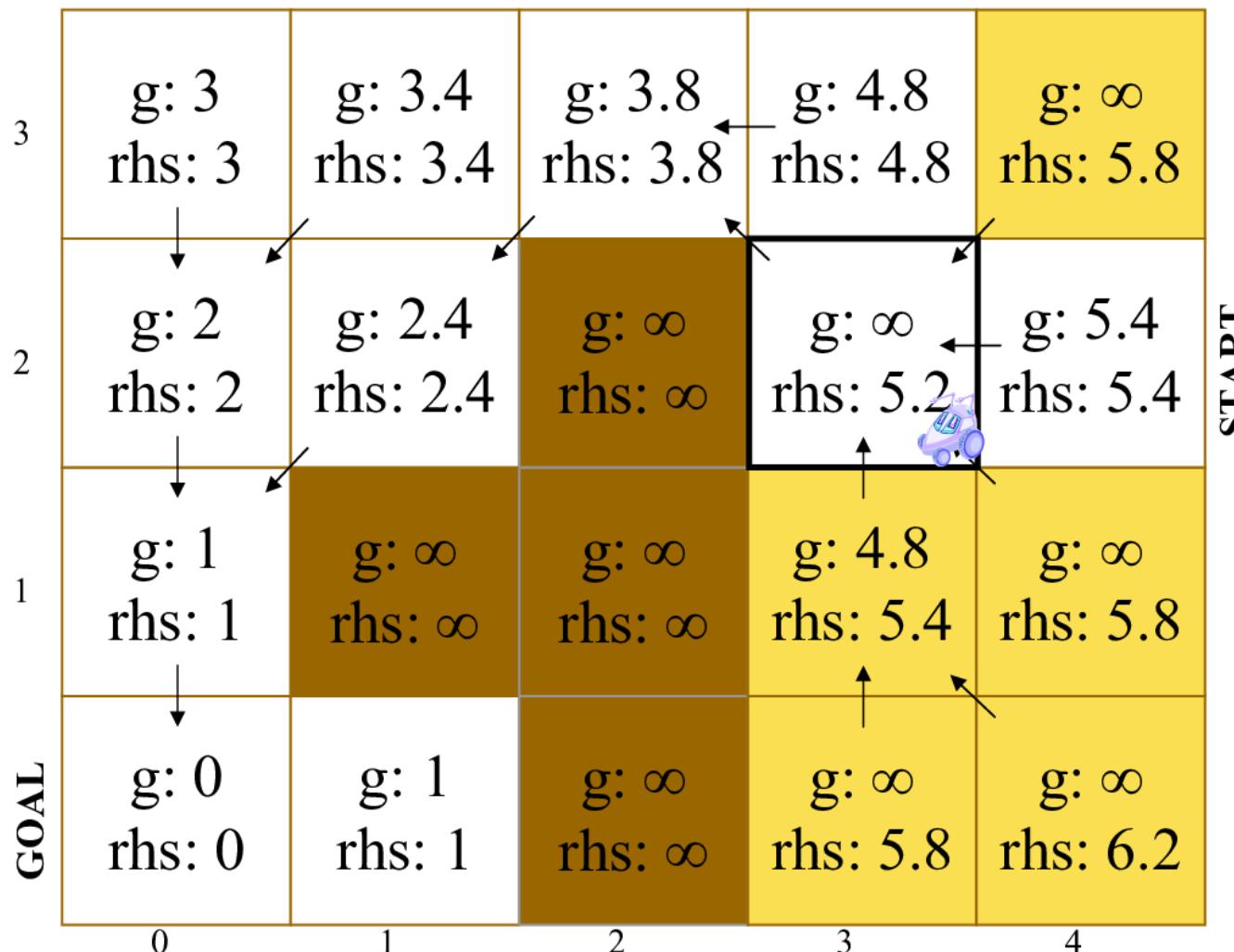
Replanification en environnement dynamique: D* Lite



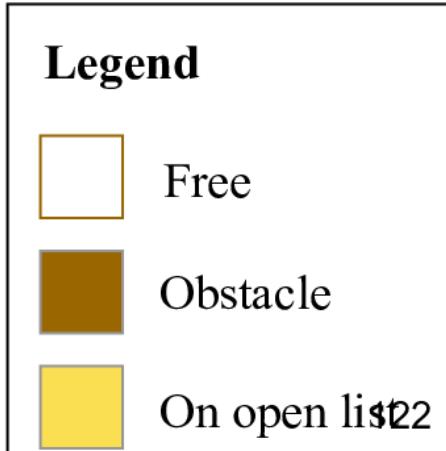
ComputeShortestPath
 Pop minimum item off open list - (2,2)
 It's under-consistent ($g < \text{rhs}$) so set $g = \infty$.



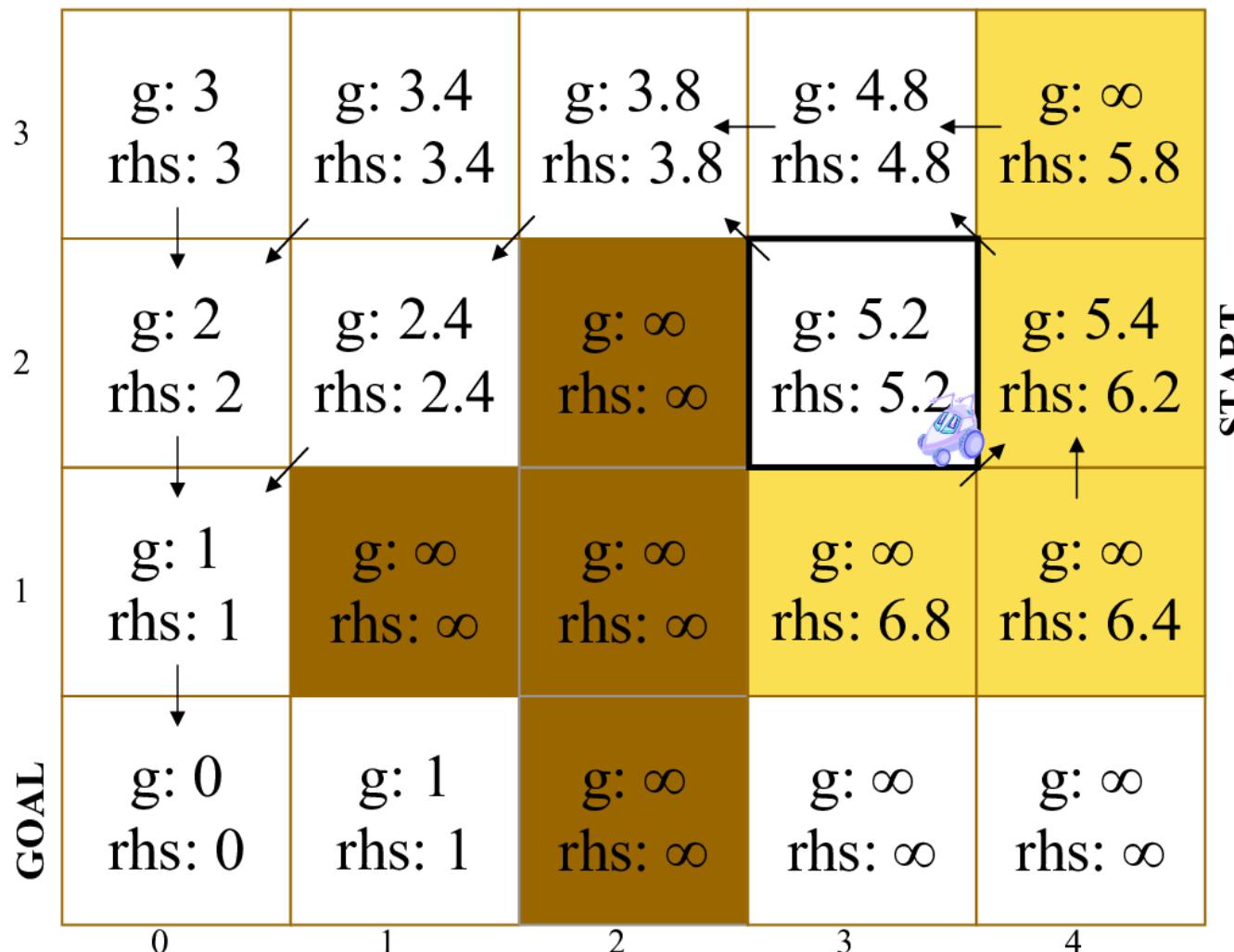
Replanification en environnement dynamique: D* Lite



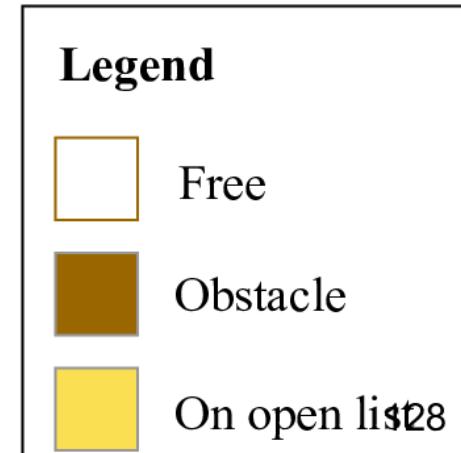
ComputeShortestPath
 Pop minimum item off open list - (3,2)
 It's under-consistent ($g < \text{rhs}$) so set $g = \infty$.



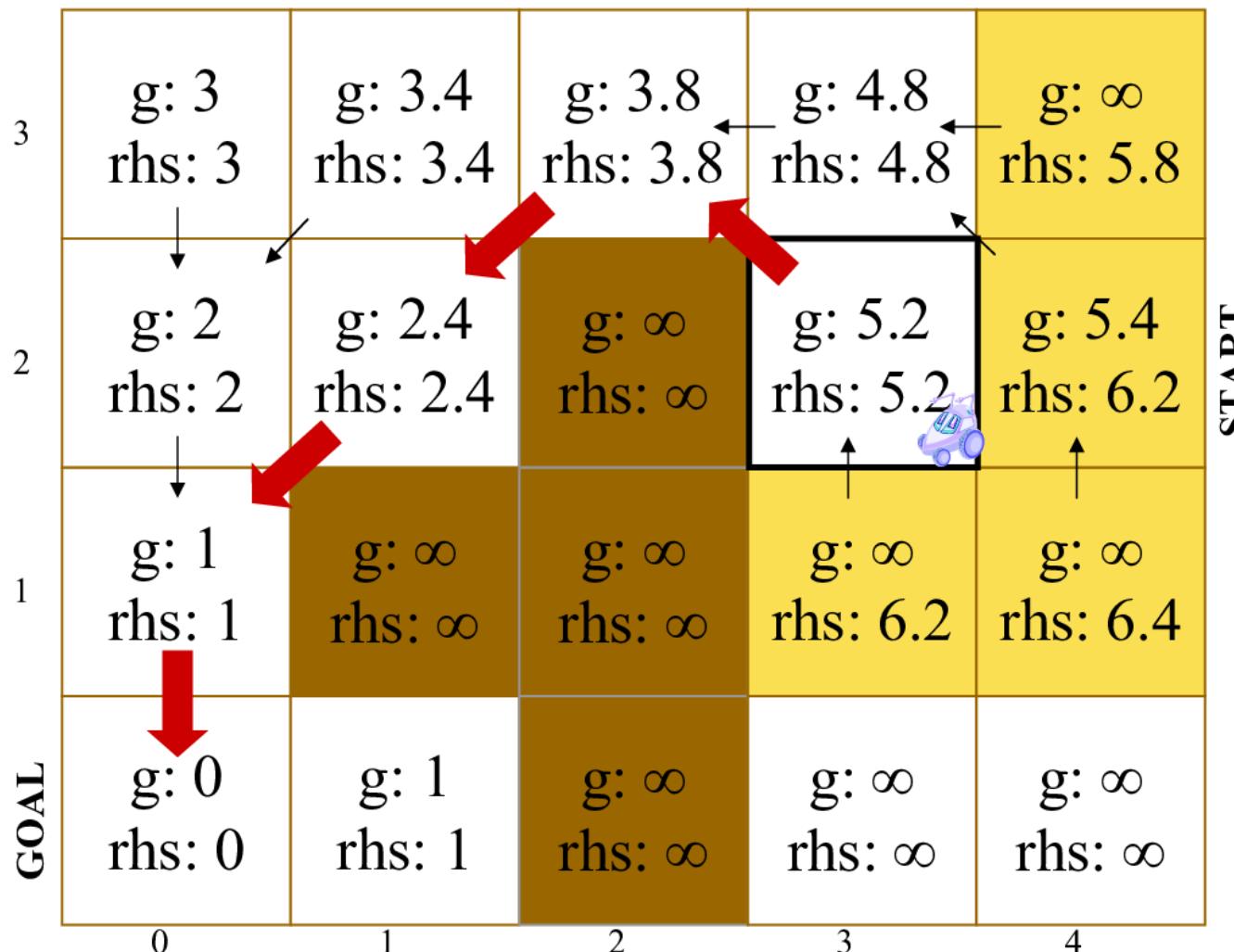
Replanification en environnement dynamique: D* Lite



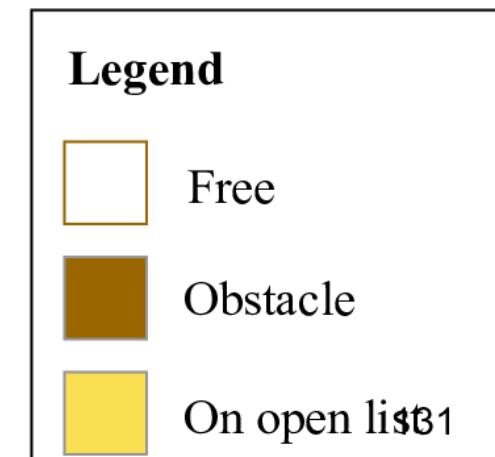
ComputeShortestPath
 Pop minimum item off
 open list - (3,2)
 It's over-consistent
 $(g > \text{rhs})$ so set $g = \text{rhs}$.



Replanification en environnement dynamique: D* Lite

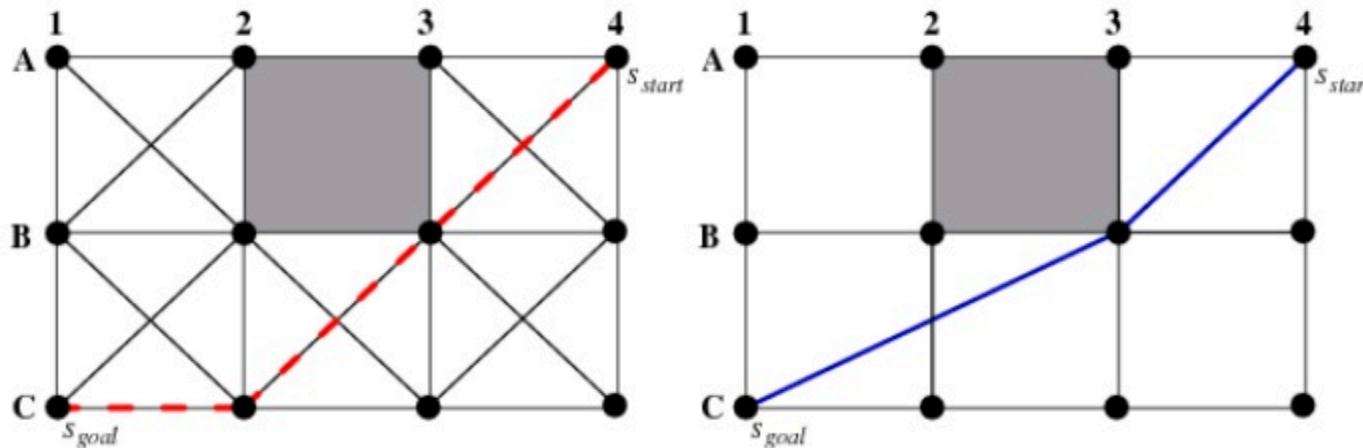


Follow the gradient of g values from the robot's current position.



NOMBREUSES variantes

- Any Angle A*, Theta*
- D*, Field D*
- Realtime variants : TRAA*



Theta* : A* + visibility graph

Couramment utilisés en pratique

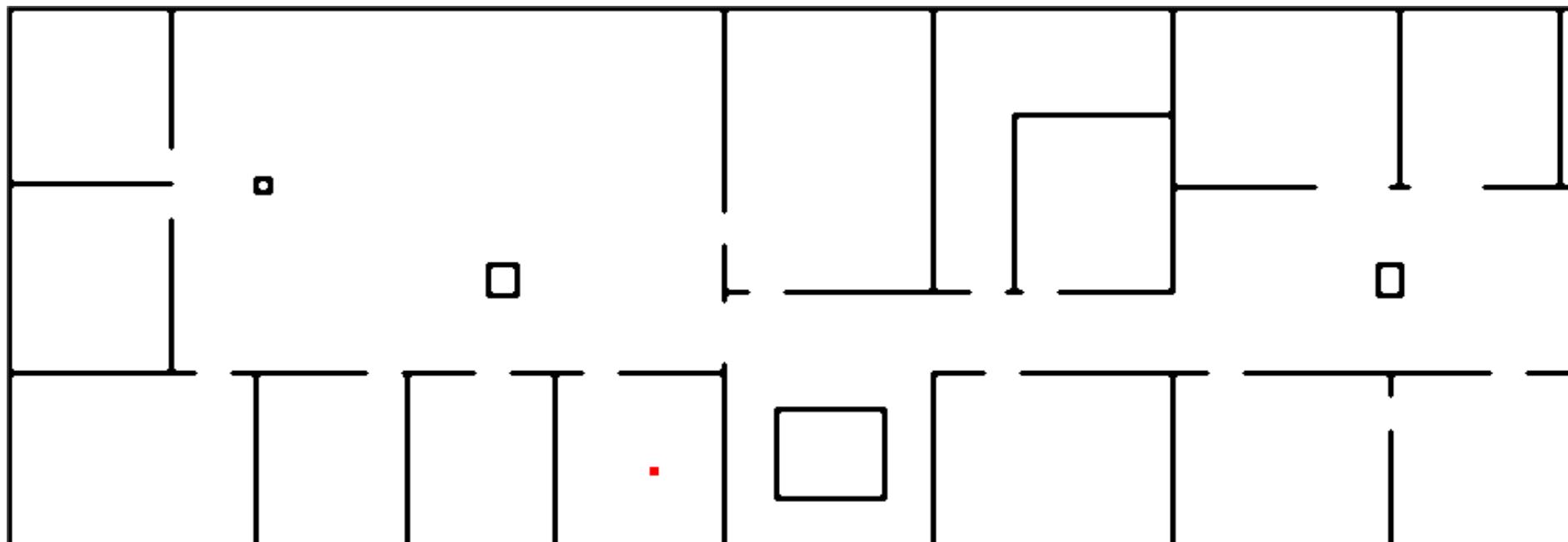
- Véhicules autonomes
- Rovers sur Mars
- Robots de service

Calcul d'une politique

Au lieu de calculer une action pour chaque cellule :

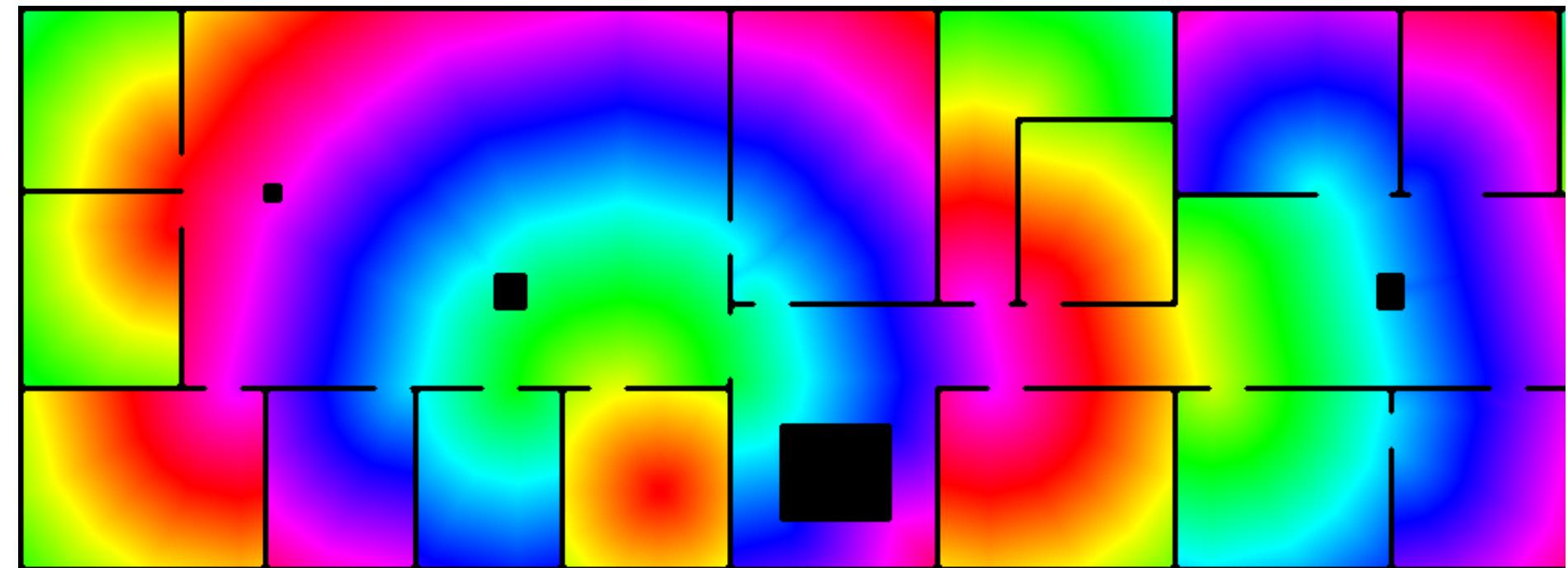
- calcul d'un poids pour chaque cellule (distance au but)
- exécution par descente de gradient

Rq : travail dans l'espace des configurations : robot = point



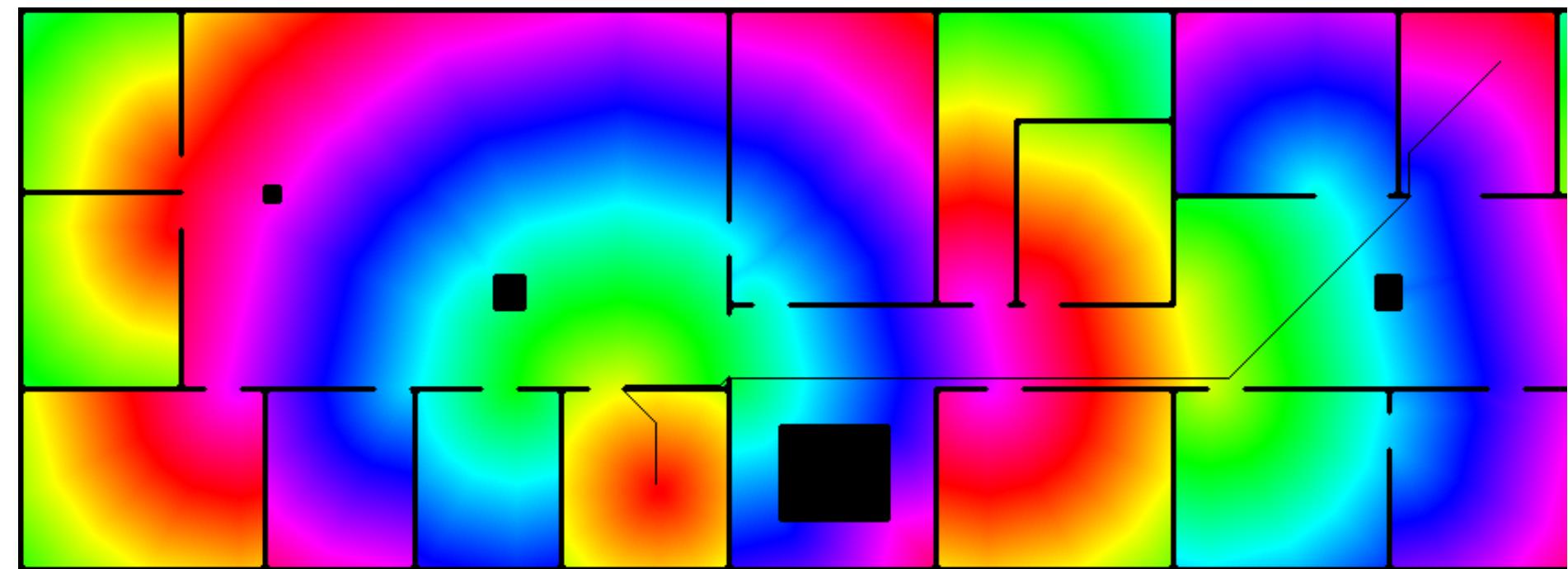
Exemple :

Poids des liens = distance euclidienne entre centre des cellules



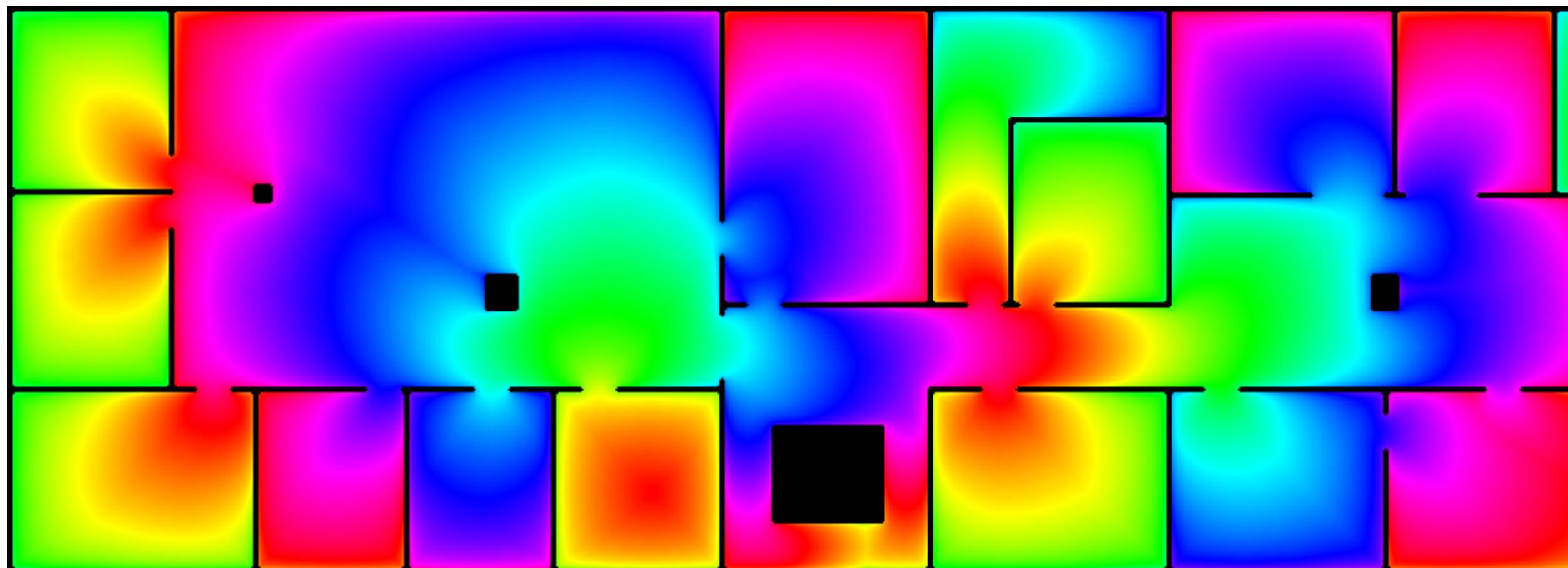
Limitation

- trajectoires proches des obstacles

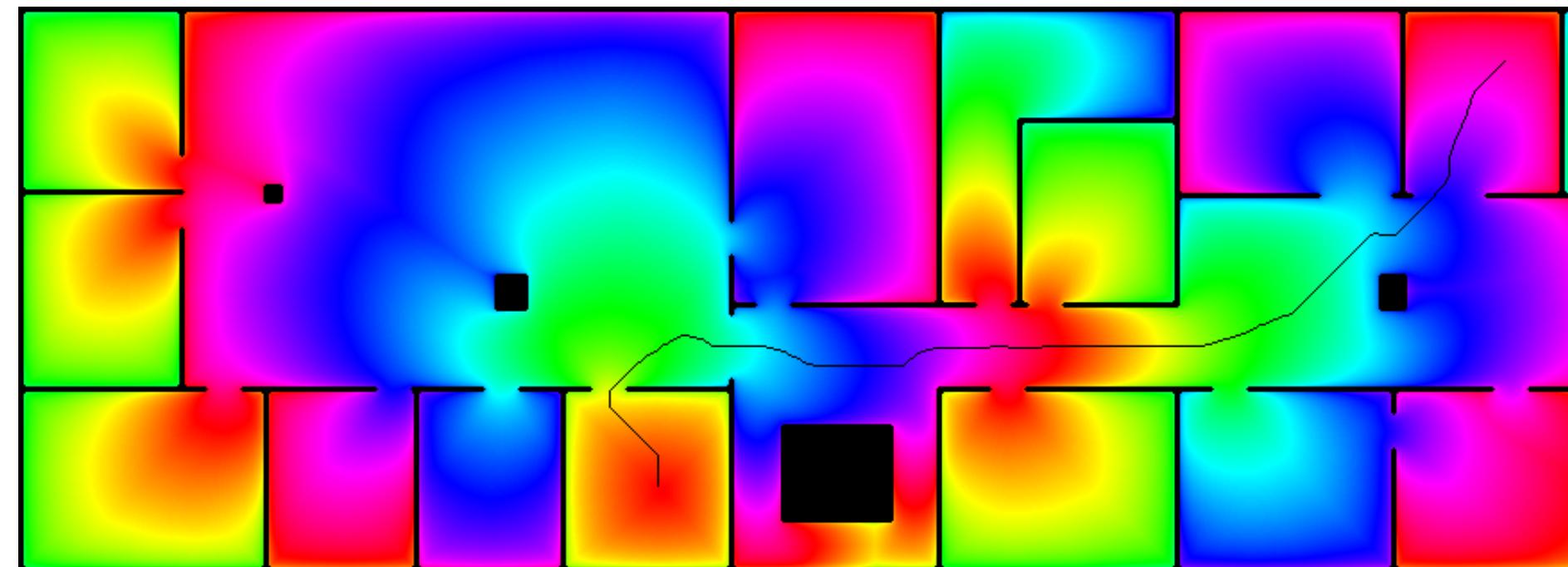


Poids associés aux nœuds

- inverse de la distance aux obstacles

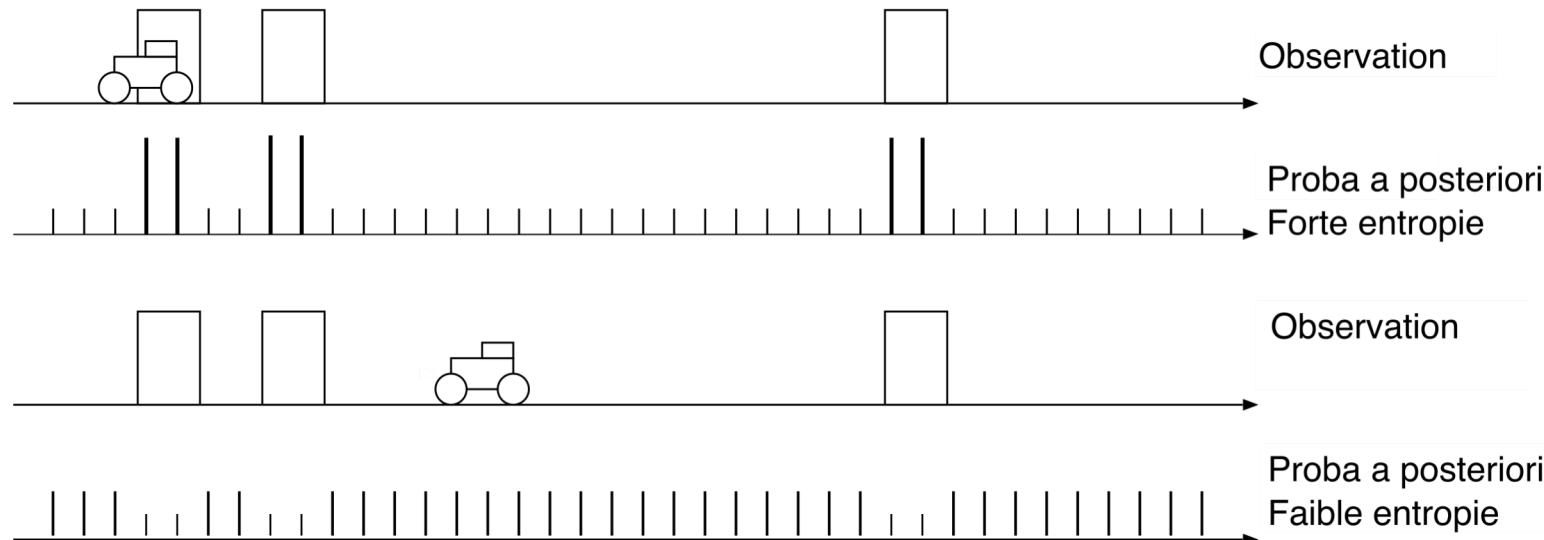


- Trajectoires moins proches des obstacles



« Coastal Navigation »

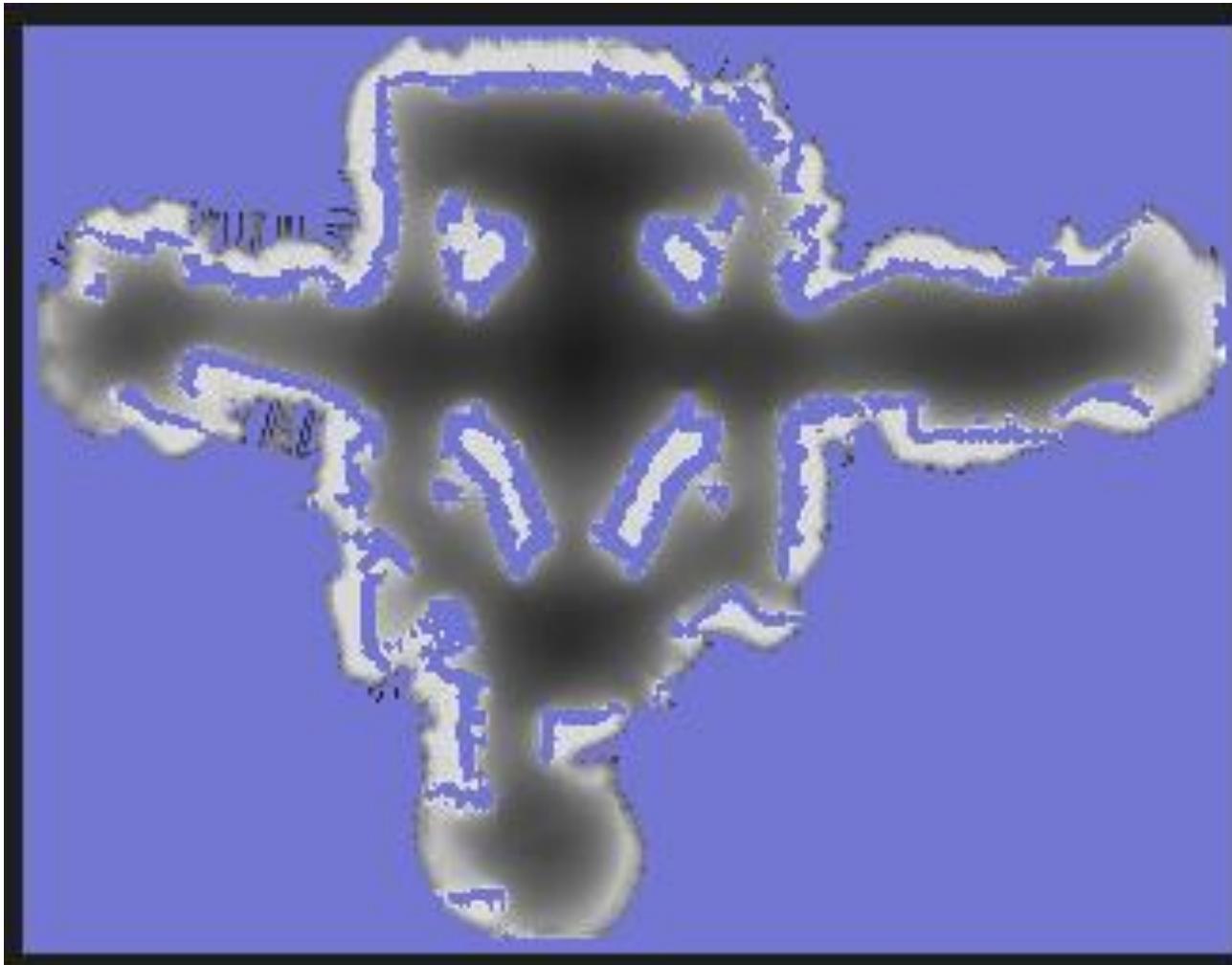
- Poids des nœuds = information disponible pour la localisation
- Calculé par la variation d'entropie après intégration des données laser pour chaque position (MCL)
- Calcul d'un chemin pour lequel la localisation est la plus fiable possible



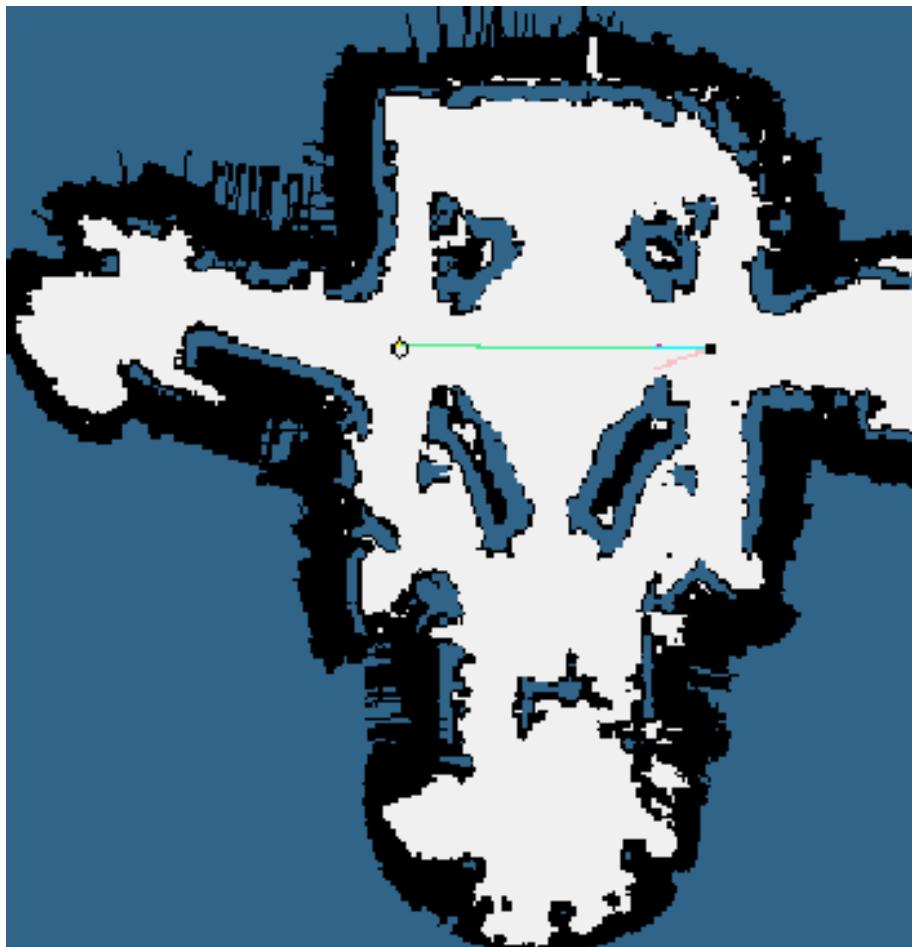
Coastal Navigation – Robot Motion with Uncertainty
Nicholas Roy, Wolfram Burgard, Dieter Fox, Sebastian Thrun

Poids des nœuds :

- Blanc : forte variation / Noir : faible variation d'entropie



Chemin le plus court



Chemin le plus « sur »



Cours « Robotic Motion planning »

- Howie Choset : <http://www.cs.cmu.edu/~choset>

Livre « Planning Algorithms »

- Steven Lavalle : <http://planning.cs.uiuc.edu>

Slides

- Introduction to Mobile Robotics , Robot Motion Planning - Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras

<http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf>

- Robotic Motion Planning: A* and D* Search, Howie Choset
- https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf

