Robotique Mobile
15 – Planification de trajectoire

David Filliat

Alexandre Chapoutot

Goran Frehse

prenom.nom@ensta-paris.fr

Planification

- Planification réactive : « planifier » sans carte
- Planification dans une carte
- Discrétisation des espaces de recherche
- Recherche de chemin dans un graphe
- Recherche de chemin stochastique

Planification réactive

Hypothèses

- Carte inconnue
- Direction du but connue
- Perception locale (tactile)

Inspiration biologique

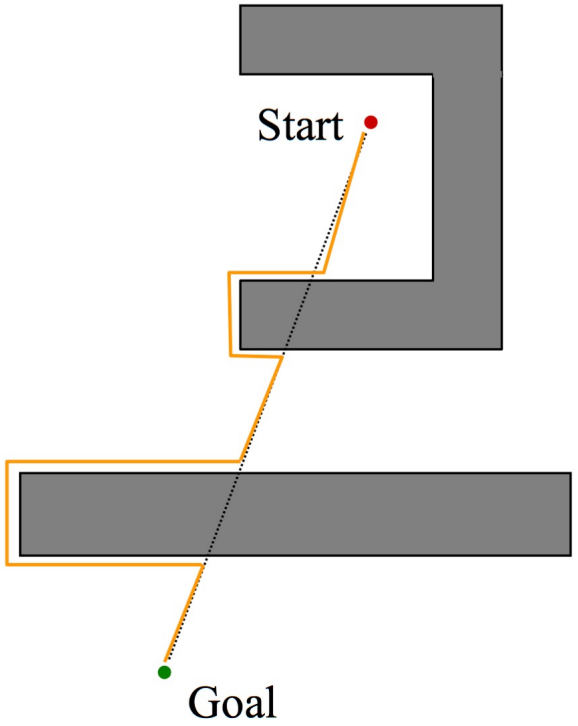
- Algorithme « Bug »
- Variantes : Bug 0, Bug 1, Bug 2, Tangent Bug

Limitations

- Ne garantis pas un chemin optimal
- N'exploite pas la connaissance de la carte
- Risque de comportements aberrants dans certains environnements

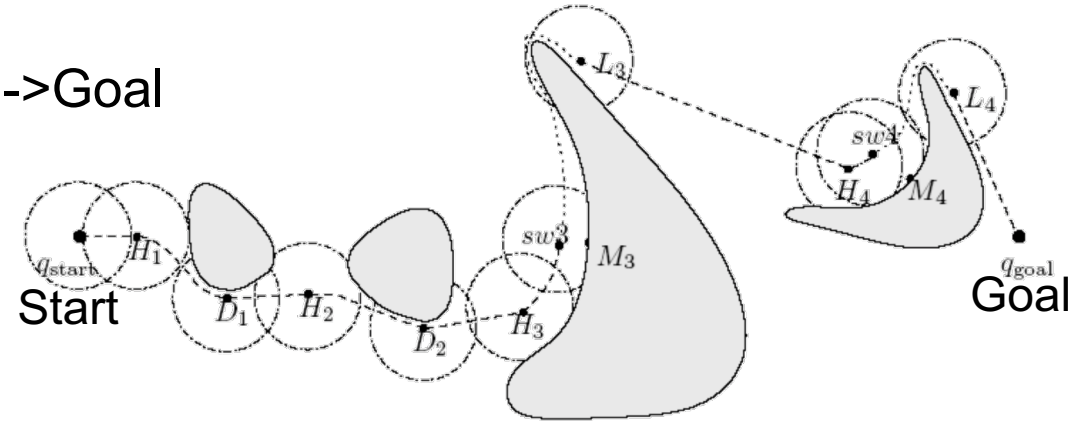
Algorithme Bug 2

- Suivre la ligne Start->Goal
- En cas d'obstacles, suivre l'obstacle jusqu'a retomber sur la ligne
- Reprendre la ligne



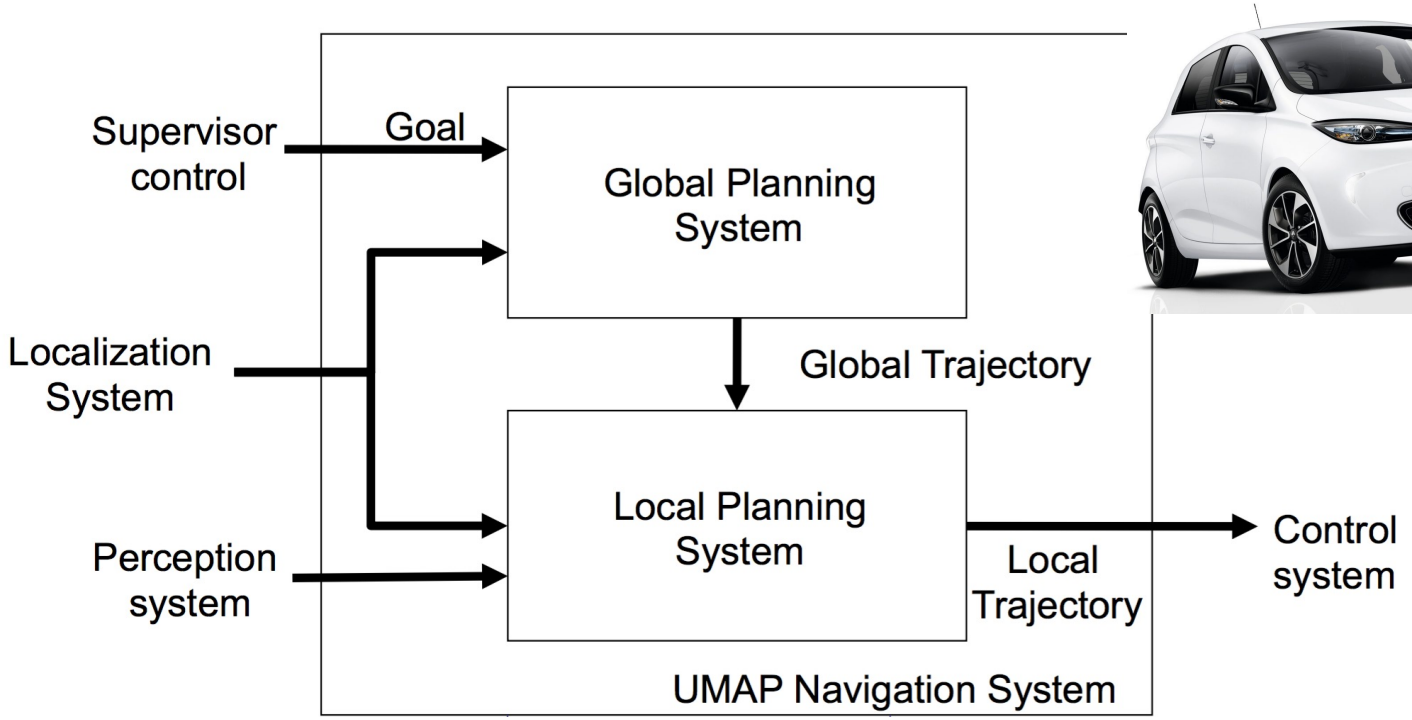
Algorithme Tangent Bug

- Suivre la ligne Start->Goal
- En cas d'obstacles, contourner jusqu'à « voir » le but
- Suivre la ligne Position->Goal



Planification dans une carte

Deux niveaux de planification (cf architecture hybrides)



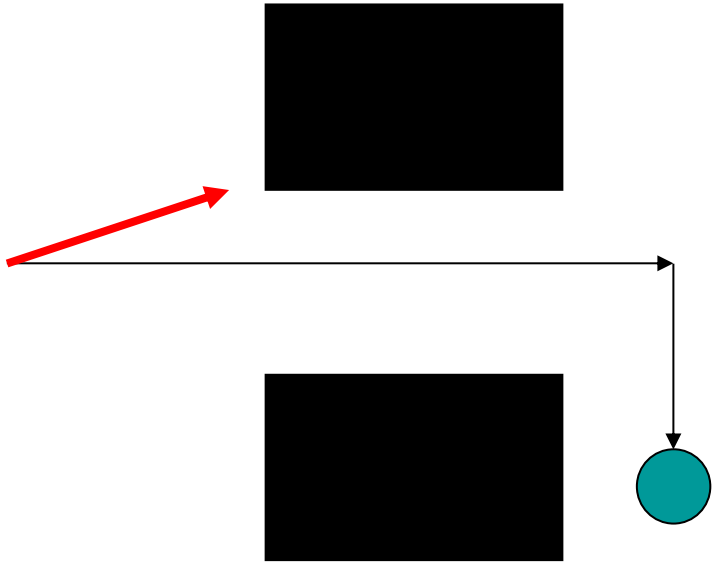
Deux types de plans haut-niveau

- Chemin
- Politique

  cf Apprentissage par renforcement, champs de potentiels …

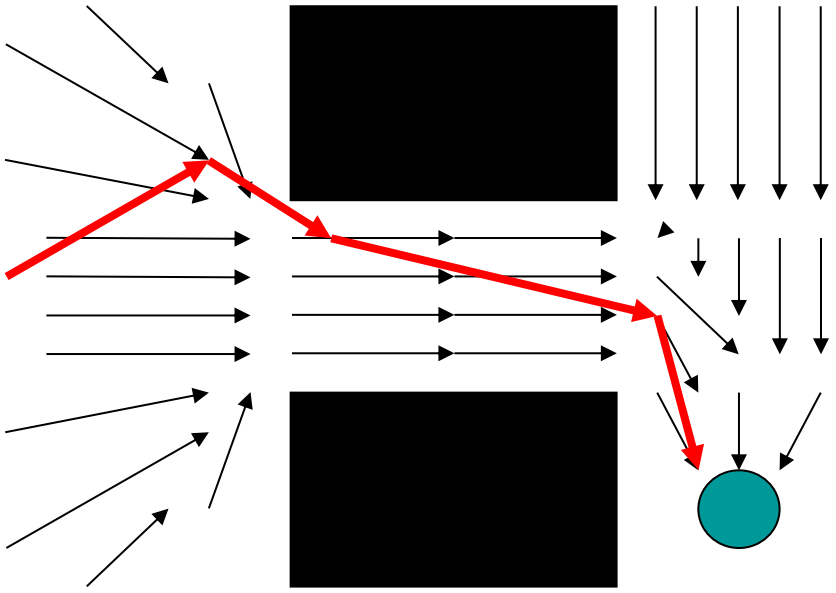
Plan = suite de nœuds et de liens ou trajectoire

- Succession prédéfinie d'actions
- Problèmes en cas d'exécution imprécise -> re-planification
- 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

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
- 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

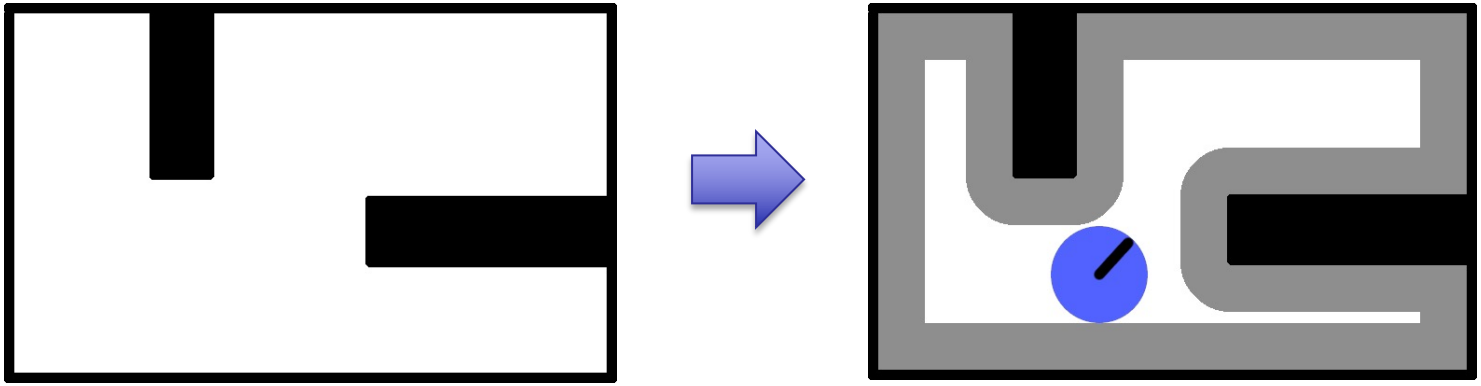
Espace continu -> discrétisation pour planifier dans le graphe correspondant

- Discrétisation en cellules
- Discrétisation en chemins

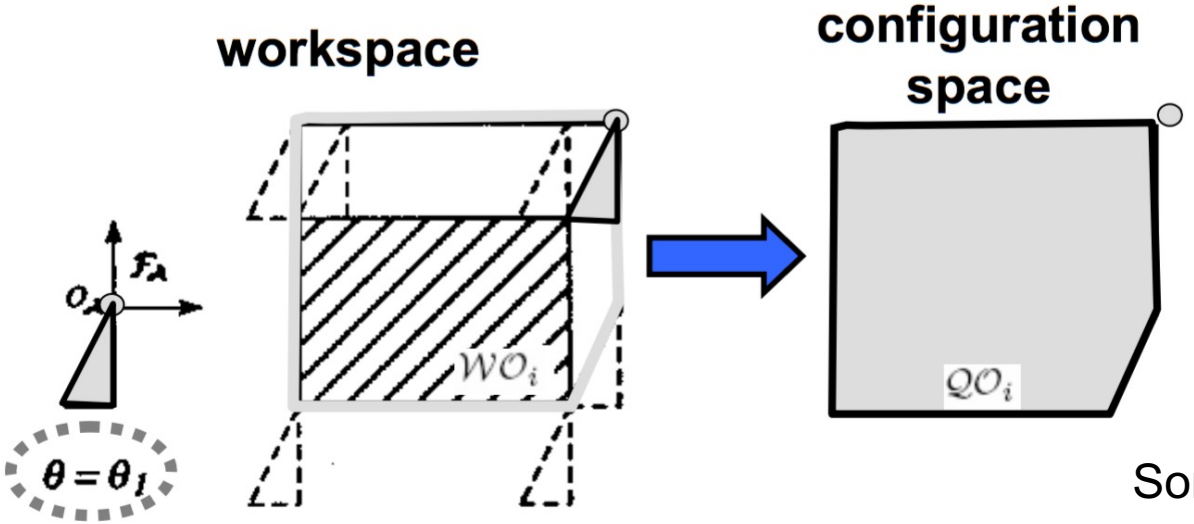
Espace des configurations

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

Cas d'un robot circulaire

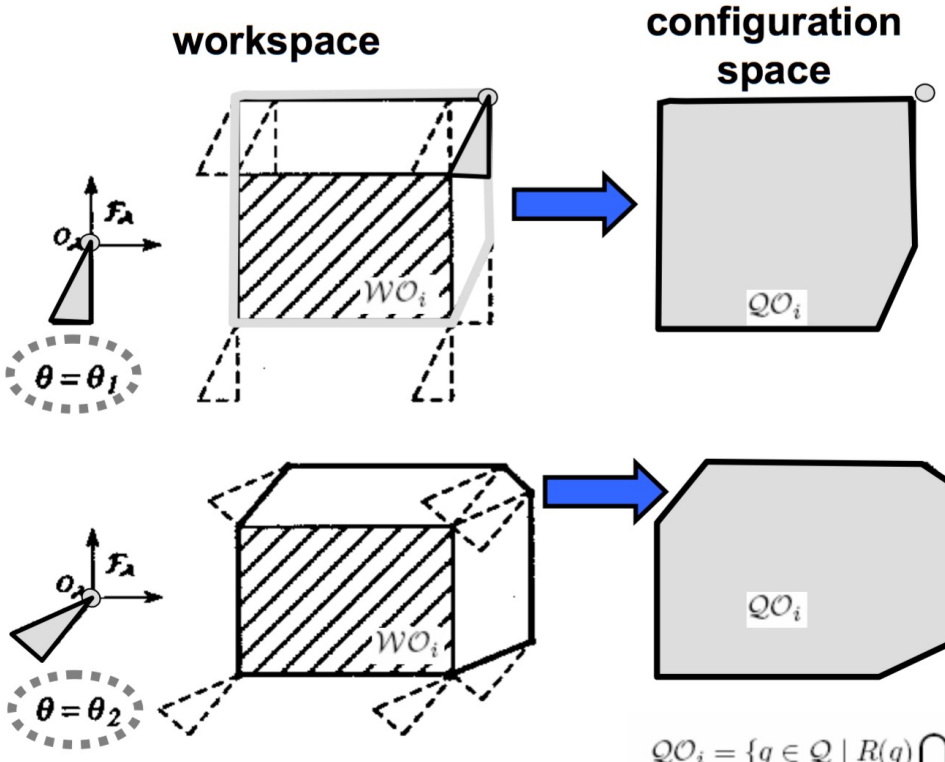


Cas d'un robot polygonal en translation



Somme de Minkovski

Cas d'un robot en translation/rotation

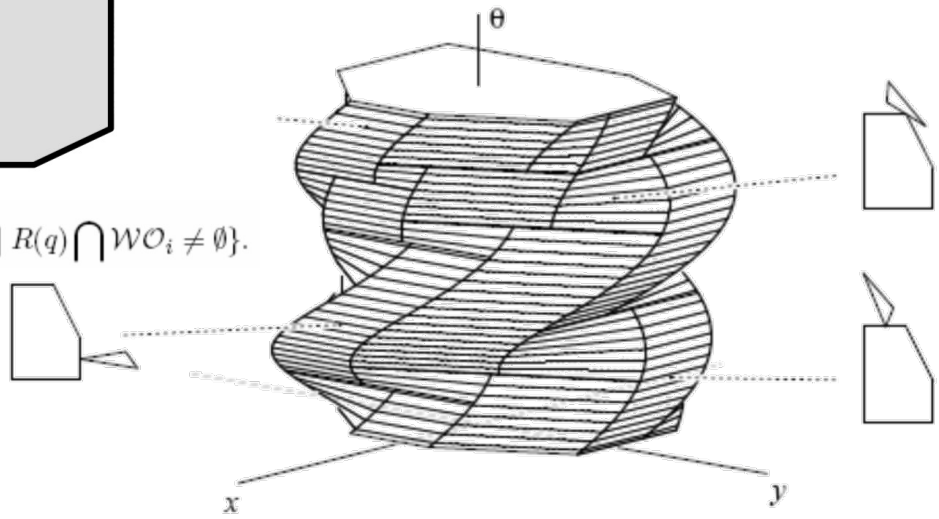


$$\mathcal{QO}_i = \{q \in \mathcal{Q} \mid R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}.$$

Espace de configuration 3D

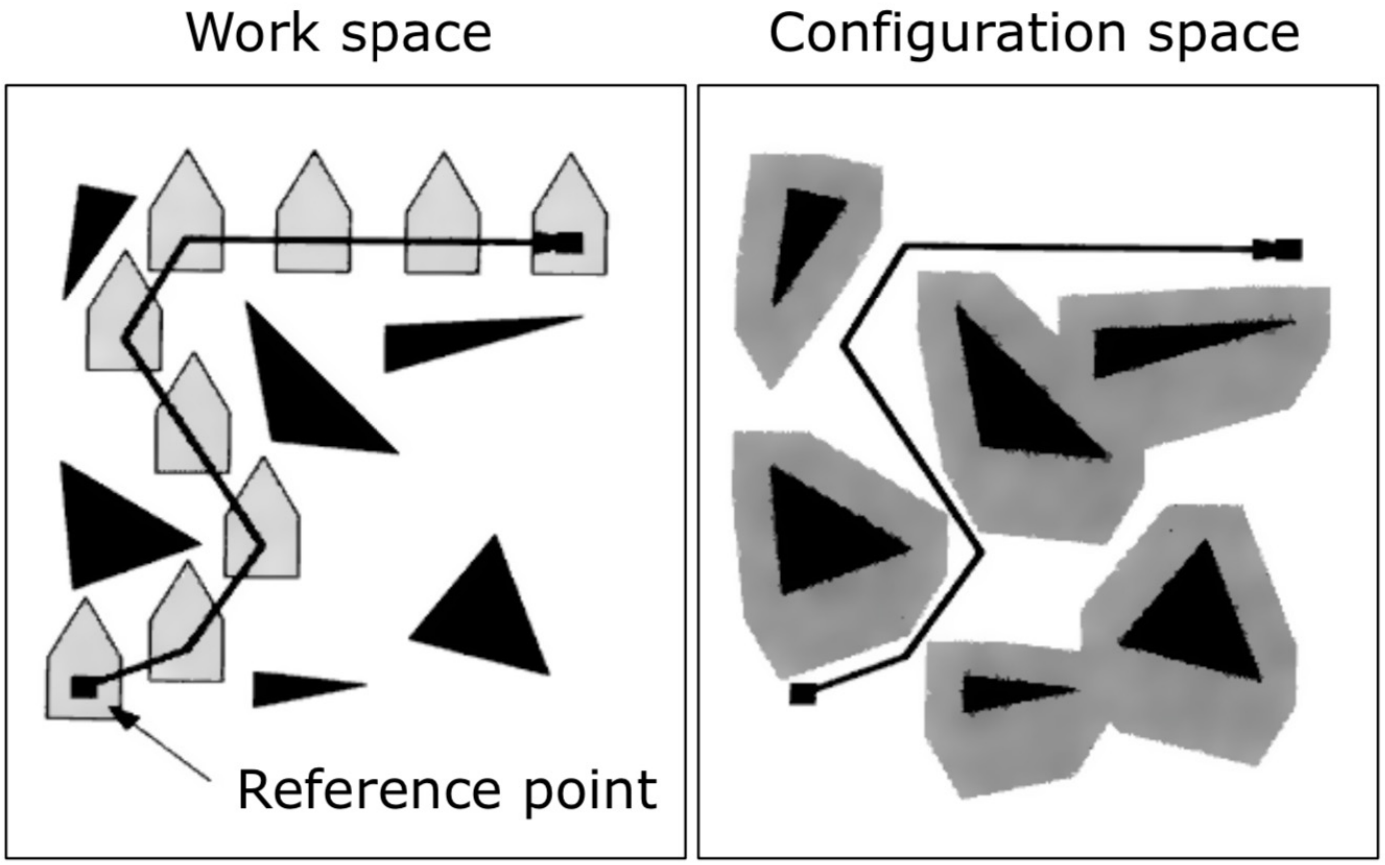
! Complexe à calculer

-> échantillonnage

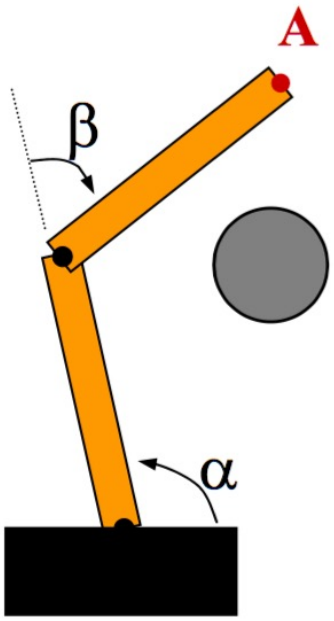


Planification dans l'espace de configuration

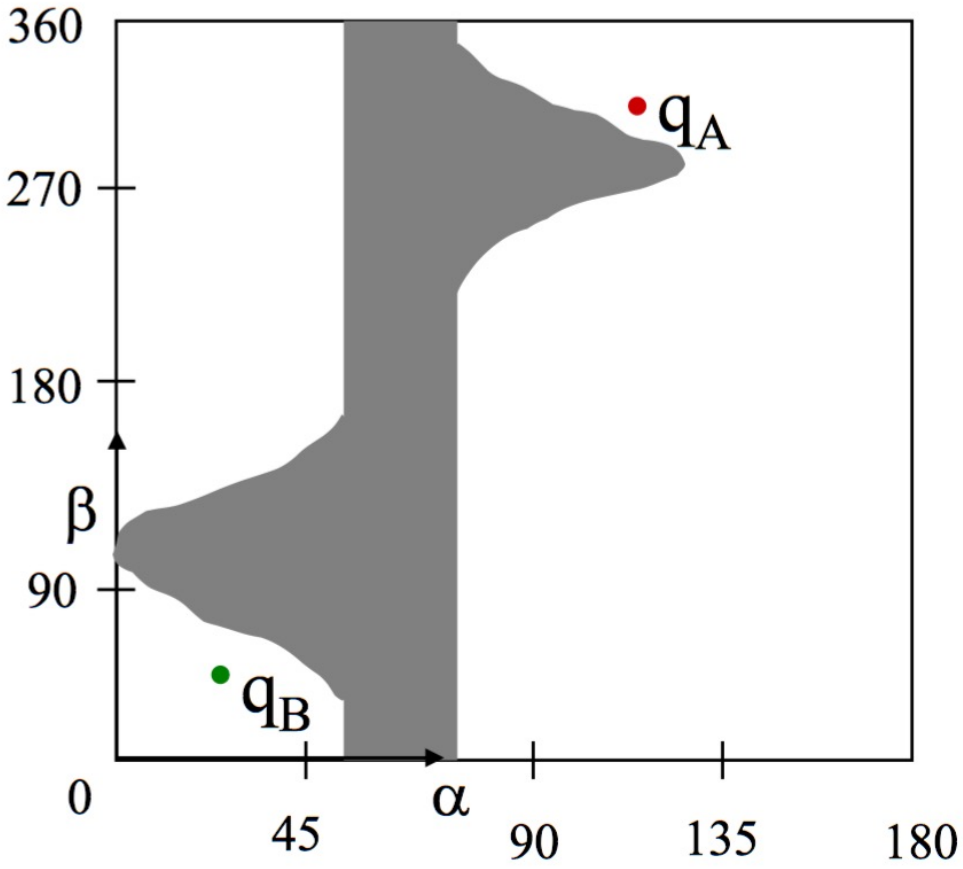
– Ramène à la trajectoire d'un point



Applicable à tous les mécanismes



B



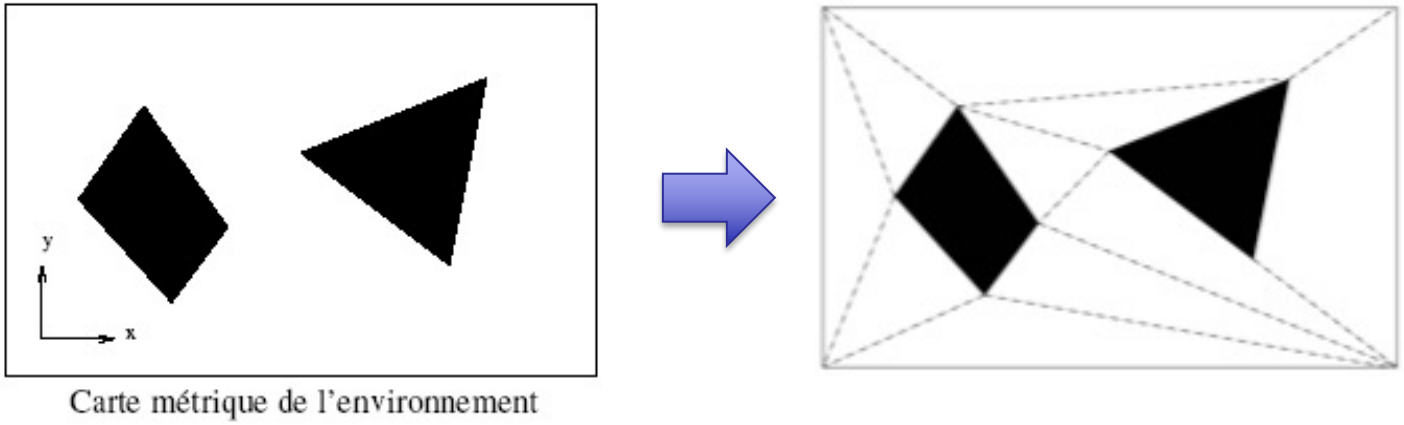
Chemin entre A et B ?

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



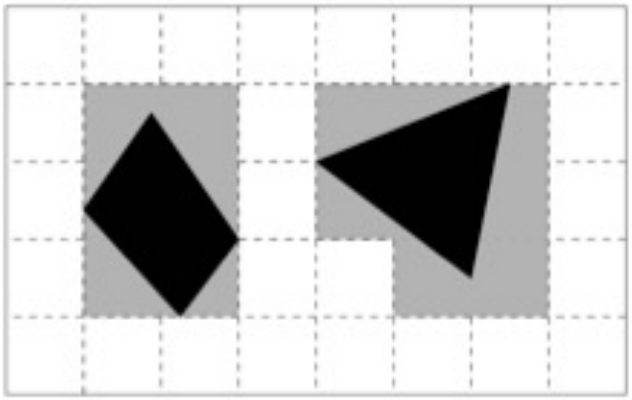
Carte métrique de l'environnement

- Pas très adapté aux carte produites par SLAM
- Utile dans des environnement simples/statiques

Discrétisation de l'espace : cellules

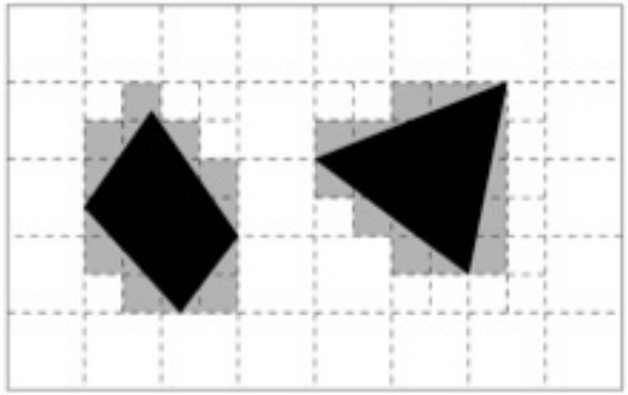
Décomposition en cellules régulières

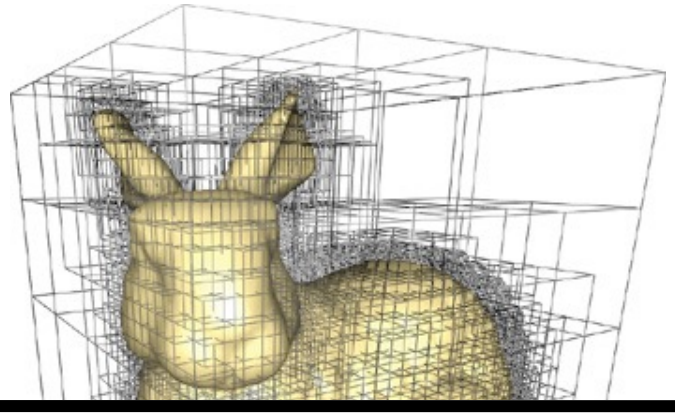
- Directement disponible avec les grilles d'occupation
- Grande consommation mémoire pour les grands environnements



Décomposition en quad-trees

- Réduction de l'espace mémoire
- Réduction de l'espace de recherche
- Simplification de grilles d'occupations
- Version 3D : Octrees



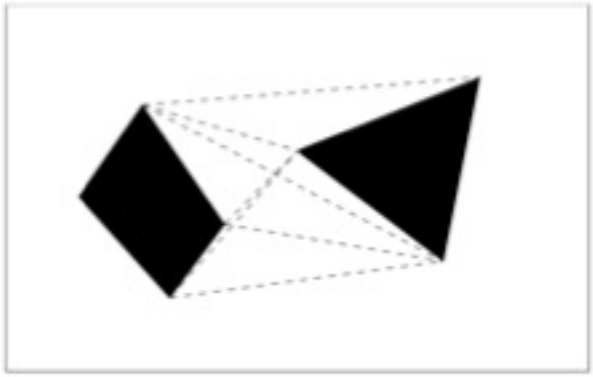


Discrétisation de l'espace de configuration

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

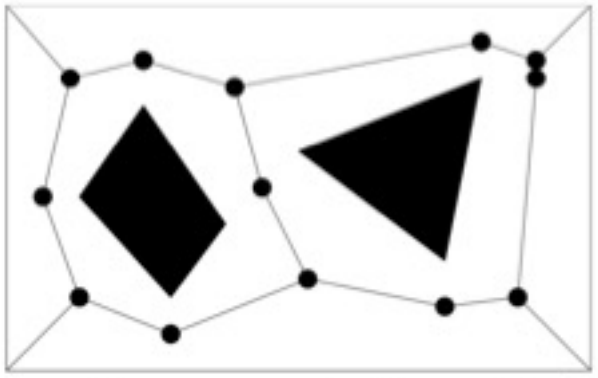
Graphe de visibilité

- Relie les coins des obstacles visibles entre eux

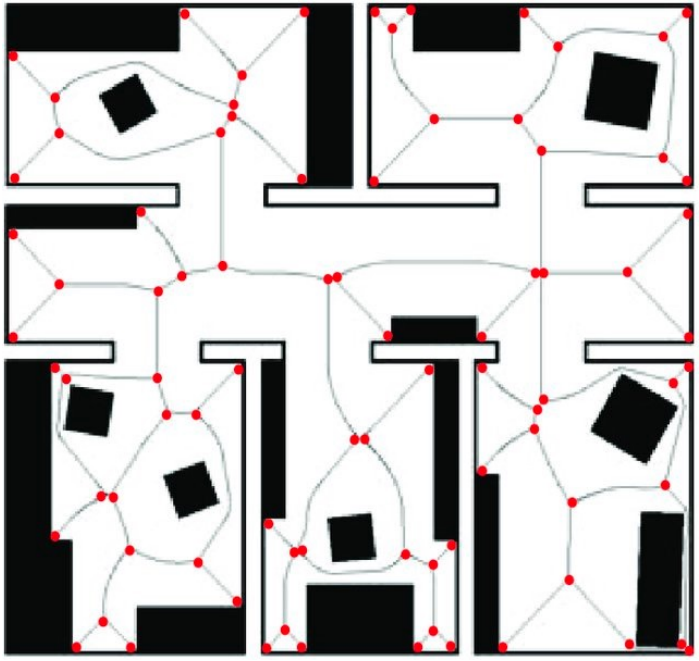


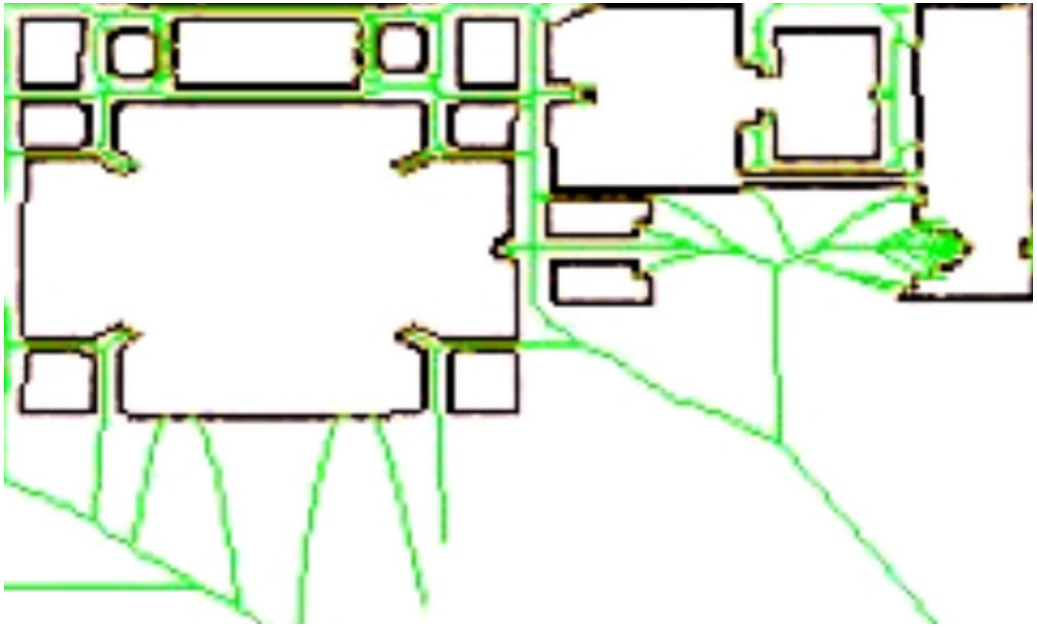
- Historiquement l'une des premières approches
- Adapté aux cartes polygonales
- Chemins optimaux, mais passent près des obstacles

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 discrétiser 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

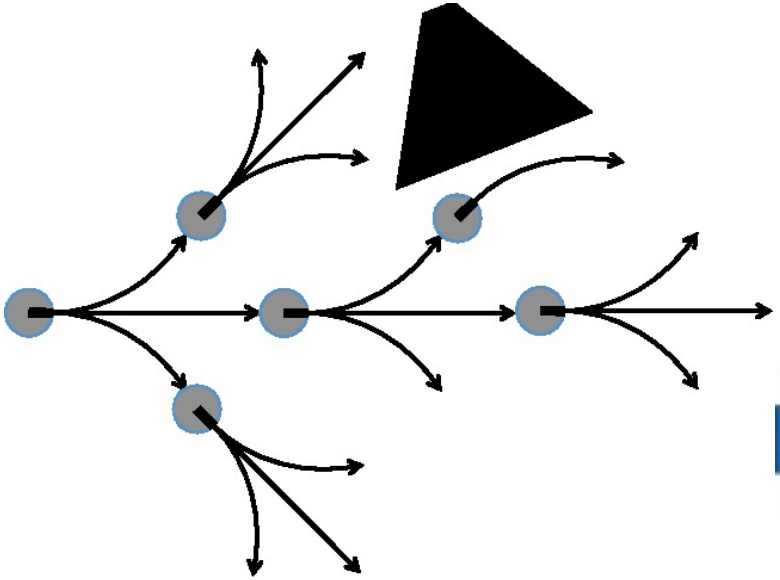
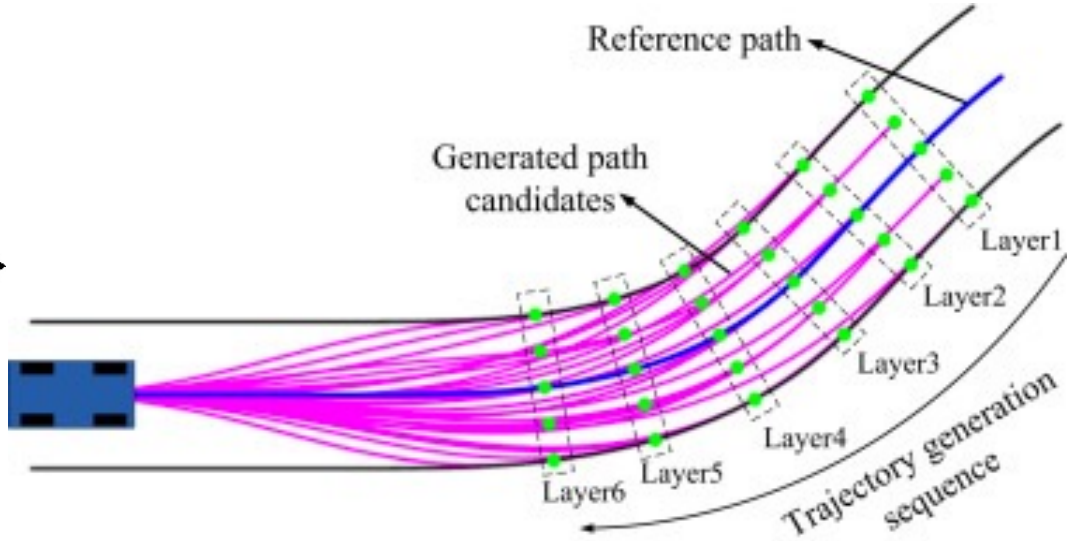
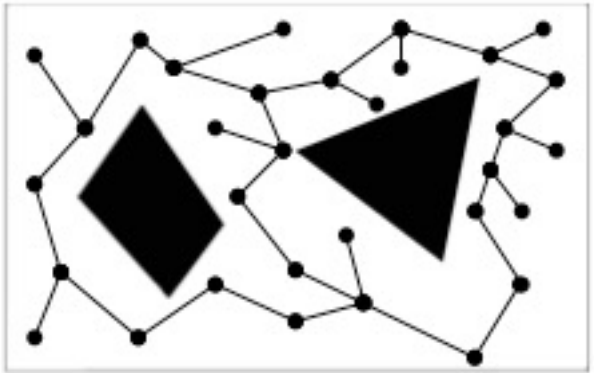


Fig. 10. An example of a lattice type graph



Echantillonnage aléatoire

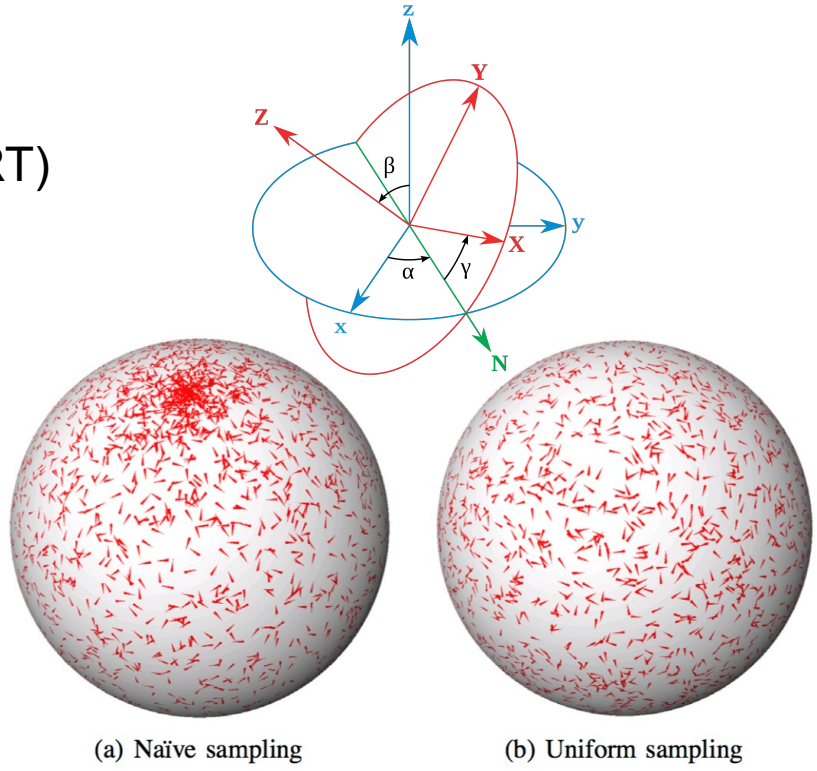


- Permet de représenter les grands espaces de manière minimale
- Nécessite simplement une procédure de détection de collision
- Différentes méthodes dont
  - Probabilistic Road Maps (PRM)
  - Rapidly exploring Random Trees (RRT)

Echantillonnage des angles

- Echantillonnage uniforme en 3D
- Attention méthode naive avec Euler

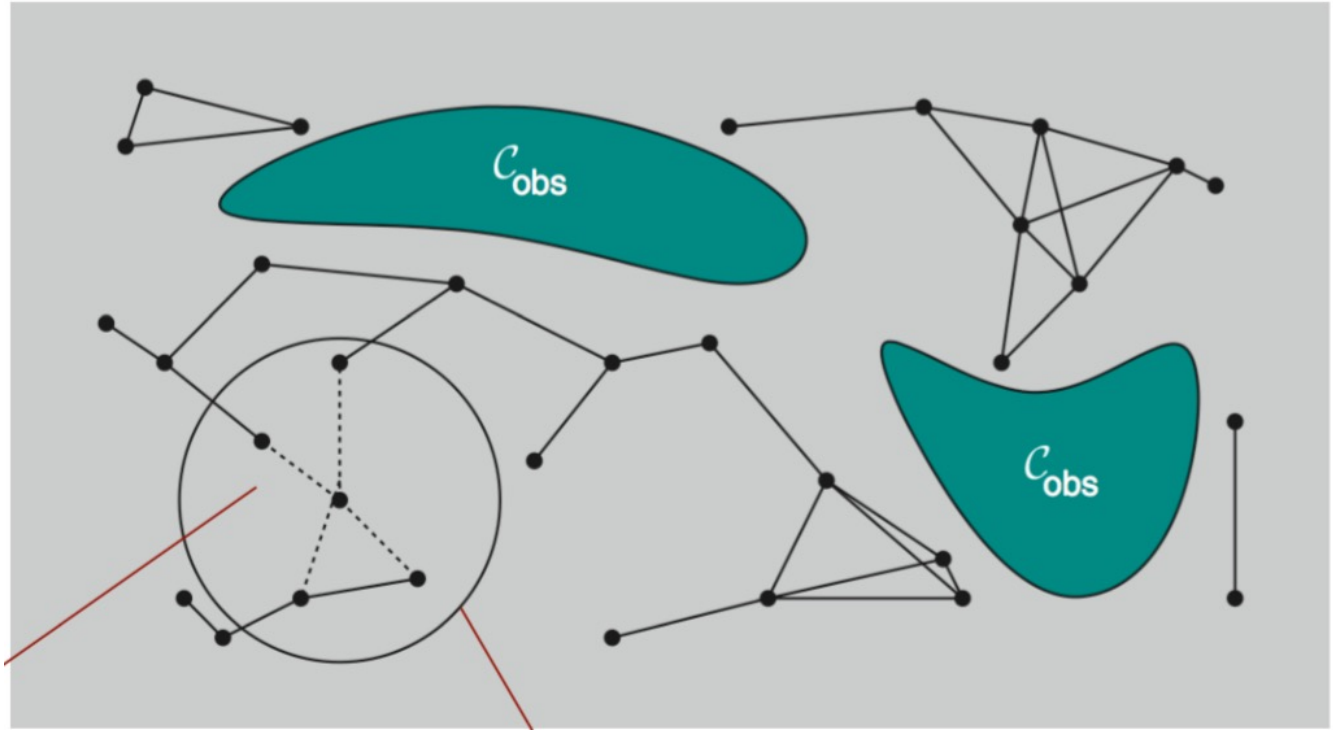
  -> quaternions

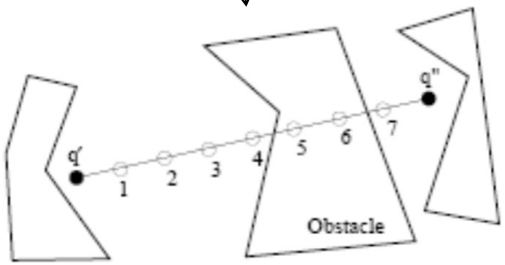


(a) Naïve sampling (b) Uniform sampling

Probabilistic Road Maps (PRM, 92)

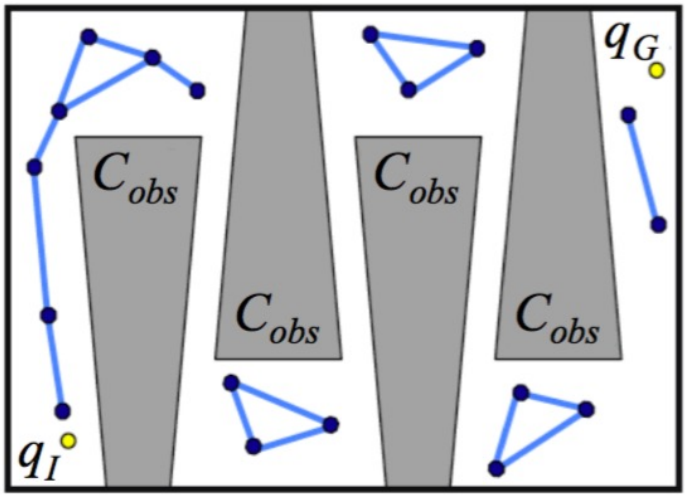
- Choisir des points aléatoirement dans l'espace libre
- Relier les points proches par un planificateur local (simple et rapide)
- Continuer jusqu'à ce que le graphe soit « assez dense »





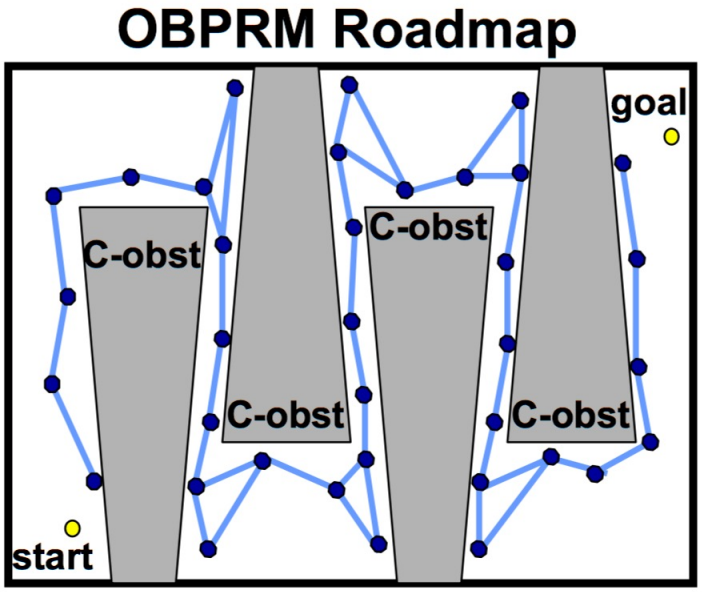
Probabilistic Road Maps (PRM, 92)

- 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
- …



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

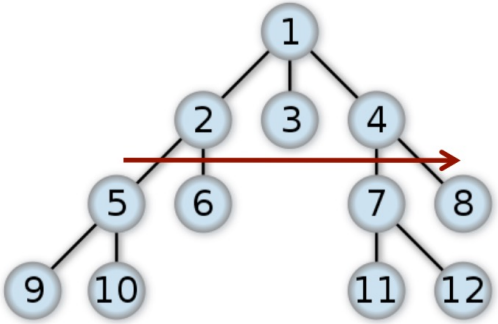
- Heuristique donnant la distance au but
- Par exemple : distance en ligne droite
- Doit être « admissible » : < longueur chemin le plus court

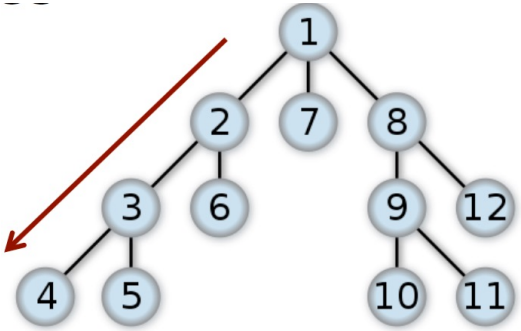
Coût

- 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





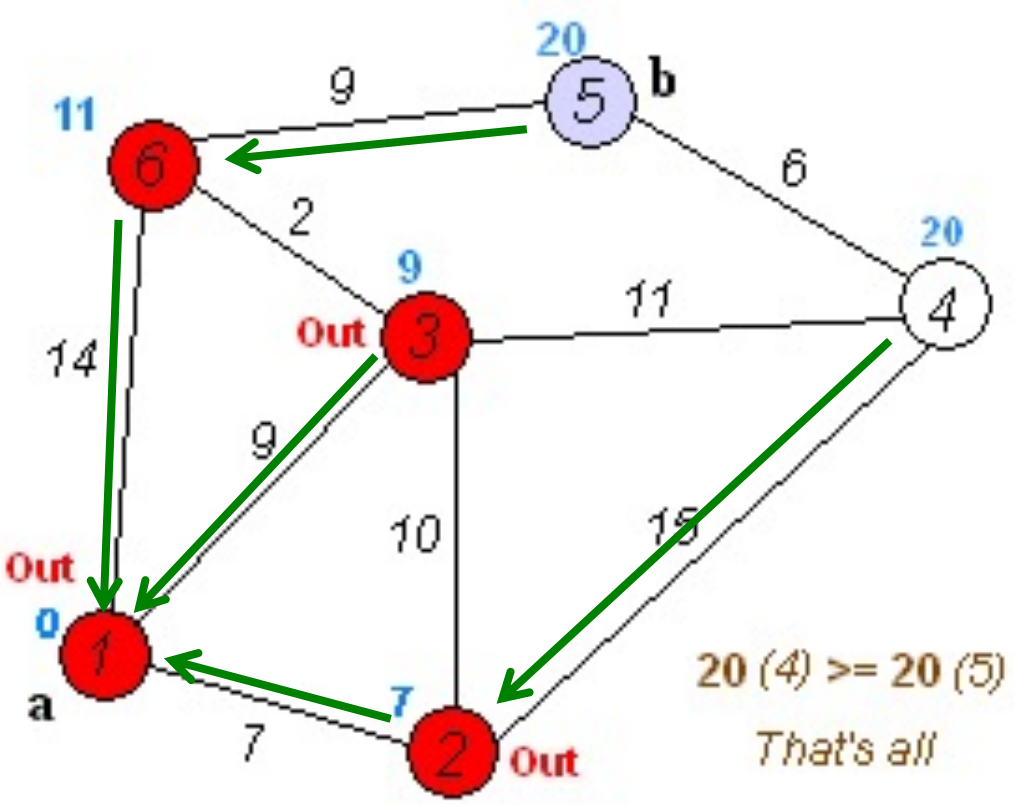
Cas de poids tous positifs : algorithme de dijkstra

- S := ensemble vide; Q := ensemble de tous les nœuds; d = 0 pour le but; d= infini pour les autres noeuds
- tant que Q n'est pas l'ensemble vide faire
  - u := Extract-Min-d(Q)
  - S := S union {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); previous[v] := u

Choix des actions

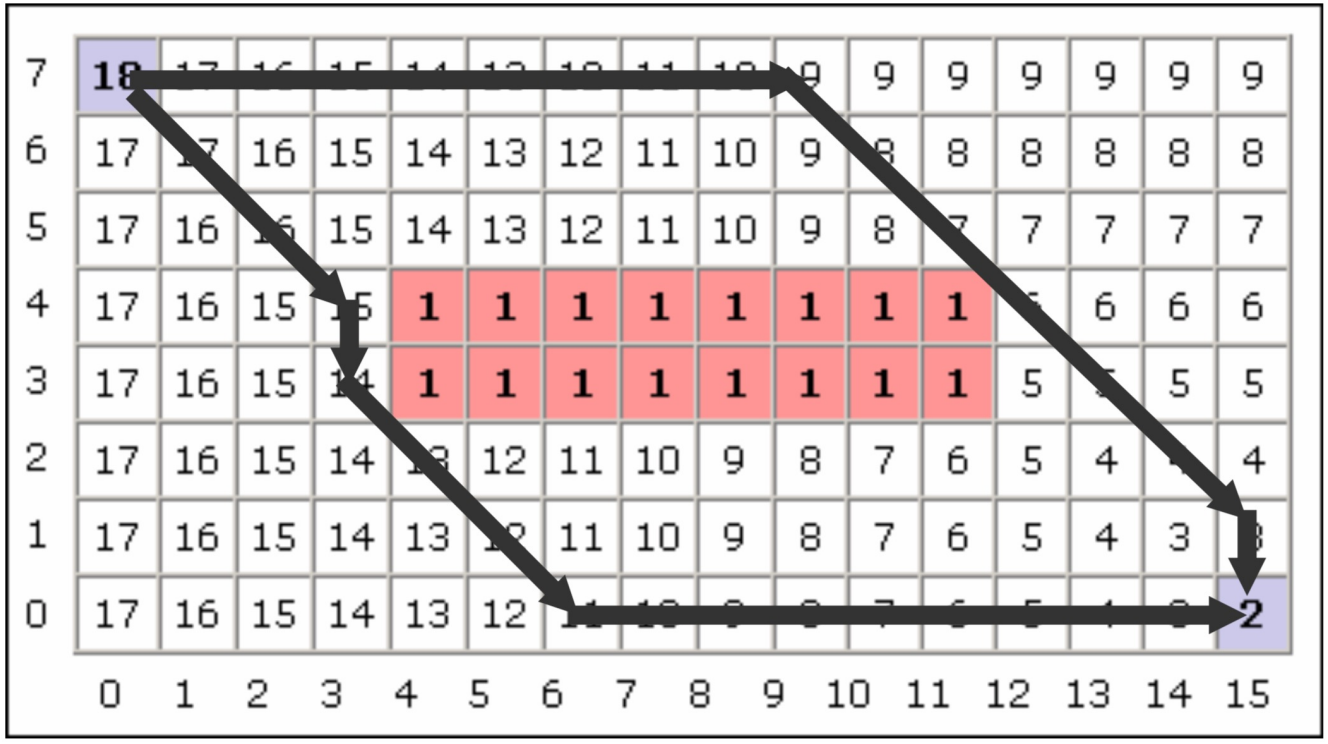
– Descente de gradient sur le cout des nœuds

Illustration



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



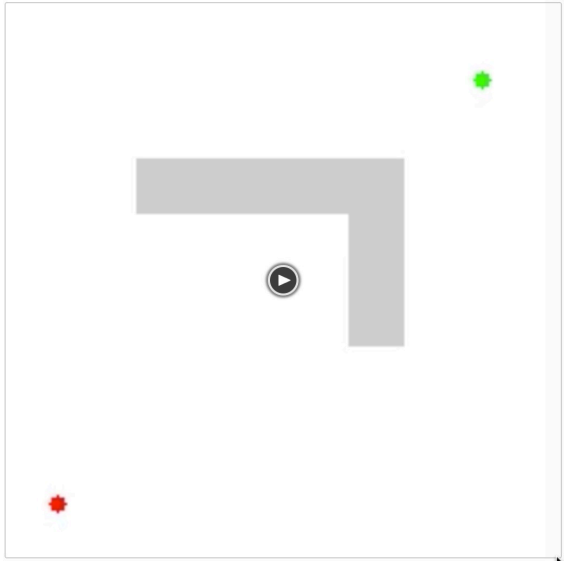
Pour calculer un seul chemin : algorithme A*

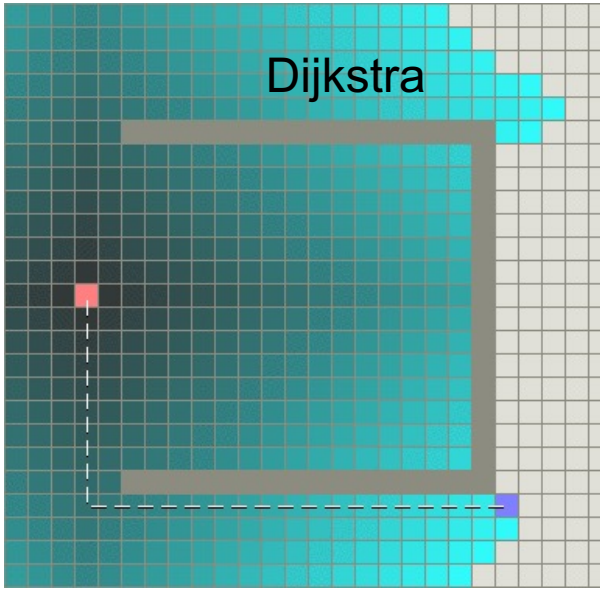
- Similaire à l'algorithme de Dijkstra
- Algo informé : ajout d'une heuristique estimant la distance d'un point au but (ex : distance euclidienne)
- Traitement en priorité des points ayant la plus faible valeur

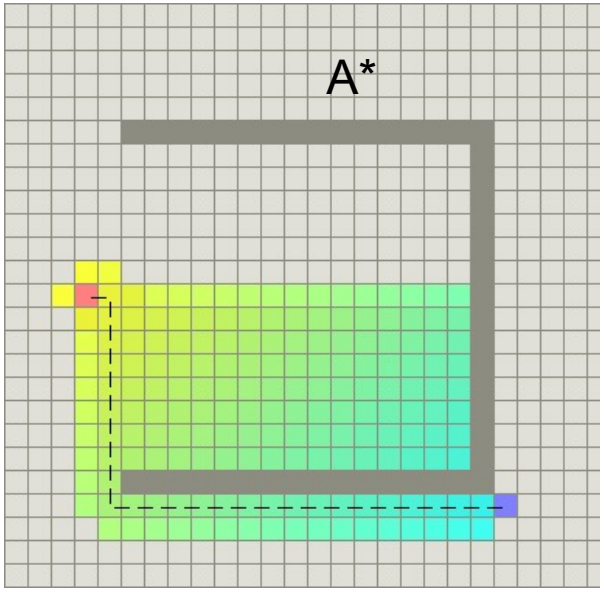
  distance (départ) + μ*heuristique

  μ règle l'influence de l'heuristique

  → Permet de ne parcourir qu'une partie des états



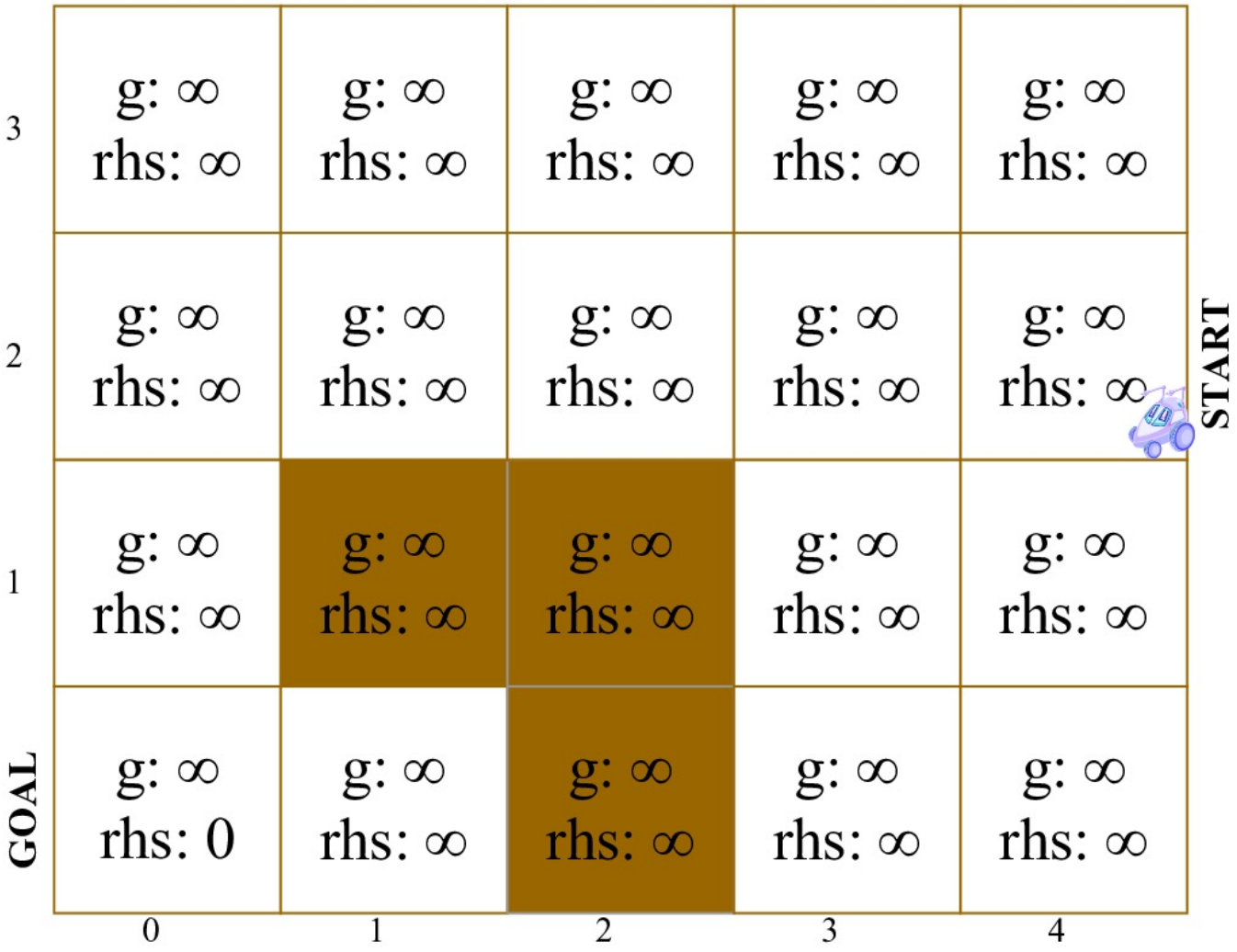




Replanification en environnement dynamique: D* Lite

- 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 != rhs -> replanifier
- Mettre les nœuds inconsistants « ouvert » et mettre à jour leurs voisins

Replanification en environnement dynamique: D* Lite



Initialization

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

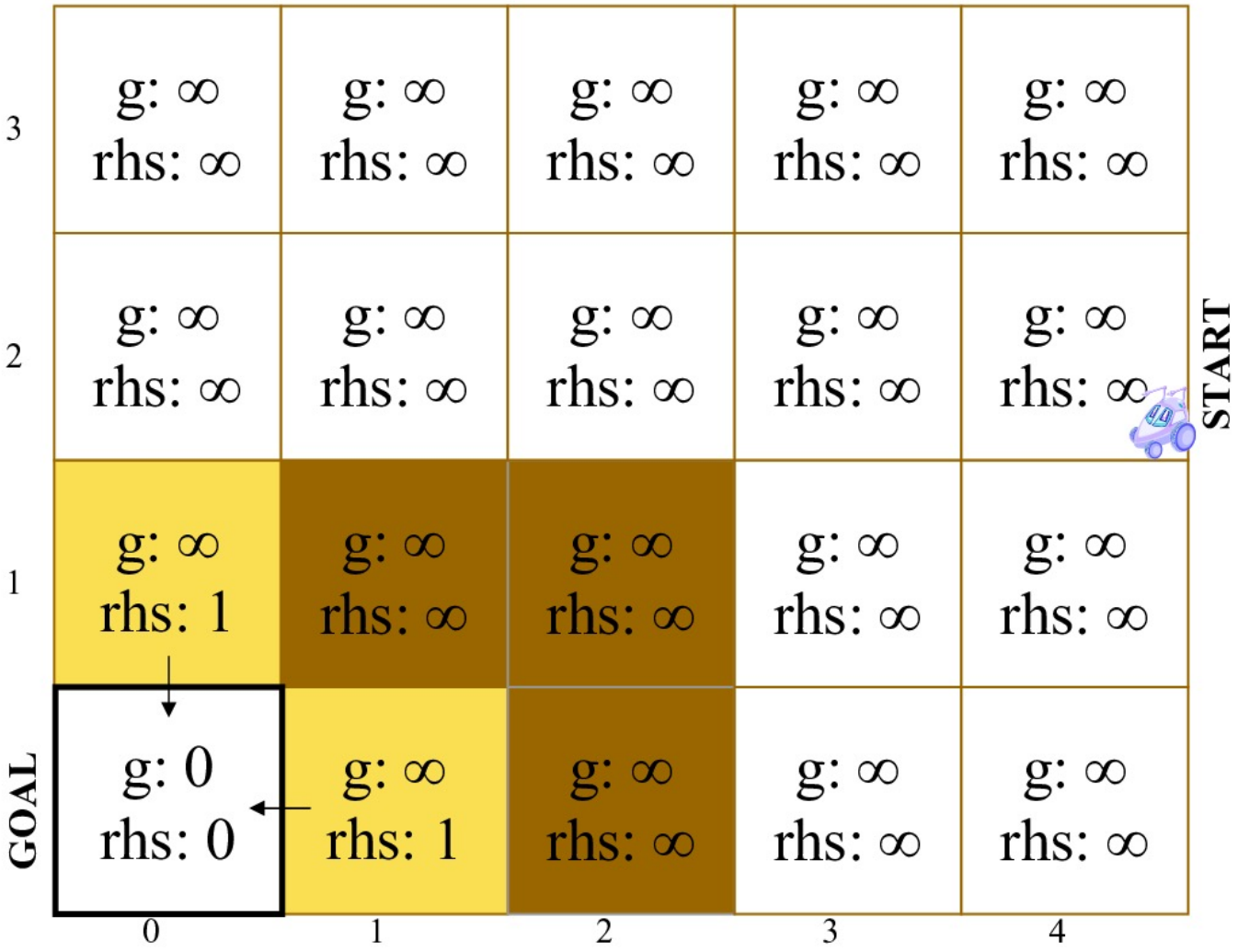
Legend

Free

Obstacle

On open list

Replanification en environnement dynamique: D* Lite

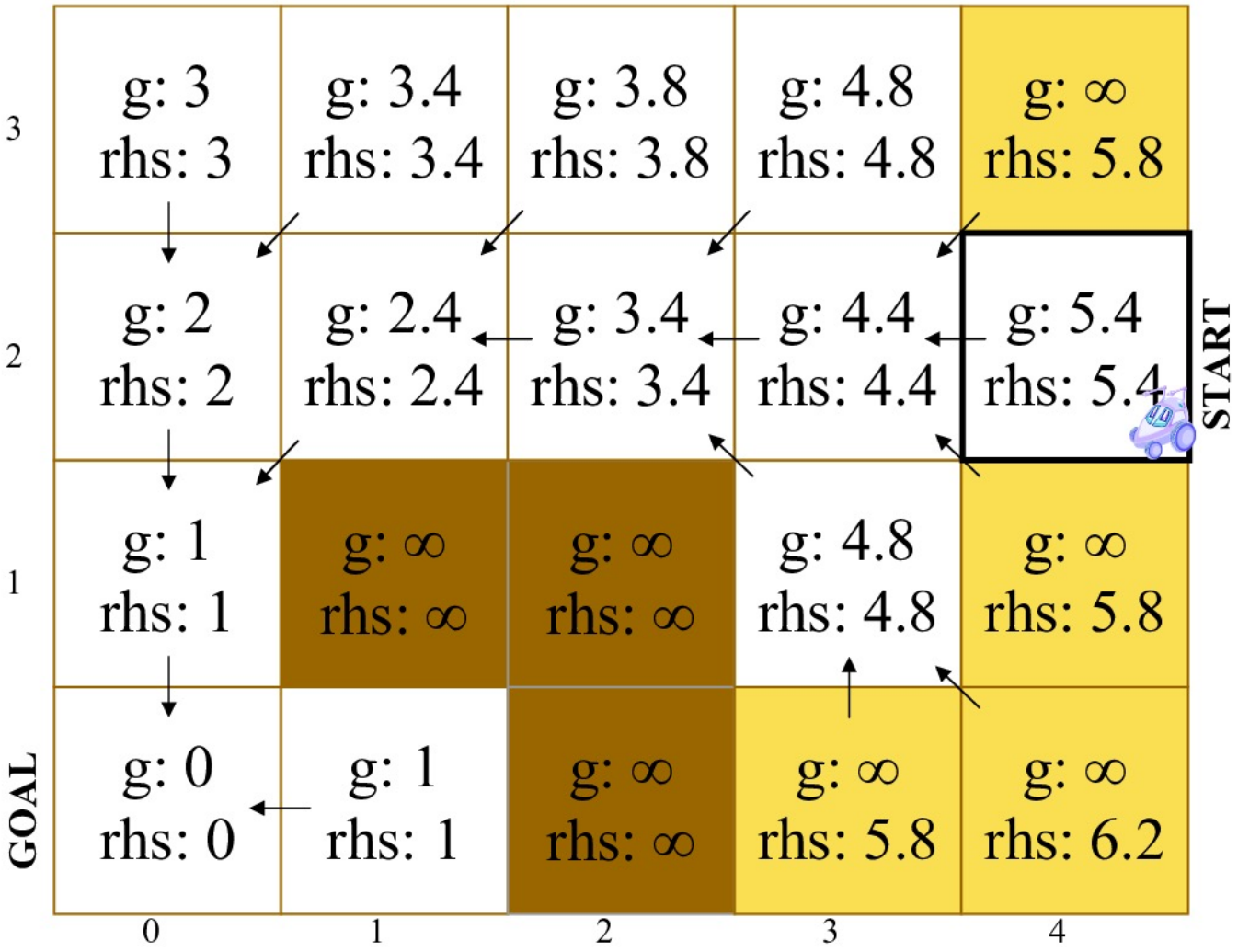


ComputeShortestPath
Expand the popped node i.e, call *UpdateVertex()* on all its predecessors in the graph. This computes *rhs* values for the predecessors and puts them on the open list if they become inconsistent.

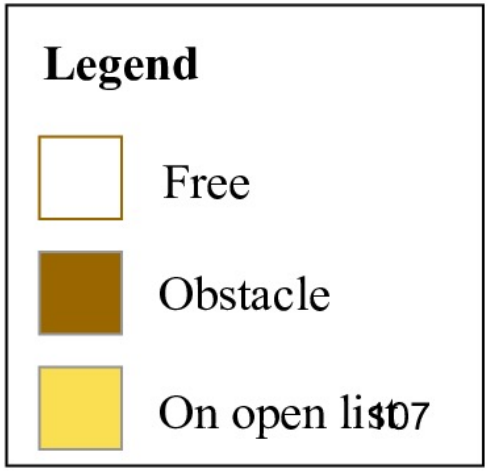
Legend

- Free
- Obstacle
- On open list

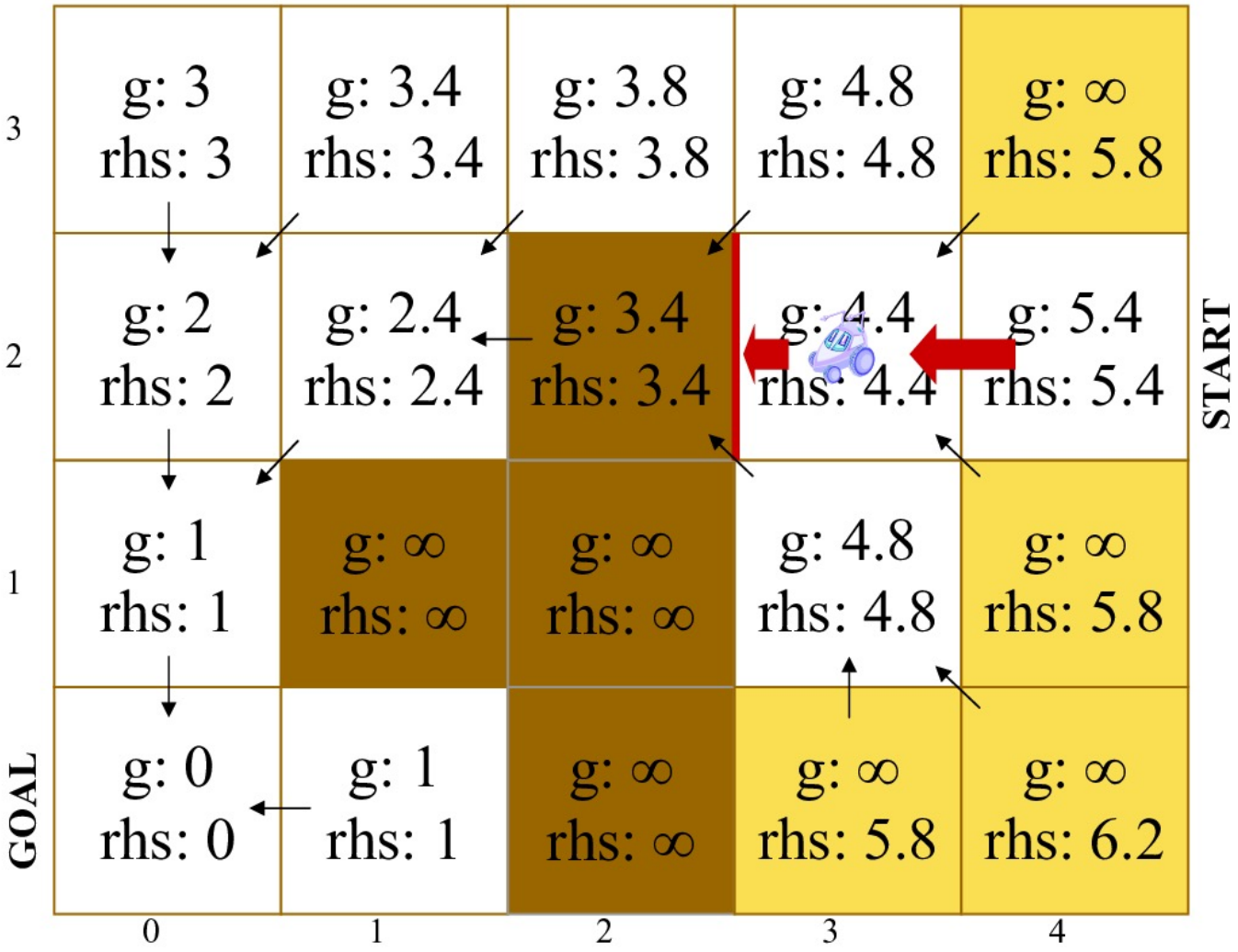
Replanification en environnement dynamique: D* Lite



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

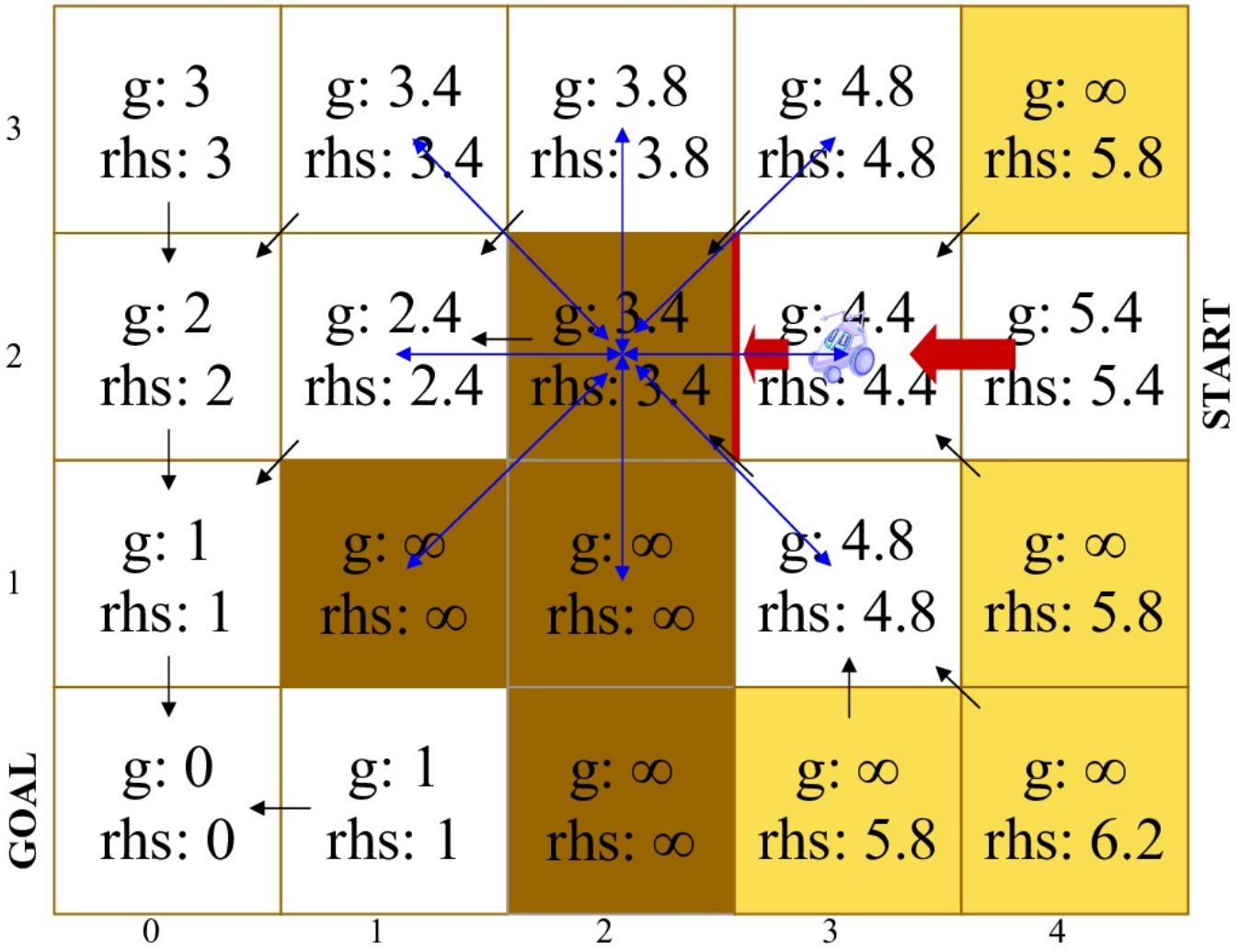


Suppose that, in trying to move from (3,2) to (2,2), the robot discovers that (2,2) is actually an obstacle. Now, **replanning** is required!

Legend

- Free
- Obstacle
- On open list

Replanification en environnement dynamique: D* Lite

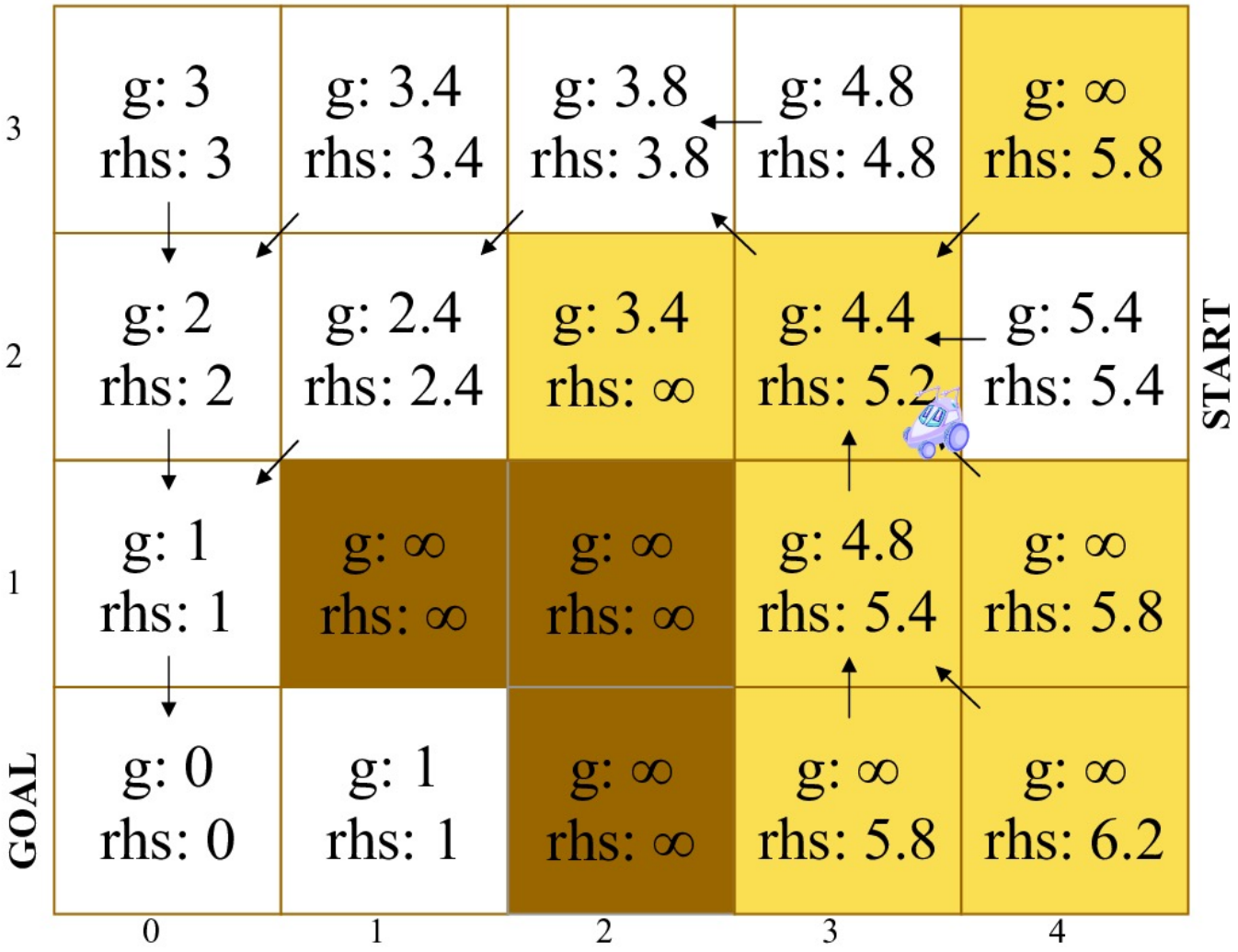


The algorithm dictates that for all directed edges (u, v) with changed edge costs, we should call *UpdateVertex(u)*. Since the edges in our graph are bidirectional and all edges into or out of (2,2) are affected, we need to consider 16 different edges!

Legend

- Free
- Obstacle
- On open list

Replanification en environnement dynamique: D* Lite

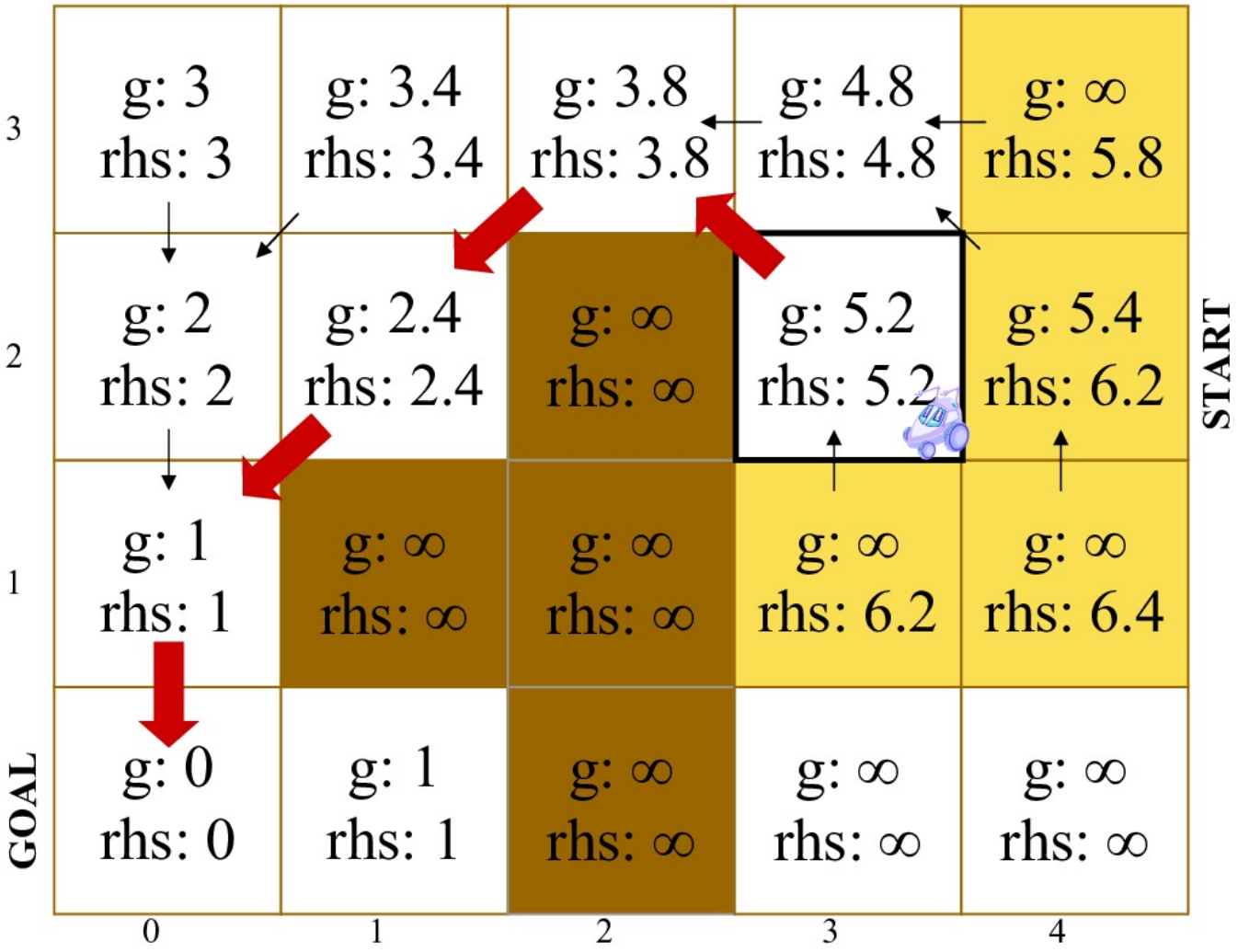


Now, we go back to calling *CompteShortestPath()* until we have an optimal path. The node corresponding to the robot's current position is inconsistent and its key is greater than the minimum key on the open list, so we know that we do not yet have an optimal path..

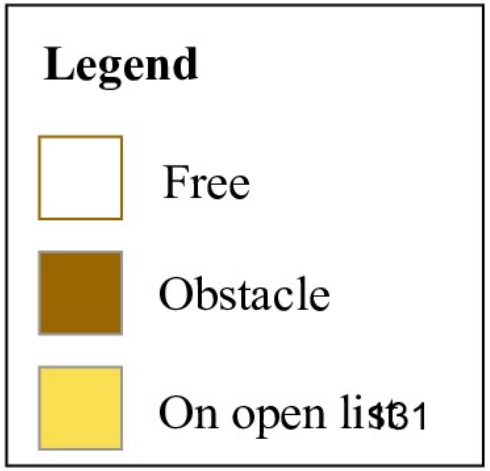
Legend

- Free
- Obstacle
- On open list

Replanification en environnement dynamique: D* Lite

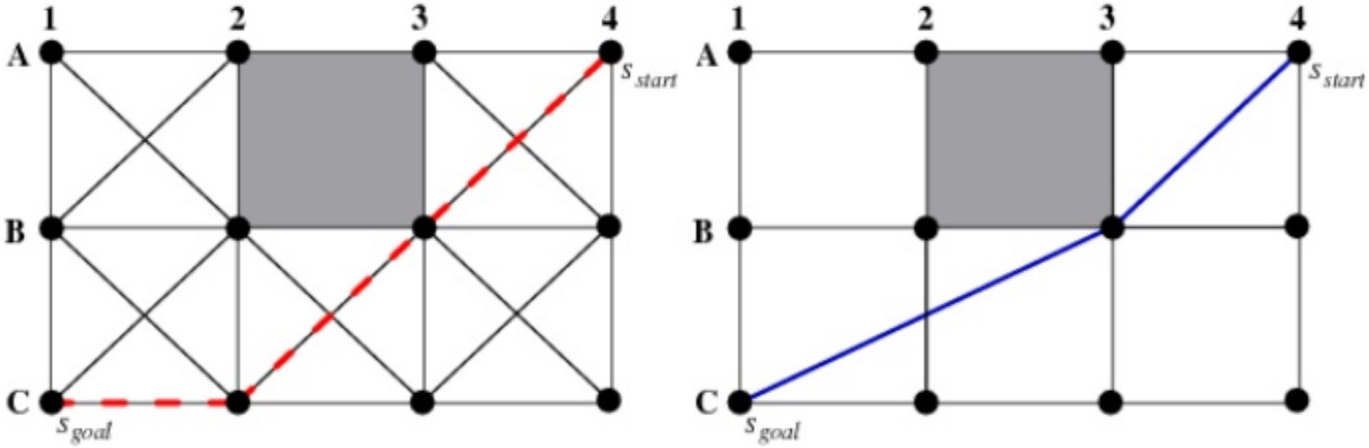


Follow the gradient of g values from the robot's current position.



Nombreuses variantes

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



Theta* : A* + visibility graph

Couramment utilisés en pratique

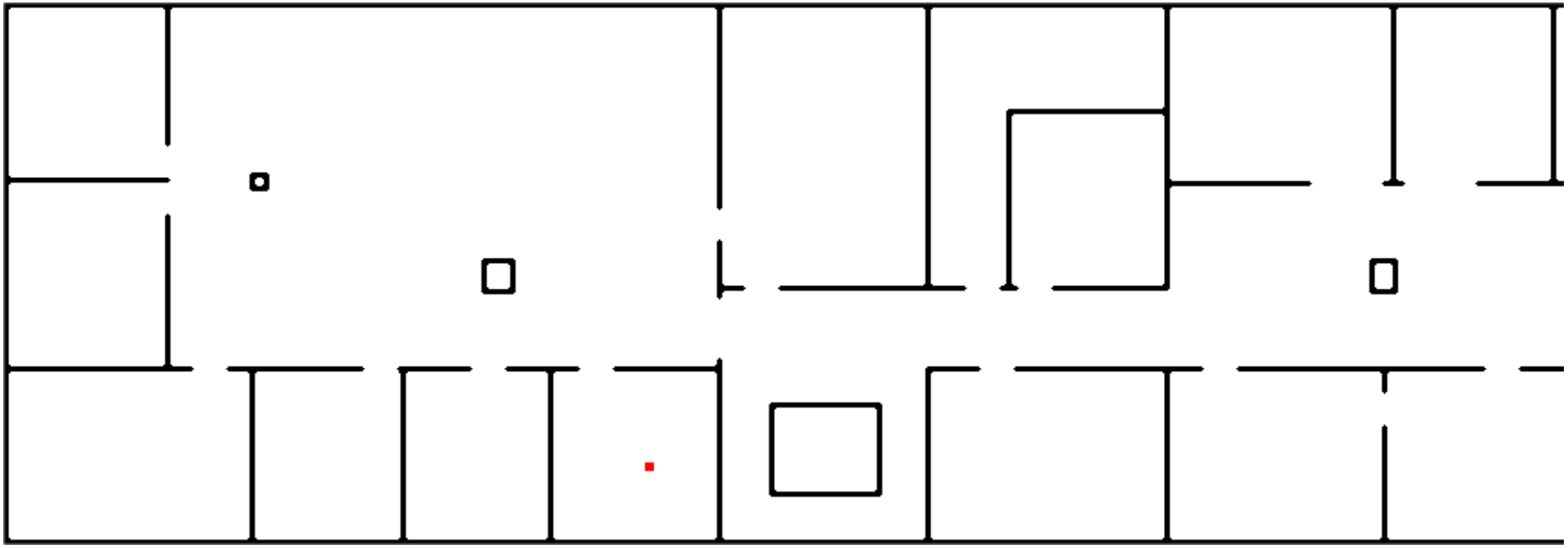
- Véhicules autonomes
- Rovers sur Mars
- Robots de service

Calcul d'une politique

Au lieu de calculer une action pour chaque cellule :

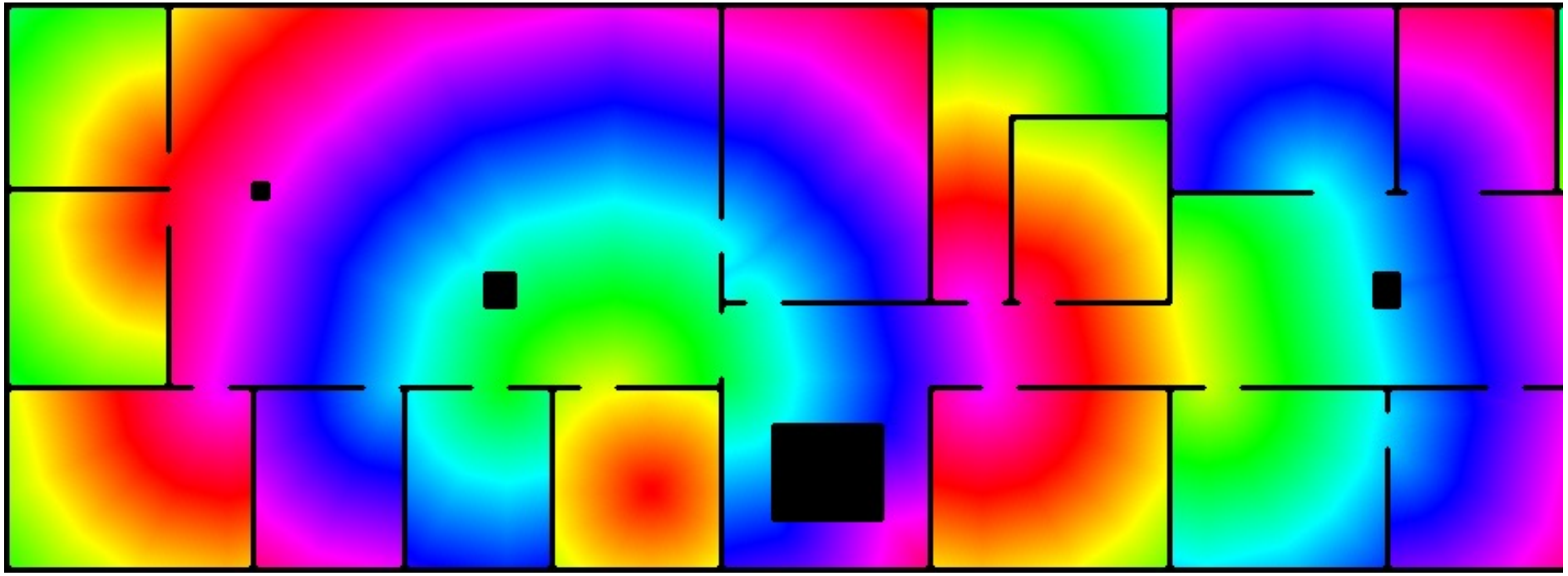
- calcul d'un poids pour chaque cellule (distance au but)
- exécution par descente de gradient

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



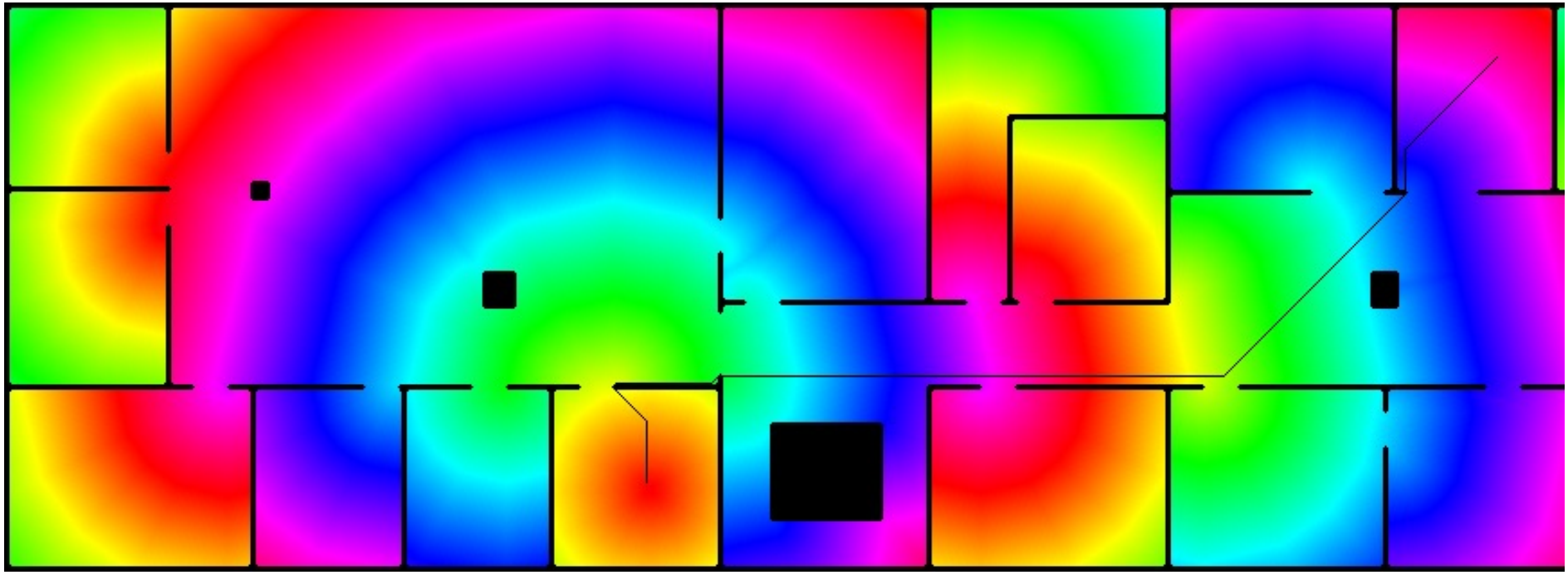
Exemple 1

– Poids des liens = distance euclidienne entre centre des cellules



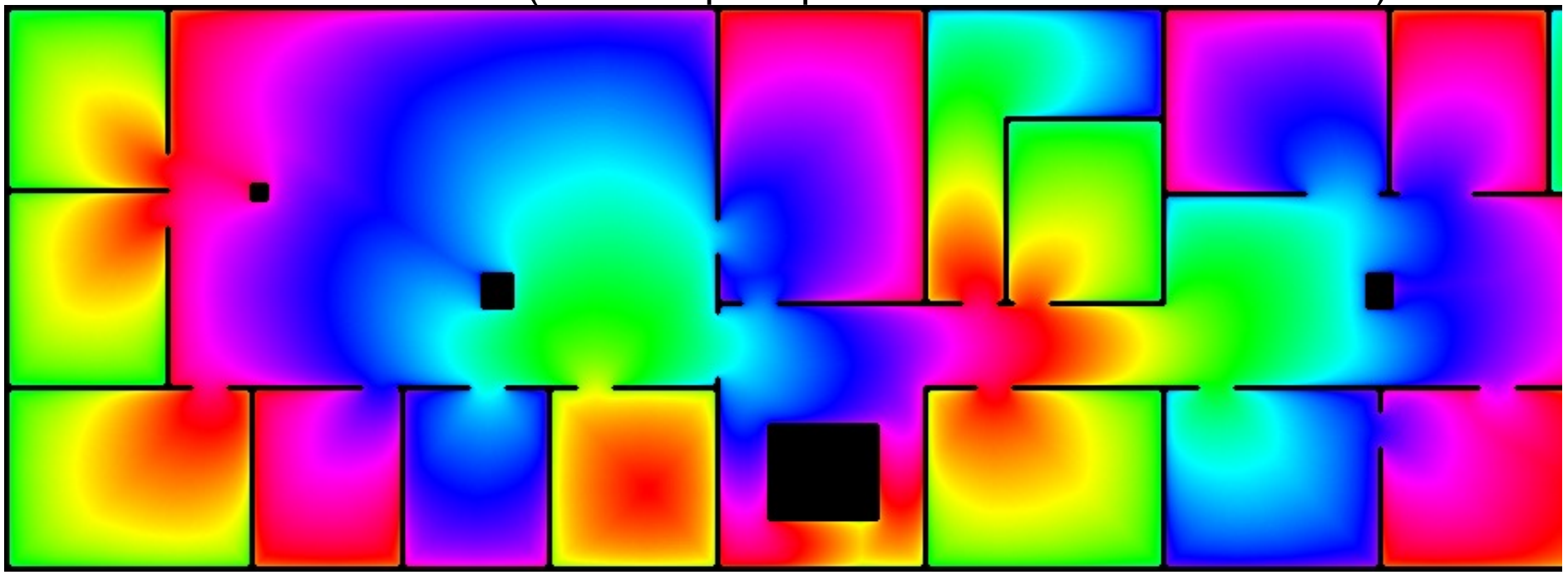
Limitations

- Trajectoires proches des obstacles
- Peut être dangereux pour des robots avec un contrôle peu précis



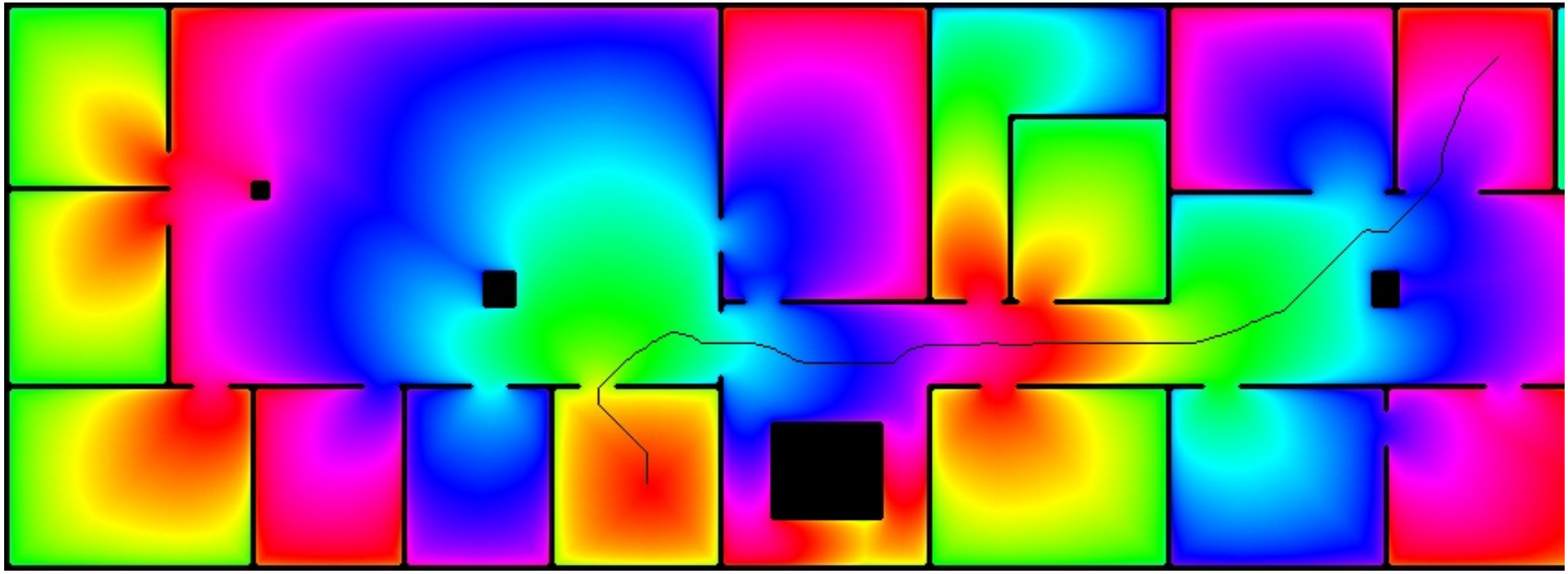
Exemple 2

- Poids des liens = distance euclidienne entre centre des cellules
- Poids associés aux nœuds : inverse de la distance aux obstacles
  (calcul rapide par Transformée en Distance)



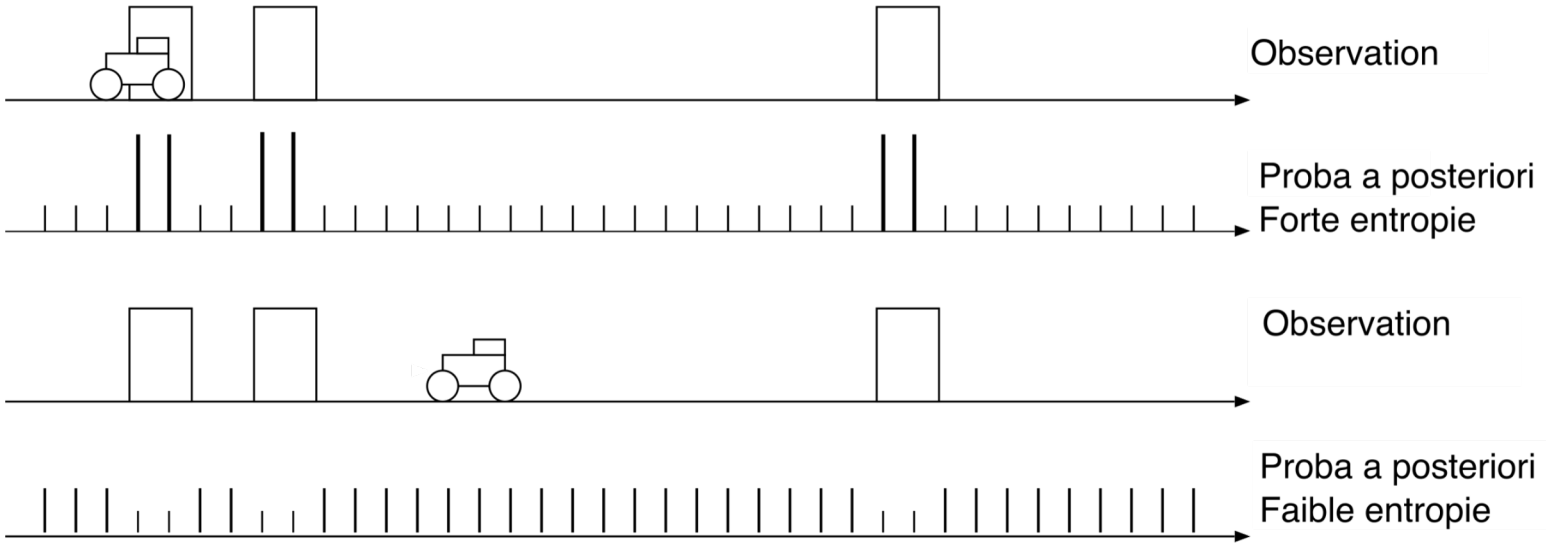
Limitations

- Chemins ne sont pas les plus courts
- Mais trajectoires moins proches des obstacles



« Coastal Navigation »

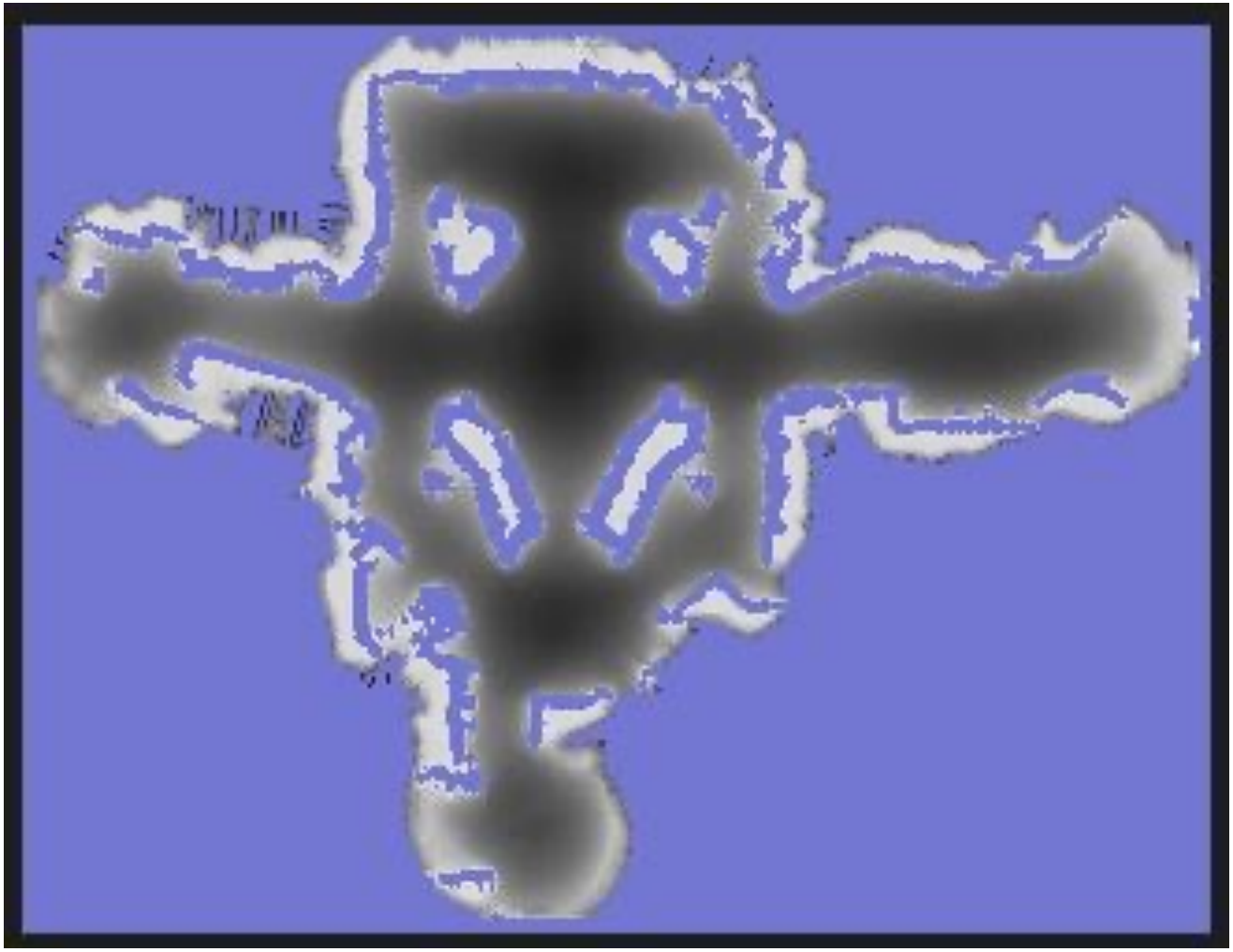
- Poids des nœuds = information disponible pour la localisation
- Calculé par la variation d'entropie après intégration des données laser pour chaque position (MCL)
- Calcul d'un chemin pour lequel la localisation est la plus fiable possible



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

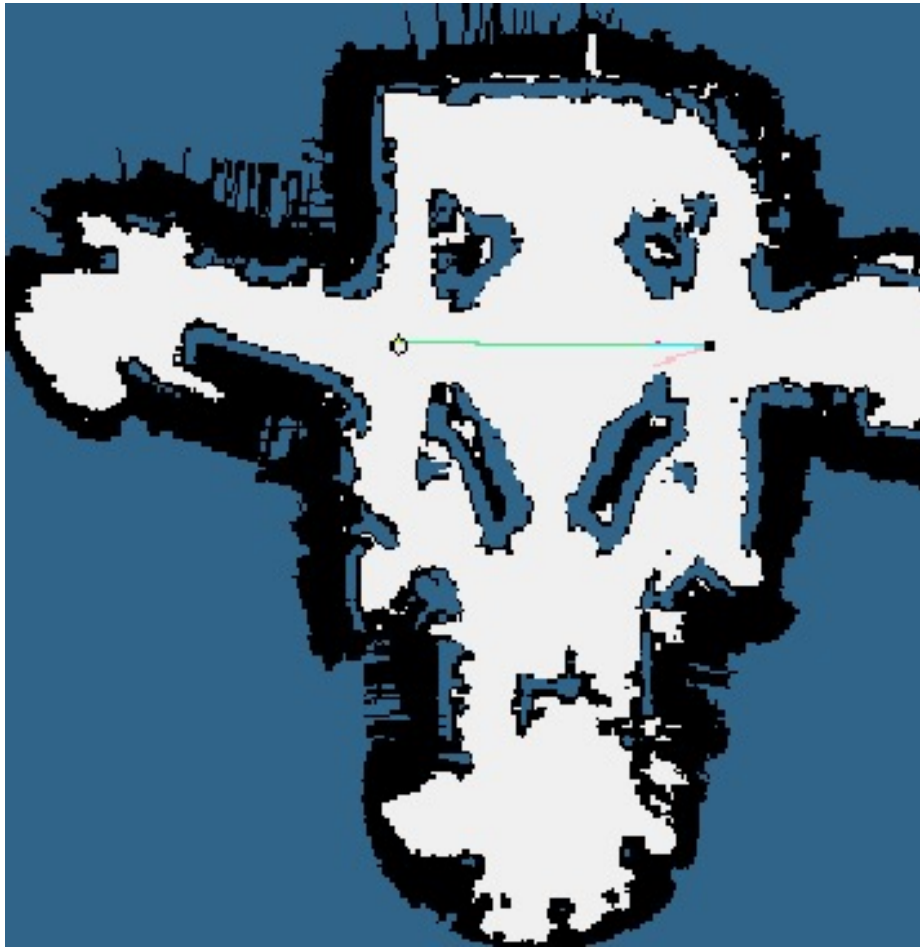
Poids des nœuds :

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

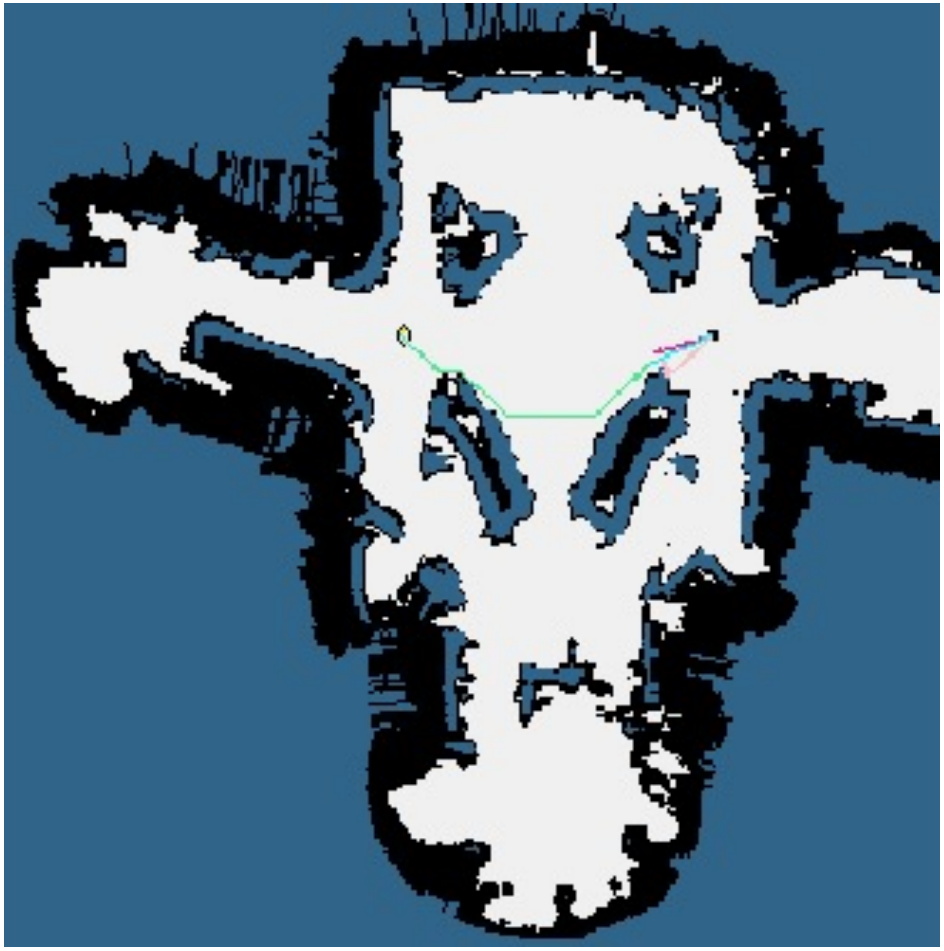


Exemple de calcul de politiques

Chemin le plus court



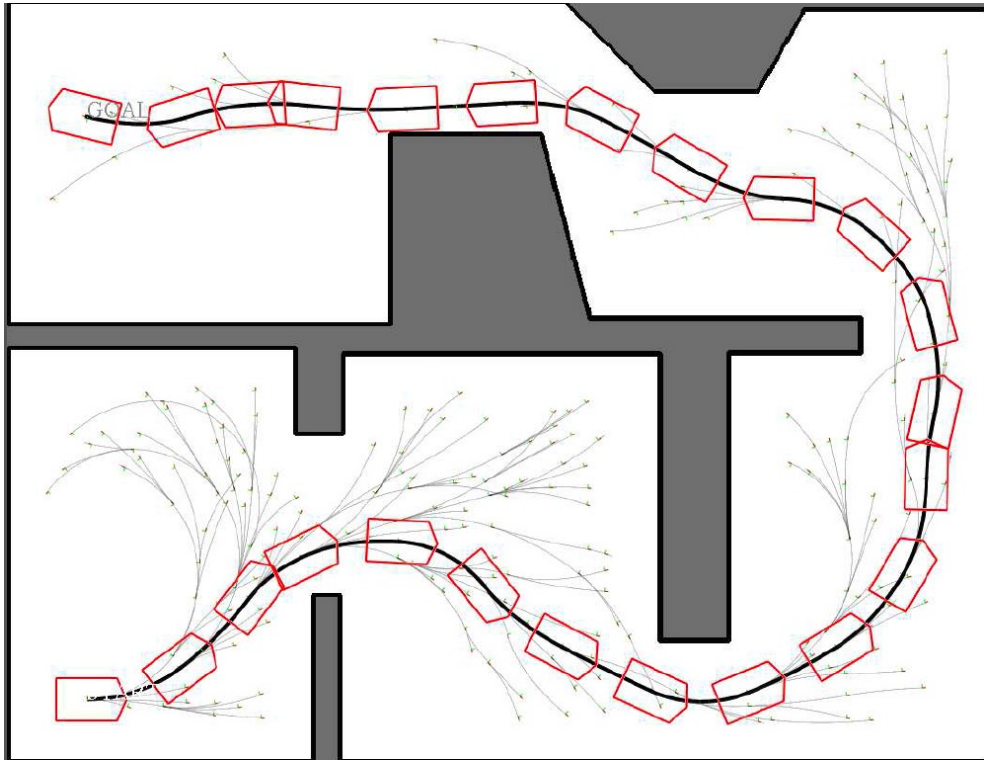
Chemin le plus « sur »



Recherche de chemin stochastique

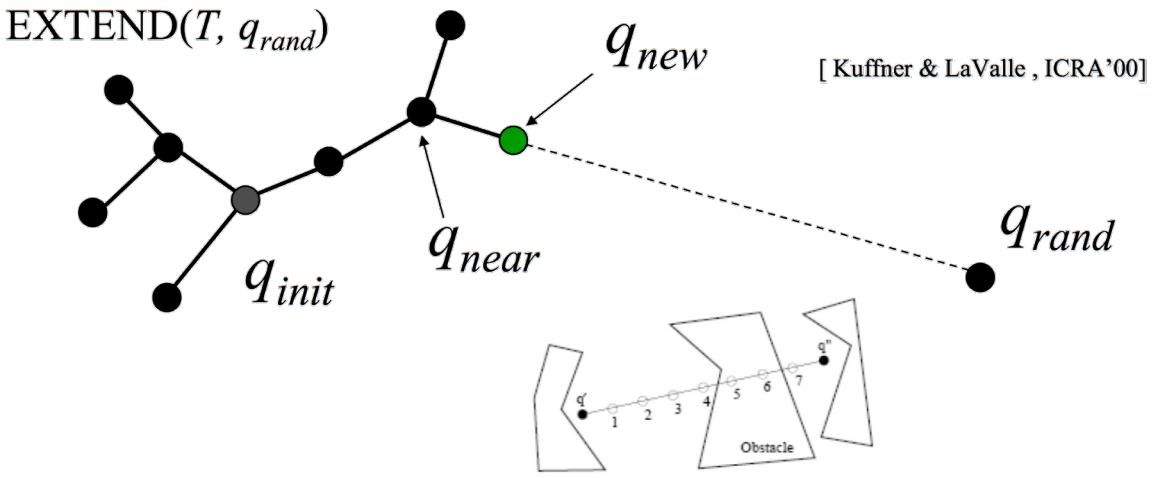
Rapidly exploring Random Trees

- Famille des méthodes de discrétisation en chemins aléatoires
- Mais ne crée par un graphe complet
- Cherche seulement un chemin pour le but courant
- Algorithm '**Single Query**' Vs '**Multiple Query**' pour PRM
- Construction incrémentale plutôt que globale



Rapidly exploring Random Trees (RRT, 99)

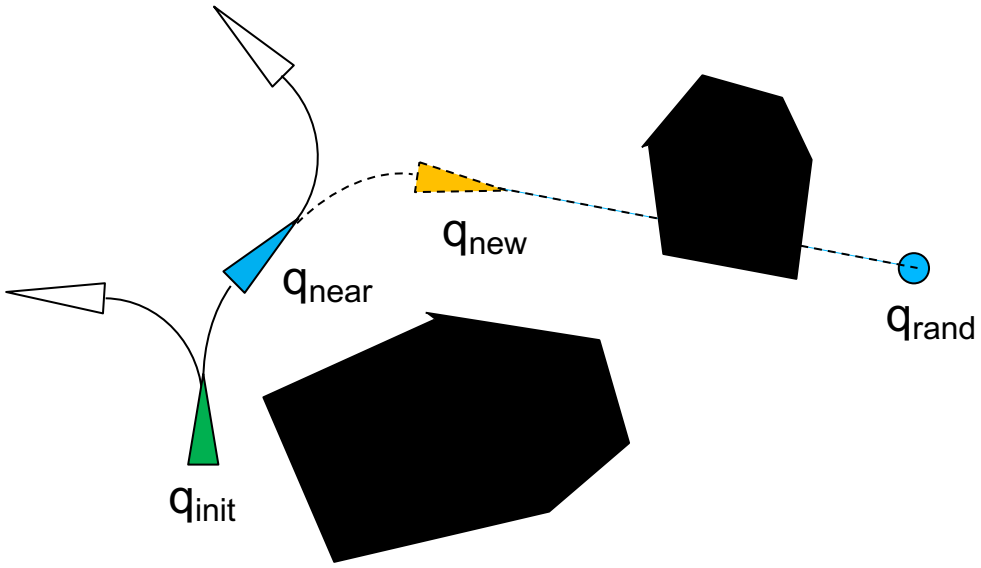
- Construction d'un arbre à partir de la position initiale
- Choisir un point q_{rand} aléatoirement
- Trouver le plus proche voisin dans l'arbre courant
- Etendre l'arbre d'une distance fixe depuis ce point
  - Utilise un planificateur local
  - Arrêt en cas d'obstacles
- De temps en temps, utiliser q_{rand} = but



Example local planner

Rapidly exploring Random Trees (RRT, 99)

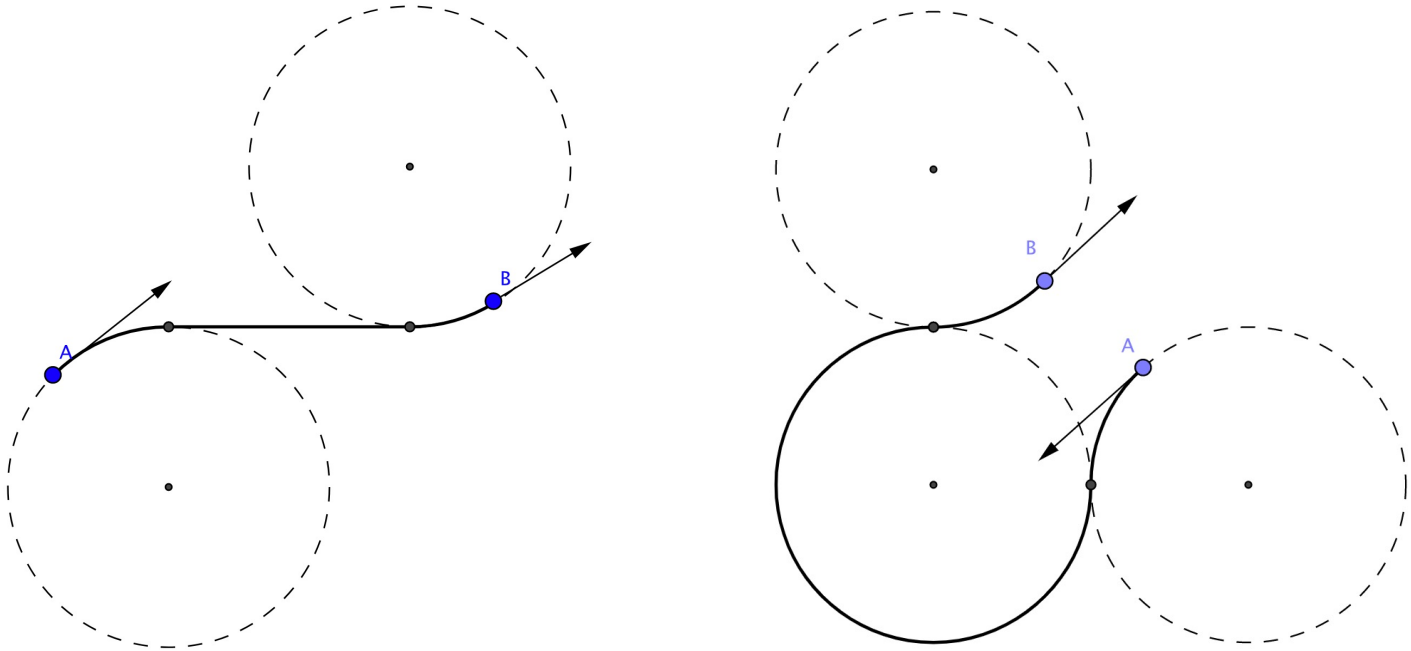
- Possibilité de planifier avec des modèles non-holonomes quelconques
- Utilisation du modèle dans le planificateur local



- Attention au calcul du ‘point le plus proche’ (fusion distance/angle)

Rapidly exploring Random Trees (RRT, 99)

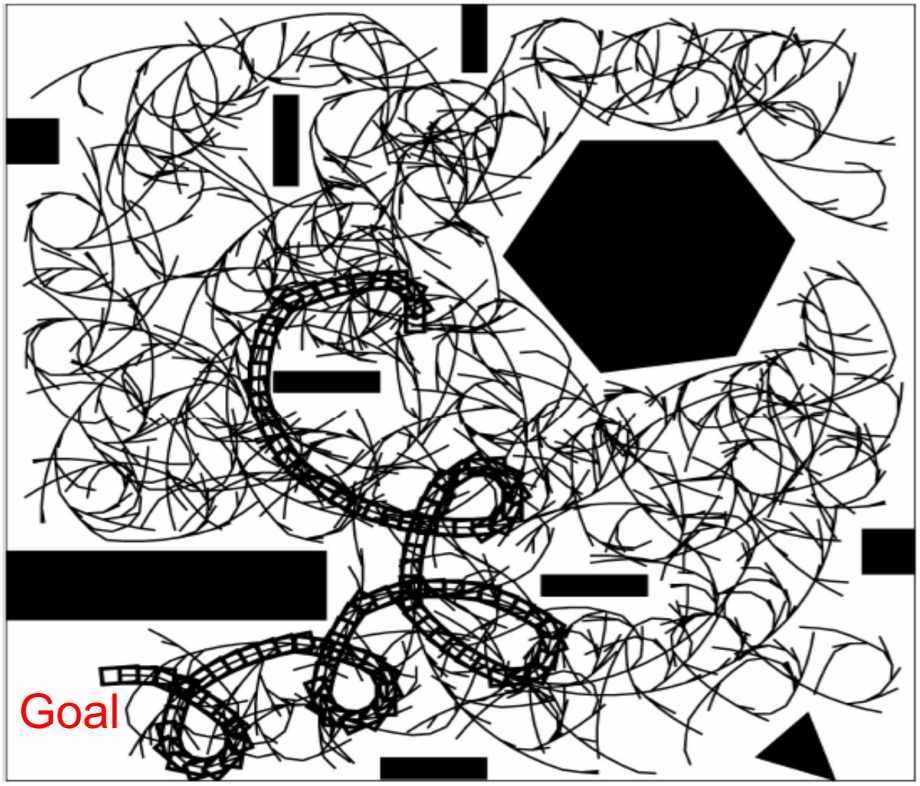
- Pour un modèle bicycle possibilité d'utiliser les courbes de dubbins
- Donnent le plus court chemin entre 2 points orientés
- Calcul géométrique ou analytique rapide



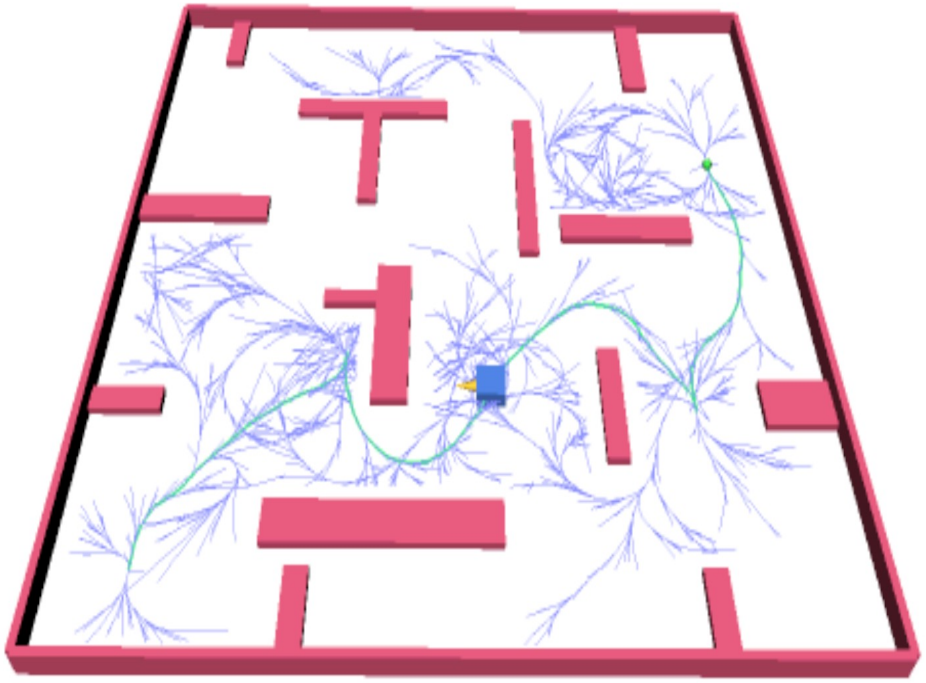
- Généralisation avec marche arrière : courbe de Reeds-Shepp

Rapidly exploring Random Trees (RRT, 99)

– Exemples non holonomes :



Voiture tournant à gauche

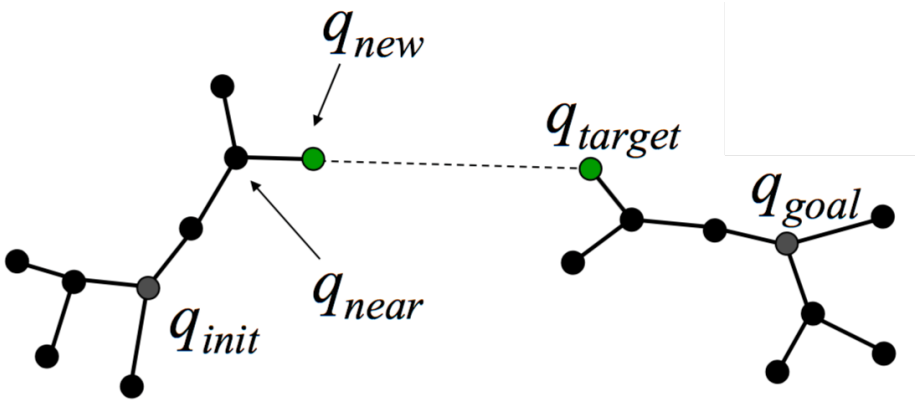


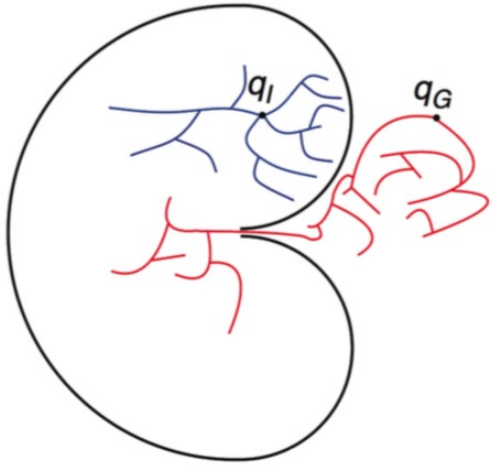
Hovercraft

– Mais plans proposés loin de l'optimal

Rapidly exploring Random Trees (RRT, 99)

- Variante Dual RRT (RRT connect) : construire 2 arbres depuis le départ et le but
- Tenter de connecter q_{new} à l'autre arbre à chaque extension





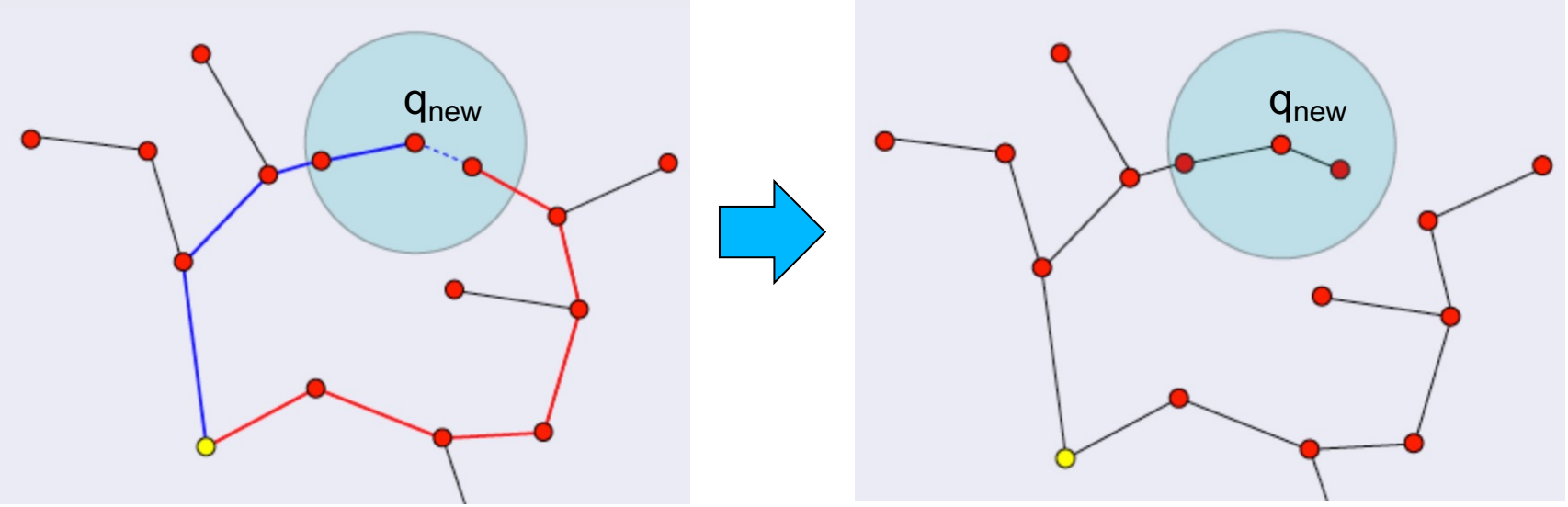
- Permet de planifier dans des environnements plus encombrés
- Nombreuses autres extensions
  - Real time
  - Anytime
  - Dynamic environments

RRT*

- RRT trouve un chemin si il existe, mais pas l'optimal
- RRT* : amélioration de la mise à jour en restructurant l'arbre
- Permet de trouver les chemins optimaux

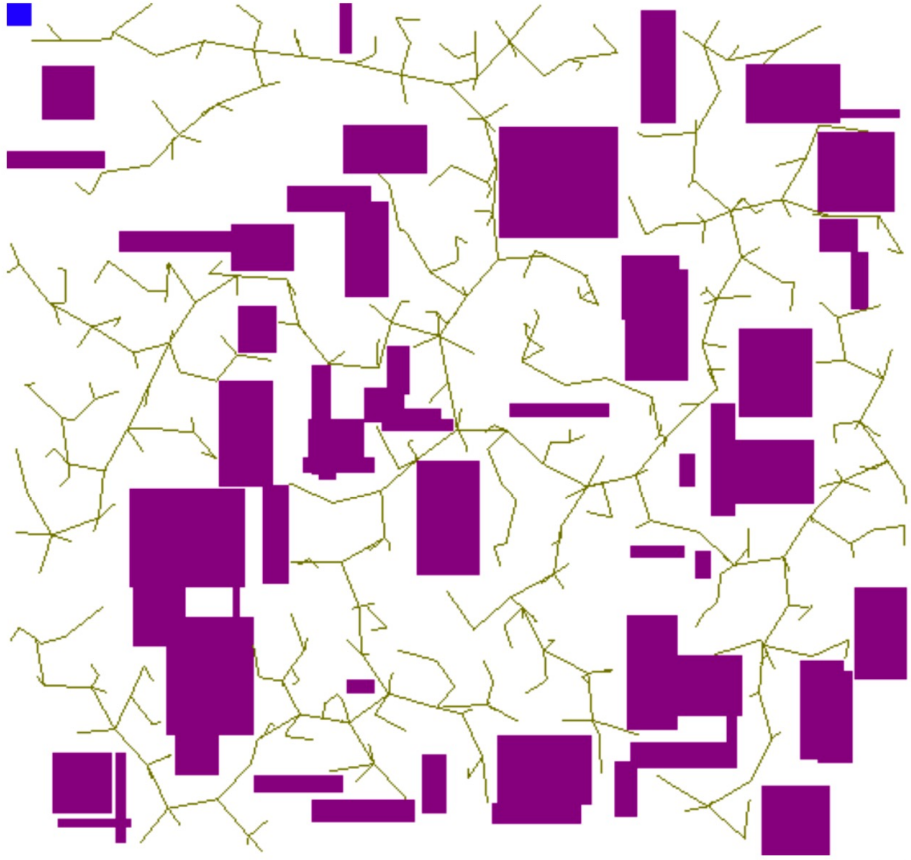
Mise a jour RRT*

- Sélectionner tous les nœuds autour de q_{new}
- Reconnecter les nœuds aux voisins permettant des chemins plus courts



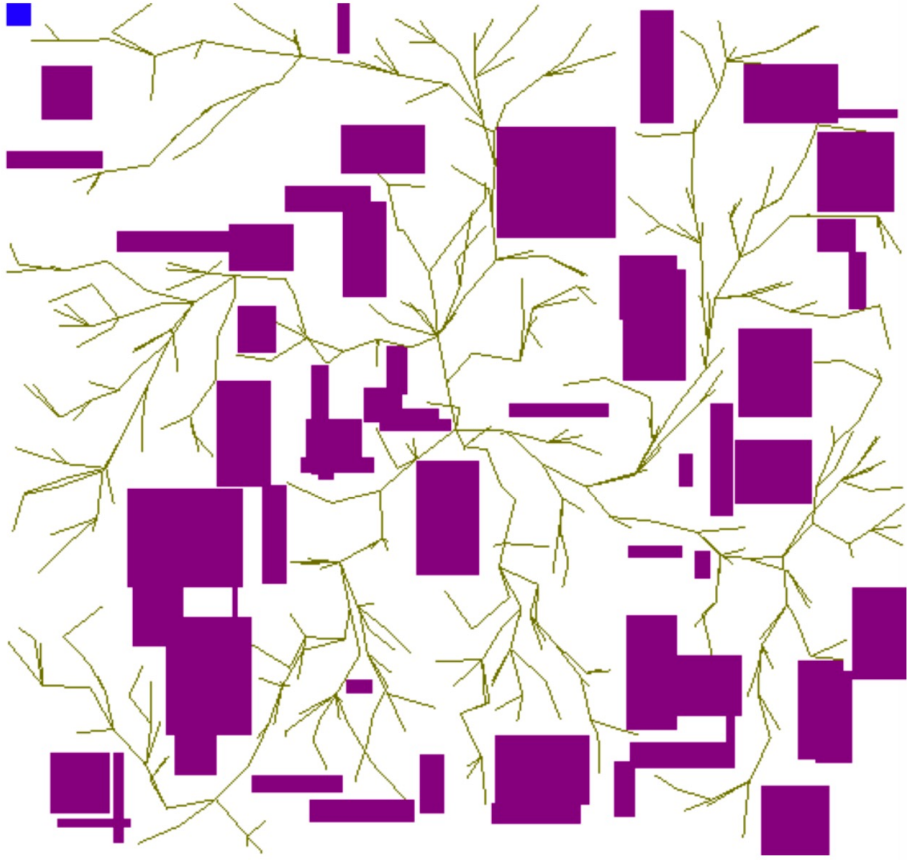
Comparaison RRT vs RRT*

– Tiré de « S. Karaman, E. Frazzoli, Sampling-based Algorithms for Optimal Motion Planning. »



RRT

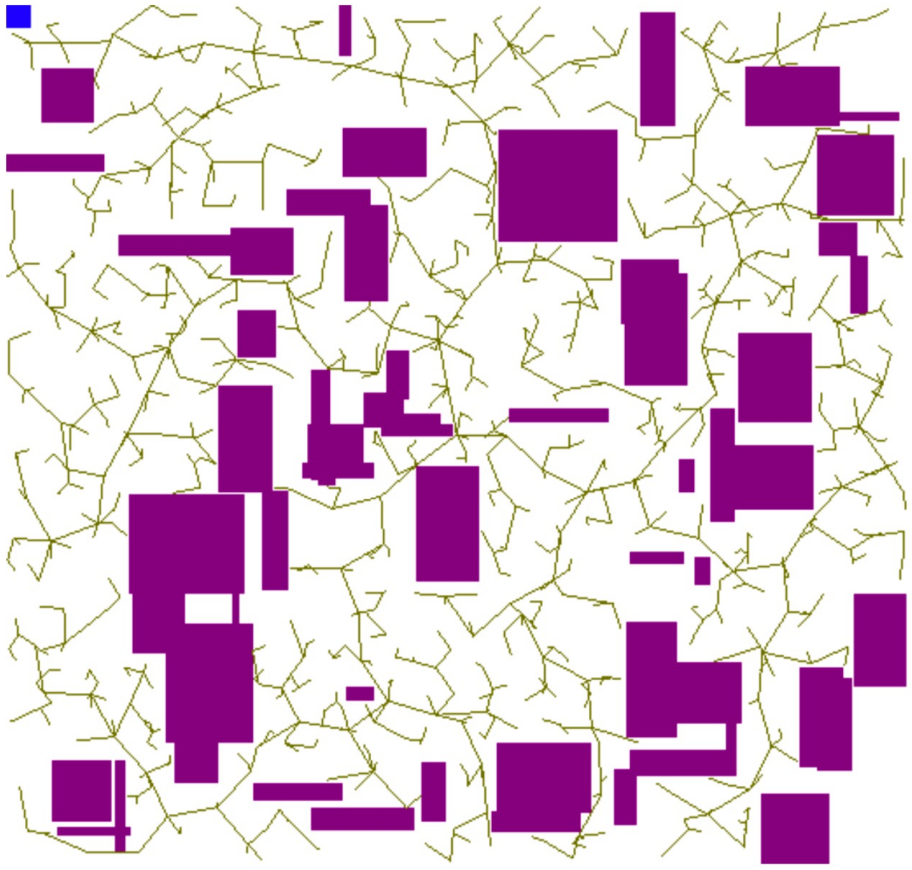
503 iterations



RRT*

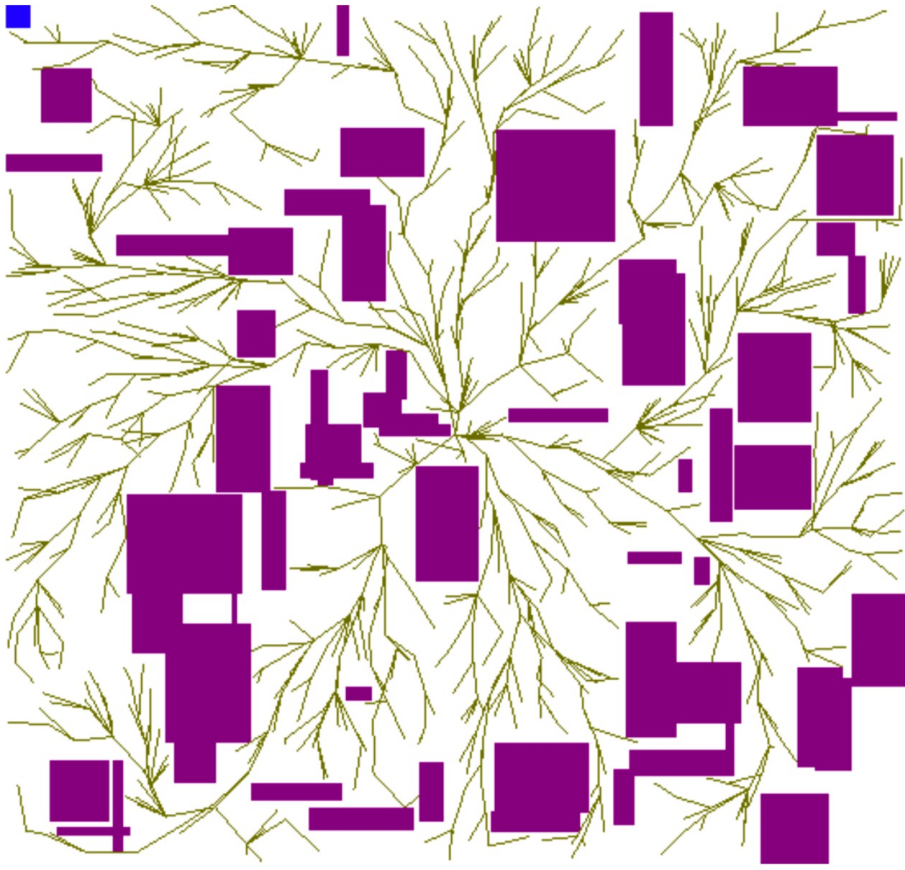
Comparaison RRT vs RRT*

– Tiré de « S. Karaman, E. Frazzoli, Sampling-based Algorithms for Optimal Motion Planning. »



RRT

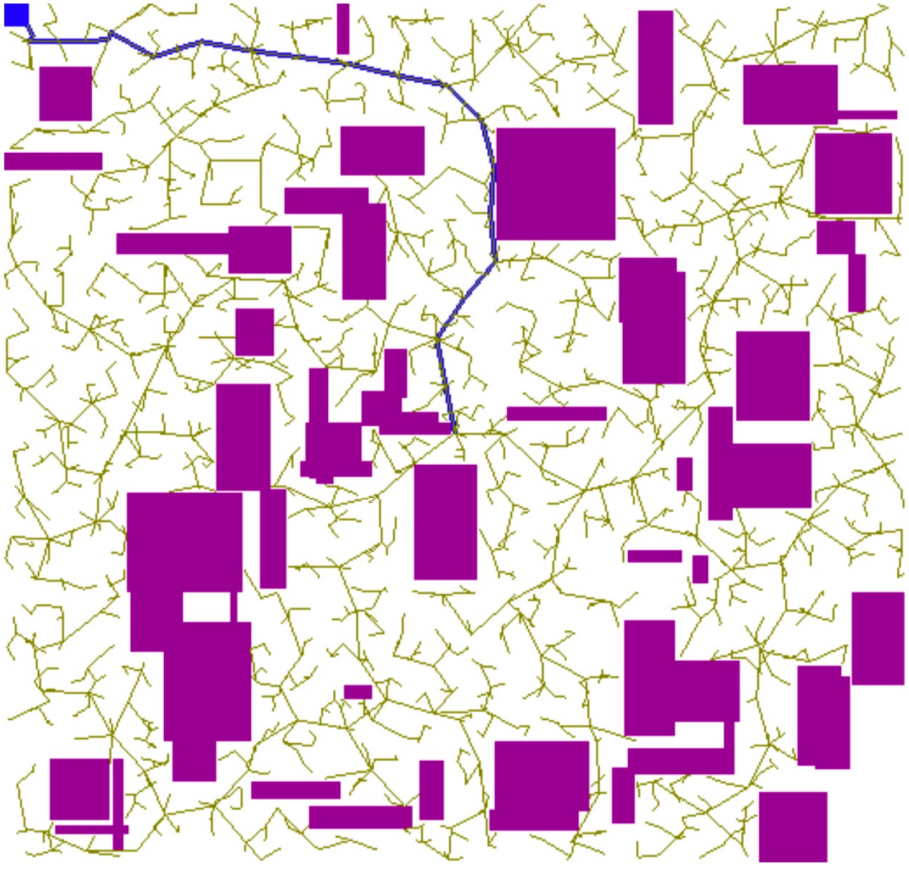
1027



RRT*

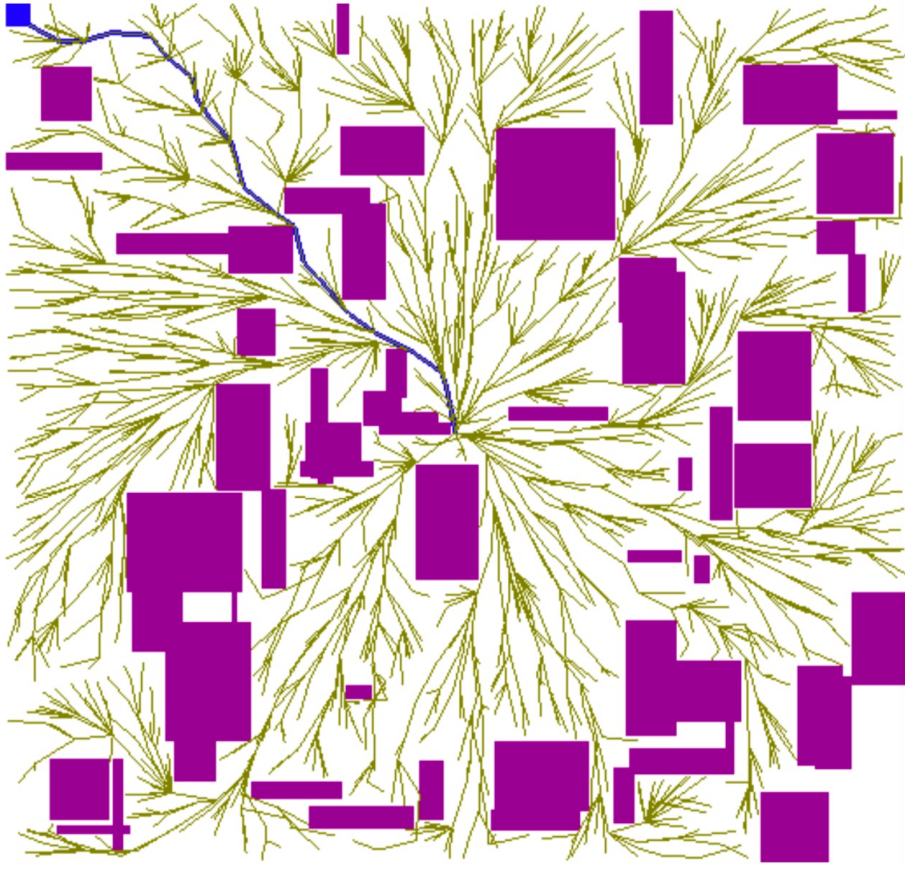
Comparaison RRT vs RRT*

– Tiré de « S. Karaman, E. Frazzoli, Sampling-based Algorithms for Optimal Motion Planning. »



RRT

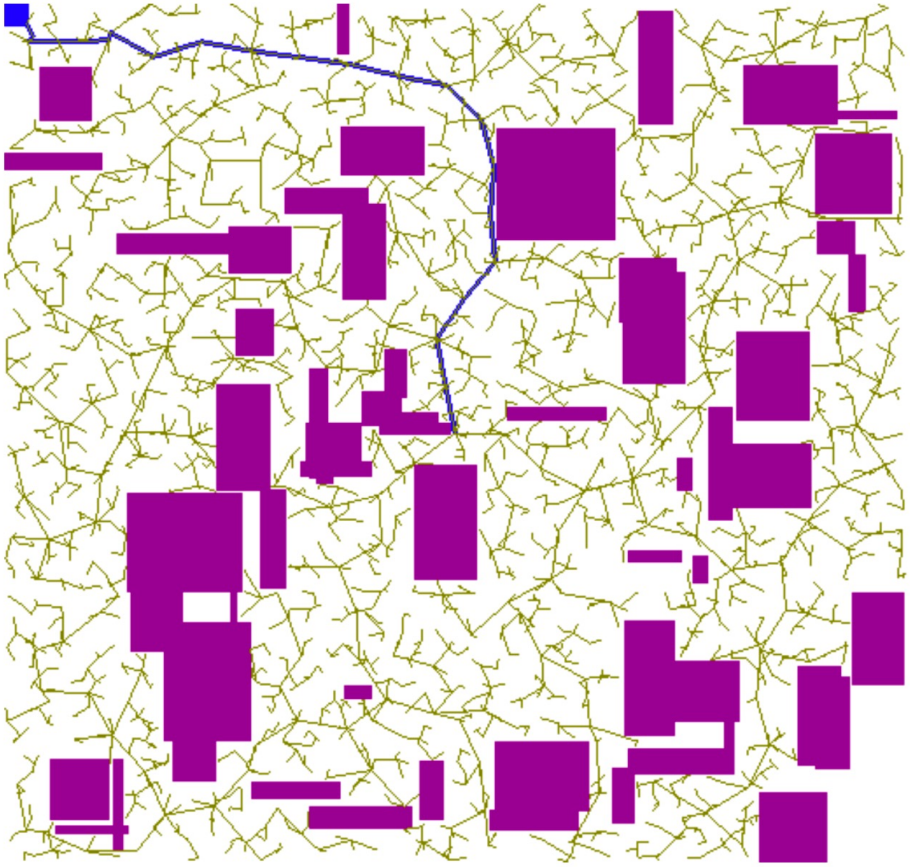
2062



RRT*

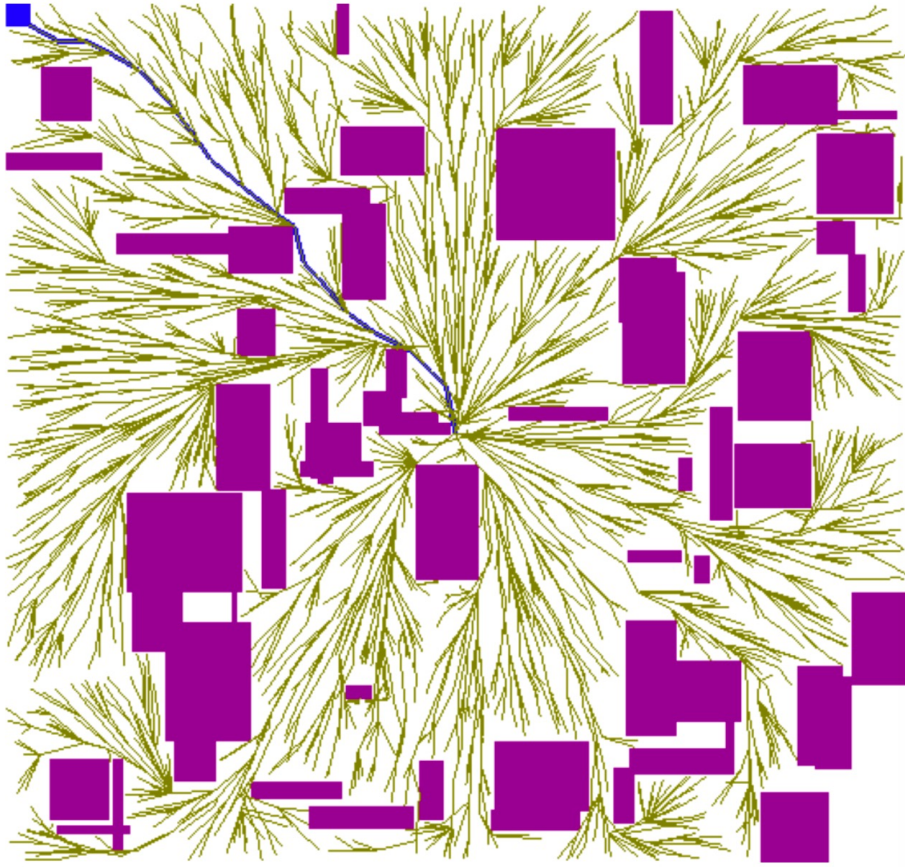
Comparaison RRT vs RRT*

– Tiré de « S. Karaman, E. Frazzoli, Sampling-based Algorithms for Optimal Motion Planning. »



RRT

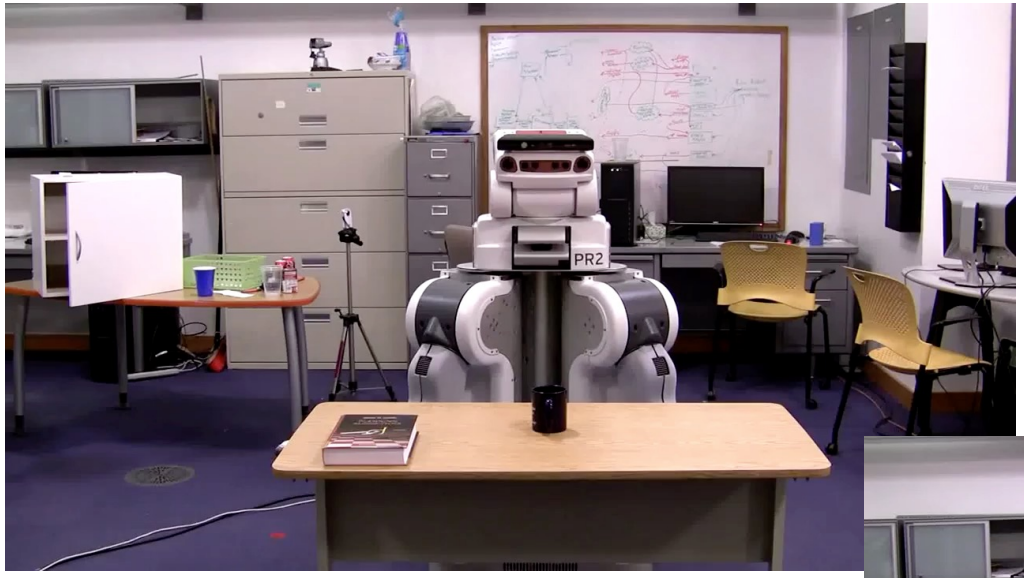
3037



RRT*

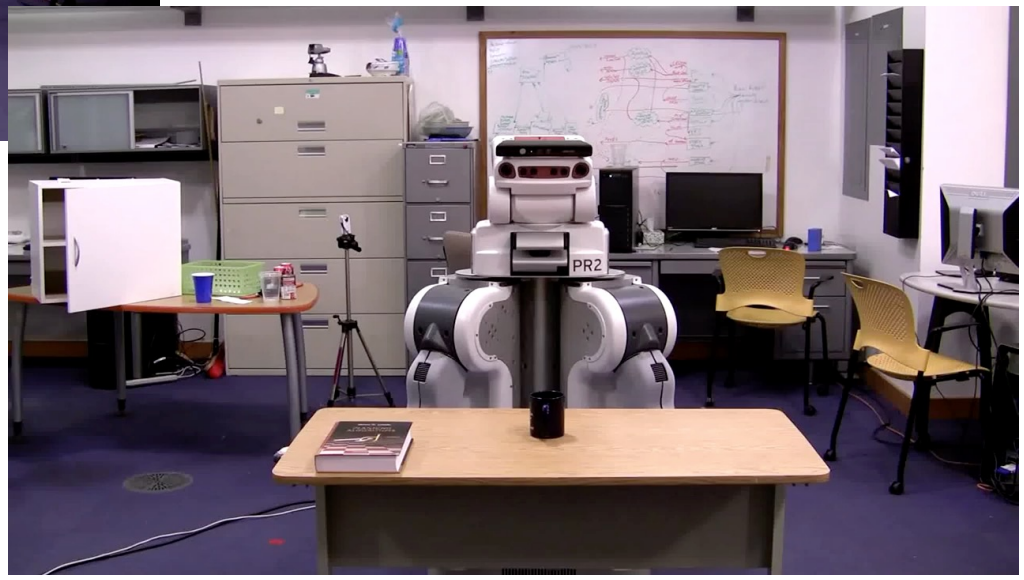
Comparaison RRT vs RRT*

– Effet de la sous-optimalité en dimension 12 !



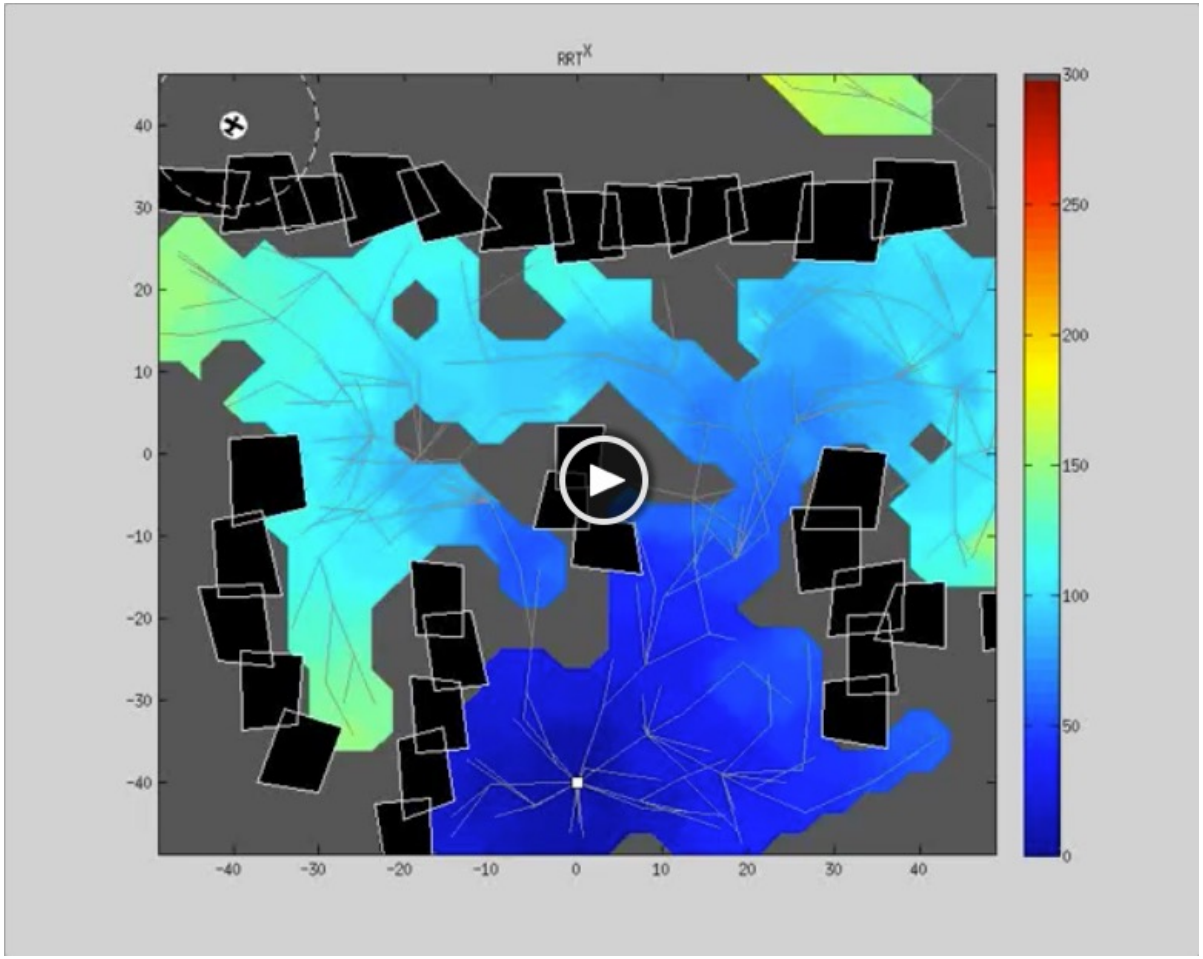
RRT

RRT*



RRT^X

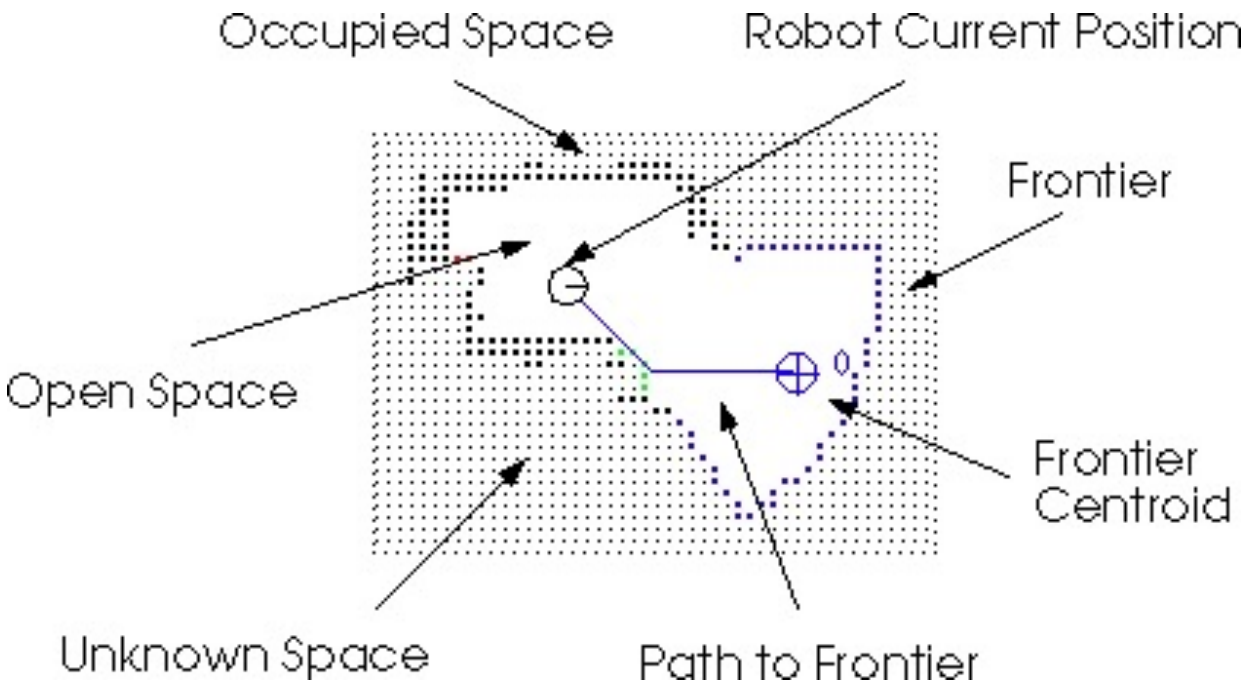
- Amélioration de RRT* pour les env. dynamiques/inconnus
- Reconnexion/amélioration permanente en fct de l'env. (cf D*)



Exploration

Exploration d'un environnement inconnu

- B. Yamauchi, « A Frontier-Based Approach for Autonomous Exploration », CIRA 97
- Planification à partir de grilles d'occupations
- Frontière : cellules « vides » touchant des cellules « inconnues »



Planification pour exploration

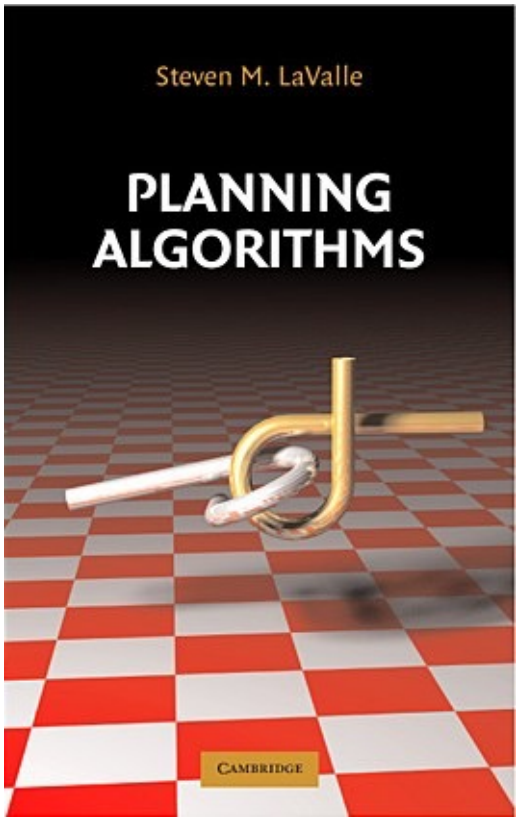
- Planification vers centre de la frontière la plus proche avec A*
- Extensions multi-robots possibles (attribution des frontières)

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

Planification de trajectoire - En résumé

- Il est possible de « planifier » sans carte, de manière réactive
- La planification nécessite de **discrétiser l'espace**, soit en cellules, soit en chemin, de manière déterministe ou aléatoire
- Une fois l'espace discrétisé, de nombreuses méthodes permettent de trouver le chemin le plus court, notamment **A*** qui utilise une heuristique pour accélérer la recherche et ses variantes
- L'algorithme **RRT** recherche directement un chemin en construisant un arbre de manière aléatoire, il permet de trouver des chemins complexes, en grand dimensions, mais les chemins ne sont pas optimaux (des variantes telles que RRT* le permettent)
- La planification peut être utilisée pour explorer en allant vers les frontières des zones connues dans la carte