

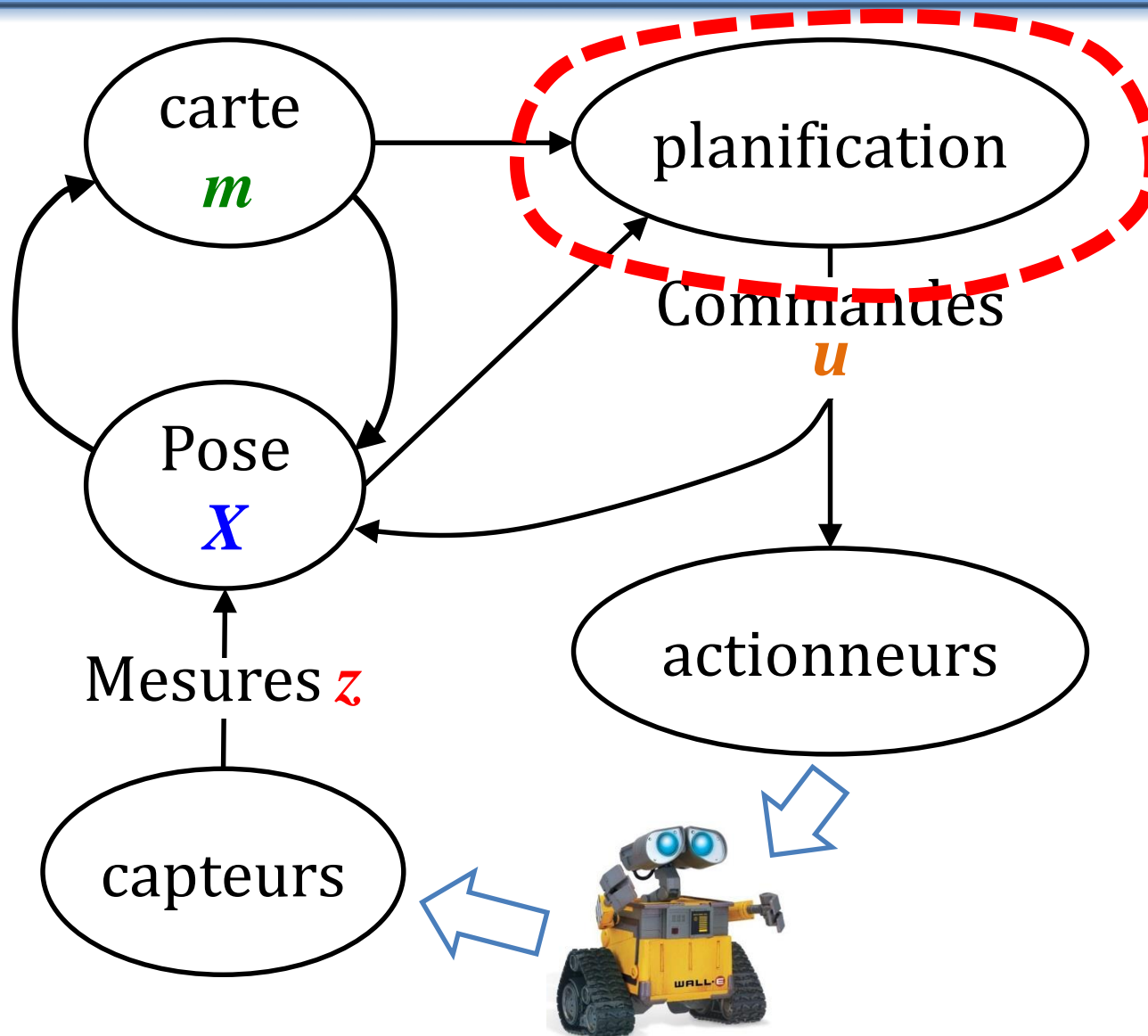


UNIVERSITÉ  
LAVAL

# GLO-4001/7021 INTRODUCTION À LA ROBOTIQUE MOBILE

## Planification

# Feuille de route



# Horizon de temps

---

- Pour un véhicule autonome

## **Long terme**

- Route à prendre entre Québec et Montréal

## **Court terme**

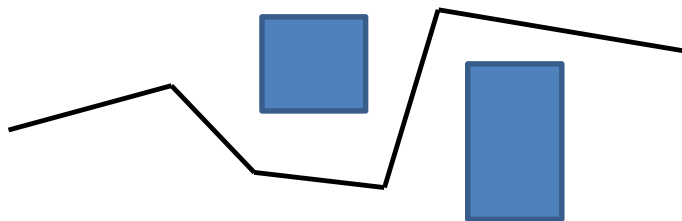
- Virage aux intersection (changement de voie, respect des feux de circulation)

## **Immédiat (*reactive planning*)**

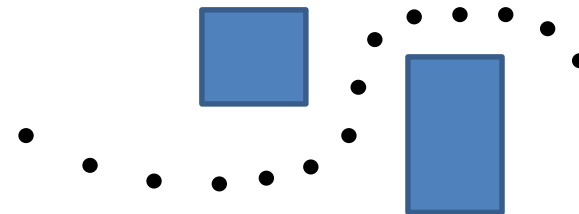
- suivre sa voie
- évitement de véhicule

# Représentation de trajectoire

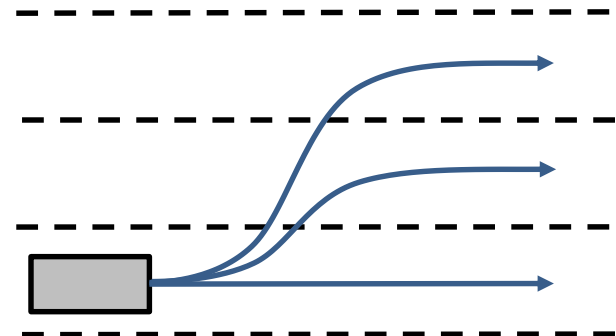
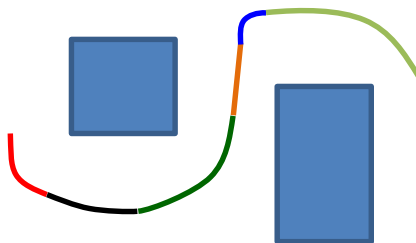
Segments de droites



Points de passage (*waypoint*)

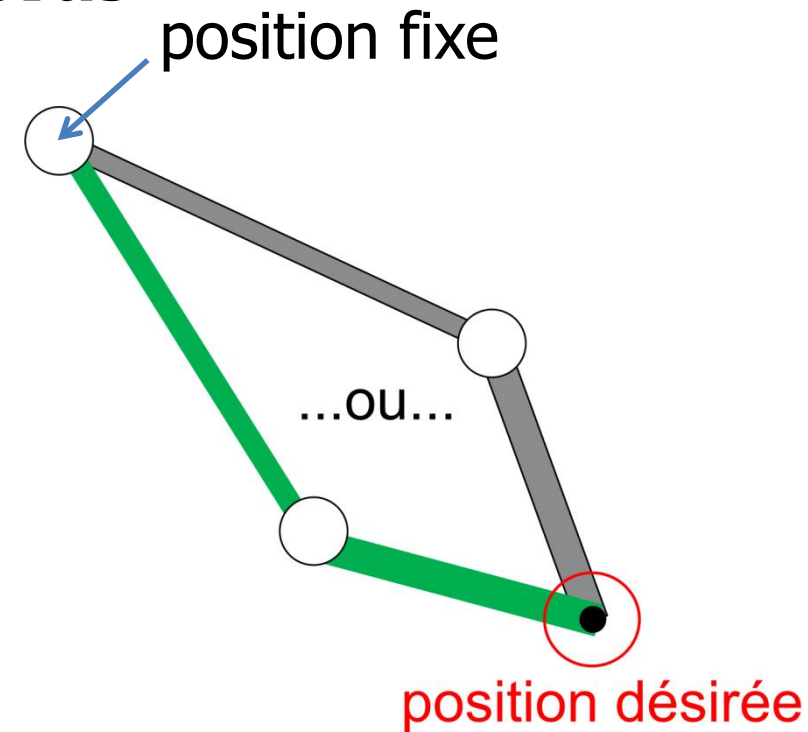
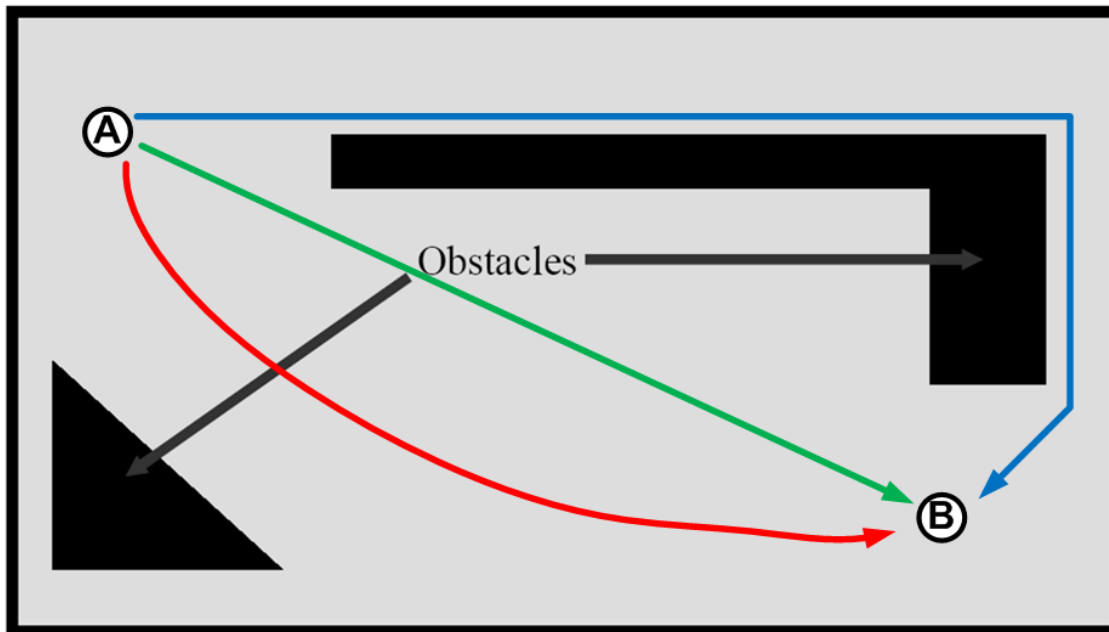


Courbes paramétrisées



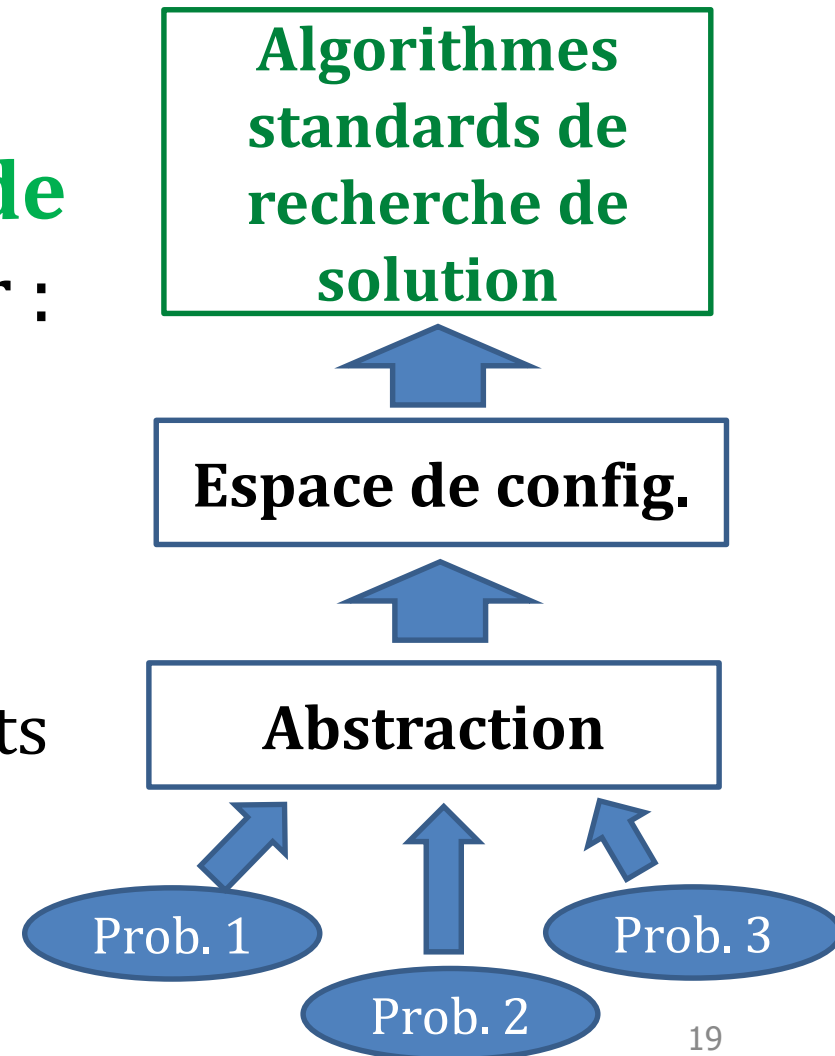
# Planification : problème mal posé

- Déplacer un robot de A vers B
- Positionner l'extrémité d'un bras

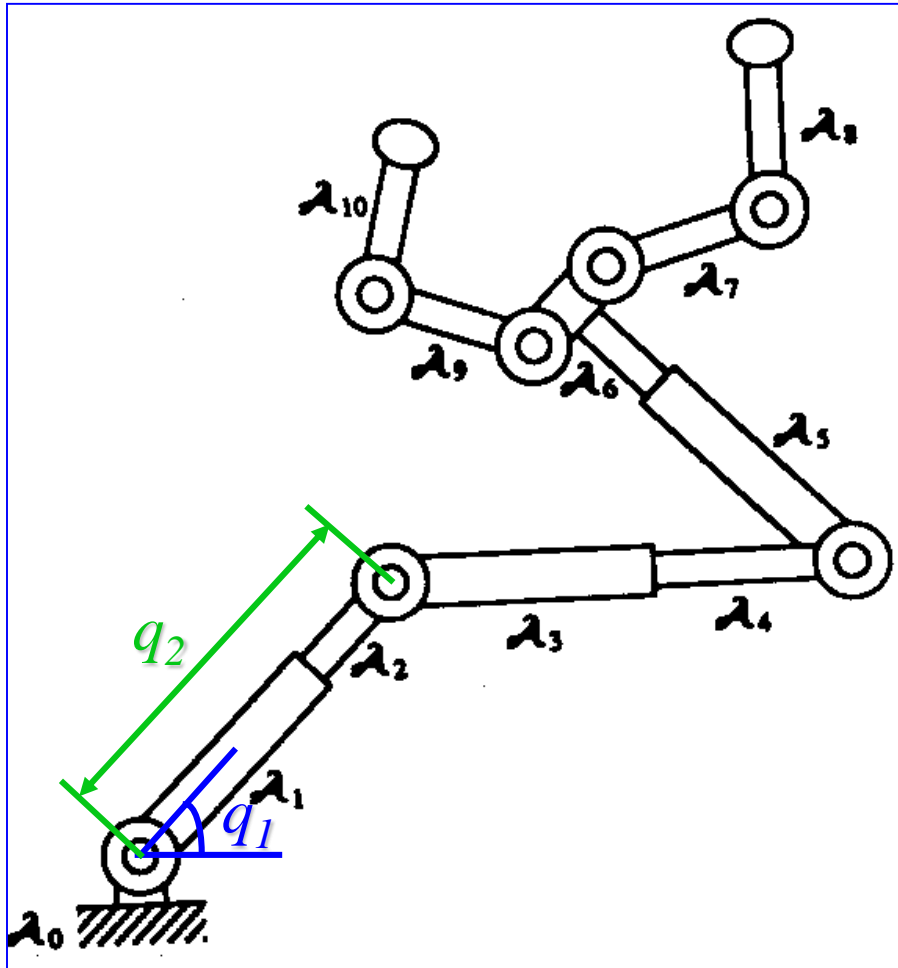


# Espace de configuration

- Représentation abstraite d'un problème de planification
- Employer des **algorithmes de recherches** génériques pour :
  - robot 2D glisse en x, y
  - robot conduite différentielle
  - bras articulé
  - robot humanoïde avec 30 joints
  - etc.



# Exemple configuration (bras articulé)



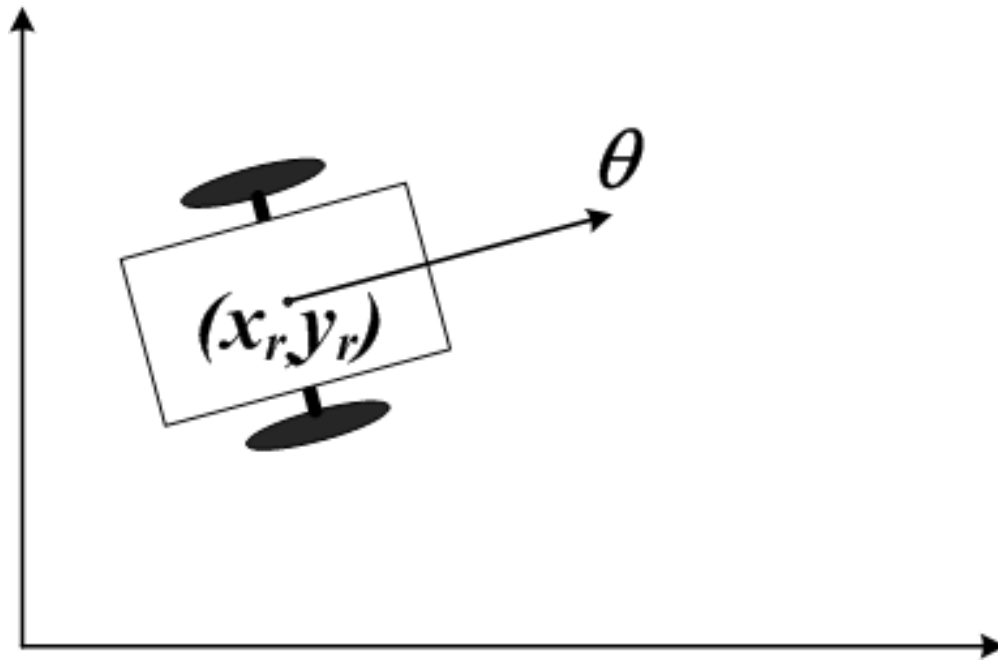
- La configuration d'un bras est l'ensemble des angles  $\theta_i$  et étirements  $d_j$ .
- Pour un robot articulé complexe de 10 actionneurs :

$$\vec{q} = (q_1, q_2, \dots, q_{10})$$

# Exemple configuration (robot mobile)

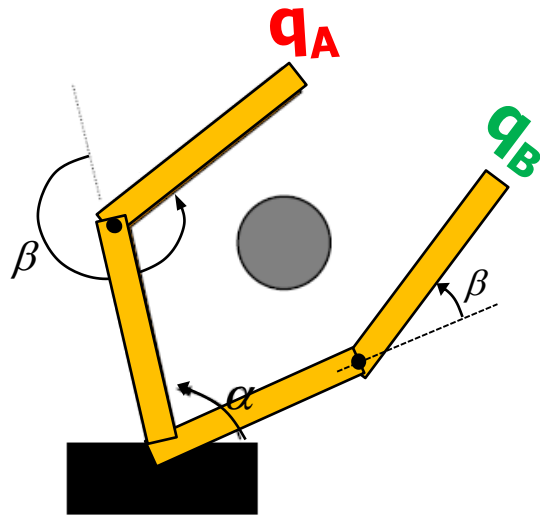
---

- Espace de travail en 2D
- Configuration :  $\vec{q} = (x_r, y_r, \theta)$



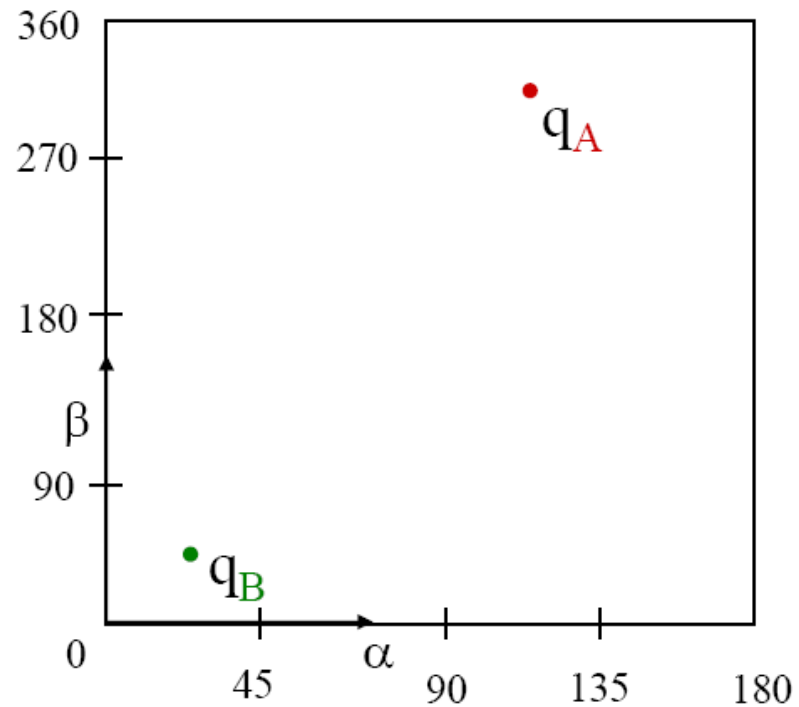


# Obstacle dans l'espace de travail



Un obstacle dans l'espace de travail

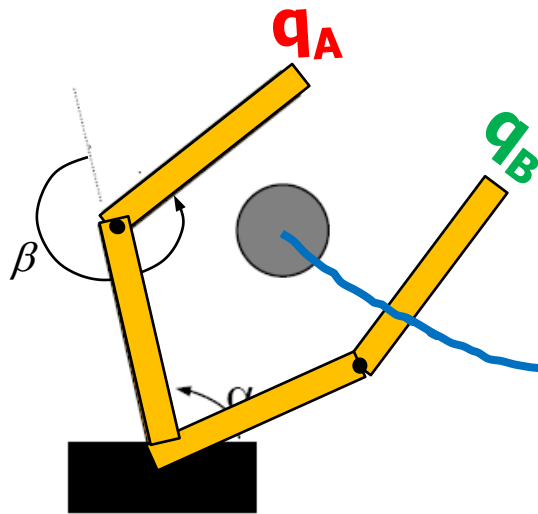
Où placer  ?



Espace de configuration

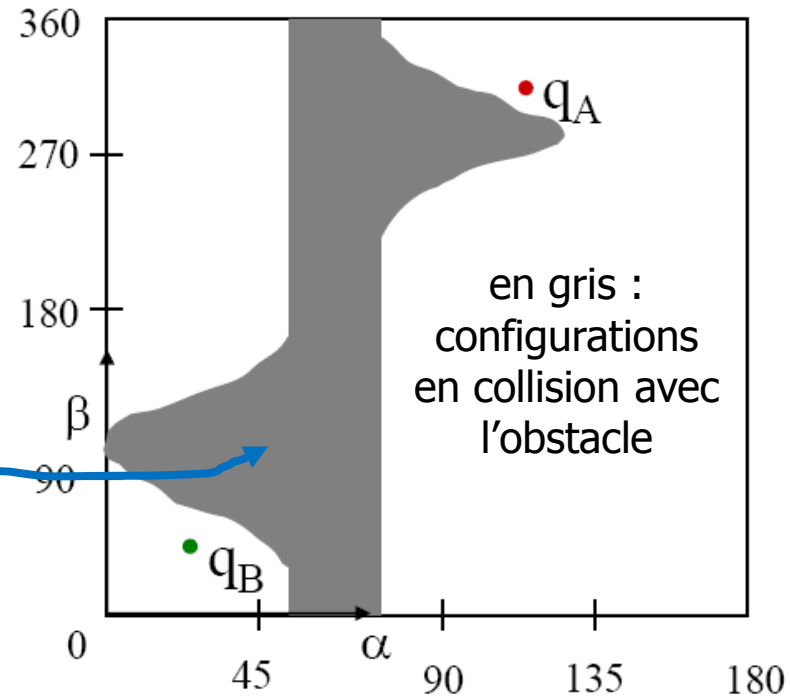
*(en 2D, car 2 joints)*

# Obstacle dans l'espace des configurations



Un obstacle dans l'espace de travail

Comment aller de  $q_A$  vers  $q_B$ ?

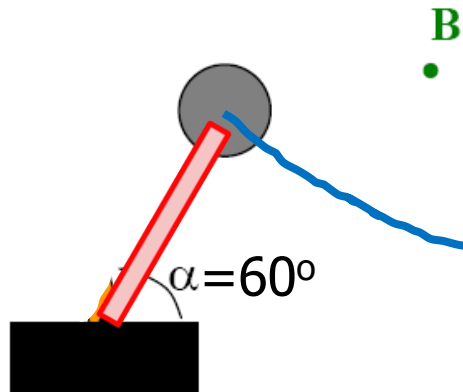


Espace de configuration

(en 2D, car 2 joints)

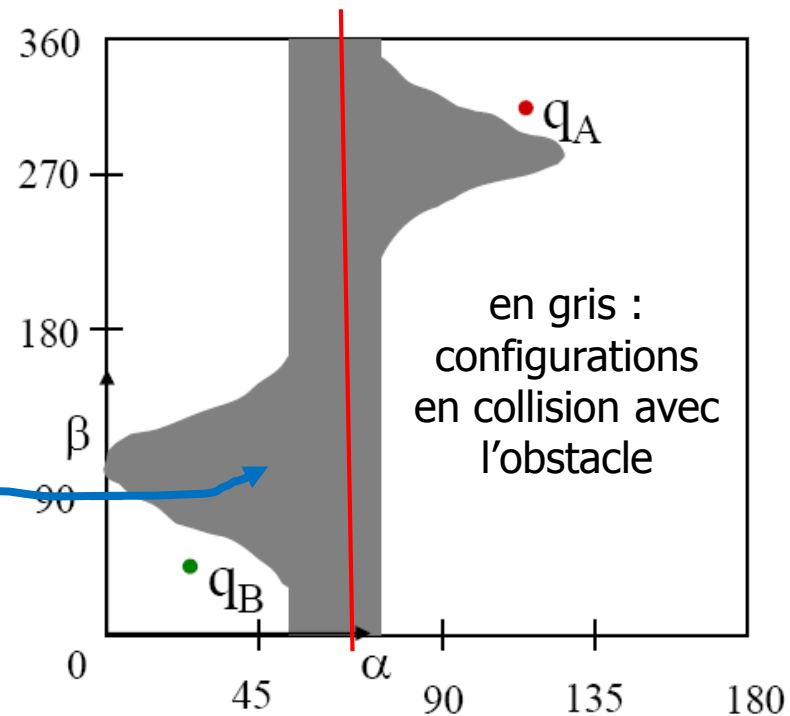
# Obstacle dans l'espace des configurations

dès que  $\alpha = 60^\circ$ , on a collision, peu importe  $\beta$



Un obstacle dans l'espace de travail

Comment aller de  $q_A$  vers  $q_B$ ?



Espace de configuration

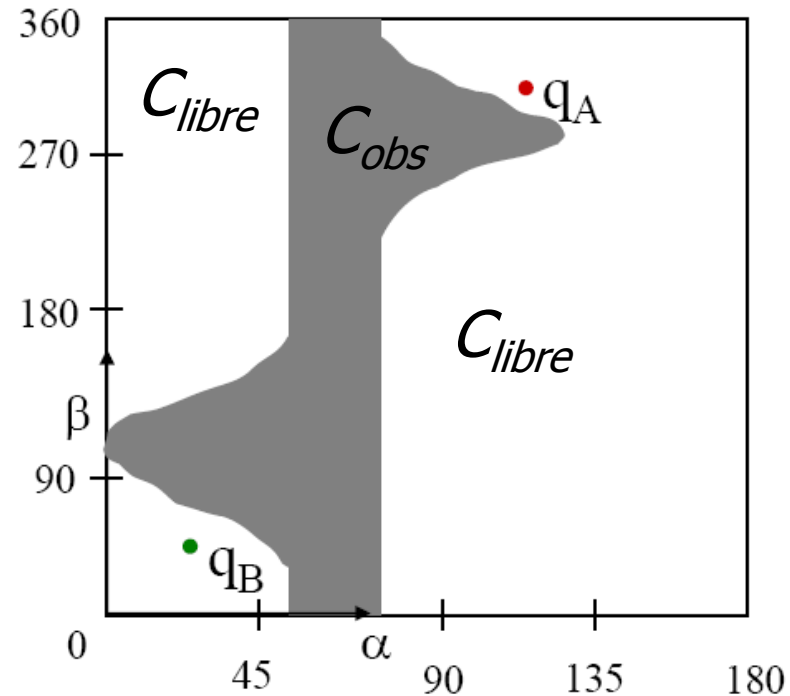
*(en 2D, car 2 joints)*

# Espace des configurations : terminologie

L'espace des configurations  $C$  est composé :

- espace libre  $C_{libre}$
- espace des obstacles  $C_{obs}$

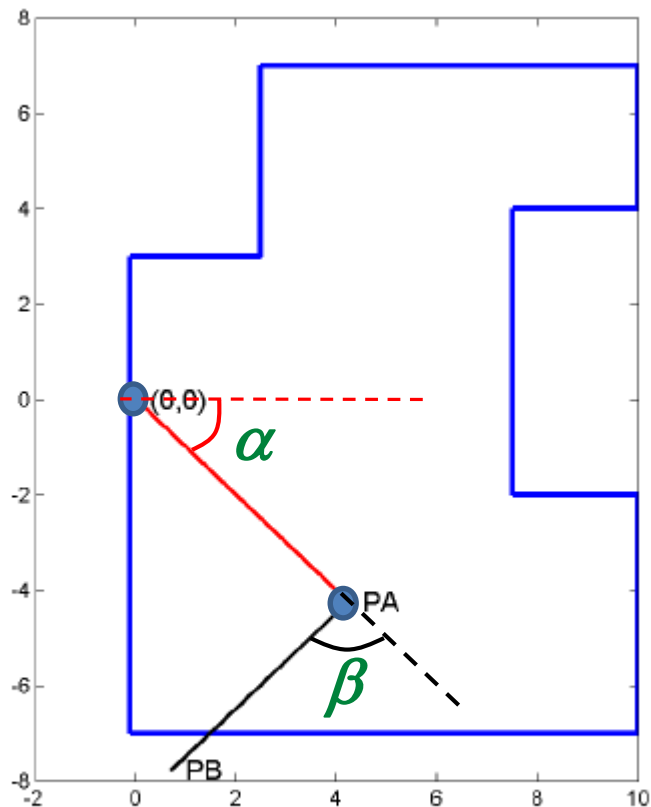
$$C = C_{libre} + C_{obs}$$



Espace de configuration  
(en 2D, car 2 joints)

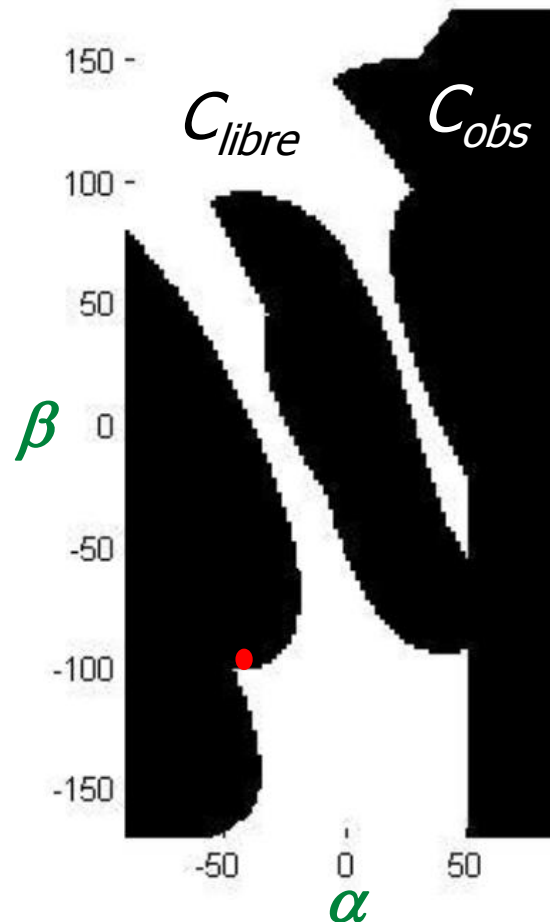
# Obstacle dans l'espace des configurations

- Tiré du TP de 2011



$$\alpha = -45^\circ, \beta = -90^\circ$$

espace des configurations



# Obstacle dans l'espace des configurations

---

construction de l'espace des configurations avec obstacles

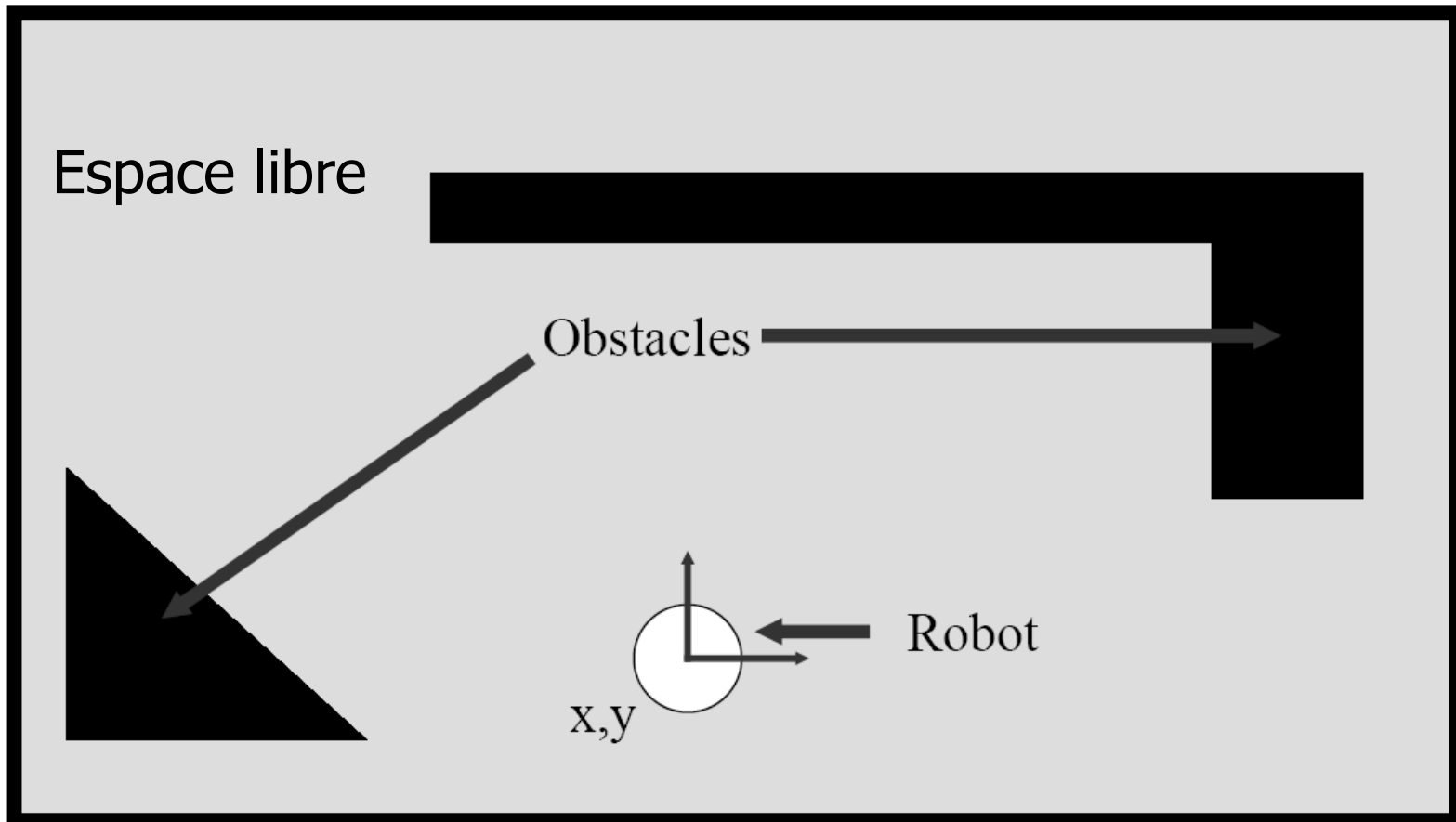


□  $C_{libre}$

■  $C_{obs}$

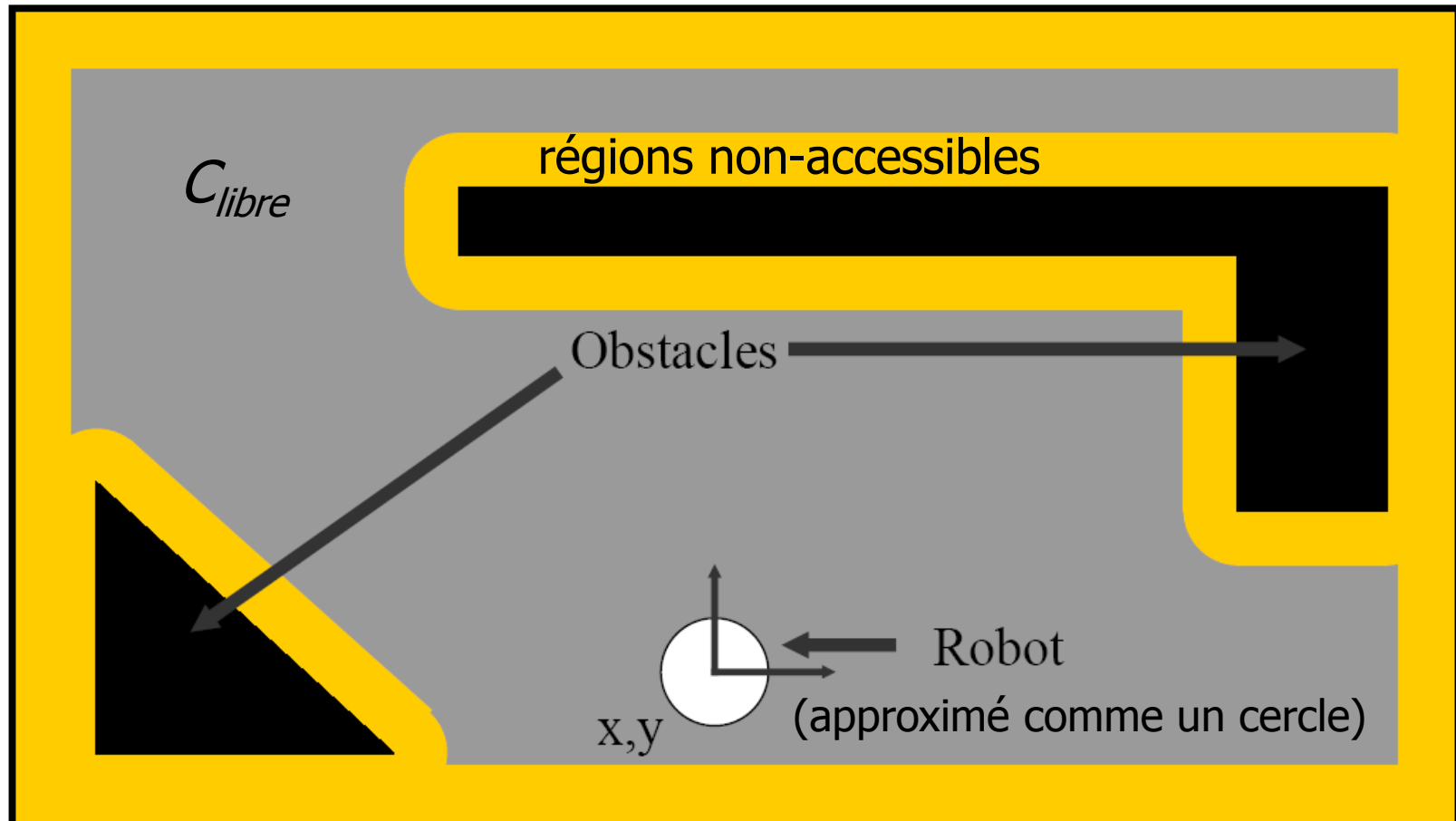
# Espace de configuration C

- Pour un robot circulaire se déplaçant en x-y



# Espace de configuration C

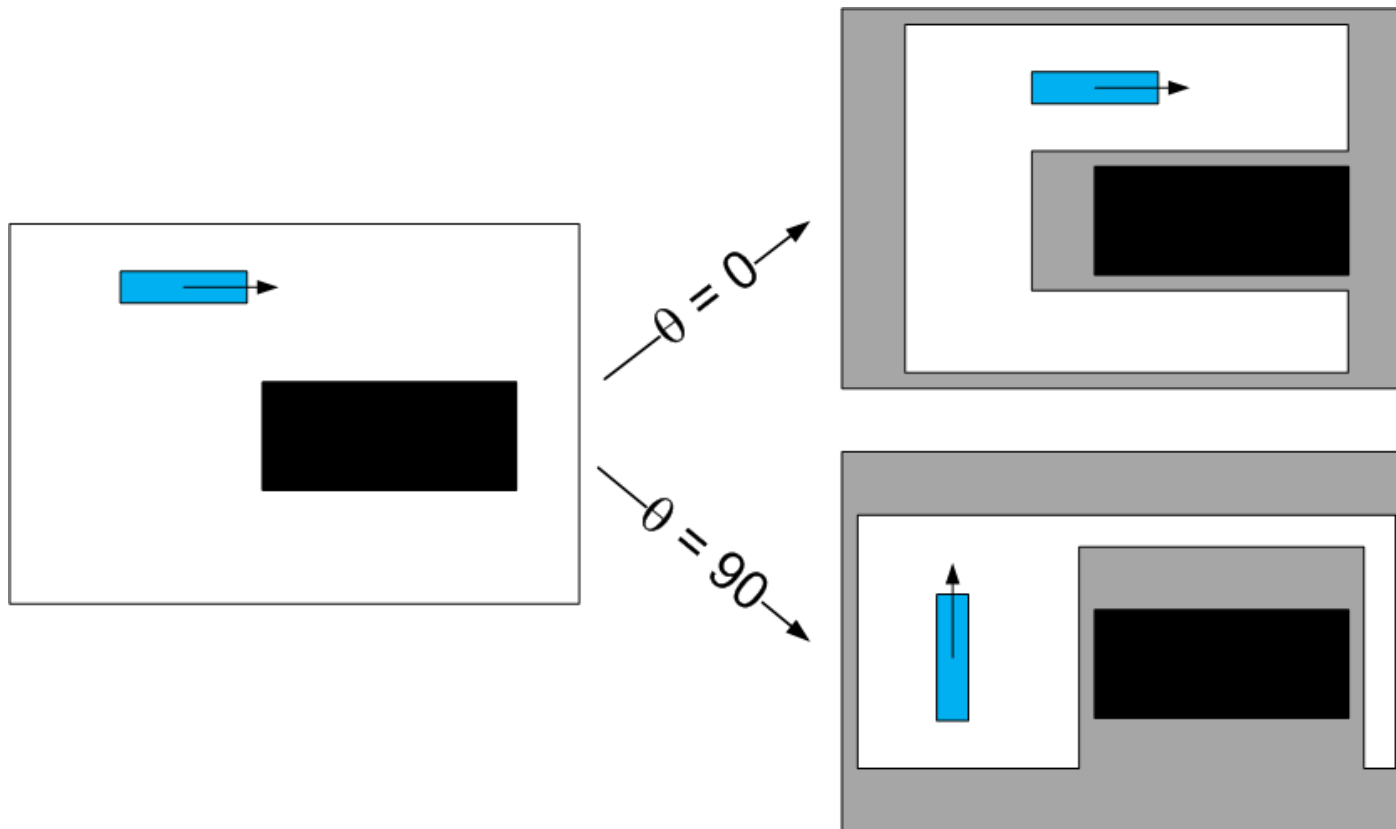
- Le centre du robot ne peut être que dans le gris





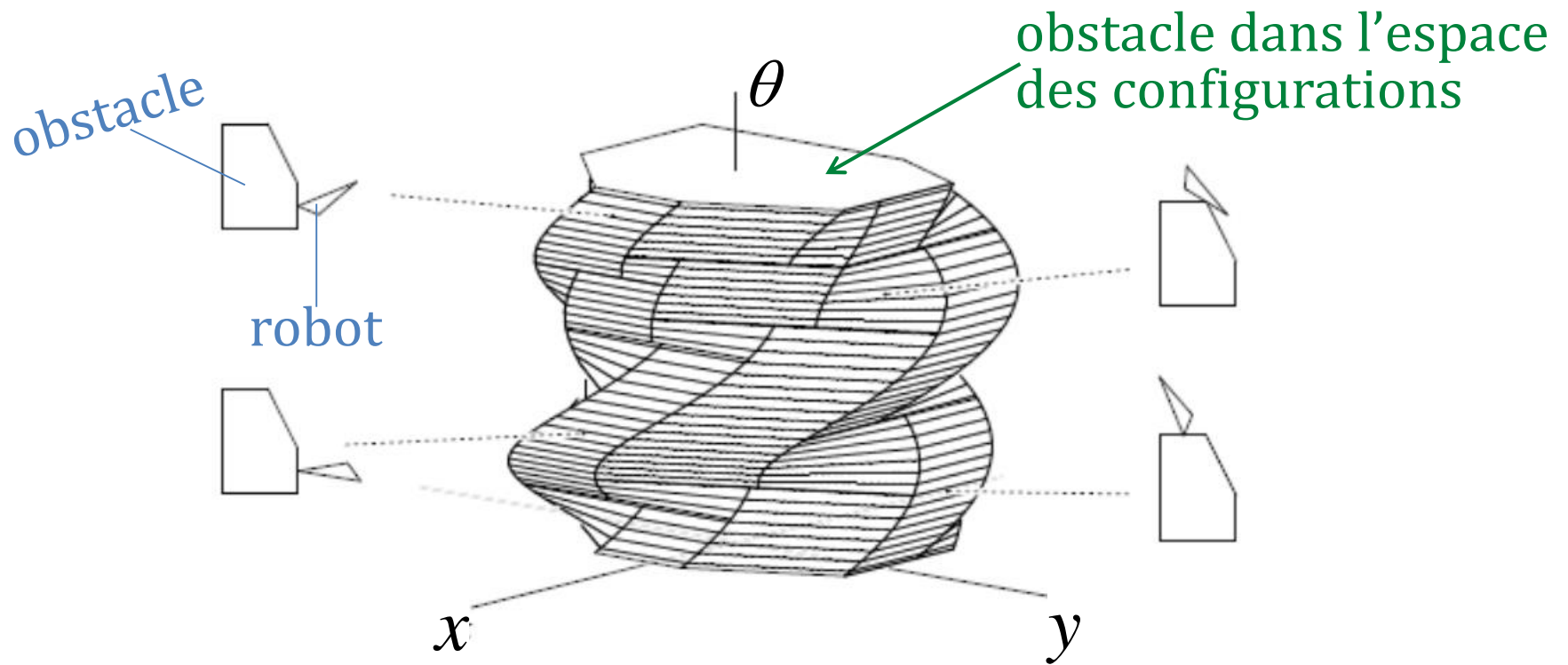
# Espace de configuration C

- Pour un robot se déplaçant en conduite différentielle :  $(x, y, \theta)$  (3 dimensions)



# Espace de configuration C

- Pour un robot se déplaçant en conduite différentielle :  $(x, y, \theta)$  (3 dimensions)



# Planification de trajectoire : abstraction

- ...aussi appelé « *piano mover's problem* »
- On a :
  - espace de travail  $W$  en  $\mathbb{R}^2$  ou  $\mathbb{R}^3$
  - régions d'obstacles  $O$
  - un robot  $R$  avec géométrie dans  $W$
  - espace configuration  $C$  (avec  $C = C_{libre} + C_{obs}$ )
  - configuration initiale  $q_1 \in C_{libre}$
  - configuration finale  $q_2 \in C_{libre}$
- Trouver une trajectoire  $\tau$  continue dans  $C_{libre}$  entre  $q_1$  et  $q_2$ .

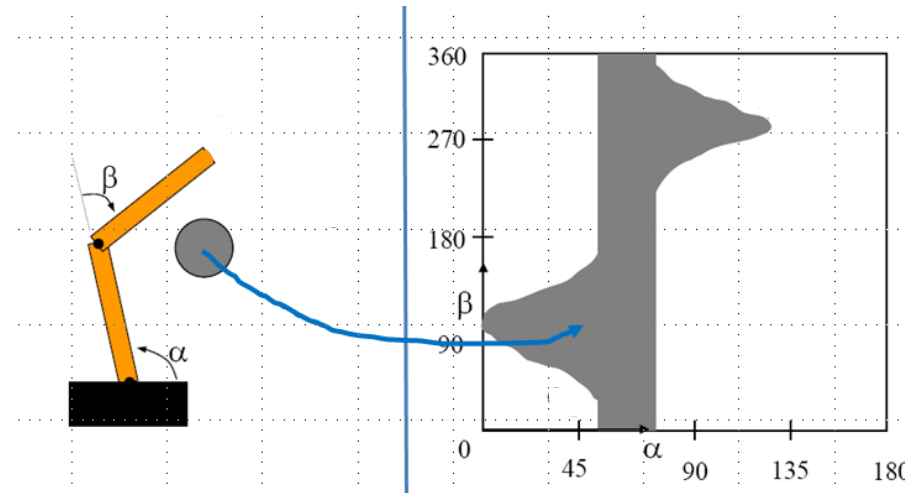
« monde physique »

# Planification de trajectoire : abstraction

- ...aussi appelé « *piano mover's problem* »
- On a :

- espace de travail  $W$  en  $\mathbb{R}^2$  ou  $\mathbb{R}^3$
- régions d'obstacles  $O$
- un robot  $R$  avec géométrie dans  $W$
- espace configuration  $C$  (avec  $C = C_{libre} + C_{obs}$ )

calcul de  $C_{obs}$   
n'est pas facile



# Planification de trajectoire : abstraction

- ...aussi appelé « *piano mover's problem* »

- On a :

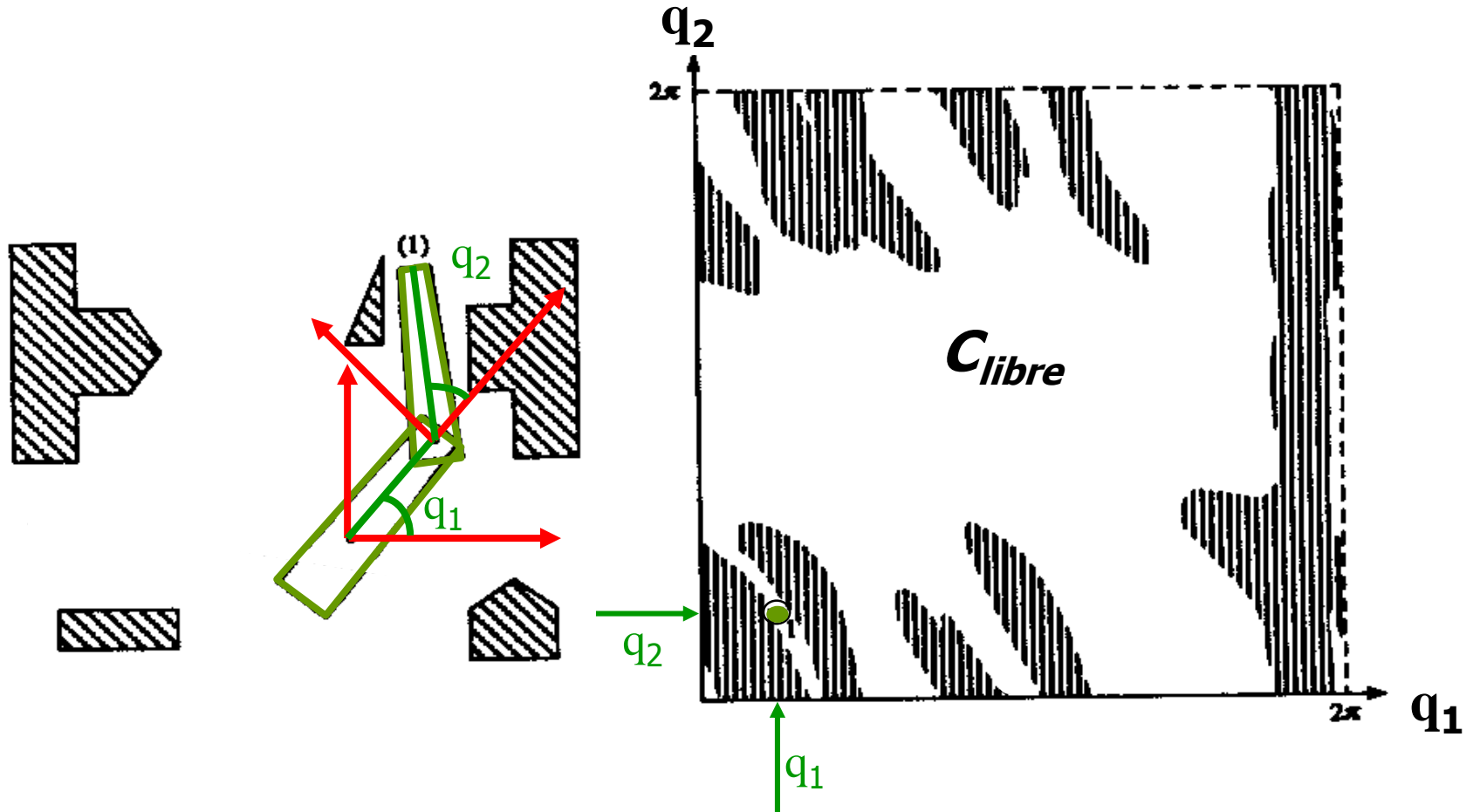
- espace de travail  $W$  en  $\mathbb{R}^2$  ou  $\mathbb{R}^3$
- régions d'obstacles  $O$
- un robot  $R$  avec géométrie dans  $W$
- espace configuration  $C$  (avec  $C = C_{libre} + C_{obs}$ )
- configuration initiale  $q_1 \in C_{libre}$
- configuration finale  $q_2 \in C_{libre}$

calcul de  $C_{obs}$   
n'est pas facile

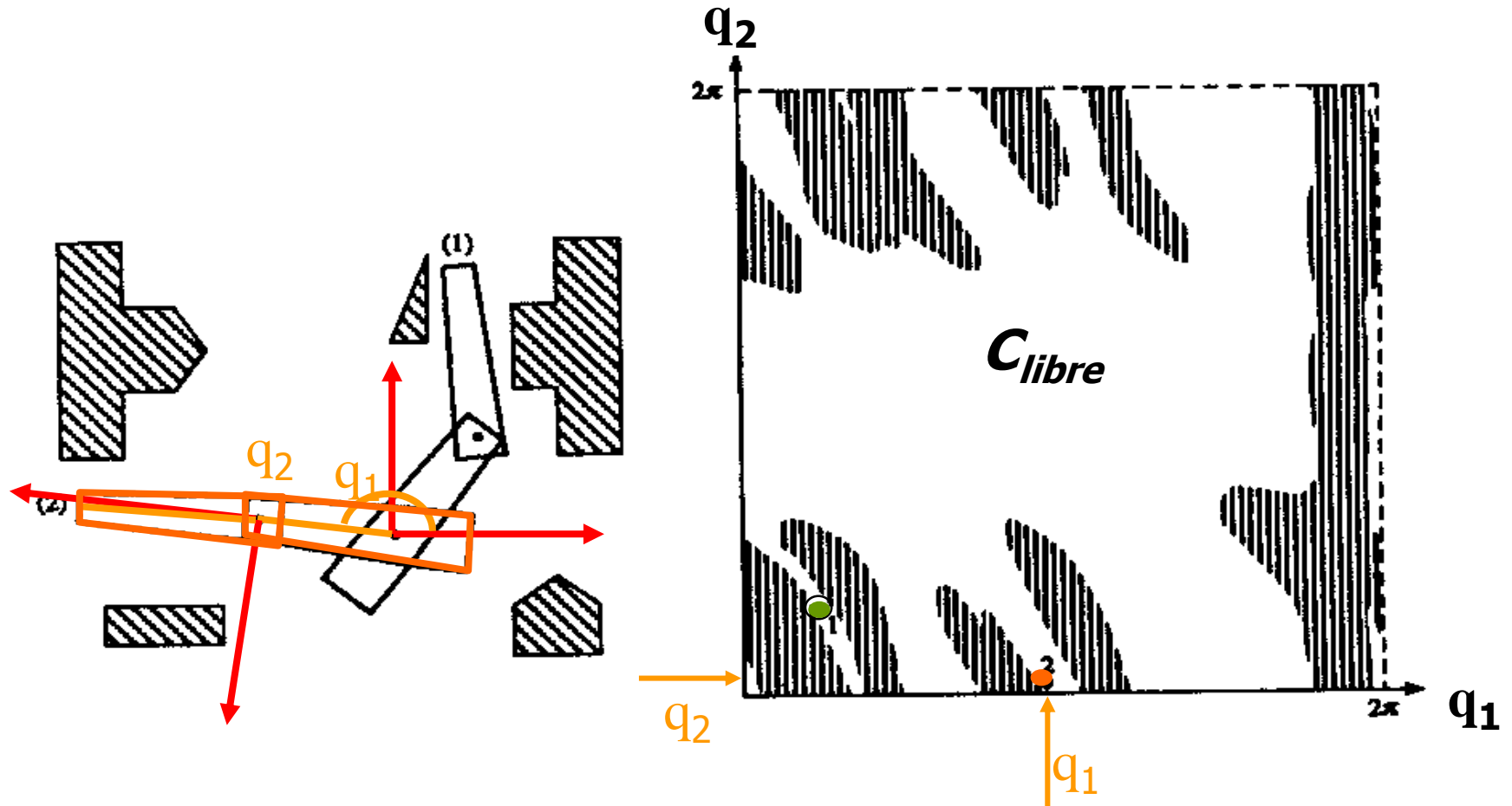
- Trouver une trajectoire  $\tau$  continue dans  $C_{libre}$   
entre  $q_1$  et  $q_2$ .  
calcul exact approximativement  
exponentiel en dimension de  $q$

# Espace de configuration : position 1

- Pour un bras avec 2 joints rotatifs  $q_1$  et  $q_2$

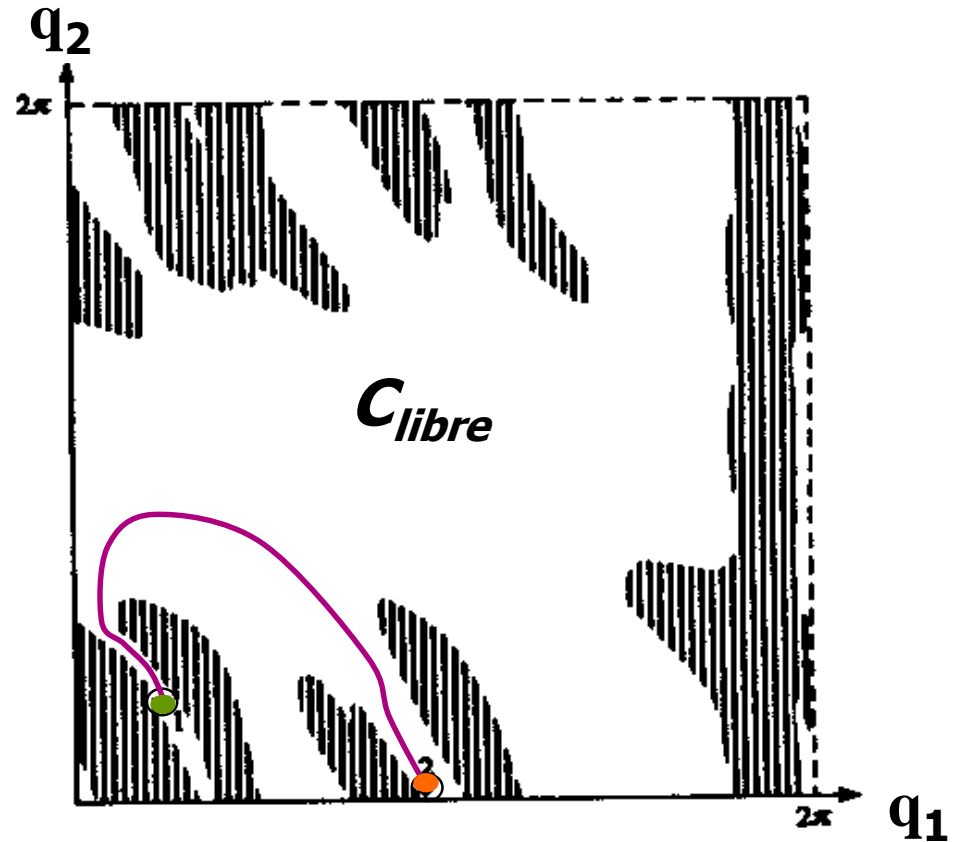
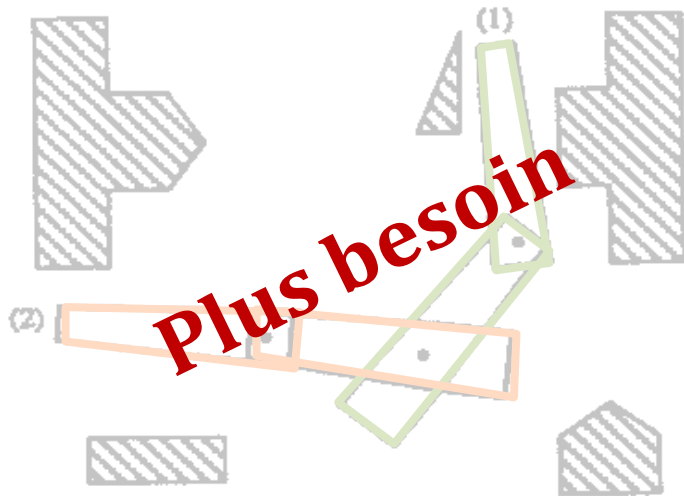


# Espace de configuration : position 2



# Espace de configuration

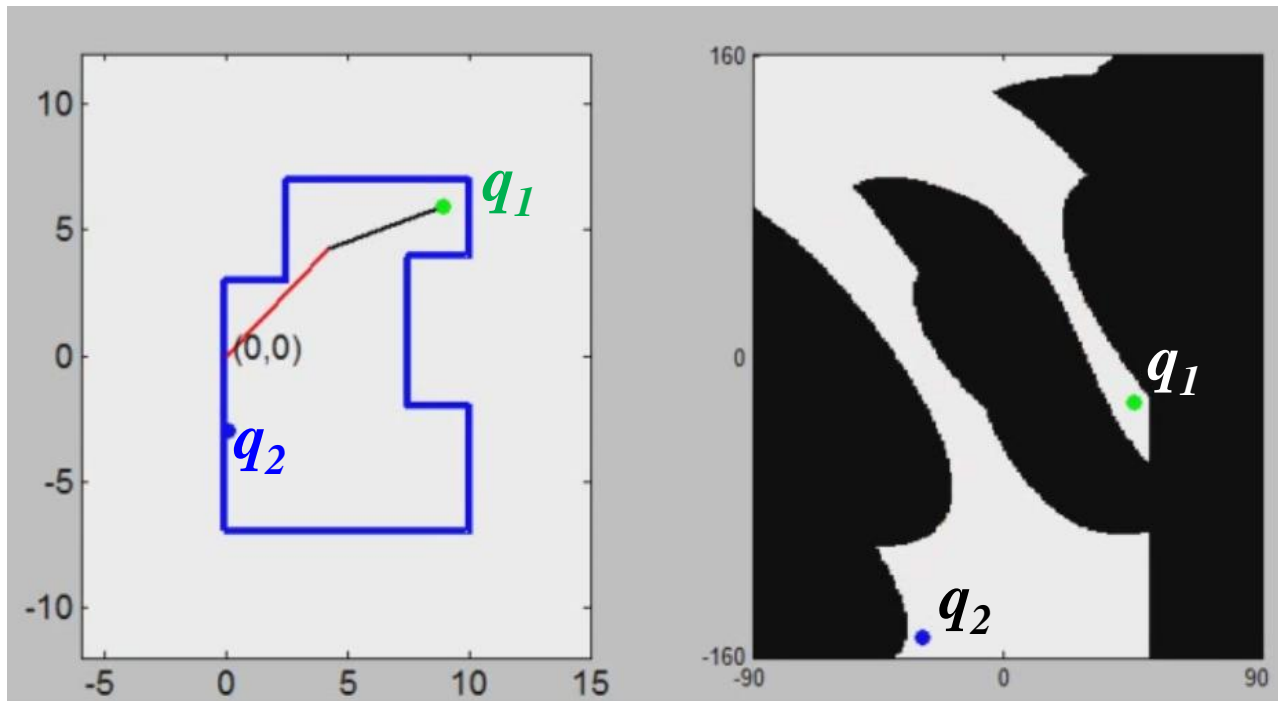
- Trajectoire possible entre  $q_1$  et  $q_2$  ? **oui!**



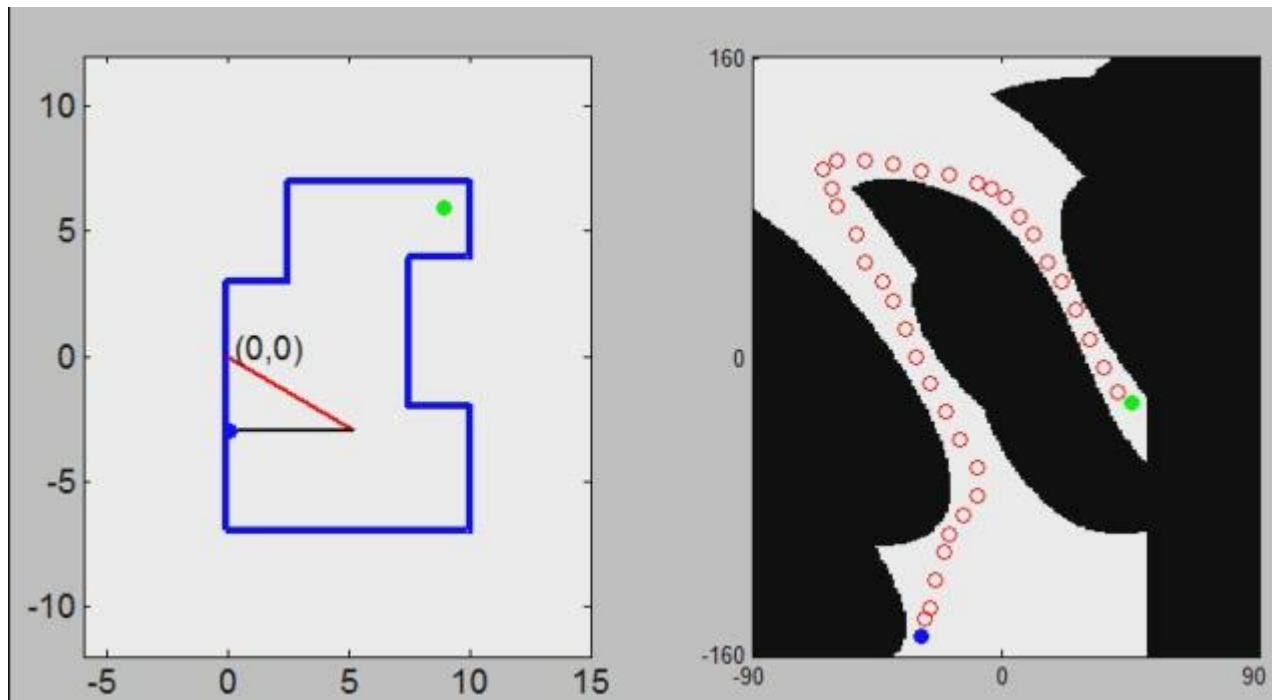


# Exemple : bras robotisé

On veut déplacer le bout du bras (*end effector*) de  $q_1$  à  $q_2$ .



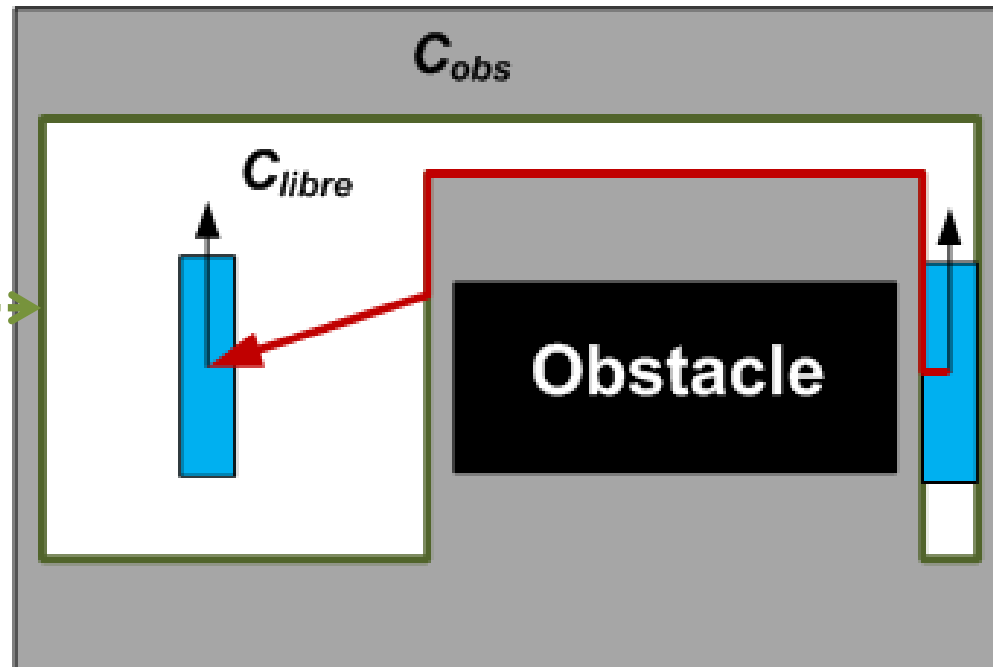
# Exemple : bras robotisé



# Configuration semi-libre

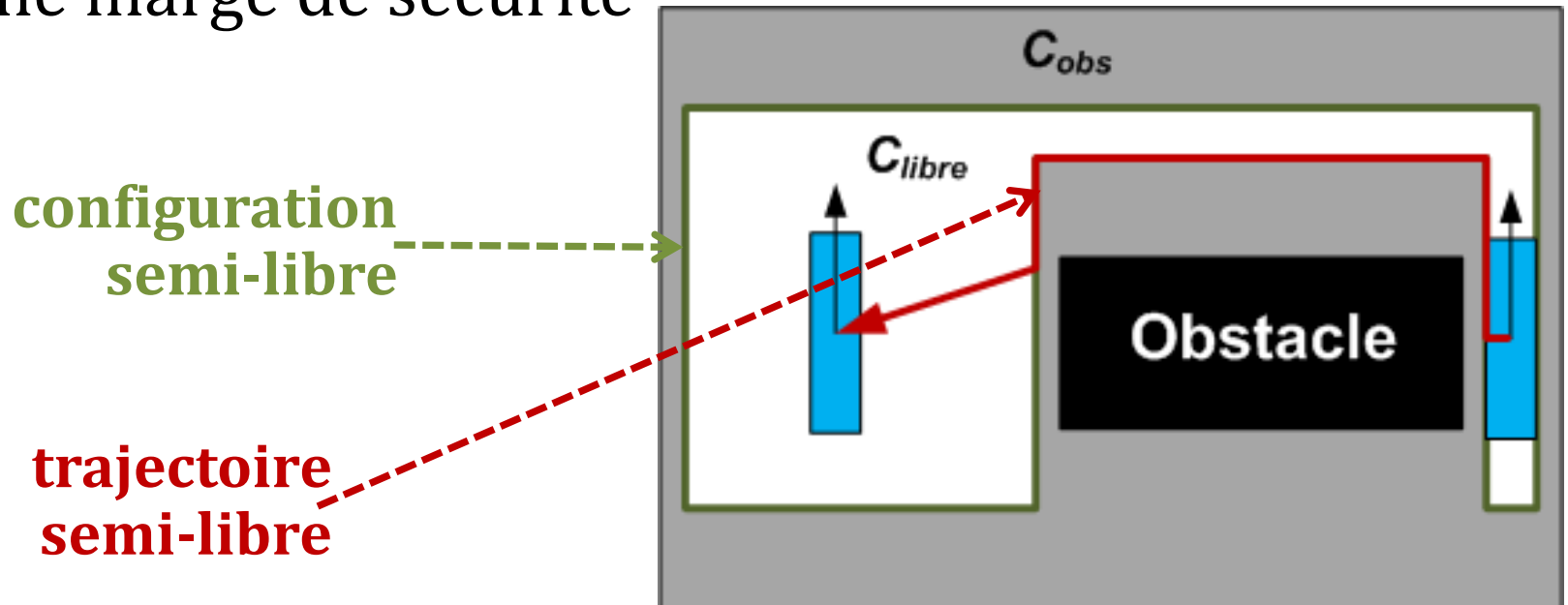
- Une configuration est semi-libre si le robot touche un obstacle, sans le pénétrer.
- Les configurations semi-libres sont à la bordure entre  $C_{obs}$  et  $C_{libre}$ .

configuration  
semi-libre



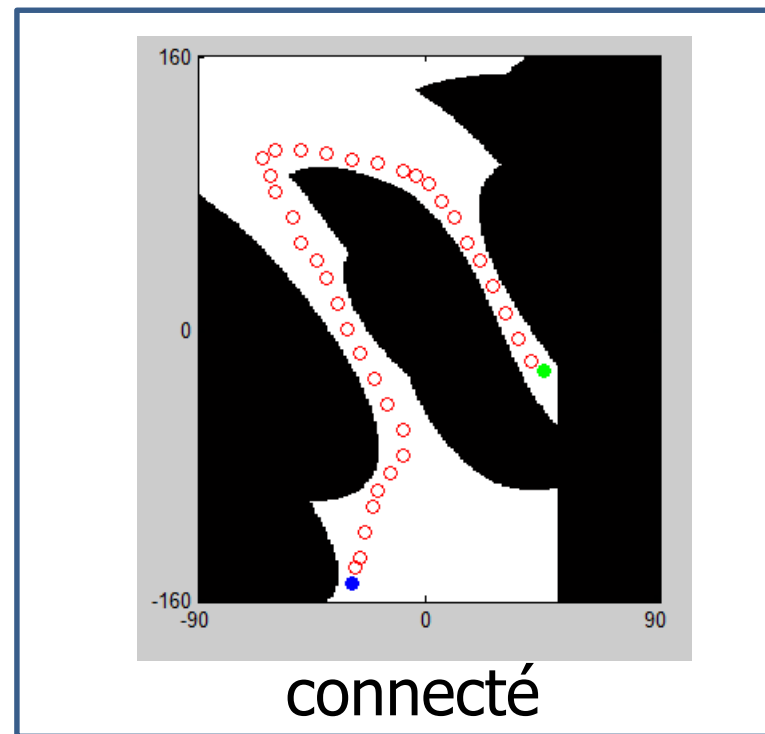
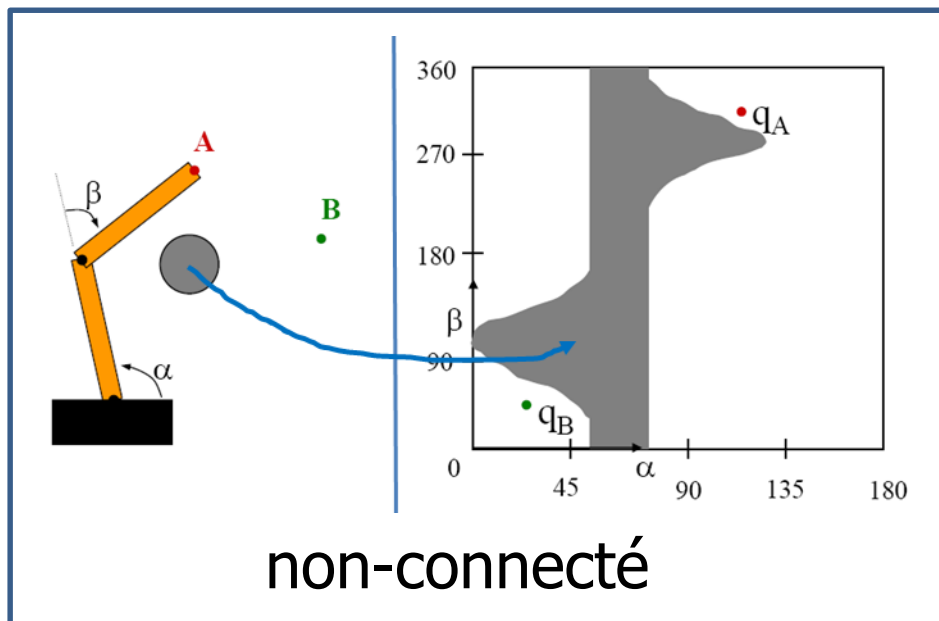
# Trajectoire semi-libre

- Trajectoire est libre si entièrement dans  $C_{libre}$ .
- Trajectoire est semi-libre si elle contient au moins une configuration semi-libre.
- On cherche, en général, à éviter ces configurations : aucune marge de sécurité



# Connectivité de l'espace-C

- L'espace de configuration  $\mathcal{C}$  est connecté s'il existe une trajectoire entre n'importe quelle paire de configuration  $q_1$  et  $q_2$  dans  $\mathcal{C}_{libre}$



# Planification de mouvements

---

- Recherche à trouver une trajectoire dans l'espace des configurations, sans collision  $\rightarrow C_{libre}$
- **Méthodes déterministes**
  - à chaque fois, l'algorithme retourne la même réponse
- **Probabilistes**
  - fonctionne par échantillonnage au hasard
  - ne trouve pas toujours la même trajectoire

# Méthodes Déterministes

---

- Par discrétisation :
  - Graphe des visibilités
  - Décomposition cellulaire verticale
  - ~~Diagrammes de Voronoï~~
- ~~Algorithmes *bug0*, *bug1* et *bug2*~~
- Champs de potentiels
- Propagation de front d'ondes

# Solutions exactes vs approximatives

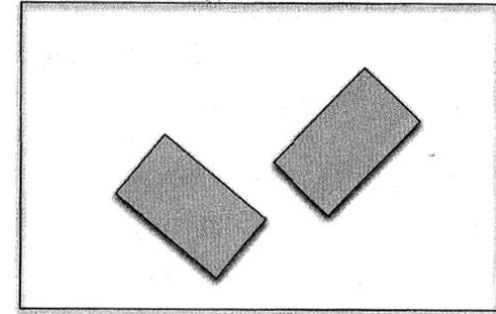
---

- Un algorithme **exact** :
  - trouvera une solution si elle existe
    - **algorithme complet**
  - temps de calcul trop important pour des problèmes intéressants (e.g. humanoïdes à 30 joints).
- Un algorithme **approximatif** :
  - n'assure pas de trouver la solution...
  - ...mais est plus rapide.
  - *e.g.* grille discrète des configurations
    - (angle multiples de  $5^\circ$ , par exemple)

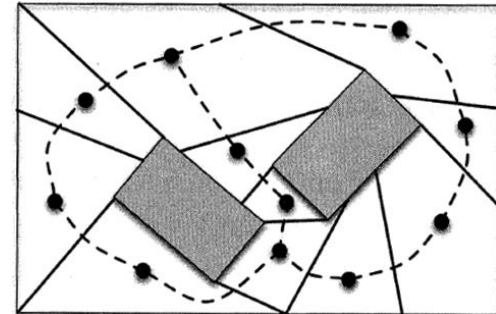


# Discrétisation de l'espace

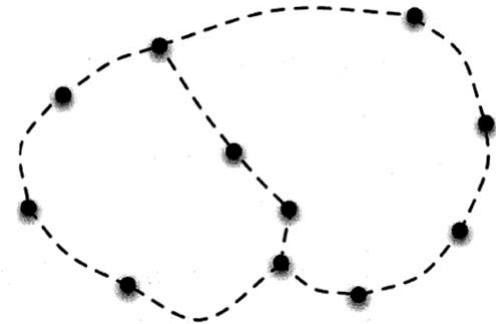
Représentation continue  
(espace des configurations)



Discrétisation  
+ graphe

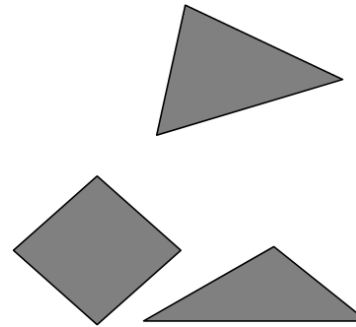


Recherche dans graphe  
(blind, best-first, A\*)

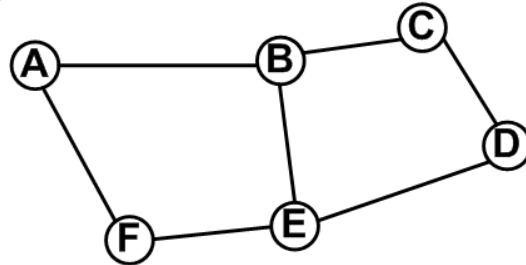


# Graphe des visibilités

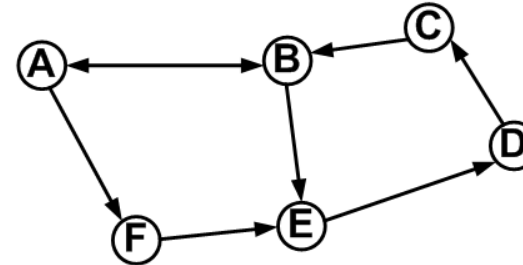
- Vous avez une carte géométrique connue
- Obstacles sont des **polygones**



- Utilise graphe  $G = (E, V)$  non-orienté comme représentation



graphe non-orienté



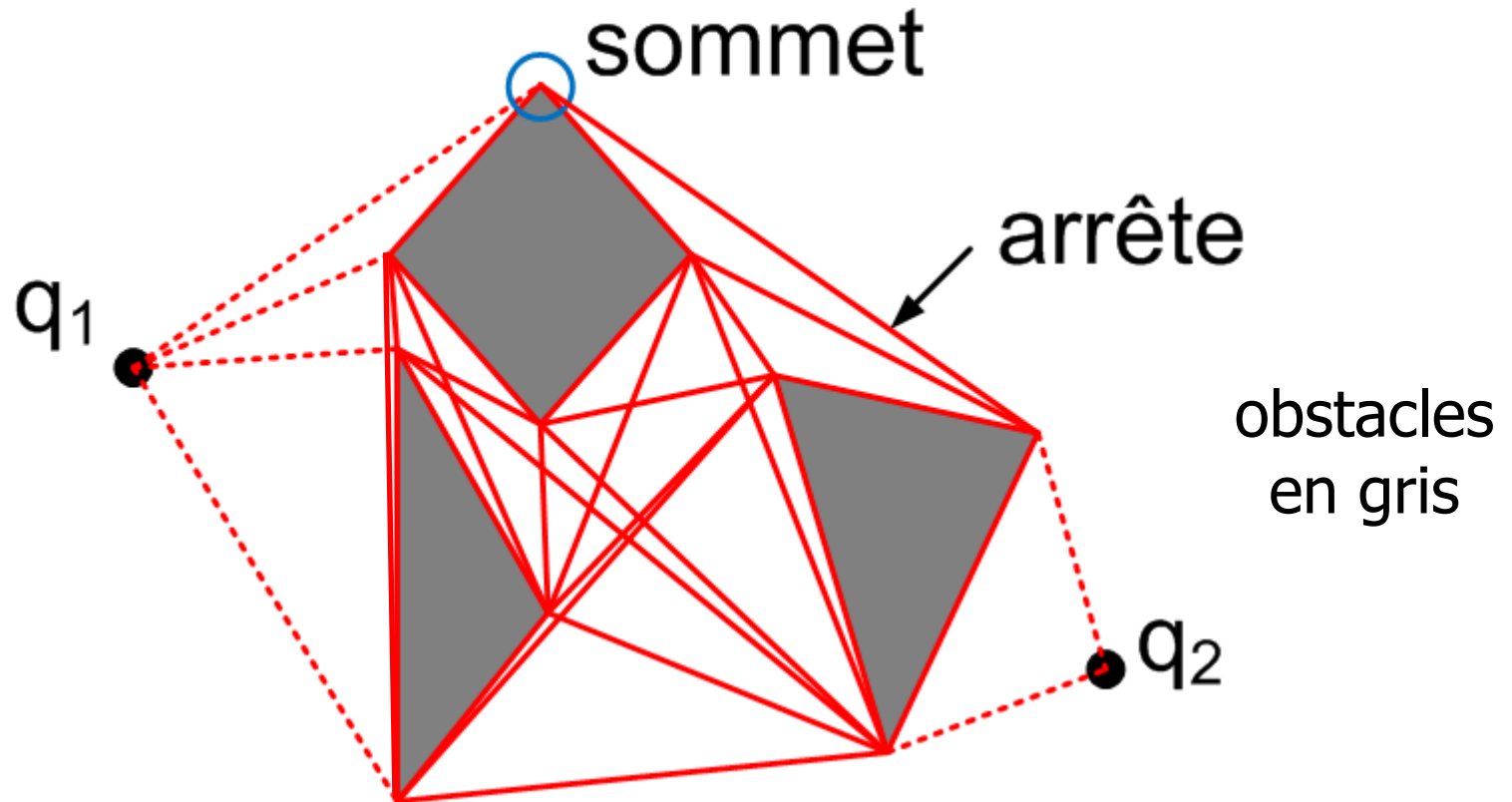
graphe orienté

- **Algorithme complet\***
- Approprié pour monde 2D
- Approche par « carte routière » : *roadmap*

\*complet : qui retourne une solution, si elle existe

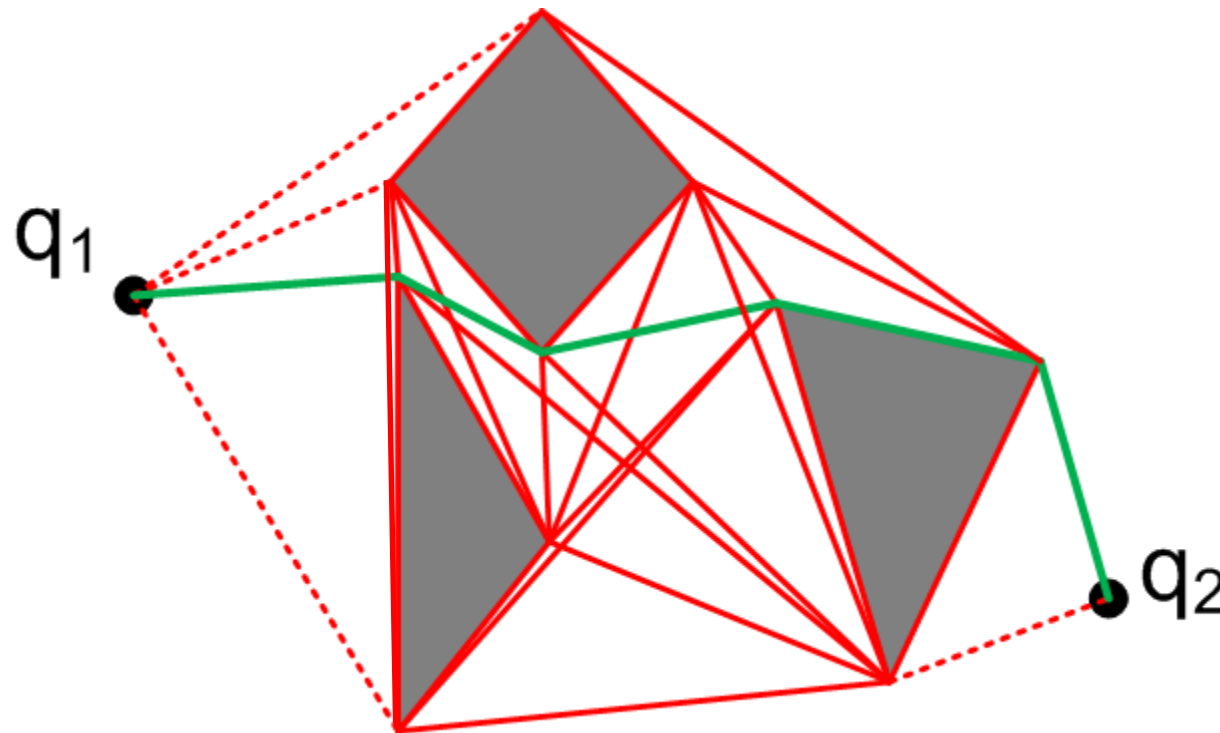
# Discrétisation : graphe des visibilités

- Les sommets  $V$  des obstacles qui sont mutuellement visibles sont connecté ensemble par une arrête  $E$ .



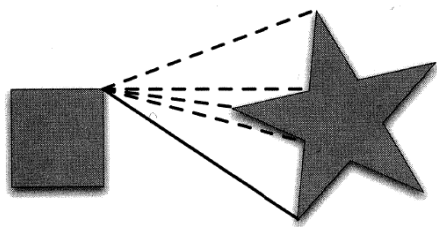
# Planification : graphe des visibilités

- Planification : recherche le chemin le plus court dans graphe  $G$  entre  $q_1$  et  $q_2$ .
- Algorithme de *Dijkstra* : complexité  $O(|E| + |V| \log |V|)$

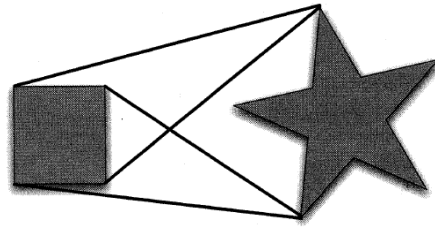


# Graphe des visibilité réduites

- Une ligne passant par deux sommets  $V_1 \in \mathbf{A}$  et  $V_2 \in \mathbf{B}$  est dite tangente si elle ne passe pas à l'intérieur d'un des deux obstacles  $\mathbf{A}$  ou  $\mathbf{B}$ .
- Vient réduire le nombre d'arrête  $E$  dans graphe  $G$ .

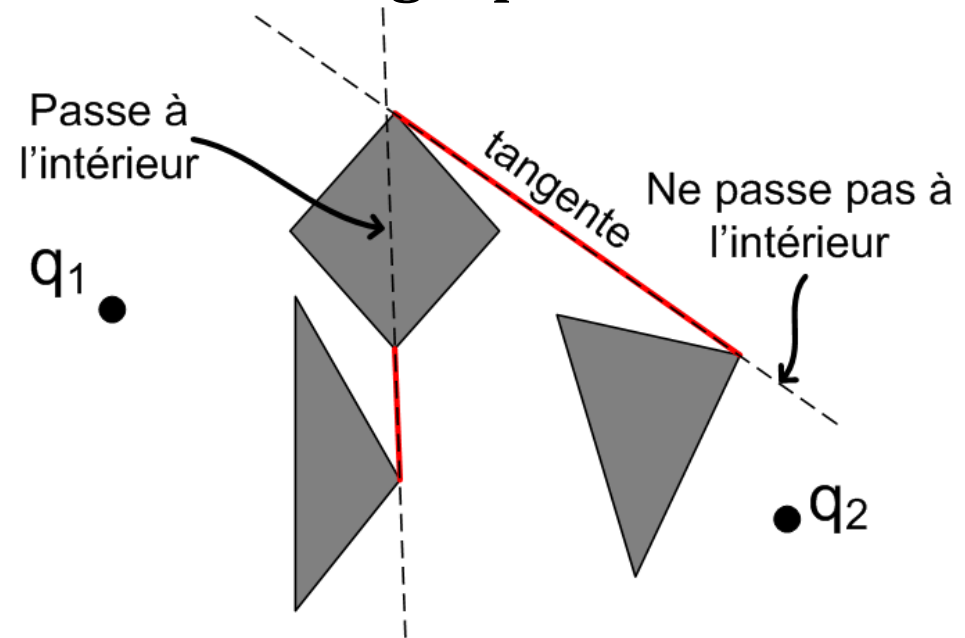


(a) Co-tangent lines from one vertex



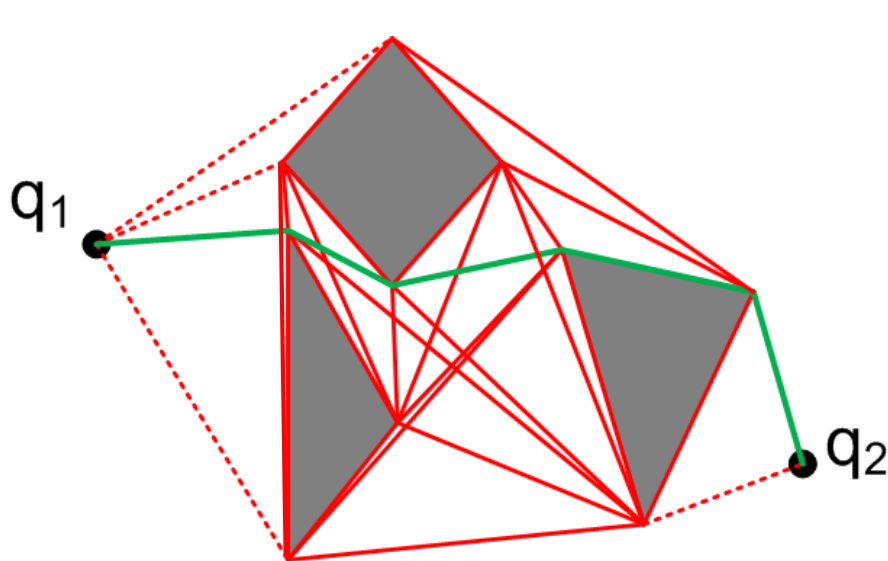
Co-tangent lines between two objects

Au maximum, 4 tangentes par  
paire d'obstacles

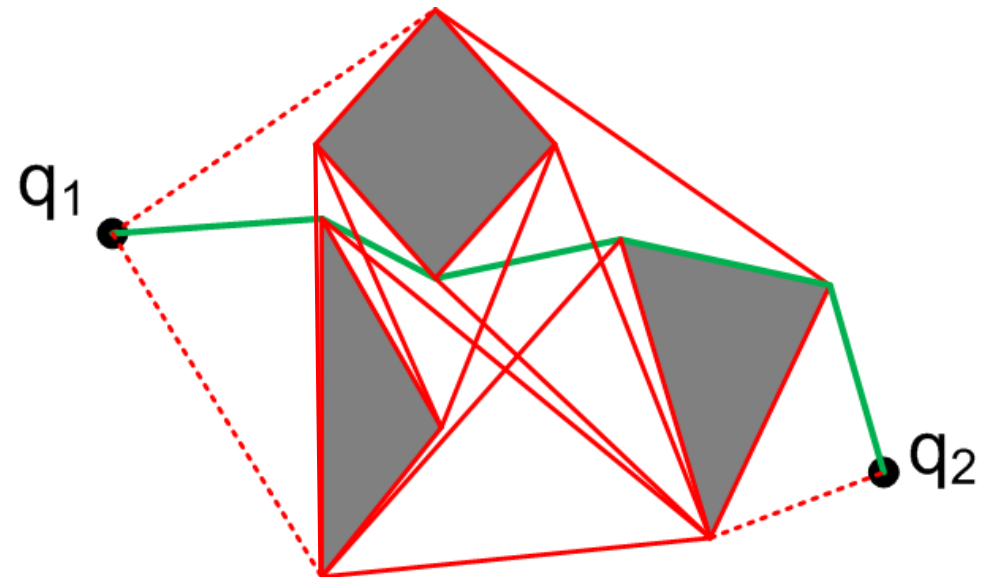


# Graphe des visibilité réduites

- Graphe plus simple = recherche plus rapide
- Donne toujours la solution la plus courte



graphe des visibilités



graphe des visibilités réduites

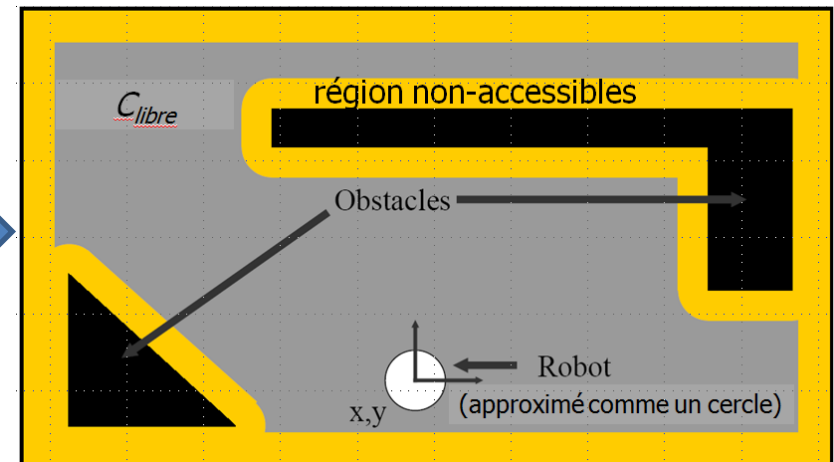
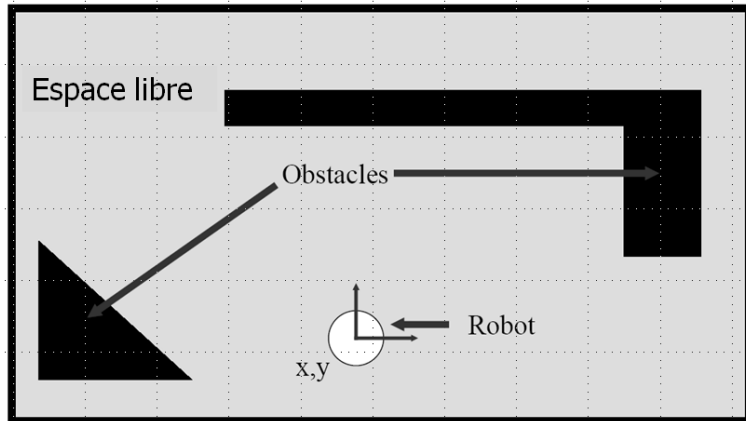
# Graphe des visibilités

---

- Produit nécessairement la trajectoire la plus courte entre  $q_1$  et  $q_2$ ...
- ...mais avec possiblement des trajectoires semi-libres...
- ...et pour un robot de la taille d'un point.

# Somme de Minkowski

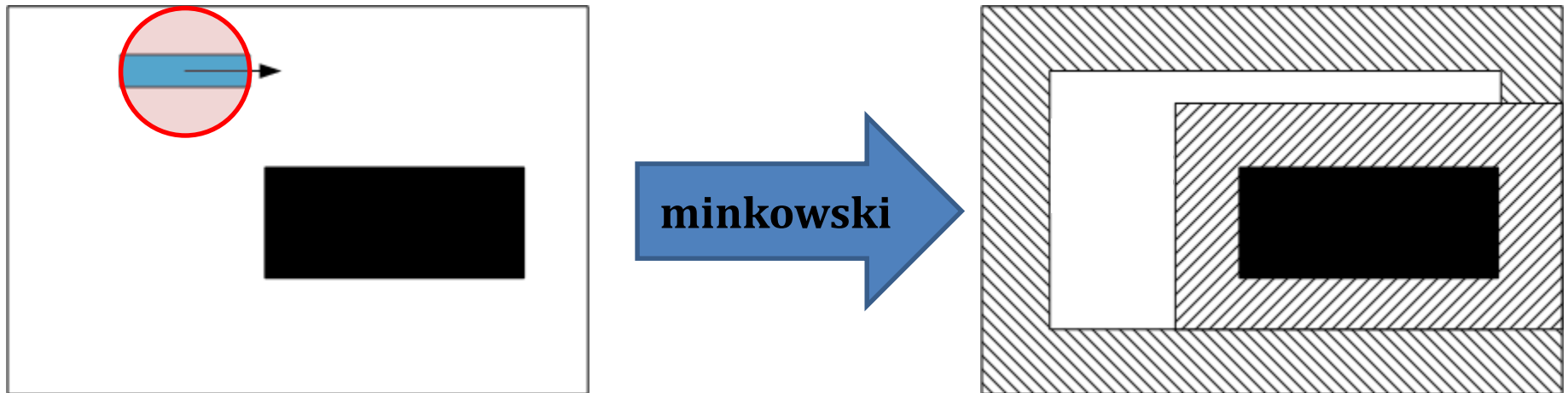
- Somme de Minkowski :
  - « gonfle » les obstacles pour tenir compte taille robot
  - robot redevient un point
- Algo complet seulement si le robot est circulaire



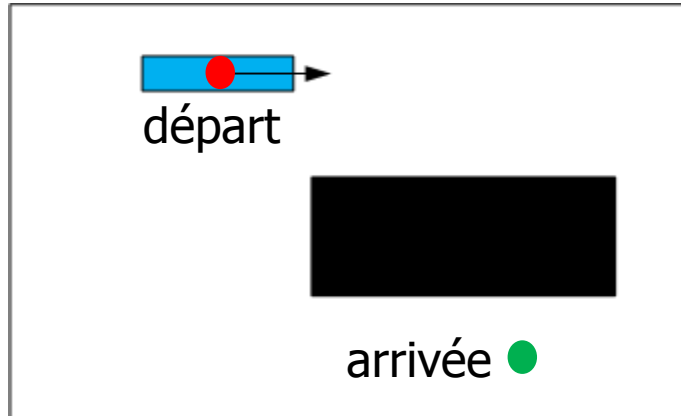


# Somme de Minkowski

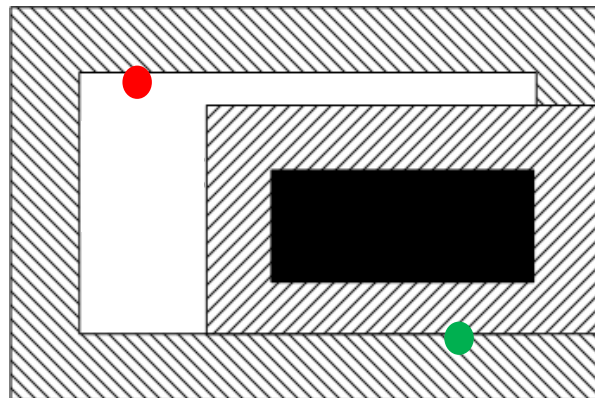
- Aucune garantie d'être complet si non-circulaire



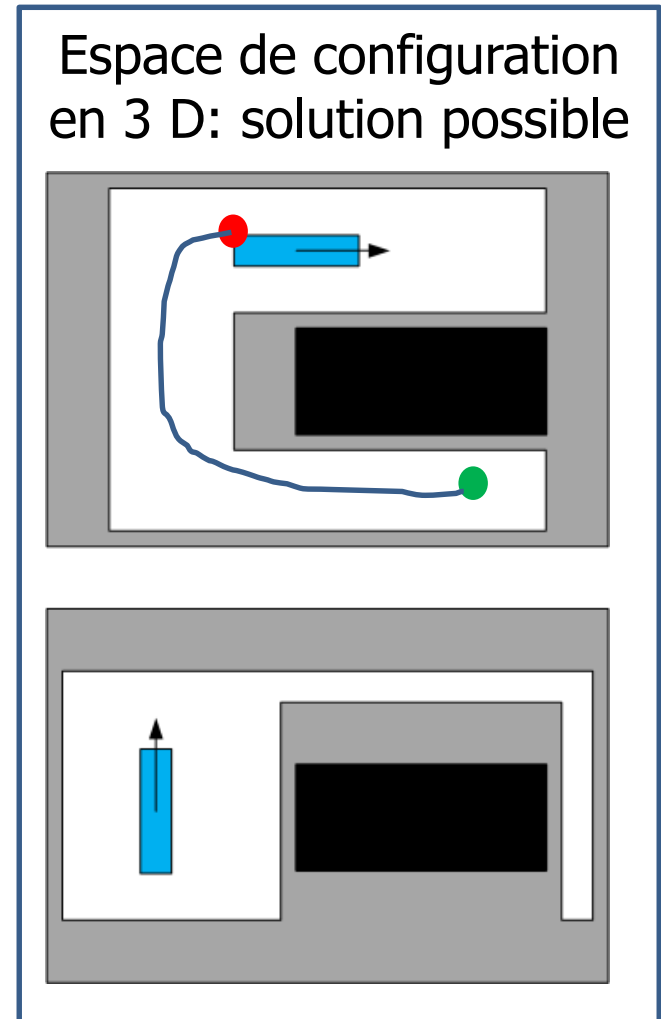
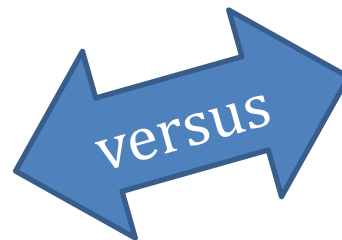
# Somme de Minkowski



**NON-COMPLET!**



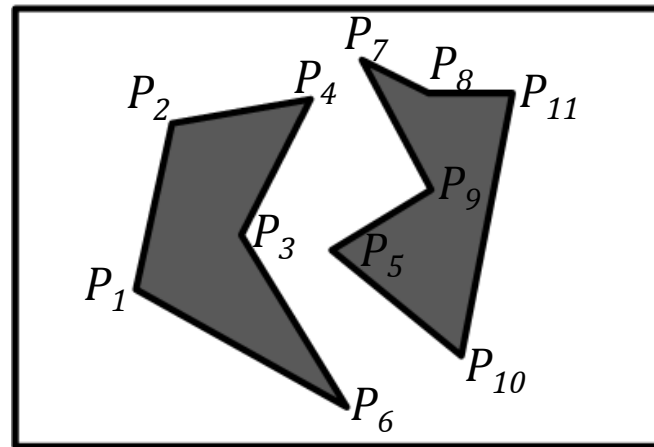
Minkowski : pas de solution



# Décomposition cellulaire verticale

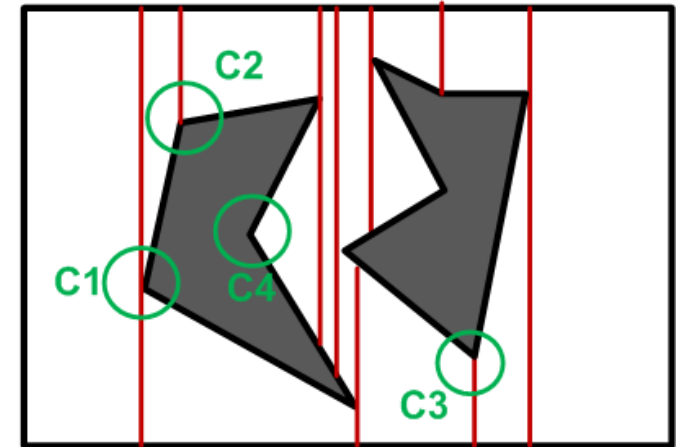
- Décomposer une carte 2D en trapèzes et triangles (éléments convexes),
- Pour chaque sommet  $P_i$  des obstacles:
  - étend **ligne verticale** en haut et en bas, jusqu'à un obstacle
- 4 cas possibles:

- **C1**:(haut,bas)
- **C2**:(haut),
- **C3**:(bas),
- **C4**:(aucun)

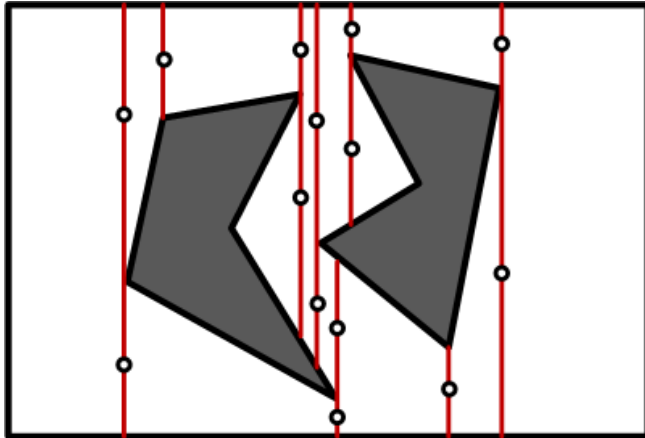


— balayage →

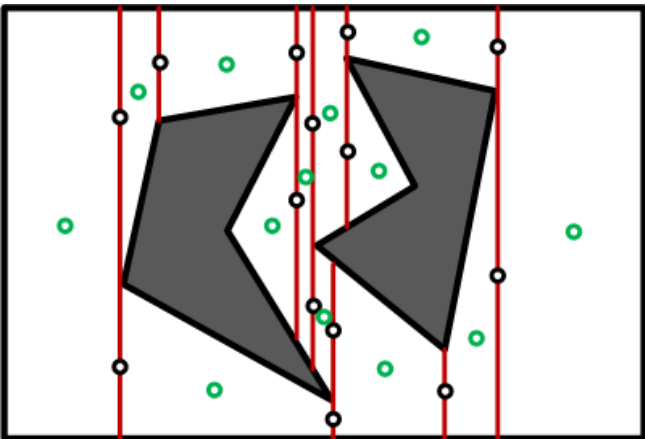
## Étape 1



# Décomposition cellulaire verticale



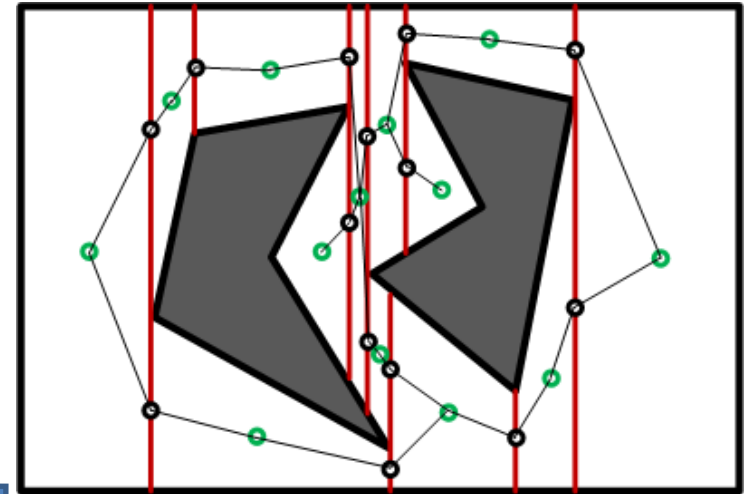
Étape 2:  
place 1 point au  
milieu frontière  
entre cellules



Étape 3:  
place 1 **point** au  
milieu de la  
cellules



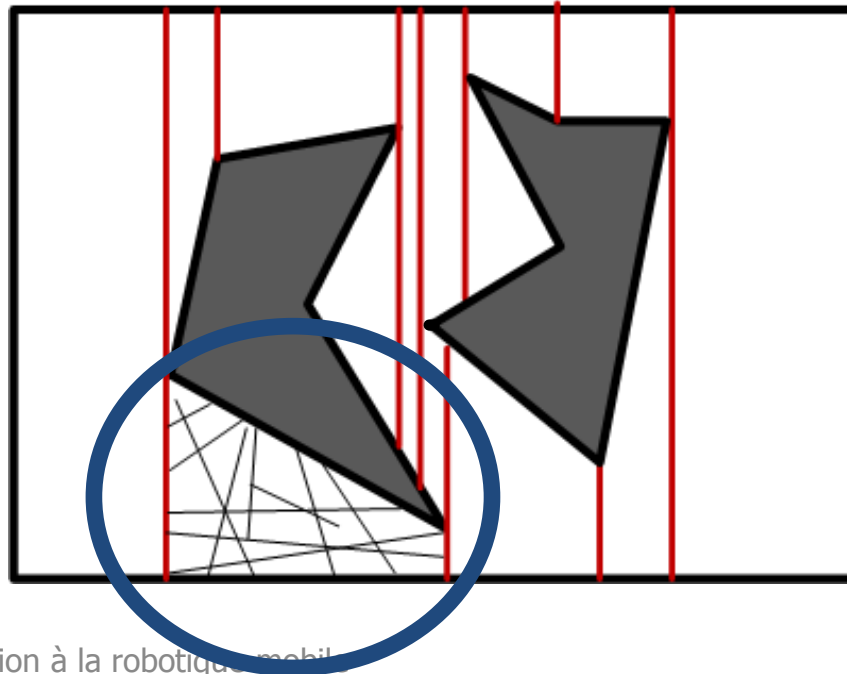
- Créer une  
carte routière



Étape 4:  
relie les **centres**  
aux points  
frontières

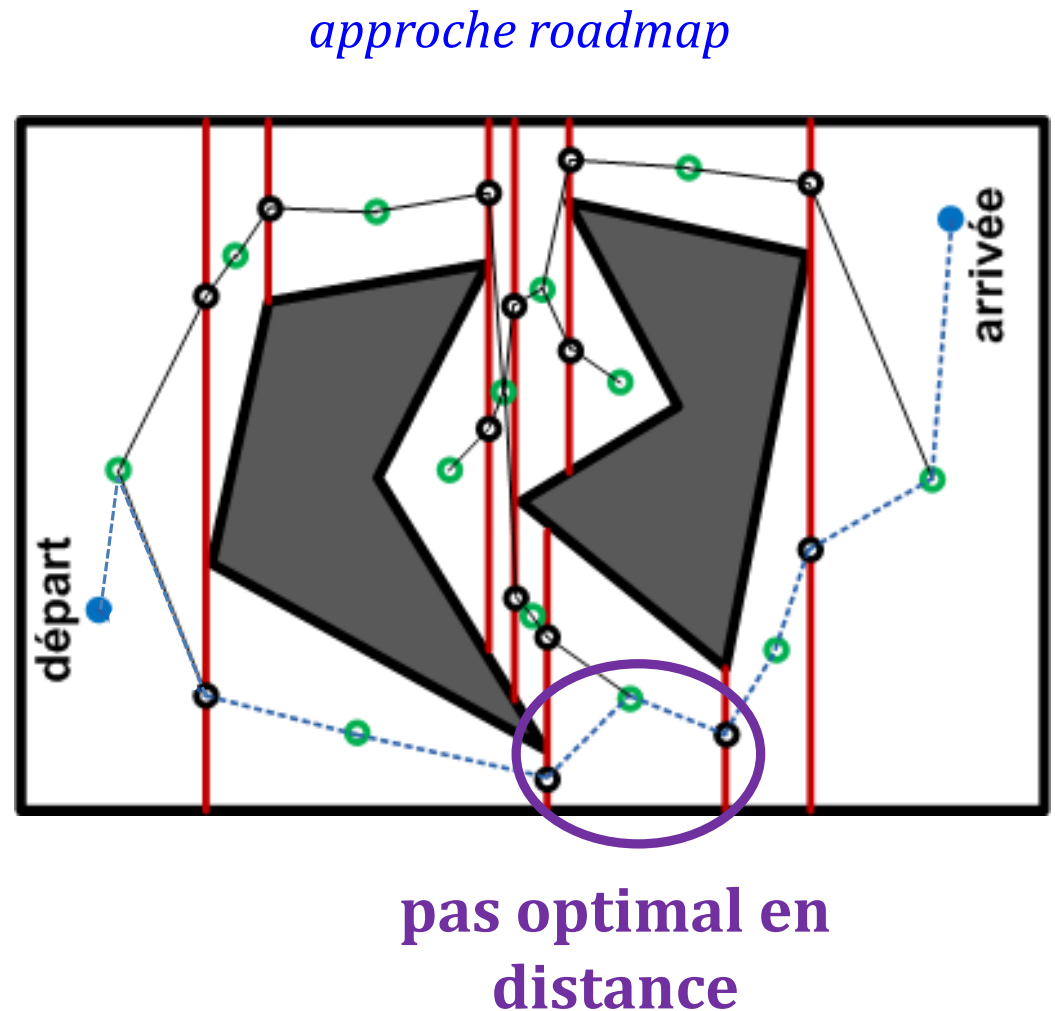
# Décomposition cellulaire verticale

- Chaque cellule est convexe
  - ce qui veut dire que dans une cellule, on peut rejoindre n'importe quel autre point à l'intérieur, en ligne droite.



# Décomposition cellulaire verticale

- Planification :  
recherche du chemin  
le plus court dans le  
graphe
- Donne des trajectoires  
qui sont en général  
loin des obstacles
- Création de la carte :  
complexité  $O(n \log n)$



# Champs de potentiels

---

- Imaginez que :
  - **robot** est une particule **+**
  - **obstacles** sont aussi chargés **+** (répulsion)
  - **but** est chargé **-** (attraction)
- Trajectoire est le chemin parcouru par la particule (robot) dans le champs de potentiel

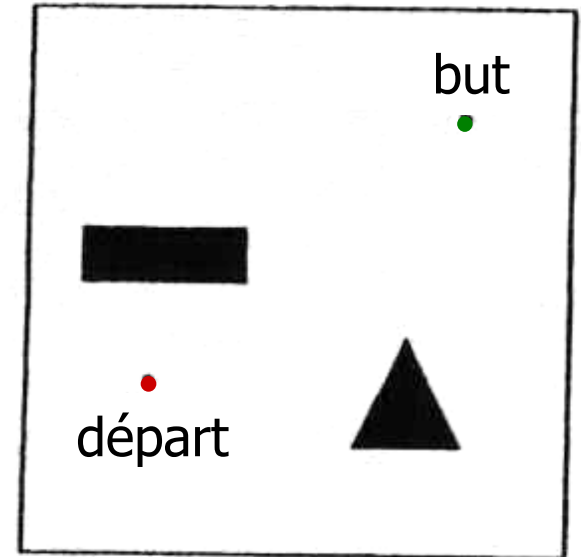
# Champs de potentiels

Calcul des forces de répulsions et d'attraction

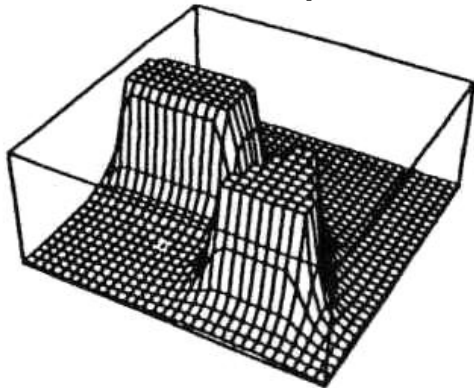
$$U(q) = U_{but}(q) + \sum U_{obstacles}(q)$$

$$F = -\vec{\nabla} U(q) = \begin{bmatrix} \frac{\partial}{\partial x} U(q) \\ \frac{\partial}{\partial y} U(q) \end{bmatrix}$$

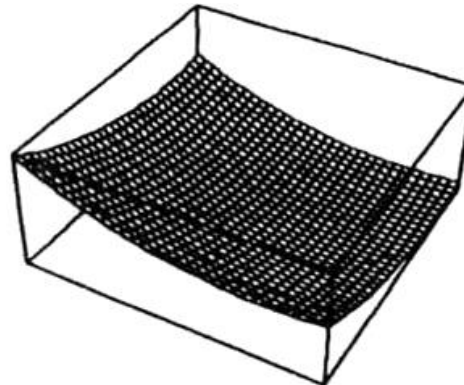
gradient  $\vec{\nabla}$  indique la direction maximale de croissance d'un champ scalaire



potentiel de répulsion

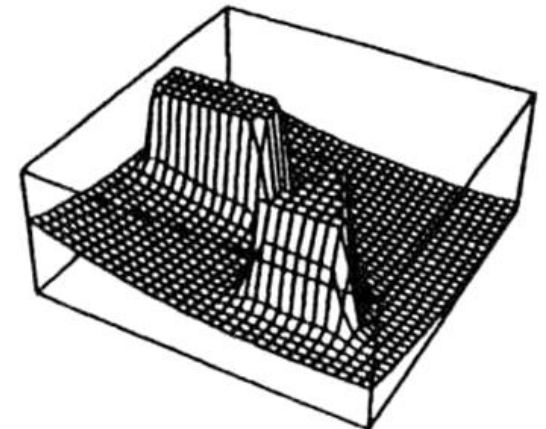


potentiel d'attraction



+

=



$$\sum U_{obstacles}(q)$$

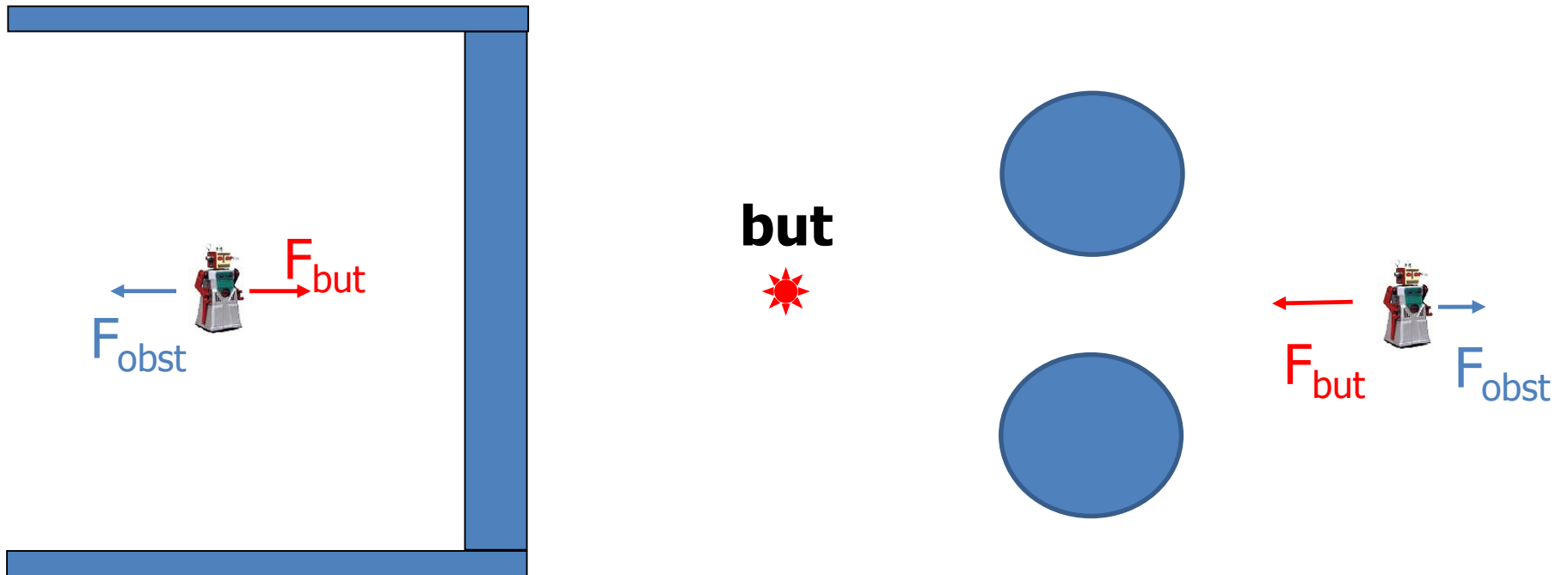
$$U_{but}(q)$$

$$U(q)$$



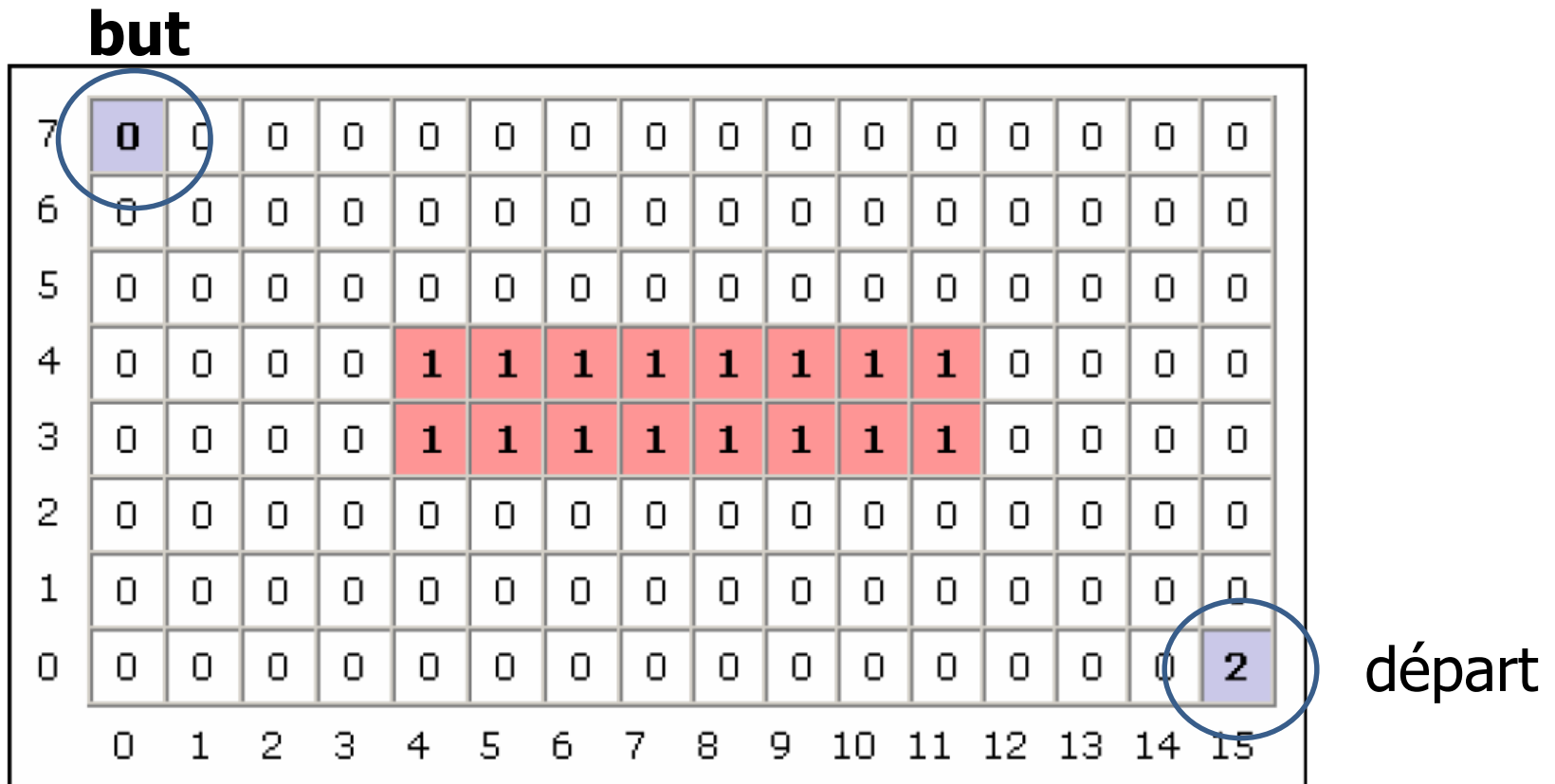
# Champs de potentiels

- Tombé en partie en désuétude parce que sensible aux minimums locaux.



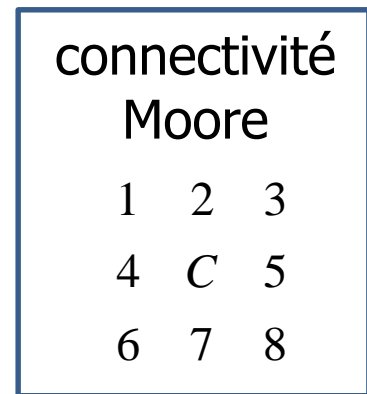
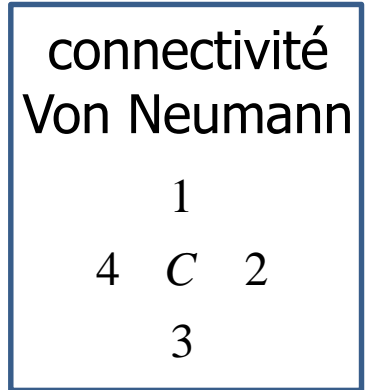
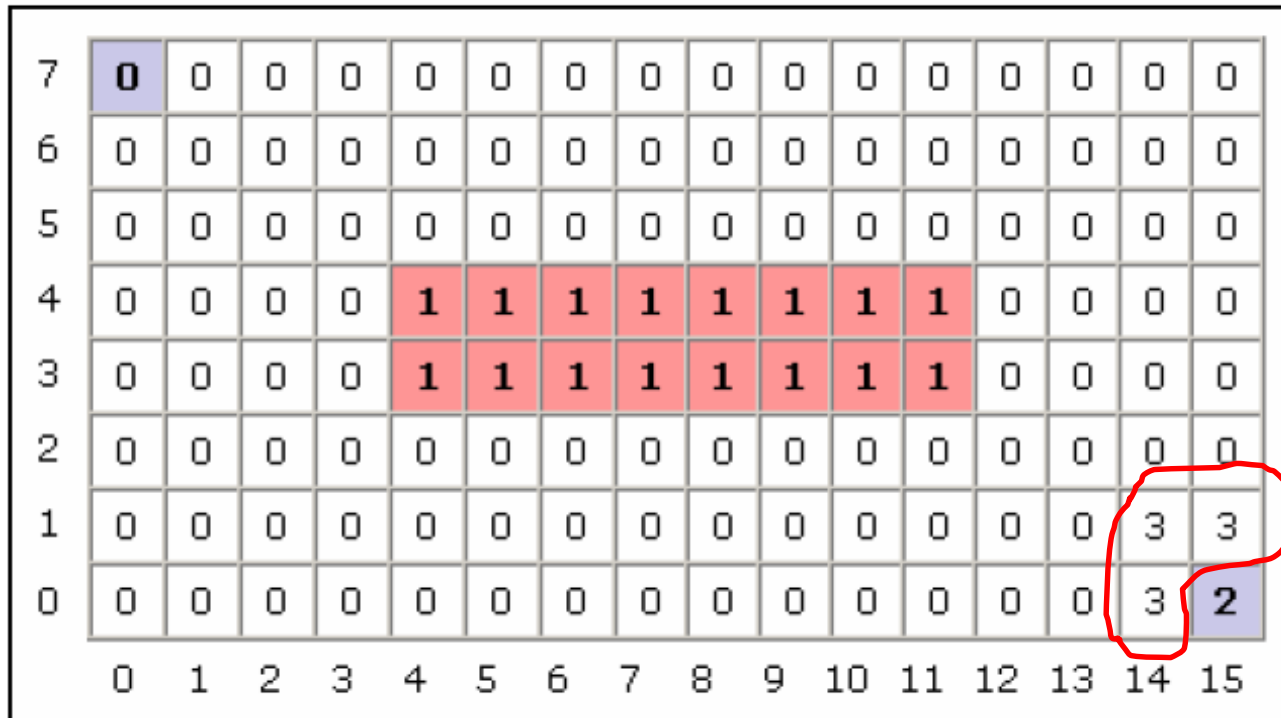
# Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ



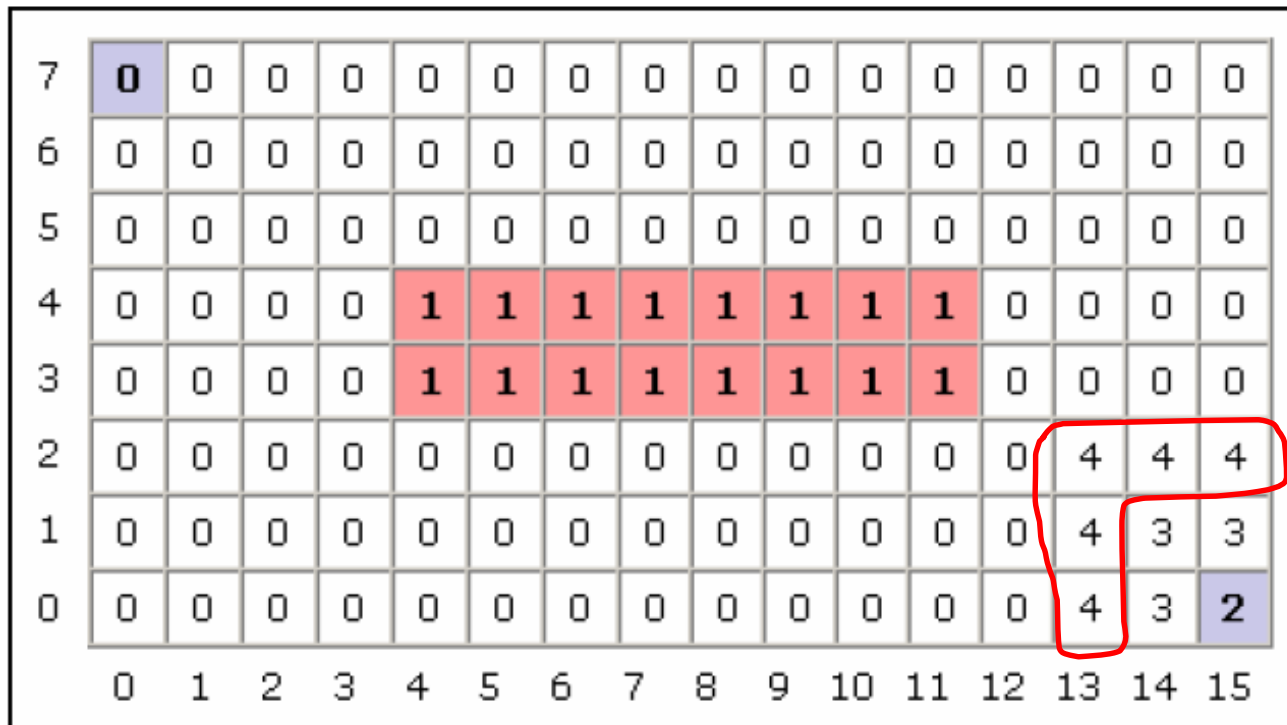
# Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ



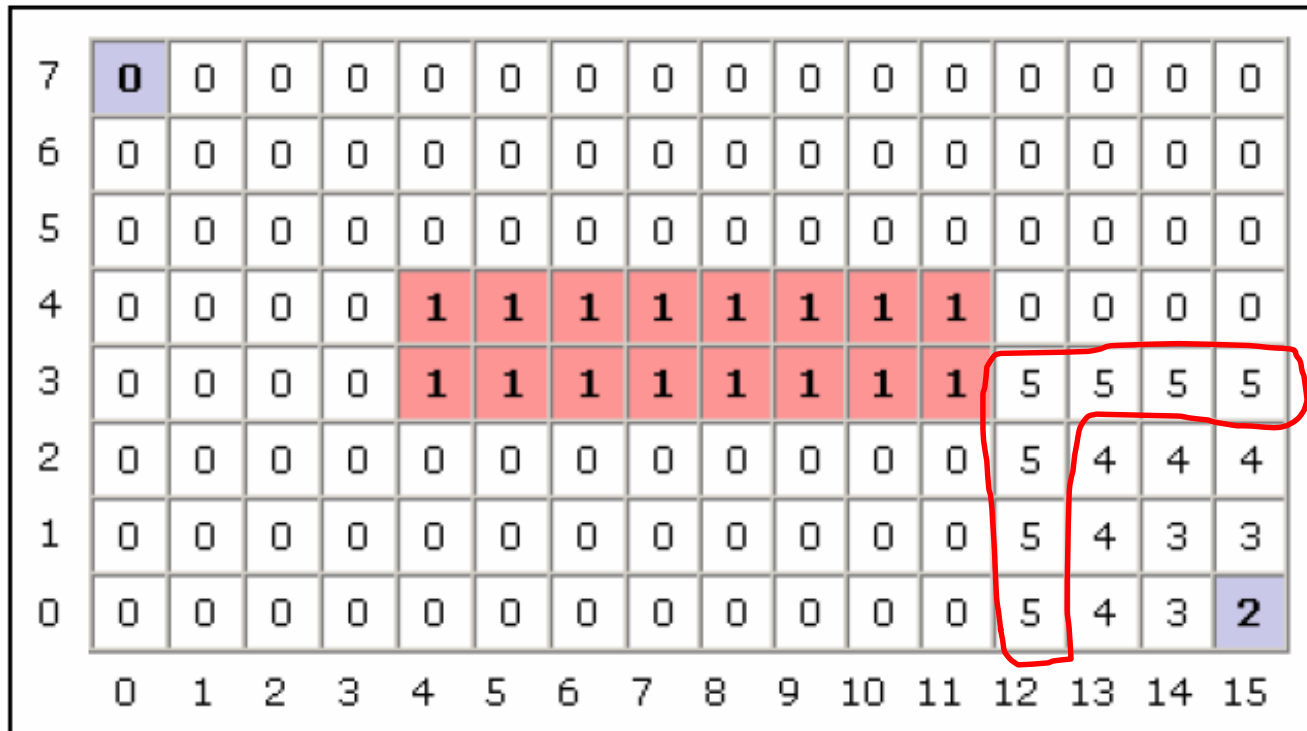
# Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ



# Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ



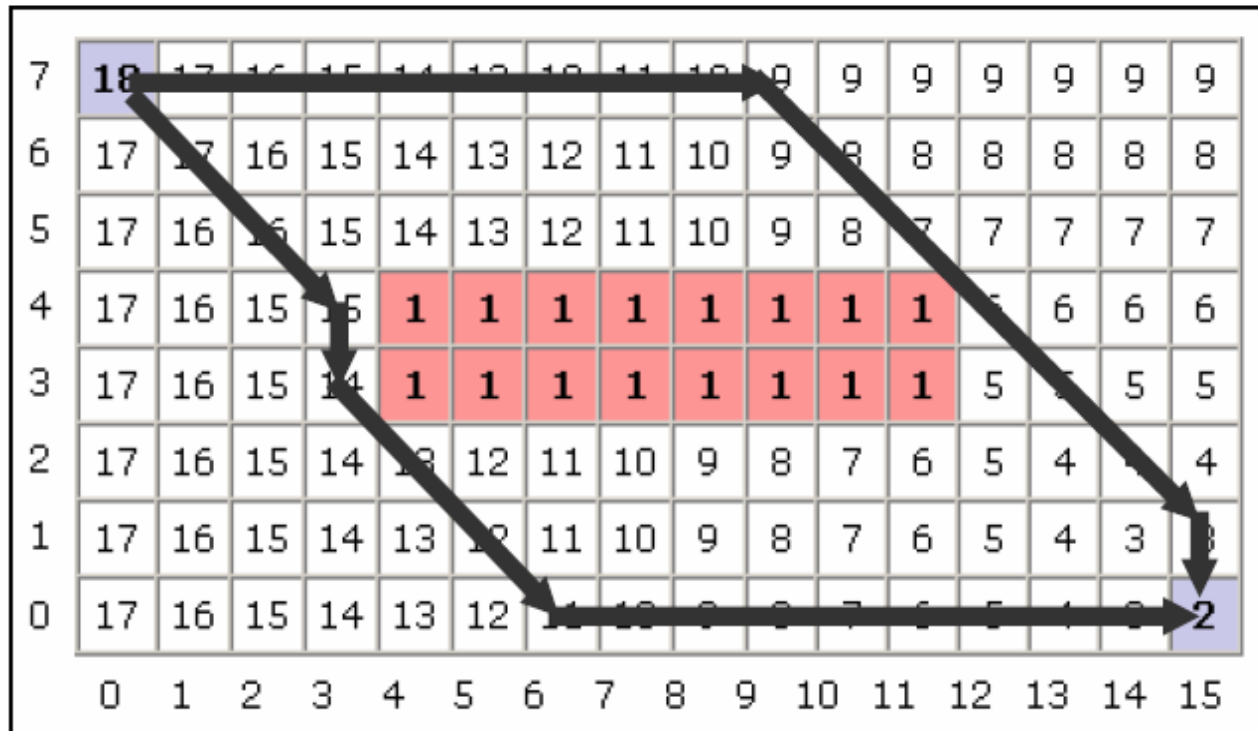
# Planification par front d'onde

- Rempli jusqu'à plein
  - les 0 indiquent les endroits non-accessibles du point de départ.

7	<b>18</b>	17	16	15	14	13	12	11	10	9	9	9	9	9	9	
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	
4	17	16	15	15	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	6	6	6	
3	17	16	15	14	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	5	5	5	
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	<b>2</b>
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

# Planification par front d'onde

- La trajectoire consiste **à partir du but** de choisir de façon vorace les cases les plus petites.



2 des  
nombreuses  
trajectoires  
possibles

# Planification par front d'onde

- Si vous faites le choix vorace des nœuds **à partir du départ** 😞, vous n'aurez pas nécessairement un chemin le plus court

7	18	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	1	1	1	1	1	1	1	1	6	6	6	6
3	17	16	15	14	1	1	1	1	1	1	1	1	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



# Méthodes probabilistes

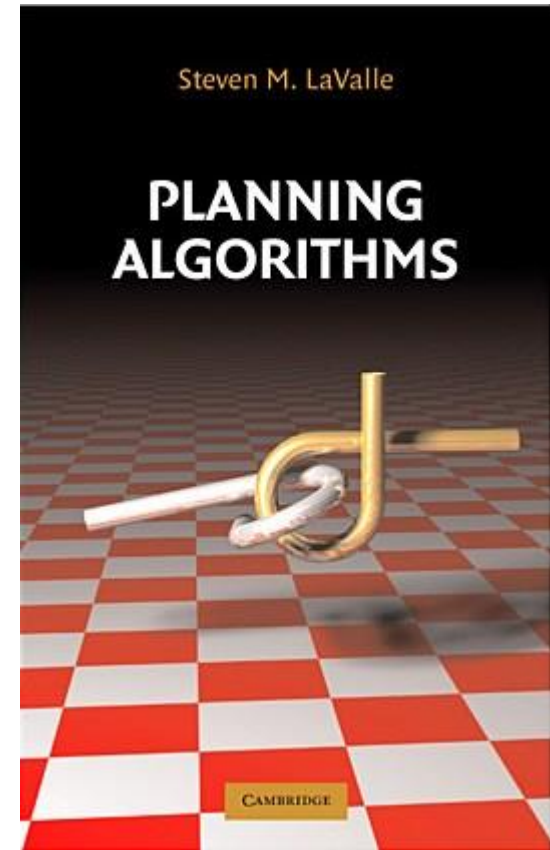
---

- Basée sur l'échantillonnage au hasard de configurations dans  $C_{libre}$ .
- Besoin d'une fonction de détection de collision
  - plus besoin de calculer le  $C_{libre}$  au complet
  - fonction très optimisée
- Approches
  - **Requête simple** : RRT-Connect
    - basée sur les Rapidely-exploring Random Trees (RRT)
  - **Requête multiple** : Probabilistic Road Maps (RPM)

Handbook of Robotics, p. 112, Springer.

# Rapidly-exploring Random Trees (RRT)

- Pour recherche exploratoire dans des espaces à haute dimensionnalité
- Développé par Steven M. LaValle et James Kuffner
- Construire un arbre de façon incrémentale, pour minimiser la distance entre un point choisis au hasard et l'arbre



<http://planning.cs.uiuc.edu/>

# Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K  
   $q_{rand} \leftarrow \text{RAND\_CONF}()$   
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$   
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$   
  G.add_vertex( $q_{new}$ )  
  G.add_edge( $q_{near}, q_{new}$ )  
end
```

K : nombre d'itérations

G : graphe de l'arbre

- arbre au départ (1 seul sommet)

# Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K
```

```
   $q_{rand} \leftarrow \text{RAND\_CONF}()$ 
```

```
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$ 
```

```
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$ 
```

```
  G.add_vertex( $q_{new}$ )
```

```
  G.add_edge( $q_{near}, q_{new}$ )
```

```
end
```

$q_{rand}$   
● sommet tiré au hasard



# Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K
```

```
   $q_{rand} \leftarrow \text{RAND\_CONF}()$ 
```

```
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$ 
```

```
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$ 
```

```
  G.add_vertex( $q_{new}$ )
```

```
  G.add_edge( $q_{near}, q_{new}$ )
```

```
end
```



$q_{near}$  sommet le plus proche de l'arbre G

# Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K
```

```
   $q_{rand} \leftarrow \text{RAND\_CONF}()$ 
```

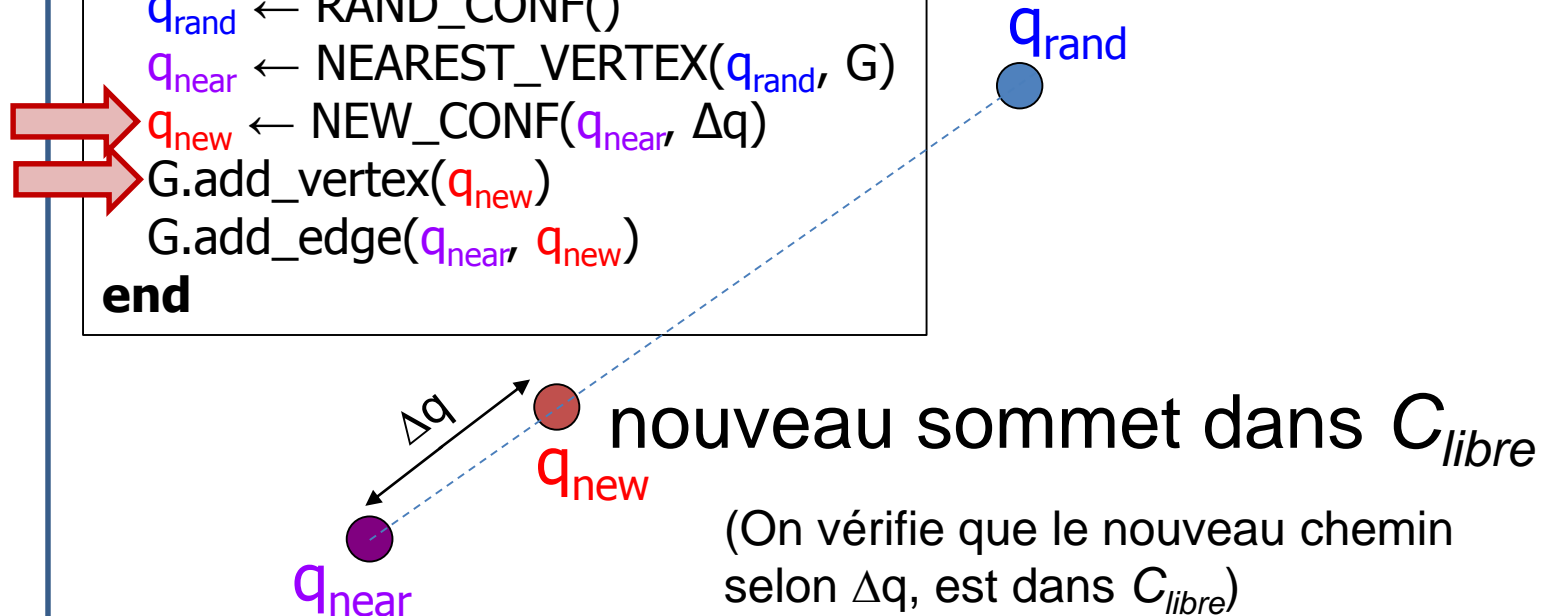
```
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$ 
```

```
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$ 
```

```
  G.add_vertex( $q_{new}$ )
```

```
  G.add_edge( $q_{near}, q_{new}$ )
```

```
end
```



# Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K
```

```
   $q_{rand} \leftarrow \text{RAND\_CONF}()$ 
```

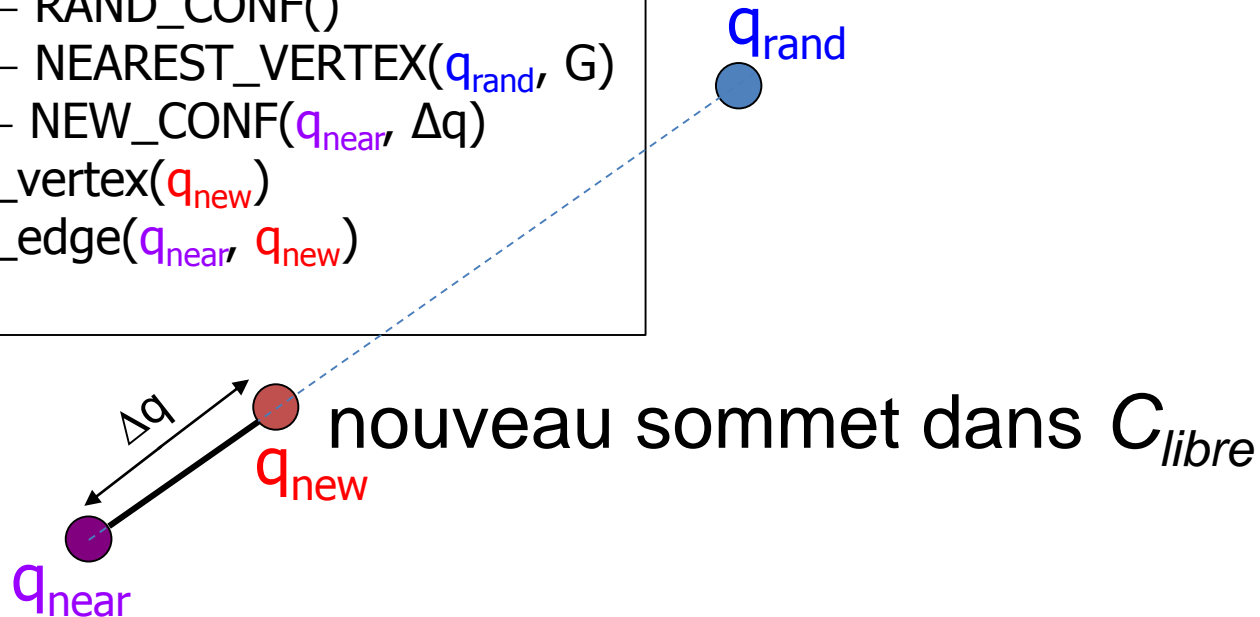
```
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$ 
```

```
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$ 
```

```
  G.add_vertex( $q_{new}$ )
```

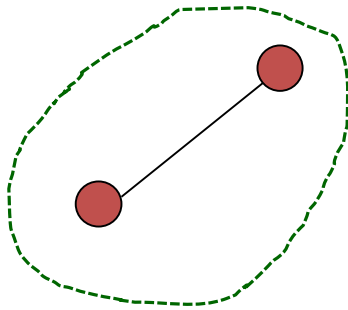
```
  G.add_edge( $q_{near}, q_{new}$ )
```

```
end
```



# Rapidly-exploring Random Trees (RRT)

Arbre G

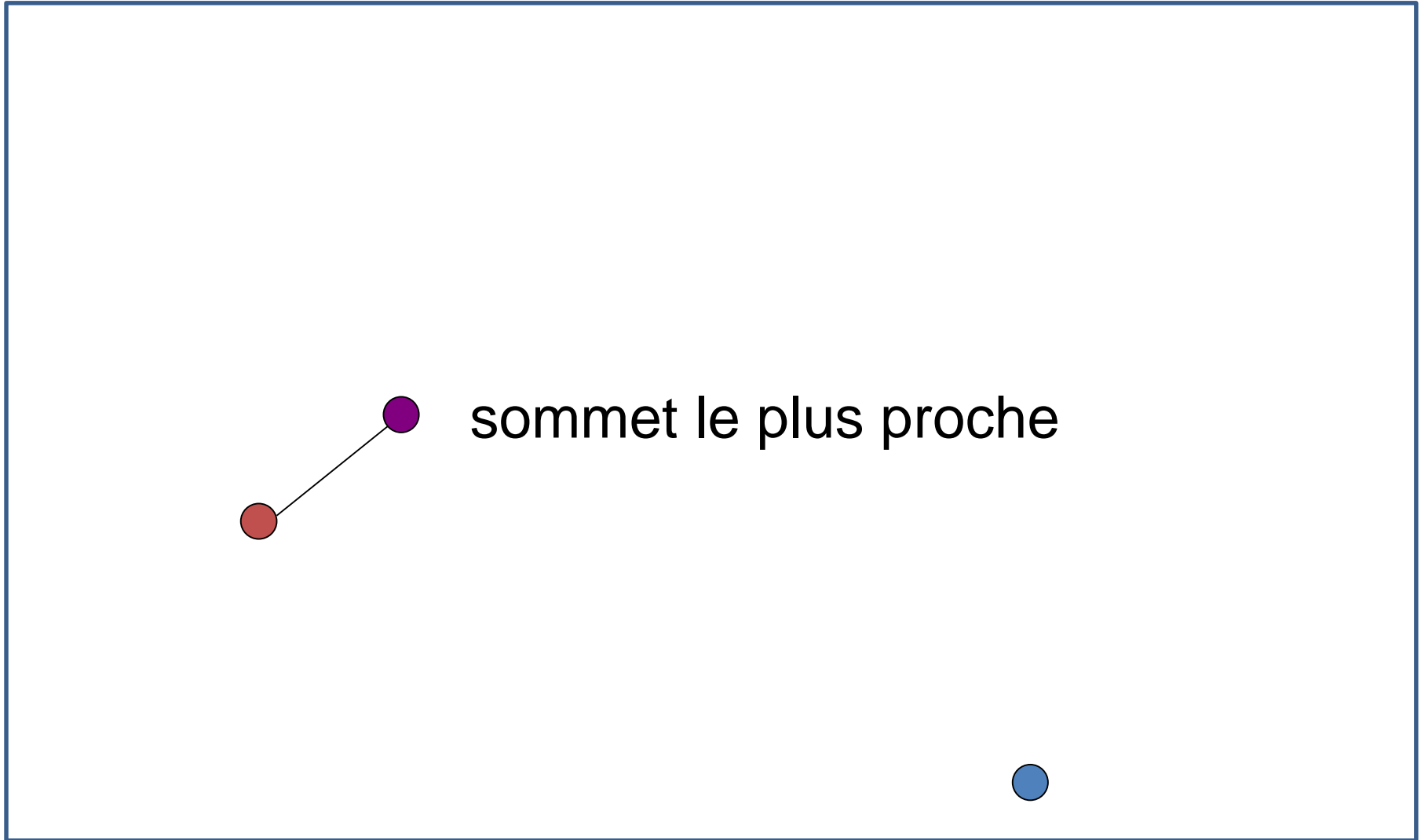


autre sommet tiré au hasard





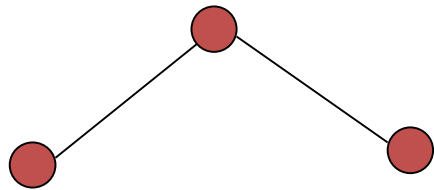
# Rapidly-exploring Random Trees (RRT)



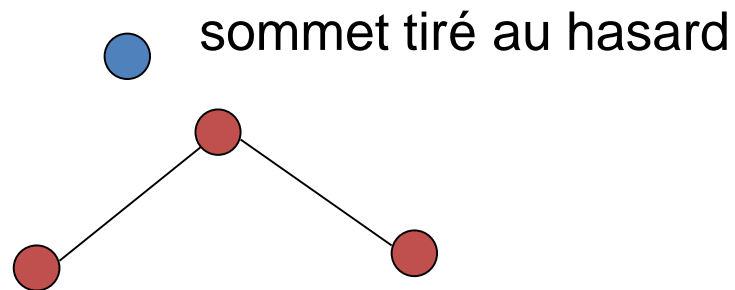
# Rapidly-exploring Random Trees (RRT)



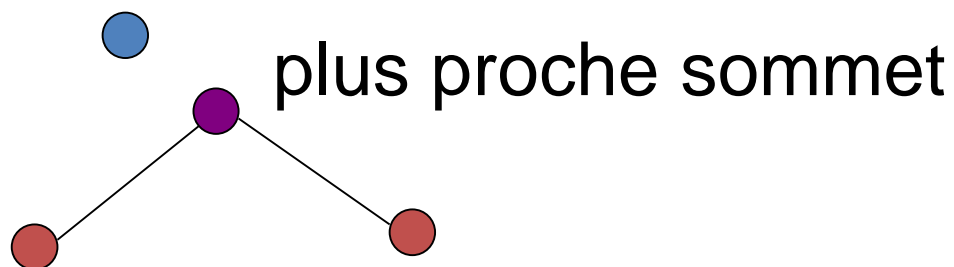
# Rapidly-exploring Random Trees (RRT)



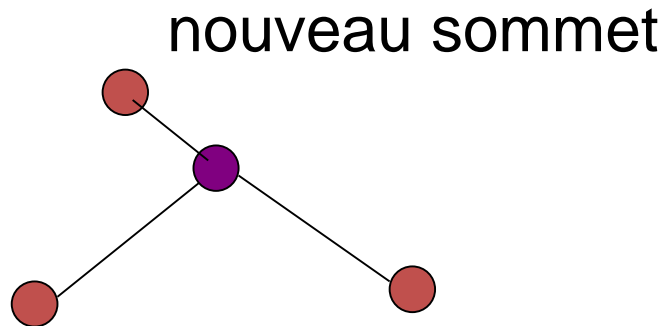
# Rapidly-exploring Random Trees (RRT)



# Rapidly-exploring Random Trees (RRT)



# Rapidly-exploring Random Trees (RRT)



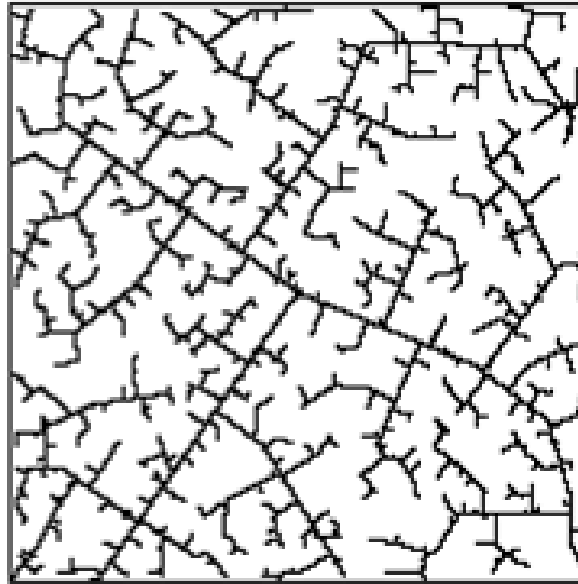
et ainsi de suite...

# Rapidly-exploring Random Trees (RRT)

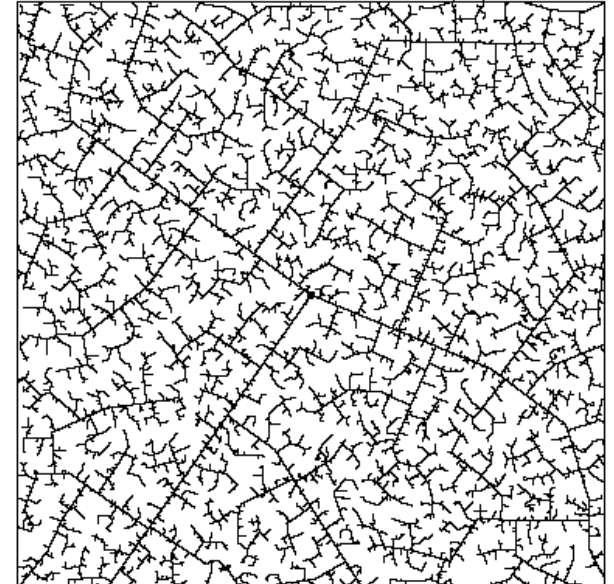
- Tend à bien remplir l'espace vide



45 itérations



390 itérations



2345 itérations

# RRT-Connect

---

- Croître deux arbres, l'un partant de la configuration de départ et l'autre d'arrivée
- À chaque tirage, on essaie de connecter un des arbres avec l'autre, de façon vorace



# RRT-Connect: exemple

---

**Phase croissance de l'arbre**

● Départ



○ Arrivée

# RRT-Connect: exemple

---

Phase croissance de l'arbre

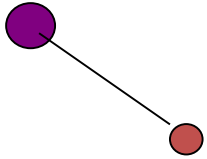


● sommet tiré au hasard



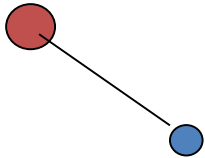
# RRT-Connect: exemple

Phase croissance de l'arbre



# RRT-Connect: exemple

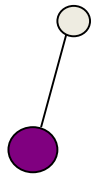
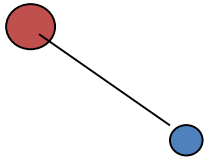
---



On essaie maintenant de  
connecter l'arbre du bas vers  
l'arbre du haut, de façon  
vorace (greedy)

# RRT-Connect: exemple

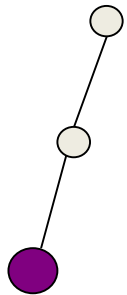
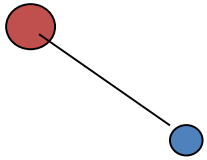
---



**Phase connexion des arbres**

# RRT-Connect: exemple

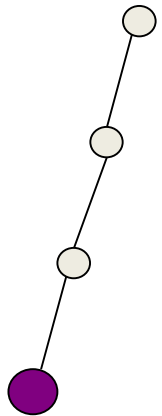
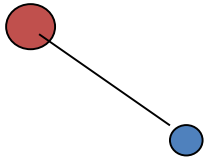
---



**Phase connexion des arbres**

# RRT-Connect: exemple

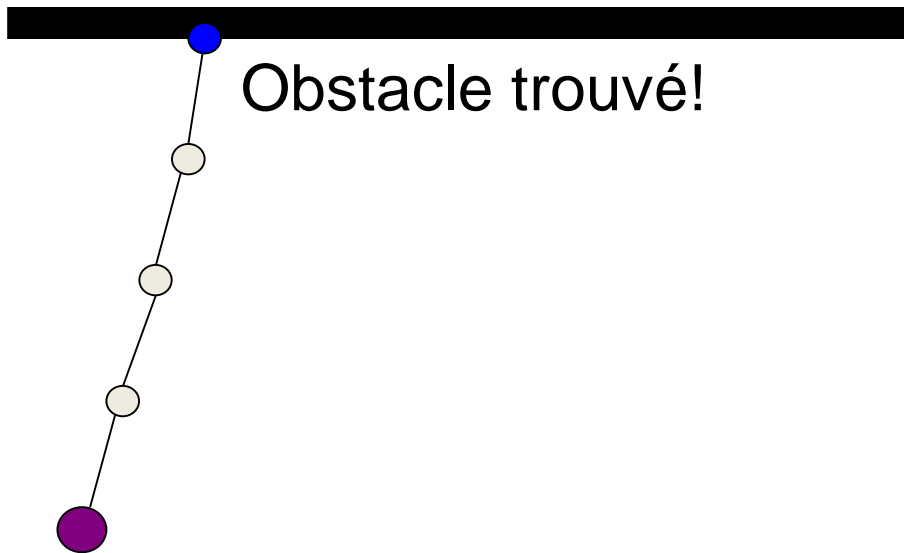
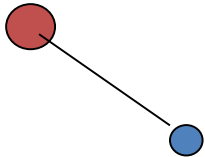
---



**Phase connexion des arbres**

# RRT-Connect: exemple

---

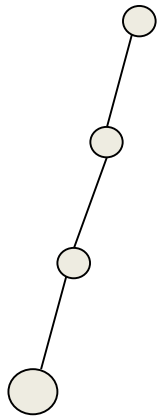
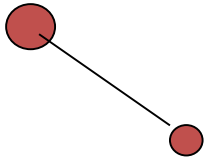


**Phase connexion des arbres**



# RRT-Connect: exemple

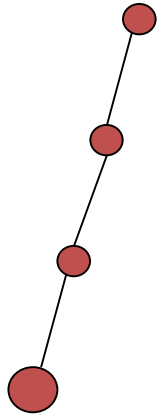
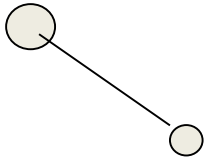
---



On inverse les rôles!

# RRT-Connect: exemple

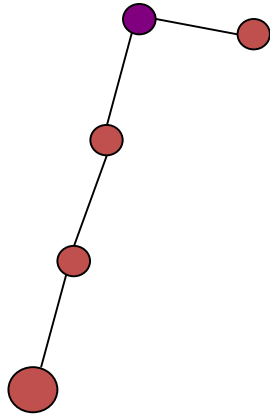
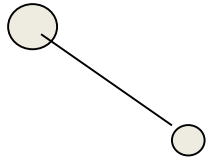
---



On inverse les rôles!

**Phase croissance de l'arbre**

# RRT-Connect: exemple

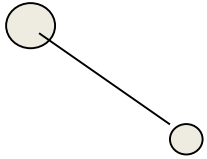


on tire un point au hasard, ajoute  
un sommet à l'arbre du bas

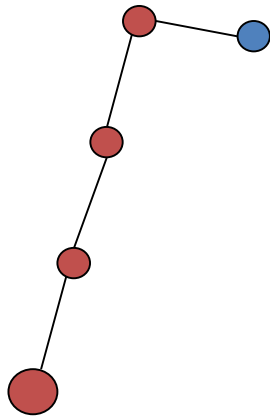
**Phase croissance de l'arbre**

# RRT-Connect: exemple

## Phase connexion des arbres

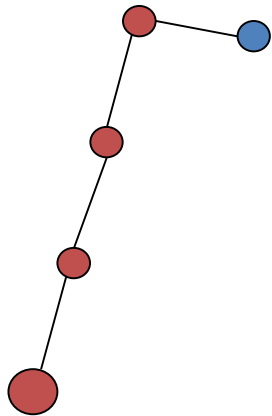
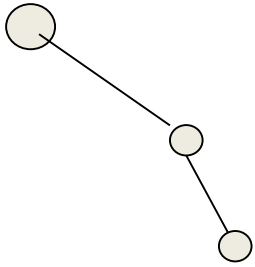


Et de façon vorace on croît l'arbre du haut



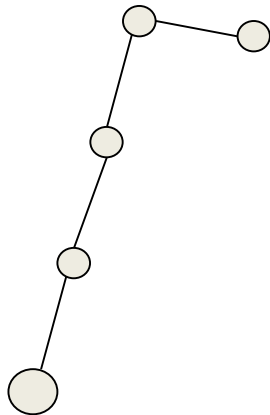
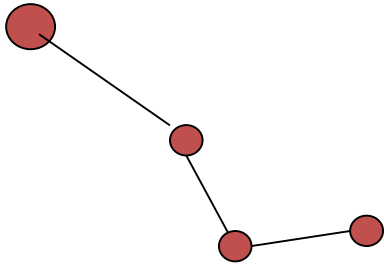
# RRT-Connect: exemple

Phase connexion des arbres



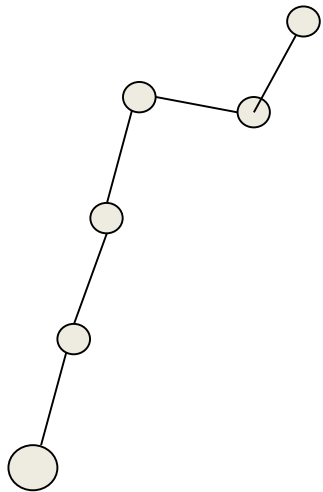
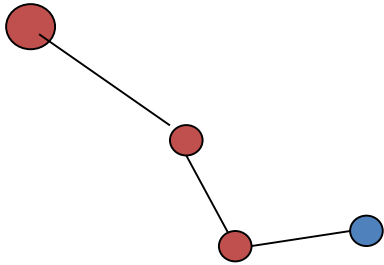
# RRT-Connect: exemple

Phase croissance de l'arbre



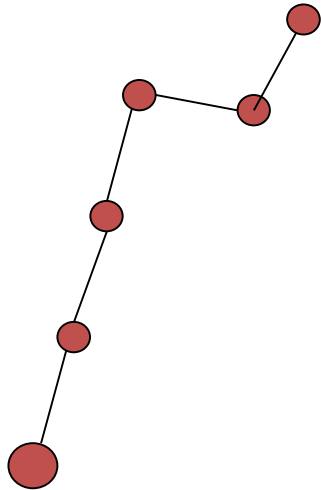
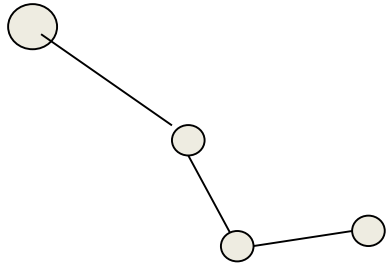
# RRT-Connect: exemple

---



**Phase connexion des arbres**

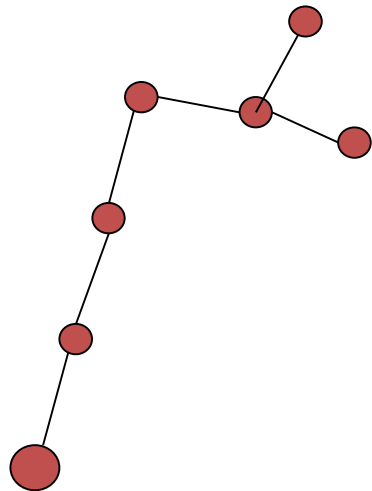
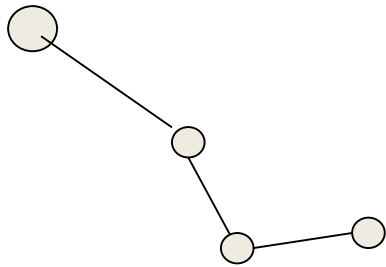
# RRT-Connect: exemple



**Phase croissance de l'arbre**



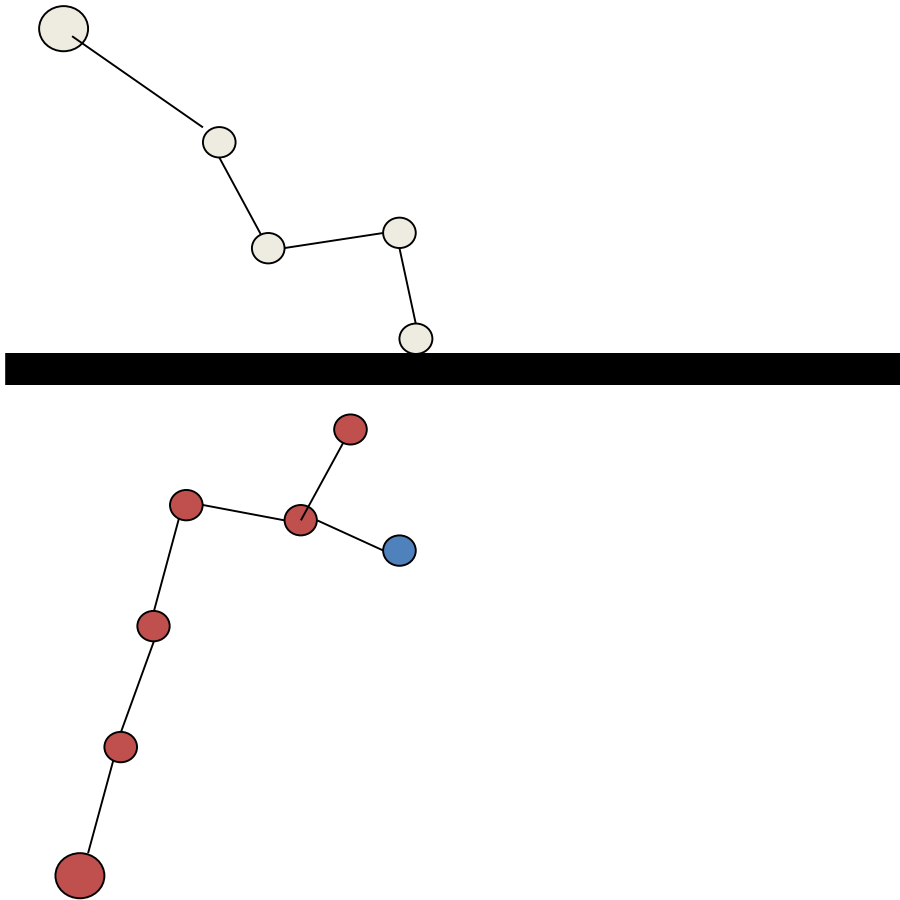
# RRT-Connect: exemple



**Phase croissance de l'arbre**

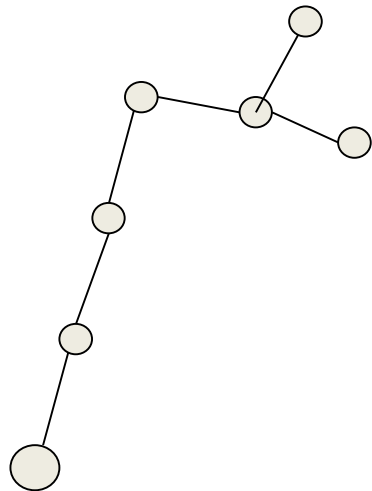
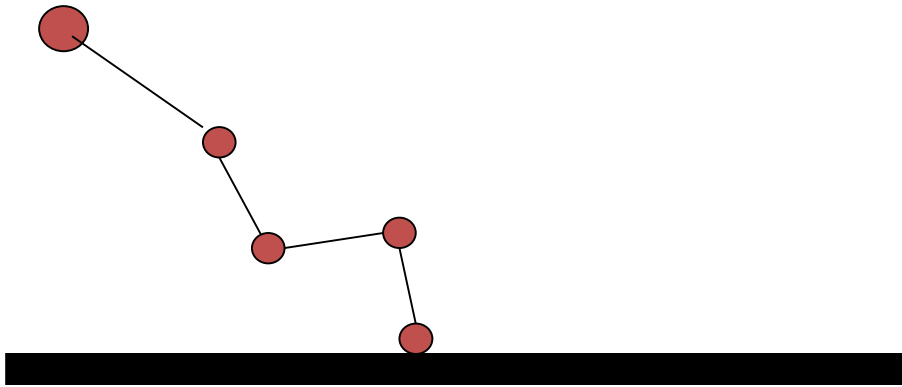
# RRT-Connect: exemple

Phase connexion des arbres



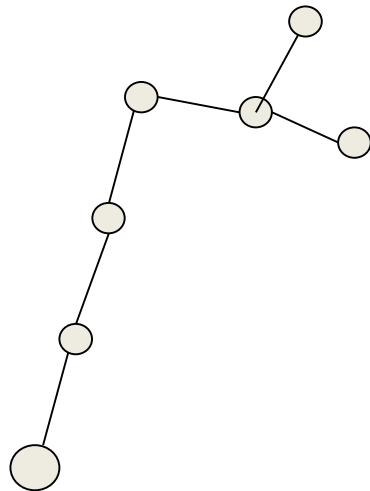
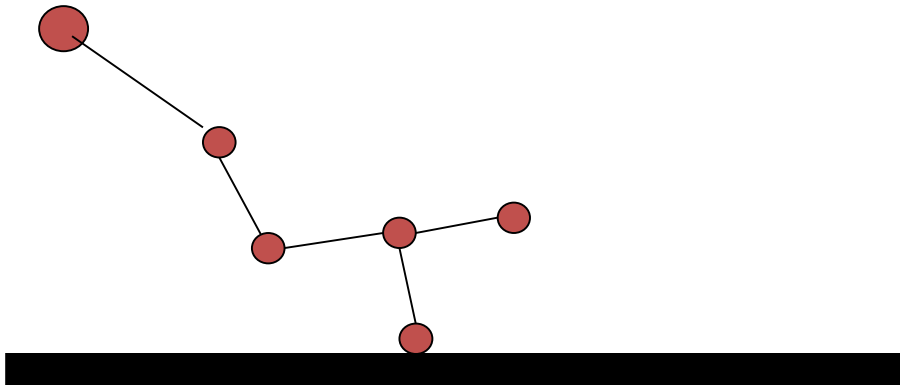
# RRT-Connect: exemple

Phase croissance de l'arbre

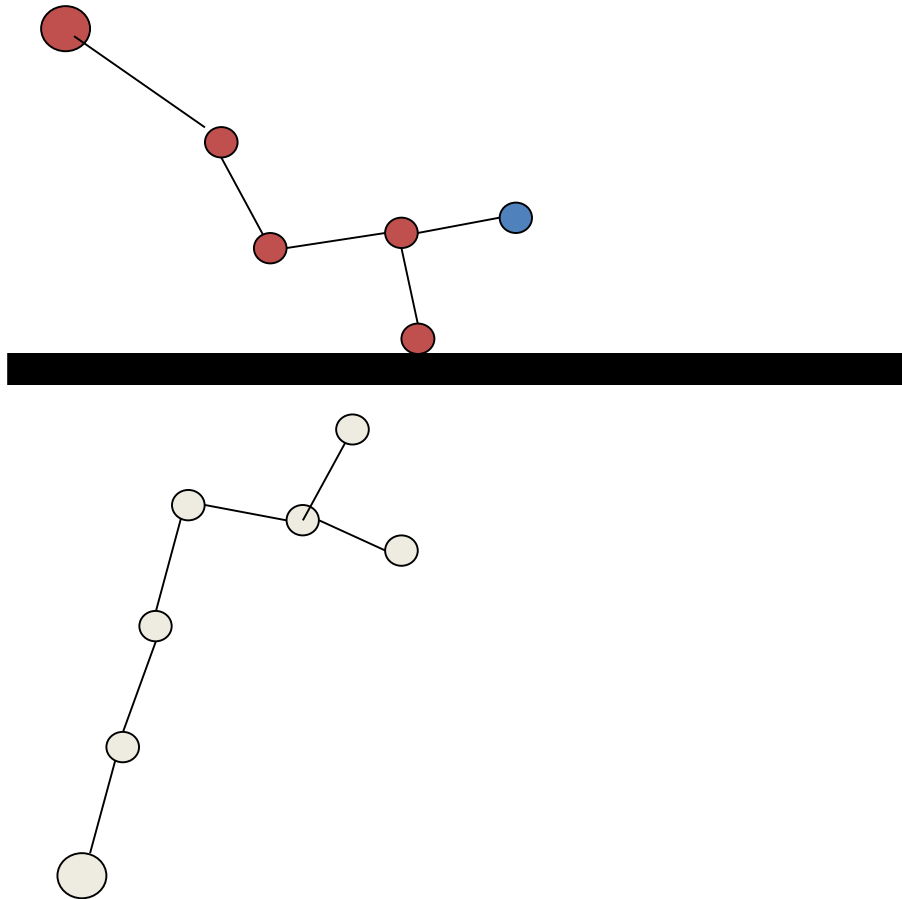


# RRT-Connect: exemple

Phase croissance de l'arbre

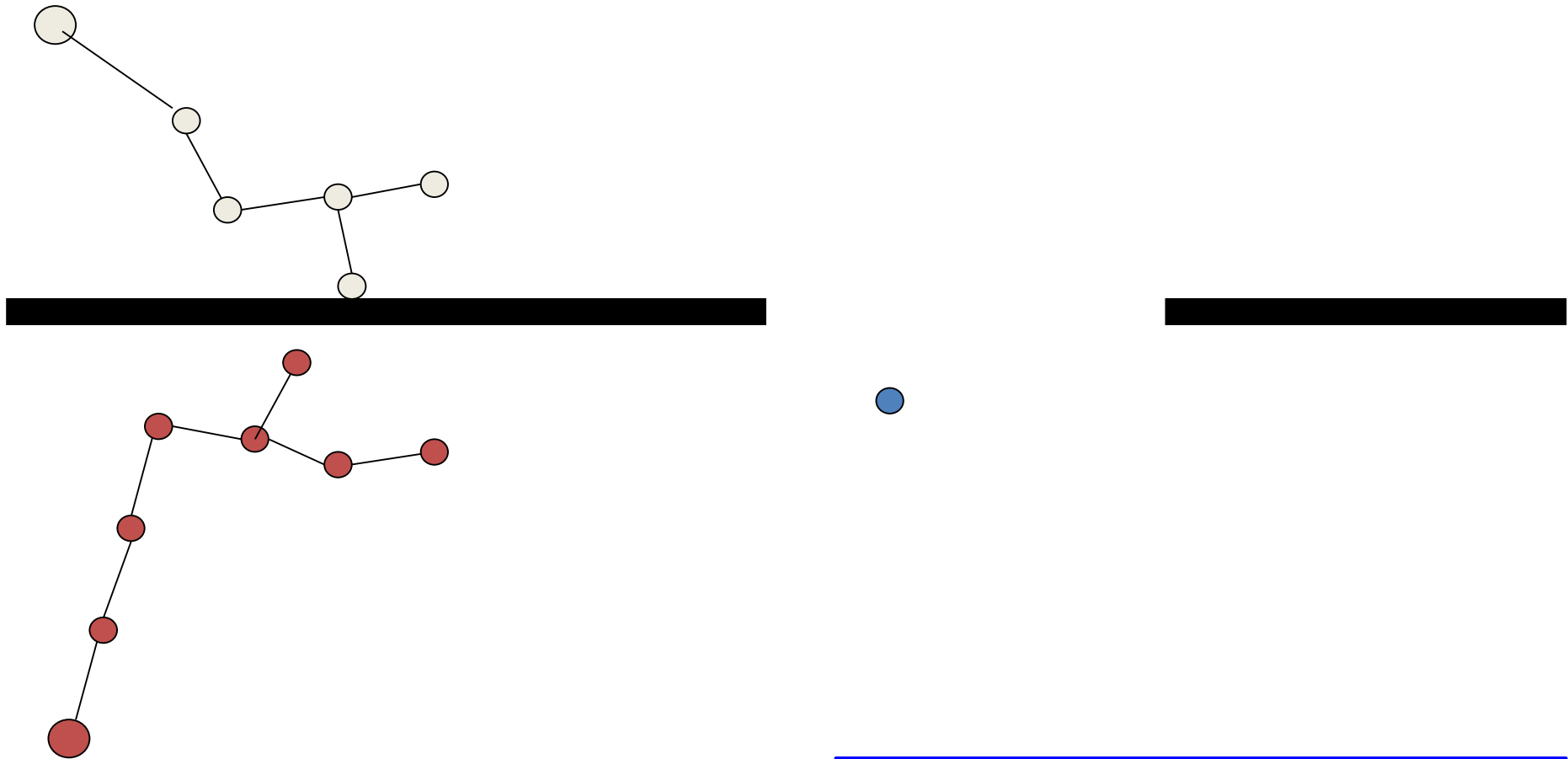


# RRT-Connect: exemple



**Phase connexion des arbres**

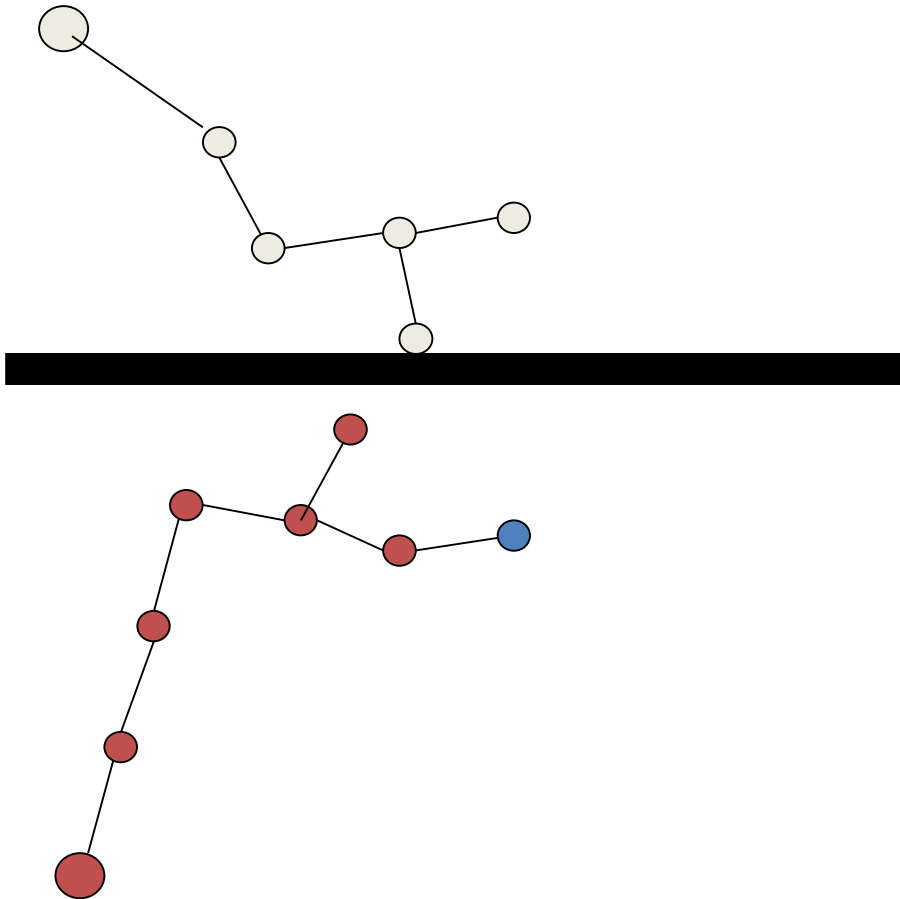
# RRT-Connect: exemple



**Phase croissance de l'arbre**

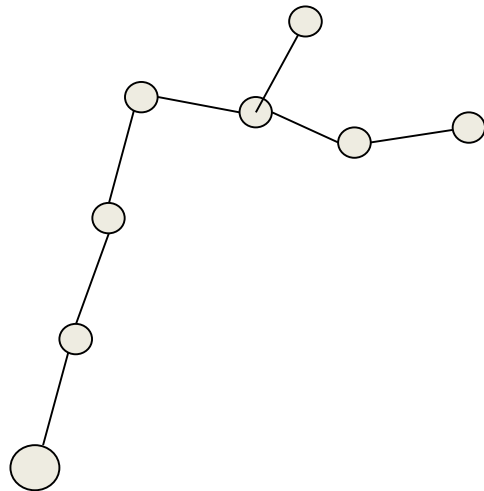
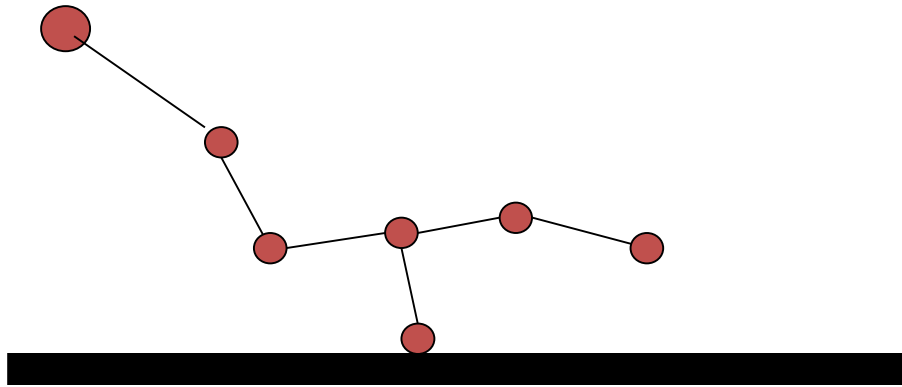
# RRT-Connect: exemple

Phase connexion des arbres



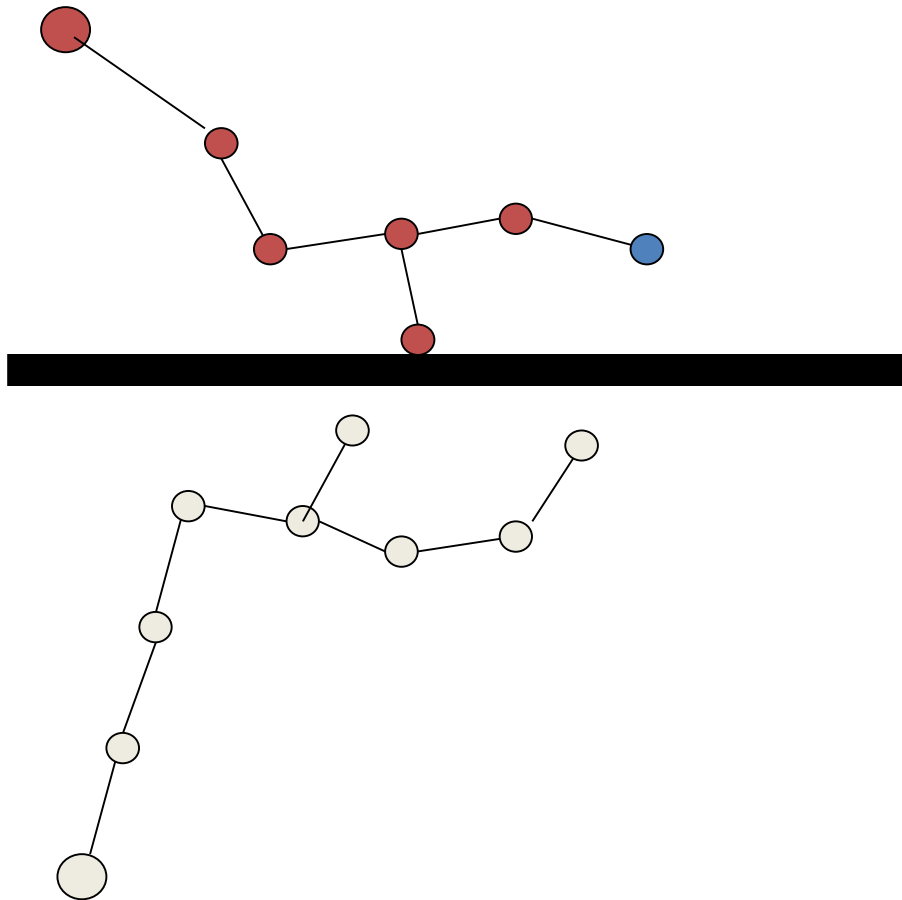
# RRT-Connect: exemple

Phase croissance de l'arbre



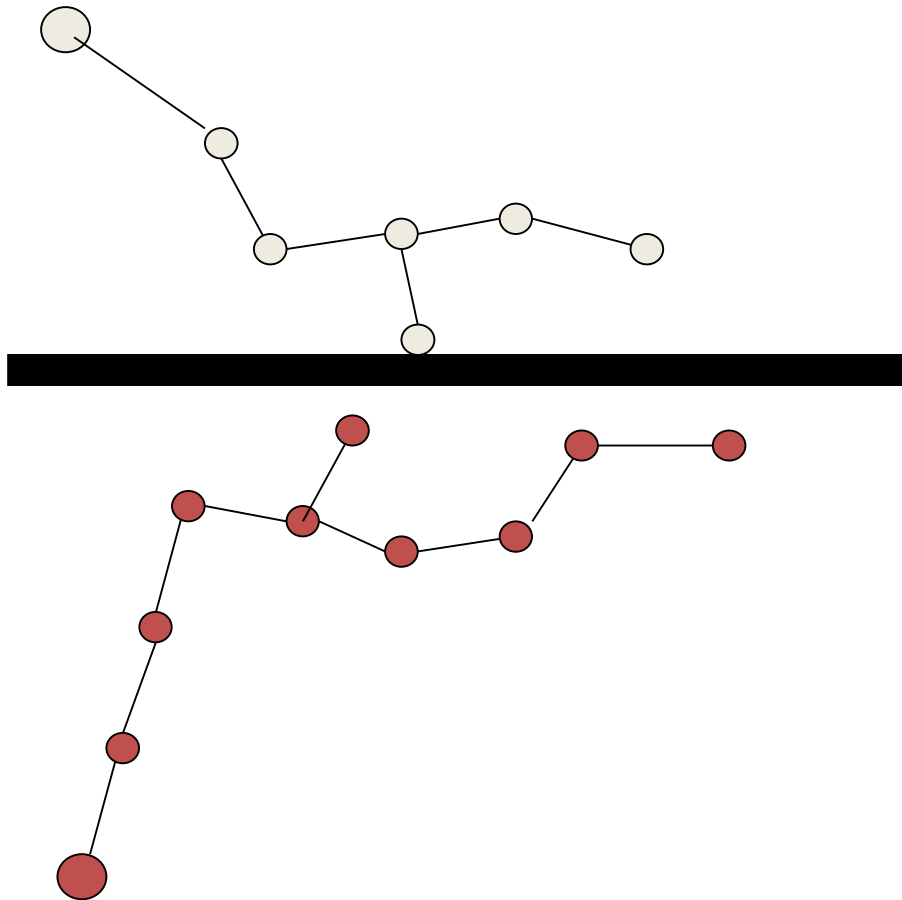


# RRT-Connect: exemple



**Phase connexion des arbres**

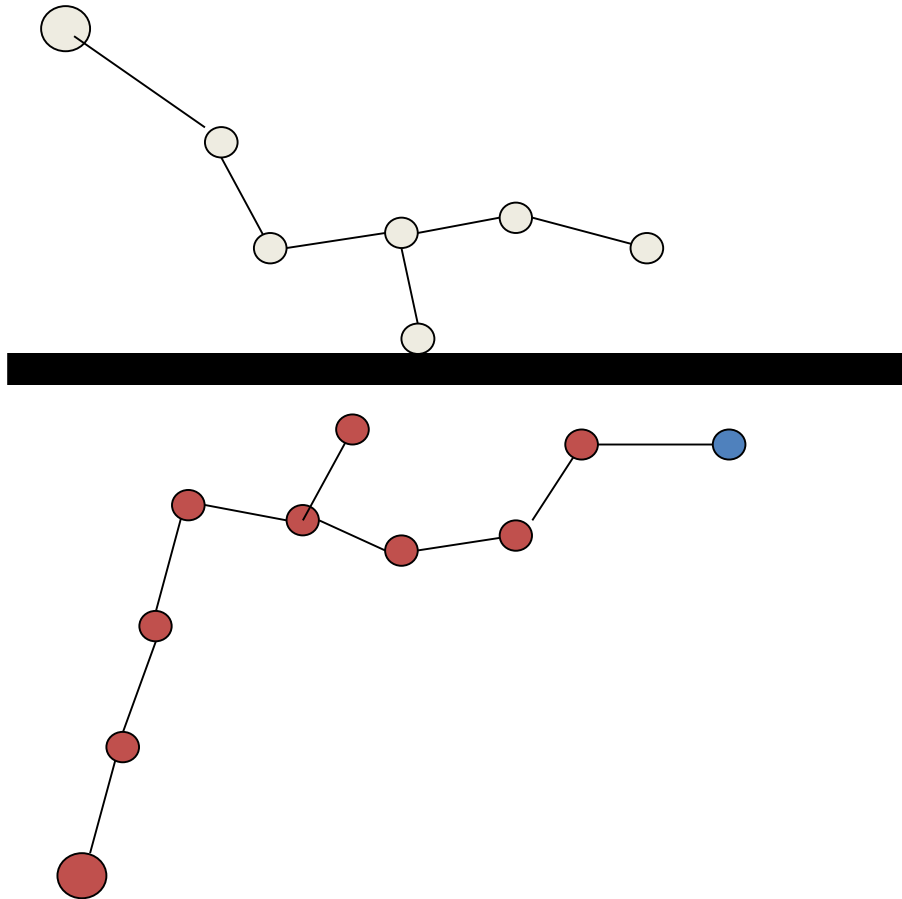
# RRT-Connect: exemple



**Phase croissance de l'arbre**

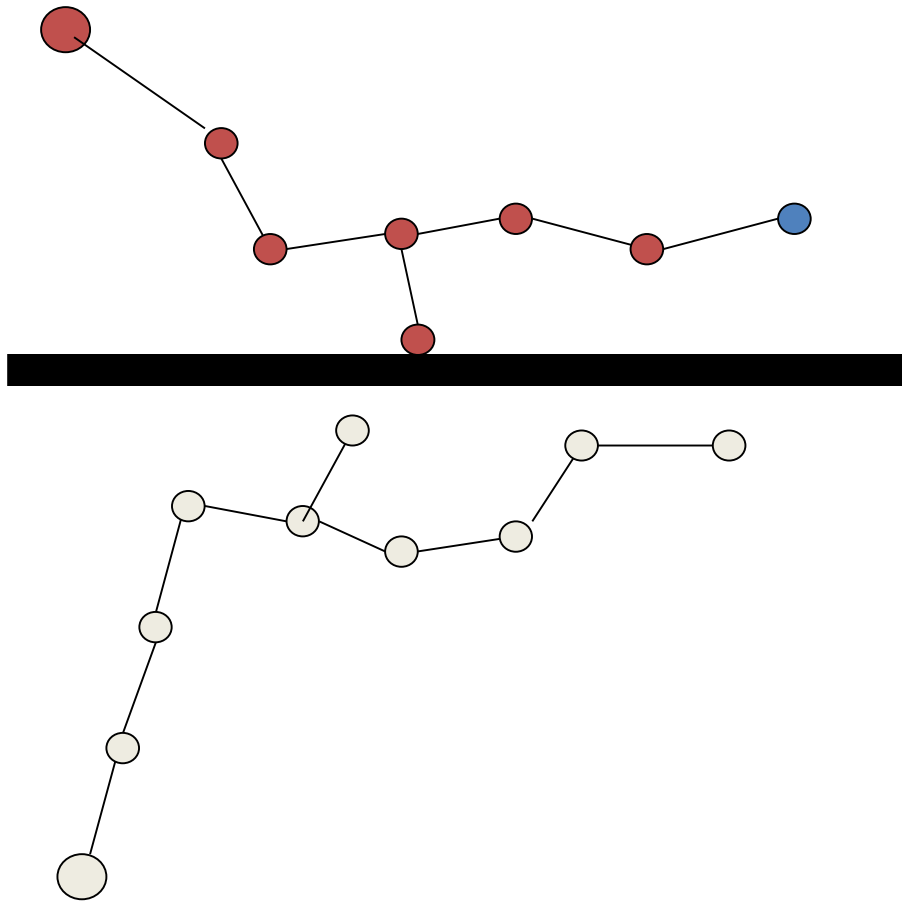
# RRT-Connect: exemple

Phase connexion des arbres



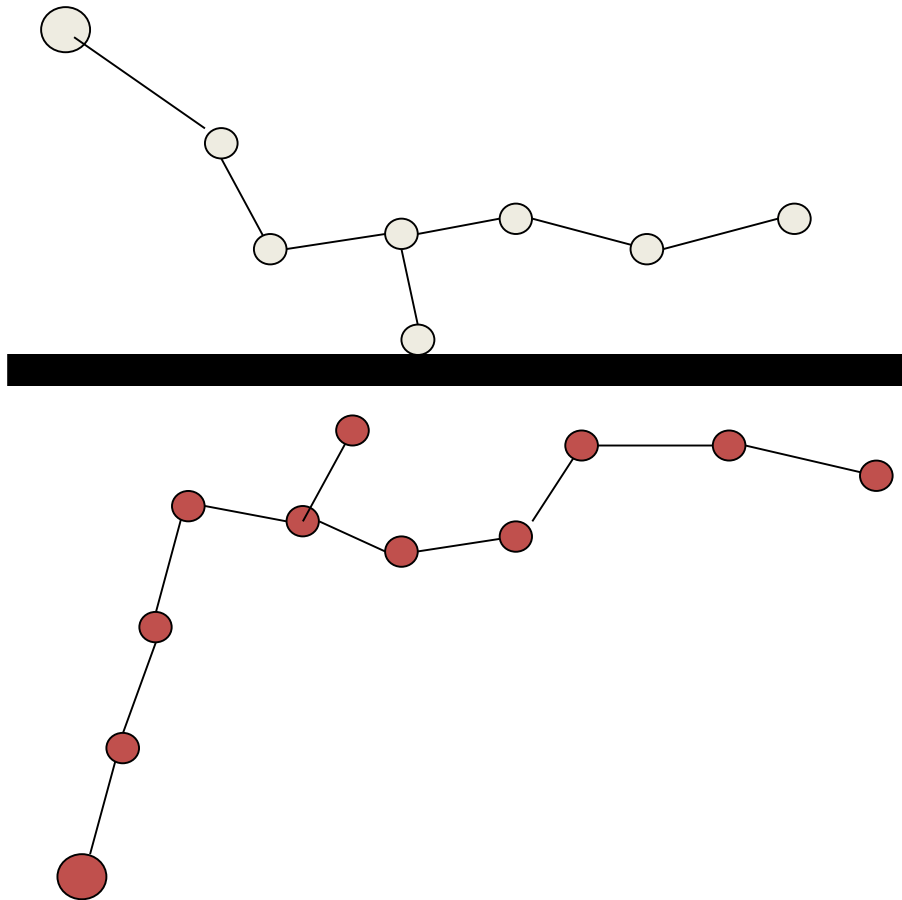


# RRT-Connect: exemple



**Phase connexion des arbres**

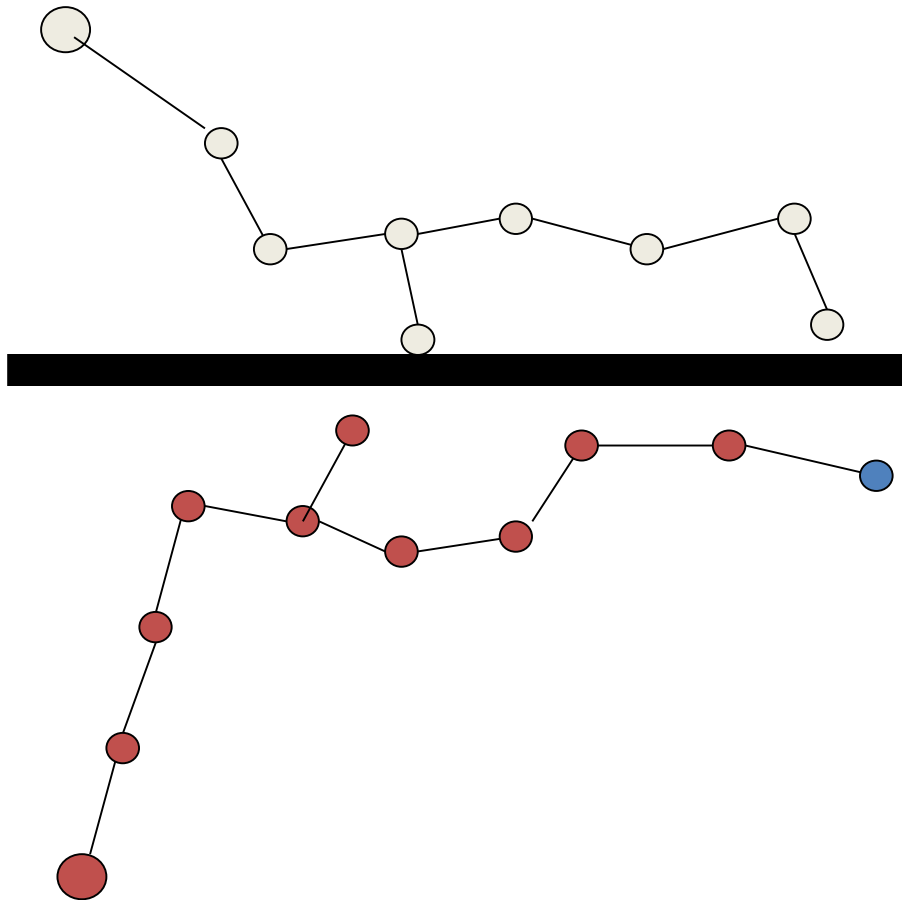
# RRT-Connect: exemple



Phase croissance de l'arbre

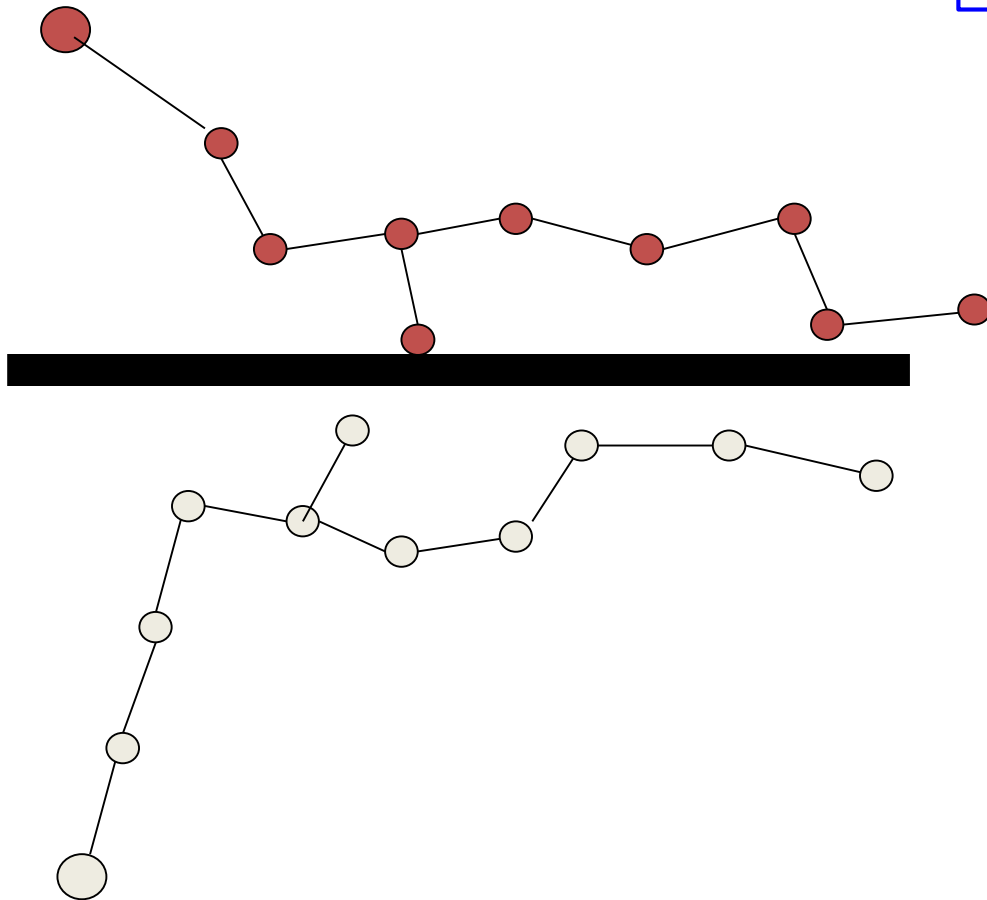
# RRT-Connect: exemple

Phase connexion des arbres



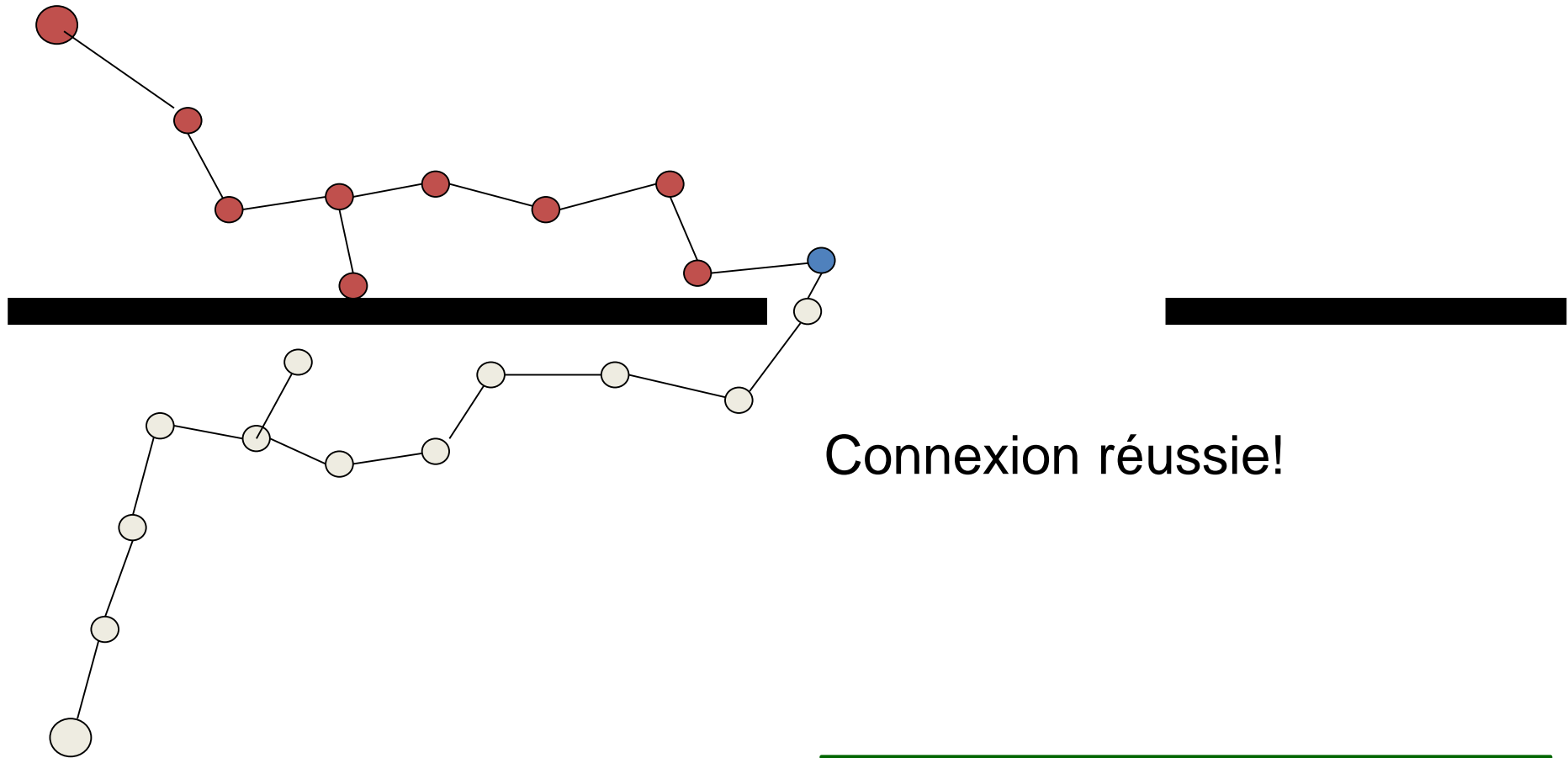
# RRT-Connect: exemple

Phase croissance de l'arbre





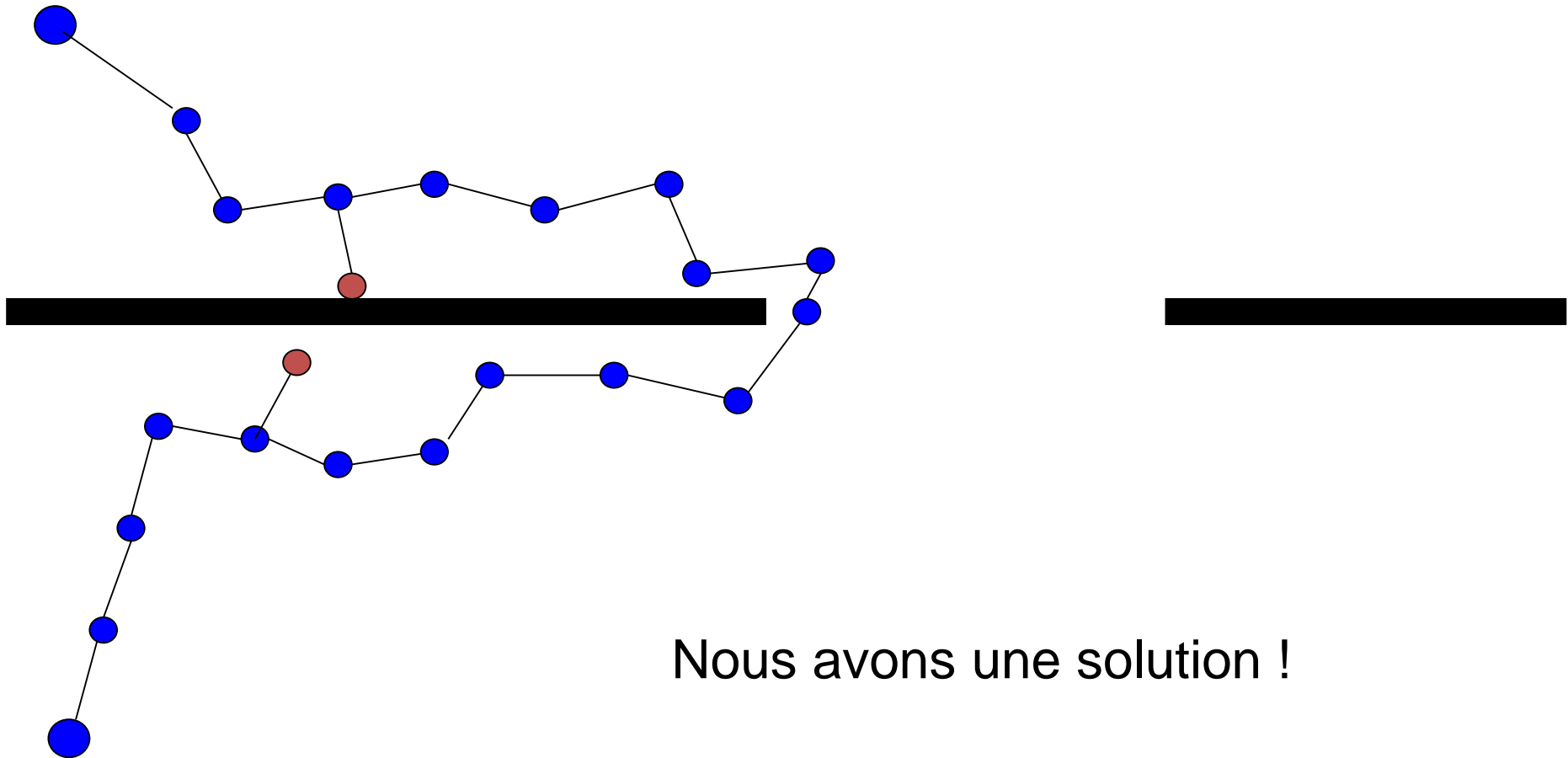
# RRT-Connect: exemple



**Phase connexion des arbres**

# RRT-Connect: exemple

---



Nous avons une solution !

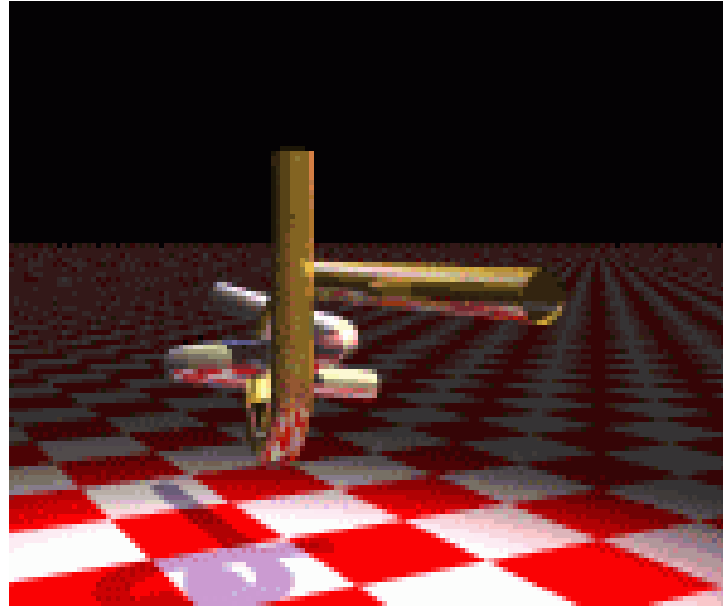




# Exemple de problème résolu avec RRT

---

- Extractions de deux tiges courbées



Problème proposé par Boris Yamrom, GE comme exemple de problème difficile. Repris par Nancy Amato, Texas A&M University. Solution en 2001 par James Kuffner et Steve LaValle. Calcul sur un PC de 2003 : quelques minutes.

# RRT-Connect si non-holonome

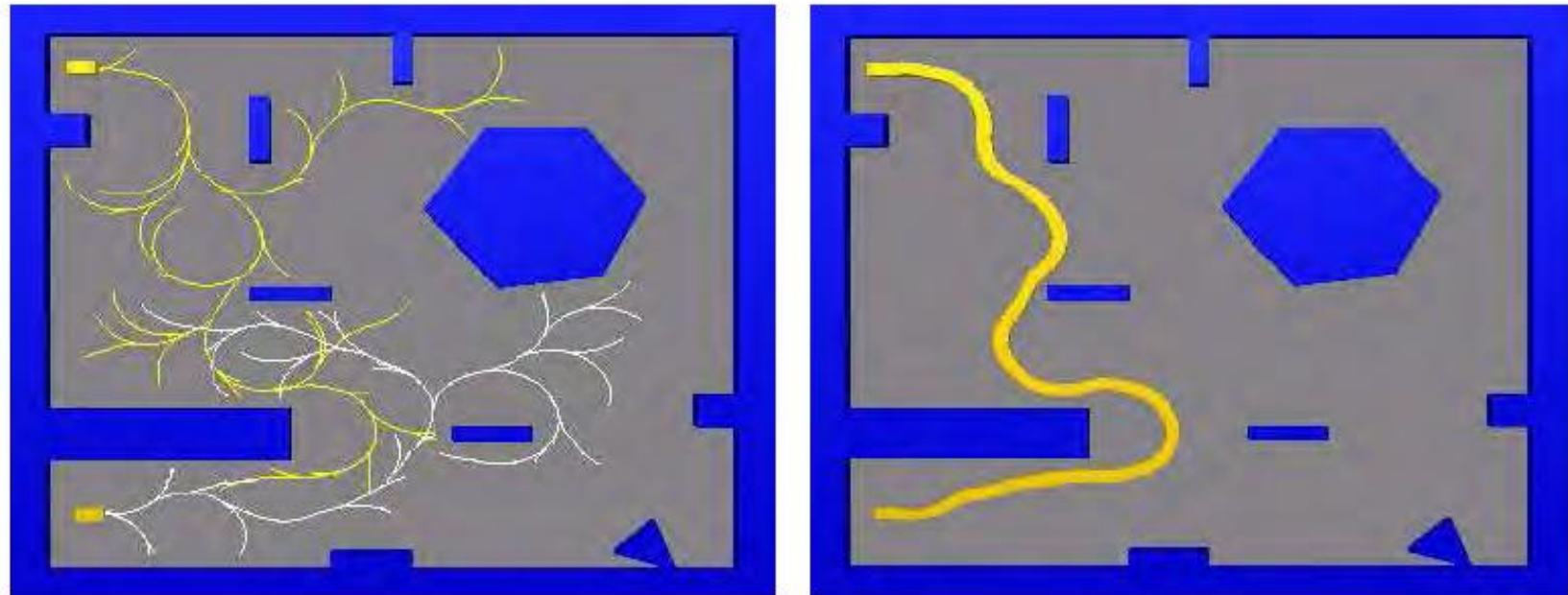


Fig. 4. The RRT obtained with a bidirectional planner (RRT-ExtExt) in 22.583 secs and 1376 nodes.

## Exploring unknown environments with RRT-based strategies

Abraham Sánchez L. and René Zapata