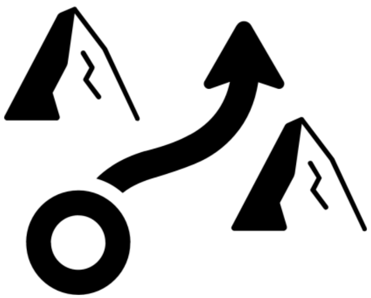
Robot Navigation and collision avoidance

Navigation: Dynamic Short Path

Dynamic short path Computation

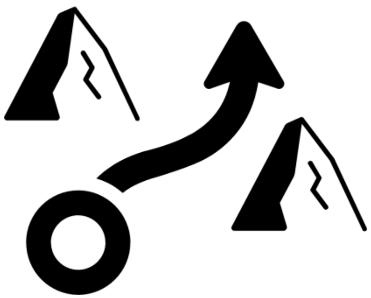
- ❑ Computation of short path becomes hard when graph structure increase
- ❑ In robotic navigation, graph structure change (dynamic obstacles)

→How to recompute short path without recomputing the entire algorithm



Lifelong Planning A* (LPA)

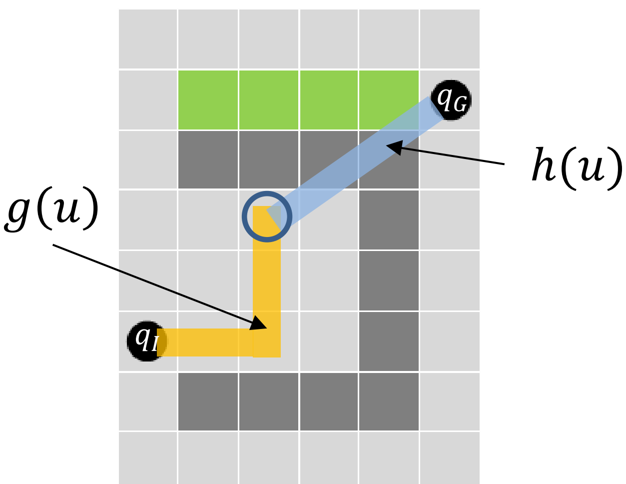
- Incremental version of A*
- Apply on known graph where edge costs can increase and decrease over the time.
- No need to recompute the entire algorithm if edge cost change



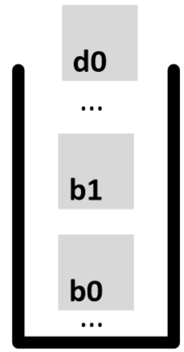
Lifelong Planning A* (LPA)

- ❑ A* reminder

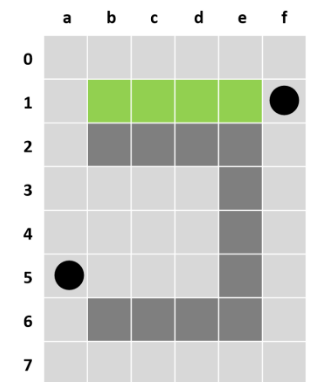
$$f_{score}(u) = g(u) + h(u)$$



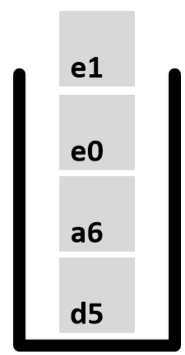
$Children[]\ children\ list\ of\ each\ node$



Closed List



A graph

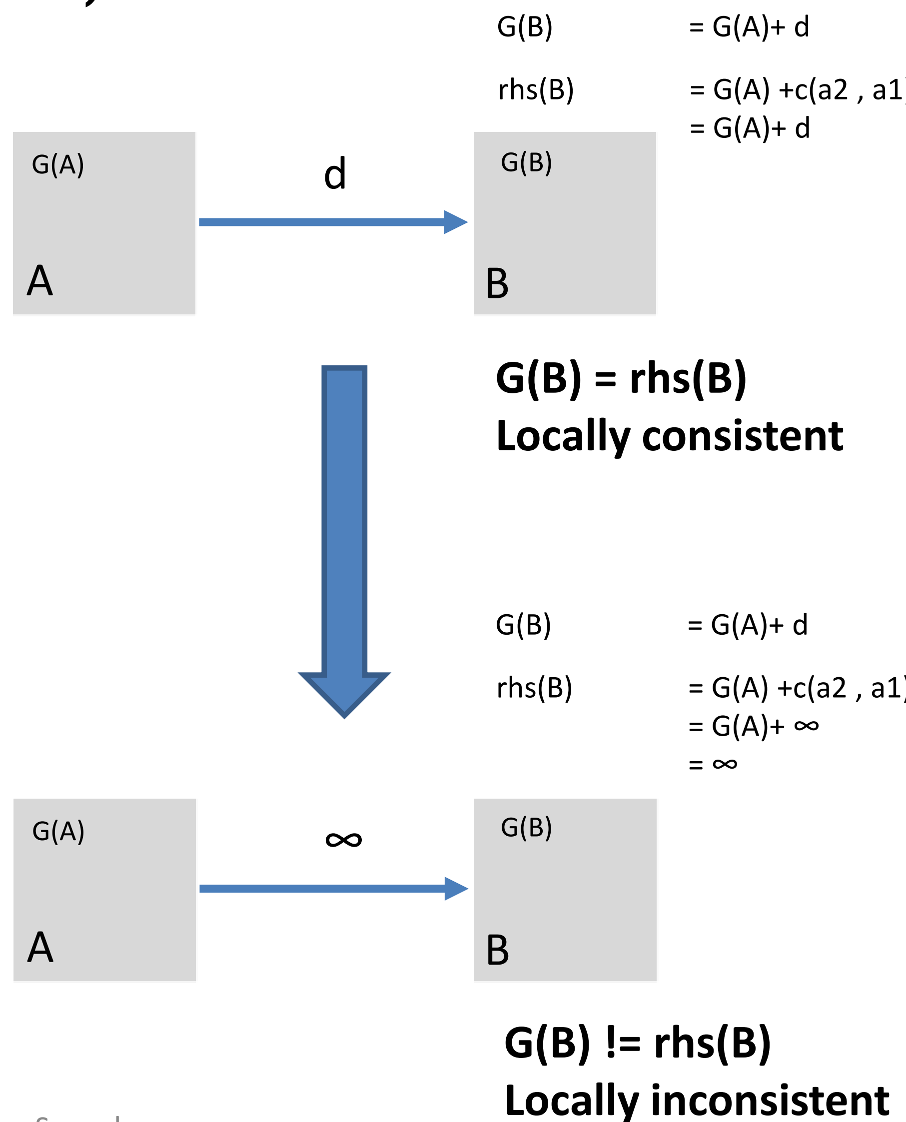


Open List

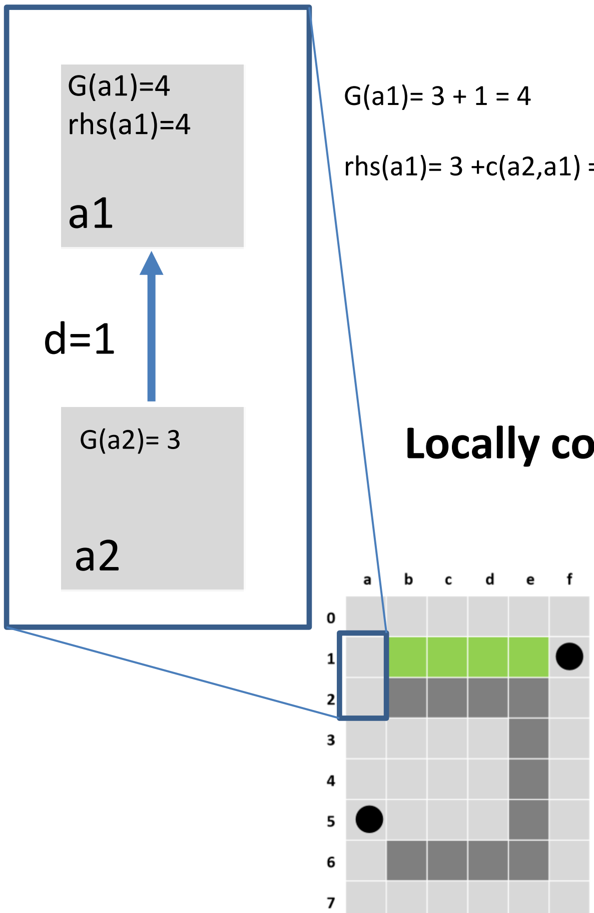
Order by min $f_{score}(u) = g(u) + h(u)$

Lifelong Planning A* (LPA)

- What new with LPA ?
  - Same basic algorithm
  - Inconsistances appear when edge cost change (obstacle)
  - LPA maintains an estimate g(n) of each vertex
  - LPA add a new value Right Hand Side (rhs) for detecting inconsistence



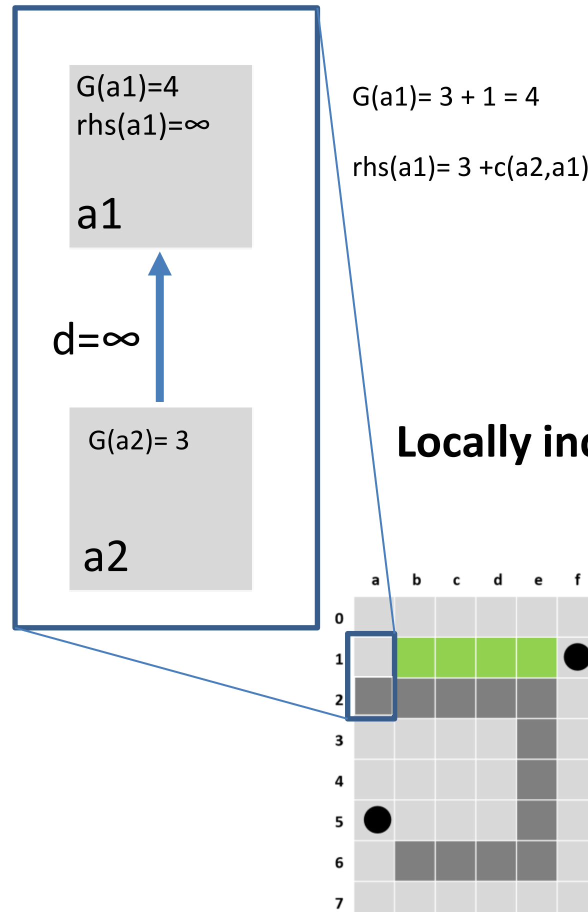
Lifelong Planning A* (LPA)



G(a1)= 3 + 1 = 4

rhs(a1)= 3 +c(a2,a1) =4

Locally consistent

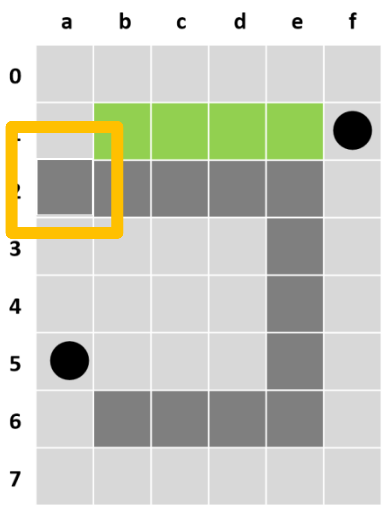


G(a1)= 3 + 1 = 4

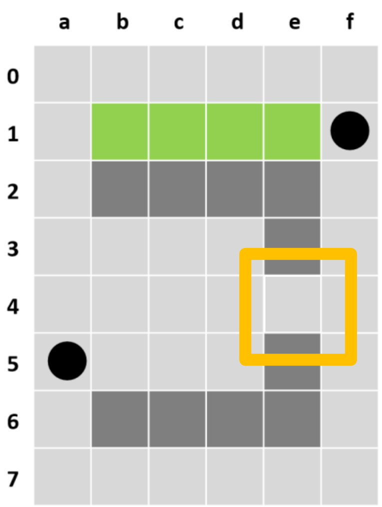
rhs(a1)= 3 +c(a2,a1) =3+ ∞=∞

Locally inconsistent

Lifelong Planning A* (LPA)



G(n) < Rsh(n)
Underconsitency
e.g Obstacle apparition



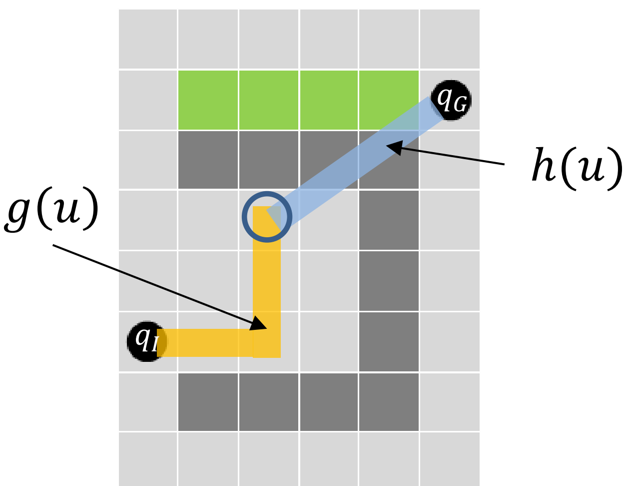
G(n) > Rsh(n)
Overconsitency
e.g Obstacle disparition

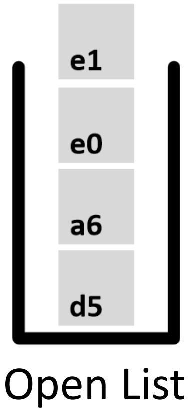
Lifelong Planning A* (LPA)

- LPA

$$f_{score}(u) = g(u) + h(u)$$

rsh(u)= min($g(u') + c(u,u')$)





Order by min ($\boldsymbol{key}$)

$\boldsymbol{key}$=[

min(*g(u), rhs(u)+h(u)*);

min(*g(u), rhs(u)*)

]

$Children[]\ children\ list\ of\ each\ node$

$Parents[]\ children\ list\ of\ each\ node$

Lifelong Planning A* (LPA)

procedure CalculateKey(s)
return $[\min(g(s), rhs(s)) + h(s, s_{goal}); \min(g(s), rhs(s))]$;

procedure Initialize$()$
$U = \emptyset$;
for all $s \in S$ $rhs(s) = g(s) = \infty$;
$rhs(s_{start}) = 0$;
U.Insert(s_{start}, CalculateKey(s_{start}));

procedure UpdateVertex(u)
if $(u \neq s_{start})$ $rhs(u) = \min_{s' \in Pred(u)}(g(s') + c(s', u))$;
if $(u \in U)$ U.Remove(u);
if $(g(u) \neq rhs(u))$ U.Insert(u, CalculateKey(u));

procedure ComputeShortestPath$()$
while (U.TopKey() $\dot{<}$ CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal})$)
  u = U.Pop();
  if $(g(u) > rhs(u))$
    $g(u) = rhs(u)$;
    for all $s \in Succ(u)$ UpdateVertex(s);
  else
    $g(u) = \infty$;
    for all $s \in Succ(u) \cup \{u\}$ UpdateVertex(s);

procedure Main$()$
Initialize();
forever
  ComputeShortestPath();
  Wait for changes in edge costs;
  for all directed edges (u, v) with changed edge costs
    Update the edge cost $c(u, v)$;
    UpdateVertex(v);

U is a priority Queue

U.TopKey() Smallest key value in U

U.Pop() return vertex u with smallest key value in U and remove u from U

Update priority of element in U

Lifelong Planning A* (LPA)

procedure CalculateKey(s)
return $[\min(g(s), rhs(s)) + h(s, s_{goal}); \min(g(s), rhs(s))]$;

Key computation

procedure Initialize$()$
$U = \emptyset$;
for all $s \in S$ $rhs(s) = g(s) = \infty$;
$rhs(s_{start}) = 0$;
U.Insert(s_{start}, CalculateKey(s_{start}));

procedure UpdateVertex(u)
if $(u \neq s_{start})$ $rhs(u) = \min_{s' \in Pred(u)}(g(s') + c(s', u))$;
if $(u \in U)$ U.Remove(u);
if $(g(u) \neq rhs(u))$ U.Insert(u, CalculateKey(u));

procedure ComputeShortestPath$()$
while (U.TopKey() $\dot{<}$ CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal})$)
    $u =$ U.Pop();
    if $(g(u) > rhs(u))$
        $g(u) = rhs(u)$;
        for all $s \in Succ(u)$ UpdateVertex(s);
    else
        $g(u) = \infty$;
        for all $s \in Succ(u) \cup \{u\}$ UpdateVertex(s);

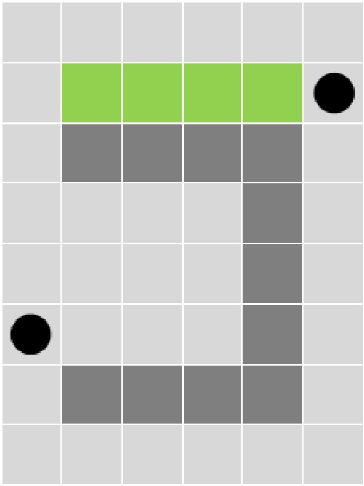
Overconsitency

Underconsitency

procedure Main$()$
Initialize();
forever
    ComputeShortestPath();
    Wait for changes in edge costs;
    for all directed edges (u, v) with changed edge costs
        Update the edge cost $c(u, v)$;
        UpdateVertex(v);

Loop until end condition
(e.g goal reached)

Lifelong Planning A* (LPA)



	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞	0	∞	∞
2	6	∞	∞													1	∞	∞
3	7	∞	∞	6	∞	∞	5	∞	∞	4	∞	∞				2	∞	∞
4	8	∞	∞	7	∞	∞	6	∞	∞	5	∞	∞				3	∞	∞
5	9	∞	0	8	∞	∞	7	∞	∞	6	∞	∞				4	∞	∞
6	10	∞	∞													5	∞	∞
7	11	∞	∞	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

Lifelong Planning A* (LPA)

It 0

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞	0	∞	∞
2	6	∞	∞													1	∞	∞
3	7	∞	∞	6	∞	∞	5	∞	∞	4	∞	∞				2	∞	∞
4	8	∞	∞	7	∞	∞	6	∞	∞	5	∞	∞				3	∞	∞
5	9	∞	0	8	∞	∞	7	∞	∞	6	∞	∞				4	∞	∞
6	10	∞	∞													5	∞	∞
7	11	∞	∞	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

procedure Initialize()
{02} $U = \emptyset$;
{03} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{04} $rhs(s_{start}) = 0$;
{05} U.Insert(s_{start}, CalculateKey(s_{start}));

U		
Key Part1	Key Part 2	Node
9	0	A5

Lifelong Planning A* (LPA)

It 1

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞	0	∞	∞
2	6	∞	∞													1	∞	∞
3	7	∞	∞	6	∞	∞	5	∞	∞	4	∞	∞				2	∞	∞
4	8	∞	1	7	∞	∞	6	∞	∞	5	∞	∞				3	∞	∞
5	9	0	0	8	∞	1	7	∞	∞	6	∞	∞				4	∞	∞
6	10	∞	1													5	∞	∞
7	11	∞	∞	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

U		
Key Part1	Key Part 2	Node
9	1	A4
9	1	B5
11	1	A6

Lifelong Planning A* (LPA)

It 2

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞	0	∞	∞
2	6	∞	∞													1	∞	∞
3	7	∞	2	6	∞	∞	5	∞	∞	4	∞	∞				2	∞	∞
4	8	1	1	7	∞	2	6	∞	∞	5	∞	∞				3	∞	∞
5	9	0	0	8	∞	1	7	∞	∞	6	∞	∞				4	∞	∞
6	10	∞	1													5	∞	∞
7	11	∞	∞	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

U		
Key Part1	Key Part 2	Node
9	1	B5
9	2	A3
9	2	B4
11	1	A6

Lifelong Planning A* (LPA)

It 3

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞	0	∞	∞
2	6	∞	∞													1	∞	∞
3	7	∞	2	6	∞	∞	5	∞	∞	4	∞	∞				2	∞	∞
4	8	1	1	7	∞	2	6	∞	∞	5	∞	∞				3	∞	∞
5	9	0	0	8	1	1	7	∞	2	6	∞	∞				4	∞	∞
6	10	∞	1													5	∞	∞
7	11	∞	∞	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

U		
Key Part1	Key Part 2	Node
9	2	A3
9	2	B4
9	2	C5
11	1	A6

Lifelong Planning A* (LPA)

It 22

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	5	5	5	6	6	4	6	6	3	7	7	2	8	8	1	9	9
1	5	4	4	4	∞	9	3	∞	11	2	∞	12	1	∞	13	0	∞	10
2	6	3	3													1	∞	∞
3	7	2	2	6	3	3	5	4	4	4	5	5				2	∞	∞
4	8	1	1	7	2	2	6	3	3	5	4	4				3	∞	∞
5	9	0	0	8	1	1	7	2	2	6	3	3				4	∞	∞
6	10	1	1													5	∞	∞
7	11	∞	2	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

U		
Key Part1	Key Part 2	Node
10	10	F1
13	2	A7
13	9	B1
14	11	C1
14	12	D1
14	13	E1

Lifelong Planning A* (LPA)

It 22

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	5	5	5	6	6	4	6	6	3	7	7	2	8	8	1	9	9
1	5	4	4	4	∞	9	3	∞	11	2	∞	12	1	∞	13	0	∞	10
2	6	3	3													1	∞	∞
3	7	2	2	6	3	3	5	4	4	4	5	5				2	∞	∞
4	8	1	1	7	2	2	6	3	3	5	4	4				3	∞	∞
5	9	0	0	8	1	1	7	2	2	6	3	3				4	∞	∞
6	10	1	1													5	∞	∞
7	11	∞	2	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

U		
Key Part1	Key Part 2	Node
10	10	F1
13	2	A7
13	9	B1
14	11	C1
14	12	D1
14	13	E1

CPE LYON
ÉCOLE SUPÉRIEURE DE CHIMIE PHYSIQUE ÉLECTRONIQUE DE LYON

Lifelong Planning A* (LPA)

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	5	5	5	6	6	4	6	6	3	7	7	2	8	8	1	9	9
1	5	4	4	4	∞	9	3	∞	11	2	∞	12	1	∞	13	0	∞	10
2	6	3	3													1	∞	∞
3	7	2	2	6	3	3	5	4	4	4	5	5				2	∞	∞
4	8	1	1	7	2	2	6	3	3	5	4	4				3	∞	∞
5	9	0	0	8	1	1	7	2	2	6	3	3				4	∞	∞
6	10	1	1													5	∞	∞
7	11	∞	2	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

procedure Main()
{17} Initialize();
{18} forever
{19} ComputeShortestPath();
{20} Wait for changes in edge costs;
{21} for all directed edges (u, v) with changed edge costs
{22} Update the edge cost $c(u, v)$;
{23} UpdateVertex(v);

Lifelong Planning A* (LPA)

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	5	5	5	6	6	4	6	6	3	7	7	2	8	8	1	9	9
1	5	4	4	4	∞	9	3	∞	11	2	∞	12	1	∞	13	0	∞	10
2	6	3	3													1	∞	∞
3	7	2	2	6	3	3	5	4	4	4	5	5				2	∞	∞
4	8	1	1	7	2	2	6	3	3	5	4	4				3	∞	∞
5	9	0	0	8	1	1	7	2	2	6	3	3				4	∞	∞
6	10	1	1													5	∞	∞
7	11	∞	2	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

```
procedure Main()
Initialize();
forever
    ComputeShortestPath();
    Wait for changes in edge costs;
    for all directed edges (u, v) with changed edge costs
        Update the edge cost c(u, v);
        UpdateVertex(v);
```

Lifelong Planning A* (LPA)

	A H(n)	A G(n)	A RHS(n)	B H(n)	B G(n)	B RHS(n)	C H(n)	C G(n)	C RHS(n)	D H(n)	D G(n)	D RHS(n)	E H(n)	E G(n)	E RHS(n)	F H(n)	F G(n)	F RHS(n)
0	6	5	∞	5	6	6	4	6	6	3	7	7	2	8	8	1	9	9
1	5	∞	∞	4	∞	∞	3	∞	11	2	∞	12	1	∞	13	0	∞	10
2	6	3	3													1	∞	∞
3	7	2	2	6	3	3	5	4	4	4	5	5				2	∞	∞
4	8	1	1	7	2	2	6	3	3	5	4	4				3	∞	∞
5	9	0	0	8	1	1	7	2	2	6	3	3				4	∞	∞
6	10	1	1													5	∞	∞
7	11	∞	2	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

procedure ComputeShortestPath()

```
while (U.TopKey() <̇ CalculateKey(s_goal) OR rhs(s_goal) ≠ g(s_goal))
  u = U.Pop();
  if (g(u) > rhs(u))
    g(u) = rhs(u);
    for all s ∈ Succ(u) UpdateVertex(s);
  else
    g(u) = ∞;
    for all s ∈ Succ(u) ∪ {u} UpdateVertex(s);
```

Underconsitency

Lifelong Planning A* (LPA)

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	13	0	∞	∞
2	6	3	3													1	∞	∞
3	7	2	2	6	3	3	5	4	4	4	5	5				2	∞	∞
4	8	1	1	7	2	2	6	3	3	5	4	4				3	∞	∞
5	9	0	0	8	1	1	7	2	2	6	3	3				4	∞	∞
6	10	1	1													5	∞	∞
7	11	∞	2	10	∞	∞	9	∞	∞	8	∞	∞	7	∞	∞	6	∞	∞

procedure ComputeShortestPath()
{09} while (U.TopKey() $\dot{<}$ CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal})$)
{10} u = U.Pop();
{11} if ($g(u) > rhs(u)$)
{12} $g(u) = rhs(u)$;
{13} for all $s \in Succ(u)$ UpdateVertex(s);
{14} else
{15} $g(u) = \infty$;
{16} for all $s \in Succ(u) \cup \{u\}$ UpdateVertex(s);

Underconsitency

Lifelong Planning A* (LPA)

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞	0	∞	∞
2	6	3	3													1	11	11
3	7	2	2	6	3	3	5	4	4	4	5	5				2	10	10
4	8	1	1	7	2	2	6	3	3	5	4	4				3	9	9
5	9	0	0	8	1	1	7	2	2	6	3	3				4	8	8
6	10	1	1													5	7	7
7	11	2	2	10	2	2	9	3	3	8	4	4	7	5	5	6	6	6

procedure ComputeShortestPath()

while (U.TopKey() $\dot{<}$ CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal})$)
  u = U.Pop();
  if ($g(u) > rhs(u)$)
    $g(u) = rhs(u)$;
    for all $s \in Succ(u)$ UpdateVertex(s);
  else
    $g(u) = \infty$;
    for all $s \in Succ(u) \cup \{u\}$ UpdateVertex(s);

Overconsitency

Lifelong Planning A* (LPA)

	A			B			C			D			E			F		
	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n	H(n)	G(n)	RHS(n)
0	6	∞	∞	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞
1	5	∞	∞	4	∞	∞	3	∞	∞	2	∞	∞	1	∞	∞	0	∞	∞
2	6	3	3													1	11	11
3	7	2	2	6	3	3	5	4	4	4	5	5				2	10	10
4	8	1	1	7	2	2	6	3	3	5	4	4				3	[illegible]	9
5	9	0	0	8	1	1	7	2	2	6	3	3				4	[illegible]	8
6	10	[illegible]	1													5	[illegible]	7
7	11	2	2	10	2	2	9	3	3	8	4	4	7	5	5	6	6	6

procedure ComputeShortestPath()
{09} while (U.TopKey() $\dot{<}$ CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal})$)
{10} u = U.Pop();
{11} if ($g(u) > rhs(u)$)
{12} $g(u) = rhs(u)$;
{13} for all $s \in Succ(u)$ UpdateVertex(s);
{14} else
{15} $g(u) = \infty$;
{16} for all $s \in Succ(u) \cup \{u\}$ UpdateVertex(s);

Overconsitency

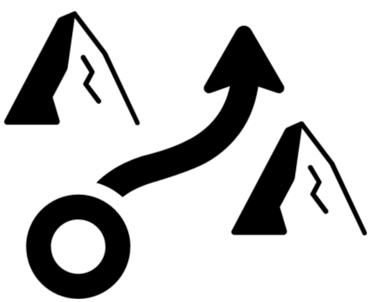
CPE LYON
ÉCOLE SUPÉRIEURE DE CHIMIE PHYSIQUE ÉLECTRONIQUE DE LYON

Lifelong Planning A* (LPA)

- Not the entire path needs to be recomputed
- Useful for dynamic obstacles with fix goal and start point

- In case of robot navigation, robot evolves on the path, **start point contineously changes.**

  → **D* Lite**



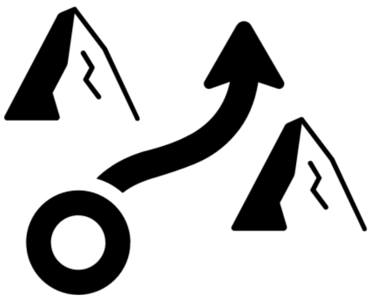
D* Lite

- ❑ Based on LPA !
- ❑ Reactive on dynamic obstacles (as LPA)
- ❑ Contineously updates start point

- ❑ Algorithm behaviors:
  - ❑ Same algorithm as LPA
  - ❑ **Start the algorithm one the goal point**

    (same as LPA with start point = goal point, goal point = start point)
  - ❑ Add an estimate start point evolution

    noted km added into key computation



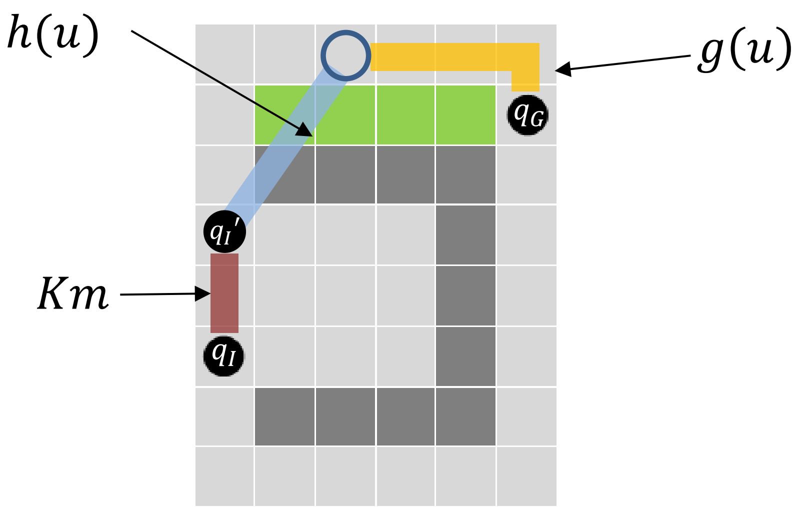
D* Lite

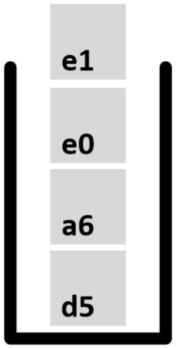
- D* Lite

$$f_{score}(u) = g(u) + h(u)$$

rsh(u)= min($g(u') + c(u, u')$)

Km=km+h(qi,qi')





Open List

Order by min (***key***)

key=[
min(*g(u), rhs(u)+h(u)+***km***);
min(*g(u), rhs(u)*)
]

$Children[]\ children\ list\ of\ each\ node$

$Parents[]\ children\ list\ of\ each\ node$

D* Lite

procedure CalculateKey(s)

return $[\min(g(s), \boxed{rhs(s)) + h(s_{start}, s) + k_m} \min(g(s), rhs(s))]$;

procedure Initialize$()$

$U = \emptyset$;

$\boxed{k_m = 0}$;

for all $s \in S$ $rhs(s) = g(s) = \infty$;

$rhs(s_{goal}) = 0$;

U.Insert$(s_{goal}$, CalculateKey$(s_{goal}))$;

procedure UpdateVertex(u)

if $(u \neq s_{goal})$ $rhs(u) = \min_{s' \in \boxed{Succ(u)}}(c(u, s') + g(s'))$;

if $(u \in U)$ U.Remove(u);

if $(g(u) \neq rhs(u))$ U.Insert$(u$, CalculateKey$(u))$;

procedure ComputeShortestPath$()$

while (U.TopKey() $\dot{<}$ CalculateKey(s_{start}) OR $rhs(s_{start}) \neq g(s_{start})$)

  $\boxed{k_{old} = \text{U.TopKey()};}$

  u = U.Pop();

  $\boxed{\begin{array}{l}\text{if } (k_{old} \dot{<} \text{CalculateKey}(u)) \\ \quad \text{U.Insert}(u, \text{CalculateKey}(u));\end{array}}$

  else if $(g(u) > rhs(u))$

    $g(u) = rhs(u)$;

    for all $s \in \boxed{Pred(u)}$ UpdateVertex(s);

  else

    $g(u) = \infty$;

    for all $s \in \boxed{Pred(u)} \cup \{u\}$ UpdateVertex(s);

procedure Main$()$

$s_{last} = s_{start}$;

Initialize();

ComputeShortestPath();

while $(s_{start} \neq s_{goal})$

  /* if $(g(s_{start}) = \infty)$ then there is no known path */

  $s_{start} = \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;

  Move to s_{start};

  Scan graph for changed edge costs;

  if any edge costs changed

    $k_m = k_m + h(s_{last}, s_{start})$;

    $s_{last} = s_{start}$;

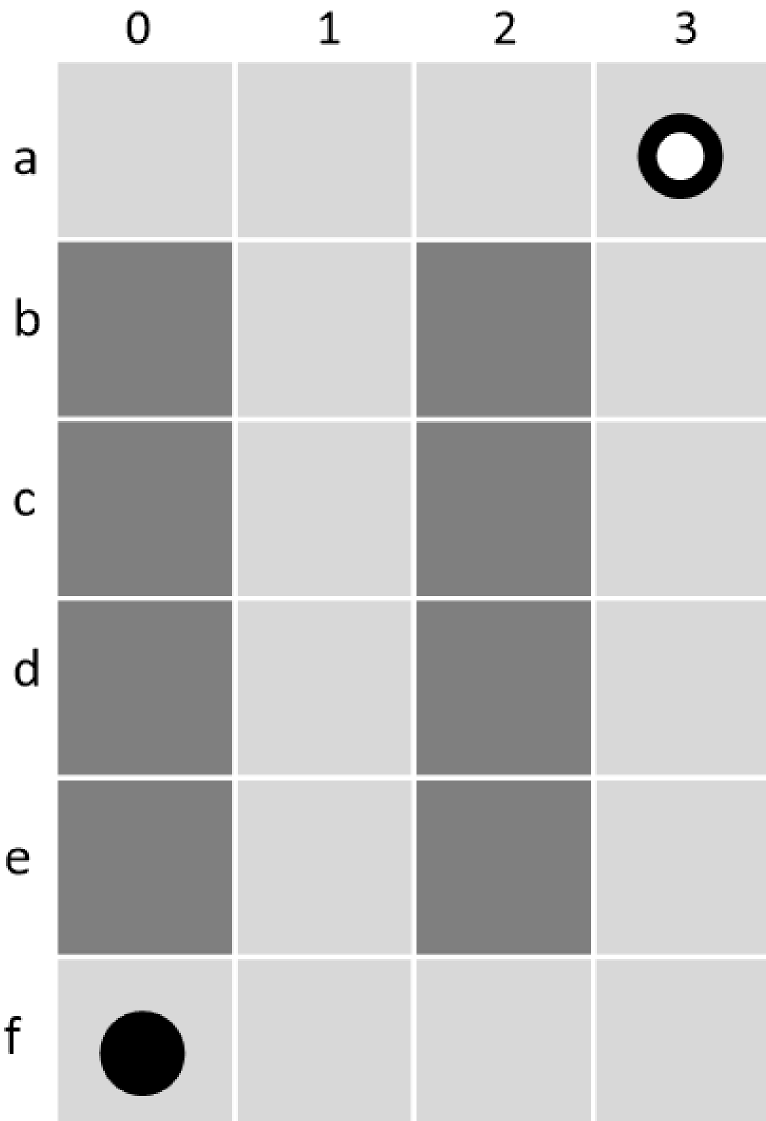
    for all directed edges (u, v) with changed edge costs

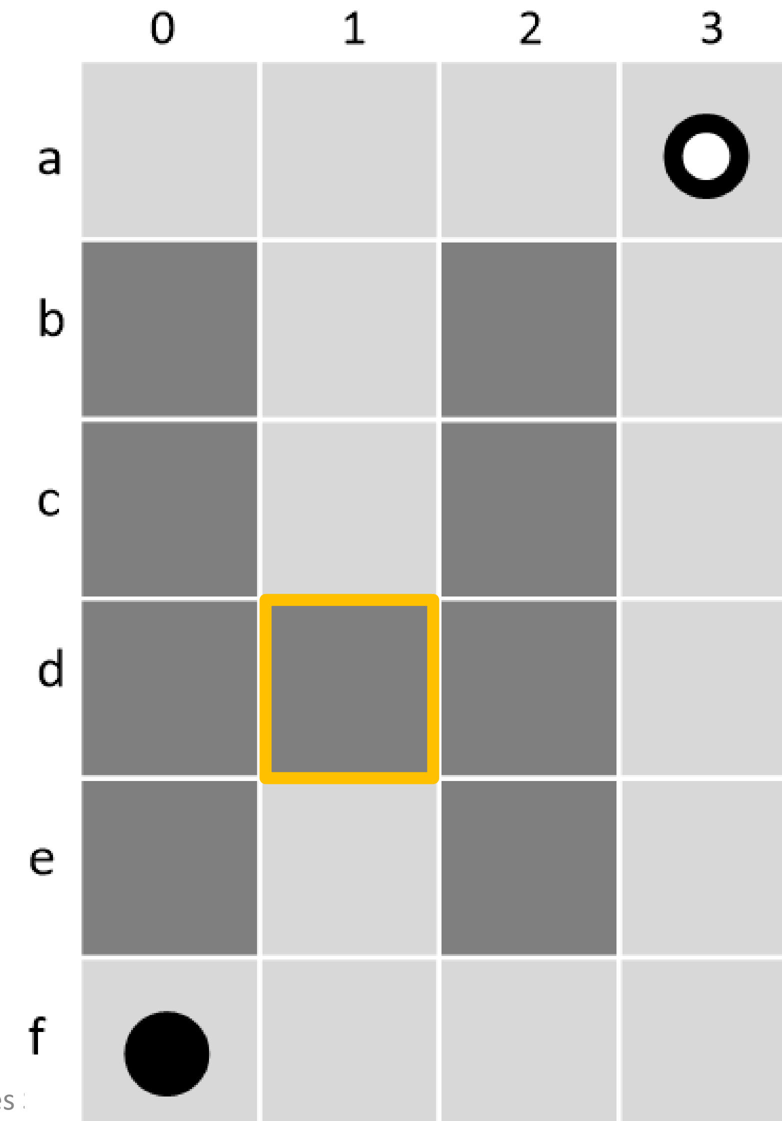
      Update the edge cost $c(u, v)$;

      UpdateVertex(u);

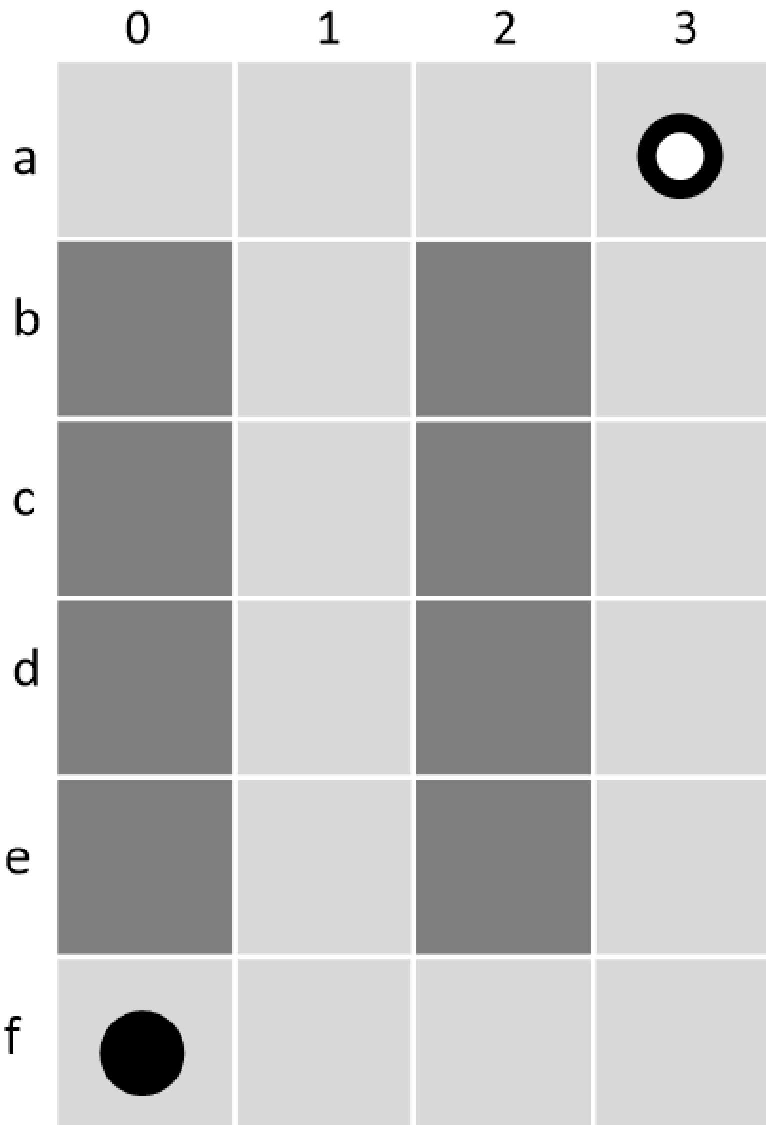
    ComputeShortestPath();

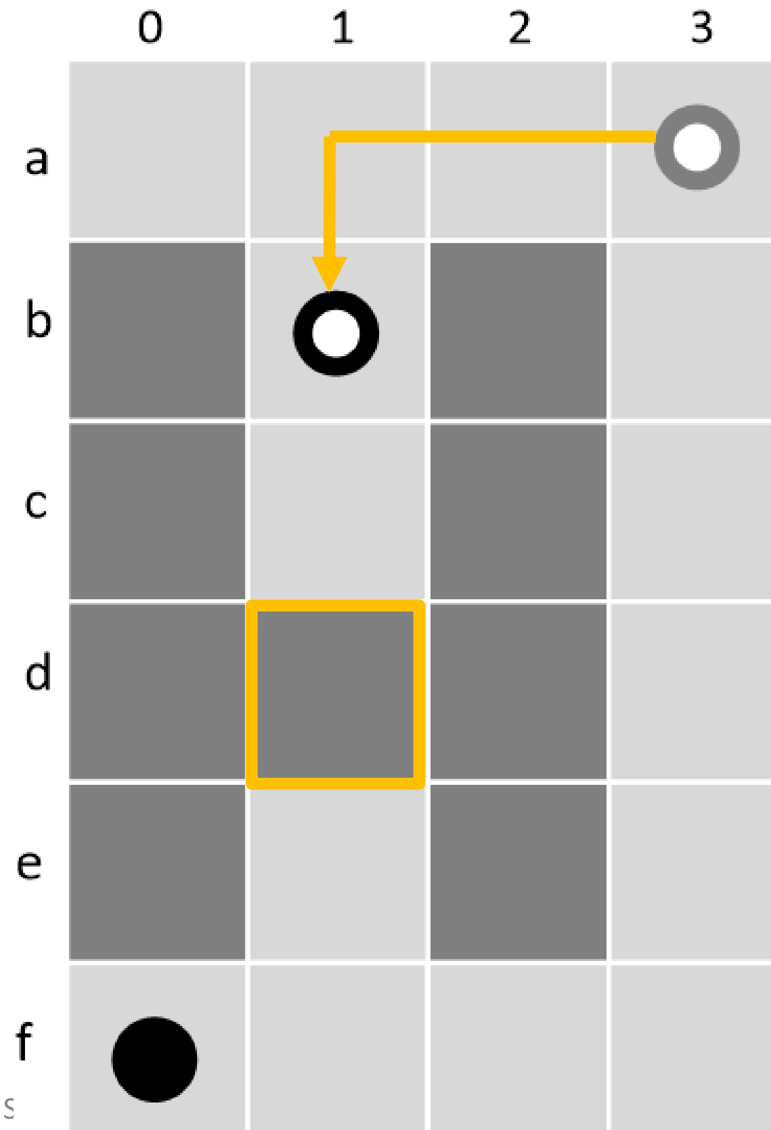
Exercice: LPA

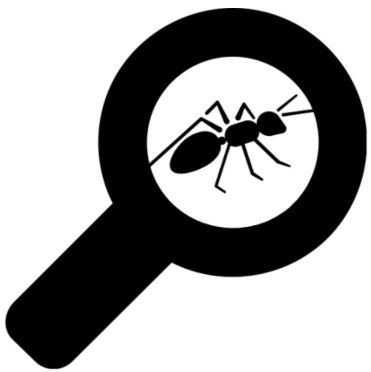




Exercice: D* lite





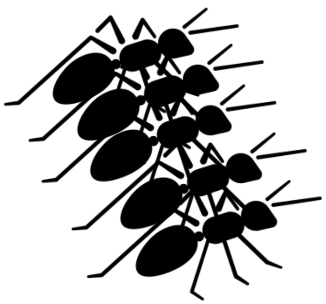


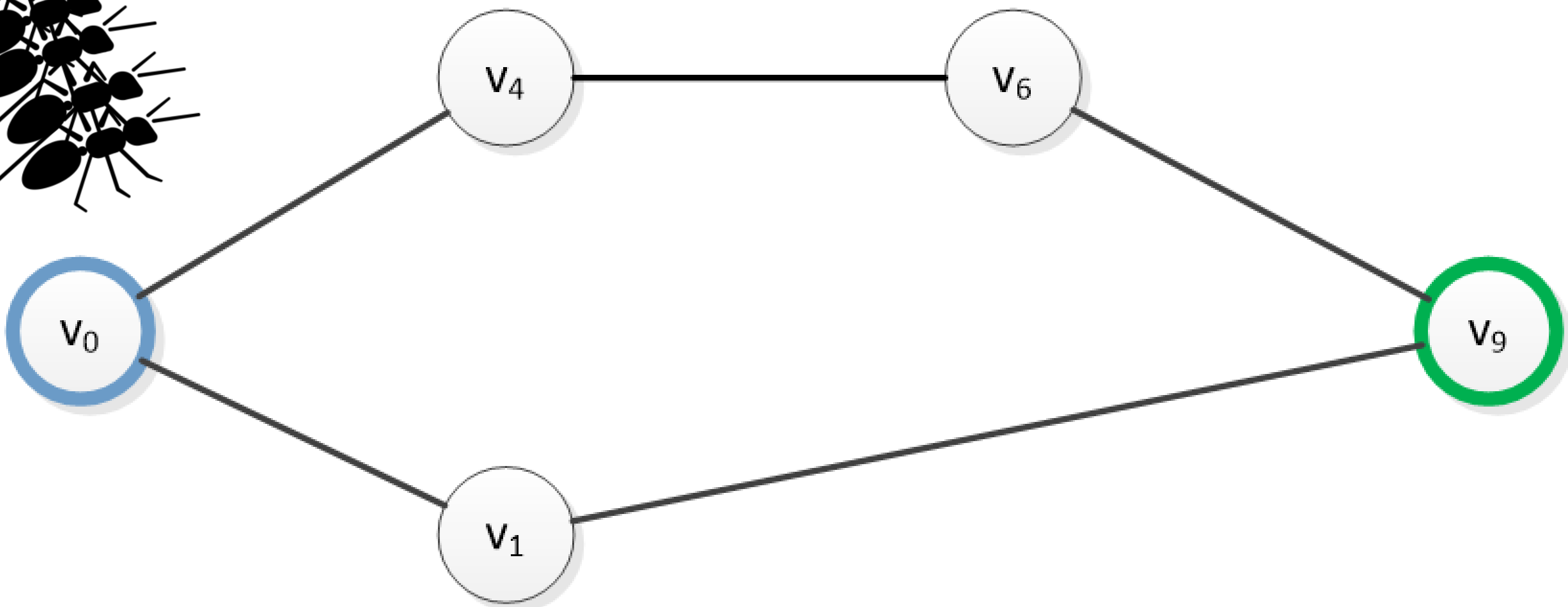
Focus on: Ant Colony

Ant Colony System

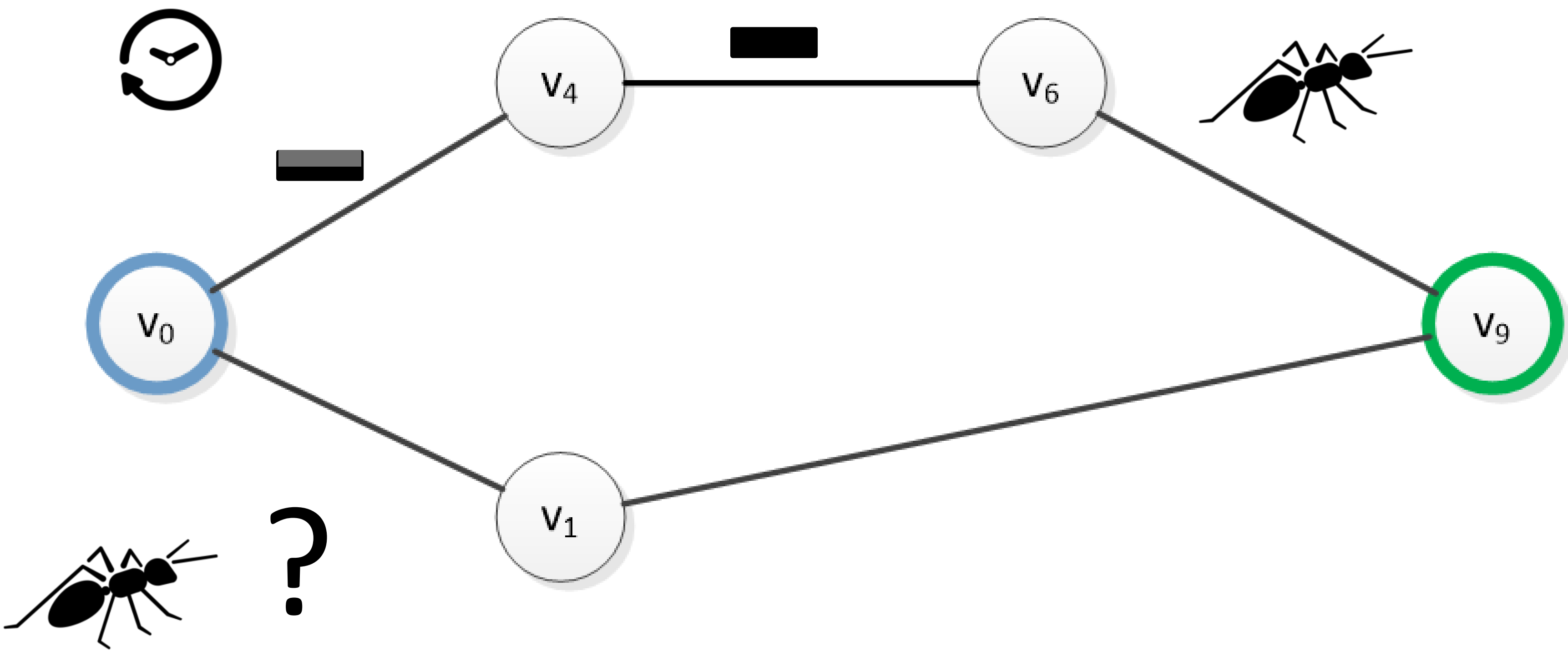
- Environment hold information
- Multiple – agents used to discover solutions
- Probabilistic technique
- Meta-heuristic optimization

Ant Colony System

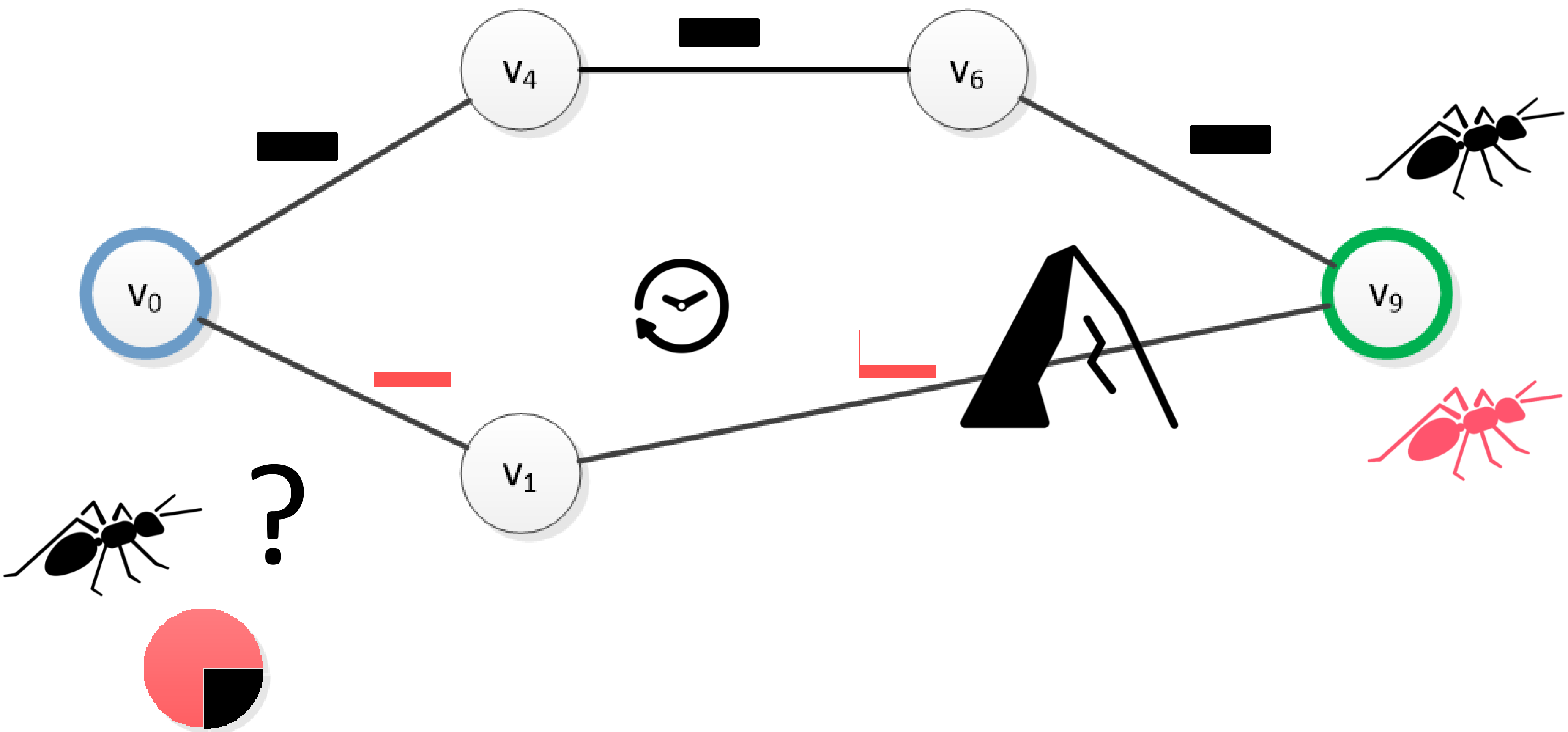




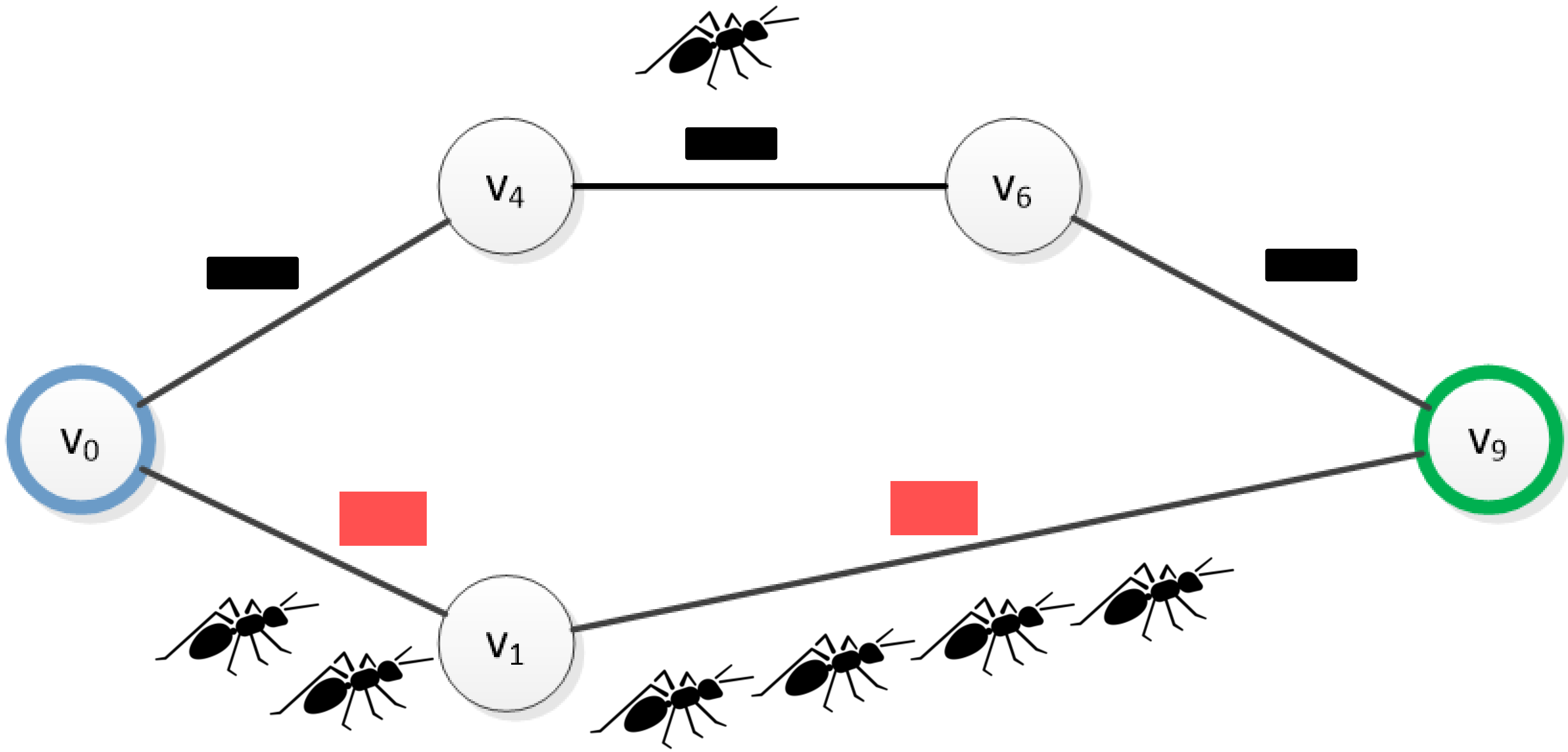
Ant Colony System



Ant Colony System



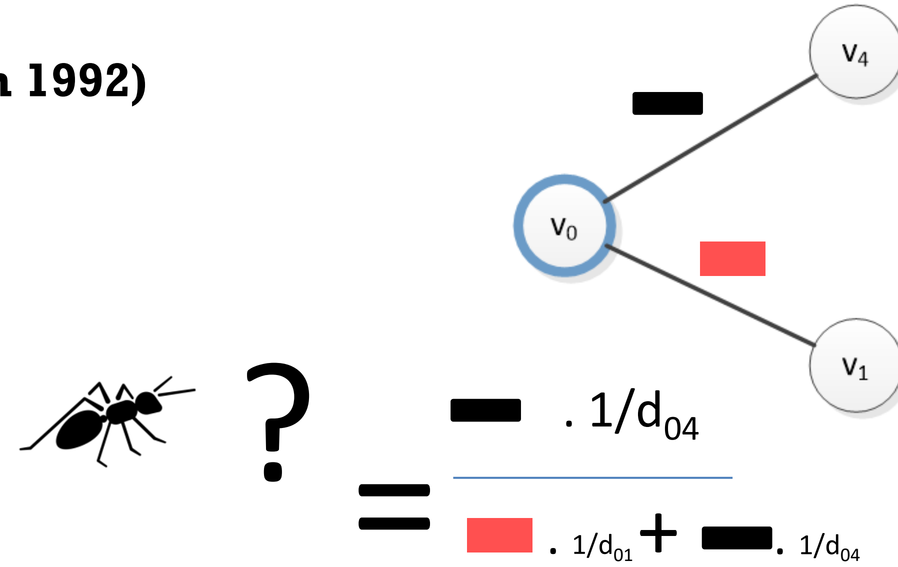
Ant Colony System



Ant System (First ACO algorithm 1992)

- Path Selection

$$p_{ij} = \frac{\tau^{\alpha}_{ij}\eta^{\beta}_{ij}}{\sum \tau^{\alpha}_{ix}\eta^{\beta}_{ix}}$$



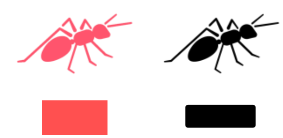
- p_{ij} → probability to select node j
- τ^{α}_{ij} → pheromone hold by edge i,j (with pheromone factor α)
- η^{β}_{ij} →edge cost (usually $1/d_{ij}$)
- X → all nodes conneted to node i

Ant System

- Pheromone update after each ant reaches objective

$$\tau_{ij}(\mathrm{t}+1) \leftarrow (1-\rho).\tau_{ij}(\mathrm{t}) + \sum_{k=1}^{m} \Delta\tau^{\mathrm{k}}{}_{ij}(\mathrm{t})$$

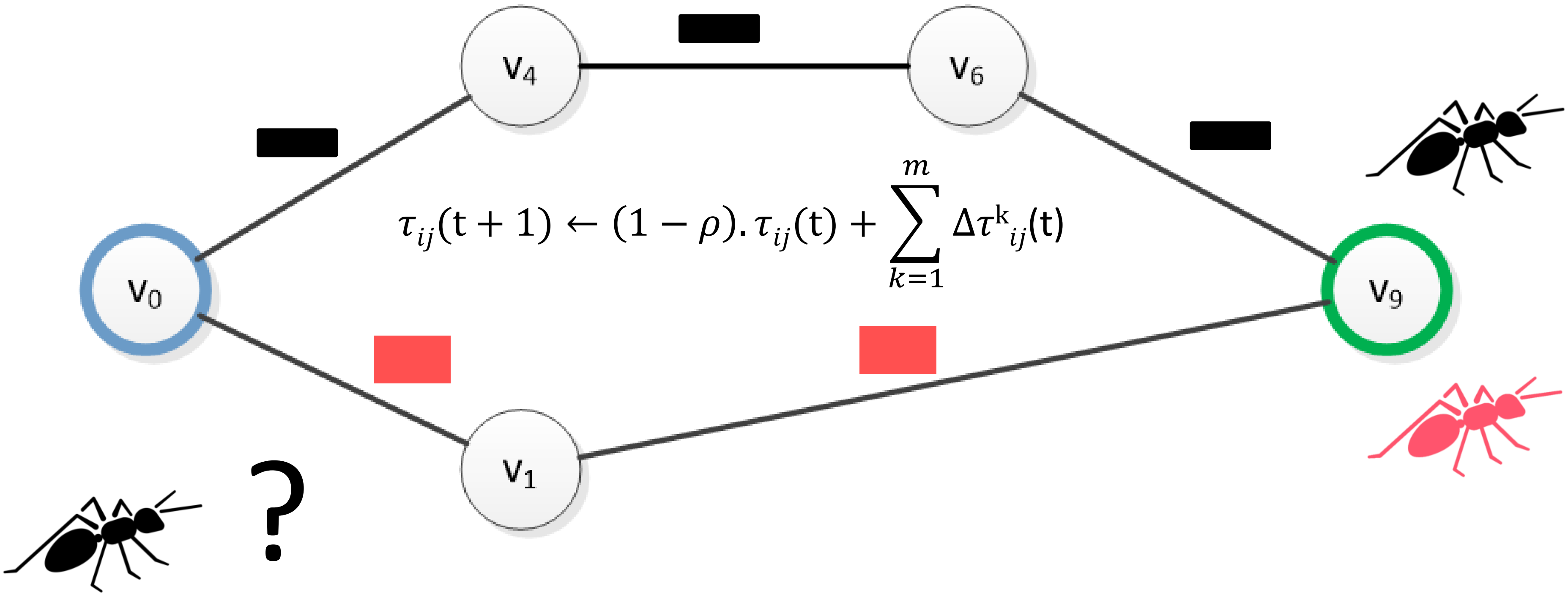
- ρ → evaporation factor
- m → number of ants
- $\eta^{\beta}{}_{ij}$ →edge cost (usually $1/d_{ij}$)
- X → all nodes conneted to node i
- $\Delta\tau^{\mathrm{ant}}{}_{ij}(\mathrm{t})$ →pheromone quantity laid on edge (i,j) by th Kth ant.



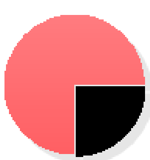
$$\Delta\tau^{\mathrm{ant}}{}_{ij}(\mathrm{t}) \begin{cases} \dfrac{1}{L_k} & if\ Kth\ ant\ travel\ on\ edge\ i,j \\ 0 & otherwise \end{cases}$$

$L_k is\ the\ path\ lenght\ of\ th\ K^{th} ant$

Ant System

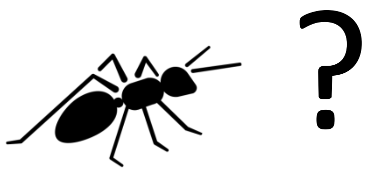


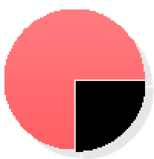
$$\tau_{ij}(t+1) \leftarrow (1-\rho).\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau^{k}{}_{ij}(t)$$



$$p_{ij} = \frac{\tau^{\alpha}{}_{ij}\eta^{\beta}{}_{ij}}{\sum \tau^{\alpha}{}_{ix}\eta^{\beta}{}_{ix}}$$

Ant Colony System



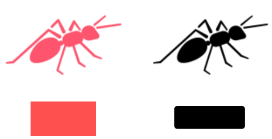


Path Selection update

q=rand[0;1]

If $q \leq q_0$

$$\max \tau^{\alpha}{}_{ix}\eta^{\beta}{}_{ix} \qquad \qquad \frac{\tau^{\alpha}{}_{ij}\eta^{\beta}{}_{ij}}{\sum \tau^{\alpha}{}_{ix}\eta^{\beta}{}_{ix}}$$



Pheromone Update

Each Step

$$\tau_{ij} = (1-\varphi).\tau_{ij} + \varphi.\tau_0$$

$\varphi \in (0{,}1]$ pheromone decay coeff

τ_0 pheromone initial value

When goal reached $\tau_{ij}(t+1) \leftarrow (1-\rho).\tau_{ij}(t) + \rho.\Delta\tau^{\text{BEST}}{}_{ij}(t)$

Ant Colony System

https://www.youtube.com/watch?v=SJM3er3L6P4

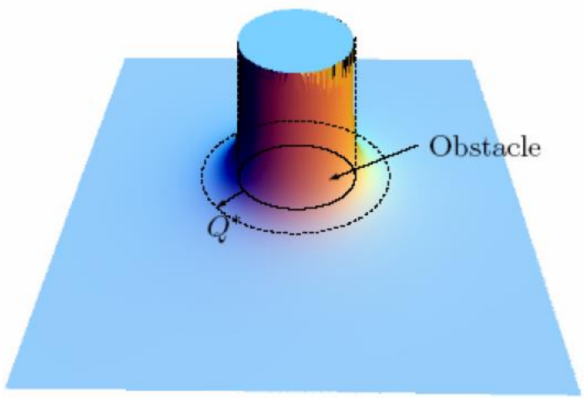
Focus on: Potential Fields

Potential Fields

- ❑ Objective

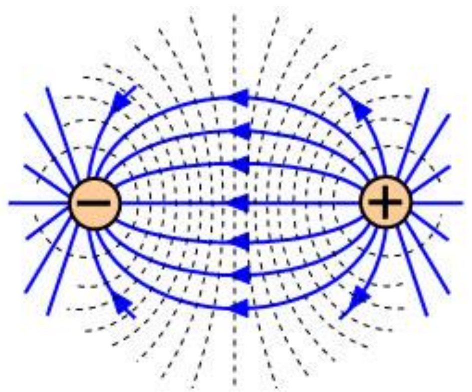
Generate attractive and repulsive potential field on the environment to drive the robot until it reaches the goal

- ❑ Algorithm
  - Obstacles generate repulsive potential field. The more the robot is closed to the obstacle, the higher the repusive potential field is,
  - Goal generates attractive potential field



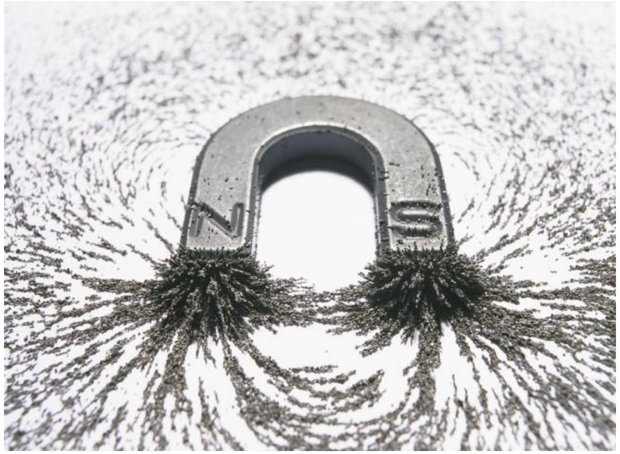
Robotic Motion Planning: Potential Functions,Robotics Institute 16-735, Howie Choset

Wath is a Potential Field ?



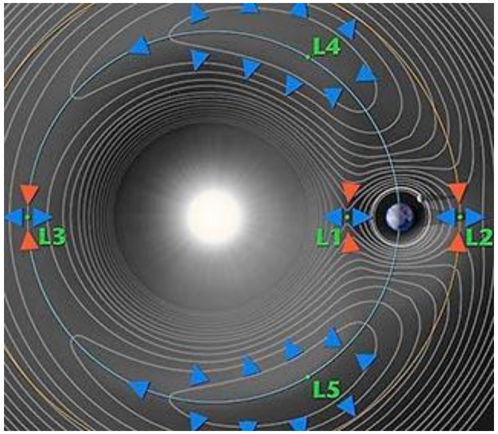
http://electricity-automation.com/img/electricity/fieldLines.jpg

Electric



Magnetic Field is a photograph by Cordelia Molloy

Magnetic

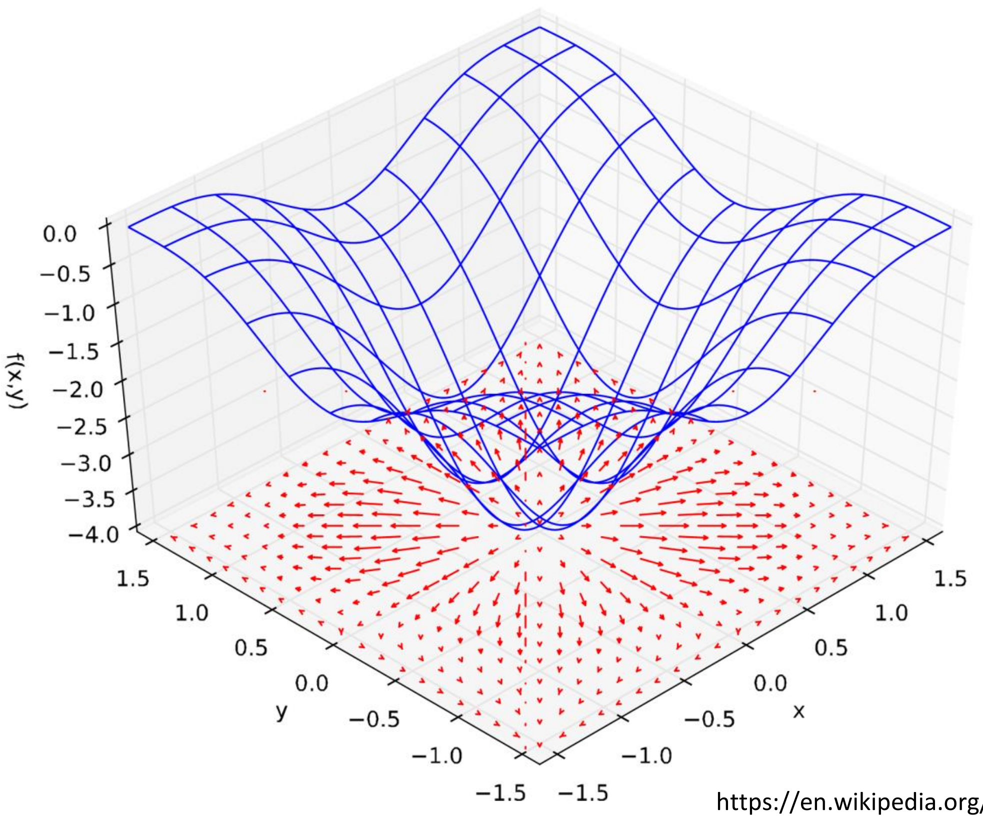


http://wmap.gsfc.nasa.gov/media/990529/990529_320.jpg

Gravity

Wath is a Potential Field ?

- Get through the Gradient computation of a function
- A gradient is the generalization of the concept of derivative to functions with multiple variables e.g $f(x, y, z)$.



https://en.wikipedia.org/wiki/Gradient

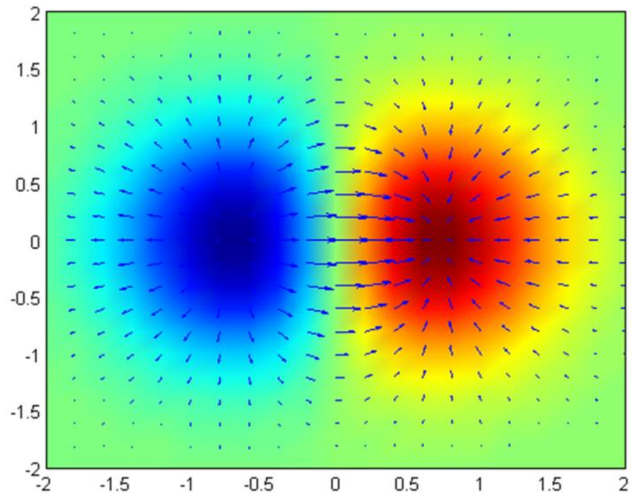
Wath is a Potential Field ?

- Notation (in cartesian coordinates)

$$\nabla f = \frac{\delta f}{\delta x}.i + \frac{\delta f}{\delta y}.j + \frac{\delta f}{\delta z}.k$$

Where i,j,k are respectively the standard unit vector

$f(x, y) = xe^{-(x2 + y2)}$

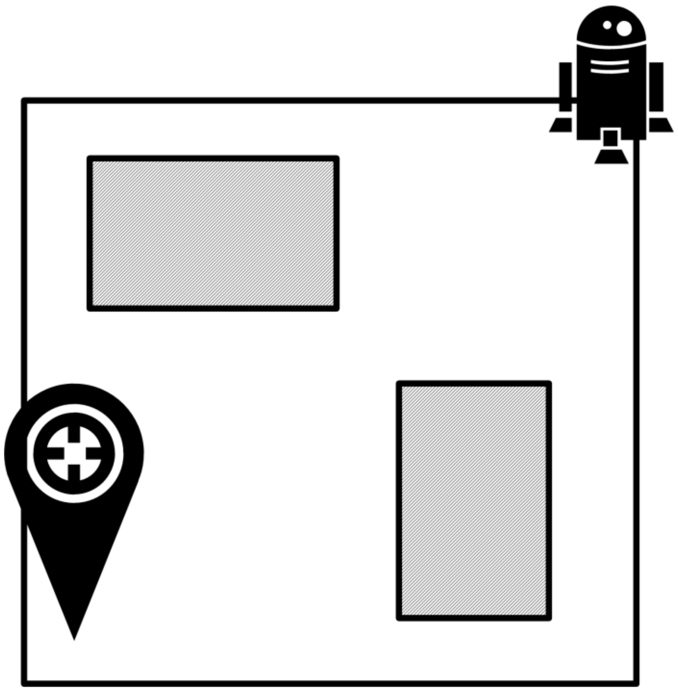


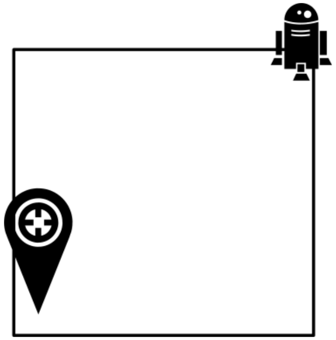
https://en.wikipedia.org/wiki/Gradient

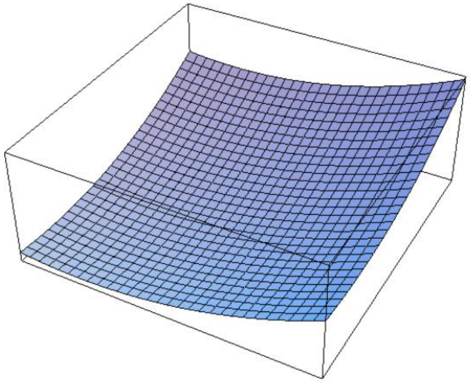
Generating artificial Potential Field

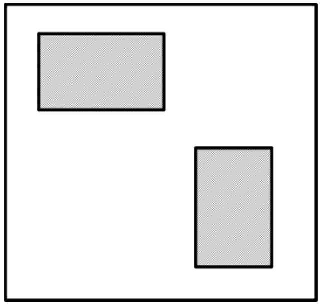
- How to use potential field into robotic ?
- Need :
  - → Move to the goal
- Constraint:
  - → Lots of obstacles between goal and robot
- Generation of artifical potential fields:
  - Goal = attrative field
  - Obstacles = repulsive fields

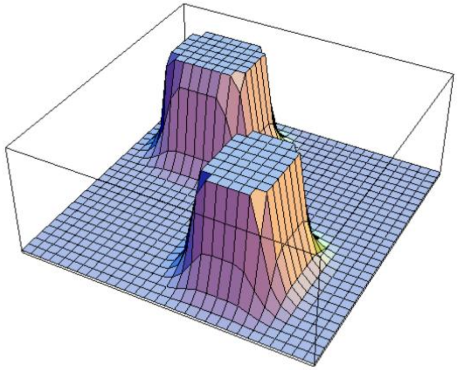
Generating artificial Potential Field



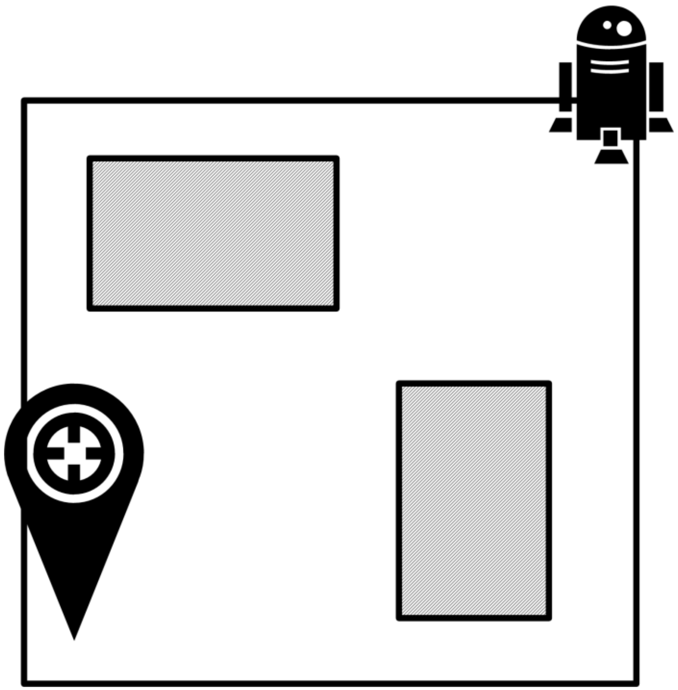


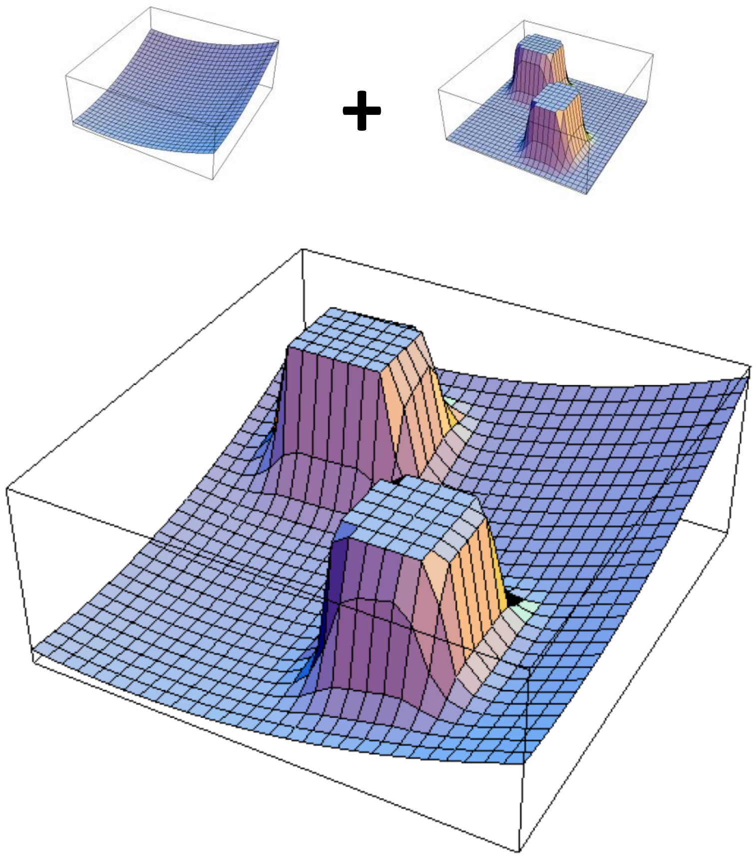






Generating artificial Potential Field

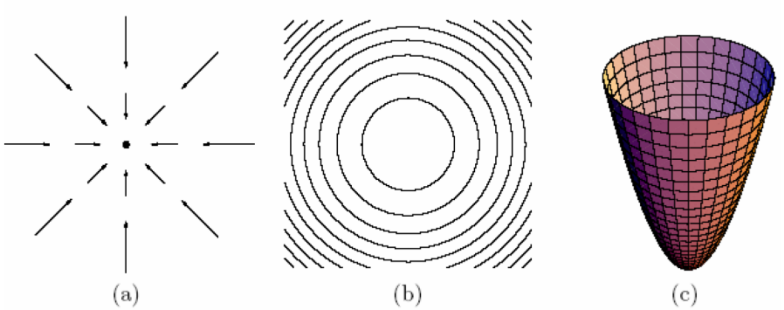




Generating artificial Potential Field

- Attractive potential field

quadratic potential



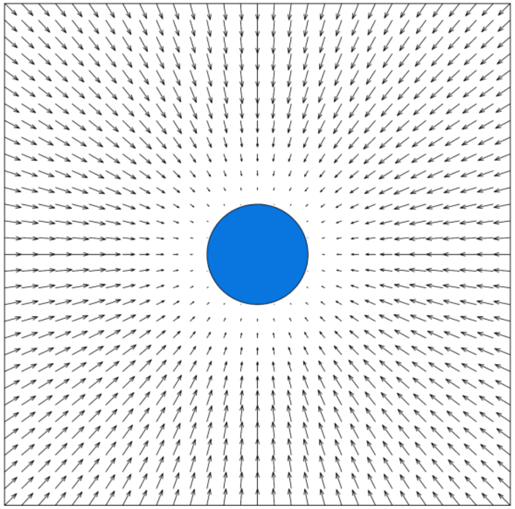
e.g of function of attraction: quadratic potential
Given:
q(x, y) $point\ of\ the\ space\ coordinate$
$q_a(x, y)$ $attraction\ sourcecoordinate$
$r\ rayon\ of\ the\ attration\ source$

$$U_{att} = \begin{cases} \frac{1}{2}\,\alpha\, d^2 & if\ d > r \\ 0 & if\ d \le r \end{cases}$$

α adjustable constant
d distance beetween point and attraction source such as:

$$d = |q - q_a| = \sqrt{\left(q_x - q_{a_x}\right)^2 + \left(q_y - q_{a_y}\right)^2}$$

Another linear attractive force field



Generating artificial Potential Field

- ❑ Repulsive potential field

e.g of function of repulsion:

Given:

$q(x,y)$ *point of the space coordinate*

$q_r(x,y)$ *repulsive source coordinate*

r *rayon of the repulsive source*

d_0 distance of repulsive source influence

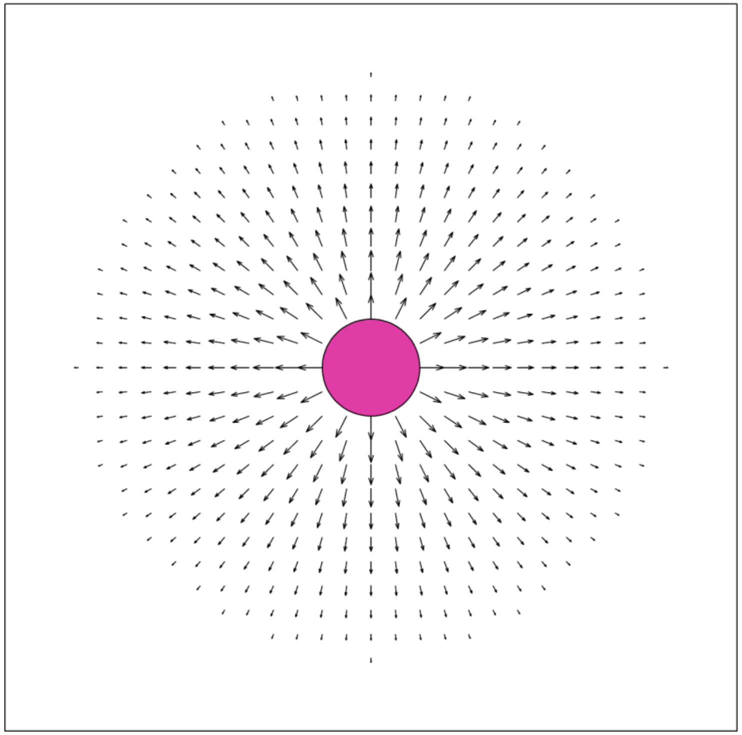
$$U_{rep} = \begin{cases} \frac{1}{2}\beta\left(\frac{1}{d} - \frac{1}{d_0}\right)^2 & if\ d \leq d_0 \\ 0 & if\ d > 0 \\ \infty & if\ d < r \end{cases}$$

β adjustable constant

d distance beetween point and repulsive source such as:

$$d = |q - q_r| = \sqrt{\left(q_x - q_{r_x}\right)^2 + \left(q_y - q_{r_y}\right)^2}$$

Another linear repulsive force field



Combining Potential Field

- Combining function, for a given function of a scalar potential field U where the robot is under the influence

$$U = U_{att} + U_{rep}$$

- The vector field of artificial forces F(p) is givent by the gradient of U

$$F(p) = -\nabla U_{att} + \nabla U_{rep}$$

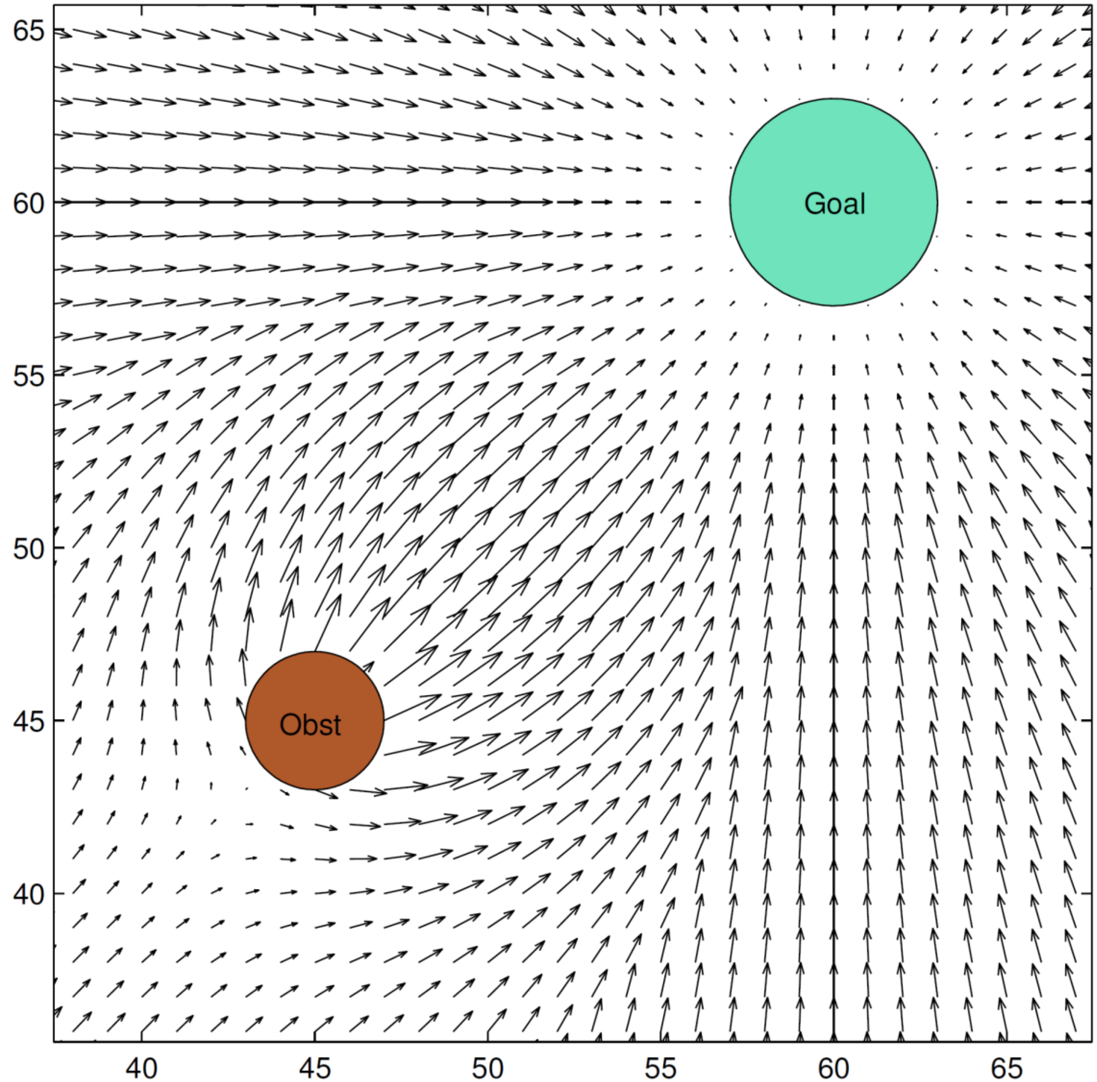
- If the cases where several repulsive forces are applied

$$F(p) = -\nabla U_{att} + \sum \nabla U_{rep_i}$$

CPE LYON
ÉCOLE SUPÉRIEURE DE CHIMIE PHYSIQUE ÉLECTRONIQUE DE LYON

Combining Potential Field

Another linear potential field combination



Combining Potential Field

$$U_{att} = \begin{cases} \frac{1}{2}\,\alpha\, d^2 & if\ d > r \\ 0 & if\ d \leq r \end{cases}$$

$$\boldsymbol{F_{att}(q) = -\nabla U_{att} = -\alpha(q - q_a)}$$

$$U_{rep} = \begin{cases} \frac{1}{2}\,\beta\,\left(\frac{1}{d} - \frac{1}{d_0}\right)^2 & if\ d \leq d_0 \\ 0 & if\ d > 0 \\ \infty & if\ d < r \end{cases}$$

- Should be:

$$\boldsymbol{F_{rep}(q) = \beta\left(\frac{1}{d} - \frac{1}{d_0}\right)(q - q_r)}$$

- But to adjust behavior and reduce local optimun in robot set of function could be used as Firas function:

$$\boldsymbol{F_{rep}(q) = \nabla U_{rep} = \begin{cases} \beta\left(\frac{1}{d} - \frac{1}{d_0}\right)\frac{(q - q_r)}{d^2} & if\ d \leq d_0 \\ 0 & if\ d > d_0 \end{cases}}$$

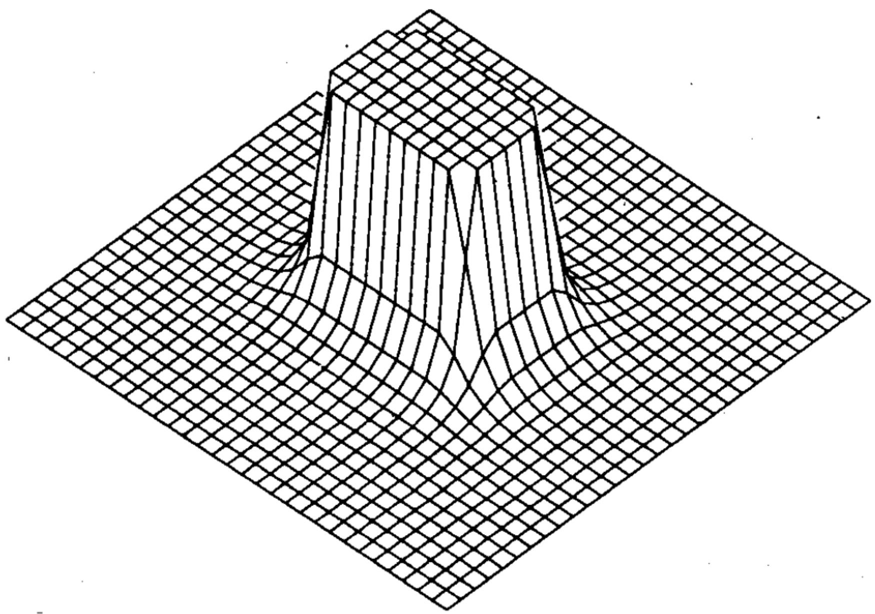


Fig. 2. FIRAS potential.

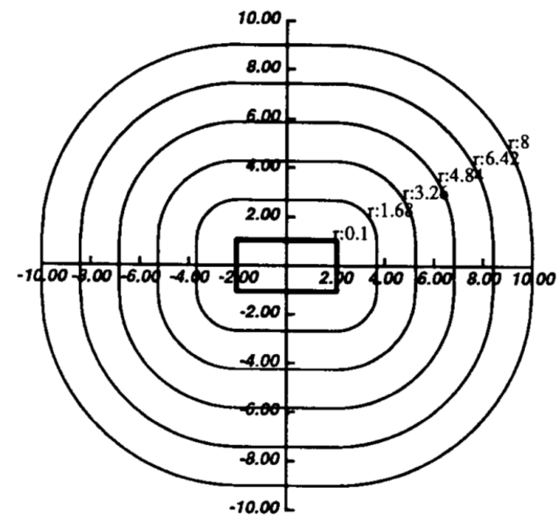
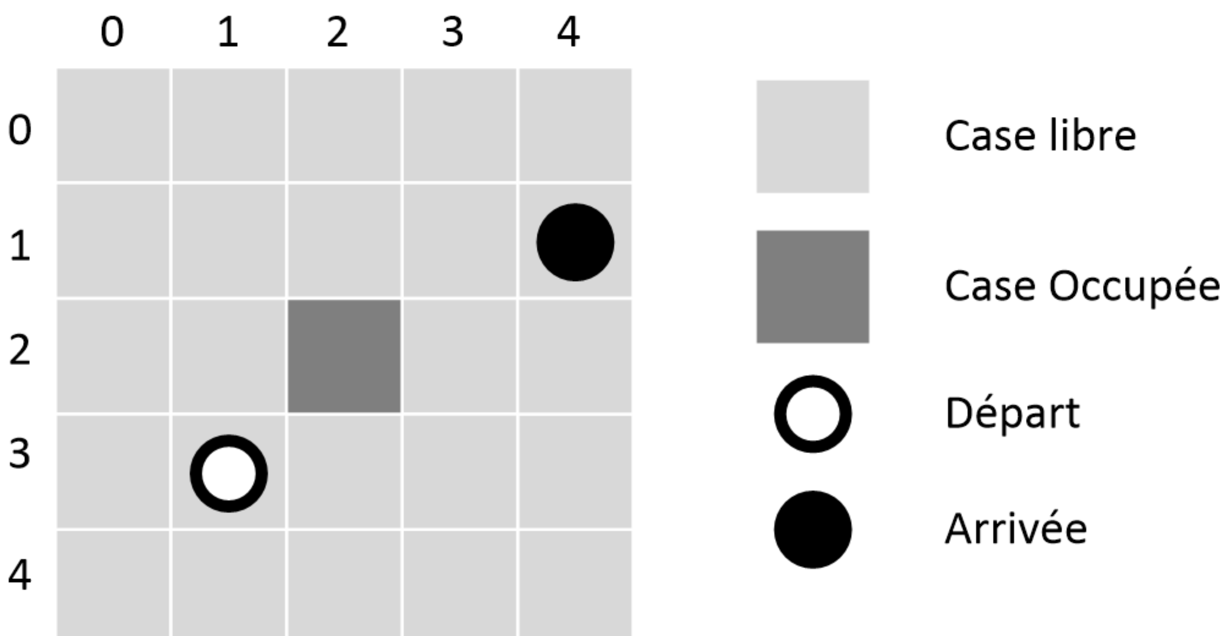


Fig. 3. Isopotential contours of FIRAS potential in Fig. 2.

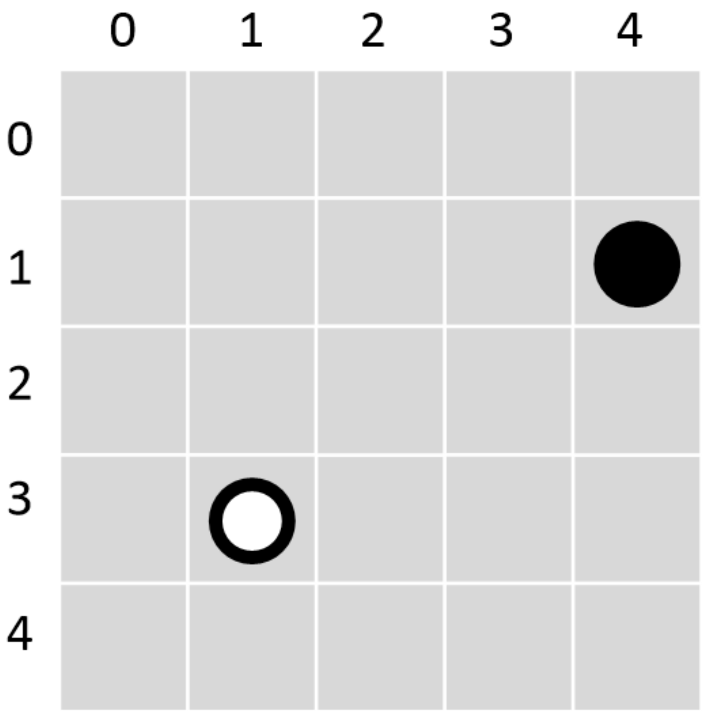
Compute your potential field on the given sample



- ❑ Compute force field applied on the robot

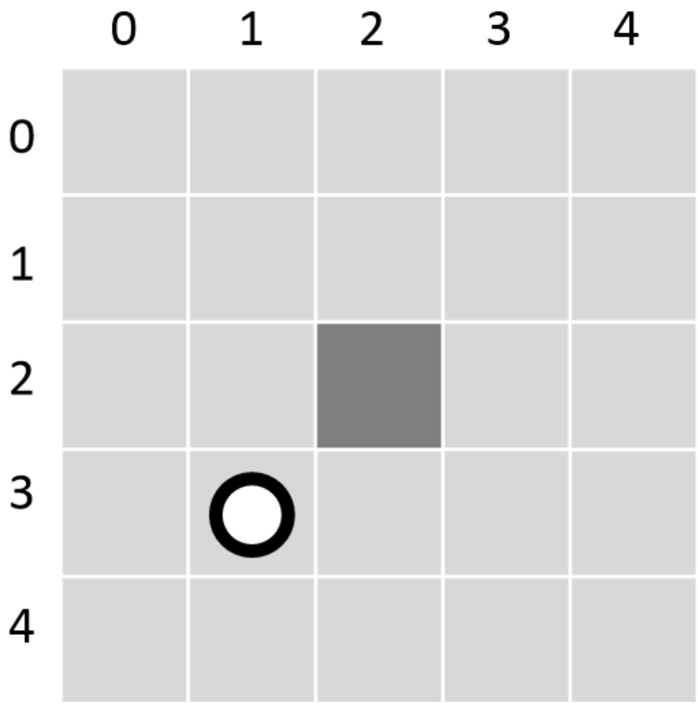
Compute your potential field on the given sample

- ❑ Compute attractive potential field applied on the robot



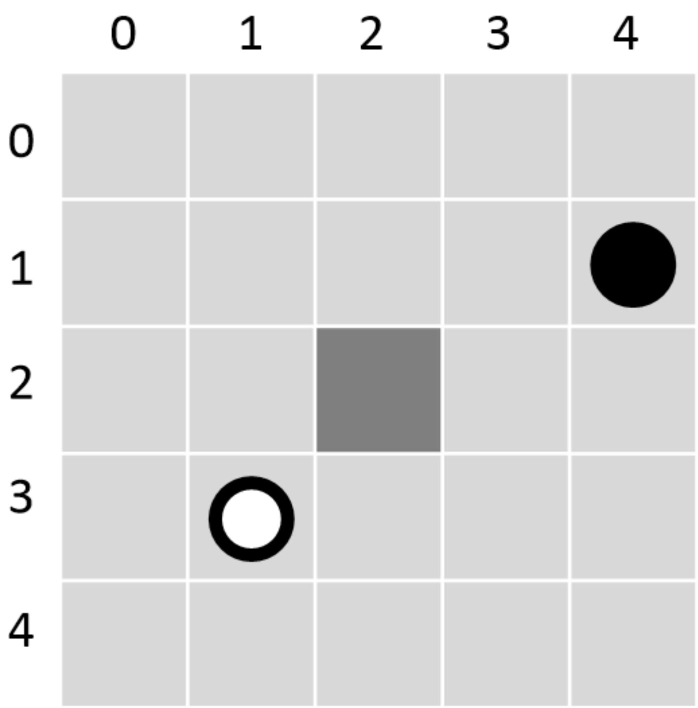
Compute your potential field on the given sample

- Compute repulsive potential field applied on the robot



Compute your potential field on the given sample

- Compute all potential fields applied on the robot



References

References (1/2)

- IEEE Standard 172-1983
- **Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004**
- **Robotics, Vision and Control, Springer, Peter Corke 2011**
- **PLANNING ALGORITHMS, Steven M. LaValle,University of Illinois, 2006**
  http://planning.cs.uiuc.edu/
- **Introduction to Mobile Robotics, Mapping with Known Poses, Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras, Uni Freiburg**
  http://ais.informatik.uni-freiburg.de/teaching/ss14/robotics/slides/08-occupancy-mapping-mapping-with-known-poses.pdf
- **Introduction to Mobile Robotics, Robot Motion Planning, Wolfram Burgard, cyrill stachniss, Maren Bennewitz, Kai Arras, Uni Freiburg, 2011**
  http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf
- **Occupancy Grids, Robotics, Benjamin Kuipers**
  https://www.cs.utexas.edu/~kuipers/slides/L13-occupancy-grids.pdf
- http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html
- http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume11/fox99a-html/node10.html
- http://www.geometrylab.de/applet-30-en
- http://cs.smith.edu/~streinu/Teaching/Courses/274/Spring98/Projects/Philip/fp/visibility.htm
- http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html
- https://fr.wikipedia.org/wiki/Diagramme_de_Vorono%C3%AF
- https://en.wikipedia.org/wiki/Voronoi_diagram
- http://www.barankahyaoglu.com/robotics/voronoirobot/
- https://en.wikipedia.org/wiki/Fortune%27s_algorithm
- http://msl.cs.uiuc.edu/rrt/index.html
- https://en.wikipedia.org/wiki/Probabilistic_roadmap
- http://msl.cs.uiuc.edu/rrt/gallery_rigid.html
- **http://www.redblobgames.com/pathfinding/a-star/introduction.html**
- https://en.wikipedia.org/wiki/A*_search_algorithm
- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
- http://buildnewgames.com/astar/

References (2/2)

- https://en.wikipedia.org/wiki/D*
- http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf
- Potential Fields Tutorial, Michael A. Goodrich, 2000
- Local Autonomous Robot Navigation using Potential Fields, Miguel A. Padilla Castañeda, Jesús Savage, Adalberto Hernández Fernando Arámbula Cosío, Intech 2008
- Robotic Motion Planning: Potential Functions, Robotics Institute 16-735, Howie Choset
- D* Lite And Dynamic Path Finding, Adrian Sotelo, Digipen Institute of Technology, 2009
- S. Koenig and M. Likhachev. D* Lite. In Proceedings of the AAAI Conference of Artificial Intelligence (AAAI), 476-483, 2002

images

- http://www.ros.org/news/2013/03/
- http://library.isr.ist.utl.pt/docs/roswiki/attachments/pr2_simulator(2f)Tutorials(2f)BasicPR2Controls/rviz_move_base_diverge.png
- http://www.youbot-store.com/developers/software/ros/youbot-ros-navigation-stack
- https://www.fsb.unizg.hr/acg/yaw_rate_estim_fig1.jpg
- http://library.isr.ist.utl.pt/docs/roswiki/costmap_2d.html
- http://joydeepb.com/Publications/biswas-rgbd11-plane-filtering.pdf
- http://www.cim.mcgill.ca/~mrl/pubs/saul/iros98.pdf
- http://www.sfbtr8.spatial-cognition.de/project/r3/HGVG/graphics/3DMzh.jpg
- http://www.mdpi.com/1424-8220/15/6/12736/htm
- http://a4academics.com/images/ProjSeminarImages/Ant-behavior-real-world.png

videos

- https://www.youtube.com/watch?v=A-fxij3zM7g
- https://www.youtube.com/watch?v=qziUJcUDfBc
- https://www.youtube.com/watch?v=zZLQ8Yh2iEE
- https://www.youtube.com/watch?v=DVnbp9oZZak
- https://www.youtube.com/watch?v=vAnN3nZqMqk
- https://www.youtube.com/watch?v=ylnI-9GctlTA

Jacques Saraydaryan
Jacques.saraydaryan@cpe.fr