Algoritmos para caminos más cortos para todos los pares de vértices

comp-420

Algoritmo de Floyd-Warshall

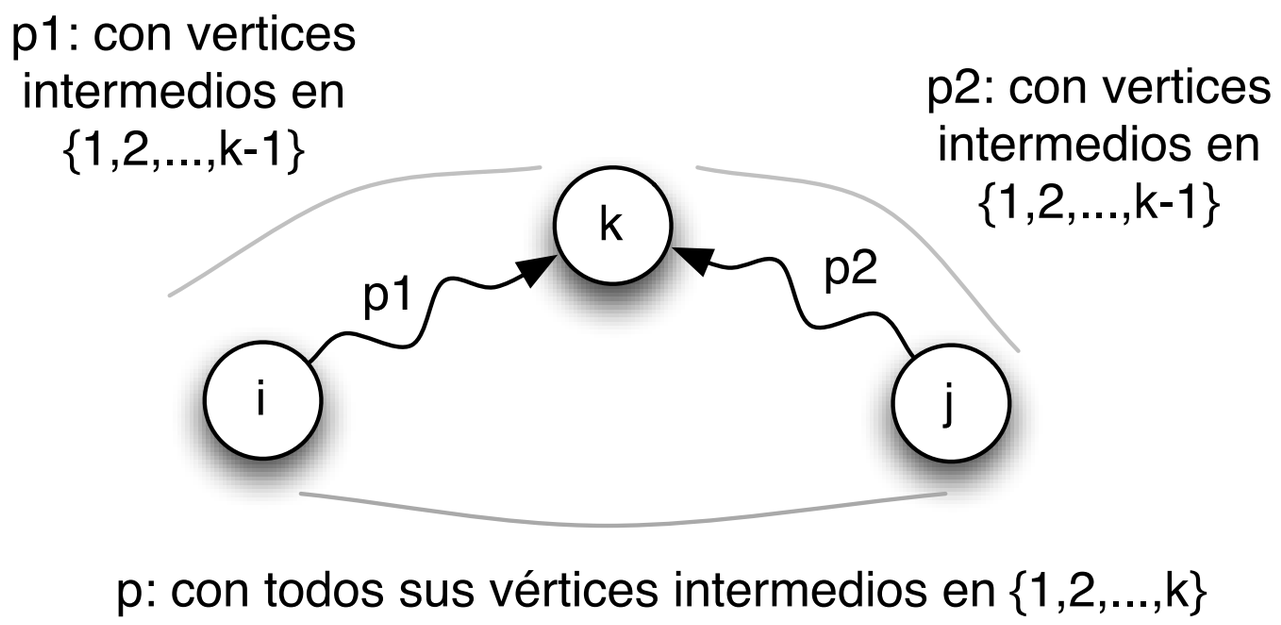
- Algoritmo de programación dinámica para encontrar los caminos más cortos entre todos los pares de vértices de un grafo dirigido G(V,E).
- Su tiempo de ejecución es de $\Theta(V^3)$.
- Puede haber aristas negativas pero no cíclos negativos.
- Algoritmo de programación dinámica:
  - Subestructura óptima.
  - Subproblemas que se traslapan.
  - "Memoisación"

Algoritmo de Floyd-Warshall

- El algoritmo considera los vértices intermedios del camino más corto.
- Para un camino simple $p=\langle v_1, v_2, \ldots, v_n \rangle$ es cualquier vértice que no sea v_1 o v_n, es decir, cualquier vértice del conjunto $\{v_2, v_3, \ldots, v_{n-1}\}$.
- Suponiendo que los vértices del grafo van de $V=\{1,2,\ldots,n\}$ consideremos el subconjunto de $V = \{1,2,\ldots,k\}$ para alguna k.
- Para cualquier par de vértices $i,j \in V$, consideramos todos los caminos que van de i a j cuyos vértices intermedios salen del conjunto $\{1,2,\ldots,k\}$.
- El algoritmo decide si un vértice k es o no un vértice intermedio del camino p.

Algoritmo de Floyd-Warshall

- Si k no es un vértice intermedio de p, entonces todos los vértices intermedios de p están en el conjunto {1,2,...k-1}.
- En este caso el camino más corto de i a j con todos sus vértices intermedios en el conjunto {1,2,...k-1} también están en el conjunto {1,2,...k}.
- Si k es un vértice intermedio del camino a p, entonces dividimos a p:



Algoritmo de Floyd-Warshall

- Algoritmo recursivo.
- Sea $d_{ij}^{(k)}$ el peso del camino más corto del vértice i al vértice j con vértices intermedios en el conjunto {1,2,...,k}.
- Con k=0 un camino del vértice i al vértice j sin vertices intermedios numerados arriba de 0 no tiene vértices intermedios del todo. Este camino tiene a lo más una arista, $d_{ij}^{(0)} = w_{ij}$.
- La recursión es entonces:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

- Para cualquier camino, todos los vértices intermedios están en el conjunto {1,2,...,n}.

Algoritmo de Floyd-Warshall

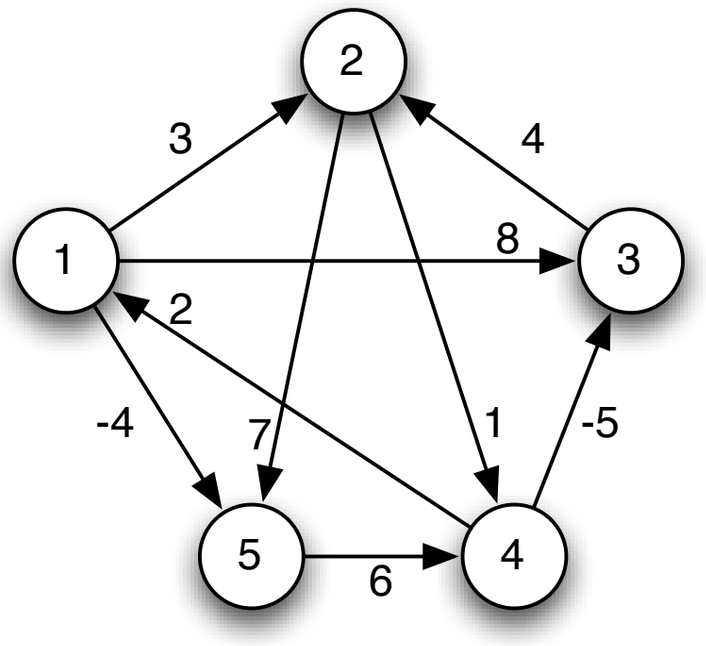
- La matriz $D^{(n)} = (d_{ij}^{(n)})$ da el resultado final:
  - $d_{ij}^{(n)} = \delta(i,j)$ para todo $i,j \in V$.
- Procedimiento bottom-up es usado para calcular dij(k) mientras se va aumentando el valor de k.
- W es una matriz de entrada con dimensión nxn definida como:

$$w_{ij} = \begin{cases} 0 & \text{si } i = j, \\ \text{el peso de la arista dirigida } (i,j) & \text{si } i \neq j \text{ y } (i,j) \in E, \\ \infty & \text{si } i \neq j \text{ y } (i,j) \notin E. \end{cases}$$

Algoritmo de Floyd-Warshall

```
FLOYD-WARSHALL (W)
 1  n ← rows[W]
 2  D^(0) ← W
 3  for k ← 1 to n
 4      do for i ← 1 to n
 5          do for j ← 1 to n
 6              do d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
 7  return D^(n)
```

Algoritmo de Floyd-Warshall

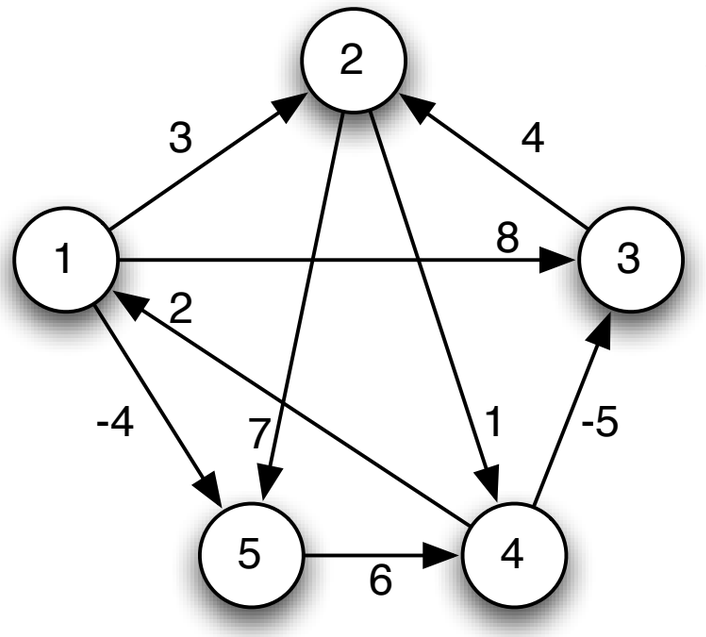


$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

Algoritmo de Floyd-Warshall



$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Algoritmo de Floyd-Warshall

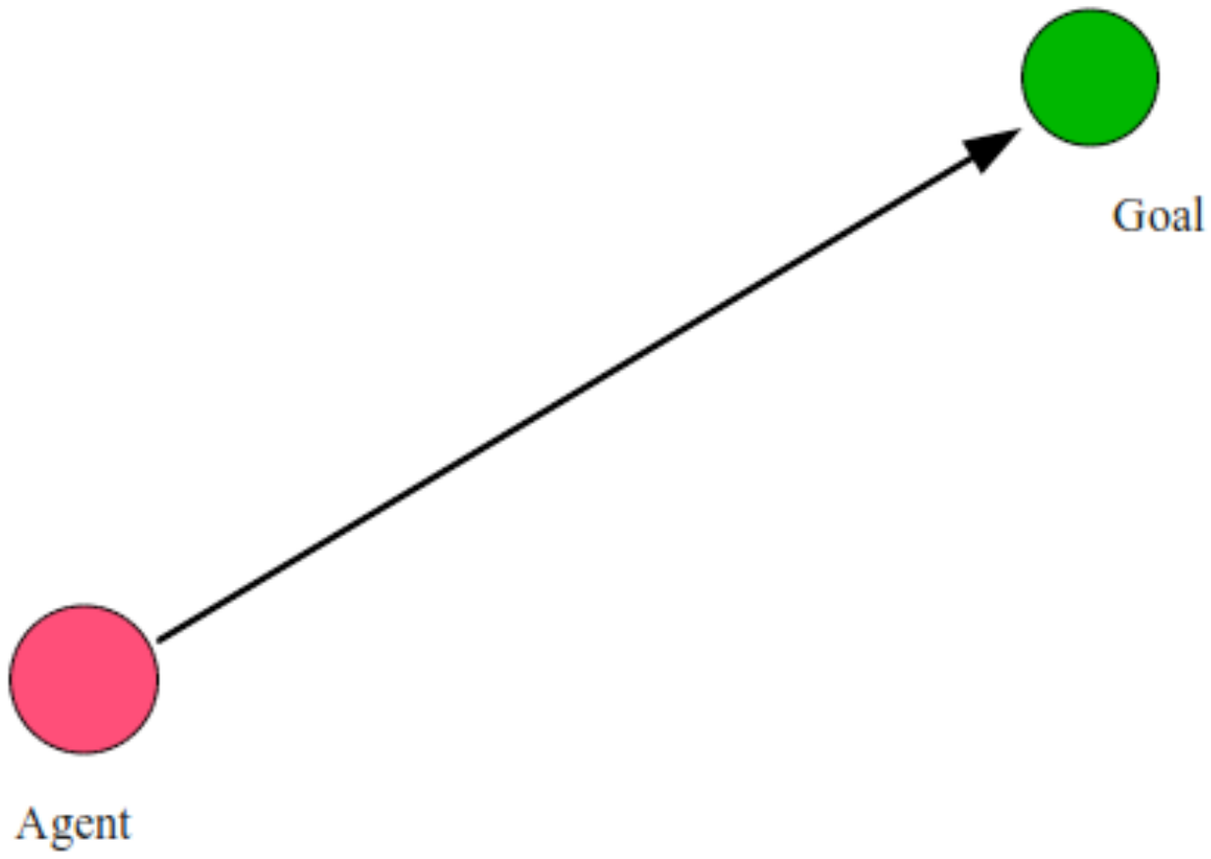
```
PRINT-ALL-PAIRS-SHORTEST-PATH (Π, i, j)
1  if i = j
2      then print i
3      else if π_ij = NIL
4          then print "no path from" i "to" j "exists"
5          else PRINT-ALL-PAIRS-SHORTEST-PATH (Π, i, π_ij)
6              print j
```

Introducción a algoritmos de Path Planning

comp-420

Moviendo de un punto A a un punto B: agentes y meta

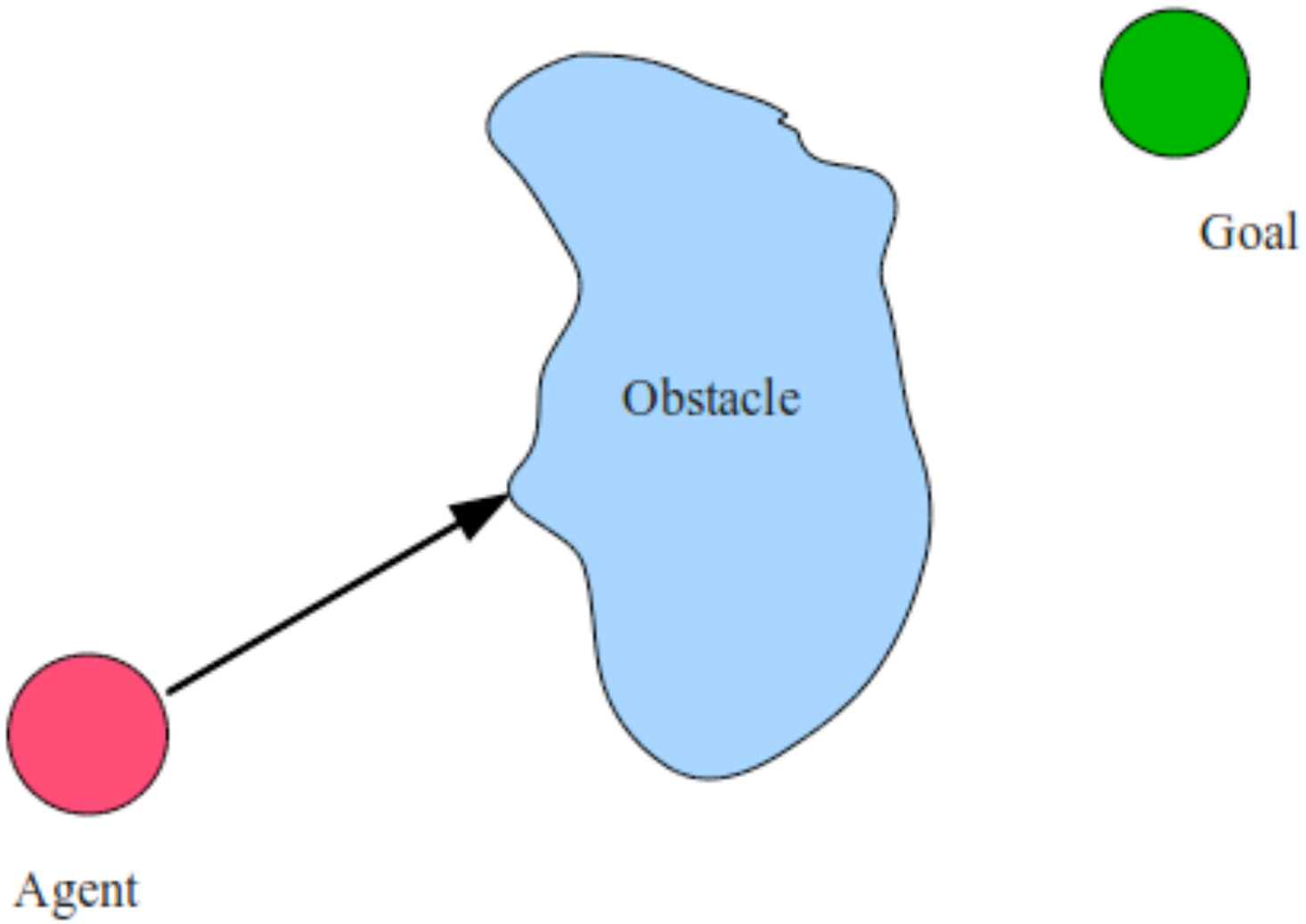
```
Algorithm: WALK TO

 1.If not at goal:
    1.Move toward goal.
 2.Else: Stop.
```



- Un segmento de recta es el camino más corto en distancia y en tiempo para llegar de un punto a otro.
- En este sentido, el algoritmo Walk TO es **Optimo**.

http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Path planning con obstáculos

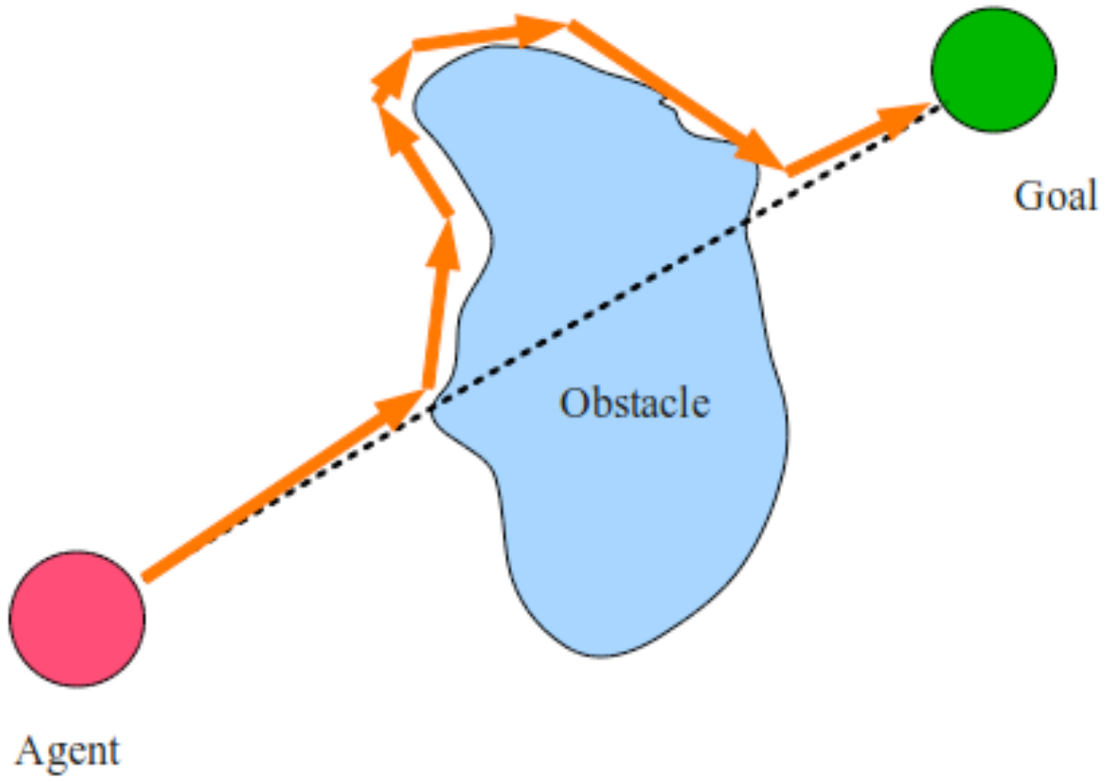


http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Planning Reactivo: Bug Algorithm

Algorithm: BUG

1. Draw a straight line from the Agent to the Goal. Call this the M-Line.
2. The Agent takes a single step along the M-Line until one of the following conditions is met:
   1. If the agent reaches the goal, stop.
   2. If the agent touches an obstacle, follow the obstacle clockwise until we reach the M-Line again, and also if we are closer to the goal than we were initially upon reaching the M-line. When we reach the M-Line, continue from step 2.



- M-Line es una Heurística
- El Bug Algorithm resuelve todas las instancias de problemas con solucion para un disco en 2D

http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Visibility Graph: caminos óptimos

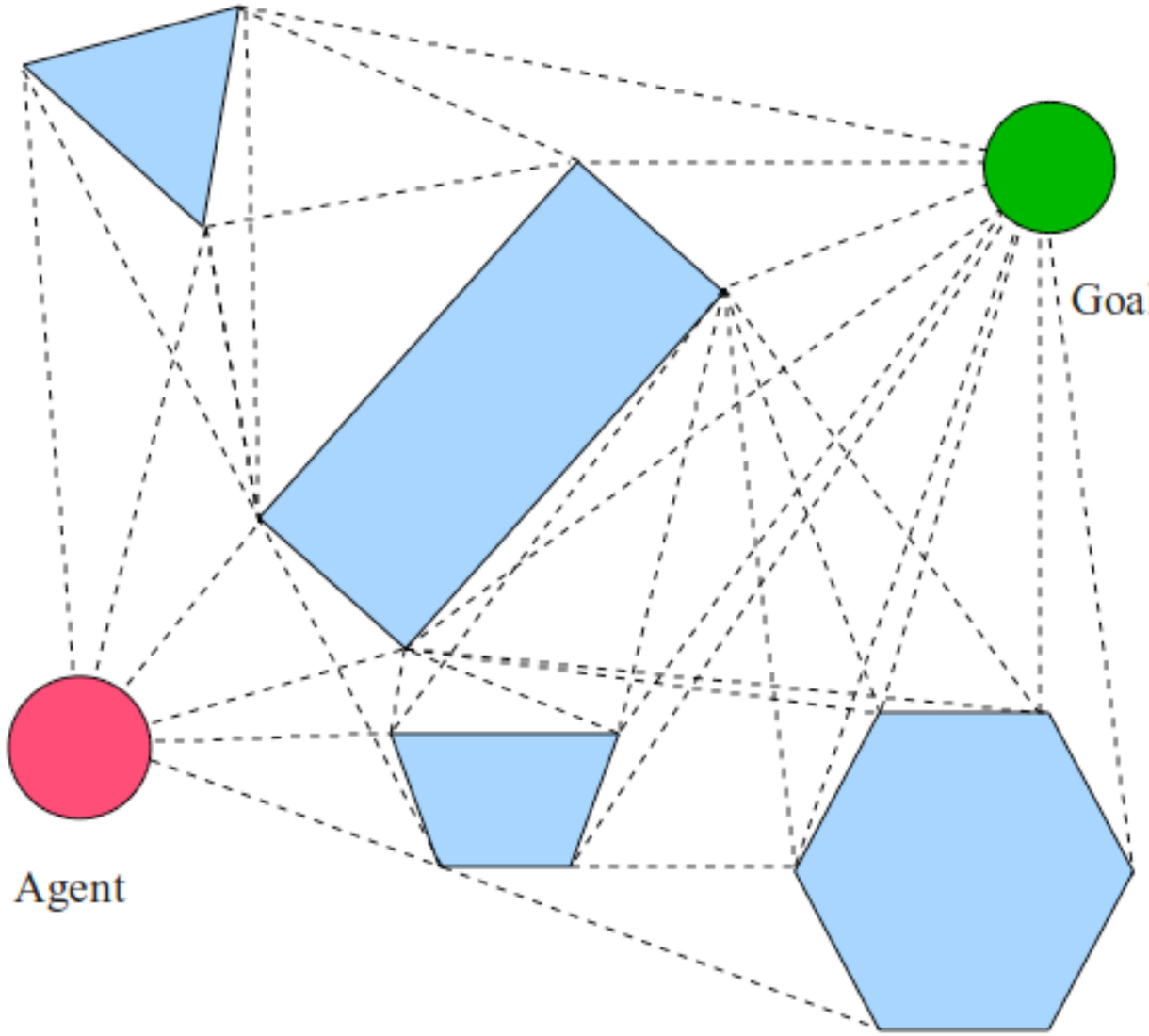
- En R^2, la trayectoria mas cortas entre dos puntos es un segmento de recta.
- La longitud de una secuencia de trayectorias formada de segmentos de recta es la suma de la longitud de cada segmento.
- Como un polígono está formado de segmentos de recta (o aristas), el camino más corto que circunavega un polígono convexo contiene todas las aristas del polígono.
- Para circunavegar un polígono no-convexo, el camino más corto contiene el envolvente convexo del polígono.
- Un punto que está exactamente en una arista del polígono colisiona con el polígono.
- Nuestro agente es un punto en R^2, infinitamente pequeño.
- Todos los obstáculos en el ambiente se pueden aproximar por polígonos cerrados.
- El agente puede moverse en cualquier dirección sobre el segmento de recta.

http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Visibility Graph: caminos óptimos

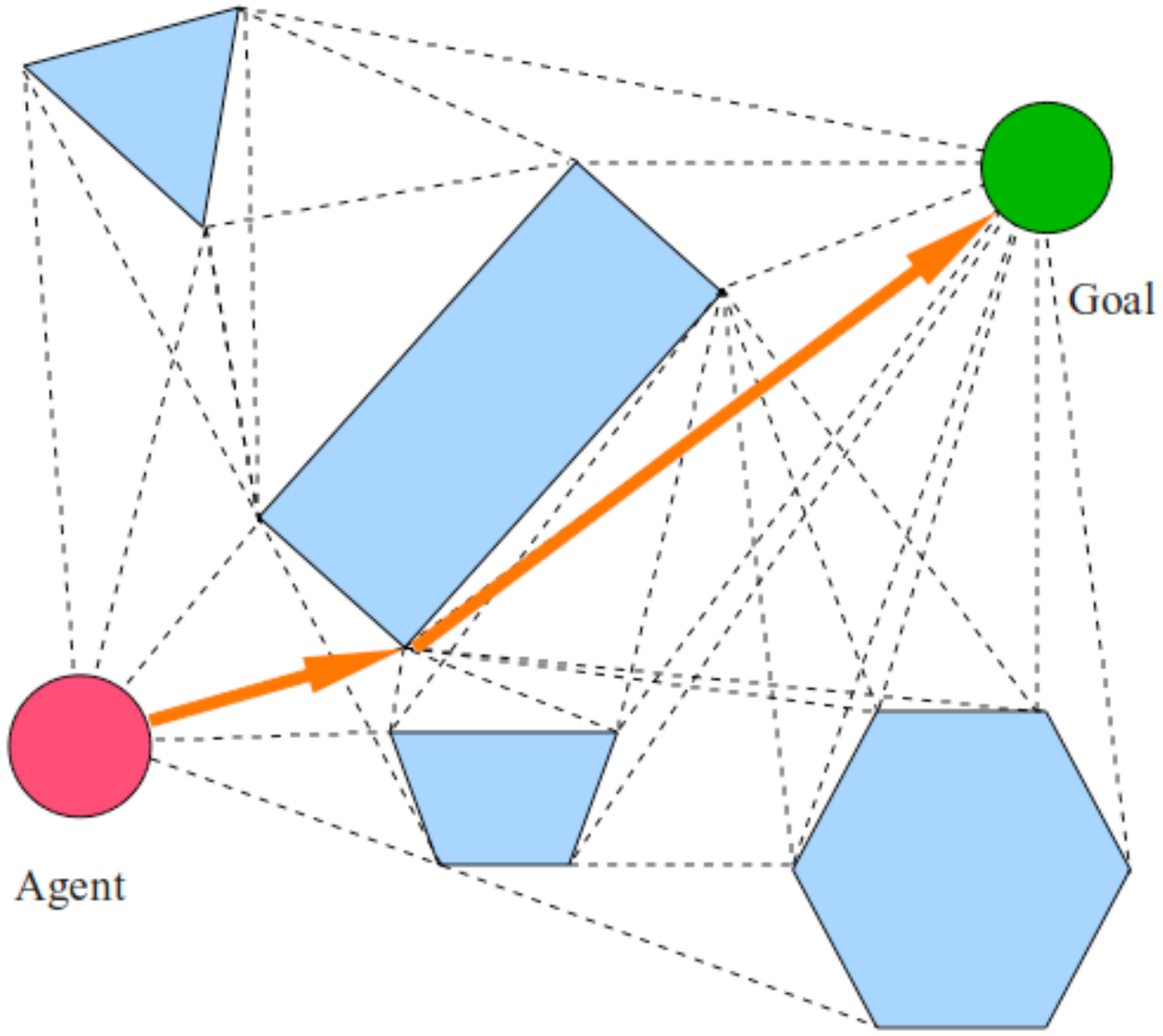
Algorithm: VISIBILITY GRAPH PLANNER

- Create an empty Graph called G.
- Add the Agent as a vertex in G.
- Add the Goal as a vertex in G.
- Place all of the vertices of all of the polygons in the world on a list.
- Connect all of the polygon's vertices in G according to their edges.
- For each vertex V on the list:
  - Try to connect V to the Agent and the Goal. If the straight line between them does not intersect any polygon, add this line as an edge in G.
  - For every other vertex in every other polygon, try to connect it to V using a straight line. If the line does not intersect any polygon, add the line as an edge in G.
  - Now, run a Graph Planner (like Dijkstra's or A*) on the Visibility graph to find the shortest path from the Agent to the Goal.



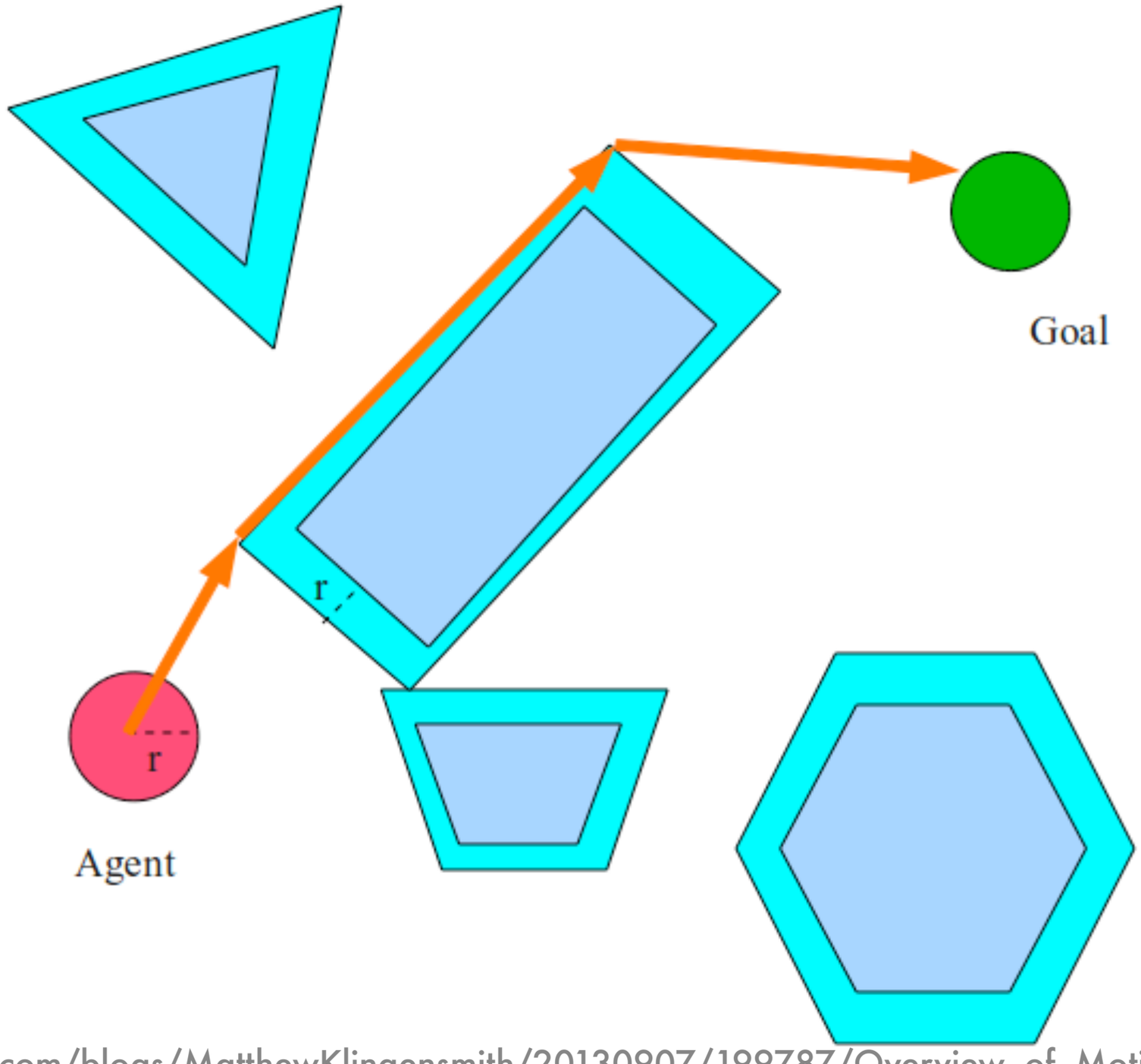
http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Visibility Graph: caminos óptimos



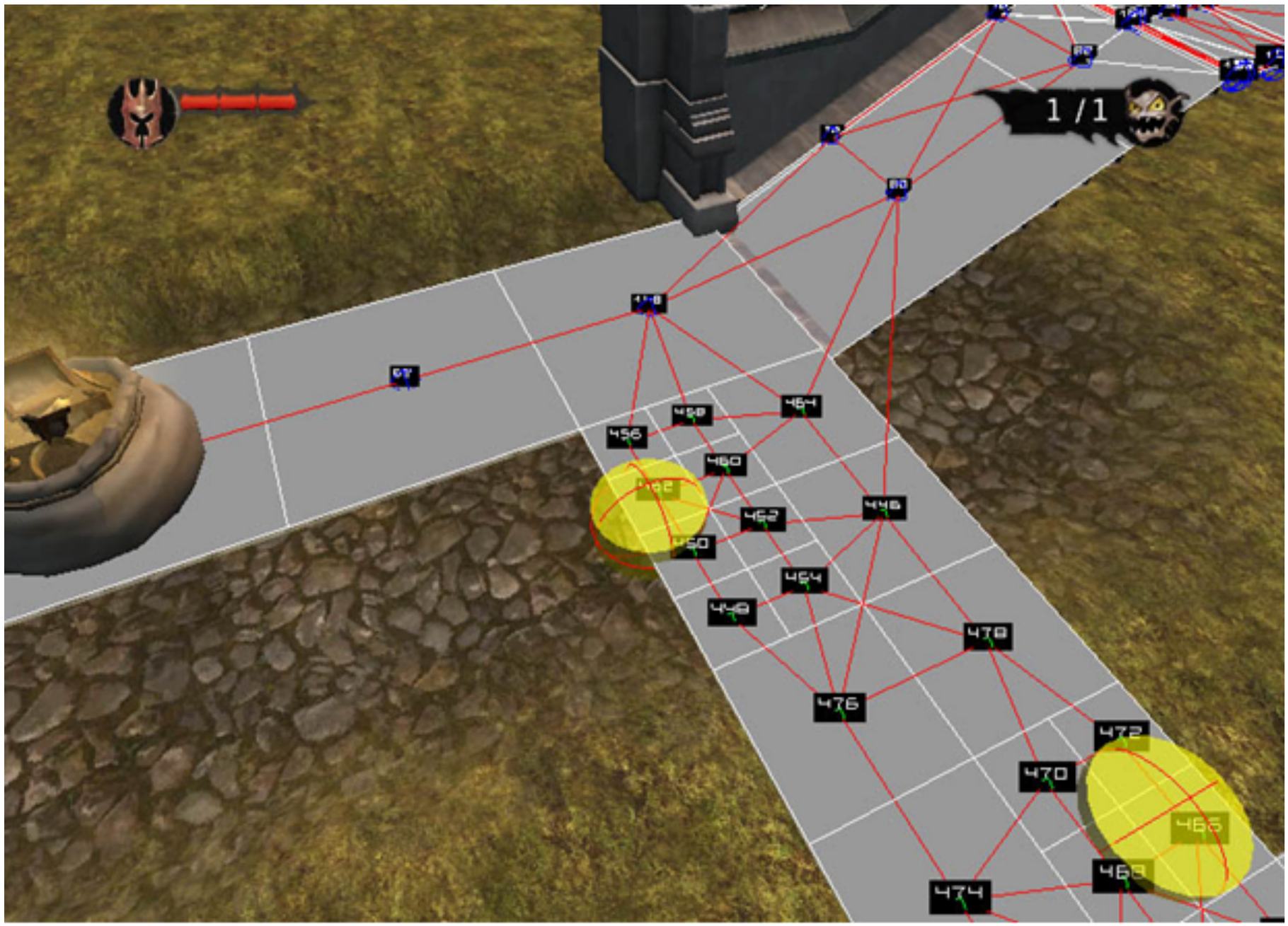
http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

¿Qué pasa si el agente no es un punto?



http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Navigation Mesh Example

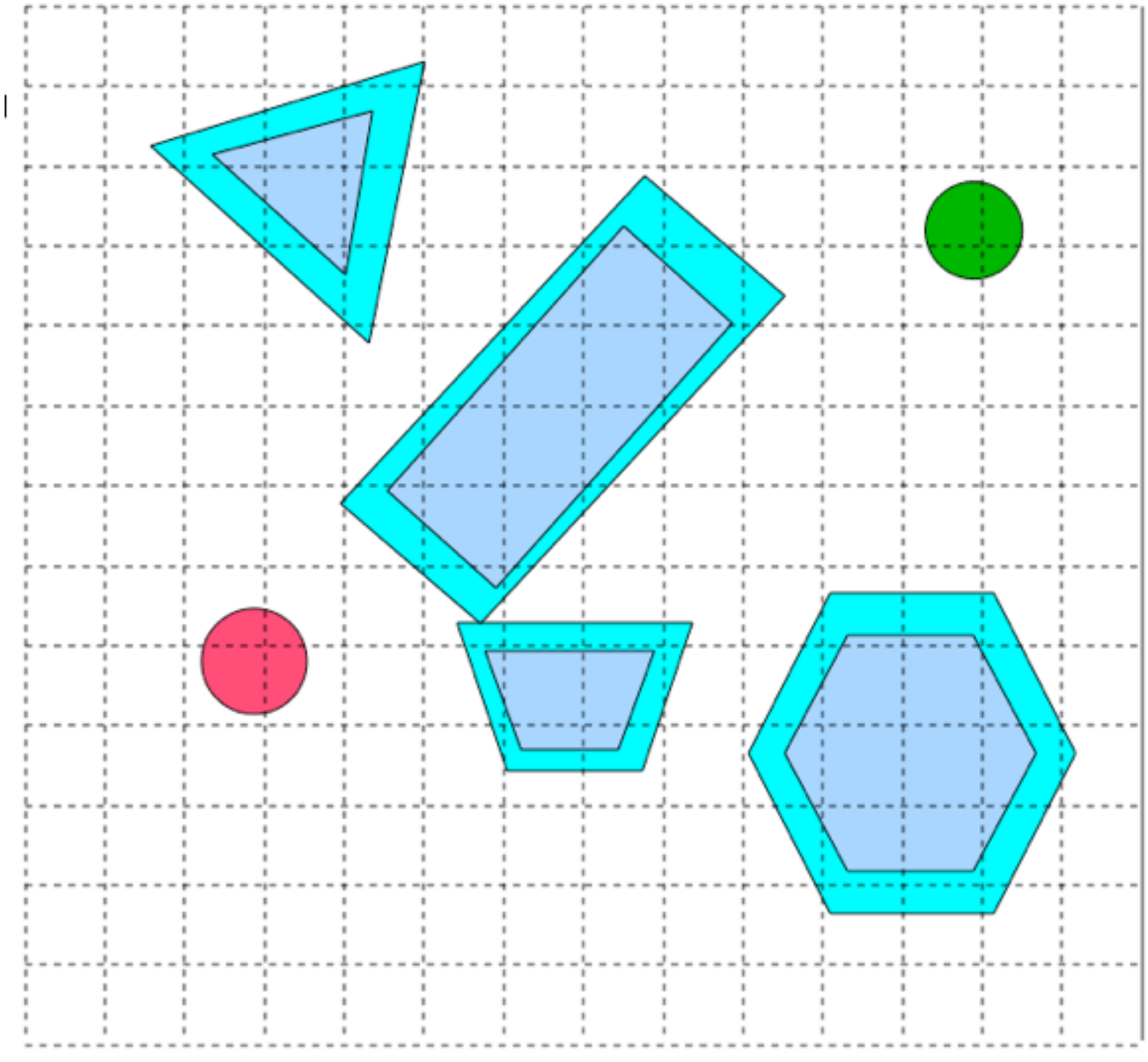


http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Lattice Grid Search: A*

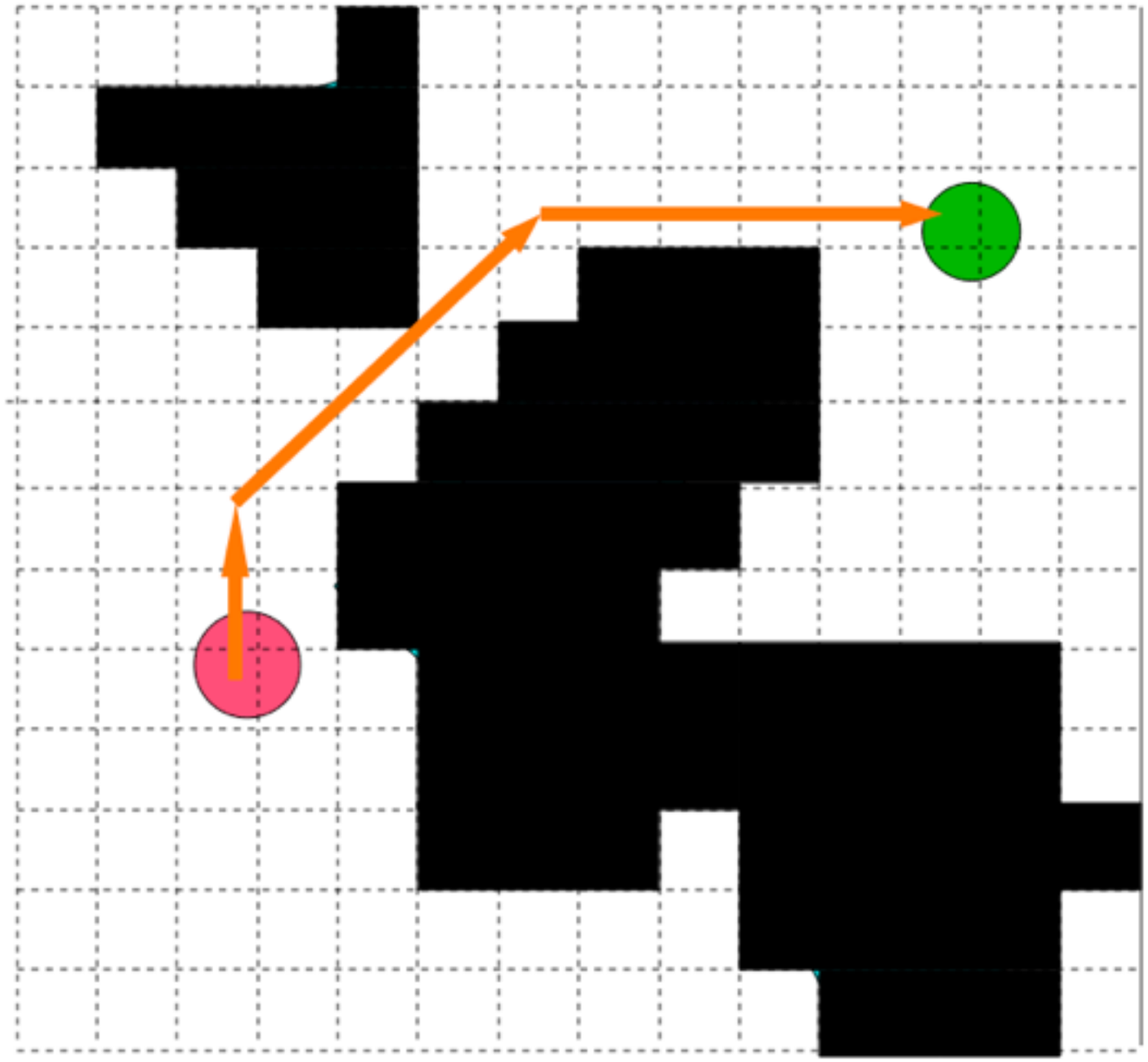
Algorithm: DISCRETIZE SPACE

- Assume the Configuration Space has some fixed size in all its dimensions.
- Discretize each dimension so that it has a fixed number of cells.
- For each cell whose center is inside an obstacle in the Configuration Space, mark it Impassable.
- Likewise, for each cell whose center is outside an obstacle, mark it Passable.
- Each Passable cell is now a Node.
- Each Node connects to all its "adjacent" Passable neighbors in the graph.



http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php

Lattice Grid Search: A*



http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php