# INSTITUTO TECNOLÓGICO DE NUEVO LAREDO

## INGENIERÍA EN SISTEMAS COMPUTACIONALES


INTELIGENCIA ARTIFICIAL

ING. BRUNO LOPEZ TAKEYAS

ALGORITMO A *


| | |
|---|---|
| LAURA JOSEFINA GALVAN HERNANDEZ | 01100205 |
| MARTHA GUADALUPE GALVAN HERNANDEZ | 01100206 |
| RUBEN GARCIA MARTINEZ | 01100211 |
| JULISSA NEREYDA GARCIA NUÑEZ | 01100212 |
| MAGDALENA VILLARREAL DÍAZ | 01100327 |


NUEVO LAREDO, TAM., A 17 DE OCTUBRE DEL 2005.

**Algoritmo A* (A - star)**

## Introducción

La búsqueda de caminos es un problema que usualmente se resuelve por medio de algoritmos de búsqueda en grafos como ancho-primero, profundidad-primero, Dijkstra y A*.

En este tipo de problemas conocemos un estado inicial, un estado final (meta) y un conjunto de reglas mediante las cuales podemos realizar "movidas" para ir avanzando en un grafo compuesto por nodos. Eventualmente, avanzando por el grafo, llegaremos al nodo que corresponde a la meta habiendo finalizado la búsqueda y habiendo obtenido el conjunto de movidas que nos lleva de un punto a otro.

El algoritmo A* es uno de los preferidos para este tipo de aplicaciones.

Contar con información sobre un espacio de estados evita a los algoritmos emprender búsquedas a ciegas, permitiendo así encontrar soluciones con más **eficiencia**.

Por esto se le considera al algoritmo A* como un método de búsqueda respaldado con información.

## ¿Cómo Surge A*?

El algoritmo de búsqueda A* surge de la combinación del método avaro y la búsqueda por costo uniforme.

La búsqueda avara permite reducir al mínimo el costo de la meta, *h(n)*, con lo que también se reduce en forma considerable el costo de la búsqueda. Desafortunadamente este tipo de búsqueda no es óptimo ni tampoco completo.

La búsqueda por costo uniforme, reduce al mínimo el costo de la ruta, *g(n)*; es óptima y completa, pero puede ser muy ineficiente.

Sería bueno poder utilizar ambas estrategias para así combinar las ventajas que ofrecen. Afortunadamente sí es posible hacerlo, al combinar las dos funciones de evaluación mediante una suma:

$$f(n) = g(n) + h(n)$$

Dado que con *g(n)* se calcula el costo de la ruta que va del nodo de partida al nodo *n*, y *h(n)* es el costo estimado de la ruta más barata que va de n a la meta, tenemos que:

*f (n)* = costo estimado de la solución más barata, pasando por *n*

Es decir, si lo que se trata es de encontrar la solución que sea más barata, entonces es razonable probar primero en el nodo cuyo valor de *f* sea el más bajo. Lo interesante de esta estrategia es que no sólo es razonable, sino que se puede demostrar que es completa y óptima, dada una sencilla restricción de la función *h.*

La restricción consiste en escoger una función *h* que *nunca sobreestima* el costo que implica alcanzar la meta. A tal *h* se le conoce como función **heurística admisible**. Por naturaleza, este tipo de heurísticas son optimistas, pues consideran que el costo para resolver un problema siempre es inferior a lo que en realidad es. Este optimismo se transfiere a la función *f: si h es aceptable, f(n) nunca sobreestimará el costo real de la mejor solución que pase por n.* A la búsqueda preferente por lo mejor, en la que se utiliza *f* como la función de evaluación y una función *h* aceptable se le conoce como **búsqueda A*.**

## ¿Qué Es A*?

A* es un algoritmo de búsqueda inteligente o respaldado por información que busca el camino más corto desde un estado inicial al estado meta a través de un espacio de problema usando una heurística optima. Como ignora los pasos más cortos (más "chatos") en algunos casos rinde una solución subóptima.

Pertenece al método de búsqueda preferente por lo mejor (Best First Search). Es muy popular en la planificación de rutas y ha sido exitosamente utilizado en áreas como Inteligencia Artificial y Robótica.

A* es un algoritmo genérico de busqueda que utiliza información heurística para determinar cual es el mejor camino hacia el destino. La información que utiliza es el costo estimado desde el nodo explorado hacia el nodo destino. Esta información es proporcionada por la función llamada *función heurística h'(n)*.

A* es un algoritmo informado que basa su comportamiento en la evaluación de una función expresada del siguiente modo:

$$f(n) = g(n) + h(n)$$

Función de evaluación:

- *g(n):* costo del mejor camino desde el estado inicial al estado *n*.
- *h(n)*: estimación (heurística) del coste desde n hasta un estado final óptimo.
- *f(n):* costo estimado de la mejor solución que pasa por el estado *n*.

$$\left.\begin{array}{l} h^*(n) \\ g^*(n) = g(n) \\ f^*(n) = g^*(n) + h^*(n) \end{array}\right\}$$ Costos reales conocidos cuando termina el algoritmo.

## Características Del Algoritmo De Búsqueda A*

Es óptimo y completo si:

- Todo nodo tiene un nº finito de sucesores.
- El costo de cada arco u operador > 0.
- La función *h(n)* es una heurística admisible.

Es *óptimo* porque encuentra la mejor solución si existen varias.

Es *completo* porque garantiza encontrar una solución, si existe.

*Heurística admisible*

Diremos que h(n) es una heurística admisible si nunca sobreestima el coste real desde n hasta un estado meta óptimo.

## ¿Cómo Funciona A*?

Incorpora la longitud del camino desde la raíz hasta el estado actual en la función de evaluación *h*, considera no sólo lo bueno que es un estado sino que tiene en cuenta cómo es el camino usado para alcanzarlo.

## Estructuras utilizadas

- *Lista abierta:* contiene los nodos que podrían formar parte del camino que queremos tomar, pero que quizás no lo hagan.
- *Lista cerrada:* contiene los nodos que ya han sido examinados y que no hace falta volver a examinar.

## Métodos de Búsqueda:

**Búsqueda preferente por lo mejor**

Se utiliza esta búsqueda cuando los nodos se ordenan de manera tal que se expande primero aquel con mejor evaluación. Su objetivo es encontrar soluciones de bajo costo, por lo general estos algoritmos utilizan alguna medida estimada del costo de la solución y tratan de reducir esta medida al mínimo.

Para enfocar la búsqueda, debe figurar algún tipo de cálculo del costo de ruta que va de un estado al más cercano a la meta. Se trata de dos tipos de aproximaciones básicas: el primero trata de expandir el nodo más cercano a la meta, y el segundo el correspondiente a la ruta de la solución menos costosa.

## Complejidad de A*

La dificultad reside en que, para la generalidad de los problemas, la cantidad de nodos que están dentro del espacio de búsqueda en el contorno de la meta sigue siendo exponencial a lo largo de toda la solución.

## Desventaja de A*

- El tiempo de cómputo

BUSQUEDA LIMITADA POR LA CAPACIDAD DE LA MEMORIA

En esta sección se exploran dos algoritmos diseñados para conservar la memoria.

- ☑ Búsqueda A* por profundización iterativa (A*PI)
- ☑ Búsqueda A*SRM (A* acota por memoria simplificada)

Búsqueda A* por profundización iterativa (A*PI): en este caso la búsqueda preferente por profundidad se modifica para utilizar un límite del costo de $f$ en lugar de un límite de profundidad. De esta forma, en cada iteración se expanden todos los nodos que están dentro del contorno del costo $f$ actual, y se echa un vistazo al contorno para determinar en donde se encuentra el siguiente contorno. Una vez concluida la búsqueda dentro de un contorno, se procede a efectuar una nueva iteración utilizando un nuevo costo $f$ Para el contorno siguiente.

A*PI es un método de búsqueda completo y optimo, con las mismas ventajas y desventajas de la búsqueda A*, pero puesto que es preferente por profundidad, solo necesita de un espacio proporcional con la ruta mas larga que explore. Si * es el mínimo costo de operador y f * es el costo de la solución optima, en el peor de los casos en A*PI se necesitan bf*/* nodos de almacenamiento. En la mayoría de los casos, bd es un estimado bastante bueno de la capacidad de memoria necesaria.

**Function** IDA* (problem) **returns** a solution sequence
**Función** A*PI(problem) **responde con** una secuencia de solución
**Entradas**: problema, un problema
**Estático**: limite-f, el limite actual de COSTO-F
Raíz, un nodo

Raíz ← HACER-NODO (ESTADO-INICIAL [problema])
Límite-f ← COSTO-raíz)
**Bucle hacer**
Solucion, limite-f ←DFS-CONTORNO (raíz, limite-f)
si solución no es nula **entonces responde** con solución
si limite-f=∞, **entonces responde con** falla; **fin**

---

**Function** DFS-CONTOUR (node, f-limit) **returns** a solution sequence and a new f-COST limit
**Función** CONTORNO-DFS (nodo, limite-f) **responde con** una secuencia de solución y un nuevo limite de COSTO-f
**Entradas**: nodo, un nodo
Límite-f, el límite actual de COSTO-f
**Estático**: siguiente-f, el límite de COSTO-f correspondiente al siguiente contorno, inicialmente ∞

**si** COSTO-f nodo] limite-f, **entonces responde con** nulo, COSTO-f[nodo
**si** PRUEBA-META[problema](ESTADO[NODO]) **entonces responde con** nodo, limite-f

```
por cada nodo s en SUCESORES(nodo) hacer
    solución, nueva-f ← CONTORNO –DFS(s, limite-f)
    si solución no es nula, entonces responde con solución, limite-f
    siguiente-f ← MIN(siguiente-f, nueva-f); fin
 responde con nulo, siguiente-f
```

Figura 1. El algoritmo de búsqueda A*PI (A* Por profundización iterativa)

La complejidad temporal de A*PI depende fundamentalmente de la cantidad de distintos valores que adopte la función heurística. La heurística de la distancia de Maniatan utilizada en el problema de las ocho placas utiliza uno de un reducido grupo de números enteros. Por lo general.

Desafortunadamente, A*PI tiene problemas en dominios mas complejos. En el problema del agente de ventas viajero, por ejemplo, el valor heurística cambia por cada estado. Es decir, en cada contorno solo figura un estado más que los del contorno previo. Si A* expande N nodos, A*PI, tendrá que pasar por N iteraciones y expanderà 1+2+….+N= O (N^2) nodos. si N es demasiado grande para la memoria de la computadora, indudablemente que N^2 tendrá que esperar bastante.

Una solución a lo anterior consiste en aumentar el límite del costo f mediante una cantidad fija en cada iteración, de manera que la cantidad total de iteraciones sea proporcional a 1/E. Esto permitira reducir el costo de búsqueda a expensas de regresar a soluciones que pueden ser peores mas que optimas por a lo mas E. A este algoritmo se le conoce admisible en E.

☑ BUSQUEDA A*SRM (acota por memoria simplificada)

Emplea toda la capacidad de memoria disponible para efectuar una búsqueda. El empleo de más memoria permite mejorar la eficiencia en la búsqueda.
A*SRM se le caracteriza por:

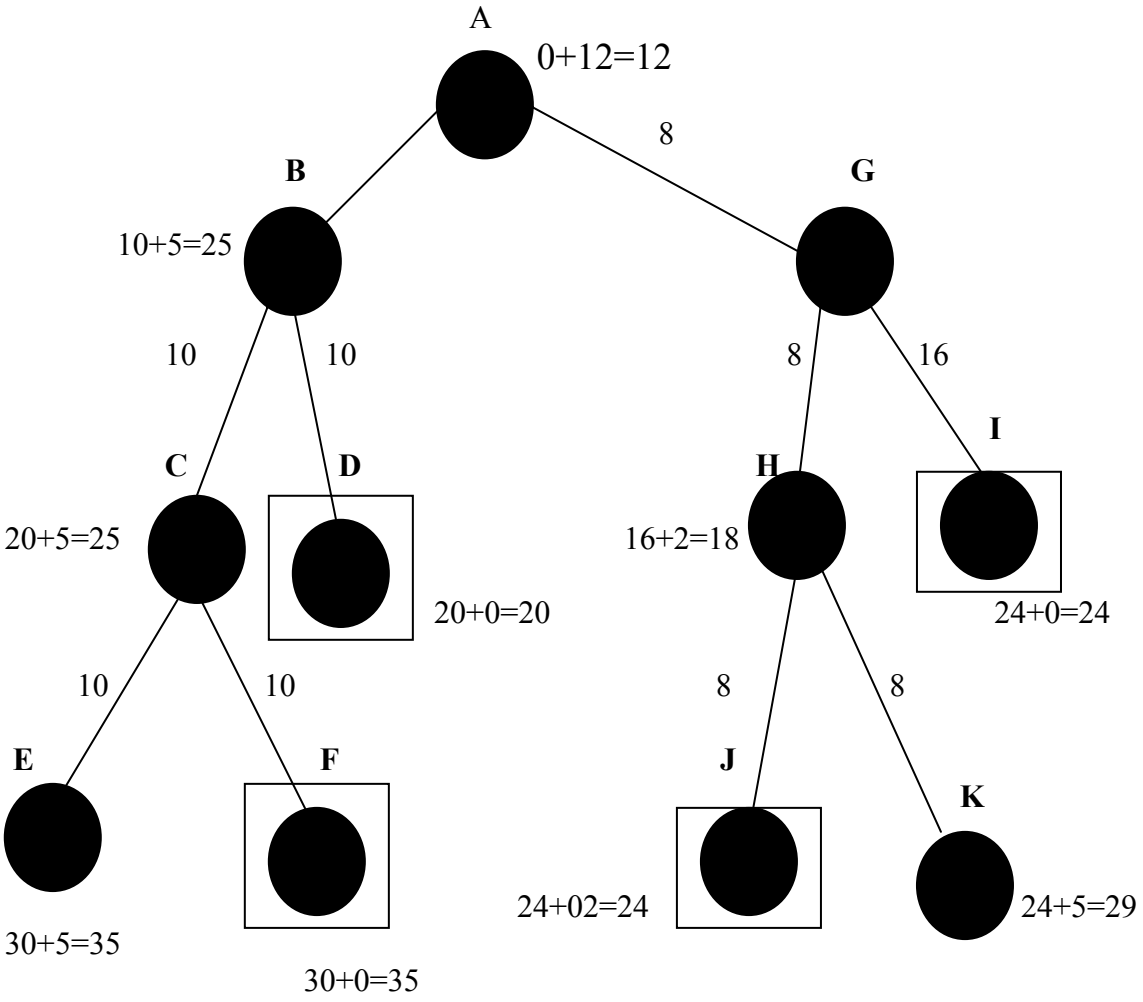
- Hará uso de toda la memoria que pueda disponer
- En la medida que se lo facilite la memoria, evitara los estados repetidos
- Es complejo si la memoria disponible tienen capacidad suficiente para guardar la ruta de solución más cercana.
- Es óptima si dispone de suficiente memoria para guardar la ruta de solución óptima más cercana. de lo contrario, produce la mejor solución que sea posible obtener con la memoria disponible.
- Si se dispone de suficiente memoria para todo el árbol de búsqueda, esta resultara óptimamente eficiente.

El diseño de A*SRM es simple, al menos globalmente. Si tiene generar un sucesor, pero ya no hay memoria, es necesario que se haga de espacio en la lista de espera. Para ello, excluir un nodo de la lista de espera; a estos nodos excluidos se les denomina nodos olvidados. Prefiere así excluir nodos que no tienen futuro, es decir, con un elevado costo f. Para evitar el regreso a explorar subárboles excluidos de la memoria, conserva la información de los nodos ancestros sobre la calidad de la mejor ruta en el subárbol olvidado. Así, solo tendrá que regenar el subárbol si las demás rutas ofrecen peores perspectivas que la ruta que se a olvidado. Otra forma de decir lo anterior es que si se

olvidan todos los descendientes de un nodo n, aunque ignoremos como ir desde n, tendremos una idea de que tanto vale la pana ir a algún sitio a partir de n.

EJEMPLO A*SRM

En la parte superior de la figura esta el espacio de búsqueda. Cada nodo se identifica mediante valores g+h=f; los nodos metas (D, F, I, J ) aparecen en cuadros. El objetivo es encontrar el nodo meta de menor costo, contando con memoria suficiente solo para tres nodos. Las etapas de la búsqueda se muestran en orden de izquierda a derecha; cada etapa esta numerada de acuerdo con la siguiente explicación. Cada uno de los nodos se identifica mediante su costo f actual, continuamente mantenido para que refleje el mínimo costo f de todos sus descendientes. Los valores entre paréntesis muestran el valor del mejor descendiente olvidado. El algoritmo procede de la siguiente manera:



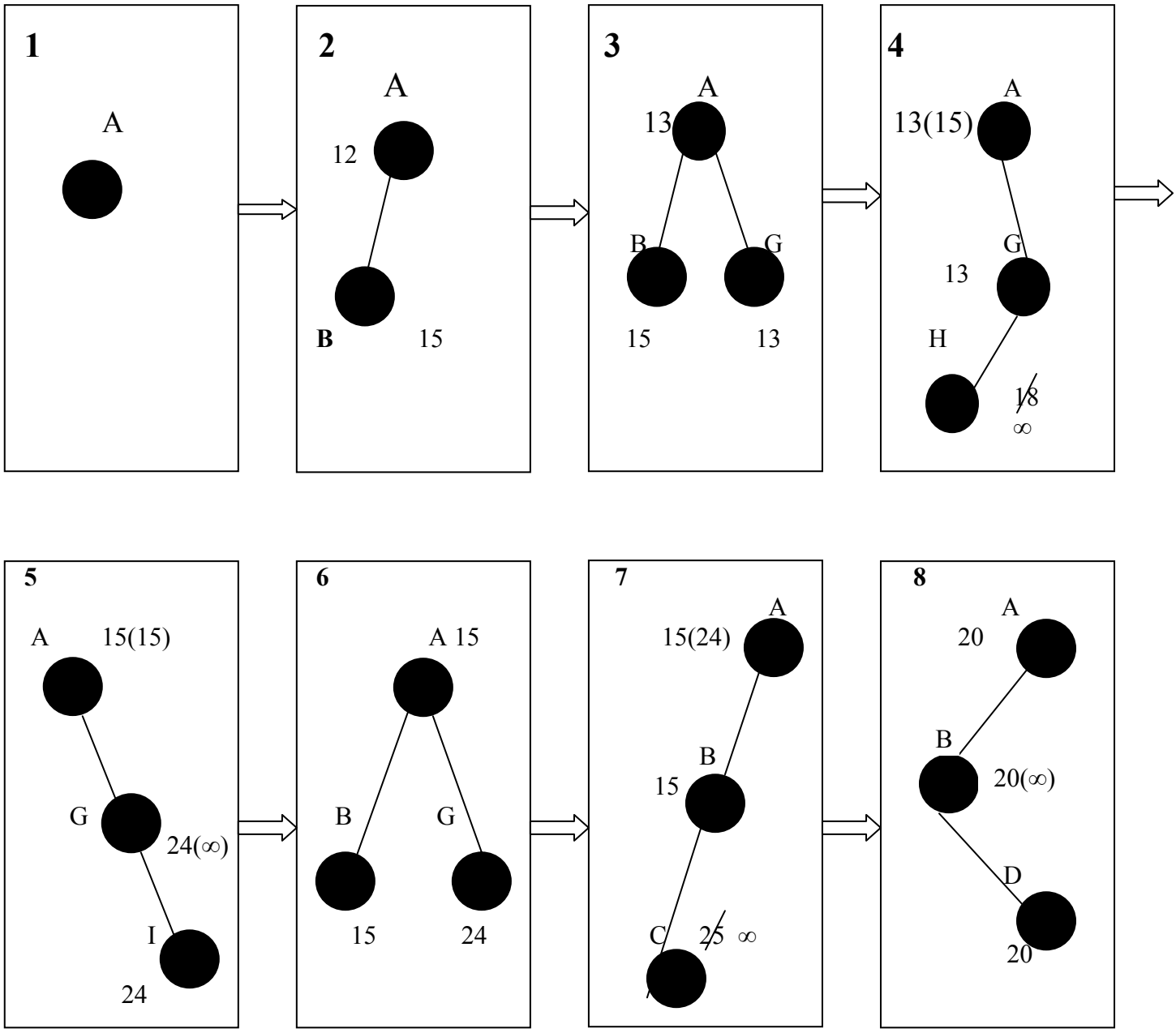



Figura 4.11 Desarrollo de una búsqueda A*SRM, cuya dimensión de memoria es tres nodos, en el espacio de estados mostrado en la parte superior. Cada nodo esta identificado por su costo-actual f. las cifras entre paréntesis muestran el valor del mejor descendiente olvidado.

1. En cada una de las etapas se añade un sucesor nodo de menor costo f más profundo cuyos sucesores no estén actualmente en el árbol. El hijo izquierdo b se añade a la raíz A.
2. Ahora, f(a) es todavía 12, por lo que se añade el hijo que esta a la derecha, G(f=13). Ahora que ya se vieron todos los hijos de A, es posible actualizar su costo f al mínimo de sus hijos, es decir, 13. la memoria ya se lleno.
3. G se escoge para efectuar una expansión, pero primero hay que excluir un nodo para hacer espacio. Se excluye la hoja de mayo costo f mas próximo, es decir, B. hecho lo anterior, notaremos que el mejor descendiente olvidado de A tiene f=15, como se muestra entre paréntesis. Se procede a añadir H, cuya f(H)=18. Desafortunadamente H no es un buen nodo meta, y la ruta a H consume toda la memoria disponible. Es decir, es imposible encontrar una solución a través de H, por lo que se define que f(H)=
4. Se expande de nuevo G. se excluye h, y se incluye I, con f(I)=24. se han visto ya los dos sucesores de G, con valores de y 24, por lo que f(G) se convierte en 24. f(A) se convierte en 15, el mínimo de 15 (valor del sucesor olvidado) y 24. nótese que aunque I es un nodo meta, quizás no sea la mejor solución debido a que el costo f de A es solamente 15.
5. Otra vez, A es el nodo mas prometedor, por lo que se genera por segunda vez B. Nos hemos dado cuenta que, después de todo, la ruta a través de G no resulto ser tan buena.
6. C, el primer sucesor de B, es un nodo no meta que esta a máxima profundidad, por lo que f(C)=
7. Para examinar el segundo sucesor, D, primero hay que excluir C . entonces, f(D)=20, valor que heredan tanto B como A.
8. Ahora el nodo de menor costo f mas profundo es D, por lo que se le escoge. Puesto que se trata de un nodo meta, concluye la búsqueda.

En este caso la memoria fue eficiente para la ruta de soluciones optima mas cercana.

En la figura 2. Se presenta un bosquejo general de A*SRM. En un programa real, hay que incorporar algunos cambios para enfrentar el hecho d que algunos nodos a veces van a parar con algunos sucesores en la memoria y otros olvidados. Las cosas se complican aun mas cuando hay que incluir en la verificación nodos repetidos. A*SRM es el algoritmo de búsqueda mas complicada que hemos visto hasta ahora.

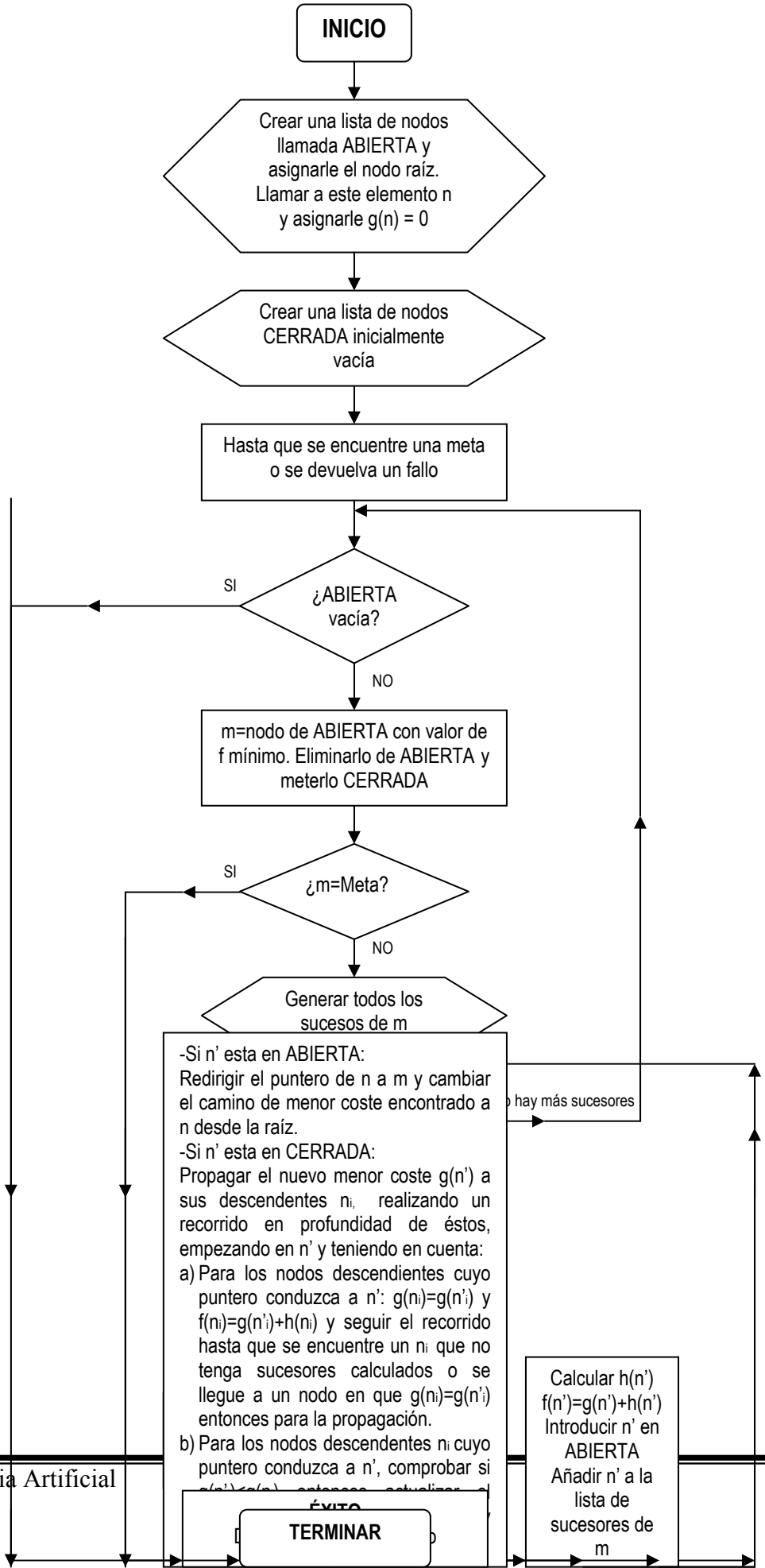
**function** SMA*(problem) **returns** a solution sequence
**funcion** A*SRM(problema) **responde con** una secuencia de solución.
**Entradas**: problema, un problema
**Estático**: Lista de espera, una lista de nodos organizada según costo-f

Lista de espera ← HACER-LISTA DE ESPERA({HACER NODO(ESTADO-INICIAL[problema]})
**bucle hacer**
**si** Lista de espera esta vacía, **entonces responde con** falla
n ← nodo mas profundo con f de mínimo costo en lista de espera
**si** PRUEBA-META(n) **entonces responde con** éxito
s ← SIGUIENTE-SUCESOR(n)
**si** s no es una meta y esta en la profundidad máxima, **entonces** f(s) ← ∞
o **bien**

f(s)←MAX (f(n), g(s)+ h(s))
**si** ya se genero a todos los n sucesores **entonces** actualice los n costos-fy, de ser necesario, los de los ancestros
**si** los SUCESORES(n) están todos en la memoria, **entonces** quite n de Lista de espera
**si** la memoria esta llena, **entonces**
borre en Lista de espera el nodo-de costo-f-mas-elevado que este mas próximo,
quítelo de la lista de sucesores de su padre
inserte su padre en Lista de espera, en caso de ser necesario
inserte s en Lista de espera
**fin**

Figura 2 Bosquejo del algoritmo A*SRM. Notese que, para mayor claridad, se han omitido numerosos detalles.

# Diagrama del Algoritmo A*



FALLO

## Bibliografía

http://www.yotor.net/wiki/es/al/algoritmo%20de%20la%20b%FAsqueda%20de%20la%20Unoestrella.htm

http://edevi.zonared.com/esp/nums/5/article.php?f=pf.php