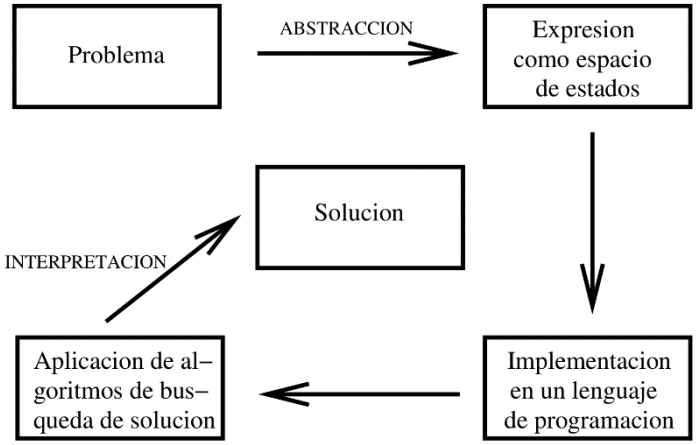
Inteligencia Artificial

Tema 3 – Planificación

1. Representación de problemas como espacios de estados



Los estados son descripciones de una posible situación en el problema. Una buena representación puede acelerar la resolución del problema mientras que una mala representación podría hacer el problema irresoluble.

Las acciones son el conjunto finito de operadores que transforman un estado en otro. Se describen mediante su aplicabilidad: Precondición y Postcondición. Las acciones dependerán de la representación de los estados, dando prioridad a las representaciones que menor nº de acciones tengan. Las acciones dan lugar a resultados (estados nuevos).

Llamamos estado inicial al estado que describe la situación de inicio/partida y estado final (o estados finales) a los conjuntos de estados que describen el objetivo.

Espacio de estados como un grafo: un espacio de estados puede visualizarse como un grado dirigido (digrafo) donde los vértices son los estados.

Ejemplo:

- Problema del granjero: un granjero se encuentra a la orilla del rio con un lobo, una cabra y una lechuga (lo típico). Su intención es pasar a la otra orilla y dispone de una barca en la que solo puede ir él junto con una sola de las cosas anteriores. El lobo cabrón se come a la cabra si no está el granjero y la cabra se come la lechuga si no está el granjero.
- Información de los estados: orilla en la que se encuentra cada elemento (granjero, lobo, cabra, lechuga). (La orilla donde esté la barca es irrelevante porque siempre será donde esté el granjero).
- Representación de los estados (formal): (x, y, z, u) (cada elemento) en {i, d}^4 → 16 estados
- Estado inicial: (i, i, i, i) (la i es orilla izquierda y la d, derecha)
- Estado final (único): (d, d, d, d)
- Acciones:
    - Pasa el granjero solo

  - Pasa el granjero con el lobo
  - Pasa el granjero con la cabra
  - Pasa el granjero con la lechuga
- Aplicabilidad de las acciones:
  - Precondición: los dos elementos que pasan en la barca han de pertenecer a la misma orilla.
  - Postcondición: al final, no deben estar el lobo, la cabra y la lechuga en una orilla sin el granjero.
- Estado resultante de aplicar la acción: cambia de orilla al elemento que sube en la barca.

2. Técnicas básicas de búsqueda en espacios de estados

Objetivo: encontrar una secuencia de acciones que, partiendo del estado inicial, obtenga un estado final.

Idea básica: explorar el grafo que representan el espacio de estados:

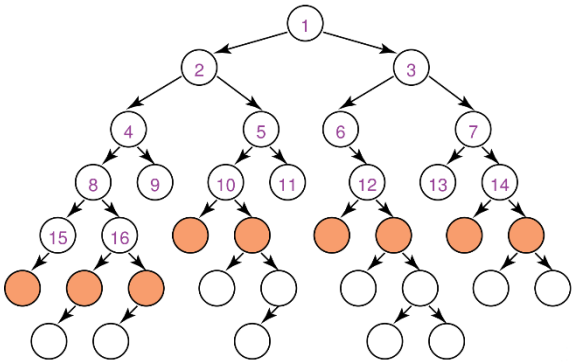
a. En cada momento se analiza un estado actual y se mantiene una "frontera de exploración" llamada "lista de abiertos".
b. Si el estado actual es el estado final, se devuelve la sucesión de acciones que nos ha llevado a él.
c. En caso contrario, obtiene los sucesores del estado actual y se añaden a la lista de abiertos.
d. A continuación, se elige un nuevo estado actual.
e. Repetimos el proceso mientras haya estados en la frontera de exploración.

La implementación de un problema como espacio de estados consta de:

a. Una variable estado_inicial que almacena la representación del estado de partida.
b. Una función es_estado_final(estado) que comprueba si un estado es final o no.
c. Una función acciones(estado) que devuelve las acciones aplicables a un estado dado.
d. Una función aplica(accion, estado) que obtiene el estado resultante de aplicar una acción a un estado (suponemos que dicho estado permite esa acción).

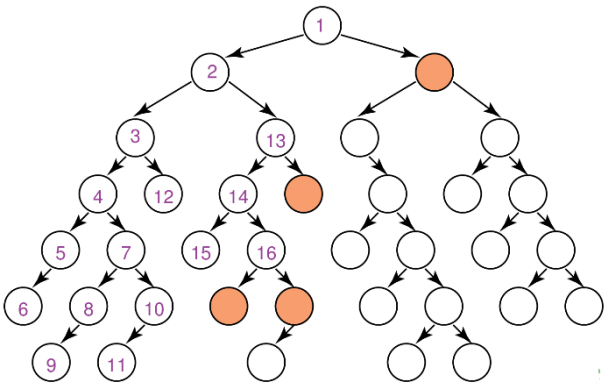
Ejemplo: Búsqueda en anchura (BFS): se gestiona la "memoria" como una FIFO:

```
FUNCION BUSQUEDA-EN-ANCHURA()
1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo
    cuyo estado es *ESTADO-INICIAL*)
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS.
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
       2.4.1 devolver el nodo ACTUAL y terminar.
       2.4.2 en caso contrario,
             2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos
                     de SUCESORES(ACTUAL) cuyo estado no está
                     ni en ABIERTOS ni en CERRADOS
             2.4.2.2 Hacer ABIERTOS el resultado de incluir
                     NUEVOS-SUCESORES al final de ABIERTOS
3. Devolver FALLO.
```



Ejemplo: Búsqueda en profundidad (DFS): se gestiona la "memoria" como una LIFO:

```
FUNCION BUSQUEDA-EN-PROFUNDIDAD()
1. Hacer ABIERTOS la pila formada por el nodo inicial (el nodo
    cuyo estado es *ESTADO-INICIAL*)
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS.
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
       2.4.1 devolver el nodo ACTUAL y terminar.
       2.4.2 en caso contrario,
             2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos
                     de SUCESORES(ACTUAL) cuyo estado no está
                     ni en ABIERTOS ni en CERRADOS
             2.4.2.2 Hacer ABIERTOS el resultado de incluir
                     NUEVOS-SUCESORES al principio de ABIERTOS
3. Devolver FALLO.
```



Ejemplo: Búsqueda en profundidad acotada (no exploramos caminos más allá de una determinada longitud):

```
FUNCION BUSQUEDA-EN-PROFUNDIDAD-ACOTADA(COTA)
1. Hacer ABIERTOS la pila formada por el nodo inicial (el nodo
        cuyo estado es *ESTADO-INICIAL*);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS.
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
       2.4.1 devolver el nodo ACTUAL y terminar.
       2.4.2 en caso contrario, si PROFUNDIDAD(ACTUAL)
             es menor que la cota,
             2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos
                     de SUCESORES(ACTUAL) cuyo estado no está
                     ni en ABIERTOS ni en CERRADOS
             2.4.2.2 Hacer ABIERTOS el resultado de incluir
                     NUEVOS-SUCESORES al principio de ABIERTOS
3. Devolver FALLO.
```

Ejemplo: modificación de la búsqueda en profundidad acotada para que, cuando no se conoce una cuota, evitar la incompletitud en las búsquedas; la llamamos búsqueda en profundidad iterativa:

```
FUNCION BUSQUEDA-EN-PROFUNDIDAD-ITERATIVA(COTA-INICIAL)
1. Hacer N=COTA-INICIAL
2. Si BUSQUEDA-EN-PROFUNDIDAD-ACOTADA(N) no devuelve fallo,
      2.1 devolver su resultado y terminar.
      2.2 en caso contrario, hacer N igual a N+1 y volver a 2
```

3. Búsquedas informadas

Las tácticas anteriores se conocen como búsquedas ciegas o búsquedas no informadas en tanto que no cuentan con ningún conocimiento previo sobre como llegar al objetivo. La búsqueda informada aplica conocimientos previos al proceso de búsqueda a fin de hacerlo más eficiente → usa heurísticas. Este conocimiento decimos que viene dado por una función de "comodidad" o "bondad" de los estados.

Como hemos dicho, la función heurística se refleja en una función que cuantifica como de "cerca" estamos de un estado final. TODO el conocimiento específico que se vaya a usar sobre el problema está codificado en la función heurística.

Ejemplo: Búsqueda del primer mejor o búsqueda voraz o codiciosa: analiza preferiblemente los nodos con heurística más baja. Su rendimiento varía en función de la "bondad" de la heurística.

```
FUNCION BUSQUEDA-POR-PRIMERO-EL-MEJOR()
1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo
    cuyo estado es *ESTADO-INICIAL*);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS.
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
       2.4.1 devolver el nodo ACTUAL y terminar.
       2.4.2 en caso contrario,
             2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos
                     de SUCESORES(ACTUAL) cuyo estado no está
                     ni en ABIERTOS ni en CERRADOS
             2.4.2.2 Hacer ABIERTOS el resultado de incluir
                     NUEVOS-SUCESORES en ABIERTOS y ordenar
                     en orden creciente de las heurísticas
                     de los estados de los nodos
3. Devolver FALLO.
```

4. Planificación en IA

Definimos planificar como encontrar una secuencia de acciones que alcanzan un determinado objetivo si se ejecutan desde un determinado estado inicial. El "plan" es la secuencia de acciones. Para este tema suponemos:

a. Un nº finito de estados observables.
b. Estados deterministas.
c. Tiempo implícito (las acciones actúan sin duración).
d. La planificación se realiza a priori.

5. Formalismo lógico: el lenguaje PDDL

Emplearemos un lenguaje formal para representar problemas de planificación formado por:

a. Constantes: objetos del "mundo"
b. Variables que representan objetos genéricos
c. Símbolos de predicados
d. Símbolos que representan acciones

Terminología:

a. Átomos: fórmulas del tipo P(o1,…,on) donde P es un símbolo de predicado y cada oj es una constante o variable.
b. Literales: átomos o negación de átomos.

Ejemplo: dado un "mundo" donde existe una mesa, cuatro cubos (un cubo puede estar sobre la mesa o apilado sobre otro cubo) y un brazo mecánico capaz de coger un cubo cada vez. Se describen los siguientes predicados:

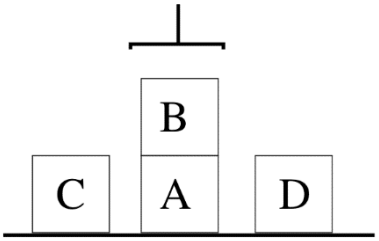
DESPEJADO(x), el bloque x está despejado.

BRAZOLIBRE(), el brazo no agarra ningún bloque.

SOBRELAMESA(x), el bloque x está sobre la mesa.

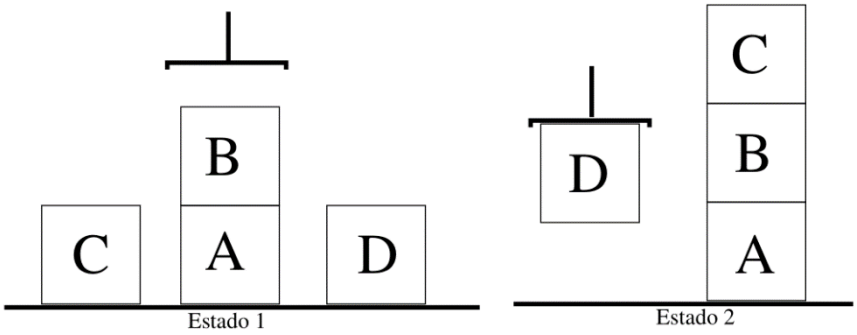
SOBRE(x,y), el bloque x está sobre el y.

AGARRADO(x), el bloque X está sujeto por el brazo.



Representación de objetivos: decimos que satisfacer un objetivo es encontrar un estado que contenga todos los literales positivos del objetivo instanciado y ninguno de los negativos. Los objetivos se representan como una conjunción (*y*) de literales.

Ejemplo: volviendo al "mundo" anterior de bloques: dados dos estados posibles de ese mundo:



Ejemplos

SOBRE(B,A), SOBRELAMESA(A), -SOBRE(C,B) es satisfecho por el estado 1 pero no por el estado 2

SOBRE(x,A), DESPEJADO(x), BRAZOLIBRE() es satisfecho por el estado 1 pero no por el estado 2

SOBRE(x,A), SOBRE(y,x) no es satisfecho por el estado 1 pero sí por el estado 2

El objetivo SOBRE(x,A), -SOBRE(C,x) es satisfecho por el estado 1 pero no por el estado 2

Acciones: se describen mediante:

a. Su nombre y todas las variables involucradas
b. Precondición: lista de literales que deben cumplirse para aplicar la acción
c. Efectos: lista de literales que indican los cambios que producirá cuando se aplique. Distinguimos entre:
    a. Efectos positivos: átomos que pasarán a ser ciertos.
    b. Efectos negativos: átomos que dejarán de ser ciertos.

Ejemplos de acciones en el mundo de bloques previamente visto:

APILAR(x,y)

Prec.: DESPEJADO(y), AGARRADO(x)

Efec.: -DESPEJADO(y), -AGARRADO(x), BRAZOLIBRE(), SOBRE(x,y), DESPEJADO(x)

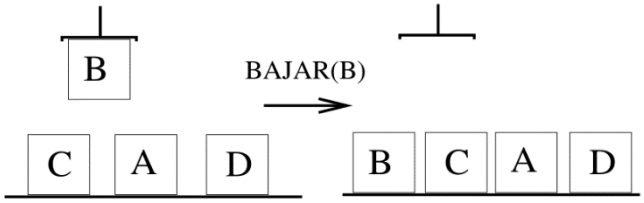
AGARRAR(x)

Prec.: DESPEJADO(x), SOBRELAMESA(x), BRAZOLIBRE()

Efec.: -DESPEJADO(x), -SOBRELAMESA(x), -BRAZOLIBRE(), AGARRADO(x)

Una acción solo es aplicable a un estado si este satisface su precondición. El resultado de aplicar una acción es un estado E resultante de:

a. Eliminar de E los átomos correspondientes a la lista de negativos.
b. Añadir a E los átomos correspondientes a la lista de positivos.



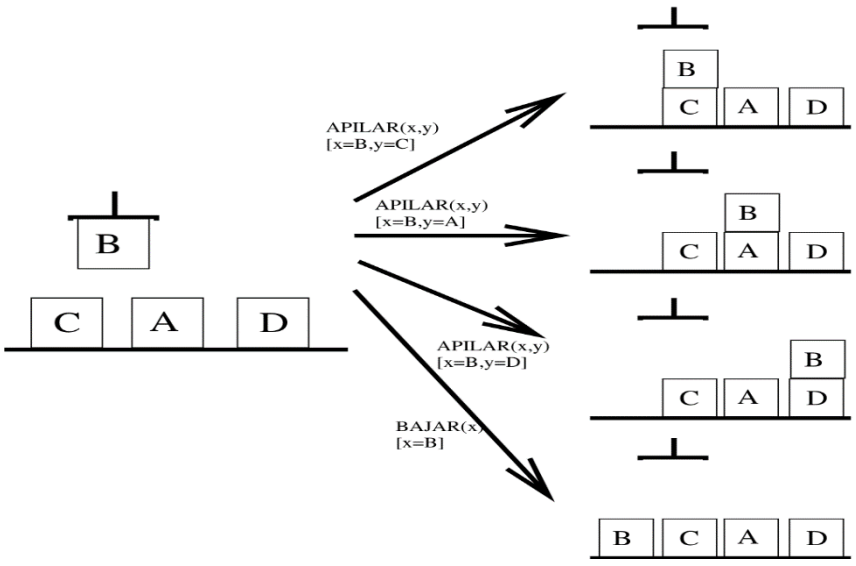

```
* Estado antes de aplicar BAJAR(B):
E = {DESPEJADO(C), DESPEJADO(A), DESPEJADO(D), SOBRELAMESA(C),
     SOBRELAMESA(A), SOBRELAMESA(D), AGARRADO(B)}

* Precondiciones de BAJAR(B):
Prec = {AGARRADO(B)}
------ Condiciones satisfechas en el estado -------
------           (acción aplicable)          -------

* Efectos de BAJAR(B):
Efec- = {AGARRADO(B)}
Efec+ = {SOBRELAMESA(B), BRAZOLIBRE(), DESPEJADO(B)}


* Estado después de aplicar BAJAR(B):
E' = (E - Efec-) U Efec+ =
{DESPEJADO(C), DESPEJADO(A), DESPEJADO(D), SOBRELAMESA(C),
 SOBRELAMESA(A), SOBRELAMESA(D),SOBRELAMESA(B), BRAZOLIBRE(),
 DESPEJADO(B)}
```

Un problema de planificación puede plantearse como un problema de espacio de estados:



6. Búsqueda hacia delante en profundidad con heurística

```
FUNCION BUSQUEDA-EN-PROFUNDIDAD-H(ESTADO-INICIAL,OBJETIVO,ACCIONES)
 Devolver BEP-H-REC({},{},ESTADO-INICIAL,OBJETIVO,ACCIONES)

FUNCION BEP-H-REC(PLAN,VISITADOS,ACTUAL,OBJ,ACCIONES)
1. Si ACTUAL satisface OBJ, devolver PLAN
2. Hacer APLICABLES igual a la lista de acciones que sean instancias
   de una acción de ACCIONES, que sean aplicables a ACTUAL y cuya
   aplicación no resulte en un estado de VISITADOS
3. Hacer ORD-APLICABLES igual a ORDENA-POR-HEURISTICA(APLICABLES)
4. Para cada ACCION en ORD-APLICABLES
   4.1 Hacer E' el resultado de aplicar ACCION a ACTUAL
   4.2 Hacer RES igual a
         BEP-H-REC(PLAN·ACCION,VISITADOS U {E'},E',OBJ,ACCIONES)
   4.3 Si RES no es FALLO, devolver RES y terminar
5. Devolver FALLO
```

- Es un algoritmo de tipo "backtracking" en el que se ordenan por heurística los sucesores del estado actual. Es un algoritmo que siempre termina (no significa que siempre proporcione la solución más corta). La complejidad en tiempo es exponencial y en el espacio, es lineal.
- Si la heurística (de la cual depende en realidad) no es buena, la búsqueda hacia adelante tiene problemas de excesiva ramificación.
- ¿Alternativa a la búsqueda hacia adelante?

7. Búsqueda hacia atrás

- Solo aplicamos hacia atrás las acciones relevantes para el objetivo.
- Se empieza por el objetivo y se aplican acciones en sentido inverso para tratar de llegar al estado inicial.

```
FUNCION BUSQUEDA-HACIA-ATRÁS-H(ESTADO-INICIAL,OBJ,ACCIONES)
  Devolver BHA-H-REC({},{},ESTADO-INICIAL,OBJ,ACCIONES)

FUNCION BHA-H-REC(PLAN,VISITADOS,ESTADO-INICIAL,G-ACTUAL,ACCIONES)
1. Si ESTADO-INICIAL satisface G-ACTUAL, devolver PLAN
2. Hacer RELEVANTES igual a la lista de acciones que sean instancias
    de una acción de ACCIONES, que sean relevantes para G-ACTUAL
    y tal que el predecesor de G-ACTUAL respecto de la acción
    no sea un objetivo que contiene a alguno de VISITADOS
3. Hacer RELEVANTES-ORDENADOS
     igual a ORDENA-POR-HEURISTICA(RELEVANTES)
4. Para cada ACCION en RELEVANTES-ORDENADOS
   4.1 Hacer G' el objetivo predecesor de G-ACTUAL respecto a ACCION
   4.2 Hacer RES igual a
        BHA-H-REC(ACCION·PLAN,VISITADOS U {G'},
                  ESTADO-INICIAL,G',ACCIONES)
   4.4 Si RES no es FALLO, devolver RES y terminar
5. Devolver FALLO
```

Concepto de “Acción relevante para un objetivo”:

- Sea G un objetivo decimos que una acción A es relevante para G si:
  - Al menos, uno de los efectos (positivo o negativo) de la acción está en G con el mismo signo.
  - Ninguno de los efectos (positivo o negativo) de la acción aparece en G con distinto signo.

Cálculo de predecesores: si A es una acción relevante para un objetivo G, entonces el objetivo predecesor de G respecto a A es: (G − efectos(A)) ∪ precond(A).

G = {DESPEJADO(A), DESPEJADO(B), DESPEJADO(D), SOBRE(B,C)}

Predecesor respecto de la acción relevante APILAR(B,C)

G' = (G − {−DESPEJADO(C), −AGARRADO(B), BRAZOLIBRE(),
SOBRE(B,C), DESPEJADO(B)})
U {DESPEJADO(C), AGARRADO(B)}
= {DESPEJADO(A), DESPEJADO(D), DESPEJADO(C), AGARRADO(B)}

8. Heurísticas para planificación basada en espacios de estados

Como hemos visto la heurística es el componente fundamental para la correcta eficiencia de los algoritmos anteriores. Esta heurística debe estimar la distancia entre estados y objetivos (el nº de acciones necesarios). Si esta estimación está por debajo del mínimo nº de acciones real, diremos que la heurística es admisible.

Heurísticas basadas en relajación: aquellas que se obtienen “relajando” las restricciones del problema y calculando el nº de acciones necesarias para ese problema “relajado” (menos restrictivo). Algunas ideas para “relajar” un problema:

a. Ignorar las precondiciones y/o efectos negativos de las acciones.
b. Suponer que cada literal de un objetivo se alcanza de manera independiente.
c. Suponer que el nº de acciones necesarias para alcanzar un objetivo es el nº de pasos máximo necesario para alcanzar uno de sus literales.
d. Ignorar todas las precondiciones de las acciones.
e. Ignorar determinados predicados.
f. etc…

La heurística Δ_0: basada en las dos primeras ideas anteriores: da una estimación del nº de acciones a realizar para llegar al objetivo:

- Dado un estado e, un átomo p y un objetivo g que solo tiene literales positivos sin variables, definimos recursivamente: Δ_0 (e,p) y Δ_0 (e,g) tal que:
  - Si p aparece en e, Δ_0(e,p) = 0.
  - Si p no aparece en e ni en los efectos positivos de ninguna acción, Δ_0(e,p) = $+\infty$ (es imposible).
  - En otro caso, Δ_0(e,p) = min{1 + sumatorio((cuando q cumple todas las precondiciones de A) de Δ_0(e,q) | p ∈ (efectos positivos de A)}.
  - Δ_0(e,g) = sumatorio((cuando p ∈ g) de Δ_0(e,p).

- $\Delta_0(e,p)$ cuenta el nº menor de pasos necesarios para que se verifique p a partir de e, suponiendo que las accione son tienen precondiciones negativas ni efectos negativos. $\Delta_0(e,g)$ es la suma de todas las estimaciones para cada átomo $p \in g$.
- Con carácter general, Δ_0 no es admisible. Suele estar por encima de la realidad.
- Δ_0 en la búsqueda hacia adelante: a cada acción A aplicable al estado actual se le asigna un valor heurístico $\Delta_{0\,(e,g+)}$ donde g+ es el conjunto de literales positivos del objetivo, y e es el estado que resulta de aplicar la acción A.
- Δ_0 en la búsqueda hacia atrás: a cada acción A relevante respecto del objetivo actual se le asigna un valor $\Delta_{0(e0,g+)}$ donde e_0 es el estado inicial, g es el predecesor a la acción A y g+ es el conjunto de literales positivos de g.