Tema-1-Sistemas-Inteligentes-y-B...

ParmigianoReg

Tecnologias de los Sistemas Inteligentes

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Tema 1 - Sistemas Inteligentes y Búsqueda

Profesor Antonio González Muñoz, Curso 20-21

Sistemas Inteligentes

- Un sistema inteligente (SI) es un sistema software que muestra un comportamiento inteligente o interactúa de forma más inteligente con su entorno y con otros sistemas.
- La barrera entre un software normal y un sistema inteligente queda un tanto difusa, al igual que la barrera entre los Sistemas Inteligentes y la Inteligencia Artificial.
- La Inteligencia Artificial se ocupa de la investigación básica en la implementación de cada una de las habilidades relacionadas con la inteligencia humana.
- Los Sistemas Inteligentes se encargan de aplicar la investigación, tratando de solucionar problemas ya existentes o mejorar nuestra forma de trabajar o calidad de vida aplicando técnicas de IA.

Problemas en IA

- Los humanos normalmente somos más eficientes que las máquinas para resolver problemas NP-Duros.
  - "Si un problema en IA tiene solución es que se ha escogido mal el problema"

Construcción de Sistemas Inteligentes

- La Inteligencia Artificial es una rama de la informática que estudia y resuelve problemas situados en la frontera de la misma.
- Se basa en dos ideas fundamentales:
  - Representación del conocimiento explícito y declarativo.
  - Resolución de problemas por heurísticas.

Búsqueda Heurística y Propiedades:

Heurísticas y tipos de problemas

- Las heurísticas son criterios, métodos o principios para decidir cuál de entre varias acciones promete ser la mejor para alcanzar una determinada meta.

- **Problema de las 8 Reinas**
  - *Colocar 8 reinas en un tablero de ajedrez sin que ninguna pueda capturar a otra.*
  - Se puede tener una heurística que cuente el número de casillas libres luego de colocar una reina y otra que obtenga el mínimo de casillas libres por cada fila; esta última es una heurística mucho mejor.
  - Cuanto más general sea una heurística a un tipo de problemas, mejor porque se puede adaptar a múltiples problemas.
  - Este es un problema de satisfacción de restricciones.
- **Problema del 8-Puzzle**
  - *Se tiene una configuración inicial de casillas y se desean mover hasta que estén en orden.*
  - La heurística puede ser el número de casillas mal colocadas; aunque una mejor es la que tiene la suma Manhattan del camino de la casilla del inicio al final ya que de esa manera se estiman mejor los pasos que tomará el problema.
  - Problema de camino. (Igual que el mapa de carreteras y TSP).
- **Problema del Mapa de Carreteras**
  - *Dadas unas ciudades conectadas por carreteras, obtener el camino más corto entre una ciudad y otra.*
  - La heurística se parece a la de 8-puzzle, se tiene en cuenta lo que he recorrido más la estimación para llegar al final.
  - Se diferencia porque contempla el costo, porque es un problema de optimización, por eso se tiene que minimizar.
- **Problema del Viajante de Comercio**
  - *Dado un conjunto de ciudades conectadas, obtener el camino más corto para recorrer todas las ciudades y retornar a la inicial.*

- Hay muchas heurísticas complejas para este problema, se debe contemplar lo que se tiene recorrido y estimar el camino de vuelta al inicio. Como es un problema tan complejo no se tiene una heurística general que funcione sobre él, las heurísticas simples no funcionan bien.
- Una heurística es el grafo óptimo de grado 2 se utiliza ya que se parece al TSP, (orden n^3)
- Se puede utilizar el árbol generador minimal, (orden n^2)

- **Problema de la Moneda Falsa**
  - *Se tienen 12 monedas, se sabe que hay una falsa porque, por ejemplo, pesa menos que el resto. Existe un instrumento para pesarla: una balanza, se quiere detectar con el menor número posibles de evaluaciones cual es la falsa, en este caso el máximo es de 3 evaluaciones.*
  - Problema de tipo descomponible, no se desea realizar una simulación, se desea un esquema genérico.
  - El estado se representa con el número de monedas.
  - Una heurística para este problema puede ser la "complejidad" de pesar una x cantidad de monedas entre sí.

Búsqueda Primero El Mejor:

Búsqueda con el Algoritmo A*

- Método más conocido de búsqueda por el mejor nodo.
- La idea es evitar explorar caminos que ya sabemos que no son interesantes por su costo.
- La función de evaluación es $f(n) = g(n) + h(n) \rightarrow A*$
  - $g(n)$ es el costo actual.
  - $h(n)$ es el costo estimado del nodo al objetivo.
  - $f(n)$ el costo total estimado al objetivo obligado a pasar por el nodo.

- El algoritmo posee una lista de nodos ABIERTOS, donde al principio está el nodo inicial. Existe otra lista, CERRADOS, que está vacío.
- Comienza un bucle que se repite hasta encontrar solución o hasta que ABIERTOS está vacío.
  - Seleccionar el mejor nodo de ABIERTO.
  - Si es el nodo objetivo terminar.
  - En caso contrario, expandir ese nodo.
  - Para cada uno de sus sucesores:
    - Si está en ABIERTOS, insertarlo manteniendo la información del mejor padre.
    - Si está en CERRADOS, insertarlo manteniendo la información del mejor padre y actualizar la información de los descendientes.
    - En otro caso insertarlo como un nodo nuevo

Modelos más generales: Algoritmos de Agendas

- Consiste en una generalización del algoritmo A*, surge de manera totalmente independiente, siendo diseñado para un algoritmo llamado AM para teoría de números.
- Este algoritmo propone en vez de Estados, Tareas. El algoritmo realizaba una tarea.
  - Una tarea consiste en una descripción de la misma, una lista de justificaciones -- los padres del nodo-- y una valoración heurística.
  - Una vez que se realizaba una tarea (que sería como expandir un nodo), surgían otras tareas que se le proponían. Por lo tanto debe gestionar dichas tareas, de aquí surge la agenda.
  - La agenda es la lista de tareas, de donde se quiere seleccionar la tarea más importante. Esto es un equivalente a la heurística de decidir cual nodo es el mejor, por lo tanto la agenda es un equivalente al conjunto de ABIERTOS.
  - El conjunto de CERRADOS no existe pero puede pensarse como las tareas ya realizadas.
  - El algoritmo toma una tarea de la agenda, luego de realizarla obtiene otra tarea, la cual puede que ya existe en la agenda pero fue propuesta por un padre diferente o una justificación distinta, si es así añade la nueva justificación a la tarea y continúa.
- Formalmente, este algoritmo y el de A* son muy similares; pero que no está ligado a encontrar un camino mínimo, se puede pensar que el A* sería una particularización del Algoritmo de Agendas.
  - Esto quiere decir que si una tarea es generada por padres diferentes, en el Algoritmo de Agendas lo

yo elijo cerveza SIN

Sea cual sea el vehículo que conduces, elige cerveza SIN.

WWW.CONDUCCIONRESPONSABLECERVEZASIN.COM

Cerveceros de España 100 años más

UNA GRAN CERVEZA.
UNA GRAN RESPONSABILIDAD.

DGT Dirección General de Tráfico · ADA · AEDIVE · AEESCAM · aesleme · aevecar · AOP · Asociación Española de la Carretera · aspaym · CEA · cnae · FESVIAL · HOSTELERÍA DE ESPAÑA · PMSV · PONS FUNDACIÓN · Rotary Distrito 2201

que se hace es que se añade a las justificaciones, de que es posible por otra tarea llegar a esa, es un algoritmo más general. En el A*, en cambio, se tiene un operador que discrimina que si un padre es mejor que otro, se selecciona que el nodo se quede con ese padre.

Modelos con Poda

- Una de las primeras maneras en la que se extendió el algoritmo A* original es que se le añadieron criterios de poda.
    - **Búsqueda Dirigida (Poda a priori)**
        - La diferencia es que restringe de qué manera se generan los descendientes, originalmente como este algoritmo guarda todos los nodos hijos ya sea en ABIERTOS o CERRADOS, si se tiene un problema en el que el factor de ramificación es muy elevado, entonces el espacio de búsqueda crece mucho.
        - Se restringe artificialmente este factor de ramificación por medio de la heurística h que utiliza, se generan los descendientes y se limita el número del mismo, el resto de los descendientes al ser podados, esto puede comprometer encontrar la solución.
    - **Algoritmo A* con Memoria Acotada (Poda a posteriori)**
        - Se delimita la estructura de datos, por ejemplo ABIERTOS puede tener solamente un tamaño determinado fijo, una vez que se llena el espacio nodos generados luego deben comparar su función $f(n)$ y si son mejores entran eliminando los peores nodos de ABIERTOS.
        - Es posteriori porque se genera el nodo y luego se ve si se poda luego de compararlo con el resto de nodos ya generados.
        - Es más ágil que el A*: se puede sacrificar obtener la solución óptima pero funciona más rápido.

Alternativas al A*

- Modifican de manera más extensiva el A* original
- Tienen la misma optimalidad que el A*, pero en ciertos problemas funcionan mejor que el propio A*.
    - **Descenso Iterativo A* (IDA*)**
        - Algoritmo popular y frecuentemente utilizado, se implementó por el autor Korf en 1985 para afrontar problemas que el A* original no podía, por ejemplo el 8-Puzzle de valores elevados
        - La función heurística se mantiene de igual, $f(n) = g(n) + h(n)$, lo que varía es la implementación. Se utiliza una estrategia retroactiva branch and bound, que va a iterar no en profundidad sino en costo de la solución.
        - Se establece un modelo de cotas, se establece una cota inicial y en las siguientes iteraciones se empieza a variar estas cotas.
            - Una vez que inicia el algoritmo, el nodo raíz es comparado con la cota, como será igual a la cota podrá generar hijos ya que la cota es siempre el h del nodo raíz. Genera un hijo, si el nodo hijo no supera la cota, el algoritmo continúa profundizando por ese nodo recursivamente. Si se llega a un nodo que supera la cota, se para la recursión. Se regresa al nodo anterior y genera otro hijo por el que continúa explorando, todo esto hasta que regresa al nodo raíz y este genera otro hijo.
            - El algoritmo no se detiene si encuentra el objetivo en un nodo que está por encima de la cota porque no está garantizado que sea la manera óptima de llegar a él, se detendrá si consigue el objetivo y este tiene un coste menor que la cota actual.
            - Si el algoritmo no ha encontrado la solución para una cota η, entonces se incrementa la cota. La cota es calculada como el valor mínimo del coste f de los nodos que han superado la cota anterior.
            - Se repite el proceso utilizando la cota nueva hasta encontrar la solución.

Procedimiento IDA* (Estado-inicial Estado-final)

EXITO=Falso
$\eta = h(s)$
Mientras que EXITO=Falso

Procedimiento IDA* (Estado-inicial Estado-final)

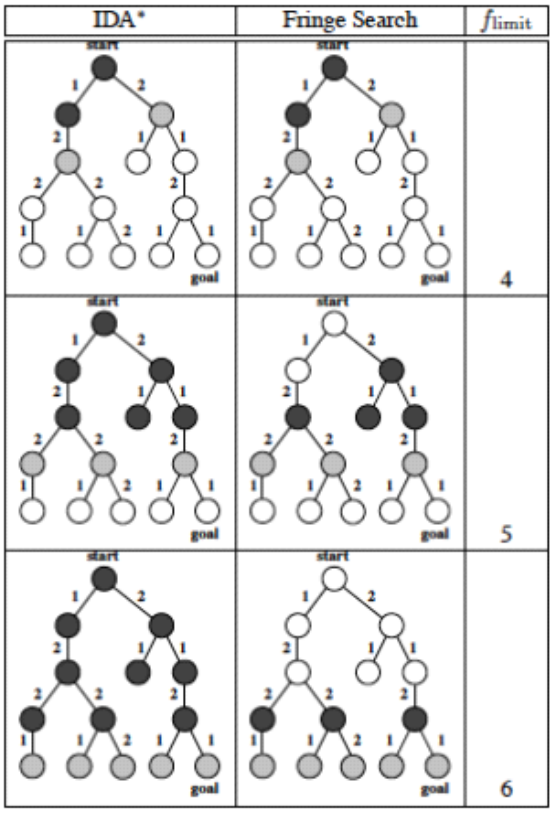
EXITO=Falso
$\eta = h(s)$
Mientras que EXITO=Falso
    EXITO=Profundidad (Estado-inicial,η)
    $\eta = \min_{i=1,n}\{f(i)\} = \min_{i=1,n}\{g(i) + h(i)\}$
Solución=camino desde nodo del Estado-inicial
        al Estado-final por los punteros

Profundidad (Estado-inicial,η)
    Expande todos los nodos cuyo coste $f(n)$ no excede η

- ¿Por qué se utiliza la cota de esa manera? Es para mantener la condición que $h(n) \leq h^*(n) \forall n$, la admisibilidad. La cota se amplía lo mínimo posible para poder buscar la solución pero estando siempre por debajo de $h^*(n)$.
- Notar que en cada iteración se repiten nodos, ya que al volver a la raíz se empiezan a generar todos los nodos explorados de nuevo, es una desventaja pero también una ventaja es que se almacenan muy pocos nodos en memoria.
- **Características:**
  - **Completitud**: Es completo, bajo las mismas condiciones que el A*, es decir en ciertos tipos de grafos y con coste positivo.
  - **Admisibilidad**: Es admisible, solo cuando $h \leq h^*$ se encuentra la solución óptima
  - **Eficiencia**: Complejidad en tiempo exponencial, pero complejidad en espacio lineal en la profundidad del espacio de búsqueda. Aunque pareciera ser lo contrario, el número de reexpansiones es solo mayor en un pequeño factor que el número de expansiones de los algoritmos Primero El Mejor
  - Primer algoritmo en resolver óptimamente 100 casos generados aleatoriamente del 15-Puzzle, pero no es mejor como tal que el A^*, pero si puede funcionar mejor en ciertos casos.

- **Búsqueda Primero El Mejor Recursivo**
  - Almacena más nodos que el IDA^* pero muchos menos que el A^* y también, se diseñó para evitar reexpandir nodos, pero si es posible que pueda hacerlo.
  - La función heurística $f(n) = g(n) + h(n)$ se mantiene, la estructura es similar al resto de los algoritmos.
  - La idea es que en cada nivel del árbol de exploración, se almacenan además de los nodos que se han explorado, también los nodos no explorados de ese mismo nivel. Se tiene un conjunto de ABIERTOS. Solo se admite mantener en memoria los nodos de un camino que se está explorando y los nodos hermanos de aquellos que están en el camino, no se admite expandir esos nodos hermanos en ese camino. Si sucede que un nodo hermano tiene un mejor f que el nodo expandido actualmente, primero, se almacena el valor f en el nodo expandido actual y se poda, luego, se continúa el algoritmo por el nodo hermano que ahora pasará a ser el expandido.
  - La función f de un nodo en principio es como se describe anteriormente, cuando se expande, la función f del nodo es el mínimo de las funciones f de sus hijos. Esto se propaga hasta llegar al nodo raíz.
- **Búsqueda por Franjas / Fringe Search**
  - Surge como una variante del IDA*, se sabe que en cada iteración va regenerando nodos una y otra vez, en ciertos problemas esto es poco eficiente. La idea que se propone aquí es almacenar la frontera de exploración aunque ahora surge el problema de que no se sabe qué camino es el que se utilizó para llegar a esa frontera.

- Se mantiene también una memoria de los nodos que se han visitado, se utiliza una tabla para almacenar los estados repetidos; esto hace que este algoritmo funcione mejor en aquellos problemas en los que se pueda acceder rápidamente a un nodo en la tabla hash, por ejemplo en planificación de caminos.
- Se mantienen también dos listas: la lista now y la lista $later$. El algoritmo sigue siendo retroactivo, comienza desde el nodo raíz que se encuentra en la lista now y expande los nodos hasta que llega a una cota, pero los nodos que superan la cota -que serían los nodo frontera- se almacenan en la lista $later$. En la siguiente recursión, la lista $later$ pasa a ser la lista now, por lo tanto la búsqueda se retoma desde la frontera, sin tener que reexplorar los nodos anteriores. Estos nodos se guardan en la tabla hash, las listas now y $later$ están siendo constantemente sobreescritas.
- Una vez que se llega al objetivo, se recupera el camino con la tabla hash que posee el nodo con mejor g por el que se ha pasado para llegar al objetivo.



Algoritmos de Búsqueda para Planificación de Caminos con Otros Modelos de Representación:

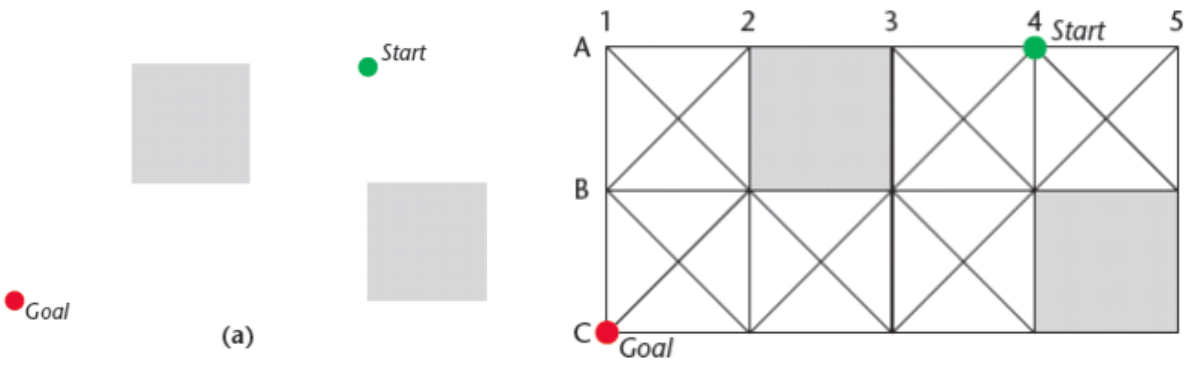
- La planificación de caminos tiene muchas representaciones del espacio, la representación que se lleva hasta ahora es que cada celda del mapa es cuadriculada y el agente ocupa la cuadrícula como tal.
- No todos los algoritmos funcionan con esta representación.

Modelo de Celdas

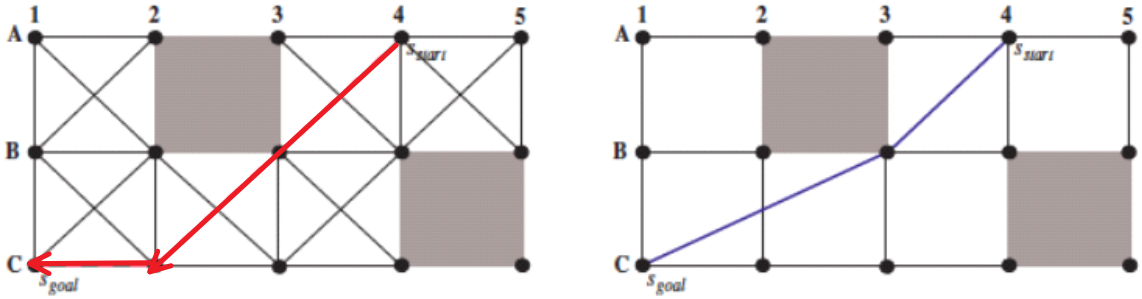
- El agente ahora se localiza en una de las esquinas de la celda en vez de ocuparla entera: es un punto en el espacio, pero los obstáculos si la pueden ocupar en su totalidad, es decir, si en la realidad un objeto ocupa parcialmente una celda se hace que esa celda entera esté ocupada.
- El agente puede pasar cerca de los "obstáculos" porque en realidad existe una sobreestimación del tamaño

de los obstáculos en la representación, por lo que esto no supone un problema.

- El agente puede moverse en 8 direcciones, las 4 originales y 4 diagonales.
- Es propia de planificación de caminos.



- El A* consigue el camino óptimo de la representación que tiene, aunque este camino no es necesariamente el camino óptimo real.

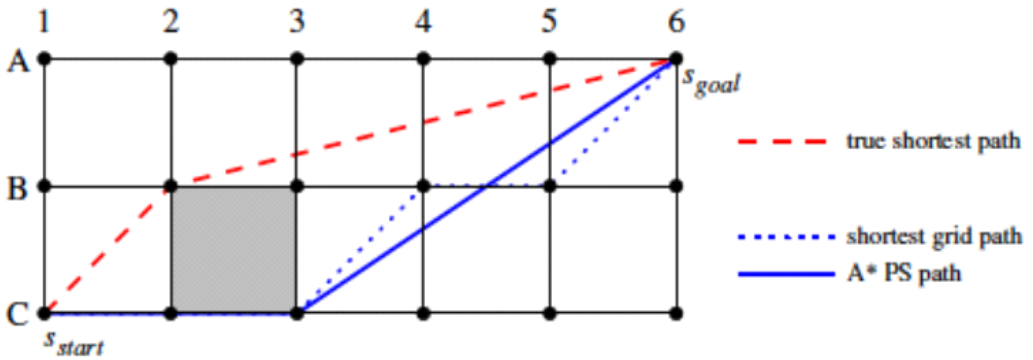


(a) Grid path (b) True shortest path

- Como esto es un problema, se han diseñado variantes que toman en consideración esto.

A* con Caminos Suavizados a Posteriori

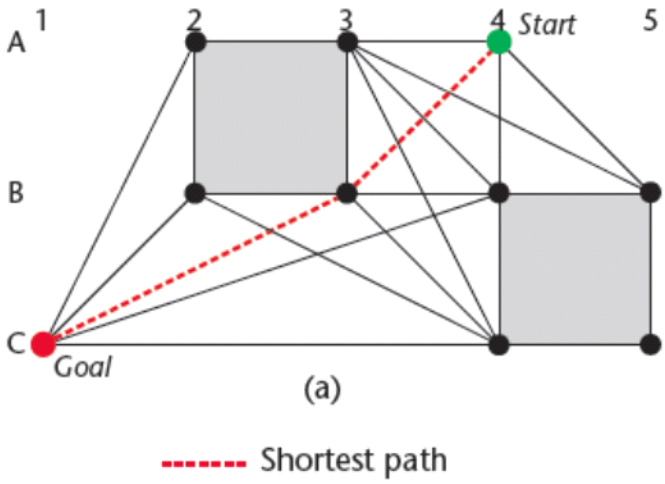
- Esta variante funciona exactamente igual que el A*, pero una vez que se ha obtenido el camino se revisa intentando conectar directamente algún vértice del camino con el vértice objetivo para acortar la distancia del camino. Se dice que dos vértices son visibles si se puede trazar una línea recta entre ellos sin que pase por un obstáculo.



A* con Grafos de Visibilidad

- Se modifica la representación de la cuadrícula, ahora partiendo desde los vértices de los obstáculos y los

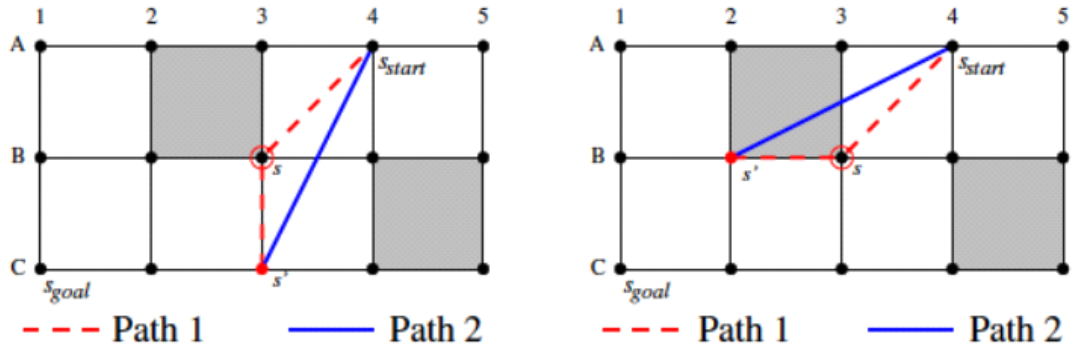
puntos de inicio y de final, se empiezan a trazar arcos entre los puntos que sean visibles de los obstáculos con los puntos de interés. La representación resultante permite que, si se realiza el A* en el grafo de visibilidad, se obtendrá el camino óptimo real.



- El problema es que esta variante es mucho más lenta que la variante original pero obtendrá el camino más corto, mientras que la discretización con rejilla si bien da caminos más largos, es mucho más rápido en su ejecución.

Algoritmo Theta* Básico

- Es una modificación del A* para que internamente logre hacer el suavizado de caminos por sí mismo.
- Dado un camino, por ejemplo, Nodo Inicio → Nodo s → Nodo s', se tendrían dos caminos:
  - El camino 1 es el camino usual que haría el A*, se considera el coste desde el nodo de inicio al nodo s más el coste del nodo s a su sucesor s'.
  - El camino 2 es realizar un camino fuera de la rejilla, considera el coste desde el mejor nodo padre de s a s' en línea recta. Esto es lo que añade Theta*.
- De estas dos opciones, se elige el que tenga el mejor coste o bien algún camino sea no sea visible y entonces se descarta el camino 2.



Descomposición de Problemas:

- Es búsqueda por el mejor nodo pero para problemas descomponibles.
- La estructura subyacente para estos problemas es un grafo Y/O; se adapta el algoritmo A* a estos problemas.

Algoritmo Y/O*

- **Grafo Y/O**
  - Combinación de nodos Y y nodos O que indican el orden de consecución de tareas a realizar para alcanzar un objetivo.
  - Para resolver un grafo Y/O cada nodo se resuelve de la siguiente manera:
    - SI es un nodo Y, se resuelven sus hijos. Se combina la solución y se soluciona el nodo, se devuelve la solución
    - Si es un nodo O, se tiene que resolver un solo hijo y ver si devuelve solución. Si no lo hace, resolver otro hijo. Cuando algún hijo esté resuelto, se combina la solución en el nodo y se devuelve.
    - Si es un nodo terminal, se resuelve el problema asociado y se devuelve.
- **Algoritmo**
  - El algoritmo puede resolver una parte de los problemas en grafos Y/O pero no todos, ya que el algoritmo espera que cada nodo, cada subproblema, sea independiente del resto. Que se puedan resolver de forma separada, pero esto no pasa en todos los problemas.
  - Por esto, no es tan utilizado para problemas complejos pues en estos, los nodos tienen dependencias entre sí.
  - En este algoritmo no tiene sentido mantener las listas de ABIERTOS o CERRADOS, se debe mantener el grafo completo ya que los nodos por si solos no permiten analizar el problema entero.

    - GRAFO contiene el nodo de inicio.
    - Comienza un ciclo que se repite hasta que el nodo inicial quede resuelto o hasta que supere un valor MAXIMO
      - Trazar el mejor camino actual desde inicio
      - Seleccionar un nodo
      - Generar los sucesores del nodo e incluirlos de forma adecuada en el GRAFO
      - Propagar la información obtenida hacia arriba en el GRAFO

  - Para que un nodo Y/O quede resuelto basta con que un hijo O esté resuelto o todos los hijos Y estén resueltos, esto es una etiqueta que se traspasa de hijo a padre. Una vez se etiquete que el nodo raíz está resuelto, finaliza el algoritmo.
  - Puede que el problema sea irresoluble, en ese caso, se posee una constante llamada MAXIMO que almacena un coste que si lo supera la raíz, termina el algoritmo. Cuando un nodo se etiqueta como irresoluble, se le asigna h =MAXIMO, y se combina con el resto de nodos.
  - La filosofía del A* original es que en cada paso, se selecciona el mejor camino disponible. En un problema descomponible, ¿qué significa que sea el mejor camino? El mejor nodo.
  - Ahora, aparte de tomar en cuenta la valoración de un nodo, también tiene que verse si es un nodo Y o nodo O, porque si es Y, la valoración de los otros nodos cuenta también.
  - Respecto a la función heurística $f = g + h$
    - h está asociada a cada nodo.
    - g, originalmente se asocia al costo de ir de la raíz al nodo. En este caso, como hay nodos que dependen de otros, no tiene sentido una g individual. Por lo tanto g es pertenece como tal a los nodos, ahora tiene que ver si es un arco Y o un arco O; esto se propaga de los nodos a su padres. Por defecto, un solo arco tiene coste unidad y un arco multiple el costo es número de problemas, este valor se añade al costo h y es lo que se propaga hacia la raíz. Los nodos terminales solo poseen un valor h, pero el resto si posee como tal el valor f conjunto de $g + h$ dependiendo de los nodos hijos que tengan y que tipo de arco poseen

Propiedades de los Métodos Heurísticos:

Estudio teórico del Algoritmo A*

- Muchos métodos heurísticos no tienen garantías excepto aquellos de primero el mejor.
  - Para estudiar esto, se necesitan dos propiedades sobre el problema, no el algoritmo.
    1. El espacio de estados del problema debe poderse representar en un **grafo localmente finito**. Un grafo es localmente finito cuando el número de descendientes de un nodo es finito, pero no significa que el grafo en general sea finito.
    2. El coste $C\left(N_i, N_j\right) > 0\ \forall_{N_i} \forall_{N_j \in suc(N_i)}$.
- **Notación**:
  - Un nodo sucesor de otro nodo se representa como, dado N_i y N_j se dice que N_j es sucesor de N_i si $N_j \in suc\left(N_i\right)$ y esta conexión tiene un coste $C(N_i, N_j)$.
  - Los nodos iniciales se representan como $S_0 \dots S_n$, porque teorícamente puede haber más de uno pero se puede tener un nodo "virtual" S que se conecte a todos los nodos $S_0, \dots, S_n$ con un coste 0 para tener siempre un único nodo inicial, sin importar el problema.
  - Los nodos objetivo se representan como $\gamma_1, \dots, \gamma_m$. El conjunto de todos se representan como Γ y este conjunto no es lo mismo que S, no se conectan. Es solo la agrupación de todos.
  - Como en A* se explora por caminos, se representa un camino entre dos nodos N_i, N_k cualesquiera que no tienen que ser sucesores directos uno del otro como $P_{N_i - N_k}$. $\mathbb{P}_{N_i - N_k}$ representa uno o más caminos entre los nodos. También se pueden tener los caminos entre un nodo a un conjunto de nodos.
  - $P^*_{N_i - N_j}$ indica el camino óptimo entre N_i y N_j. De igual manera se tiene $\mathbb{P}^*_{N_i - N_k}$.
  - $k(N_i, N_k)$ representa el coste óptimo del camino P^* entre N_i y N_k.
  - $g^*\left(N_i\right)$ es el coste del camino óptimo entre S y N_i. Equivalente a $k\left(S, N_i\right)$. Se diferencia de la g actual porque g^* es el mejor camino teórico de todos aunque el A^* no lo haya considerado, se puede pensar que A* lo que intenta es estimar la g^* que desconoce.
  - $h^*(N_i)$ es el coste del camino óptimo P^* entre N_i y al mejor objetivo. Equivalente a $\min\left(k\left(N_i, \gamma\right)\right), \gamma \in \Gamma$.
  - C^* es el coste del camino óptimo entre S y el o los mejores objetivos, equivalente a $h^*(S)$. Es equivalente también a $g^*(\Gamma^*)$.
  - $\Gamma^* \subseteq \Gamma$, es el subconjunto de los nodos objetivo a los que se puede llegar por un camino de coste C^*.
  - $f^*\left(N_i\right) = g^*\left(N_i\right) + h^*(N_i)$ es la función teórica del mejor camino que va de S a N_i sumado al mejor camino que va de N_i a γ.
    - $f^*(S) = h^*(S) = g^*(\gamma) = f^*(\gamma) = C^*\ \forall \gamma \in \Gamma$
    - El óptimo de $h^*(S)$ es lo mismo que $g^*(\gamma)$ si es el nodo objetivo óptimo y también coincide con el coste asociado.

Propiedades de f^*:

- La propiedad $f^*(N) = C^*\ \ \forall N \in P^*_{S-\Gamma}$, o sea que la suma de h^* y g^* es el Coste óptimo si N forma parte de un camino óptimo del nodo inicio al objetivo.

  $\forall N \in P^*_{S-\Gamma}$
  $\rightarrow \exists p\ tal\ que\ g_p(N) + h_p(N) = C^*$
  $\rightarrow \mathrm{g^*(N) + h^*(N) \leq g_p(N) + h_p(N) = C^*}$

  ¿Qué pasa si $g^*(N) + h^*(N) < C^*$? $\rightarrow \exists p'\ tal\ que\ g^*(N) = g_{p'}(N)\ y\ h^*(N) = h_{p'}(N)$
  No puede ser porque no puede existir un camino que sea mejor que el óptimo porque en ese caso ese mismo camino ya sería el óptimo, $g^*(N) + h^*(N) < C^*$ es imposible.

  $\therefore f^*(N) = g^*(N) + h^*(N) = C^*$

- La propiedad $f^*(N) > C^*\ \ \forall N \notin P^*_{S-\Gamma}$, o sea que el mejor camino de un nodo que no forma parte del camino óptimo no podrá ser igual o menor que C^*.

$\forall N \notin P^*_{S-\Gamma}$ suponer $f^*(N) \leq C^*$
$\rightarrow g^*(N) + h^*(N) \leq C^*$
$\rightarrow \exists p' \; tal \; que \; g_{p'}(N) + h_{p'}(N) \leq C^*$
$\rightarrow N \in P^*_{S-\Gamma}$ lo que es una contradicción, por lo tanto queda demostrado.

- El algoritmo A* desea obtener un camino de costo óptimo entre S y Γ. Hace uso de f, g y h.
  - f es la estimación que hace el A* de la f^*, porque si se pudiera obtener la f^* no habría necesidad realizar una búsqueda porque ya se tiene el mejor camino posible.
  - g es la estimación que hace el A* de la g^*, es el mejor camino de los visto hasta ahora.
    - Se tiene que $g(S) = 0$
    - $g_p(N_i)$ es el coste acumulado de los arcos desde S hasta N_i por un camino cualquiera.
    - Propiedad: $g_p(N_i) \geq g^*(N_i)$ esto es así porque g^* es el mejor de todos, por lo tanto g_p no puede ser mejor porque entonces sería ya g^*.
  - h es una heuristica definida que utiliza el A*.
    - Se tiene que $h(\gamma) = 0$
    - $h_p(N_i)$ es el coste acumulado desde N_i hasta γ por un camino cualquiera.
    - Propiedad: $h_p(N_i) \geq h^*(N_i)$ esto es así porque h^* es el mejor de todos, por lo tanto h_p no puede ser mejor porque entonces sería ya h^*.

Completitud: *¿A* encuentra solución?*

- Se dice que un algoritmo es completo si alcanza una solución cuando esta existe.
- Primero, se comienza suponiendo que el grafo es finito…
  - En cada paso del algoritmo se selecciona un nodo, y junto a ese nodo también se está seleccionando un camino entonces un nodo equivale a un camino. Entonces cada paso equivale a seleccionar un camino y se detiene una vez se llega a la solución.
  - El hecho de que el coste no pueda ser negativo evita que se produzcan ciclos, ya que si un nodo tiene un padre que es un ciclo, tendrá un coste peor a otro padre que no forma un ciclo. Por lo tanto, A* no explora los ciclos.
  - Como se tiene que es un grafo finito, entonces la cantidad de caminos acíclicos es finito, entonces, A* está garantizado que finaliza.
  - En un grafo finito, A* es completo:
    $\exists P_{S-\gamma}$ pero supongamos que A* no es completo entonces ABIERTOS se quedará vacío y no encontrará solución, esto no se cumple ya que, en primer lugar, S entrará en ABIERTOS, y cuando salga, entrarán todos su sucesores. Por lo tanto, entrará un sucesor que estará en el camino P solución y eso se repetirá hasta que salga γ y como encuentra solución se contradice con que ABIERTOS se quede vacío.
- Ahora, A* es completo suponiendo que el grafo es localmente finito y el coste es siempre positivo.
  - Se supone nuevamente que $\exists P_{S-\gamma}$ pero el algoritmo no la encuentra.
  - Nuevamente, siempre habrá un nodo de $P_{S-\gamma}$ en ABIERTOS porque inicialmente entra S en ABIERTOS, luego entran los hijos de S y eventualmente entrará un hijo que sea parte de ese camino.
  - Ahora, si no encuentra solución pueden pasar dos cosas:
    - El algoritmo termina, esto es una contradicción análoga a la demostración anterior en un grafo finito.
    - El algoritmo no termina, esto quiere decir que está indefinidamente explorando nuevos caminos, pero se sabe que los grafos localmente finitos poseen una cantidad finita de caminos de longitud finita entonces si el algoritmo no termina es porque ya exploró esos caminos sin encontrar solución (que si existe) y está explorando caminos infinitos pero en ese caso pasaría en $f(n) = g(n) + h(n)$ que $g(n) = \infty$ lo cual haría que $f(n) = \infty$ y esto no puede suceder ya que si hay nodos en ABIERTOS que tienen valoración finita e infinita, se seleccionarán siempre los finitos primero por el criterio de selección de A*, por lo tanto se llega a una contradicción porque eventualmente llegaría a la solución por un camino finito.
  - Por lo tanto, queda demostrado que A* es completo si el coste es siempre positivo y se exploran grafos localmente finitos.

Admisibilidad: *¿A* encuentra la solución óptima?*

- Para que una heurística h sea admisible se tiene que puede serlo si y solo si $h(n) \leq h^*(n) \forall n$.

- **Lema 1**: Para cualquier nodo n que no esté en CERRADOS, y para cualquier camino óptimo entre S y n llamado P siempre se puede seleccionar un nodo n' que está en P y está en ABIERTOS en el que $g(n') = g^*(n')$.
  - Partiendo de n se selecciona un camino óptimo $P = (n_0 = S, n_1, \dots, n_k = n)$ ahora queremos encontrar n'.
  - Si inicialmente S está en ABIERTOS, entonces S está en P y en ABIERTOS, y $g(S) = g^*(S) = 0 \rightarrow n' = S$ luego se demuestra el Lema 1.
  - Si S no está en ABIERTOS, primero, se define un conjunto Δ que contiene nodos m que están en CERRADOS y en P para los que $g(m) = g^*(m)$. $\Delta \neq \emptyset$ ya que $S \in \Delta$, si de Δ se escoge el nodo de mayor índice (de la numeración anterior, $n_0 = S, n_1, \dots$) y se llama n^* que es un nodo que está en CERRADOS, está en P, tiene $g(n^*) = g^*(n^*)$ y tiene el índice más alto. Se tiene un n' es el sucesor de n^* en P pero n' estaría en ABIERTOS porque si estuviera en CERRADOS, ya sería n^*.
  - Se tiene que $g(n') \leq g(n^*) + C(n^*, n')$ porque n' sería igual al costo g del padre más el costo o bien podría tener otro padre mejor. Como n^* está en Δ se sabe que $g(n^*) = g^*(n^*)$, entonces $g(n^*) + C(n^*, n') = g^*(n^*) + C(n^*, n') = g^*(n')$ y si se conectan los extremos, se tiene que $g(n') = g^*(n')$ y por lo tanto se demuestra el Lema 1.
- **Lema 2:** Si h es admisible y se supone que A* no ha terminado entonces para cualquier camino óptimo P entre S y γ siempre se puede seleccionar un nodo n' en P y en ABIERTOS tal que la $f(n') \leq C^*$.
  - Si A* no ha terminado, entonces γ no puede estar en CERRADOS entonces tomando el Lema 1, siempre se puede encontrar un camino P de S a γ donde $g(n') = g^*(n')$
  - $f(n') = g(n') + h(n') = g^*(n') + h(n')$ y si h es admisible entonces $h(n') \leq h^*(n')$, entonces $g^*(n') + h(n') \leq g^*(n') + h^*(n')$ y queda entonces que $f(n') \leq f^*(n') = C^*$. Luego, queda demostrado el Lema 2.
  - Lo que dice este lema es que si A* no ha terminado siempre habrán nodos por debajo de la cantidad óptima si h es admisible.
- **Teorema**: Si h es admisible entonces A* es admisible.
  - Si se supone que h es admisible pero A* no es admisible entonces A* no encuentra la solución óptima, o sea que es $> C^*$. El nodo que da esa solución se le llama $t \in \Gamma$.
  - Si se sitúa en el momento que t está en ABIERTOS y es el mejor nodo en ese instante. Por lo que dice el Lema 2, en ABIERTOS también estará un nodo n' tal que $f(n') \leq C^*$. En ese caso se tendrá que $f(n') \leq C^* < f(t)$ por lo que no puede pasar que t sea el mejor nodo de ABIERTOS, se ha encontrado una contradicción.
- Por lo tanto, queda demostrado que A* encuentra la solución óptima solamente si la heurística es admisible y que el grafo sea localmente finito y que el coste sea siempre positivo.

Dominancia: ***Cuando hay dos heurísticas para un problema, ¿cuál es la mejor?***

- Dada una versión del A* A_1 con una heurística h_1 domina a otra versión A_2 con h_2 si cada nodo expandido por A_1 también es expandido por A_2, A_1 domina a A_2 porque si cada nodo que expande A_1 también lo expande A_2, entonces A_2 expande más nodos luego A_1 llega a la solución expandiendo menos nodos y por lo tanto es más eficiente.
- En general, no hay un estudio teórico que garantice que h_1 sea mejor o peor que h_2 excepto en el caso que ambas sean admisibles y que $h_1 > h_2$ para todo nodo, se dice que h_1 está más informada que h_2. La dominancia está relacionado con una heurística más informada y por lo tanto, mejor.
- Esto se puede entender como que, si bien h_1 y h_2 están por debajo del óptimo h^*, si se tiene que $h_1 > h_2$ eso quiere decir que h_1 está más cerca de h^* y por lo tanto expande menos nodos (si $h_1 = h^*$ sería ir al camino óptimo directamente), por lo tanto, si está más cerca es que la estima mejor. Si está más alejada y más cercana a cero expande más nodos y empieza a asemejarse al costo uniforme.
- ¿Qué ocurre con $h_1 \geq h_2$?
  - Depende del criterio que tome cada versión del algoritmo en el desempate, no se puede concluir claramente una dominancia pero los algoritmos tendrán un comportamiento similar pero no igual, no tienen porque expandir los mismos nodos por el criterio de desempate que aplican.
  - Aunque el número de nodos expandidos por h_1 será más pequeño que h_2.
- ¿Cuál es el interés de utilizar $f(n) = g(n) + \omega \cdot h(n)$?
  - Se tiene un peso ω
    - Cuando vale 1, es el A* normal.
    - Cuando vale 0, es el algoritmo de costo uniforme.

        - Generalmente se puede usar 2 o 3 porque potencian la heurística y aceleran mucho el algoritmo.
    - Muchas heurísticas no están bien informadas, por lo que están alejadas de h^*, al multiplicar por un valor positivo lo que se hace es hacerla más informada, aunque esto tiene el riesgo de que si el valor es muy alto se supere h^* y entonces la heurística deje de ser admisible y no se garantice la solución óptima.
    - Son estudios experimentales, el valor de ω se va variando y visualizando la reducción de los nodos expandidos para los valores.

¿Es posible encontrar heurísticas que hagan más eficiente el A*?

- Heurísticas Monótonas
    - El algoritmo A* puede tecnicamente utilizar cualquier función h, aunque esto no garantiza que funcione bien.
    - Hay una propiedad de h^* que se va a necesitar que tenga h.
        - Si se tiene S, n y γ se puede ver que el coste óptimo de S a γ será más pequeño o igual que el coste de pasar de S a n y de n a γ. Es una desigualdad triangular $k(S,\gamma) \leq k(S,n) + k(n,\gamma)$ y como se habla de h^*, $h^*(S) = k(S,\gamma)$ y entonces $h^*(S) \leq k(S,n) + h^*(n)$.
        - Se puede generalizar como $h^*(n) \leq k(n,n') + h^*(n')\ \forall n, n'$
        - **Consistencia**: Lo que pasa con h es que no tiene porqué cumplir esta propiedad, pero h será consistente si cumple que $h(n) \leq k(n,n') + h(n')\ \forall n, n'$.
        - **Monotonía**: h es monótona si $h(n) \leq C(n,n') + h(n')\ \ \forall n\ \forall n' \in suc(n)$.
        - Si se tiene una función consistente, también es monótona, dado que si se verifica para cualquier par de nodos, se verifica para el nodo y su sucesor. Y si una función es monótona, también es consistente. Son propiedades equivalentes.
    - **Teorema**: Toda heurística consistente o monótona es admisible y por lo tanto garantiza la solución óptima.
        - Si h es consistente entonces $h(n) \leq k(n,n') + h(n')$, si $n' = \gamma \in \Gamma^*$ entonces $\mathrm{h(n)} \leq \mathrm{k}(n,\gamma) + h(\gamma)$, $h(\gamma) = 0$, entonces $k(n,\gamma) = h^*(n)$ y por lo tanto $h(n) \leq h^*(n)$.
    - **Teorema**: Un algoritmo A* guiado por una heurística monótona alcanza el camino óptimo a todos los nodos expandidos, es decir, $g(n) = g^*(n)\ \forall n \in CERRADOS$.
        - Si suponemos un caso en que se tiene un nodo n que es el mejor de ABIERTOS, va a entrar en CERRADOS pero no verifica el teorema, entonces $g^*(n) < g(n)$.
        - Se tiene un camino P_{S-n} y hay un nodo n' que es antecesor de n en P y en ABIERTOS que verifica que $g(n') = g^*(n')$, luego se tiene que $f(n') = g(n') + h(n') = g^*(n') + h(n') \leq g^*(n') + k(n',n) + h(n) = g^*(n) + h(n)$.
        - Por lo tanto, $f(n') < f(n)$ por lo que no se podría tener que n es el mejor de ABIERTOS porque n' es mejor.
    - Esto es importante porque esto evita que se tengan que revisar nodos en CERRADOS, lo que es mucho más costoso que revisarlos en ABIERTOS.

<u>Heurísticas a través de Modelos Simplificados:</u>

¿Cómo se generan las heurísticas?

- Se tiene un problema donde se desea obtener la h, entonces se simplifica el problema, ya sea quitar restricciones o simplificar condiciones, que sea básicamente lo mismo pero que pueda resolverse de manera algorítmica, al resolverlo se sabe cuantos pasos hay desde ese estado a la solución y se conoce la h^* y por lo tanto la h es la h^* de un problema simplificado.

¿Una heurística obtenida de esta manera tienen buenas propiedades?

- **Teorema**: Toda heurística obtenida por un modelo simplificado es consistente, por lo tanto, monótona y por lo tanto admisible.
    - h_s^* del problema simplificado cumple con $h_s^*(n) \leq C_s(n,n') + h_s^*(n')$, como es la h_s^* cumple con la condición de monotonía del problema simplificado.
    - El costo de $C_s(n,n') \leq C(n,n')$
    - Como la h del problema real es la h_s^* del simplificado, se tendrá que h es también monótona y admisible, $h(n) \leq C(n,n') + h(n')$.
- Por lo tanto, toda heuristica obtenida de esta manera está garantizada que es monótona y admisible.
- Esta técnica funciona pero si se puede resolver el problema simplificado de manera exacta y algorítmica. Otro

problema es que las heurísticas que se deducen son muy pocos informadas y por lo tanto no son tan buenas.

Proceso Sistemático

- Es una de las formas de como llegar a un modelo simplificado con una metodología sistemática.
- Para obtener versiones simplificadas de un problema, es necesario tener una descripción formal del problema, por ejemplo, en lógica de predicados.
- Una vez se tienen los predicados y las acciones, esto es lo que realmente determina que acciones pueden realizarse y cuales no.
- Las versiones simplificadas no deben cambiar la esencia del problema, pero deben quitarse las condiciones que no modifiquen la idea general, al menos la que lo hagan en menor grado.
- Esta técnica es muy utilizada en IA, especificamente, muchos programas que resuelven problemas de satisfacción de restricciones generan heurísticas de esta manera.