Tema 3: Búsqueda heurística: métodos locales y de búsqueda por el mejor nodo.

- Heurísticas
- Métodos de escalada
- Búsqueda primero el mejor

Objetivos

- Continuar con el proceso de
  - adquirir las habilidades básicas para construir sistemas capaces de resolver problemas mediante técnicas de IA.
  - entender que la resolución de problemas en IA implica definir una representación del problema y un proceso de búsqueda de la solución.
  - analizar las características de un problema dado y determinar si es susceptible de ser resuelto mediante técnicas de búsqueda. Decidir en base a criterios racionales la técnica más apropiada para resolverlo y saber aplicarla.

Objetivos

- Entender que la resolución de problemas en IA implica definir una representación del problema y un proceso de búsqueda de la solución.
- Conocer la representación de problemas basados en estados (estado inicial, objetivo y espacio de búsqueda) para ser resueltos con técnicas computacionales.
- Conocer las técnicas más representativas de búsqueda no informada en un espacio de estados (en profundidad, en anchura y sus variantes), y saber analizar su eficiencia en tiempo y espacio.
- Entender el concepto de heurística y analizar las repercusiones en la eficiencia en tiempo y espacio de los algoritmos de búsqueda.
- Conocer las técnicas más representativas de búsqueda informada en un espacio de estados (búsqueda local, algoritmo A^*).

Estudia el tema en …

- Nils J. Nilsson, "*Inteligencia Artificial: Una nueva síntesis",* Ed. Mc Graw Hill, 2000. pp. 53-62, 125-146, 163-174
- • S. Russell, P. Norvig, "*Inteligencia Artificial: un Enfoque Moderno"*, Segunda Edición, Ed. Pearson-Prentice Hall, 2004.

Contenido

- Diseño de un agente deliberativo: búsqueda
- Ejemplos
- Búsqueda sin información
- Problemas descomponibles y búsqueda
- Búsqueda con información

Agente Deliberativo

- El agente dispone de un modelo del mundo en el que habita
- El agente dispone de un modelo de los efectos de sus acciones sobre el mundo
- El agente es capaz de razonar sobre esos modelos para decidir que hacer para conseguir un objetivo

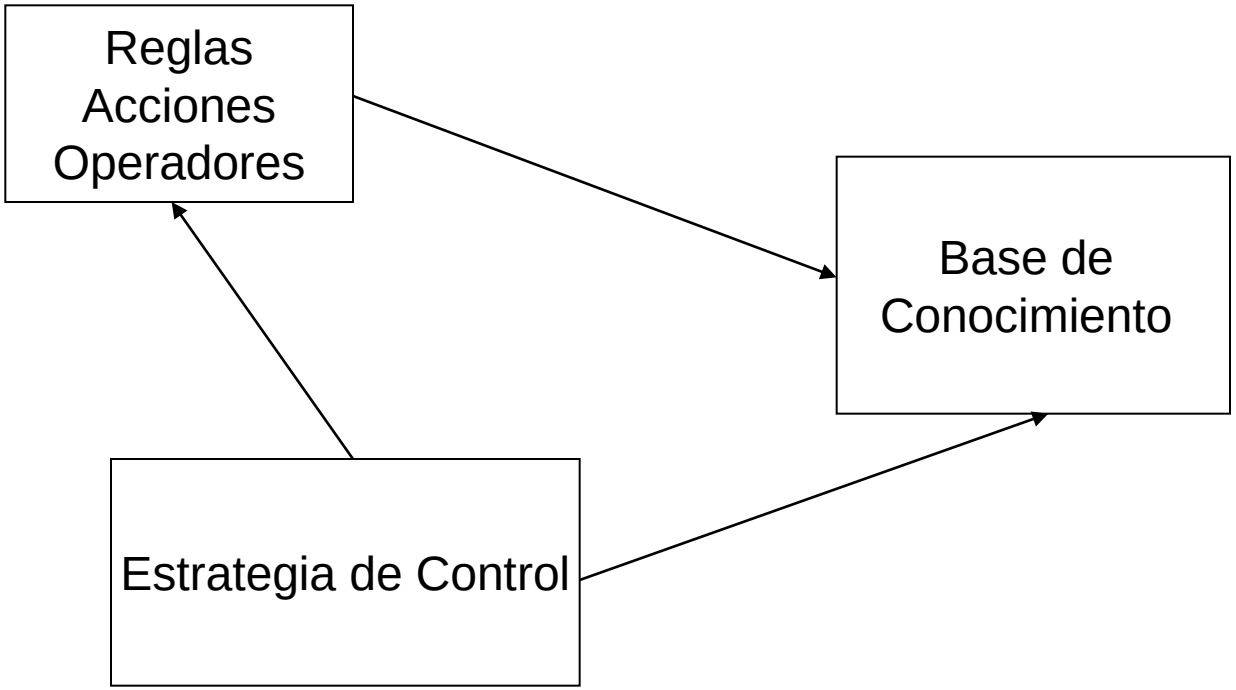
Espacio de estados

- •Espacio de estados –Representación del conocimiento a través de las acciones del agente
- •Búsqueda en el espacio de estados – Resolución del problema mediante proyección de las distintas acciones

Espacio de estados Grande

- Imposible representar explícitamente todo el grafo de estados.
- Necesidad de recurrir al grafo implícito y explicitar solo el grafo o árbol de búsqueda.
- Necesidad de técnicas de búsqueda.

Técnicas de Búsqueda



Estrategias de control (Russelll-Norvig)

- Escoger el nodo a expandir,
- Escoger como expandirlo.

- Debe producir cambios
- Debe ser sistemática
- Debe “garantizar” construir la solución

Procedimiento Búsqueda

1. DATOS ← base de datos inicial
2. **until** DATOS satisface la condición de terminación
   do
3. **begin**
4. **select** alguna regla R en el conjunto de reglas que pueda ser aplicada a DATOS
5. DATOS ← resultado de aplicar R a DATOS
6. **end**

Estrategias de control (Russelll-Norvig)

```
función BÚSQUEDA-ÁRBOLES(problema,frontera) devuelve una solución o fallo
  frontera ← INSERTA(HACER-NODO(ESTADO-INICIAL[problema]), frontera)
  hacer bucle
    si VACIA?(frontera) entonces devolver fallo.
    nodo ← BORRAR-PRIMERO(frontera)
    si TEST-OBJETIVO[problema] aplicado al ESTADO[nodo] es cierto
      entonces devolver SOLUCIÓN(nodo)
    frontera ← INSERTAR-TODO(EXPANDIR(nodo,problema),frontera)
```

Estrategias de control(Russell-Norvig

```
función EXPANDIR(nodo,problema) devuelve un conjunto de nodos
  sucesores ← conjunto vacío
  para cada (acción,resultado) en SUCESOR-FN[problema](ESTADO[nodo]) hacer
    s ← un nuevo NODO
    ESTADO[s] ← resultado
    NODO-PADRE[s] ← nodo
    ACCIÓN[s] ← acción
    COSTO-CAMINO[s] ← COSTO-CAMINO[nodo] + COSTO-INDIVIDUAL(nodo,acción,s)
    PROFUNDIDAD[s] ← PROFUNDIDAD[nodo] + 1
    añadir s a sucesores
  devolver sucesores
```

Rendimiento de una estrategia de Control

- **Completitud:** ¿encuentra una solución(si existe)?
- **Optimalidad:** ¿Encuentra una solución optima en algun sentido?
- **Complejidad en tiempo,**
- **Complejidad en espacio, irrevocable vs tentativa**
- **Costo de la búsqueda, Costo total.**

Tipos de busqueda

- Búsqueda sin información: El siguiente nodo a expandir se escoge sin emplear conocimiento explicito sobre la distancia entre el nodo en curso y el objetivo.
- Búsqueda heurística: El siguiente nodo a expandir emplea algún tipo de información para guiar la búsqueda.
- En el resto del tema desarrollaremos estos dos tipos de búsqueda

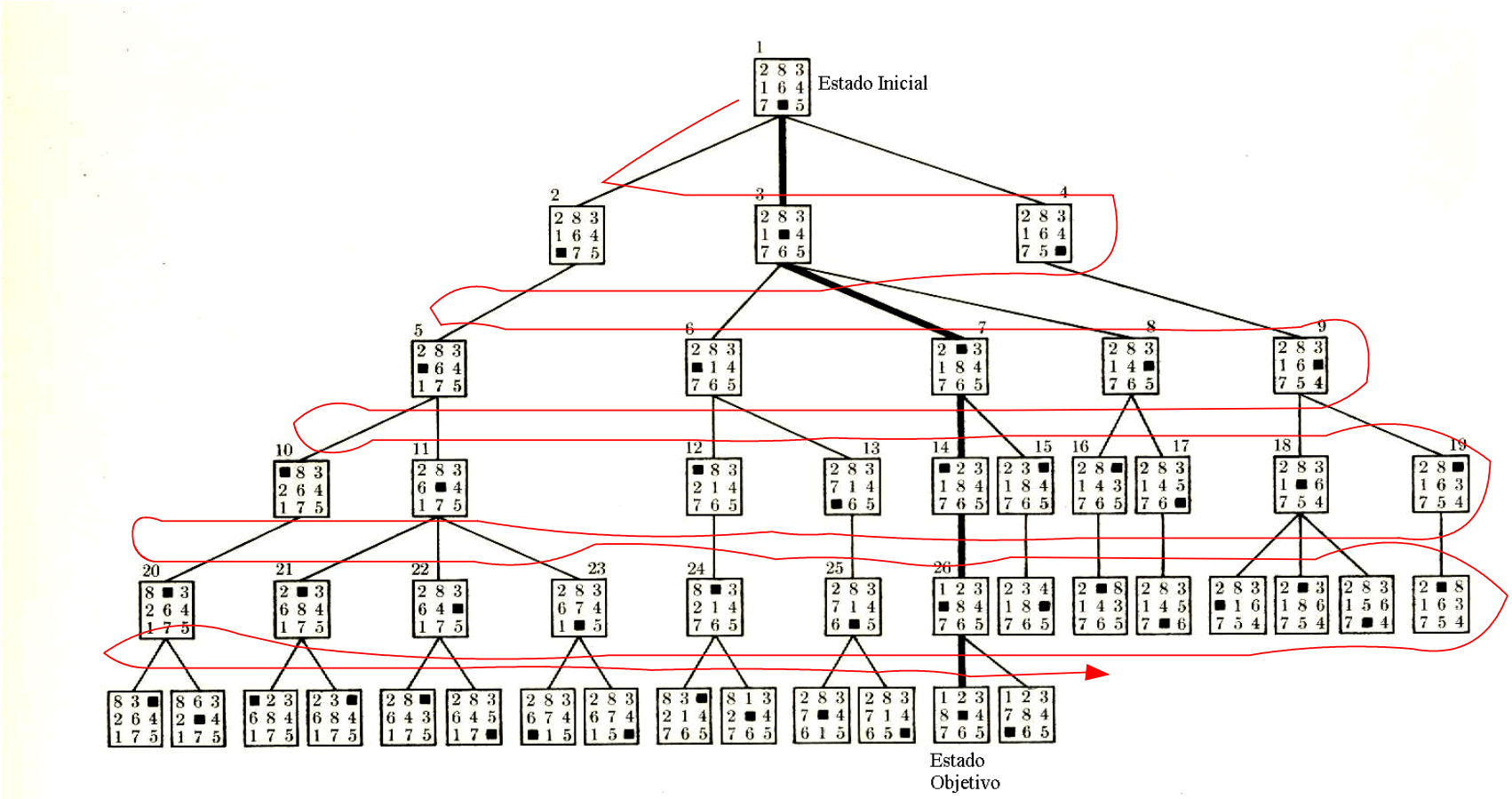
Búsqueda sin información

- Búsqueda en anchura,
- Búsqueda con costo uniforme,
- Búsqueda en profundidad,
- Búsqueda bidireccional.

Búsqueda en anchura

Desde el estado inicial analizar todos los sucesores de cada nodo antes de pasar al nivel siguiente en el arbol de búsqueda

Búsqueda en anchura



Búsqueda en anchura

1. Crear una variable llamada LISTA-NODOS y asignarle el estado inicial
2. Hasta que se encuentre un estado objetivo o LISTA-NODOS esté vacía, hacer:
   1. Eliminar el primer elemento de LISTA-NODOS y llamarlo E. Si LISTA-NODOS está vacía, terminar
   2. Para cada regla que se empareje con el estado descrito en E hacer:
      1. Aplicar la regla para generar un nuevo estado
      2. Si el nuevo estado es un estado objetivo, terminar y devolver este estado
      3. En caso contrario, añadir el nuevo estado al final de LISTA-NODO

Búsqueda con costo

Se supone que el expandir el nodo Ei con la regla Rij para obtener el nodo Ej tiene un costo Cij.

De este modo cualquier nodo Ek en el árbol de búsqueda tiene un costo Ck igual a la suma de los costos de los arcos que conducen a él desde la raíz del árbol.

El procedimiento de búsqueda expande el nodo Ep con menor Cp de todos los de la frontera.

Búsqueda con costo

1. Poner el nodo inicial s en una lista, llamada ABIERTOS, de nodos no expandidos.
2. Crear una lista, llamada CERRADOS, de nodos expandidos, inicialmente vacía.
3. Si ABIERTOS está vacía no existe solución. Terminar.
4. Suprimir de ABIERTOS el nodo i con mínima g(i) y colocarlo en CERRADOS.
5. Si i es un nodo objetivo se ha encontrado la solución. Terminar.
6. En otro caso, expandir el nodo i, si no tiene sucesores ir a 3.
7. Para cada nodo sucesor j del nodo i, insertarlo correctamente en ABIERTOS, asignando g(j)=g(i)+c(i,j).
8. Ir a 3.

Búsqueda con costo

Esta a estrategia se reduce a la búsqueda en anchura si Cij=1 para todo i y todo j.

El método es completo y optimal en costo si Cij es mayor que una cierto C para todo i y todo j.

Se puede mejorar considerando el hijo de menor costo de todos los de la frontera (complica algorítmicamente)

Búsqueda primero en profundidad

- Desde el estado inicial ir analizando un sucesor del nodo de mayor nivel generado hasta el momento.

Búsqueda primero en profundidad

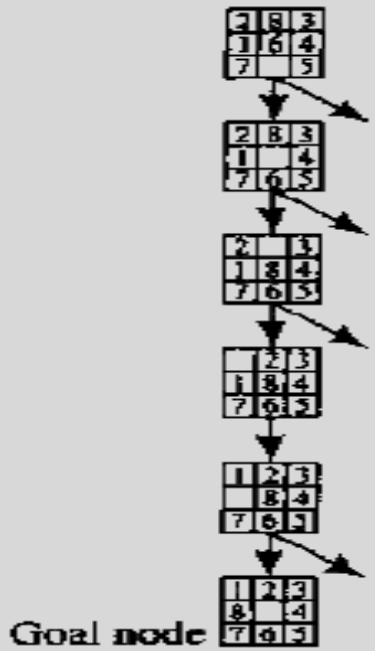


Figure 8.4

The Graph When the Goal Is Reached in Depth-First Search

Búsqueda primero en profundidad

1. Si el estado objetivo es un estado final: STOP/ EXITO
2. Hasta encontrar un éxito o un fracaso hacer:
   a) Generar un sucesor, **E**, del estado inicial que no haya sido analizado. Si no lo hay marcar FRACASO.
   b) Llamar recursivamente a este algoritmo con **E** como estado inicial.
   c) Alcanzado un ÉXITO o un FRACASO Stop.

Decisión ¿Qué guardar?

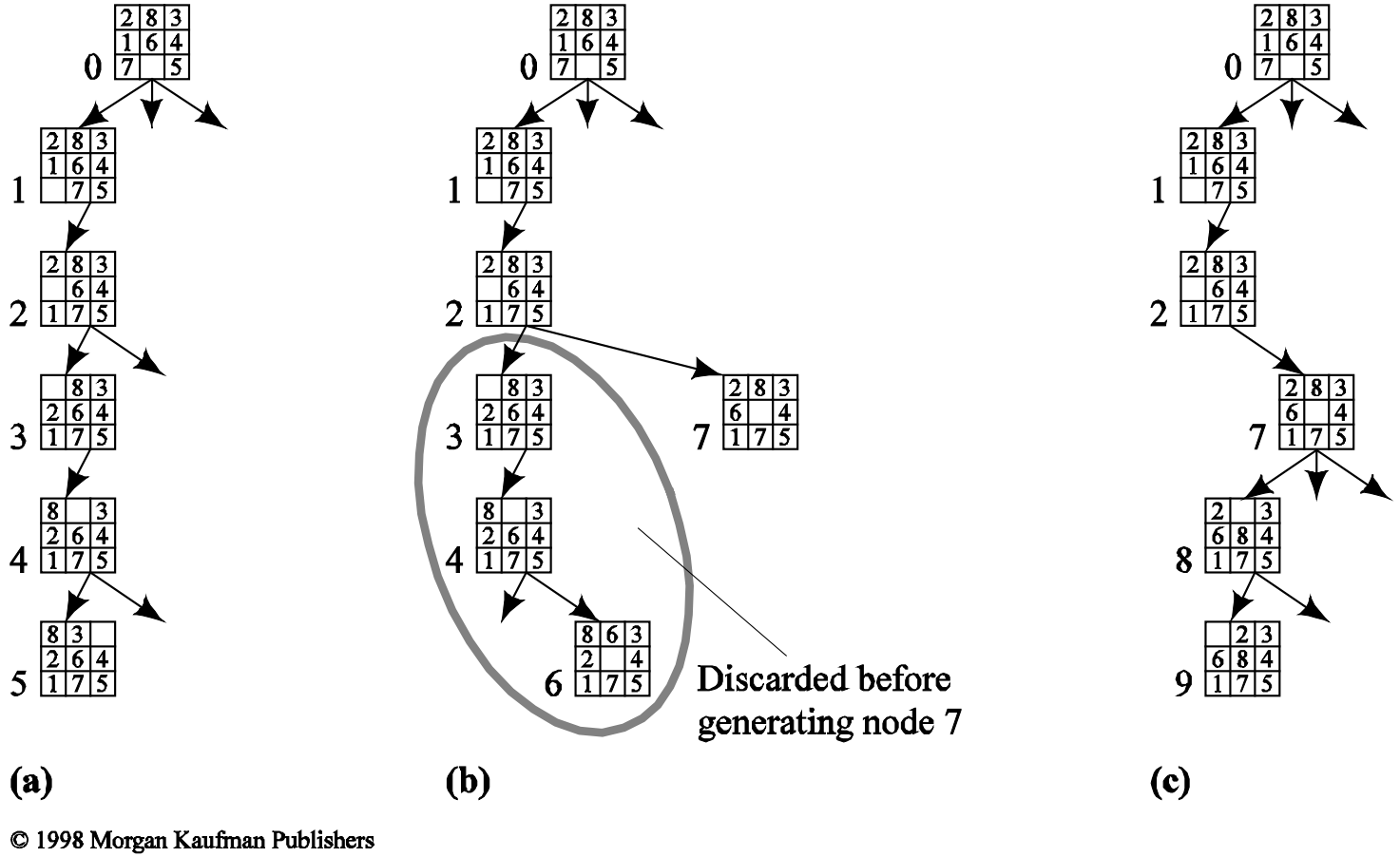
Búsqueda primero en profundidad

- Problema: profundizar indefinidamente

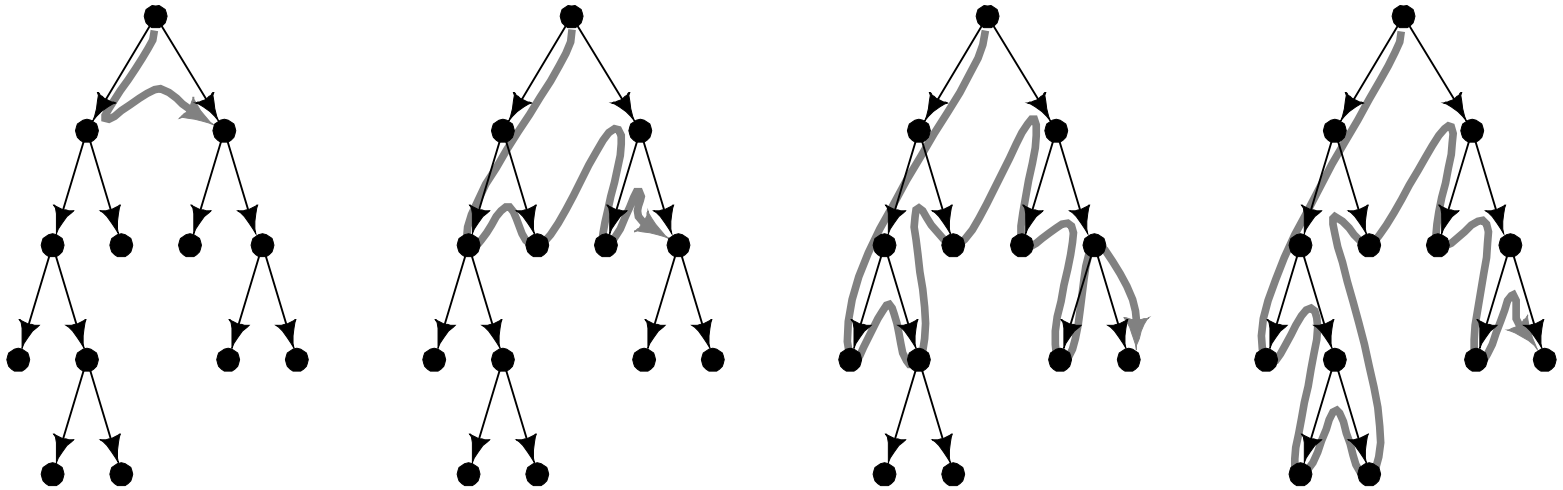
  Solución: Acotar la profundidad,

  - Busqueda con retroceso (Backtraking) a profundidad fijada o retroceso cronológico.
  - Búsqueda con profundización iterativa

Búsqueda con backtraking cronológico



Descenso iterativo



Anchura /Profundidad

- La búsqueda en profundidad consume menos memoria,
- Con “un poco de suerte” la busqueda en profundidad encuentra pronto el camino del nodo inicial al nodo final,
- La búsqueda en anchura no queda atrapada en callejones sin salida,
- La búsqueda en anchura encuentra siempre el camino, si existe, y además el más corto.

Búsqueda en grafos: Nodos repetidos

- Descripción para etiquetar el nodo inicial
- Las funciones para transformar las descripciones de los estados: operadores
- Una memoria para guardar los nodos previamente obtenidos
- Una condición de finalización.

Problemas descomponibles

- Son aquellos en los que el problema se puede dividir en subproblemas que podemos resolver de forma independiente .

- La solución global se obtiene “integrando” las soluciones de los subproblemas.

Problemas descomponibles

- Base de datos inicial (C,B,Z)
- Operadores

  R1: C $\longrightarrow$ (D,L)

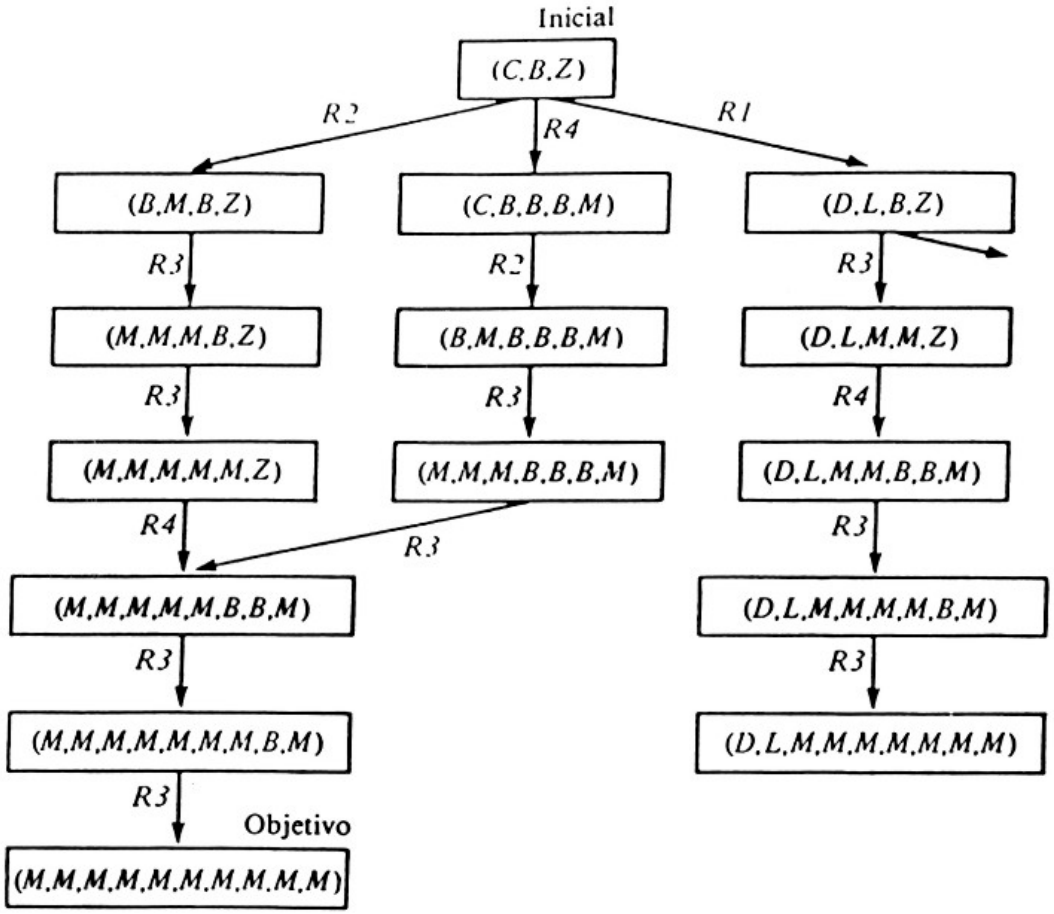
  R2: C $\longrightarrow$ (B,M)

  R3: B $\longrightarrow$ (M,M)

  R4: Z $\longrightarrow$ (B,B,M)

- Objetivo: (MMMMMMMMMMM)

Resolución del problema



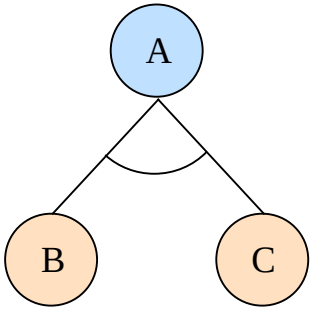
Grafo Y/O

- Descomposición de problemas: arcos Y
- Resolución de problemas: arcos O

- Concepto de solución: subgrafo solución

Grafo Y/O

- **Grafo Y:** Para completar el objetivo/tarea **A**, es necesario terminar antes los objetivos/tareas **B** y **C**.

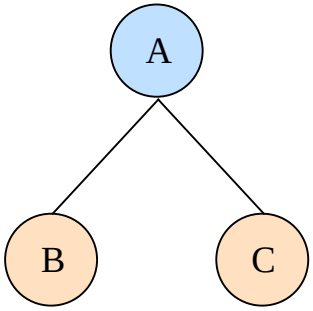


- En el cálculo proposicional, la expresión del grafo Y anterior correspondiente sería de la siguiente forma:

$$B \cdot C \rightarrow A$$

Grafo Y/O

- **Grafo O:** Para completar el objetivo/tarea **A**, es necesario terminar antes o bien el objetivo/tarea **B**, o bien el objetivo/tarea **C**.

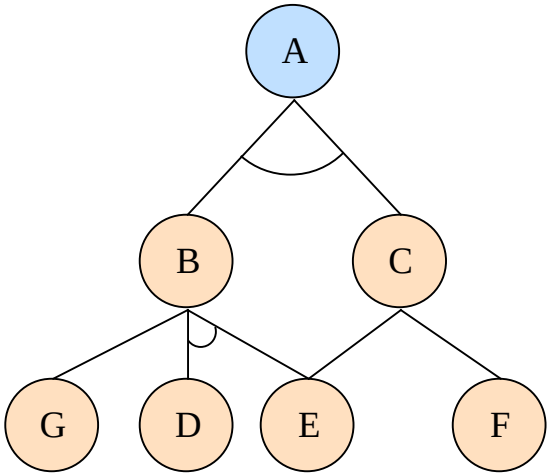


- En el cálculo proposicional, la expresión del grafo O anterior correspondiente sería de la siguiente forma:

$$B+C \rightarrow A$$

Grafo Y/O

- **Grafo Y/O:** Combinación de grafos Y y grafos O que indican el orden de consecución de tareas a realizar para alcanzar el objetivo.



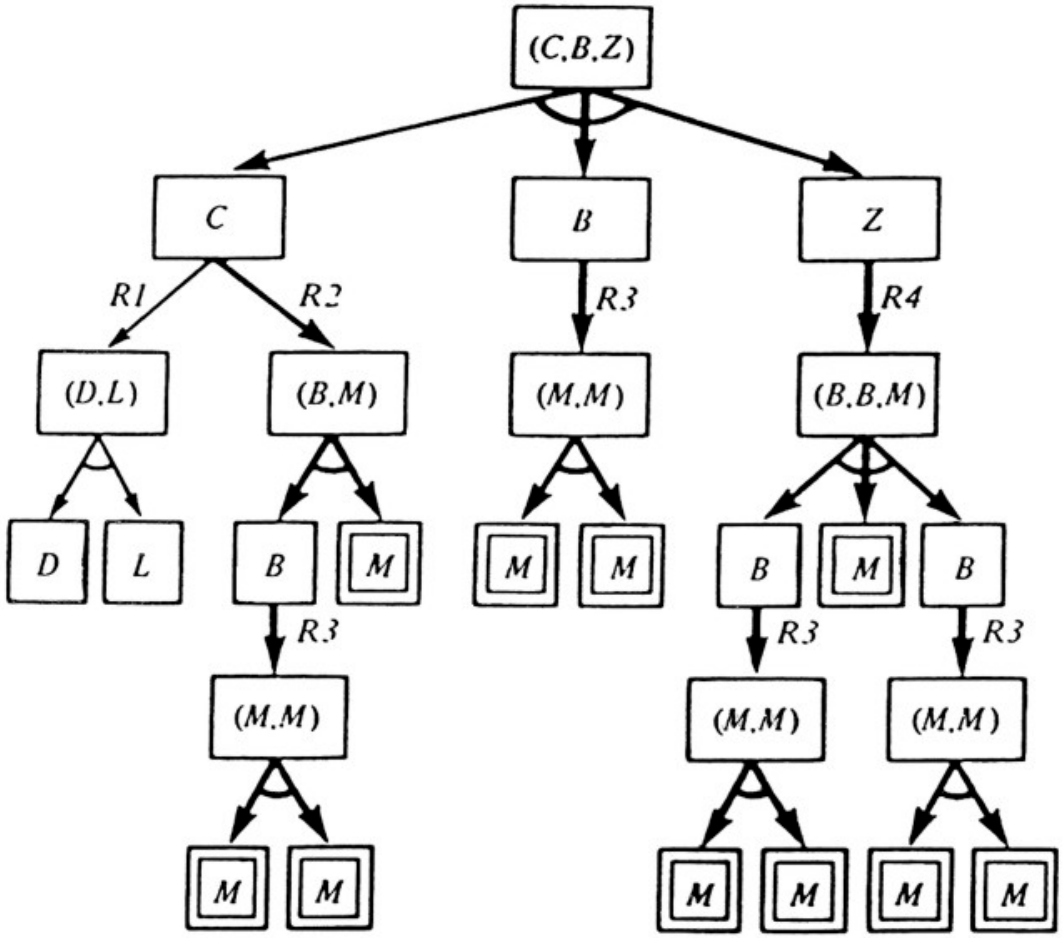
- En el cálculo proposicional, la expresión del grafo Y/O anterior correspondiente sería de la siguiente forma:

$$B \cdot C \rightarrow A;\ G + D \cdot E \rightarrow B;\ E + F \rightarrow C$$

Grafo Y/O

- **Para resolver un grafo Y/O**, cada nodo se resuelve de la siguiente manera:
  - Si es un nodo Y: Resolver todos sus hijos. Combinar la solución y solucionar el nodo. Devolver su solución.
  - Si es un nodo O: Resolver un hijo y ver si devuelve solución. En caso contrario, resolver el siguiente hijo, etc. Cuando ya esté resuelto algún hijo, combinar la solución en el nodo y devolverla.
  - Si es un nodo terminal: Resolver subproblema asociado y devolverla.
- **Mejora:** Para seleccionar el orden de resolución de nodos hijos, se puede utilizar alguna medida de estimación del coste de resolución.

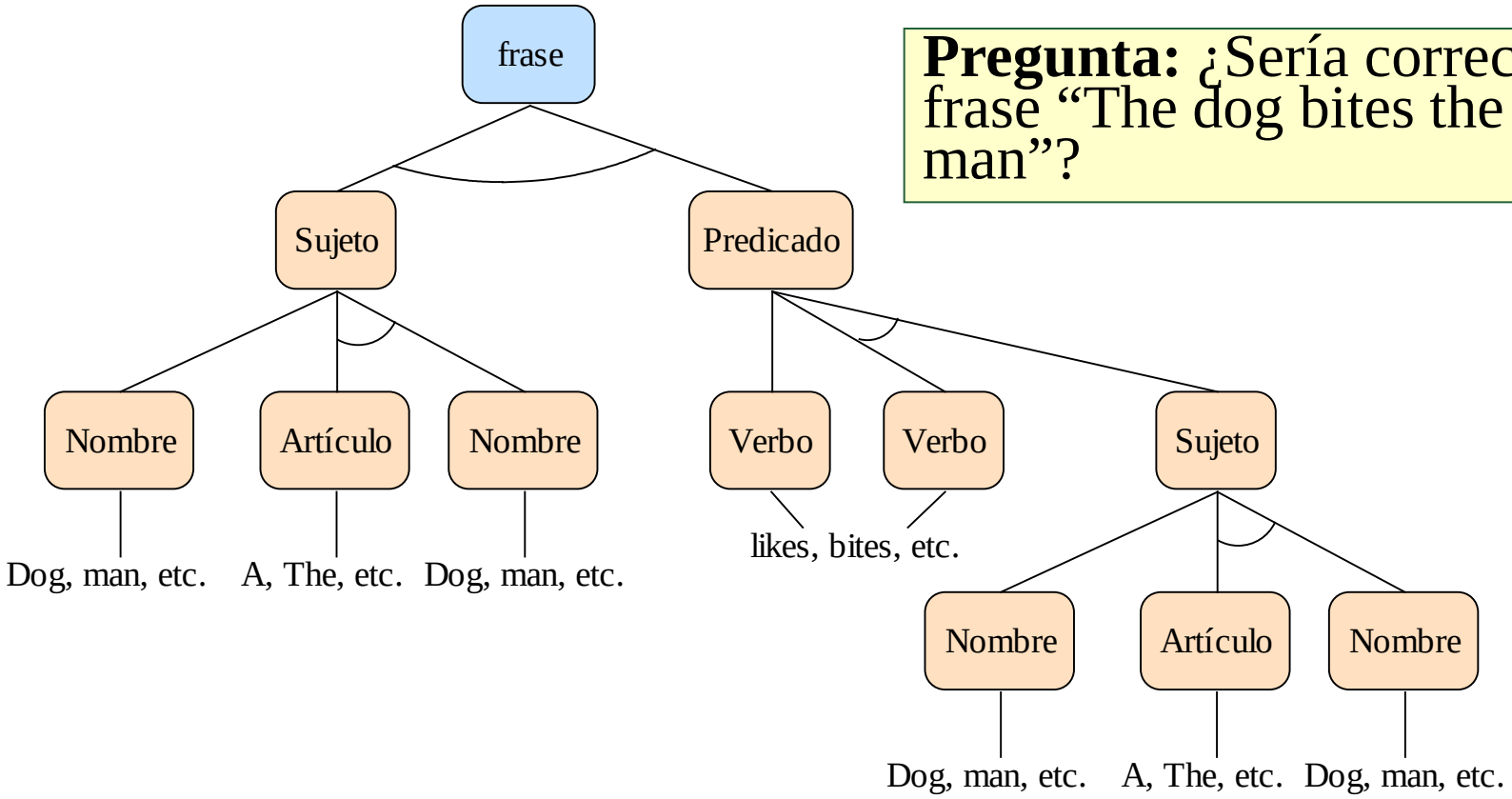
Nueva resolución del problema



Escenario 1: Reconocimiento de frases de lengua inglesa

- Una frase está formada por un sujeto seguido de un predicado.
- El sujeto puede ser un sustantivo o un artículo seguido de un sustantivo.
- El predicado puede ser un verbo, o un verbo seguido de un complemento directo cuya estructura es idéntica a la del sujeto de la frase.

Escenario 1: Reconocimiento de frases de lengua inglesa



Pregunta: ¿Sería correcta la frase "The dog bites the man"?

Escenario 2: Resolución de integrales

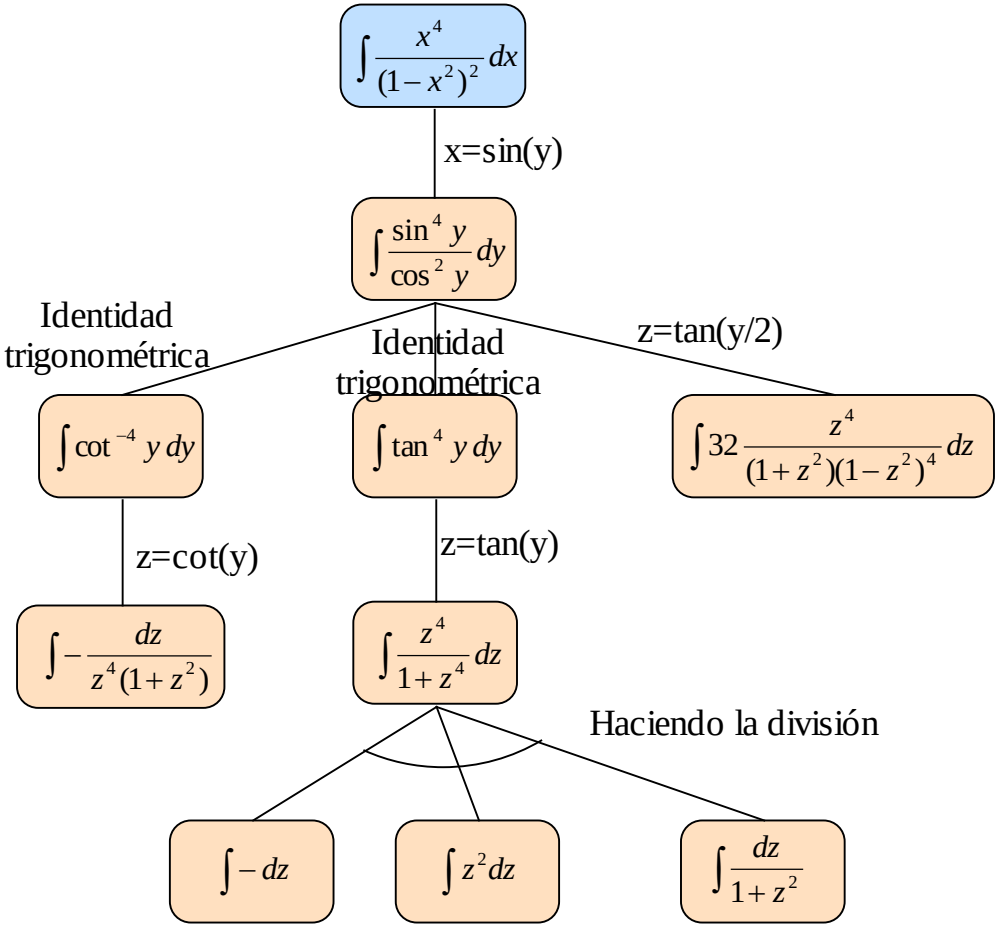
- Para simplificar, supongamos que el computador conoce las transformaciones y técnicas de integración, incluidas en una Base de Datos o de Conocimiento.

- Esta técnica es la que implementa el programa MACSYMA, muy utilizado por matemáticos.

- Supongamos que queremos hacer la siguiente integración:

$$\int \frac{x^4}{(1-x^2)^2}\,dx$$

Escenario 2: Resolución de integrales



Heurísticas

- La(s) **Heurística**(s) son criterios, métodos, o principios para decidir cual, entre una serie de cauces alternativos de acción, promete ser más efectivo a la hora de lograr alguna meta.
- Esto representa un compromiso entre dos exigencias: la necesidad de que tales criterios sean simples y, al mismo tiempo, puedan discriminar correctamente entre buenas y malas opciones.

Heurísticas

- En los problemas de búsqueda que estamos tratando los cauces de acción se formula básicamente buscando el estado al que ir en el siguiente paso del proceso de búsqueda.
- La calidad de un nodo se estima por una función de evaluación heurística f(n) que, en general, es una medida de la distancia entre un nodo dado y el objetivo.

Heuristicas

En general, f(n) depende de:

- la descripción de los nodos,
- la información obtenida hasta ese punto de la búsqueda y
- cualquier conocimiento extra sobre el dominio del problema.

Heuristicas

- La construcción de funciones heurísticas puede considerarse un proceso de descubrimiento.
- Es muy difícil articular el mecanismo por el cual se llega a estas funciones.
- En general: *las heurísticas se descubren a partir de modelos simplificados del dominio del problema.*

Heurísticas

- Los valores de la función heurística son usados para determinar cual operación ejecutar a continuación, típicamente seleccionando la operación que conduce al estado (nodo)con máxima o mínima evaluación.

Heuristicas

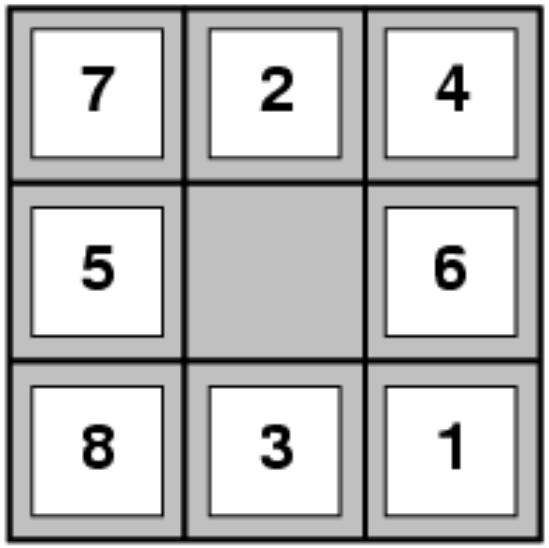
- Un inconveniente de los métodos heurísticos es que en ocasiones no es posible conocer la calidad de la solución, es decir, cuán cerca está el óptimo (x*) la solución heurística encontrada (xheu).

- Si por ejemplo, el problema es de maximización lo único que sabemos es que (xheu) ≤ x*.

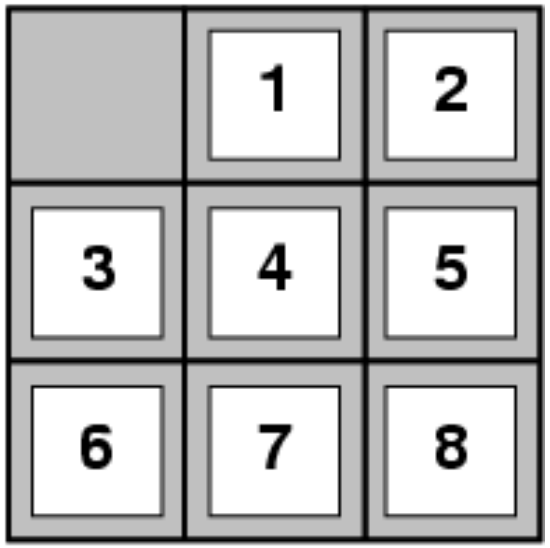
Heuristicas

- En IA basada muy fuertemente en búsqueda, los métodos heurísticos han sido empleados desde el principio.
- Una heurística encapsula el conocimiento específico/experto que se tiene sobre un problema, y sirve de guía para que un algoritmo de búsqueda pueda encontrar una solución.
- Los métodos heurísticos, en general no garantizan la solución óptima, pero que en media produce resultados satisfactorios en la resolución de un problema.
- Eventualmente, una heurística puede devolver soluciones óptimas bajo ciertas condiciones (requiere demostración).

8-puzzle



Start State

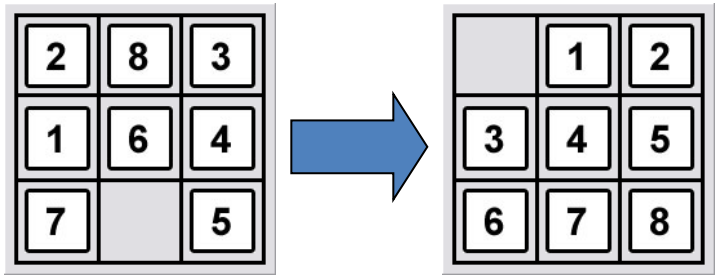


Goal State

8-puzzle

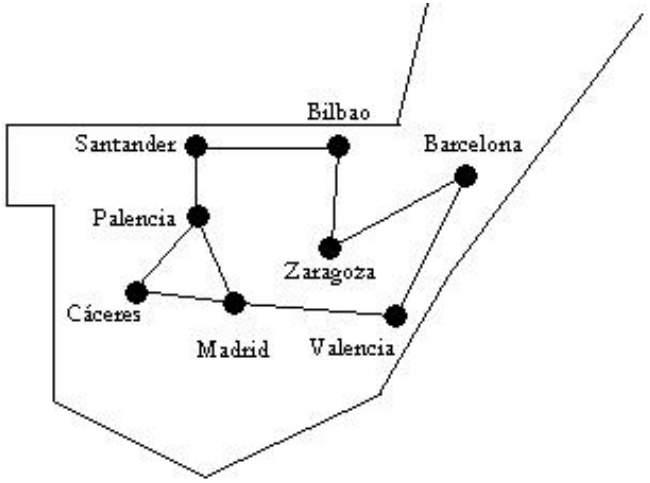
- **En IA, implementaremos heurísticas como funciones que devuelven un valor numérico**, cuya maximización o minimización guiará al proceso de búsqueda a la solución.
- **Ejemplo de heurística en el problema del 8-puzzle:**
  - **f(n)**= nº de fichas descolocadas en comparación con la posición objetivo a alcanzar.
  - **Objetivo:** Minimizar **f**.
  - **n** es un estado (posición de las piezas) del problema.
  - **f(n)** es la ***función heurística***.

f((2, 8, 3, 1, 6, 4, 7, 0, 5))= 9



Un problema de recorridos

- **Una persona quiere viajar de Madrid a Santander considerando el mapa de carreteras de la figura**

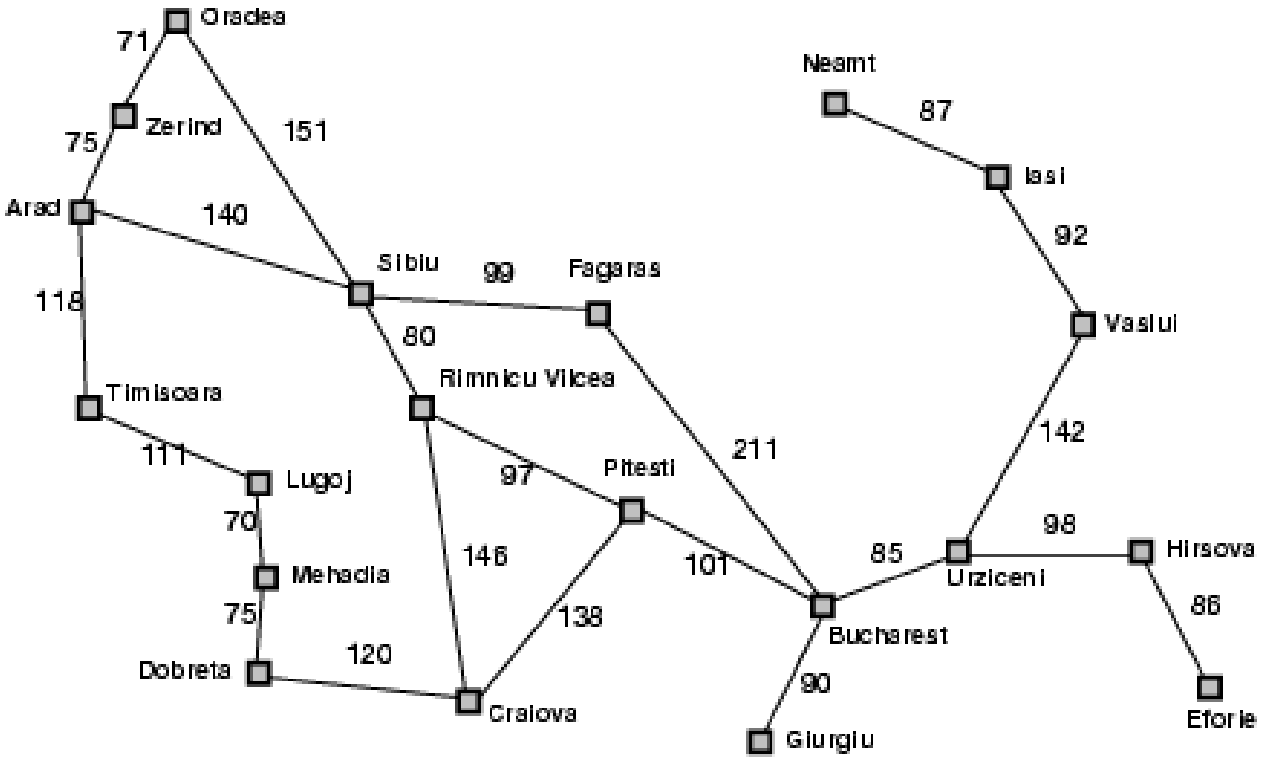


Un problema de recorridos

- Función heurística considerando distancias parciales:
- Distancia(Madrid,Santander)= Distancia Madrid,x) + Distancia(x, Santander).

Distancia (Madrid,Santander)=Distancia(Madrid, Palencia)+Distancia(Palencia, Santander)

Mapa de carreteras: Russell Norvig

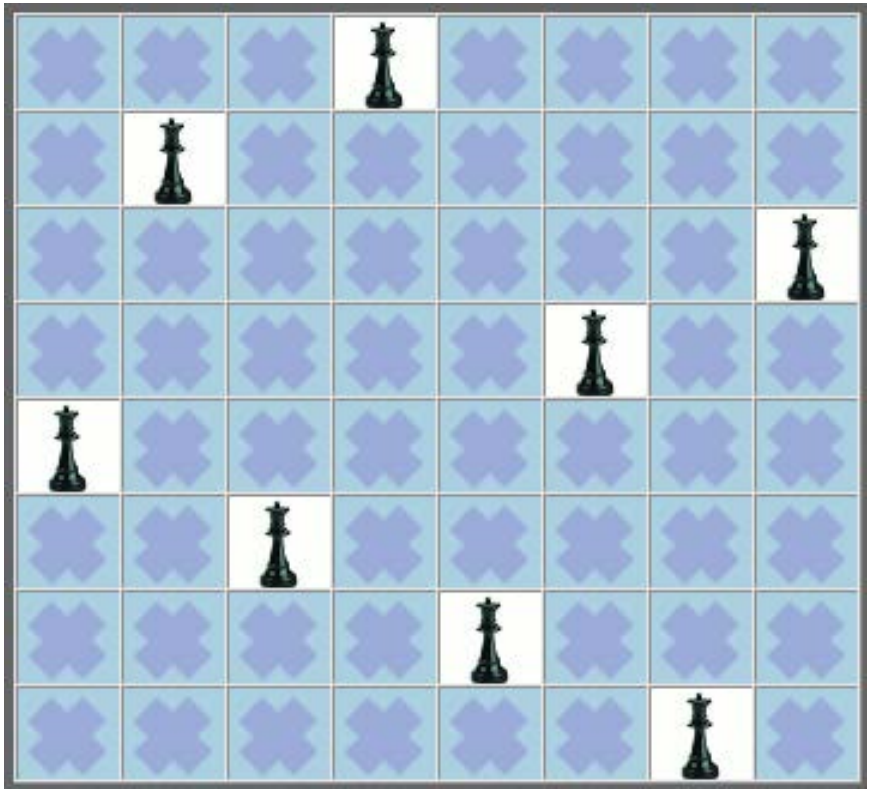


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

El problema de las Ocho Reinas

- El problema consiste en colocar en un tablero de de ajedrez (8×8) ocho reinas sin que puedan atacarse entre ellas. Por ejemplo:

El problema de las Ocho Reinas



El problema de las Ocho Reinas

- El método de resolución es ir colocando una reina cada vez e ir analizando que lugares quedan libres para el resto de las reinas.

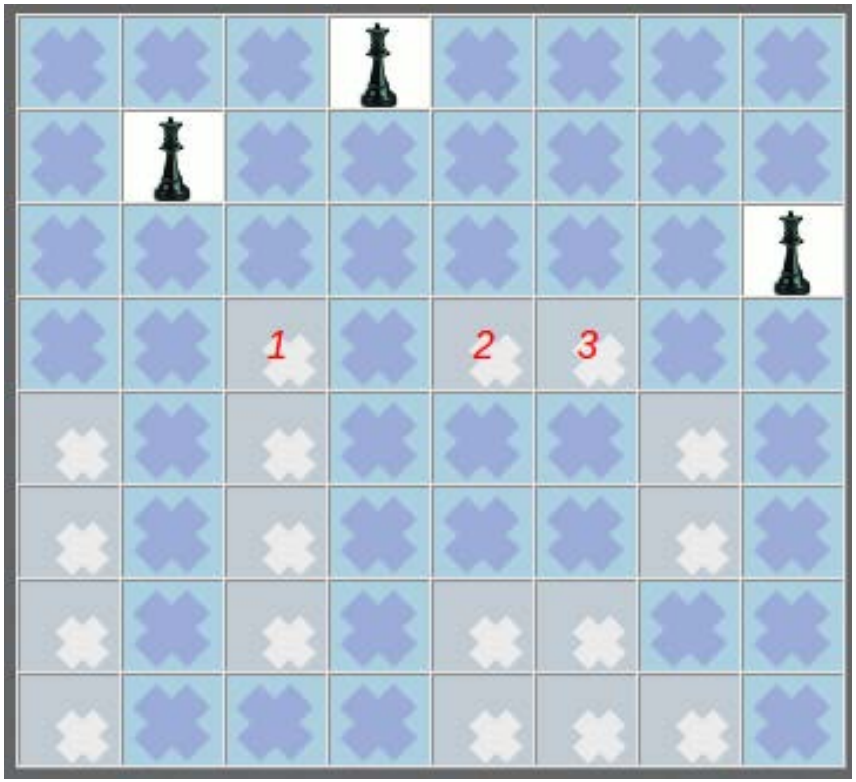
Podemos definir una función heurística **H1** como sigue:

- **H1: Número de celdas sin amenazar por esta celda y las reinas colocadas anteriormente.**

El problema de las Ocho Reinas

- Elegiremos la celda que deje más celdas libres luego de colocar la próxima reina en ese lugar.
- Supongamos que ya colocamos 3 reinas en los lugares 1-4, 2-2, 3-8

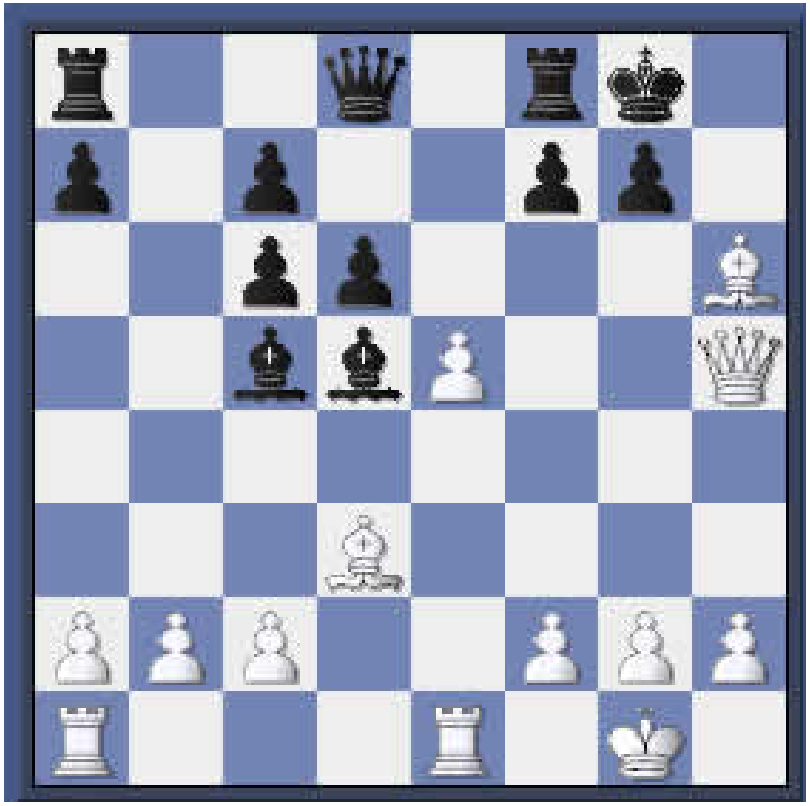
El problema de las Ocho Reinas



El problema de las Ocho Reinas

- En la siguiente fila, nos quedan 3 celdas libres, cuyo valores de H1 son:
- **H1(1)= 8 H1(2)= 9 H1(3)= 10**
- La heurística nos sugiere la celda etiquetada con **3** como el lugar donde colocar la siguiente reina. La colocamos ahí y el proceso se repite.

Ajedrez



Ajedrez

Una heurística conceptual para el ajedrez:

- Valorar cada situación del tablero (p.e. diferencia de piezas con el contrario)
- Determinar que pieza, y a que posición hay que moverla, para mejorar de situación.

En realidad en el ajedrez el problema es mucho más complicado.

Técnicas de búsqueda con Información

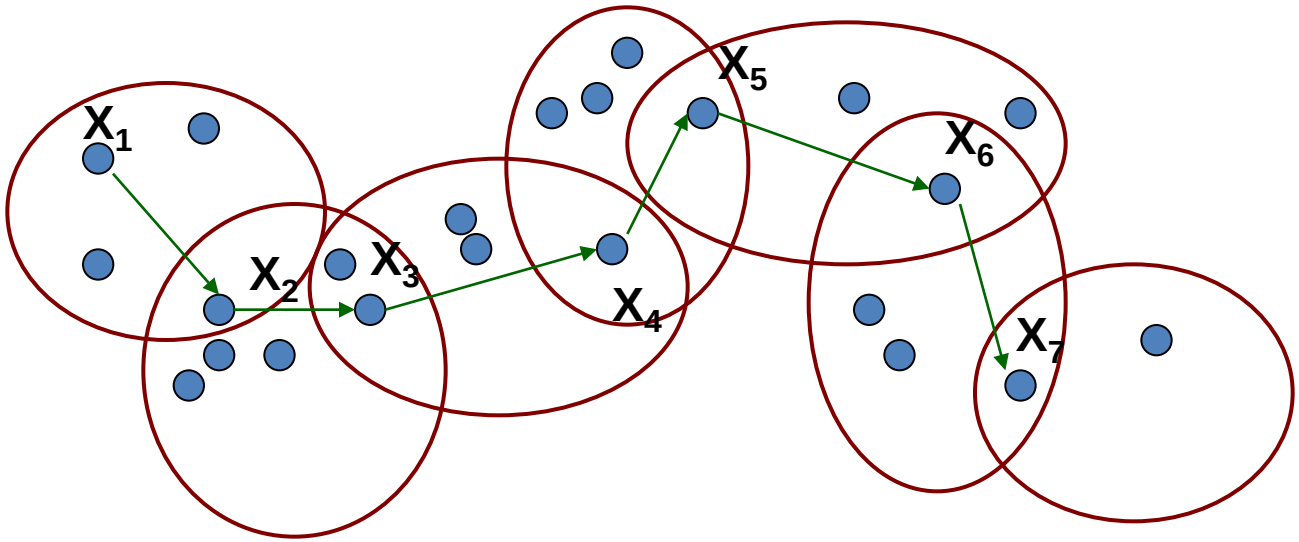
- Métodos de escalada: buscan en un entorno local del nodo en curso.

  Modificación de la búsqueda en profundidad.
- Métodos de Búsqueda “Primero el Mejor”: Avanzan a través del mejor nodo encontrado hasta el momento.

  Modificación de la búsqueda en anchura.

Métodos de escalada

- Si dibujamos las soluciones como puntos en el espacio, una **búsqueda local** consiste en seleccionar la solución mejor en el vecindario de una solución inicial, e ir viajando por las soluciones del espacio hasta encontrar un óptimo (local o global).



Métodos de Búsqueda Local

- Los distintos métodos de búsqueda local se diferencian en:

Como determinar la "vecindad" de cada nodo en estudio.

- Los métodos de escalada que veremos a continuación pertenecen a esta categoría.

Métodos de escalada

- Algoritmo de escalada simple
- Algoritmo de escalada por la máxima pendiente
- Algunas variaciones estocásticas
- Algoritmos genéticos

Algoritmo de escalada simple

Considera que la vecindad de un nodo es un conjunto de hijos obtenidos secuencialmente hasta que aparece uno mejor que el padre (según la función de evaluación).

- Se parte de un nodo inicial.
- Para cada nodo en curso se obtiene la vecindad según el criterio anterior.
- Si no se obtiene un hijo mejor que el padre stop.
- Cuando se obtiene un hijo mejor que el padre el hijo pasa a ser el nodo en curso y el proceso continua.

Algoritmo de escalada simple

- Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como estado actual.
- Repetir hasta que se encuentre una solución o hasta que no queden nuevos operadores que aplicar al estado actual:
  - Seleccionar un operador que no haya sido aplicado con anterioridad al estado actual y aplicarlo para generar un nuevo estado.
  - Evaluar el nuevo estado.
    - Si es un estado objetivo, devolverlo y terminar.
    - Si no es un estado objetivo, pero es mejor que el estado actual, convertirlo en el estado actual.
    - Si no es mejor que el estado actual, continuar con el bucle.

Algoritmo de escalada por la máxima pendiente (hill climbing)

- Considera que la vecindad de un nodo es el conjunto de todos sus hijos obtenidos secuencialmente.
- Se parte de un nodo inicial.
- Para cada nodo en curso se obtiene la vecindad según el criterio anterior.
- Se selecciona el hijo mejor que pasa a ser el nodo en curso y el proceso continua.
- Si no se obtiene un hijo mejor que el padre stop.

Algoritmo de escalada por la máxima pendiente (hill climbing)

1.- Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como estado actual.

2.- Partir de la solución actual. Expandirla.

- 2.1- Buscar el hijo de mejor calidad según la función de evaluación.
- 2.2 - Si es “mejor” que la solución actual entonces sustituir la solución actual por dicho nodo.

- 2-3- volver a 2
- 2.4- Si no se encuentra un hijo mejor que el padre entonces parar.

Características

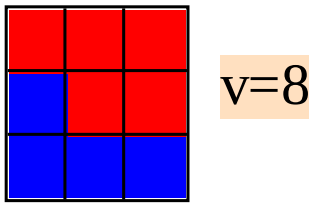
- **Completitud**: no tiene porque encontrar la solución
- **Admisibilidad**: no siendo completo, aun menos será admisible
- **Eficiencia**: rápido y útil si la función es monótona (de)creciente

Ejemplo

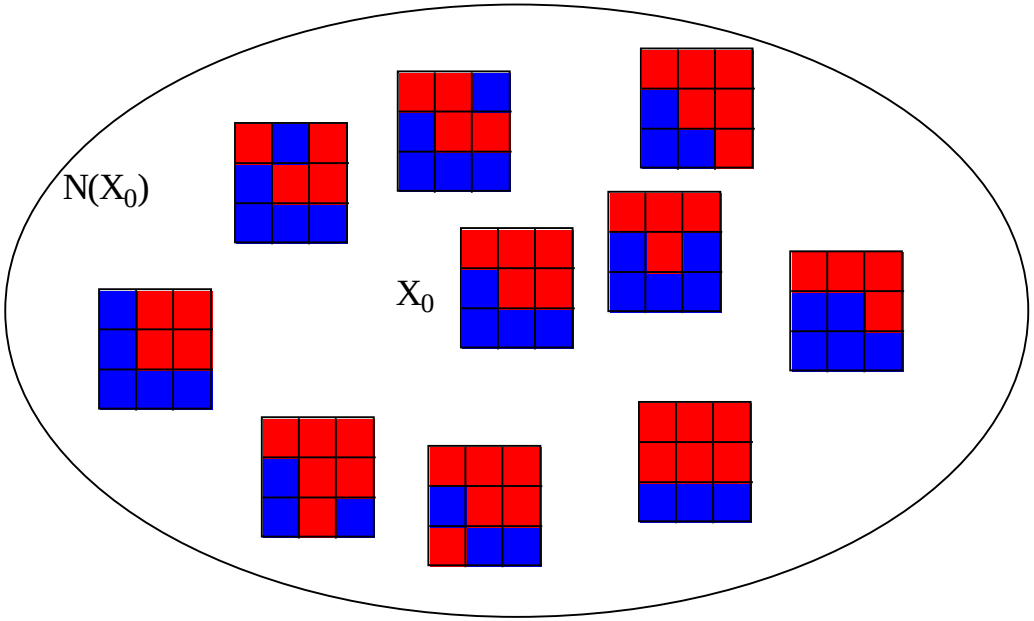
- **Ejemplo:** Colorear una matriz **M*N** con **n** colores, de modo que cada celda tenga el mínimo número de celdas adyacentes del mismo color. Asumimos que las celdas adyacentes son las que se encuentran una casilla hacia arriba, abajo, izquierda o derecha de la casilla considerada.
  - **Función objetivo:** Minimizar la suma del número de pares de casillas adyacentes del mismo color.
  - **Definición del entorno:** El vecindario de una solución estará formado por aquellas soluciones cuyos colores varíen en una única posición de la solución dada.

Ejemplo

- **Ejemplo:**
  - **Solución inicial:** Generada de forma aleatoria. Supongamos que se ha generado la siguiente para una matriz de **3*3**, a rellenar con **2 colores rojo y azul**, con valor de función objetivo ***v=8*:**

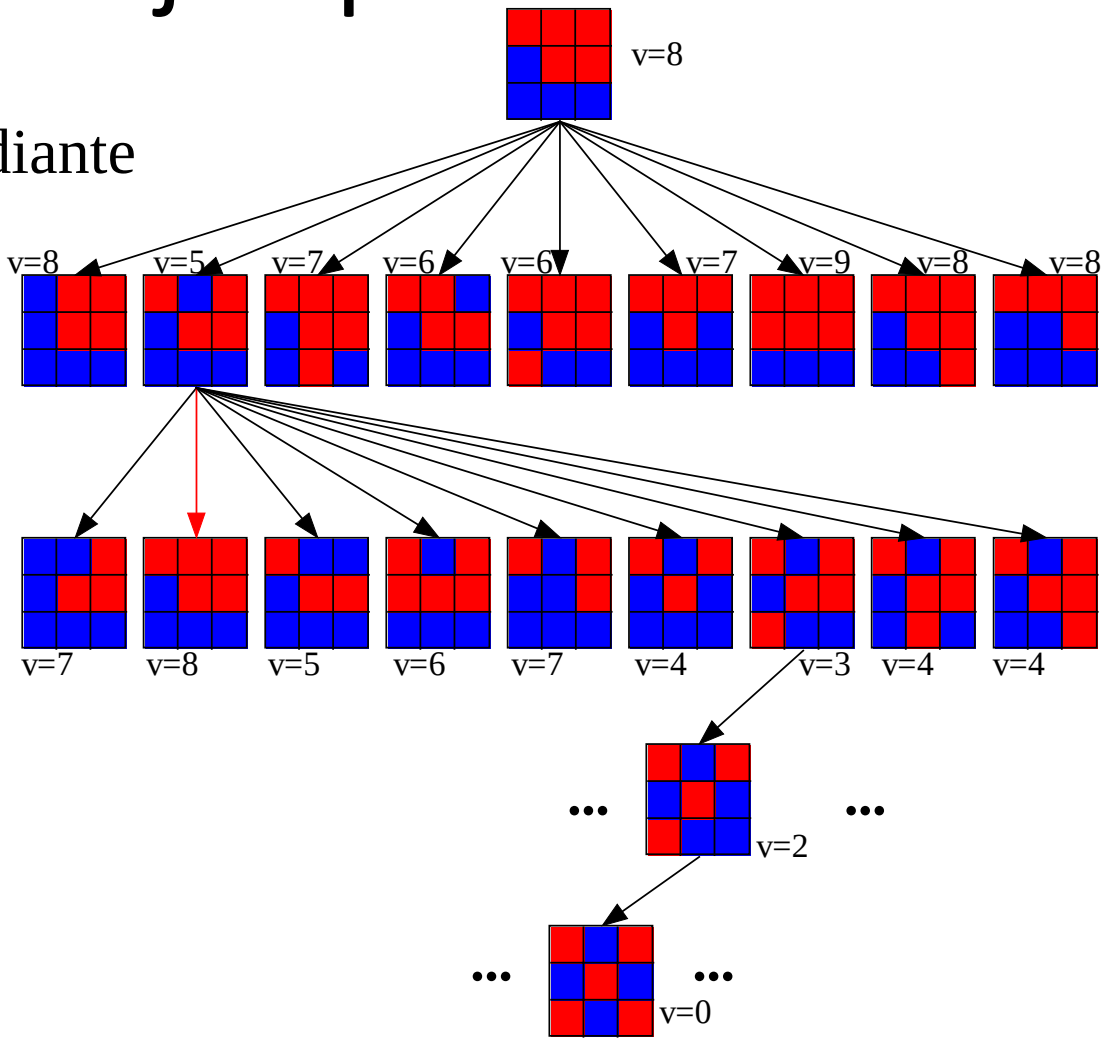


v=8

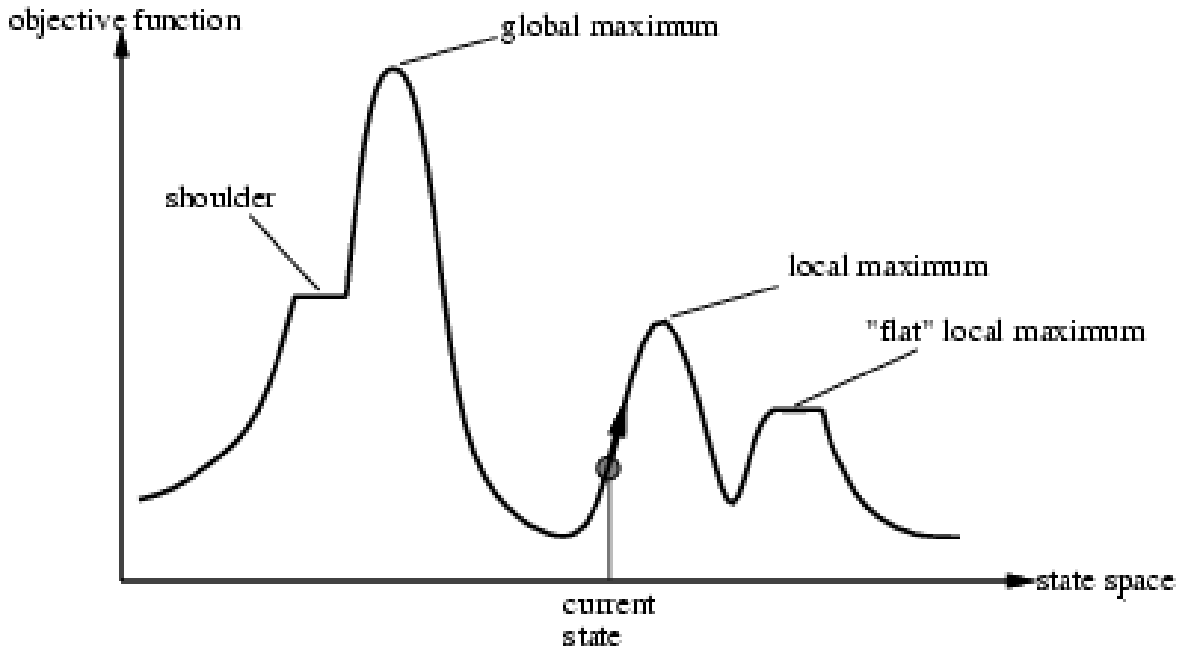


Ejemplo

- **Ejemplo:** Solución al problema anterior mediante ascensión de colinas.



Métodos de escalada



Algunas variaciones estocásticas

- Algoritmo de escalada estocástico
- Algoritmo de escalada de primera opción
- Algoritmo de escalada de reinicio aleatorio
- Enfriamiento simulado
- Algoritmos genéticos

Variantes del método de escalada

- Escalada estocástica
  - Escoge aleatoriamente entre los sucesores con mejor valoración que el estado actual.

- Escalada de primera opción
  - Se generan aleatoriamente sucesores, escogiendo el primero con mejor valoración que el estado actual

Variantes del método de escalada

- Escalada con reinicio aleatorio
  - Se repite varias veces la búsqueda, partiendo cada vez de un estado inicial distinto, generado aleatoriamente
  - "si no te sale a la primera, inténtalo otra vez"
  - Si la probabilidad de éxito de una búsqueda individual es p, entonces el número esperado de reinicios es 1/p

Algoritmo de Enfriamiento Simulado

- Un modo de evitar que la búsqueda local finalice en óptimos locales, hecho que suele ocurrir con los algoritmos tradicionales de búsqueda local, es permitir que algunos movimientos sean hacia soluciones peores
- Pero si la búsqueda está avanzando realmente hacia una buena solución, estos movimientos “de escape de óptimos locales” deben realizarse de un modo controlado

Algoritmo de Enfriamiento Simulado

- En el caso del Enfriamiento Simulado (ES), esto se realiza controlando la frecuencia de los movimientos de escape mediante una función de probabilidad que hará disminuir la probabilidad de estos movimientos hacia soluciones peores conforme avanza la búsqueda y por tanto estamos más cerca, previsiblemente, del óptimo local
- Se basa en principios de Termodinámica.
- Así, se aplica la filosofía habitual de búsqueda de diversificar al principio e intensificar al final

Algoritmo de enfriamiento simulado

- **Analogía entre el proceso de enfriamiento y el algoritmo de enfriamiento simulado:**
  - Los **estados** por los que pasa el sistema físico de partículas equivalen a las **soluciones factibles** del algoritmo.
  - La **energía E del estado actual** del sistema es el valor de la **función objetivo de la solución actual**. Ambos tienen que minimizarse.
  - Un **cambio de estado** en el sistema equivale a **explorar el entorno de una solución y viajar a una solución vecina**.
  - El **estado final estable** (congelado) es la **solución final** del algoritmo.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Se parte de una Temperatura inicial T_i, y de un estado (solución) **X**, con una energía (coste, bondad) asociada **Bondad(X)**

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

El sistema se modelará hasta que se alcance una temperatura final deseada.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

En cada estado (solución), se exploran sus estados cercanos (vecindario). Si el problema es muy grande, la exploración se limita a **N(T)** vecinos.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Se genera un vecino. Si es mejor, se acepta directamente. Si no se genera q mediante U(0,1). Si q es menor que e^(-F/T), **también se acepta**.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

La temperatura se reduce, indicando que nos acercamos al estado estable.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Se devuelve la mejor solución X que se haya encontrado en todo el proceso de enfriamiento simulado

Algoritmo de enfriamiento simulado

- La **solución inicial** se puede generar de forma aleatoria, por conocimiento experto, o por medio de otras técnicas.
- La **actualización de temperatura** también es heurístico, y hay varios métodos:
  - $T \leftarrow \alpha \cdot T$, con α en (0,1)
  - $T \leftarrow 1/(1+k)$, con k= número de iteraciones del algoritmo hasta el momento, etc.
- **Número de vecinos a generar**: Fijo N(T)= cte, dependiente de la temperatura N(T)= f(T), etc.

Algoritmo de enfriamiento simulado

- Tanto la temperatura inicial como la temperatura final $\mathbf{T_i}$ y $\mathbf{T_f}$ son parámetros de entrada al algoritmo.
- Es difícil asignar un valor concreto a $\mathbf{T_f}$, por lo que la condición de parada se suele sustituir por un número específico de iteraciones a realizar.

- **Ventajas:**
  - Al ser un método probabilístico, tiene capacidad para salir de óptimos locales.
  - Es eficiente.
  - Es fácil de implementar.

Algoritmo de enfriamiento simulado

- **Inconvenientes:**
  - Encontrar la temperatura inicial $\mathbf{T_i}$, el método de actualización de temperatura $\boldsymbol{\alpha}$, el número de vecinos a generar en cada estado y el número de iteraciones óptimo es una tarea que requiere de **muchas pruebas de ensayo y error** hasta que ajustamos los parámetros óptimos.

- Pese a todo, el algoritmo puede proporcionar soluciones mucho mejores que utilizando algoritmos no probabilísticos.

Procedimiento Primero el Mejor
Variante del algoritmo de Dijkstra

Usaremos dos listas de nodos (ABIERTOS Y CERRADOS)

- **Abiertos:** nodos que se han generado y a los que se les ha aplicado la función heurística, pero que aún no han sido examinados (es decir, no se han generado sus sucesores).
- Esta lista se ordena según el valor de la heurística.

Procedimiento Primero el Mejor Variante del algoritmo de Dijkstra

- **Cerrados:** nodos que ya se han examinado. Es necesaria para ver si cuando se genera un nuevo nodo ya ha sido generado con anterioridad.

Procedimiento Primero el Mejor
Variante del algoritmo de Dijkstra

- Crear grafo de búsqueda G, con el nodo inicial, I (Estado-inicial)
- ABIERTOS=I , CERRADOS=Vacío, EXITO=Falso
- Hasta que ABIERTO esté vacío O ÉXITO
- Quitar el primer nodo de ABIERTOS, N y meterlo en CERRADOS

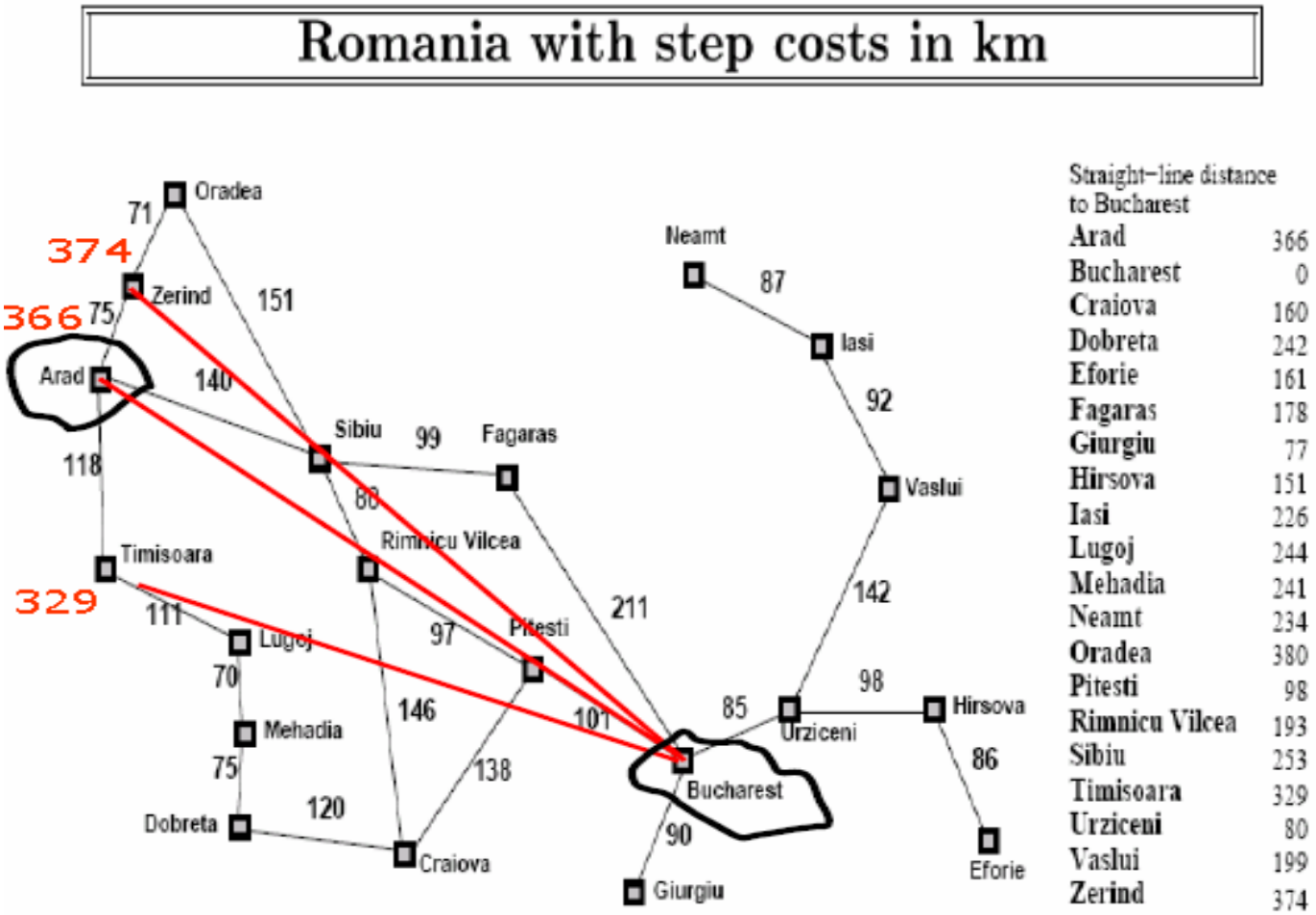
Procedimiento Primero el Mejor

- SI N es Estado-final ENTONCES EXITO=Verdadero
- SI NO Expandir N, generando el conjunto S de sucesores de N, que no son antecesores de N en el grafo
- Generar un nodo en G por cada s de S
- Establecer un puntero a N desde aquellos s de S que no estuvieran ya en G
- Añadirlos a ABIERTOS

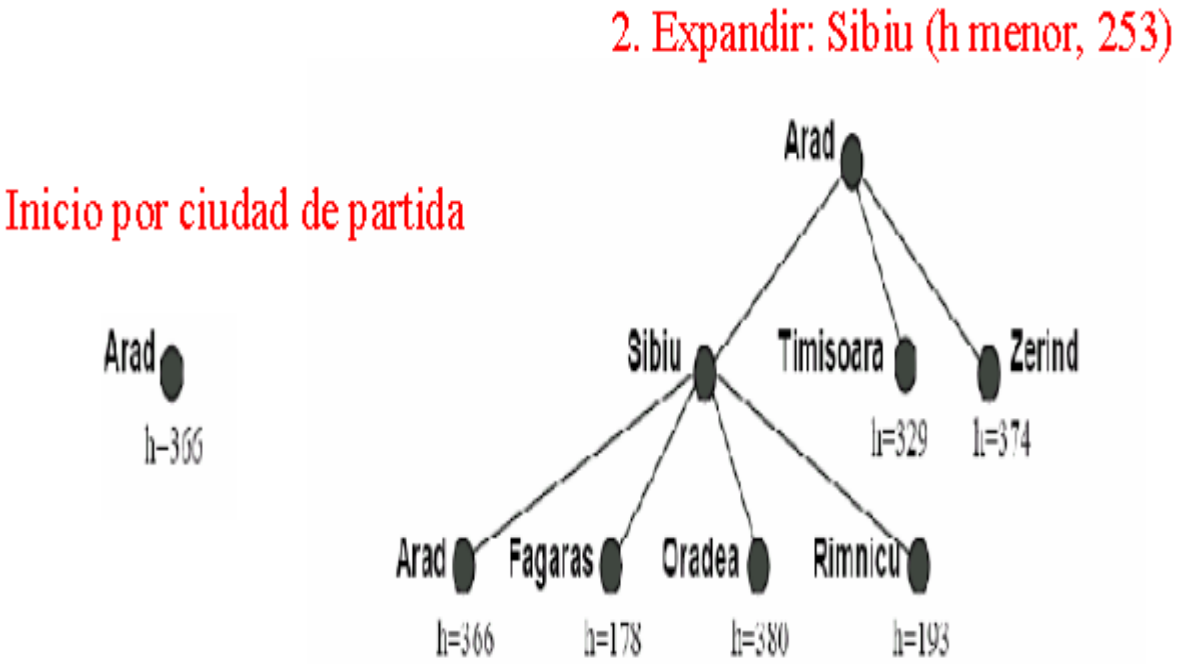
Procedimiento Primero el Mejor

- Para cada s de S que estuviera ya en ABIERTOS o CERRADOS decidir si redirigir sus punteros hacia N
- Para cada s de S que estuviera ya en CERRADOS decidir si redirigir los punteros de los nodos en sus subárboles
- Reordenar ABIERTOS según f (n)
- Si EXITO Entonces Solución=camino desde I a N a través de los punteros de G
- Si no Solución=Fracaso

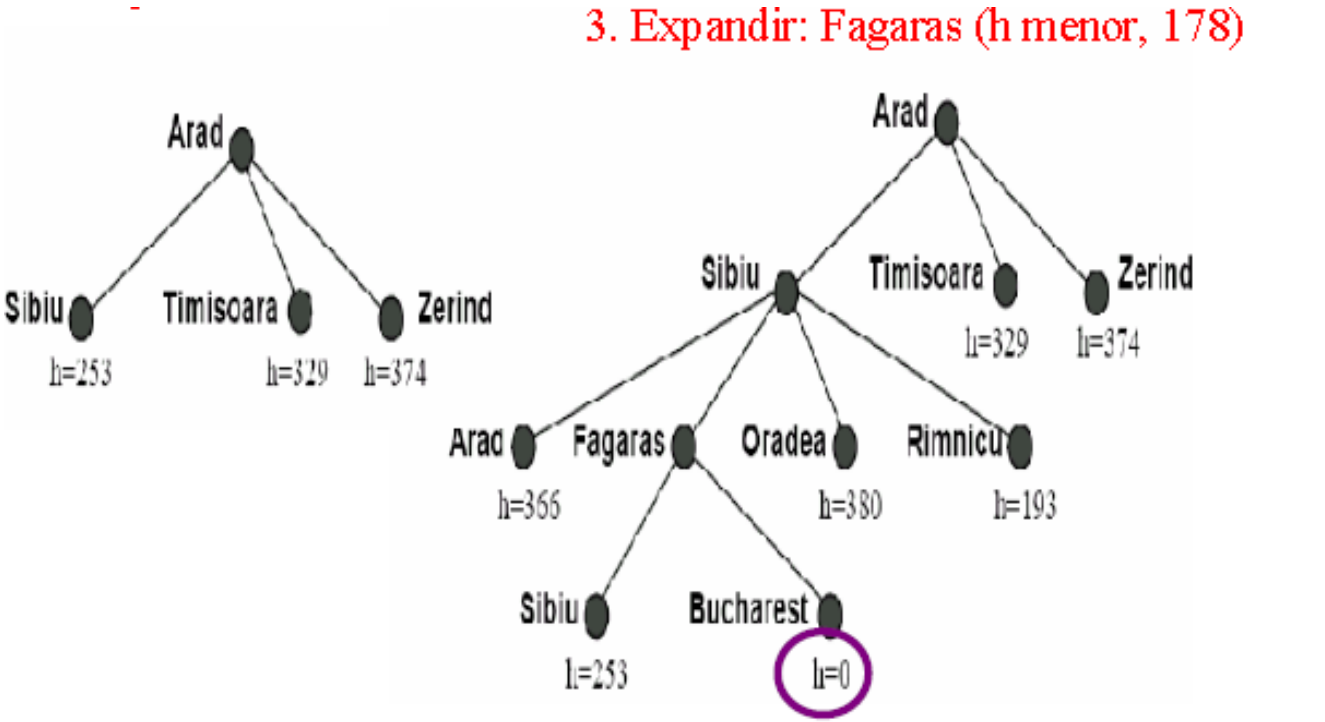
Primero el Mejor (voraz)



Primero el Mejor (voraz)



Primero el Mejor (voraz)



Búsqueda primero el mejor

- Algoritmo A*
- Búsqueda dirigida

Algoritmo A*

Se trata de un algoritmo Mejor primero donde f(n) es la suma de dos componentes:

- Una medirá la distancia actual desde el nodo origen hasta el nodo a etiquetar,
- Otra expresará la distancia estimada desde este nodo a etiquetar hasta el nodo destino.

Algoritmo A*

Un nodo n tendría

- **f(n)= g(n) + h(n)**
- Donde:
- -**g(n)** indica la distancia del mejor camino hasta el momento desde el nodo inicial **I** al n.
- -**h(n)** expresa la distancia estimada desde el nodo n hasta el nodo objetivo **O**.

Algoritmo A*

- En cada paso se selecciona el nodo más prometedor que se haya generado hasta ese momento (función heurística).
- A continuación se expande el nodo elegido generando todos sus sucesores.
- Si alguna de ellos es meta el proceso acaba. Si no continúo con el algoritmo.

Algoritmo A*

- Es una búsqueda en anchura. Si en una rama por la que estoy explorando no aparece la solución, la rama puede parecer menos prometedora que otras por encima de ella y que se habían ignorado.
- Podemos pues abandonar la rama anterior y explorar la nueva.

Algoritmo A*

- Sin embargo al vieja rama no se olvida. Su último nodo se almacena en el conjunto de nodos generados pero aún sin expandir.

Algoritmo A*

1.- Crear un grafo de búsqueda ***G***. Inicializar ***G*** con **I.**

2.- Crear ABIERTOS con **I**, crear CERRADOS vacío.

3.- Si ABIERTOS está vacía terminar con FALLO.

4.- Tomar primero nodo, **N**, de ABIERTOS e insertarlo en CERRADOS.

5.- Si **N** = **O** entonces ÉXITO. Devolver **N** y el camino de **I** a **N** (usando el árbol de búsqueda dado por los punteros que se construyen en el paso 7).

6.- Expandir **N** y crear el conjunto ***M*** con todos sus sucesores. Eliminar de ***M*** aquellos nodos que sean ancestros de **N** en ***G***. Añadir ***M*** a los sucesores de **N** en ***G***.

Algoritmo A*

7.- Establecer un puntero a N desde aquellos nodos de M que no estén ni en ABIERTOS ni en CERRADOS (no han sido visitados en G). Insertar estos elementos en ABIERTOS.

Para cada nodo de M que esté en ABIERTOS o en CERRADOS modificar el puntero para que este apunte a N siempre que el mejor camino encontrados hasta el momento hacia dicho nodo pase por N.

Para cada nodo de M que ya esté en CERRADOS, modificar los punteros de cada uno de sus descendientes de modo que apunten hacia atrás a los mejores caminos encontrados hasta el momento a esos descendientes.

Algoritmo A*

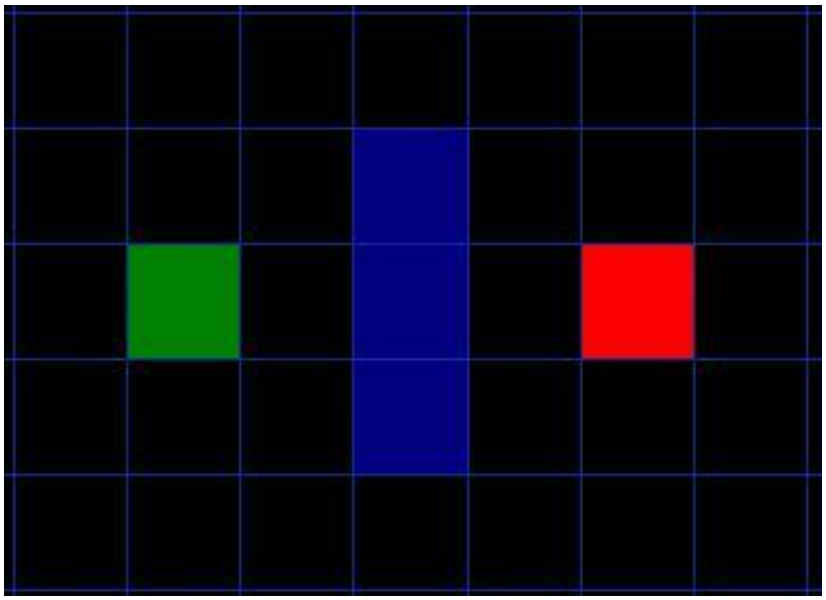
8.- Reordenar ABIERTOS según f(n) de menor a mayor. En caso de empate emplear el criterio de profundidad en el arbol de búsqueda.

9.- Volver al paso 3.

Un ejemplo

http://www.policyalmanac.org/games/aStarTutorial_es.htm
http://www.policyalmanac.org/games/aStarTutorial.htm

Encontrar el camino mas corto entre el cuadrado verde y el rojo (sus Centros)



Un ejemplo

- Un camino puede usar cuadros en horizontal/vertical o en diagonal.
- El costo de recorrer un cuadrado en horizontal/vertical lo estimaremos en 10 unidades.
- El costo de recorrer un cuadrado en diagonal lo estimaremos en 14 (aproximación de la diagonal de un cuadrado de lado 10).

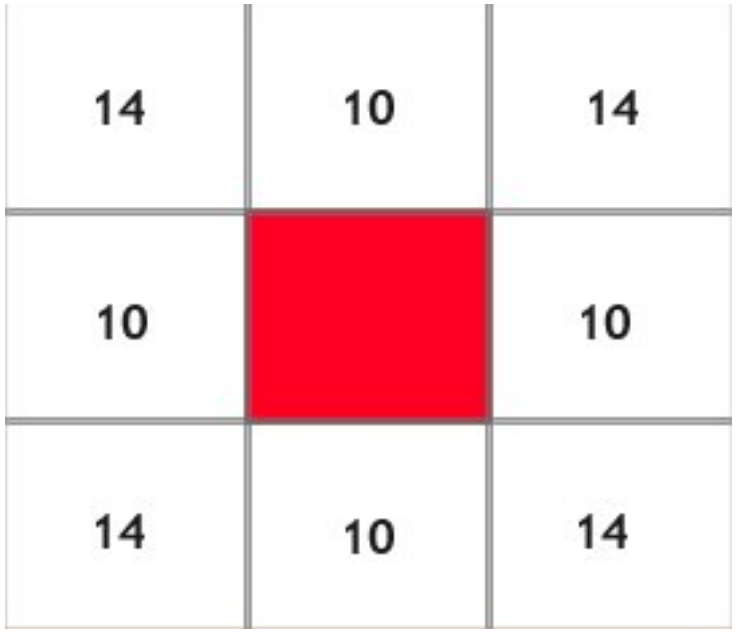
Un ejemplo

- Entonces para cualquier cuadrado C

$$g(C) = 10*n + 14*m$$

siendo n el número de cuadrados recorridos en horizontal/vertical y m el número de cuadrados recorridos en diagonal para llegar del cuadrado inicial al cuadrado C

Un ejemplo



Un ejemplo

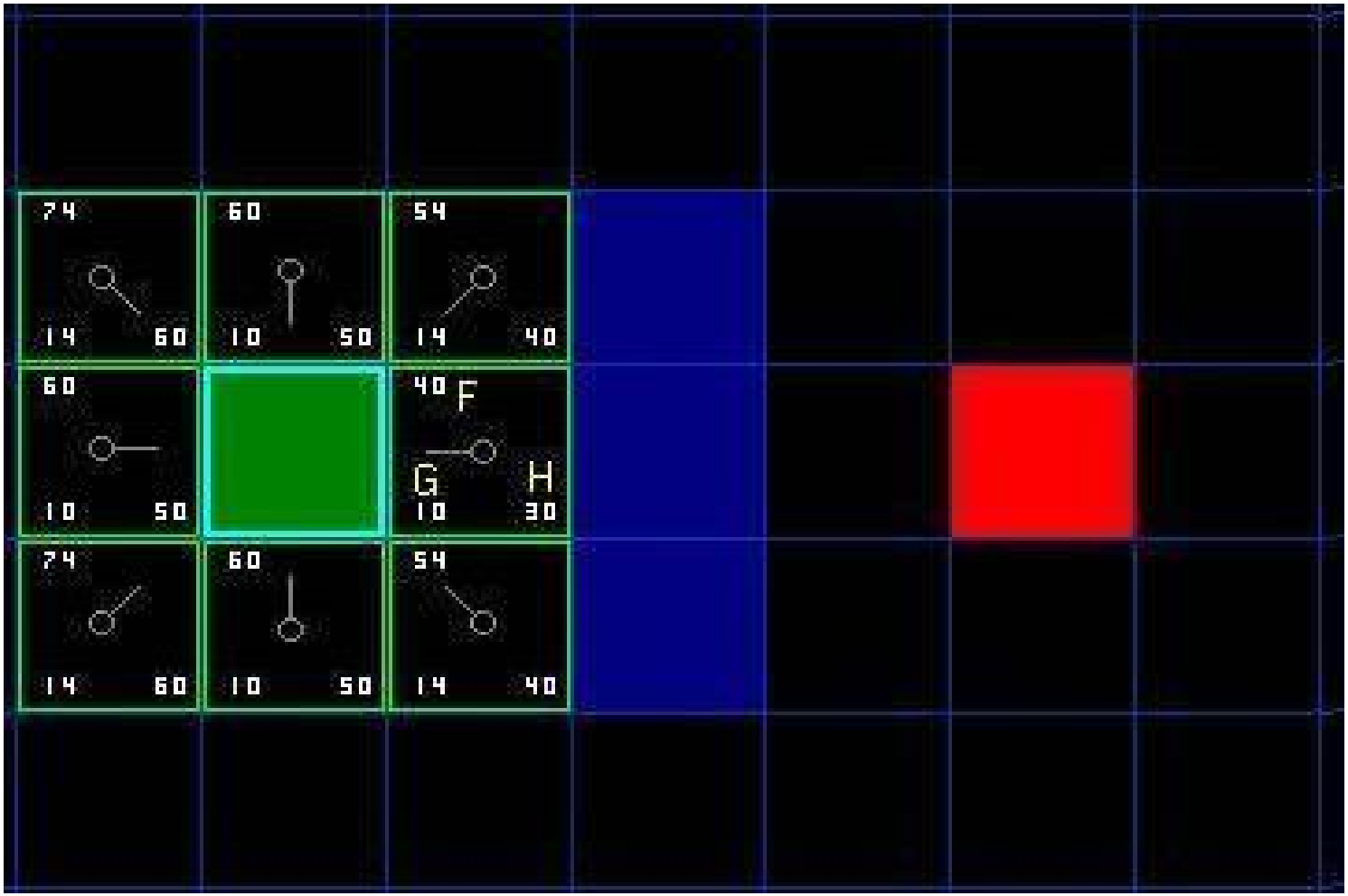
Para la heurística h emplearemos la distancia de Manhattan sin tener en cuenta los obstáculos.

Entonces para cualquier cuadrado C

$$h(C) = 10*k$$

Donde k es el número de cuadrados recorridos en horizontal/vertical que separan C del cuadrado destino (distancia Manhattan)

Un ejemplo


74
60
54
14
60
10
50
14
40
60
40
F
G
H
10
50
10
30
74
60
54
14
60
10
50
14
40

Un ejemplo

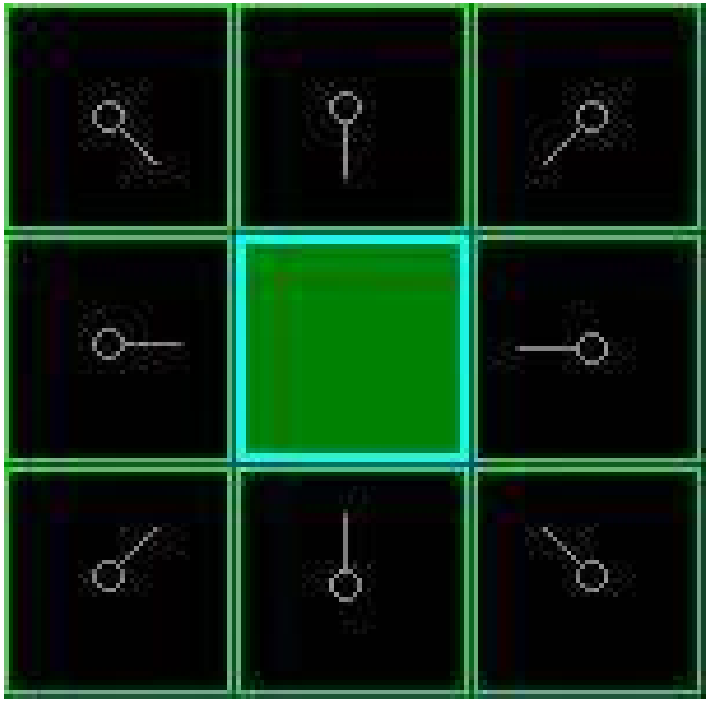
- **1)** Empezar en el punto inicial A y añadirlo a la lista ABIERTOS (cuadrados a tener en cuenta). Básicamente es una lista de los cuadrados que necesitan ser comprobados.
- **2)** Mirar todos los cuadrados alcanzables o transitables adyacentes al punto de inicio, ignorando cuadrados "intransitables". Añadirlos también a ABIERTOS.

Un ejemplo

- Por cada uno de esos cuadrados, guardar el punto A como su “cuadrado padre”. El cuadrado padre es muy importante para trazar nuestro camino.
- **3)** Sacar el cuadro inicial A de ABIERTOS y colocarlo en CERRADOS (cuadrados que , por ahora, no necesitan ser mirados de nuevo).

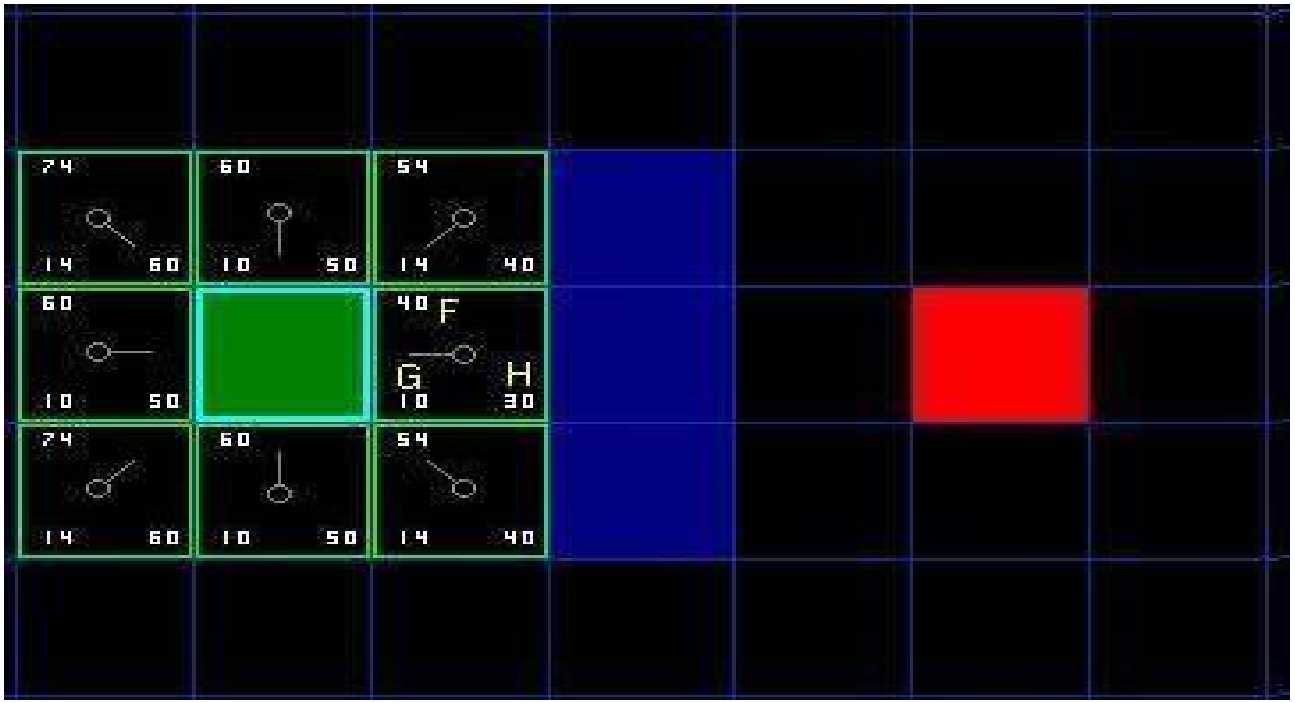
Un ejemplo

- ABIERTOS: recuadrado en verde
- CERRADOS: recuadrado en azul

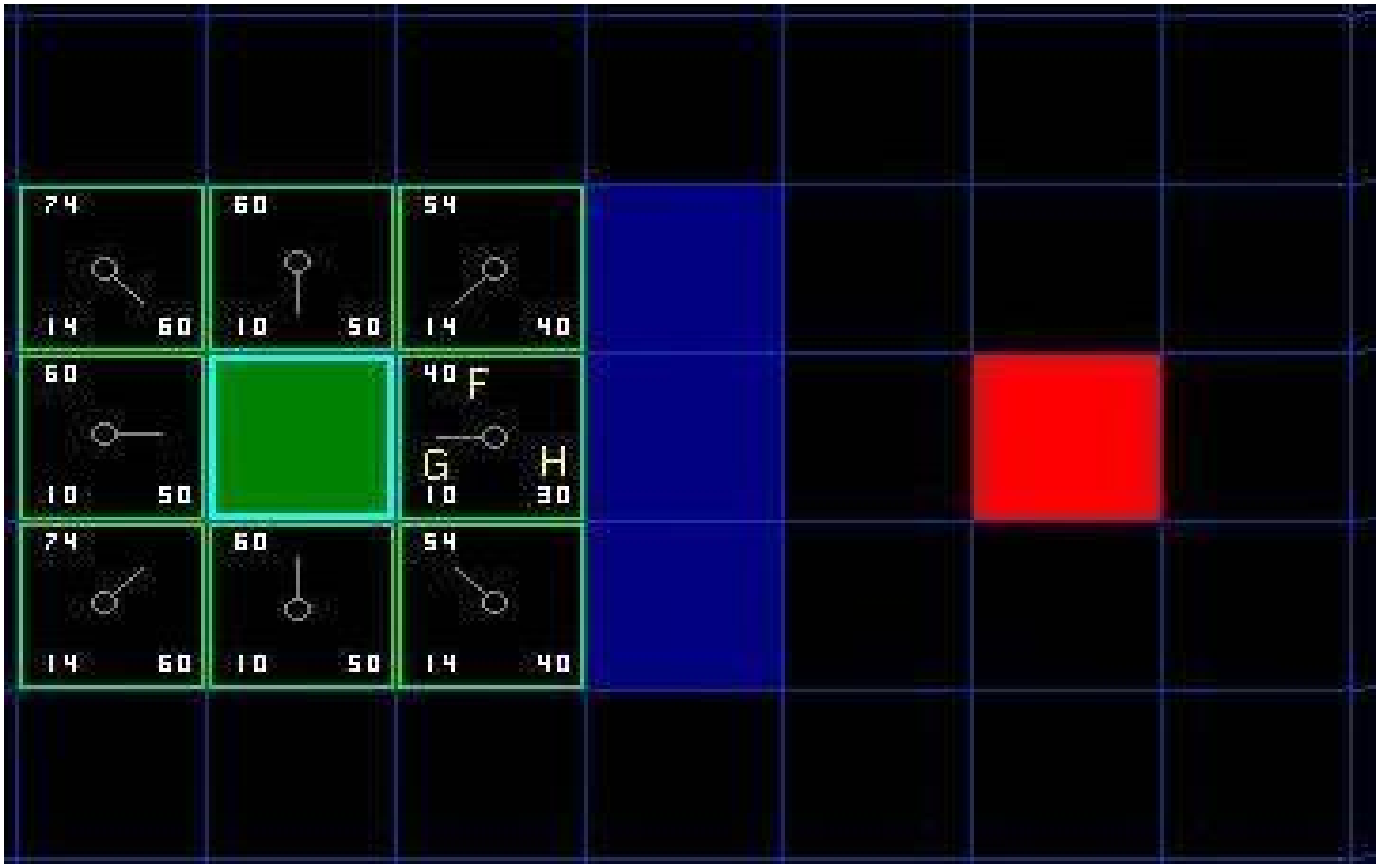


Un ejemplo

- Valorar los cuadrados de la lista ABIERTOS de acuerdo con f(.)

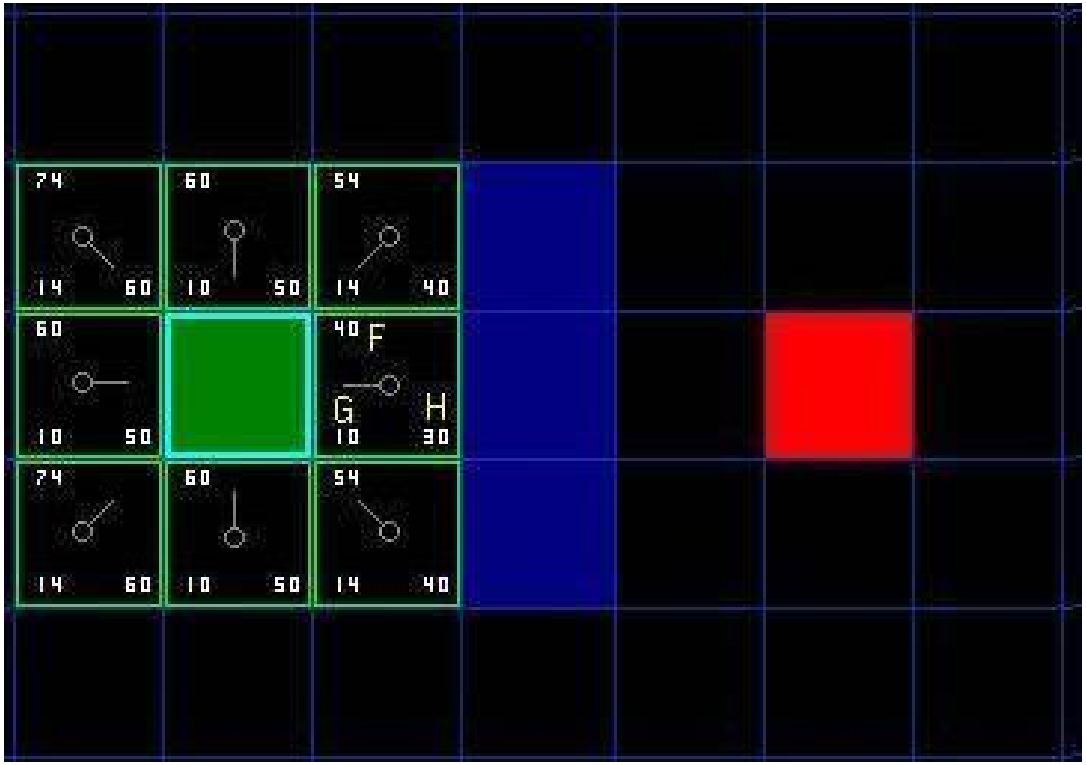


Un ejemplo


74
14
60
60
10
50
54
14
40
60
10
50
40
F
G
10
H
30
74
14
60
60
10
50
54
14
40

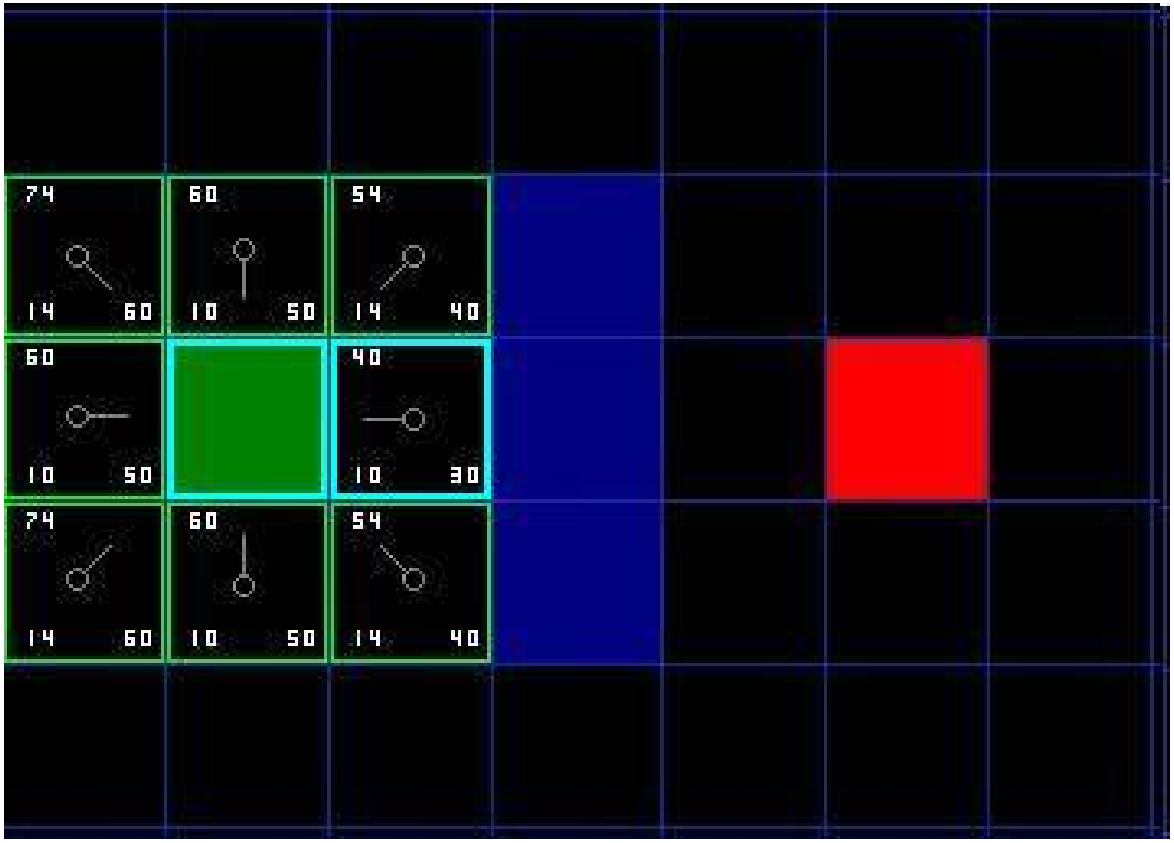
Un ejemplo

- De los 9 cuadros iniciales, hay 8 en ABIERTOS después de que el cuadrado inicial fuera incluido en CERRADOS.
- De estos, el que tiene el coste F más bajo es el INMEDIATO a la derecha del cuadro inicial, con una F de 40.


74
60
54
14
60
10
50
14
40
60
40
F
G
H
10
50
10
30
74
60
54
14
60
10
50
14
40

Un ejemplo

- Seleccionamos este cuadrado como nuestro siguiente cuadrado.
- Lo incluimos en la lista CERRADOS (resaltado en azul)


74
14
60
60
10
50
54
14
40
60
10
50
40
10
30
74
14
60
60
10
50
54
14
40

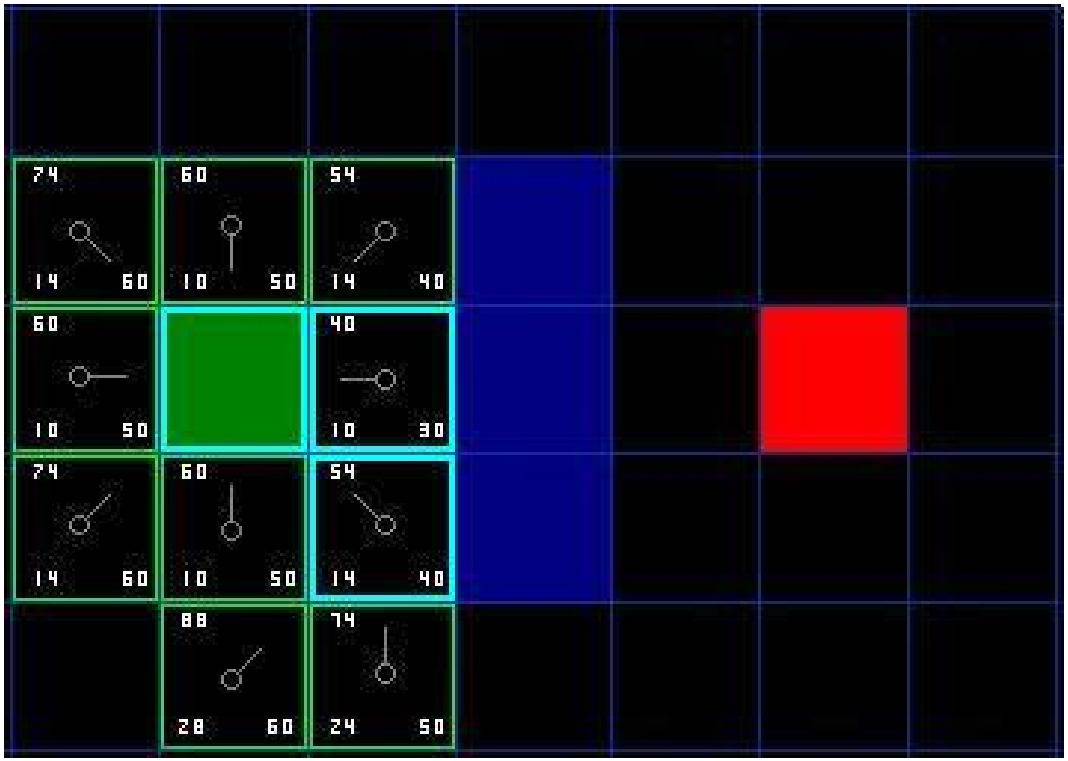
Un Ejemplo

- Ahora comprobamos los cuadros adyacentes. Todos los que hay a la derecha son cuadros de muro, así que no los tenemos en cuenta. El de la izquierda es el cuadrado inicial; ese está en la lista cerrada, así que lo ignoramos también.
- Los otros 4 cuadrados ya están en la lista abierta, así que necesitamos comprobar si alguno de los caminos hasta esos cuadros es mejor que el del cuadrado actual hasta ellos.

Un ejemplo

- Examinemos el cuadrado debajo del cuadrado actual.
- Su G actual es 14. Si fuésemos a través del cuadro actual hasta allí, la G sería igual a 20 y como este dista 40 del destino, la F estimada es de 60.
- Sin embargo, si nos movemos directamente en diagonal tendremos un valor de F igual a 54.
- Así pues seleccionamos movernos un cuadro en diagonal. Lo incluimos en CERRADOS.
- Abandonamos el cuadrado horizontal y mantenemos el puntero del cuadrado diagonal al cuadrado inicial.

Un ejemplo


74
14 60
60
10 50
54
14 40
60
10 50
40
10 30
74
14 60
60
10 50
54
14 40
88
28 60
74
24 50

Un ejemplo

- Ahora volvemos a comprobar los cuadrados adyacentes. El de la izquierda y el superior son cuadros de muro así que los ignoramos.
- Tampoco tenemos en cuenta el cuadro que está debajo del muro:

  No podemos llegar a ese cuadrado sin que el agente "raspe" la esquina al pasar en diagonal; lo mejor es dar un pequeño rodeo, primero bajando y luego yendo hacia la derecha. (Nota: Esta regla es opcional. Su uso depende del contexto)

Un ejemplo

- Eso deja otros 5 cuadros. Los otros 2 cuadros bajo el actual no están en la lista abierta así que los añadimos y el cuadro actual se convierte en su padre.
- De esos otros 3 cuadros, 2 ya están en la lista cerrada (el cuadro inicial, y el cuadro que hay encima del actual, ambos resaltados en azul en el diagrama) así que los ignoramos.

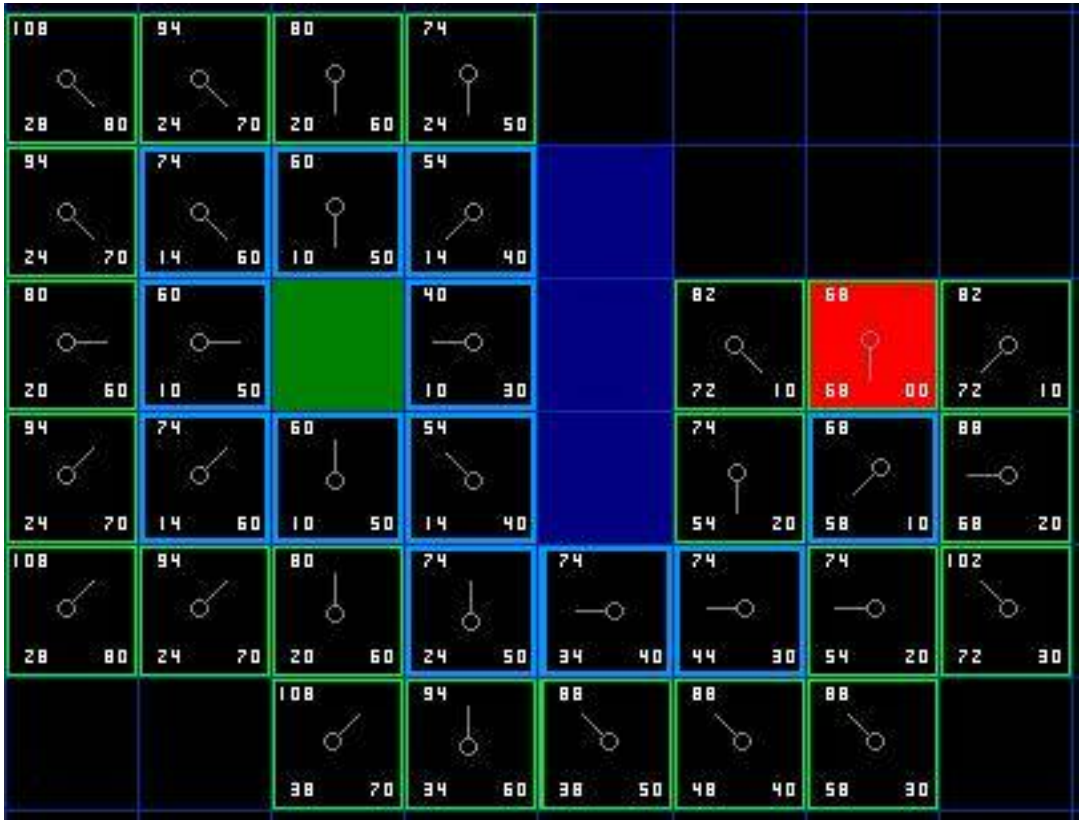
Un ejemplo

- El último cuadro, el de la izquierda del cuadro actual, se comprueba para ver si el coste G hasta él desde el cuadro actual, es menor que llegando directamente desde el cuadro inicial. Lo hacemos y no hay suerte, así que ya estamos listos para comprobar el siguiente cuadro de nuestra lista abierta.

Un ejemplo

- Repetimos este proceso hasta que añadimos el cuadro objetivo a la lista abierta, en ese momento parecería algo como la ilustración siguiente

Un ejemplo


108 28 80
94 24 70
80 20 60
74 24 50
94 24 70
74 14 60
60 10 50
54 14 40
80 20 60
60 10 50
40 10 30
82 72 10
68 68 00
82 72 10
94 24 70
74 14 60
60 10 50
54 14 40
74 54 20
68 58 10
88 68 20
108 28 80
94 24 70
80 20 60
74 24 50
74 34 40
74 44 30
74 54 20
102 72 30
108 38 70
94 34 60
88 38 50
88 48 40
88 58 30

Un ejemplo

- Observese que el cuadro padre para el cuadrado dos cuadros por debajo del cuadro inicial ha cambiado desde la ilustración anterior.
- Antes tenía un coste G de 28 y apuntaba al cuadrado encima suya y a la derecha. Ahora tiene una puntuación de 20 y apunta al cuadrado encima suya.

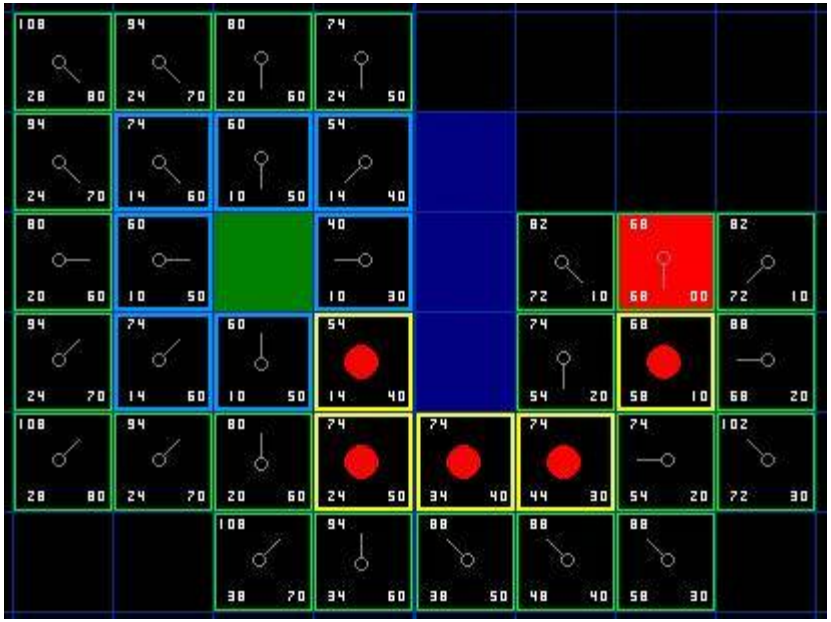
Un ejemplo

- Esto ocurrió en algún momento por la forma en la que se ejecuta la búsqueda, donde la puntuación G fue comprobada y reducida usando un nuevo camino - el padre cambió y G y F fueron recalculadas.
- Este cambio no parece importante en este ejemplo, pero hay muchas posibles situaciones donde este constante control significará la diferencia en la determinación del mejor camino hasta tu objetivo.

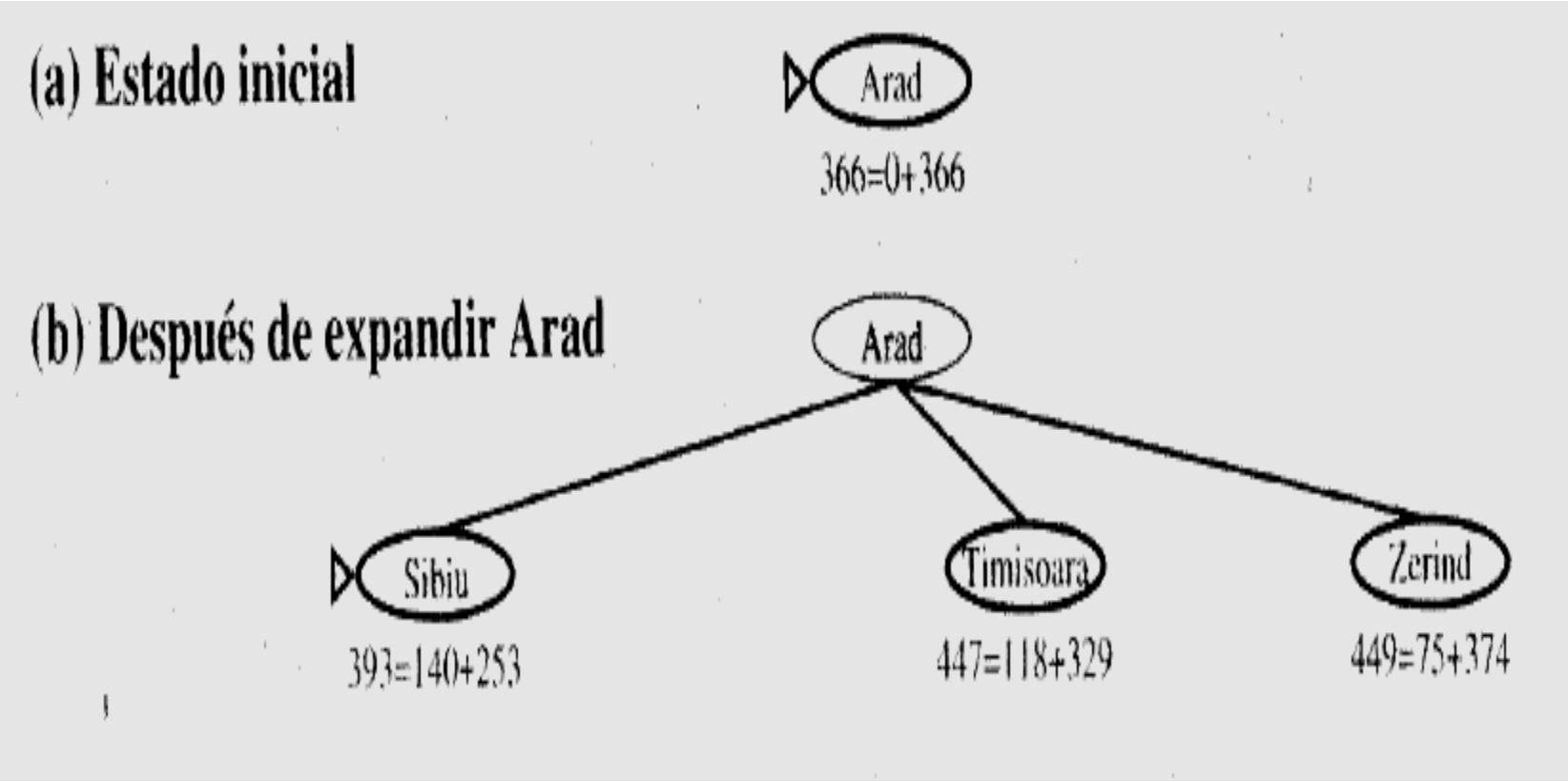
Un ejemplo

- Para determinar el camino optimo empezamos desde el cuadro objetivo rojo, y vamos hacia atrás de un cuadrado a su padre, siguiendo las flechas.
- Eso lleva de vuelta al cuadrado inicial y determina que movimientos serán el camino a seguir.

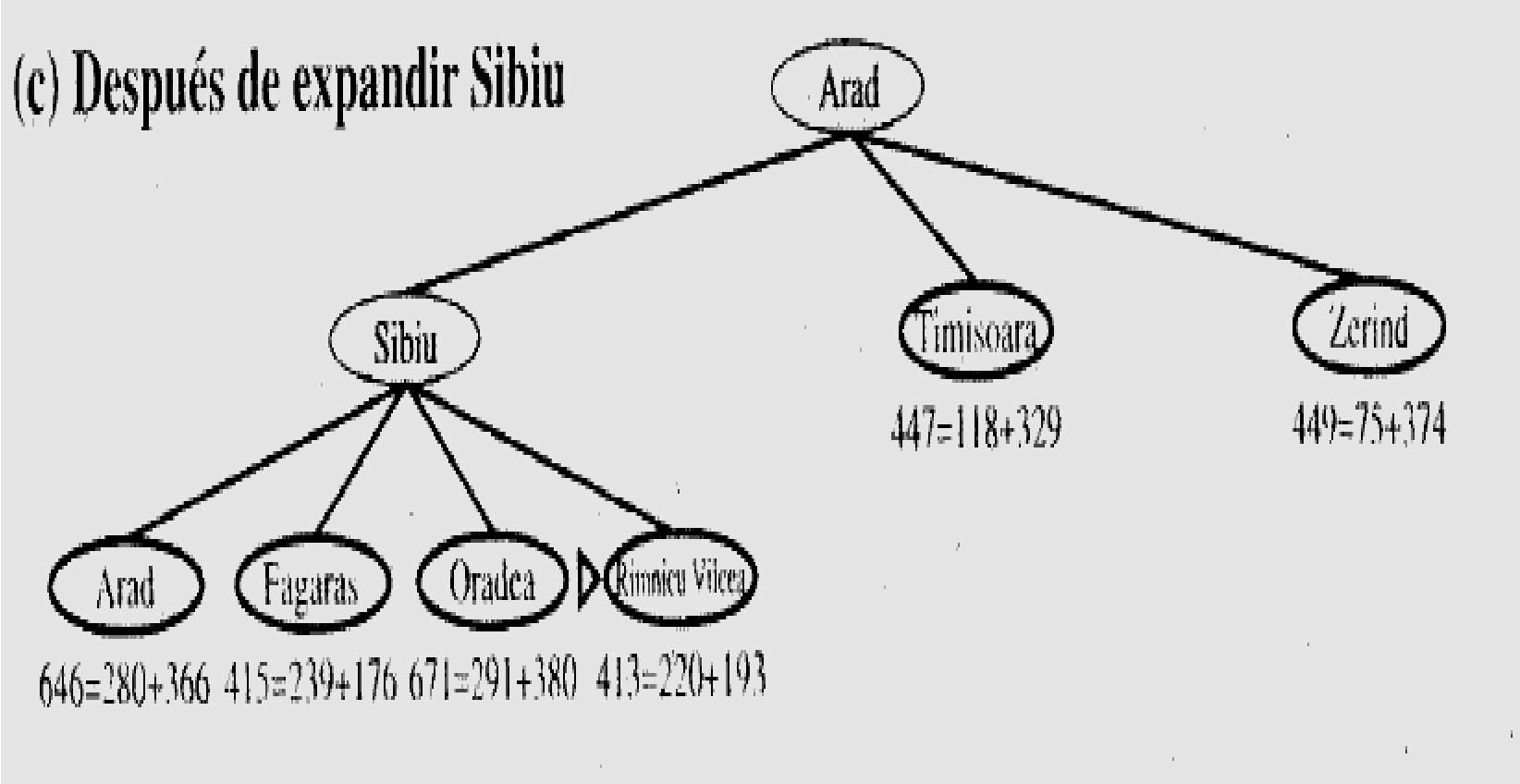
Un ejemplo



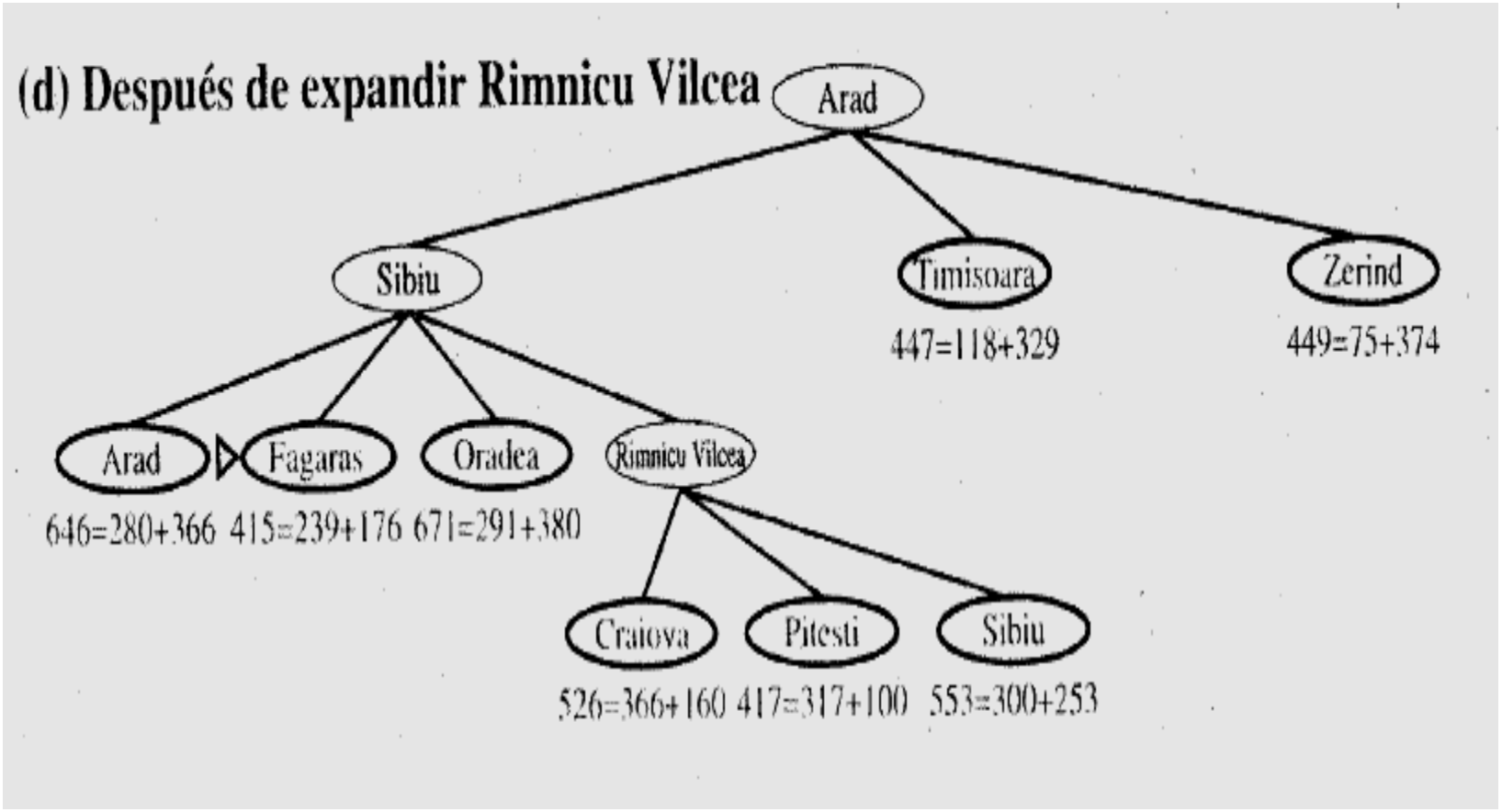
Ejemplo: Russell-Norvig



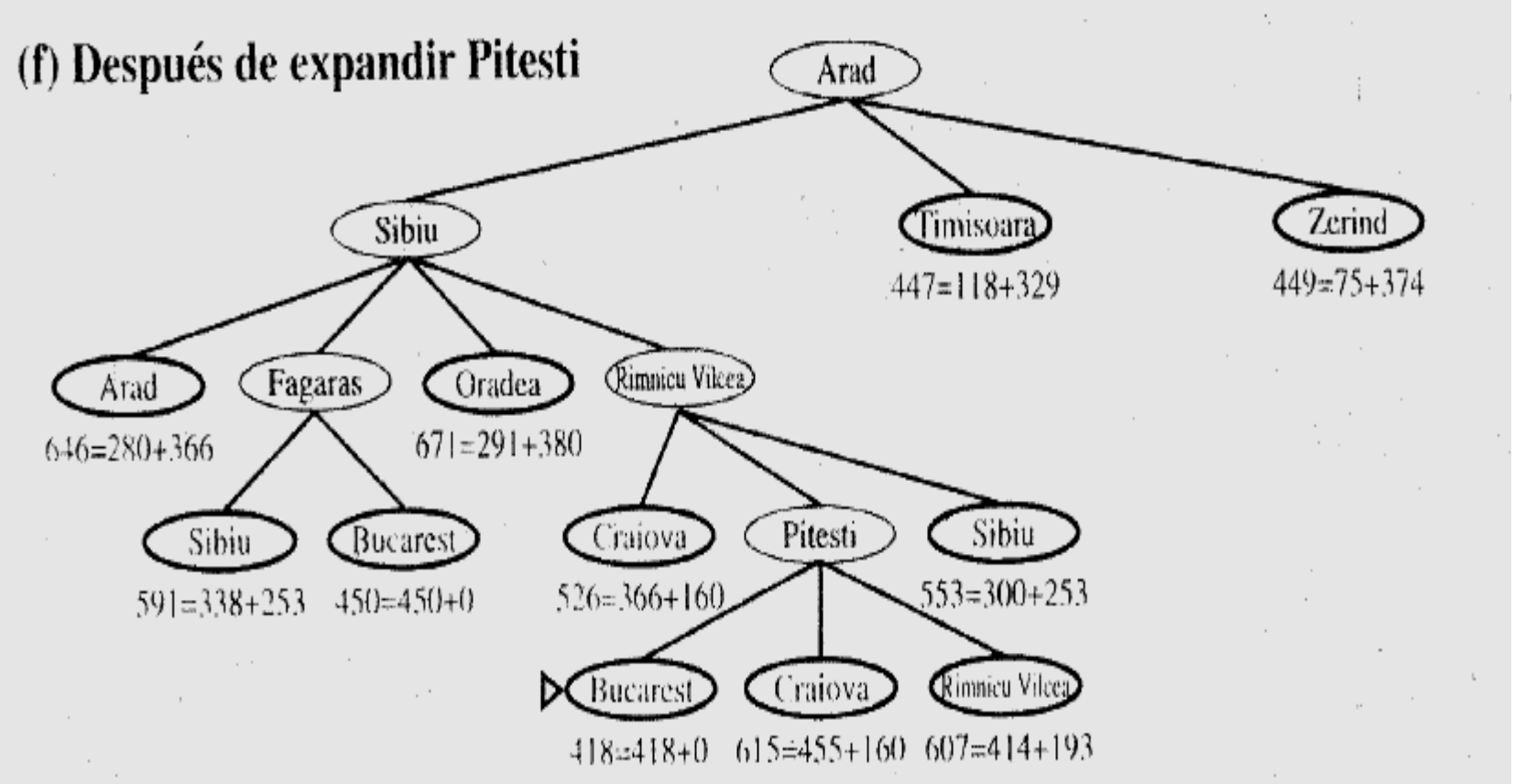
Ejemplo: Russell-Norvig



Ejemplo: Russell-Norvig

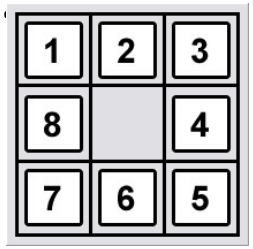


Ejemplo: Russell-Norvig

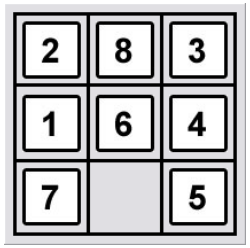


Ejemplo del 8-Puzzle

- Supongamos la heurística h(x)= nº de fichas descolocadas.
- Supongamos g(x) = nivel de x en el arbol de búsqueda
- Estado final deseado:



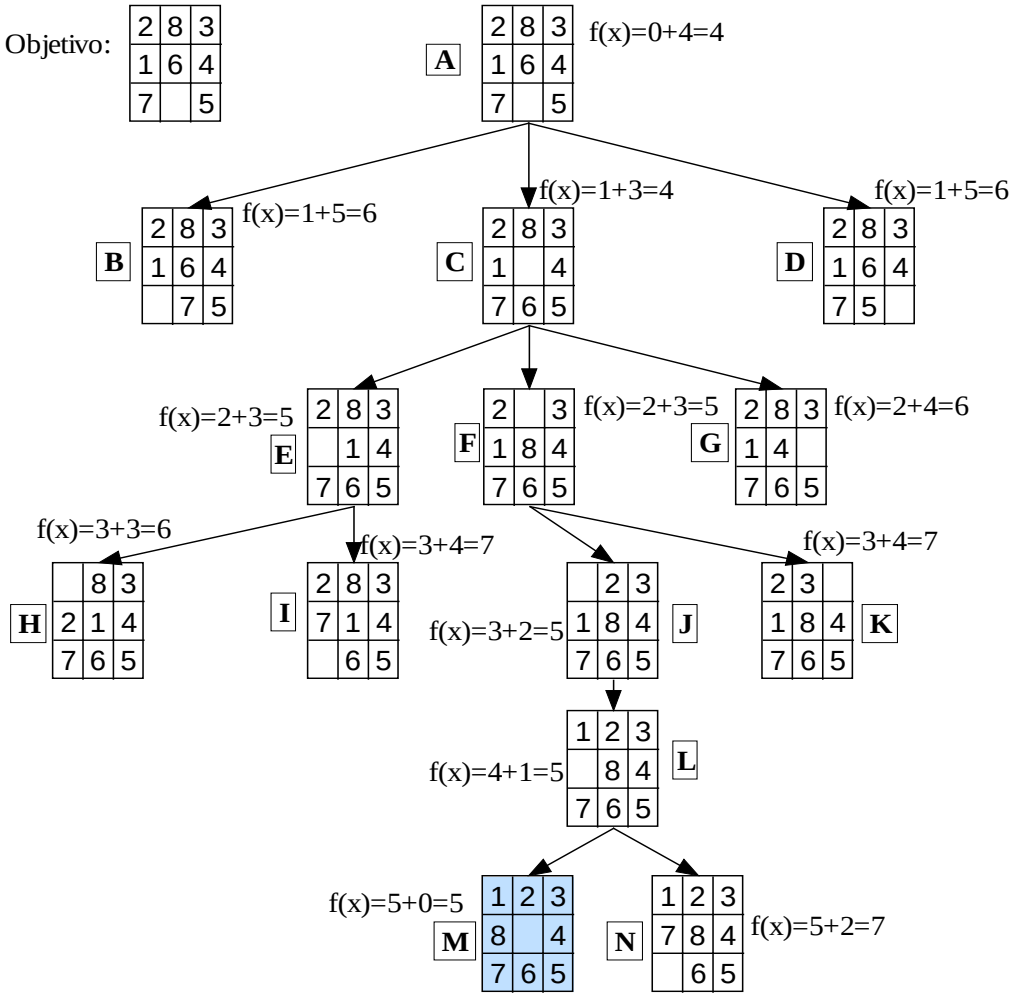
- Estado inicial $\mathbf{x_0}$:



$\mathbf{g(x_0)= 0}$

$\mathbf{h(x_0)= 4}$

Ejemplo del 8-Puzzle.



Pasos A*

1.- en **A:** Abre **B, C, D**

2.- en **C:** Abre **E, F, G**

3.- en **E**: Abre **H, I**

4.- en **F:** Abre **J, K**

5.- en **J:** Abre **L**

6.- en **L**: Abre **M, N**

7.- **M:** Objetivo alcanzado

Algoritmo A*: Propiedades

- Supongamos $\mathbf{h^*(x)}$ el coste del camino óptimo en el árbol de estados para alcanzar la solución desde el nodo **x**.
- La función heurística óptima $\mathbf{f^*(x)= g(x)+h^*(x)}$ es el coste del camino óptimo entre el estado inicial y el objetivo, pasando por **x**.
- Como $\mathbf{h^*(x)}$ es desconocida, tenemos que estimarla mediante **h(x)**.

Algoritmo A*: Propiedades

- Si hacemos **h(x)= 0**, el algoritmo es idéntico al algoritmo de Dijsktra a una búsqueda en anchura.
- Si **h(x)= h*(x)**, entonces no habría proceso de búsqueda: Llegaríamos directamente a la solución.
- Si **h(x)>h*(x)** para **algún x**, puede que no encontremos solución óptima.

Algoritmo A*: Propiedades

- Las heurísticas **h(x)** tales que **0<= h(x) <= h^*(x)** para **todo x**, se llaman **admisibles.**
- Si **0<= h(x) <= h^*(x)** para **todo x**, entonces se alcanzará la solución óptima (mínimo número de pasos para llegar al objetivo).
  - **Ejemplo 1:** En problemas de caminos mínimos (por ejemplo, rutas entre ciudades), la heurística de "distancia en línea recta entre dos ciudades" siempre será menor que el coste real del viaje por carretera.
  - **Ejemplo 2:** En el problema del 8-puzzle, el número de piezas descolocadas siempre será menor (o igual) que el número de pasos necesarios para colocarlas en su posición correcta.

Algoritmo A*: Propiedades

- Si las heuristicas $\mathbf{h_1(x)}$ y $\mathbf{h_2(x)}$, cumplen

  $\mathbf{0<=h_1(x)<=h_2(x)<=h^*(x)}$, para cualquier nodo/estado **x**,

entonces se dice que $\mathbf{h_2(x)}$ está ***mejor informada*** que $\mathbf{h_1(x)}$, dado que se acerca más a la heurística óptima.

- Una heurística **A** mejor informada que otra heurística **B** requerirá menos pasos para alcanzar la solución destino.

Algoritmo A*: Propiedades

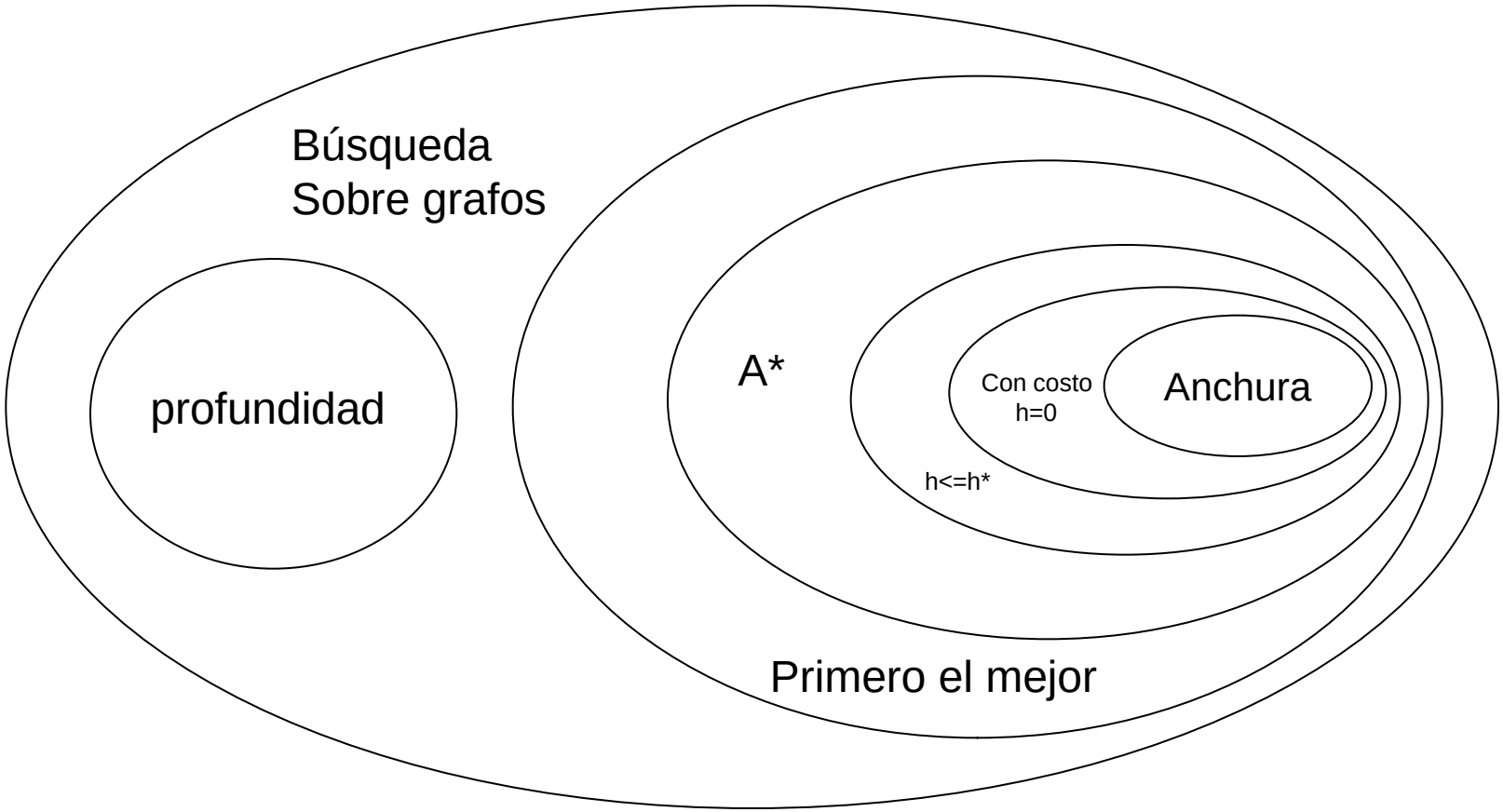
- Supongamos dos nodos del grafo de estados generado por A^*, $\mathbf{x_i}$ y $\mathbf{x_j}$, tales que $\mathbf{x_j}$ es sucesor de $\mathbf{x_i}$.
- Decimos que la heurística **h(x)** ***es consistente*** si:

  $\mathbf{h(x_i)-h(x_j)<=c(x_i,x_j)}$, donde $\mathbf{c(x_i, x_j)}$ es el coste del arco que une $\mathbf{x_i}$ con $\mathbf{x_j}$.
- Si se cumple la condición de consistencia, entonces **f(x) es no decreciente**; es decir, $\mathbf{f(x_j)>=f(x_i)}$ para cualquier par de nodos $\mathbf{x_j}$ sucesor de $\mathbf{x_i}$.
- Esto implica que, **cada vez que A^* expande un nodo, ha encontrado un camino óptimo al mismo.**

Algoritmo A*: Propiedades

- **Limitaciones de A*:**
  - Al igual que en cualquier algoritmo de búsqueda en grafos de los estudiados, la expansión de un número grande de nodos puede llevar al algoritmo a agotar la memoria del computador.
  - Puede no ser viable encontrar una heurística admisible y consistente para algún problema.
  - No en todos los problemas se conoce el estado final al que se desea llegar: En su lugar, se tiene una función objetivo que se desea optimizar.
  - Es necesario conocer un estado inicial del que se parte para solucionar el problema.

Algoritmos de búsqueda



Dificultades del proceso deliberativo

- Los procesos de percepción no siempre pueden obtener la información necesaria acerca del **estado** del entorno
- Las acciones pueden no disponer siempre de modelos de sus **efectos**
- Pueden haber otros procesos físicos, u **otros agentes**, en el mundo, influyendo en los resultados de las acciones

Dificultades del proceso deliberativo

- En el tiempo que transcurre desde la construcción de un plan, el mundo puede cambiar de tal manera que el plan ya no sea adecuado
- Podría suceder que se le requiriese al agente actuar antes de que pudiese completar una búsqueda de un estado objetivo
- Aunque el agente dispusiera de tiempo suficiente, sus recursos de memoria podrían no permitirle realizar la búsqueda de un estado objetivo

Modificaciones del proceso de búsqueda

- Búsqueda orientada a subobjetivos
- Búsqueda jerárquica

- Búsqueda con horizonte