PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

INTELIGENCIA ARTIFICIAL

GONZALO TELLO VALENZUELA

Profesora: Silvanna Roncagliolo

1er Semestre, 2020

Índice

Lista de Figuras

Lista de Tablas

1. Representación en espacio de Estados

1.1. Definición

Para resolver un problema, se requiere una representación de éste, luego un algoritmo de búsqueda será que resuelva el problema con dicha representación. La abstracción del problema se basa en identificar:

- Punto de partida
- Objetivo
- Acciones para realizar desde el punto inicial hacia el punto final.

Se ocupa la representación mediante espacio de estados que consta por:

- Estados
  - Existe un número finito de estados
  - Es una representación de los elementos que describen el problema en un momento dado.
  - Pueden ser 'Estado Inicial', 'Estado cualquiera' o 'Estado Final'(también conocido como Estado Objetivo).
  - El estado final pueden ser 1 o varios. Se hablamos de un 1 estado final diremos que es un objetivo especifico, en caso contrario, diremos que es un objetivo genérico.
- Operadores (acciones/movimientos)
  - Condición de aplicación.
  - Conlleva a un estado actual, hacia un nuevo estado.

Por ejemplo:

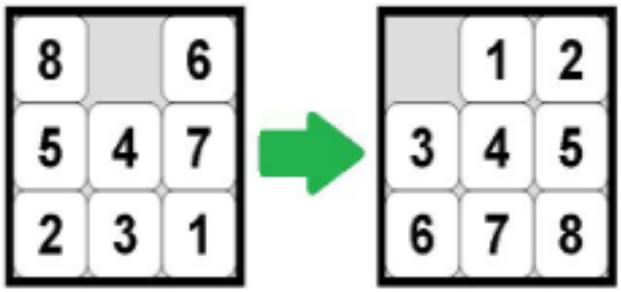


Figura 1: Ejemplo operadores del estado i al estado j

El conjunto de todos los estados conforma lo que se denomina como **espacio de estados**. Se tiene la representación de todos los caminos que hay entre todos los estados. El **Espacio de estado** forma un grafo (nodes = states & edges = actions), lo cual, representará los cambios que pueden existir desde todos los estados y además mostrará el camino hacia el estado objetivo.

1.2. Solución

Secuencia de operadores que permite llegar desde el estado inicial al estado final,o también, solución está en estado final.

(búsqueda ....)(TIPO DE SOLUCIÓN)

1.2.1. Representación gráfica espacio de estado

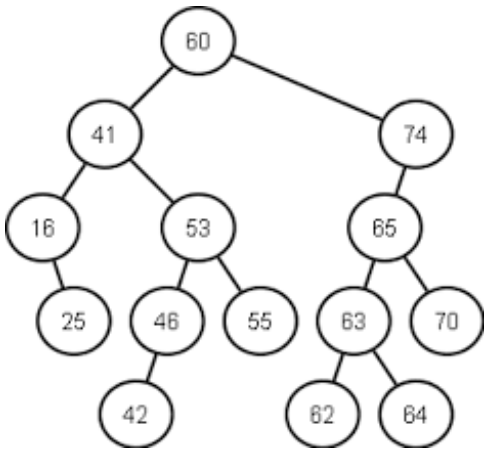


Figura 2: Grafo de Búsqueda

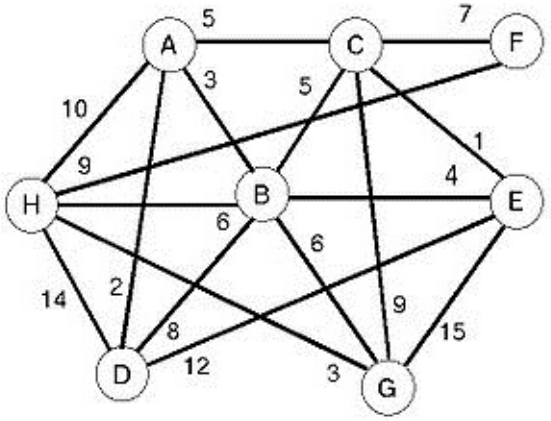


Figura 3: Árbol de búsqueda

1.2.2. Branching factor(b)

El factor de ramificación o branching factor considera el número de descendientes de cada nodo, lo que se calcula de la siguiente manera:

$$b = \frac{valor}{rango} \tag{1}$$

Desde el estado actual, la cantidad de estados que son accesibles aplicando todos los posibles operadores.

1.2.3. Depth(d)

Profundidad en la que se encuentra la (primera) solución.

(d max)

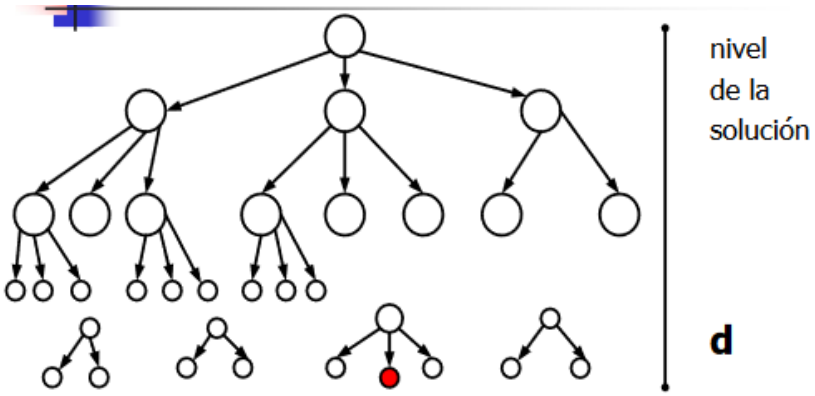


Figura 4: Profundidad máxima de un grafo

2. Búsqueda - Análisis

¿Como es posible calcular la cantidad total de estados en un árbol?

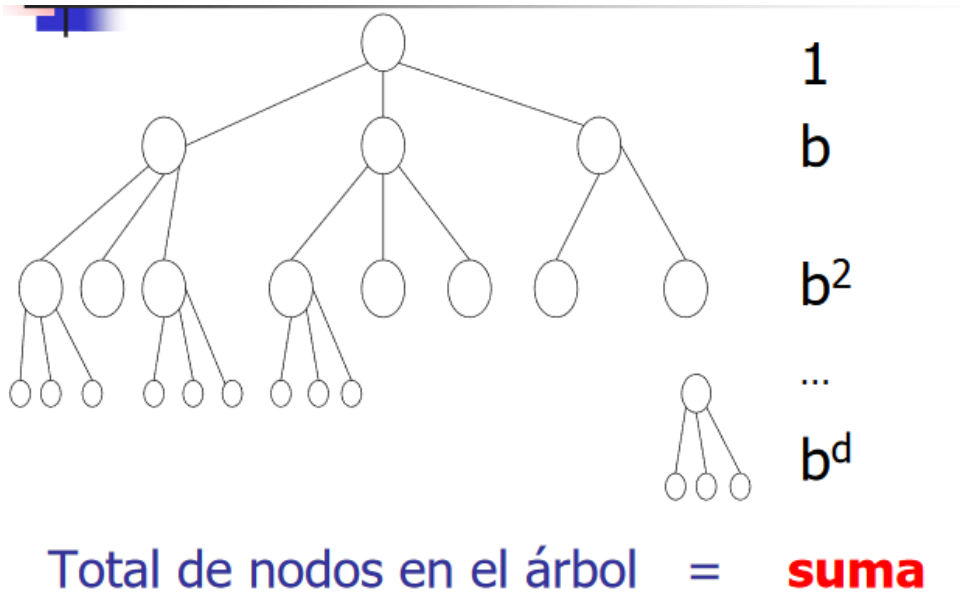


Figura 5: Total de nodos en el árbol

Lo cual, sumados nos entrega lo siguiente:

$$1 + b + b^2 + ... + b^d = \frac{b^{d+1} - 1}{b - 1} \tag{2}$$

$S = 1 + b + b^2 + ... + b^d$ (multiplicando por b)
$Sb = b^2 + b^3 + ... + b^{d+1}$(Restando con lo anterior)
$Sb - S = b^{d+1} - 1$
$S(b - 1) = b^{d+1} - 1$

$$0(b^d)$$

El tiempo de cada búsqueda en relación a su complejidad corresponde a la siguiente tabla:

b=10	10^0	1
	10^1	10
	10^2	100
	10^3	1000
	10^4	10000
	10^5	100000
	10^6	1000000
	Suma	1.111.111

Tabla 1: Nodos totales de b=10

Complejidad	$n = 10$	$n = 20$	$n = 30$	$n = 40$
n	0,00001 seg.	0,00002 seg.	0,00003 seg.	0,00004 seg.
n^2	0,0001 seg.	0,0004 seg.	0,009 seg.	0.0016 seg.
n^3	0,001 seg.	0,008 seg.	0,027 seg.	0.064 seg.
2^n	0,001 seg.	1 seg.	17,9 min.	12,7 días
3^n	0,059 seg.	58 min.	6,5 años	3.855 sigles
$n!$	3,63 seg	771 siglos	8x1016 siglos	3x1032 siglos

Figura 6: Complejidad de un grafo a partir de la cantidad de nodos

2.1. Tipos de búsqueda

	BFS	DFS	ID-DFS
Termina?	Si existe solución grafo finito	Solo si es finito	Si existe solución grafo finito
Encuentra solución	Si	No necesariamente	Si
Solución 'óptima'?	Si	No necesariamente	Si

Tabla 2: Tipos de búsqueda

A partir de aquí sale la denotación Q que consiste en la lista de caminos 'visitados'. Dicho, el espacio y tiempo de cada búsqueda consiste en:

	BFS	DFS	ID-DFS
Tiempo	$O(b^d)$	$O(b^d)$	$O(b^d)$
Espacio			
Q	$O(b^d)$	$O(b * d)$	$O(b * d)$
Q + visitados	$O(b^d)$	$O(b^d)$	$O(b^d)$

Tabla 3: Tiempo y espacio según su búsqueda

2.2. Ejemplos

2.2.1. Ejemplo 1

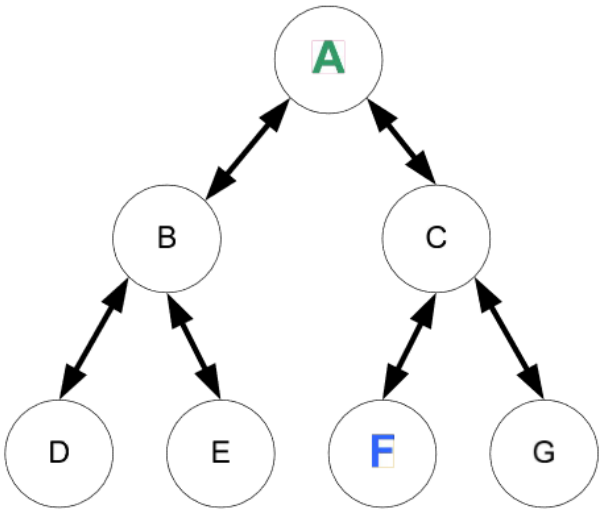


Figura 7: Ejemplo 1 Búsqueda y análisis

Q
{A-}
{B(a),C(a)}
{C(a),A(b),D(b),E(b)}
{A(b),D(b),E(b),A(c),F(c),G(c)}
{D(b),E(b),A(c),F(c),G(c),B(a),C(a)}
{E(b),A(c).F(c),G(c),B(a),C(a),B(d)}
{A(c),F(c),G(c),B(a),C(a),B(d),B(e)}
{F(c),G(c),B(a),C(a),B(d),B(e),B(a),C(a)}

Tabla 4: Con BFS sin revisión de duplicados Q/visitados

Solución: A-C-F

Q
{A-}
{B(a),C(a)}
{A(b),D(b),E(b),C(a)}
{B(a),C(a),D(b),E(b),A(c)}
{A(b),D(b),E(b),C(a),D(b),E(b),C(a)}

Tabla 5: Con DFS sin revisión de duplicados Q/visitados

Solución: LOOP

2.2.2. Ejemplo 2

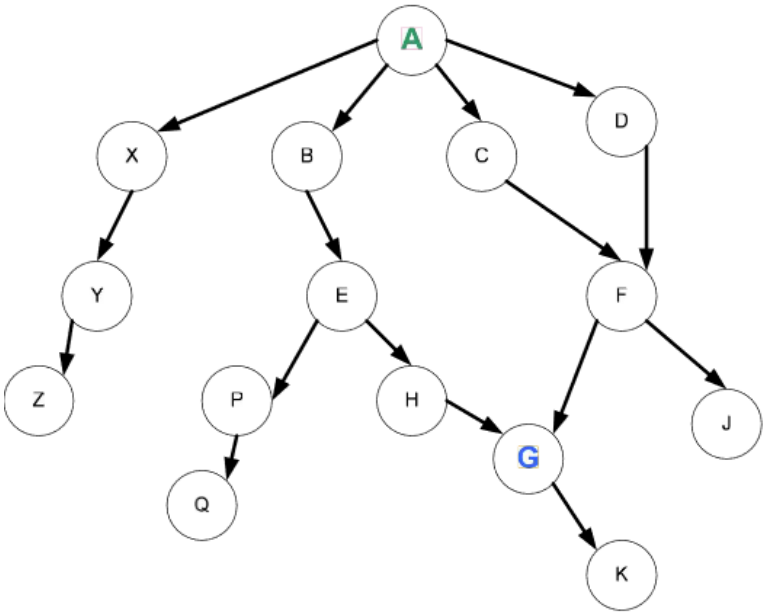


Figura 8: Ejemplo 2 Busqueda y análisis

Q	visitados
{A-}	{ }
{Xa,Ba,Ca,Da}	{A-}
{Ba,Ca,Da,Yx}	{A-,Xa}
{Ca,Da,Yx,Eb}	{A-,Xa,Ba}
{Da,Yx,Eb,Fc}	{A-,Xa,Ba,Ca}
{Yx,Eb, Fc}	{A-,Xa,Ba,Ca,Da}
{Eb, Fc,Zy}	{A-,Xa,Ba,Ca,Da,Yx}
{Fc,Zy,Pe,He}	{A-,Xa,Ba,Ca,Da,Yx,Eb}
{Zy,Pe,He,Gf,Jf}	{A-,Xa,Ba,Ca,Da,Yx,Eb,Fc}
{Pe,He,Gf,Jf}	{A-,Xa,Ba,Ca,Da,Yx,Eb,Fc,Zy}
{He,Gf,Jf,Qp}	{A-,Xa,Ba,Ca,Da,Yx,Eb,Fc,Zy,Pe}
{Gf,Jf,Qp}	{A-,Xa,Ba,Ca,Da,Yx,Eb,Fc,Zy,Pe,He}

Figura 9: BFS (por nivel) con revisión de duplicados Q/visitados

Solución: A - C - F - G

Q	visitados
{A-}	{ }
{Xa,Ba,Ca,Da}	{A-}
{Yx,Ba,Ca,Da}	{A-,Xa}
{Zy,Ba,Ca,Da}	{A-,Xa,Yx}
{Ba,Ca,Da}	{A-,Xa,Yx,Zy}
{Eb,Ca,Da}	{A-,Xa,Yx,Zy,Ba}
{Pe,He,Ca,Da}	{A-,Xa,Yx,Zy,Ba,Eb}
{Qp,He,Ca,Da}	{A-,Xa,Yx,Zy,Ba,Eb,Pe}
{He,Ca,Da}	{A-,Xa,Yx,Zy,Ba,Eb,Pe,Qp}
{Gh,Ca,Da}	{A-,Xa,Yx,Zy,Ba,Eb,Pe,Qp,He}

Figura 10: Con DFS (en profundidad) con revisión de duplicados Q/visitados.

Solución: A - B - E - H - G

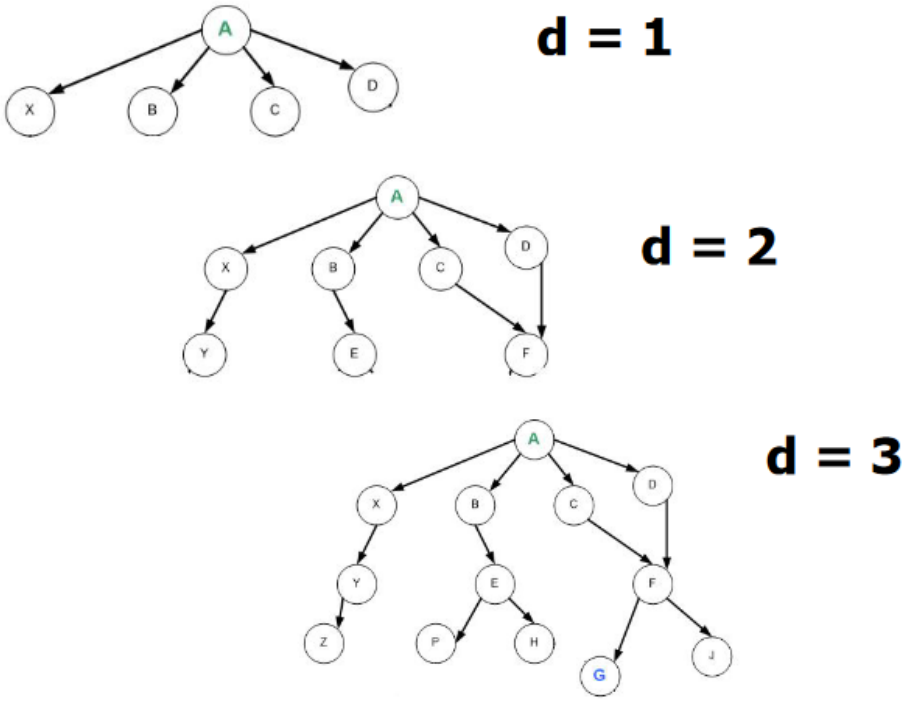


Figura 11: I-DFS con revisión de duplicados Q/visitados

d max	Q	visitados
1	{A-}	{ }
	{Xa,Ba,Ca,Da}	{A-}
	{Ba,Ca,Da}	{A-,Xa}
	{Ca,Da}	{A-,Xa,Yx}
	{Da}	{A-,Xa,Yx,Zy}
	{ }	{A-,Xa,Ba,Ca,Da}

Figura 12: I-DFS para D=1

d max	Q	visitados
2	{A-}	{ }
	{Xa,Ba,Ca,Da}	{A-}
	{Yx,Ba,Ca,Da}	{A-,Xa}
	{Ba,Ca,Da}	{A-,Xa,Yx}
	{Eb,Ca,Da}	{A-,Xa,Yx,Ba}
	{Ca,Da}	{A-,Xa,Yx,Ba,Eb}
	{Fc,Da}	{A-,Xa,Yx,Ba,Eb,Ca}
	{Da}	{A-,Xa,Yx,Ba,Eb,Ca,Fc}
	{ }	{A-,Xa,Yx,Ba,Eb,Ca,Fc,Da}

Figura 13: I-DFS para D=2

d max	Q	visitados
3	{A-}	{ }
	{Xa,Ba,Ca,Da}	{A-}
	{Yx,Ba,Ca,Da}	{A-,Xa}
	{Zy,Ba,Ca,Da}	{A-,Xa,Yx}
	{Ba,Ca,Da}	{A-,Xa,Yx,Zy}
	{Eb,Ca,Da}	{A-,Xa,Yx,Zy,Ba}
	{Pe,He,Ca,Da}	{A-,Xa,Yx,Zy,Ba,Eb}
	{He,Ca,Da}	{A-,Xa,Yx,Zy,Ba,Eb,Pe}
	{Ca,Da}	{A-,Xa,Yx,Zy,Ba,Eb,Pe,He}
	{Fc,Da}	{A-,Xa,Yx,Zy,Ba,Eb,Pe,He,Ca}
	{Gf,Da}	{A-,Xa,Yx,Zy,Ba,Eb,Pe,He,Ca,Fc}

Figura 14: I-DFS para D=3

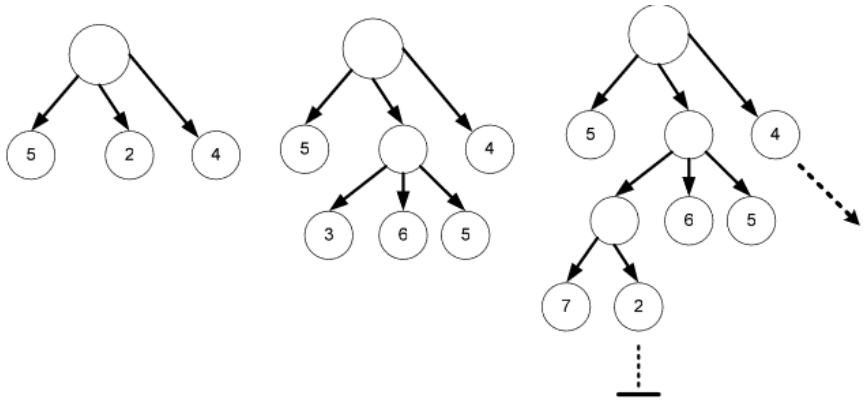
Solución IDFS: A - C - F - G

3. Best First Search

Para analizar el próximo nodo a visitar, es posible escoger entre los siguientes métodos de búsqueda:

- BFS (Breadth First Search), que consiste entre nodos del mismo nivel.
- DFS (Depth First Search), que basa su busqueda en los descendientes de un nodo, llegando hacia el más profundo o el máximo nivel para dicho nodo.

Dada un estado, se calcula un valor asociado a sus características que será otorgado mediante una función de evaluación. Según sea el valor, que obtendrá una descripción de que tan cerca está el objetivo de dicho estado.



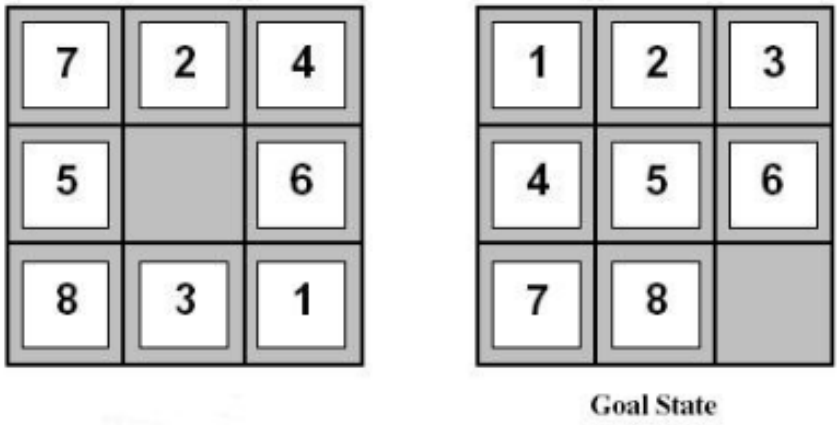
3.1. Heurística

Desde aquí se desprende el concepto Heurística, lo cual es un método para aumentar el conocimiento, donde para efectos de nuestro problema, se usará en sinonimia de hallar e inventar.

h(n) = Estimación del número de pasos para llegar al estado objetivo

3.1.1. Ejemplo

The 8-puzzle



Propuestas de heurísticas

- h(n) = Cantidad de números bien úbicados (x)
- h(n) = Cantidad en posición errónea (x)
- Manhattan distance

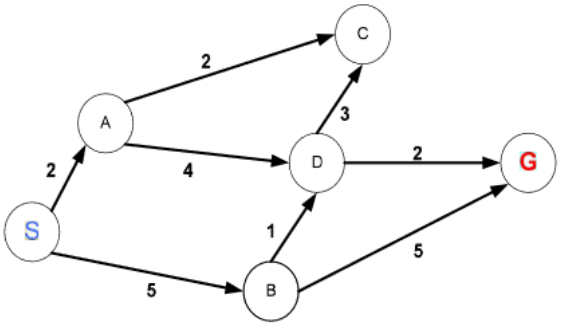
$$\text{h(n)} = \sum |fila_{actual} - fila_{objetivo}| + |col_{actual} - col_{objetivo}|$$

Considerando x cada elemento en posición errónea.

3.2. Algoritmo Greedy - algoritmo voraz

Este algoritmo sólo considera la heurística h propuesta.

3.2.1. Ejemplo



n	h(n)
a	2
b	3
c	1
d	4
s	10
g	0

Q
(s=10)
(a-s=2)(b-s = 3)
(b-s = 3) (c-a-s=1)(d-a-s = 4)
(c-a-s=1) (b-s=3) (d-a-s = 4)
(b-s = 3) (d-a-s=4)
(d-a-s=4) (d-a-s=4) (g-b-s=0)
(g-b-s = 0) (d-a-s=4)

Tabla 6: Caminos recorridos usando Greedy

Solución encontrada: (g-b-s = 0) con costo real 10.

Al generar los sucesores de un nodo,si alguno de estos ya fue visitado, no se incluye, análogo si ya está en Q, debido a que tendrá el mismo valor.

3.3. Algoritmo A

El Algoritmo de búsqueda A está designado bajo la siguiente fórmula:

$$F(n) = G(n) + h(n) \tag{3}$$

Donde:

- G(n): costo actual para llegar a n desde el estado inicial.

- h(n): Estimación de costo para llegar desde n al objetivo (valor heurístico).
- F(n): Estimación total del costo del camino desde el nodo inicial, al objetivo, pasando por n.

Se dirá que el algoritmo de búsqueda es BFS cuando h(n) = 0 y g(n)=1. Por otro lado, diremos que la búsqueda es de costo uniforme cuando:

- h(n) = 0
- g : costo prop. operador

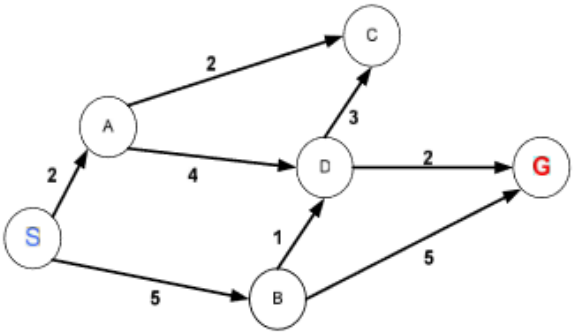
El costo de aplicar el operador dará un valor a cada estado, y se utiliza el método de el primer mejor.

3.4. Búsqueda de costo uniforme

Consideraciones al generar descendientes por medio del método de búsqueda de costo uniforme.

- Si se llega a un nodo duplicado con valor mayor **o igual**...no se incluye en Q.
- Si se llega a un nodo duplicado ahora con menor valor, sólo queda el nuevo de menos valor.

3.4.1. Ejemplo



n	h(n)
a	2
b	3
c	1
d	4
s	10
g	0

Q
(s=0+10 = 10)
(a-s=2+2=4) (b-s = 5+3 = 8)
(b-s=8) (c-a-s=4+4)(d-a-s = 6+4=10) **(b-s=8)**(c-a-s=8)(d-a-s=10)
(c-a-s=8)(d-a-s=10)(d-b-s=6+4=10)(g-b-a=10+0=10) **(c-a-s=8)**(d-a-s=10)(g-b-s=10)
(d-a-s=10)(g-b-s=10)
(g-b-s=10) (c-d-a-s=9+1=10)(g-d-a-s=8+0=8) (g-d-a-s)=8

Tabla 7: Q con búsqueda de costo uniforme

Solución encontrada: (S-A-D-G) costo real de 8.

4. Algoritmos A y A*.

Recordar: Una estrategia de búsqueda está definida por el orden en la elección de los nodos a expandir. Éstas estrategias pueden ser BFS, DFS y Best First Search. Considerando la fórmula

$$F(n) = g(n) + h(n) \tag{4}$$

- Si **h(n) = 0** y **g(n) = 1**, la estrategia utilizada será BFS.
- Si **g(n) = 0** y **solo se considera h(n)**, la estrategia utilizada será Greedy search.

$$F(n) = h(n) \tag{5}$$

- Si **h(n) = 0** y **g(n) es el costo propio del operador**, la estrategia utilizada será búsqueda de costo uniforme.

$$F(n) = g(n) \tag{6}$$

4.1. Heurísticas admisibles

- h(n) es una heurística admisible, si para todo nodo n:

$$h(n) \leq h^*(n) \tag{7}$$

  En donde h*(n) es el **verdadero costo para llegar al objetivo desde n**.Una heurística admisible nunca sobreestima el costo para llegar al objetivo.

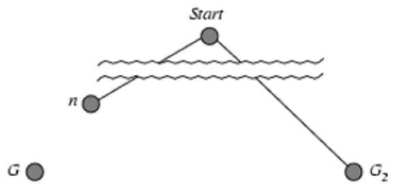
$$AlgortimoA \Longrightarrow F(n) = g(n) + h(n) \tag{8}$$

  Si $h(n) \leq h^*(n)$ (h es una heurística admisible), lo denominamos **Algoritmo A***

- Teorema:

  h(n) es admisible $\Longrightarrow A^*$ encuentra solución óptima

  Demostración:



- Asumir que G2 es un objetivo no-óptimo que podría ser visitado
- Sea n un nodo **aún no visitado**, además siendo parte del camino más corto al objetivo G.

  $G^*(n)$ = Largo del camino **más corto** desde nodo inicial a n
  $h^*(n)$ = Largo del camino **más corto** desde n hacia el objetivo
  $f^*(n)$ = Largo del camino **más corto** desde el nodo inicial a objetivo pasando por nodo n.

  $F(G_2) \geq F(G)$
  $F(G) = F^*(G)$
  $F^*(S) = F^*(G) = F^*(N) \Longrightarrow F(G_2) \geq F^*(S)$

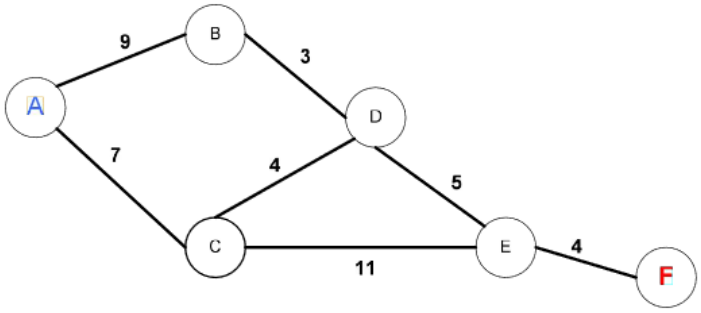
  $G(n) = G^*(n)$ (n en camino óptimo)
  $h(n) \leq h^*(n)$ (h admisible)
  $G(n) + h(n) \leq G^*(n) + h^*(n)$
  $f(n) \leq f^*(n)$
  $f(n) \leq f^*(S)$
  $f(G_2) < f^*(S)$
  $f(n) \leq f^*(S) < f(G_2)$
  $f(n) < f(G_2)$
  G_2 **NO** se expande

- Si $h_2(n) \geq h_1(n)$ para todo n (ambas admisibles):
  - h_2 **dominates** h_1
  - h_2 **more informative** h_1
  - h_2 da mejores resultados en la búsqueda.
- Procedimiento
  open = { estado_inicial }, closed = {}
  repetir mientras open $\neq$ {}
  sacar el primer nodo de open, X
  si X es objetivo, retornar solución
  generar hijos X
  <hace para todo hijo siguiente >
  fin_repetir
  return(fail)

- $f^{'}$: Evaluación nueva

si hijo $\in$ open
    si $f^{'}$(hijo) $\leq$ f(hijo)
        cambiar & odernar open
si hijo $\in$ close
    si $f^{'}$(hijo) $<$ f(hijo)
        sacarlo de closed
        reinsertar en open & ordenar open
si hijo $\notin$ open $\wedge$ hijo $\notin$ closed
    evaluar, incluir en open & ordenar open

4.1.1. Ejemplo



n	h(n)
A	16
B	10
C	13
D	8
E	4
F	0

$$A * F(n) = g(n) + h(n) \tag{9}$$

Q/Open	**Visitados/Closed**
(A=0+16=16)	
(B-A=9+10=19)(C-A=7+13=20)	(A=16)
(D-B-A=12+8=20)(C-A=7+13=20)	(A=16)(B-A=9)
(C-A=7+13=20)(E-D-B-A=17+4=21)	(A=16)(B-A=9)(D-B-A=20)
(D-C-A=11+8=19)(E-D-B-A=17+4=21)	(A=16)(B-A=9)(C-A=20)
(E-D-C-A=16+4=20)(E-D-B-A=17+4=21)	(A = 16)(B - A = 9)(C - A = 20) (D-C-A=19)
(E-D-C-A=16+4=20)	(A = 16) (B - A = 9)(C - A = 20) (D-C-A=19)
(F-E-D-C-A=20+0=20)	(A = 16) (B - A = 9) (C-A = 20) (D-C-A=19)(E-D-C-A=20)

Tabla 8: Ejemplo con Algoritmo A

5. Alpha - Beta

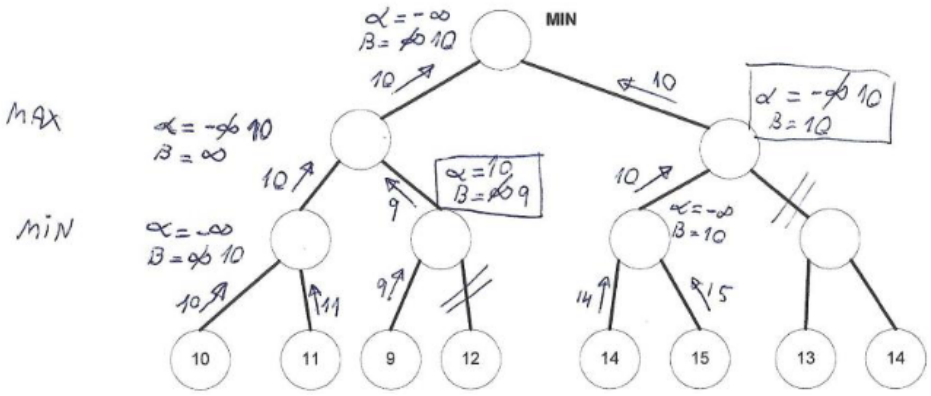
- Comenzar propagación de valores inmediatamente.
- No explorar/expandir nodos que no pueden afectar la decisión respecto de que mover, es decir, NO explorar aquellos nodos que se pueden probar que NO son mejores que los mejores encontrados hasta el momento.
- Alpha α: Límite inferior para valores que un nodo MAX puede ser asignado.
- Beta β: Límite superior para valores que un nodo MIN puede ser asignado.
- Búsqueda se pondra en:
  - En nivel MIN:
    si un valor menor que α es encontrado $(\alpha - cutoff)$
  - En nivel MAX:
    Si un valor mayor que β es encontrado $(\beta - cutoff)$
- Resumen
  - Iniciar nodo raíz con:
    $\alpha = \infty$
    $\beta = -\infty$
  - Nodo MIN actualiza β y al finalizar retorna β
  - Nodo MAX actualiza α y al finalizar retorna α
  - En todo momento $\alpha \geq \beta \Longrightarrow$ podar

5.1. Ejemplo 1

```
    /*need additional code to return operator*/
int AlphaBeta(state, player, depth, alpha, beta)
    if(state=root alpha=-infinito beta=infinito)
        if(deepth==limit or state is terminal)
            return the static evaluation of state
        if(player is min)
            untill all successors s are examined or alpha ≥ beta
            val=AlphaBeta(s,max,depth+1,alpha,beta)
            if(val<Beta) Beta = val
```

return beta
if(player is max)
untill all successors s are examined or **alpha** $\geq$ **beta**
val=AlphaBeta(s,max,depth+1,alpha,beta)
if(val >alpha) alpha = val
return alpha

5.2. Ejemplo 2



6. Heurísticas y Metaheurísticas

6.1. Heurística

- La heurística trata de métodos o algoritmos exploratorios durante la resolución de problemas en los cuales las soluciones se descubren por la evaluación del progreso logrado en la búsqueda de un resultado final.
- Las técnicas heurísticas no aseguran soluciones óptimas sino solamente soluciones válidas, aproximadas; y frecuentemente no es posible justificar en términos estrictamente lógicos la validez del resultado.
- 'La heurística es una serie de procedimientos o estrategias (algoritmo) de las que se supone que llevan a un destino deseado; pero esto no tiene porque ser siempre cierto'.

6.2. Metaheurística

- Las **metaheurísticas** son heurísticas (estrategias) para aplicar heurísticas a problemas.
- En general, la metaheurística consiste en un proceso repetitivo de la siguiente forma: sobre un problema se establece una estrategia de ataque del problema: es la heurística'.
- Esta se emplea para tener una tentativa de solución. Esta tentativa es evaluada para comprobar si es suficientemente buena para los intereses, si no cumple con los objetivos, se modifica la solución y se repite el proceso hasta que la tentativa de solución sea suficientemente válida.
- Un buen método de búsqueda da un equilibrio entre explotar la mejor solución que se conoce (explotación) y explorar el espacio de búsqueda (exploración). El estado inicial puede ser elegido:
  - Al Azar
  - Ser resultado de otra búsqueda.
- Mientras que para los criterios de término, existen factores que pueden hacer interesante sus usos para esto.
  - Cuando no hay un método exacto de resolución, o éste requiere mucho tiempo de cálculo y memoria (ineficiente).

- Cuando no se necesita una solución óptima, solo basta una de buena calidad

- Problemas:
  - Solución depende de punto inicial
  - Solución encontrada puede ser óptimo local
  - No existe información respecto a cuanto difiere la solución encontrada respecto del óptimo (global o local).

6.3. Algoritmos Metaheurísticos

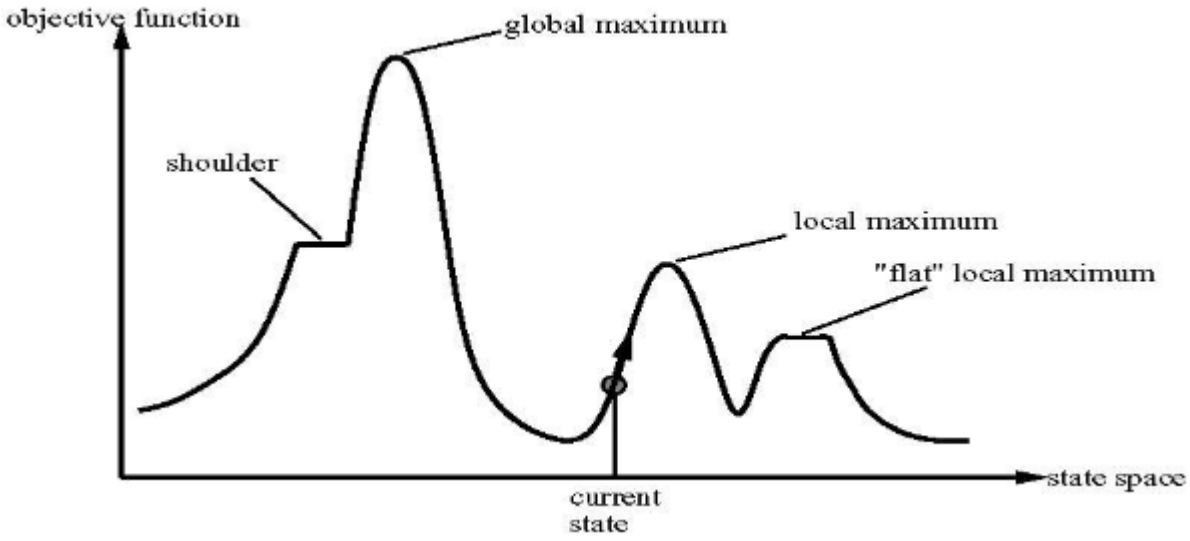


Figura 15: Acercamiento hacia un objetivo con una metaheurística

- Hill Climbing
- Simulated Annealing
- Tabu Search
- Branch and Bound
- Algoritmos genéticos
- Colonias de hormigas

- Algoritmos meméticos
- Híbridos

6.3.1. Hill Climbing

Sucesor mejor que nodo actual.

- **Hill Climbing (simple)**: Primer sucesor mejor que nodo actual

```
function Hill-Climbing(problem)
    returns a state thar is a local maximum
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node
current <- Make-node(initial-State[problem])
    loop for all posible succesors of curret
        neighbor <- a succesor of current
        if Value[neighbor] better than Value[current] then
            current <- neighbor
    end-loop current does not change (there is no better state)
return State[current]
```

- **Steepest Ascent hill climbing** Todos los sucesores con comparados y el mejor es considerado.

```
function Hill-Climbing(problem)
     returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node
current <- Make-Node(initial-state[problem])
loop do
    neighbor <- THE best-valued successor of current
    if Value[neighbor] not better than Value[current] then
        return state[current]
    current<-neighbor
end
```

En ambos casos falla si entre los sucesores no hay un nodo mejor. *Steepest Ascent hill climbing* es similar a búsqueda el mejor primero, el último intenta

con todas las posibilidades y el primero solo intenta con una.

Ejemplo

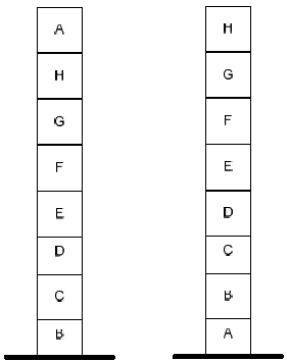


Figura 16: Ejemplo Hill-Climbing

- Heurística 1: Sumar 1 punto por cada block que esté sobre lo que le corresponde y restar 1 punto por cada block que esté sobre el incorrecto.
- Heurística 2: Por cada block que tiene una estructura de soporte correcta sumar 1 punto por cada block en dicha estructura de soporte. Por cada block que tiene una estructura de soporte incorrecta restar 1 punto por cada block en dicha estructura de soporte.

6.3.2. Simulated Annealing

El método Simulated Annealing se generó a partir de una analogía entre el fenómeno de enfriamiento de sustancias puras y un problema de optimización. Se puede decir que el enfriamiento de una sustancia corresponde a un proceso de optimización cuyo objetivo es minimizar la energía del sistema.

- Recocido simulado
- Enfriamiento simulado
- Endurecimiento simulado
- Temple simulado
- Escapar del máximo local permitiendo algunos malos movimientos pero gradualmente disminuir tamaño y frecuencia.

El método realiza una búsqueda local al generar una solución vecina de manera aleatoria, si la solución es mejor que la solución actual se actualiza ésta y se reinicia el proceso de búsqueda a paritr de esta nueva solución, en caso contrario, es decir, en el caso que la solución actual se le da una oportunidad probabilística al método para que acepte una solución de peor calidad que la de la solución actual. Esta oportunidad obedece las leyes naturales que se presenta en el fenómeno del enfriamiento de las sustancias puras en términos matemáticos, obedece con la distribución de probabilidad de Bolzman.

Este proceso de búsqueda se repite hasta que el algoritmo ya no es capaz de generar soluciones de mejor calidad que la actual, típicamente es un proceso que se detiene después de haber realizado un gran conjunto de iteraciones sin que se haya mejorado la calidad de la mejor solución que se tiene hasta el momento.

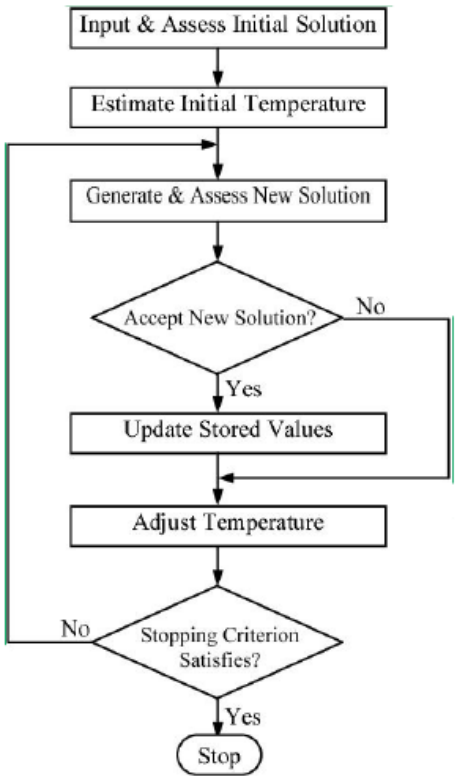


Figura 17: Diagrama de Flujo para Simulated Annealing

A continuación se expone el pseudo código asociado:

function Simulated-Annealing(problem, schedule)
returns a solution state
inputs: problem, a problem
schedule, a mapping from time to 'temperature'
local variables: current, a node
next, a node
T, a 'temperature' controlling prob. of downward steps
current <- make-node(initial-state[problem])
for t <- 1 to ∞ do
T <- schedule[t]
if T = 0 then return current
next <- a randomly selected successor of current
ΔE <- Value[next] - Value[current]
$\Delta E > 0$ then current <- next
else current <- next only with probability $e^{\Delta E/T}$

6.3.3. Tabu Search

Idea: Partir de una solucion inicial e iterativamente reemplazr por otra de su vecindario... de mejor calidad (Algoritmo de mejoramiento iterativo).

Se comienza con una solución al azar, en cada iteración a partir de la solución actual se pasa a la mejor solución de su vecindario (excepto si está en lista tabú - prohibidos momentaneamente). El movimiento siempre se acepta a pesar de que la nueva solución sea peor que la anterior. Se usa una memoria adaptativa la que podrá modificar la estructura de entornos.

Algunas soluciones examinadas recientemente son 'memorizadas' y se vuelven tabú (prohibidas) (en una ventana de tiempo) al tomar decisiones acerca del siguiente punto de búsqueda.

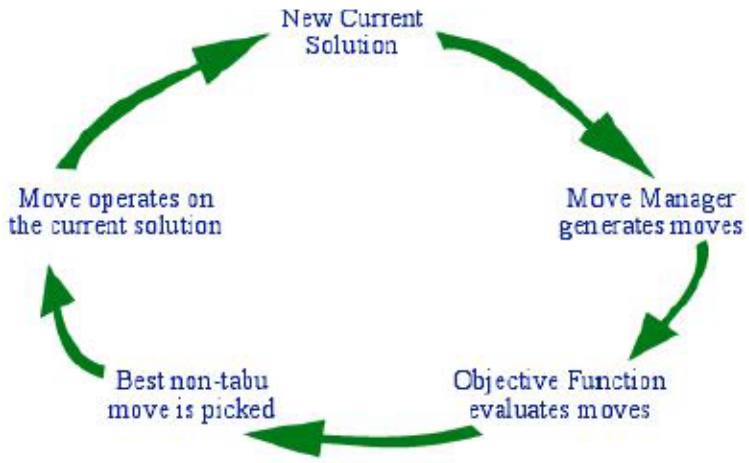


Figura 18: Búsqueda Tabú

El método de búsqueda tabú se basa en utilizar memoria, para recorrer el espacio W. en cada etapa el método genera la mejor solución perteneciente a una vecindad, y verificando el cumplimiento de ciertas restricciones de memoria y ciclaje, dirige la búsqueda hasta esta nueva solución generada a partir de la cual inicia una nueva etapa.

Procedimiento

1. Selecciona un estado $x \in X$ inicial y sea $x^* := x, k = 0$ (contador de iteración) y $T := 0$.

2. Si $S(x) - T = \emptyset$ vé a 4, sino sea $K := k + 1$ y selecciona $s_k \in S(x) - T$ tal que $s_k = OPTIMO(s(x) : s \in S(x) - T))$

3. Sea $x := s_k.Sic(x) < c(x^*)$ (donde x^* es la mejor solución encontrada hasta el momento), sea $x^* := x$

4. Si se agoto el número de interacciones o si $S(x) - T = \emptyset$ entonces se detiene, sino, actualiza T (añade el movimiento actual a la lista tabú y posiblemente elimina el elemento más viejo) y regresa a 2.

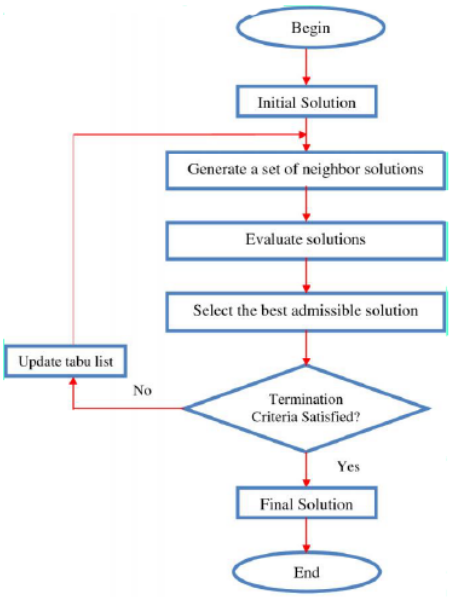


Figura 19: Diagrama de flujos de Tabu Search

6.3.4. Branch and Bound(BB)

Is a general algorithmic method for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization.

It's basically an enumeration approach in a fashion thar prunes the non-promising search space.

6.3.5. Memetic Algorithms

Is a population-based approach for heuristic search in optimization problem. They have shown that they are orders of magnitude faster than traditional Genetic Algorithms for some problem domain. Basically, they combine local search heuristics with crossover operator. For this reason, some researchers have viewed them as Hybrid Genetic Algorithms.

7. Lógica

- Sintáxis:

(wff = well formated formulae)

- Conjunto de reglas que especifican como escribir correctamente.
- Semántica
- Interpretación de sentencias legales

Una sentencia/frase básica/primaria/ atómica es:

- Cada símbolo de verdad (V-F)
- Cada símbolo/variable proposicional

7.1. Lógica Proposicional

- Símbolos: p, q, r, ...
- Valores de verdad:
  - Verdadero (V)
  - Falso (F)
- Conectores:
  - $\vee$
  - $\wedge$
  - $\neg$
  - $=$
  - $\rightarrow$
  - $\equiv$
- Proposición: Frase del mundo que puede ser V o F
- Una sentencia/frase en lógica proposicional es:
  - La negación de una sentencia
  - La conjugación de 2 ó más sentencias

- La disyunción de 2 o más sentencias
- La implicación de una frase a otra
- La equivalencia de 2 sentencias.

- Tabla de verdad.
  - A proposciones atómicas se consideran todos los posibles valores de verdad.
  - 2 expresiones son equivalentes si tiene el mismo valor de verdad para todas las combinaciones posibles de valores de verdad.
  - Esto se puede probar usando tablas de verdad
  - $\neg(\neg p) \equiv p$
  - $p \vee q \equiv \neg p \rightarrow q$
  - $\neg(p \wedge q) \equiv \neg p \vee \neg q$
  - $\neg(p \vee q) \equiv \neg p \wedge \neg q$
  - $p \wedge q \equiv q \wedge p$
  - $p \vee q \equiv q \vee p$
  - $p \wedge (p \wedge r) \equiv (p \wedge q) \wedge r$
  - $p \vee (p \vee r) \equiv (p \vee q) \vee r$
  - $p \vee (p \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
  - $p \wedge (p \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

7.2. Lógica de predicado

Mientras que la lógic proposicional representa hechos acerca del mundo, la lógica de primer orden o lógica de predicado describe un mundo que consta de objetos y propiedades de esos objetos.

7.2.1. Síntaxis

- Símbolos.

  Concatenaación de: a, b, c, ..., 0, 1, 2, ...,9.

  Debe comenzar con una letra.

- Constantes.

  Objetos o propiedades específicas del mundo. Deben comenzar con letra minúscula, por ejemplo:

lluvia, alto, pillin, maria, azul.

- Variables.

  Generaliza la clase o propiedad, debe comenzar con la letra mayúscula. Por ejemplo:

  Perro, X, Color, A, B

- Función.

  Debe comenzar con letra minúscula. Es un mapeo de 1 o más elementos de un conjunto llamado dominio a un valor único en otro conjunto llamado rango. De aquí nace el concepto denominado como **cardinalidad**, lo cual consiste en el número de elementos del dominio que son mapeados. Por ejemplo:

  padre, color, hijo

- Expresión de función.

  Símbolo de función seguido por sus argumentos, los cuales van entre paréntesis y separados por una cuma (,).

  Los argumentos son los elementos del dominio y su cantidad corresponde a la cardinalidad.

  padre(maria), color(manzana), hijo(juan, ana)

- Valor de función.

  Objeto único del rango al que se mapea la función con sus argumentos.

- Evaluación

padre(maria)	juan
color(manzana)	rojo
hijo(juan, ana)	maria

7.2.2. Predicado

Símbolo que comienza con letra minúscula. Especifica relación entre objetos:

mujer gusta hijo_de parte_de pais

Cardinalidad: números de objetos relacionados

Sentencia atómica.

Predicado de cardinalidad n, seguido por n términos entre paréntesis, y separados por comas.

Valor de verdad: Verdadero - Falso.

También lo son los valores de verdad: verdadero & falso.
Sentencia atómica = expresión atómica = átomo
Ejemplo:

mujer(ana)
hombre(padre(tiene))
color(manzana, rojo)
gusta(pillin, hueso)
hijo_de(fernando, maria)
parte_de(motor, automovil)
pais(chile)
nacionalidad(pele, brasileño)

Un símbolo de predicado se puede utilizar con diferentes números de argumentos representando 2 relaciones diferentes. Una relación de predicado está definida por su nombre y su cardinalidad.

Sentencias en lógica de predicado:

- Sentencia atómica
- Combinación de sentencias atómicas con operadores lógicos
  - $\vee$
  - $\wedge$
  - $\neg$
  - $=$
  - $\rightarrow$
  - $\equiv$

Valor de verdad: verdadero-falso

Ejemplo:

amigos(padre(juan), profesor(rene))
fruta(manzana) $\wedge$ verdura(lechuga)
mes(junio) $\vee$ dia(septiembre)
ambulancia(xz_4522) $\rightarrow$ color(xz_4522, blanco)

7.2.3. Cuantificadores.

Cuantificador Universal $\forall$.

La sentencia es verdadera para cada individuo del mundo (conjunción infinita). Como por ejemplo, las afirmaciones:

- 'todos los gatos son mamíferos':

$$\forall \text{ x gato(X)} \rightarrow \text{mamifero(X)}$$

- 'Todos los alumnos que cursan IA son inteligentes':

$$\forall \text{ X cursa(X,ia)} \rightarrow \text{inteligente(X)}$$

  Esto tiene otro significado puesto que refiere que 'Todos cursan IA y son inteligentes'

$$\forall \text{ X cursa(X,ia)} \wedge \text{inteligente(X)}$$

Cuantificador Existencial $\exists$.

La sentencia es verdadera si es verdadera para algún individuo del mundo o del universo que se menciona (disyunción infinita). Como por ejemplo:

- 'Algunos gatos son de color blanco':

$$\exists \text{ x gato(X)} \wedge \text{color(X, blanco)}$$

- 'Mancha tiene una hermana que es gato':

$$\exists \text{ Z hermana(Z, mancha)} \wedge \text{gato(Z)}$$

- 'Algunos alumnos que cursan IA son inteligentes'

$$\exists \text{ X cursa(X,ai)} \wedge \text{inteligente(X)}$$

Sin embargo, existe otro significado el cual abarca a aquellos alumnos que no cursan IA, pero donde la afirmación será verdadera

$$\exists \text{ X cursa(X,ai)} \rightarrow \text{inteligente(X)}$$

Cuantificador Anidados.

- 'Todos somos buenos para alguna cosa'

$$\forall \text{ X } \exists \text{ Y buenopara(X,Y)}$$

- 'Existe algo para lo que todos son buenos'

$$\exists \text{ X } \forall \text{ Y buenopara(X,Y)}$$

Equivalencias

- $\forall X \neg Y \iff \neg \exists X \text{ P}$
- $\neg \forall X \text{ P} \iff \exists X \neg P$
- $\forall X \text{ P} \iff \neg \exists X \neg P$
- $\exists X \text{ P} \iff \neg \forall X \neg P$

Igualdad.

Con frecuencia el símbolo de igualdad (=) se incluye como un símbolo especial. El símbolo de relación binaria ordinaria corresponde:

$$\text{padre(juan)} = \text{jose}$$

La igualdad puede ser usada para decir que hay dos o más individuoscon una propiedad particular.

- 'Hay un(a) X y un(a) Y que son hermanas de Mancha y no son el mismo individuo'.

$$\exists X, \text{ Y hermana(mancha,X)} \wedge \text{hermana(mancha,Y)} \wedge \neg(X{=}Y)$$

El símbolo de igualdad también puede ser usado para restringir el número de objetos que tienen cierta propiedad.

- 'Todo par de objetos con la propiedad p son iguales'

$$\forall X,Y \text{ p(X)} \wedge \text{p(Y)} \iff X = Y$$

Esta afirmación los restringe a ser un objeto con la propiedad P.

7.2.4. Interpretación

Dominio + Posible forma en que puede ser el mundo

Asignación de entidades de dominio D a cada una de las constantes y variables, con un interpretación para las funciones y los predicados. Dicho eso, se obtiene un valor de verdad para la sentencia. Por ejemplo, si:

$$\mathbf{p(a)}$$

- Interpretación.

  p(X): X es un animal - a: ardilla $\Longrightarrow$ TRUE

- Interpretación.

  p(X): X es un color - a: domingo $\Longrightarrow$ FALSE

$$\forall\ \mathbf{X}\ \mathbf{p(X)}$$

- Interpretación.

  p(X): X es un número - Dom(X): números $\Longrightarrow$ TRUE

- Interpretación.

  p(X): X es un número par - Dom(X): números $\Longrightarrow$ FALSE

- Interpretación.

  p(X): X es un número par - Dom(X): números pares $\Longrightarrow$ TRUE

$$\exists\ \mathbf{X}\ \mathbf{p(X)}$$

- Interpretación.

  p(X): X es un número - Dom(X): números $\Longrightarrow$ TRUE

- Interpretación.

  p(X): X es un número par - Dom(X): números $\Longrightarrow$ TRUE

- Interpretación.

  p(X): X es un felino - Dom(X): aves $\Longrightarrow$ FALSE

Dada una sentencia S, ¿Se satisface?. ¿Existe una intepretación con la cual es verdadera?.

Valida: Se satisface con toda interpretación

Sentencia S es verdadera para toda interpretación. Una interpretación no se satisface en caso que se encuentren inconsistencias en ella. Por ejemplo, haciendo contrastes entre sentancias válidas y no válidas:

- Válida.

  $\forall$ X p(X) $\vee\neg$p(X)

- Inconsistente

  $\exists$ X p(X) $\wedge\neg$p(X)

7.3. Forward Chaining

Modus ponens(MP)
$p \Longrightarrow q$ q $------$ q
Modus tolens(MT)
$p \Longrightarrow \neg q$ $\neg q$ $------$ $\neg q$
Eliminación de AND(EA)
$p \wedge q$ $------$ p q
Introducción de AND(IA)
p q $------$ $p \wedge q$
Instanciar Cuantificador Universal (ICU)
$\forall X\ p(X)$ $a \exists Dom(X)$ $------$ $p(a)$

Tabla 9: Reglas de Inferencia - Forward Chaining

7.4. Backward Chaining

8. Deep Learning

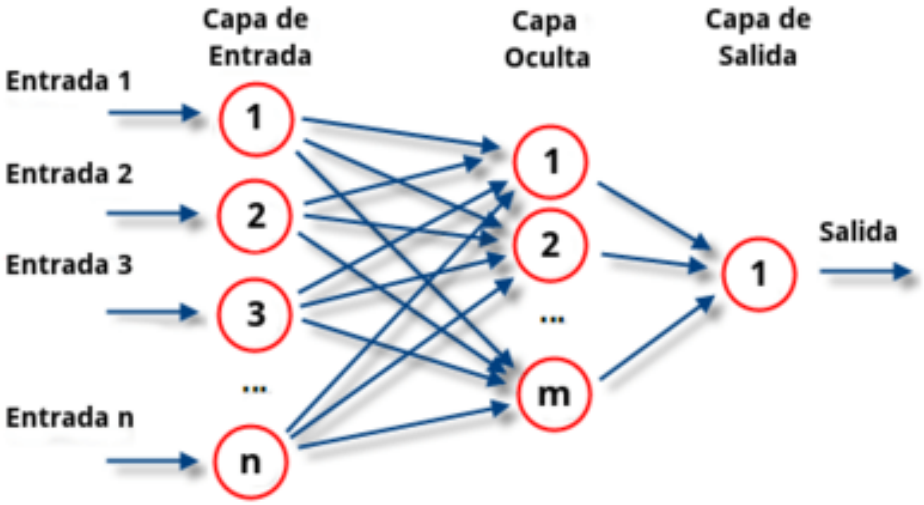
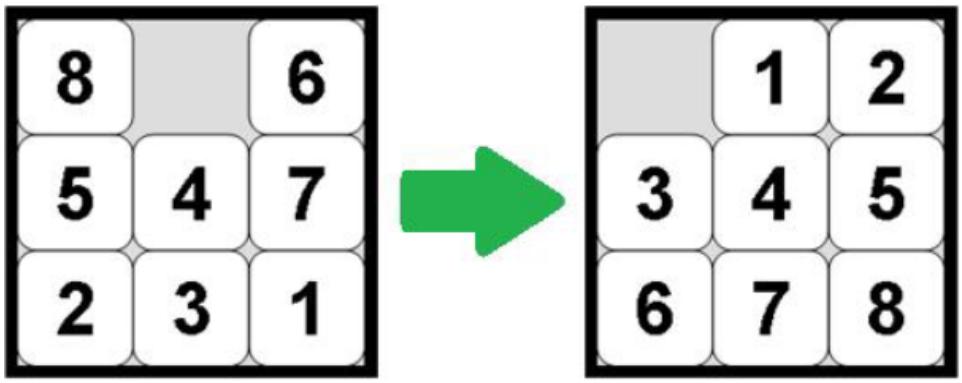


Figura 20: Red de Neuronas artificiales

9. Ejemplos.

9.1. Espacio de Estado

9.1.1. Puzzle de 8 piezas



A B

- Representación:
  - Se utiliza una matriz de 3x3, donde cada elemento para (i,j) va a contener uno de los números o el blanco
  - Estado inicial = A
  - Estado final = B
  - Estado cualquiera = Misma representación
- Operadores

1	2	6
7	3	
8	4	5

1,1	1,2	1,3
2,1	2,2	2,3
3,1	3,2	3,3

Opciones máximas = 4 min = 2

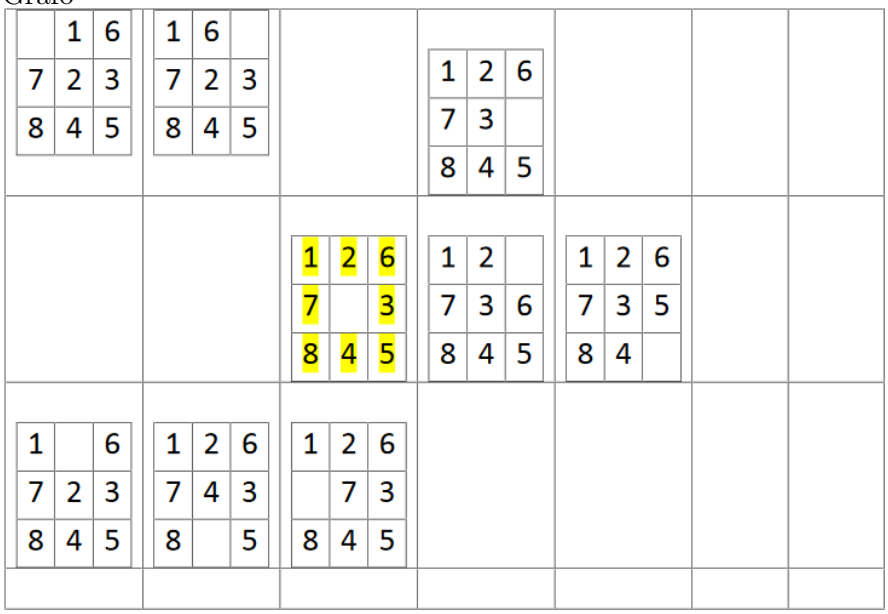
Subir(i,j)
cond=’arriba está el blanco’
Blanco en posición(i-1,j)
Nuevo Estado = Estado actual
Intercambio elemento (i,j) con el de arriba

Para bajar se realiza lo mismo, pero el blanco se sitúa en (i+1, j)

mov_derecha(i,j)
cond=’blanco está a la derecha’
blanco en posición (i,j+1)
Nuevo Estado = Estado actual
Intercambio elemento (i,j) con el de la derecha

Para mov_izq se realiza lo mismo, pero el blanco están en la posición (i, j-1)
tipo de solución = secuencia de operadores

- Grafo



$B_{max} = 4$

9.1.2. 1 Balde de agua y 2 botellas

2 Baldes - 2 botellas

- Capacidad
  - Grande 5
  - Chica 2
- Representación
  - x: representación de la cantidad de agua del B grande
  - y: representación de la cantidad de agua del B chico
  - Estado inicial = (5,2)
  - Estado final = (0,1)
  - Estado cualquiera = (a,b), con a en rango [0,5] y b en rango [0,2]
  - Ejemplo:
    (0,2)(1,1)(0,0)
- Operadores
  - ¿Cuántos?¿Cuáles? 4
  - Estado actual (a,b)
    Vaciar B grande
    Cond a >0
    Nuevo Estado = (0,b)

    Vaciar B chico
    Cond b >0
    Nuevo estado = (a,0)

Trasp de grande a chico
    Cond a>0 & b<2
    Nuevo estado
        if(a+b)<2
            Nuevo estado = (0,a+b)
        else
            Nuevo estado = (a+b-2,2)

Trasp de chico a grande
    Cond a<5 & b>2
    Nuevo estado
        if(a+b)<5
            Nuevo estado = (a+b,0)
        else
            Nuevo estado = (5,a+b-5)

$b_{max} = 4$

- Rango = [0,4], puede ser 0 si estado es (0,0)
- tipo de solución = secuencia de operadores!!

9.1.3. Tablero N damas

Ubicar las N damas tal que no se ataquen entre si. A modo de ejemplo, pensar en un tablero de N=4.

- Representación:
    - Matriz NxN
- Estados:
    - Estado inicial: Matriz vacía
    - Estado final:
        - La matríz de NxN con las N damas ubicadas
        - Que las N damas no se ataquen entre si.
    - Estado cualquiera:

- Matriz NxN con M damas ubicadas
  $0 < m < N(4) \implies$ hay estados con N(4) que son estados cualquiera, ya que no son final porque las damas se atacan entre si.
  $0 \leq m \leq N(4) \implies$ incluye todos (estado inicial y final también)

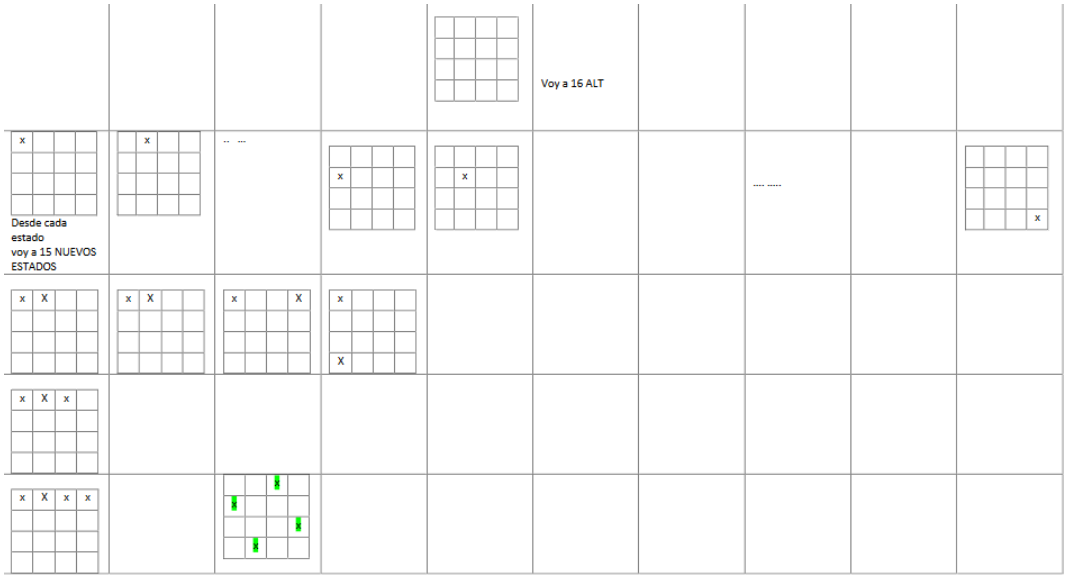
- Operadores:
  - Ubicar dama (i,j)
    Cond: Pos(i,j) debe estar vacía & aún faltan damas
  - Nuevo estado:
    Tomar la matriz y agregar la dama a posición (i,j)

- Tipo de solución
  - Es el estado final (No el camino)
  - ¿Como saber si un estado es o no final (solución)?
    - Que contenga N damas
    - Que las damas no se ataquen entre si

- Importante:
  Revisar si se atacan entre si o revisar si ataca una cierta posición **NO SON OPERADORES**, son funcionalidades requeridas para sabersi un estado es o no objetivo.

- Factor de ramificación (b):
  1er Nivel = NxN = 16, cada siguiente nivel es 1 menos que el anterior

- Nivel donde está la solución (d): N, es decir 4



- **Otra alternativa**
  - Operador:
    Ubicar_Dama(i,j)
    Se tiene matriz
    Cond: Pos(i,j) vacía & Aún faltan damas y la nueva dama en posición (i,j) no se ataca con otras que ya están en la matriz.
  - Nuevo estado:
    Tomar Matriz y agregar dama a posición (i,j)
  - d=4(N)
  - b (primer nivel) = 16(NxX)
    En los siguientes niveles disminuye (más que en 1), y no en una razón conocida, depende caso a caso.
  - Estado final = 4 damas colocadas (en este ejemplo N=4)
- **Una Tercera alternativa**
  Comenzar con una matriz con damas
  - Operador:
    Mover_dama(cual, a_donde) (i,j)Inicial (i,j)Destino
    B en el ejemplo 12x4

- Otra forma de operador:
  Mover_dama_en_su_Columna(origen,destino) (i,j_inicial+filadestino)

9.1.4. Caníbales y misioneros

- Propuesta inicial de representación (a,b,c)(x,y,z)
  (M,C)
  (3,3)(0,0)
  (1,1)(2,2)

- Observación: Dónde está el bote?
  5 datos
  (r (1,1)(2,2))
  (a (1,1)(2,2))
  (b (1,1)(2,2))
  (a 1,1,2,2)

- Estados
  (r mA cA mB cB)
  - 1ro - Orilla A o B
  - 2do - Cantidad de misioneros en A
  - 3ro - Cantidad de canibales en A
  - 4to - Cantidad de misioneros en B
  - 5to - Cantidad de Canibales en B
  - inicial (b 0 0 0 3 3)
  - final (a 3 3 0 0)
  - Cualquiera:
    - (a/b x y z w)
    - x+z = 3
    - y+w = 3

- si $x > 0 \Longrightarrow x \geq y$
- si $z > 0 \Longrightarrow z \geq w$

- Viajar 1 M de A a B:
  - Cond: orilla = a
  - $x - 1 > 0 \Longrightarrow x - 1 \geq y$
  - $z + 1 > 0 \Longrightarrow z + 1 \geq w$
  - Nuevo estado:
    (b x-1 y z+1 w)

- Operadores:
  - Viaja 1 M de A a B
  - Viaja 1 C de A a B
  - Viajan 2 M de A a B
  - Viajan 2 C de A a B
  - Viajan 1 M y 1 C de A a B
  - 5 operadores más relacionados al viaje de B a A

9.1.5. Grandero + Lobo + Cabra + Repollo

- Representación
  - Tupla de 4 elementos
  - Las orillas del río son:
    - A-B
    - Este - Oeste
    - E-W
  - (G L C R): En que orilla está cada uno de ellos
- Estado
  - Estado inicial: (a a a a)
  - Estado final: (b b b b)
  - Estado cualquiera: (b a b a)
- Operadores
  - (a a a a)

- (b a b a)
- (a a a a) (a a b a)

- Operaciones
  Siendo estado actual: (posG posL posC posR)

  - Cruzar A-B granjero solo
    Cond: posG = a
    y que estado nuevo quede 'bien'
    y si posC=a entonces posL<>a
    y si posC=a entonces porR<>a

  - Cruzar A-B granjero con lobo
    Cond: posG = a y posL = a
    y que estado nuevo quede 'bien'
    y si posC<>posR
    Nuevo_estado: (B B posC posR)

  - Cruzar A-B granjero con cabra
  - Cruzar A-B granjero con repollo
  - Total operaciones = 8
    En cada estado a lomás se pueden aplicar 4
    b = entre 1 y 4 en todo nivel
    d(nivel de solución) = no se conoce, pero existe d_{max}
  - 4 de (A-B) y 4 de (B-A)

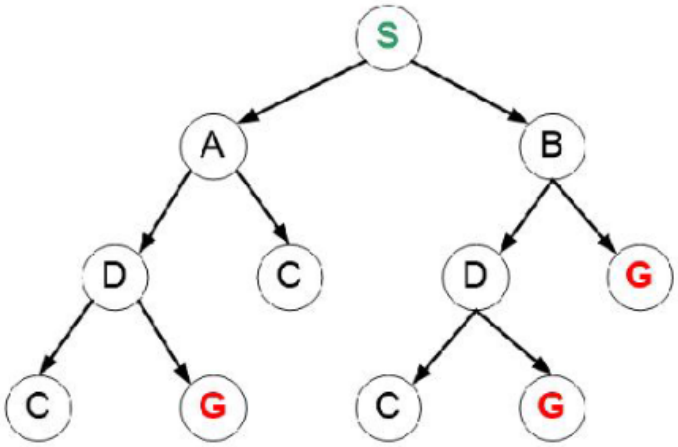
- Observación
  Se pueden tener 4 operadores:

  - Viajar solo
  - Viajar con lobo
  - Viajar con cabra
  - Viajar con repollo

  Siempre se parte donde está el granjero

9.2. Búsqueda y análisis

9.2.1. Ejemplo 1



- BFS

Q(por visitar)	**Visitados**
{**S**}	{}
{ **A(s)**, B(s) }	{ S- }
{**B(s)** ,D(a) ,C(a) }	{S-, A(s)}
{**D(a)**, C(a), D(b), G(b) }	{S-, A(s), B(s)}
{**C(a)**, D(b), G(b), C(d), G(d) }	{S-, A(s), B(s), D(a)}
{ D(b), G(b), C(d), G(d) }	{S-, A(s), B(s), D(a), C(a)}
{ G(b), C(d), G(d), C(d), G(d) }	{S-, A(s), B(s), D(a)}
OBJETIVO ENCONTRADO	

Tabla 10: BFS - Ejemplo 1

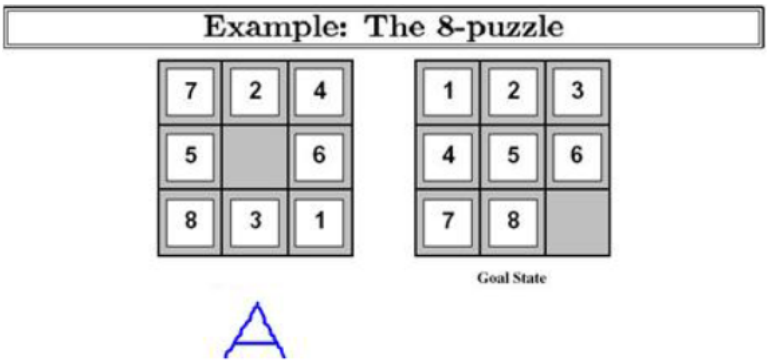
- DFS

Q(por visitar)	Visitados
{**S**}	{}
{ **A(s)**, B(s) }	{ S- }
{**D(a)**, C(a), B(s) }	{S-, A(s)}
{**C(d)**, G(d), C(a), B(s) }	{S-, A(s), D(a)}
{ G(d), C(a), B(s)}	{S-, A(s), D(a), C(d)}
OBJETIVO ENCONTRADO	

Tabla 11: DFS - Ejemplo 2

9.3. Heurística

9.3.1. The 8 puzzle



- Tile es el nombre de cada número/ficha
- Propuesta de heurística 1:
  Contar cuantos tiles están úbicados h(a) = 2, así se obtiene un valor asociado al estado, pero no es una buena heurística, no nos dice que tan cerca estamos del objetivo.
- Propuesta de heurística 2:
  Contar cuántos tiles están fuera de posición objetivo, ¿Contamos o no el blanco?
  - Estado B

1	2	3
4	5	6
7		8

- h(B) = 1 (sin contar blanco)
- h(B) = 2 (contando blanco)
- Respuesta:
  No, así en este estado cercano al objetivo, da como respuesta 1, que es exactamente la cantidad de movimientos necesarios para llegar el objetivo
- h(A) = 6
- Se tiene un valor asociado al estado que representa la cantidad mínima de movimiento necesarios para llegar al objetivo, se suma 1 por cada tile mal ubicado

- Propuesta de heurística 3:
  Manhattan Distance
  - $1 -> 4$
  - $2 -> 0$
  - $3 -> 3$
  - $4 -> 3$
  - $5 -> 1$
  - $6 -> 0$
  - $7 -> 3$
  - $8 -> 1$
  - Suma = 15
  - $h(A) -> 15$

9.4. Logica de predicado

Cinco amigos compiten en una carrera, en la que no hubo empates. Cada uno declara lo siguiente:

- Tito: 'Juan llegó primero, y yo segundo'
- Diego: 'Juan llegó segundo y yo cuarto'
- Pato: 'Santiago llegó quinto y yo tercero'
- Juan: 'Pato llegó primero y yo quinto'
- Santiago: 'Juan llegó tercero y yo cuarto'

Cada uno dijo al menos una verdad. ¿En qué orden llegaron?.

Resolución.

p1: Juan 1	p2: Tito 2
p3: Juan 2	p4: Diego 4
p5: Santiago 5	p6: Pato 3
p7: Pato 1	p8: Juan 5
p9: Juan 3	p10: Santiago 4

Considerando que cada uno dijo al menos una verdad, es o son verdad:

$$p1 \vee p2$$
$$p3 \vee p4$$
$$p5 \vee p6$$
$$p7 \vee p8$$
$$p9 \vee p10$$

p1: Juan 1 $\rightarrow$ Juan no llegó en otro lugar.

'No hubo empates"$\rightarrow$ Nadie más llegó en ese lugar (1ero) entonces también es verdadero:

$$p1 \rightarrow \neg p7 \wedge \neg p8 \wedge \neg p9$$
$$p2 \rightarrow \neg p3$$
$$\ldots$$
$$p10 \rightarrow \neg p5 \wedge \neg p8$$

Tabla de verdad

1	Pato
2	Tito
3	Juan
4	Diego
5	Santiago

10. Glosario

1. Algoritmo A*.

   El Algoritmo A* es un algoritmo de búsqueda que puede ser empleado para el cálculo de caminos mínimos en una red. Se va a tratar de un algoritmo heurístico, ya que una de sus principales características es que hará uso de una función de evaluación heurística, mediante la cual etiquetará los diferentes nodos de la red y que servirá para determinar la probabilidad de dichos nodos de pertenecer al camino óptimo.

   Esta función de evaluación que etiquetará los nodos de la red estará compuesta a su vez por otras dos funciones.

   - Una de ellas indicará la distancia actual desde el nodo origen hasta el nodo a etiquetar.
   - La otra expresará la distancia estimada desde este nodo a etiquetar hasta el nodo destino hasta el que se pretende encontrar un camino mínimo

   Si se pretende encontrar el camino más corto desde el nodo origen **S**, hasta el nodo destino **T**, un nodo intermedio de la red **N** tendría la siguiente función de evaluación F(N) como etiqueta:

   $$F(N) = G(N) + H(N) \tag{10}$$

   Donde:

   - **G(N)**: Indica la distancia del camino desde el nodo origen S al N.
   - **H(N)**: Expresa la distancia estimada desde el nodo N hasta el nodo destino T.

   H(N) se trata de una función heurística, expresa la idea de cúan lejos aún se está de alcanzar el nodo destino, y de su correcta elección dependerá en gran medida el rendimiento del algoritmo A* al aplicarlo en una red. Así, en el caso de que esta función heurística nunca sobreestime el valor de la distancia real entre el nodo y el destino, se dice que es una **heurística admisible**, y está garantizada la solución óptima.

   A la función de evaluación F que caracteriza a un nodo y que sirve para etiquetarlo, también se la conoce como mérito de ese nodo, y expresa la probabilidad del nodo de estar en el camino más corto. Cuanto

menor sea el mérito de un nodo, es decir, cuanto menor sea el valor de su función de evaluación, más probable será que el camino más corto atraviese ese nodo. En este mérito de los diferentes nodos se basará el fundamento del Algoritmo A*.

METODOLOGÍA DE BÚSQUEDA

El algoritmo irá explorando nodos de la red y sus sucesores (aquellos nodos con lo que les une algún enlace) basándose en su valor de mérito. Es decir, mantendrá una lista ordenada por valor creciente de mérito de los nodos que pueden ser explorados, y de ahí seleccionará el de menor valor, que será el primero de la lista. El algoritmo empezará analizando el nodo que se toma como origen para el problema del camino más corto. Calculará su mérito, y a continuación pasará a explorar sus nodos sucesores, es decir, los nodos con los que eté unido por un enlace este nodo fuente. Para estos nodos se calculará su mérito, y se seleccionará aquél de ellos que presente un menor valor, que será el que se someterá al análisis. Ahora el algoritmo continuará su ejecución explorando los nodos vecinos de ese nodo con merito menor, y así sucesivamente, hasta llegar al caso donde el nodo a analizar sea el nodo destinto del problema, momento en que el algoritmo termina y ya se dispone de la solución.

Para llevar a cabo este funcionamiento será necesario un registro donde el algoritmo irá guardando el conjunto de nodos que han sido explorados con sus valores correspondientes de méritos de mérito, y de donde se irán eliminando aquellos que sean seleccionados para ser analizados.

1.*Establecer el nodo S como origen. Hacer* $f(s) = 0$ *y* $f(i) = \infty$ *para todos i diferentes del nodo s. Iniciar el conjunto Q vacío.*

2.*Calcular el valor de* $f(S)$ *y mover el nodo S al conjunto Q.*

3.*Seleccionar el nodo i del conjunto Q que presente menor valor de la función* $f(i)$ *y eliminarlo del conjunto Q.*

4.*Analizar los nodos vecinos j de i. Para cada enlace* (i, j) *con coste* c_{ij} *hacer:*

a) Calcular: $f'(j) = g(i) + c_{ij} + h(j)$

b) Si $f(j)' < f(j)$

- Actualizar la etiqueta de j y su nuevo valor será: $f(j) = g(i) + c_{ij} + h(i)$
- Insertar el nodo j con Q

c) Si $f(j)' > f(j)$

- Dejar la etiqueta de j como está, con su valor $f(j)$

5.*Si Q está vacío el algoritmo se termina. Si no está vacío, volver al paso 3.*

2. Algoritmos de búsqueda Local

Búsqueda local es la base de muchos de los métodos usados en problemas de optimización. Se puede ver como un proceso iterativo que empieza en una solución y la mejora realizando modificaciones locales. Básicamente empieza con una solución inicial y busca en su vecindad por una mejor solución. Si la encuentra, reemplaza su solución actual por la nueva y continua con el proceso, hasta que no se puede mejorar la solución actual.

Procedimiento Búsqueda Local:

s = genera una solución inicial
while *s no es óptimo local* ***do***
s′ ∈ N(s) con f(s) < f(s′)
(solución mejor dentro de la vecindad de s)
s ← s'
end
return *s*

Claramente el diseño de la vecindad es crucial para el desempeño de éste, y de muchos otros algoritmos. La vecindad son **todas las posibles soluciones que se consideran en cada punto**. El cómo se busca la vecindad y cuál vecino se usa en el reemplazo a veces se conoce como regla de pivoteo (*pivoting rule*), que en general puede ser:

- Seleccionar el mejor primer vecino de todos (*best-improvement rule*).

- Seleccionar el primer vecino qu mejora la solución (*first-improvement rule*).

Búsqueda local tiene la ventaja de encontrar soluciones muy rápidamente. Su principal desventaja es un método determinístico y sin memoria. Dada una entrada, regresa la misma salida.

De hecho, un conjunto de entradas no generan la misma salida (mínimo local). Esto se puede ver como *pozo de atracción*. Esto sigue siendo muy superior a búsqueda aleatoria y el ingrediente principal es la definición de vecindad que permite moverse de una cierta solución a una mejor solución.

3. BackPropagation
4. Búsqueda informada
5. Búsqueda primero el mejor
6. Búsqueda voraz primero el mejor
7. Estado del arte
8. Feed Fordward
9. Función Heurística
10. Gradient Descent
11. Greedy Search
12. Hill Climbing
13. Machine Learning
14. Método de búsqueda local
15. Simulated annealing
16. Suma del error cuadrático
17. Tabu Search