Solución de Problemas Mediante Búsqueda (2)

Carlos Hurtado

Depto de Ciencias de la Computación,

Universidad de Chile

Recapitulando: Estrategias de Búsqueda

- Búsqueda de Costo Uniforme
  - Caso particular: Búsqueda Primero en Anchura
- Búsqueda Primero el Mejor
  - Caso particular: Búsqueda Primero en Profundidad

Algoritmo A*

- [Hart, Nilsson and Raphael, 1968]
- Combina aspectos de Búsqueda Costo Uniforme y Búsqueda Primero el Mejor
- Calidad de cada nodo en la frontera está dado por:

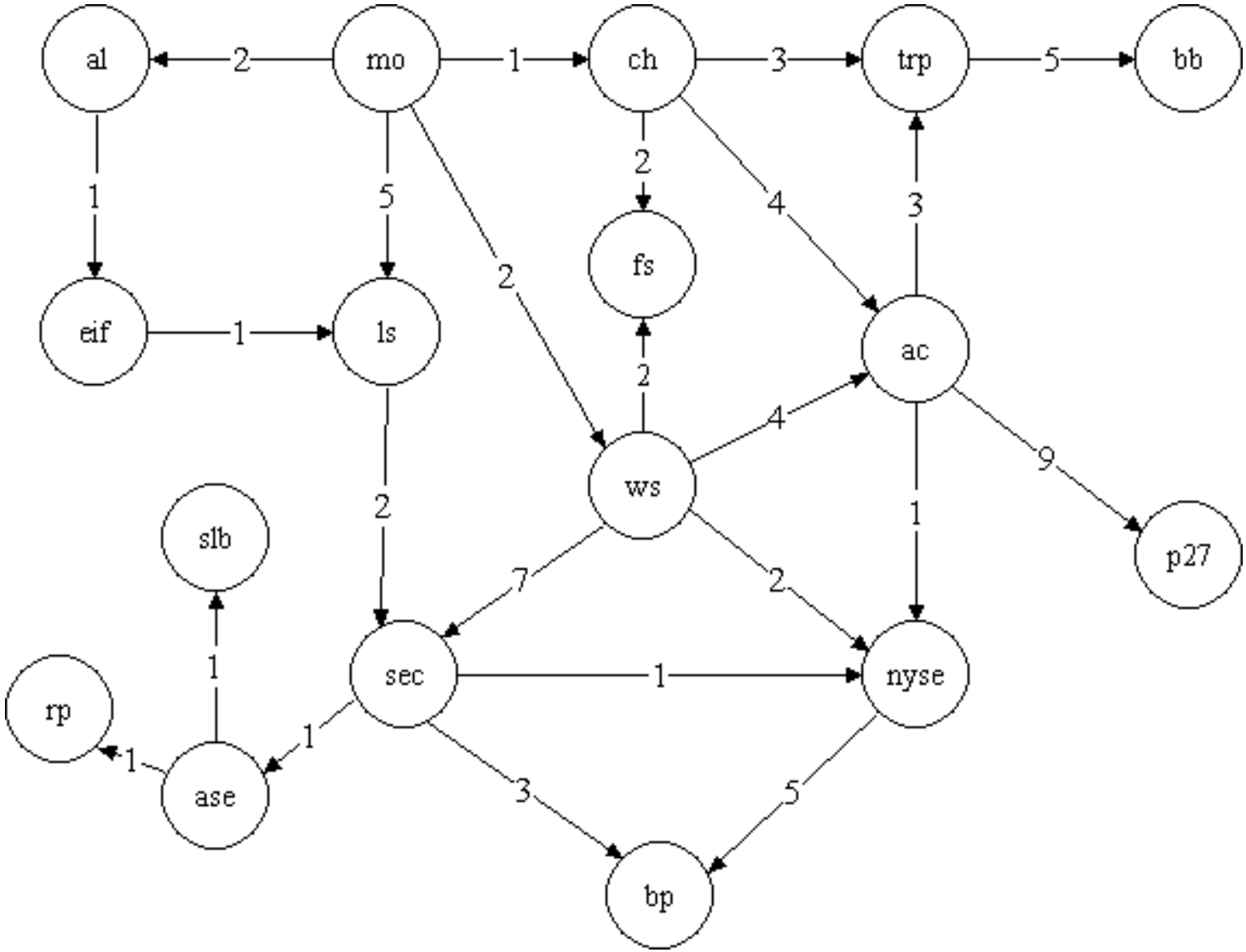
$$f(n) = g(n) + h'(n)$$

- Los nodos son ordenados en FRONTERA de menor a mayor f(n)

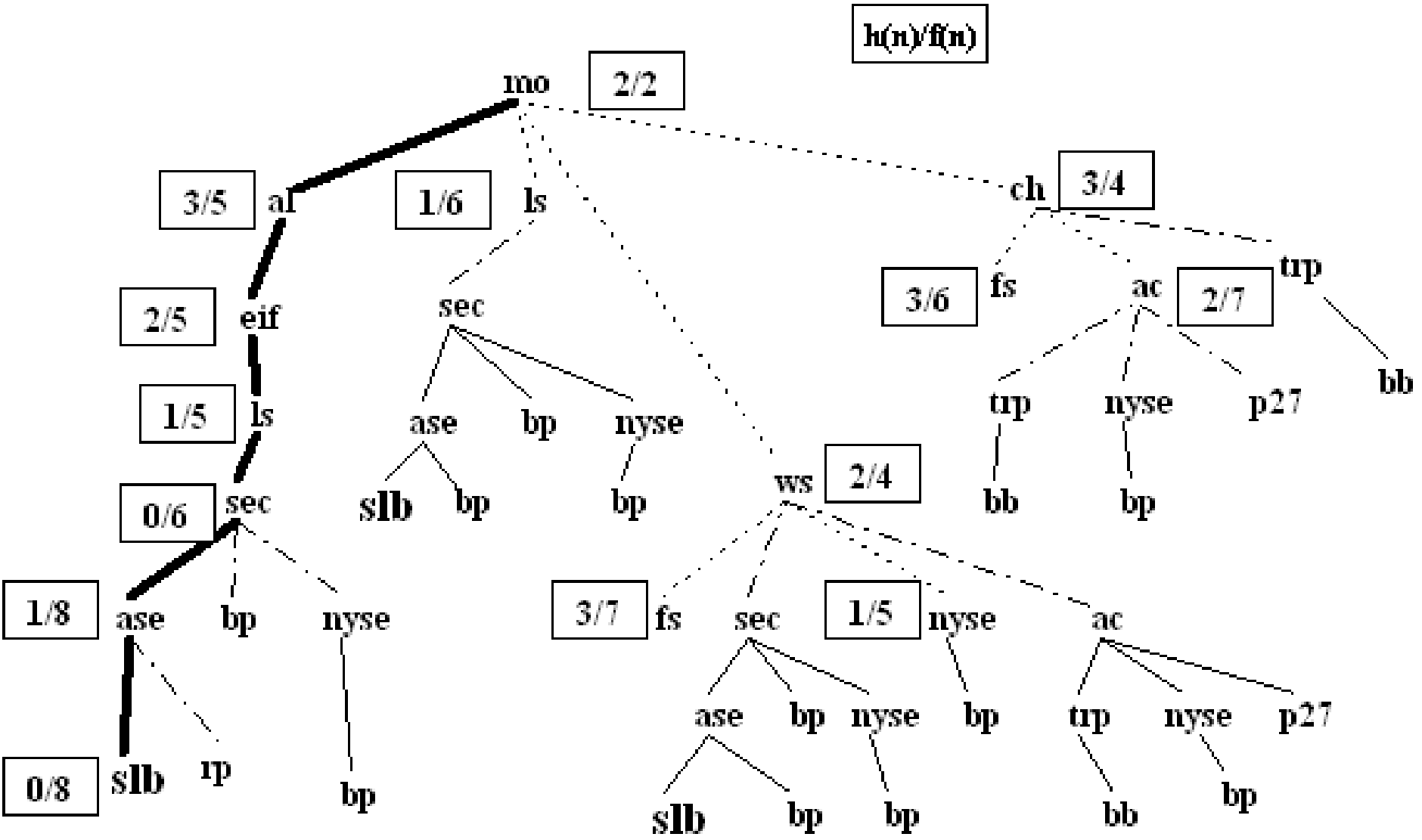
Algoritmo A*

- Input: (V,E), s, G
- Variables:
  - FRONTERA: lista de nodos a visitar, inicialmente s
  - Tr: árbol de búsqueda, originalmente contiene un nodo etiquetado con s
- While (FRONTERA no es vacío) do
  - Sacar primer nodo n de FRONTERA
  - Si n está en G entregar solución
  - Agregar al final de FRONTERA nodos P apuntados por n
  - **Para cada nodo agregado computar f(n) = g(n) + h'(n)**
  - Por cada nodo r en P, agregar un nuevo nodo a Tr, etiquetarlo con r y conectarlo desde el nodo de Tr asociado a n
  - ***Reordenar FRONTERA de acuerdo a función f***

Manhattan Bike Curier (Acíclico) Ref. Curso IA U. of Toronto



A*: mo - slb


h(n)/f(n)
mo
2/2
3/5
al
1/6
ls
ch
3/4
2/5
eif
sec
3/6
fs
ac
2/7
trp
1/5
ls
ase
bp
nyse
trp
nyse
p27
bb
0/6
sec
slb
bp
bp
ws
2/4
bb
bp
1/8
ase
bp
nyse
3/7
fs
sec
1/5
nyse
ac
0/8
slb
rp
bp
ase
bp
nyse
bp
trp
nyse
p27
slb
bp
bp
bb
bp

Análisis de A*

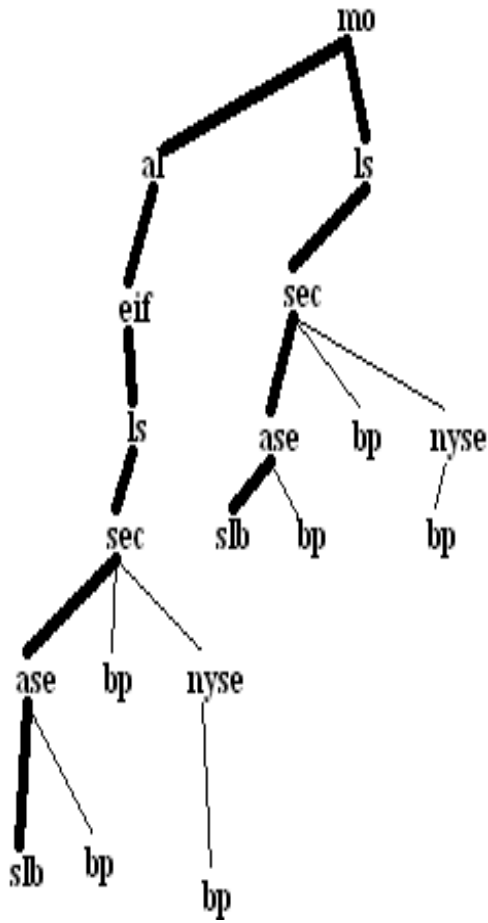
- En el ejemplo A* llega rápidamente al objetivo (slb)
  - Expande sólo 7 nodos que no llegan a la solución
- A* encuentra el camino de costo mínimo
- Combina lo mejor de
  - Búsqueda de Costo Uniforme: encuentra el mejor camino
  - Búsqueda Primero Mejor: se dirige rápidamente al objetivo

Propiedades de A*

- ¿A* siempre encuentra el mejor camino?

Propiedades de A*

- ¿A* siempre encuentra el mejor camino?
  - No necesariamente:
  - Supongamos que h(al) =17
  - El nodo al no será expandido antes de del camino por ls



Heurística Admisible

- **Teorema**: Supongamos que existe una solución (solución = camino de menor costo). Entonces A* la encuentra si:
  1. Cada nodo del grafo tiene un número finito de sucesores.
  2. El costo de cada nodo en el grafo es mayor que cero.
  3. Para todo nodo n, h’(n) <= h(n) (**heurística admisible**).

Heurística Admisible

- En el ejemplo del MBC la heurística usada es admisible
- Caso especial: h’(n) = 0
  - f(n) = g(n),
  - A* se reduce a búsqueda de costo uniforme
- En este caso la heurística es admisible pero no es informativa

Demostración

- La demostración del teorema consiste en probar dos condiciones:
  - Condición A: A* termina.
  - Condición B: El camino que retorna A* (luego de terminar) es el camino de costo mínimo.

Demostración Condición A

- Supongamos que se cumplen las condiciones 1, 2 y 3.
- Sea p* el camino óptimo.
- Luego de un número de iteraciones dada, f(n) aumenta.
- Esto por que los costos son positivos (condición 2).
- Luego, por condición 1 eventualmente en alguna iteración f(n) > costo(p*) y habremos llegado a un nodo objetivo.

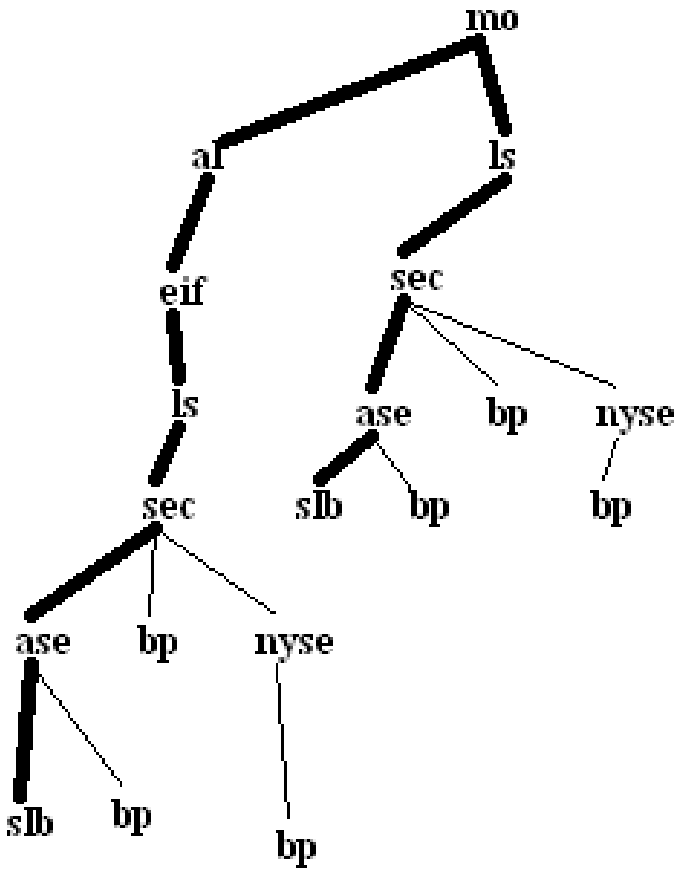
Demostración Condición B

- Supongamos que se cumple 1, 2 y 3.
- Sea p un camino subóptimo con costo c(p)
- Sea p* el camino óptimo
- Todo nodo t de p* tiene f(t)<= c(p*) <= c(p) (por condición 3)
- Por lo tanto antes de expandir el último nodo de p, expandiremos todos los nodos de p*
- Es decir, encontramos p* antes de p

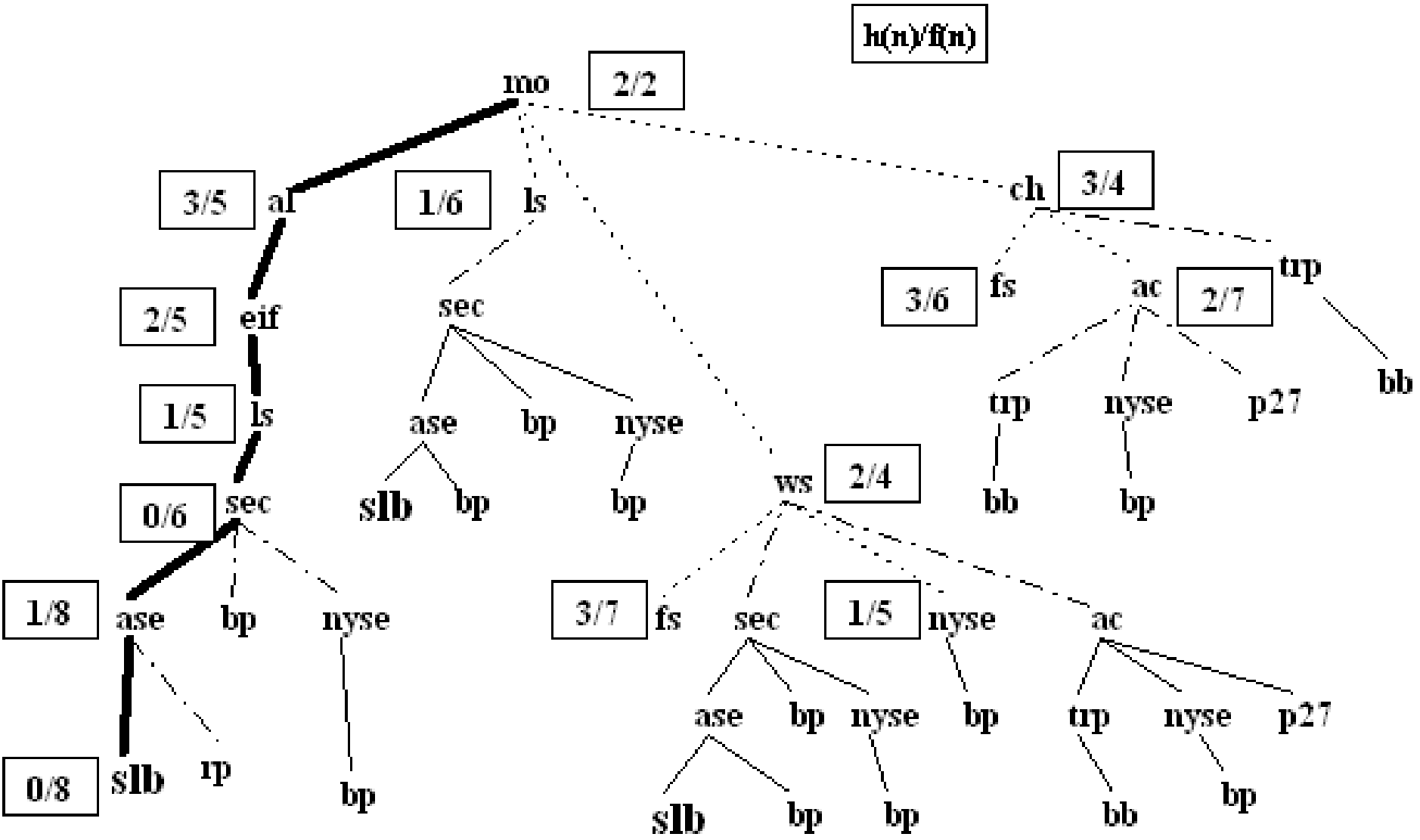
Expansiones Redundantes

- En A* podemos expandir más de una vez un mismo nodo

Ejemplo: para un h', ls se expandiría dos veces y sólo vale la pena hacerlo desde el camino más corto mo-ls



Expansiones redundantes: mo - slb



Expansiones Redundantes

- Una heurística h’ es **consistente** si para cada arco (u,v) del grafo, se cumple:

$$h'(u)-h'(v) <= c(u,v)$$

- **Teorema**: Si h’ es consistente, entonces la primera expansión de un nodo n genera (en el árbol de búsqueda) el camino óptimo desde el nodo inicial a n.

Ejercicio

- Demostrar el teorema anterior.

Comparación de dos Heurísticas

- Dadas dos versiones A1* y A2* de A* con heurísticas admisibles h1’ y h2’, respectivamente.
- **Teorema**: Si para todo nodo n, h2’(n)>h1’(n), entonces todo nodo expandido por A2* también será expandido por A1*.
- Intuición: mientras mejor la heurística h’ aproxime a h, menos nodos se expanden.

Complejidad de A*

- Si no hay expansiones redundantes A* toma tiempo en O(n^2)
  - Si la heurística es muy buena puede mejorar significativamente una búsqueda de costo uniforme.
- Si hay expansiones redundantes, en el peor caso, toma tiempo O(b^n), donde b es el factor de ramificación.

Uso de Heurísticas

- Si una heurística no es buena más vale la pena no usarla.
  - Es decir tenemos búsqueda de costo uniforme que me asegura encontrar el óptimo y evita nodos redundantes.

Búsqueda Genérica en Prolog

```
search( F ) :- select(Node, F, RemF), is_goal(Node).

search( F ) :- select(Node, F, RemF),
               nbs(Node, NbList),
               add_to_frontier(RemF, NbLists, NewF),
               search( NewF ).
```

Define nodos objetivo

Define grafo

Búsqueda Genérica en Prolog (sin extracción de caminos)

- La Frontera se maneja como una lista de nodos.
  - search(F) es verdadero si existe un camino de un nodo de F a un nodo en G
  - El algoritmo se invoca inicialmente como search([s])
- Select(N,F,RemF) es verdadero si al sacar el nodo N de la frontera esta se transforma en RemF.
- Add_to_frontier(RemF,NbList,NewF) es verdadero si al agregar los nodos NbList a RemF resulta en NewF.

Búsqueda Genérica en Prolog

```
search( F ) :- select(Node, F, RemF), is_goal(Node).

search( F ) :- select(Node, F, RemF),
               nbs(Node, NbList),
               add_to_frontier(RemF, NbLists, NewF),
               search( NewF ).
```

Define nodos objetivo

Define grafo

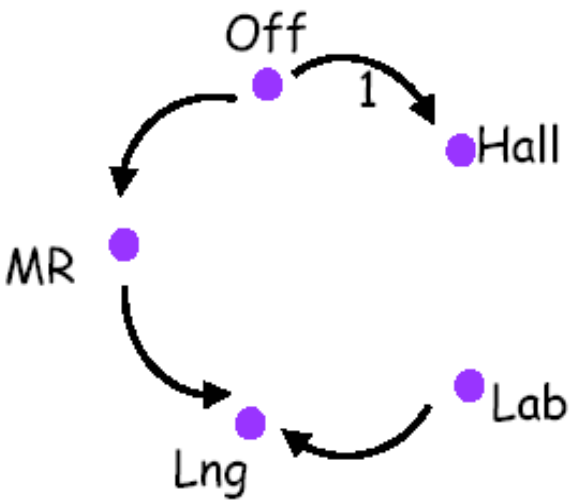
Búsqueda Genérica: Instanciación

```
search( F ) :- select(Node, F, RemF), is_goal(Node).
search( F ) :- select(Node, F, RemF), nbs(Node, NbList),
               add_to_frontier(RemF, NbList, NewF),
               search( NewF ).
```

```
select(Node, [Node|RemF], RemF).

add_tf (RemF, NbList, NewF) :-
                    append(NbList, RemF, NewF).
```

Ejemplo (modificado)



Graph:

```
nb(off,[mr,h]).
nb(h, []).
nb(mr, [lng]).
nb(lng, []).
nb(lab, [lng]).
```

Goal Node:

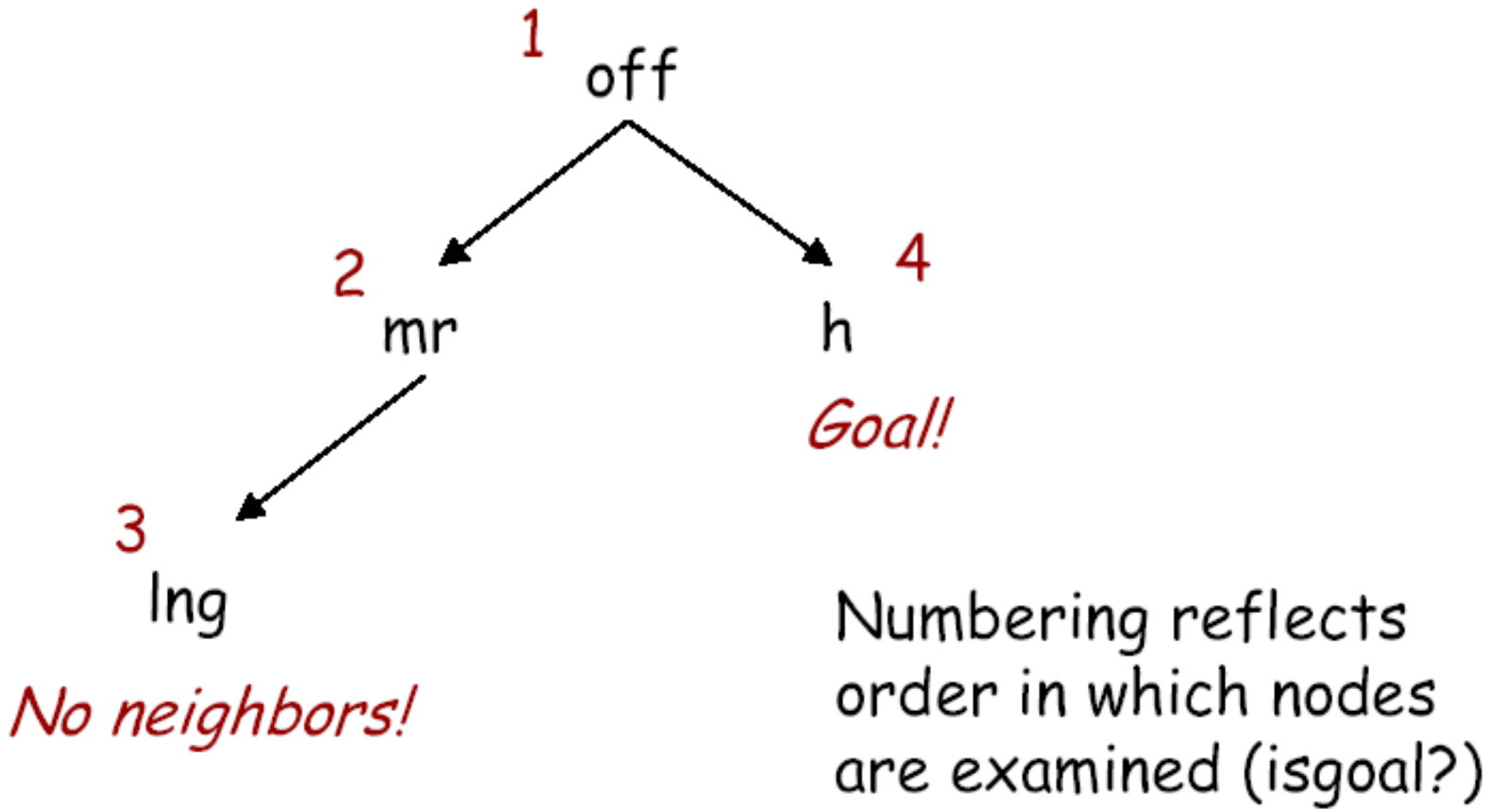
```
isgoal(h).
```

Start Node:

office

```
search([off]).
  select off, RF = []
  not a goal, NF = [mr, h]
search([mr, h]).
  select mr, RF = [h]
  not a goal, NF = [lng, h]
search([lng, h]).
  select lng, RF = [h]
  not a goal, NF = [h]
search([h]).
  select h, RF = []
  is a goal
Return yes!
```

Arbol de Búsqueda (ejemplo modificado)



What if we added an edge from lng to lab?

Extracción de Caminos

```
search( F ) :- select(Node, F, RemF), is_goal(Node).

search( F ) :- select(Node, F, RemF),
               nbs(Node, NbList),
               add_to_frontier(RemF, NbLists, NewF),
               search( NewF ).
```

Define nodos objetivo

Define grafo

- El procedimiento que hemos implementado no entrega el mejor camino.
- Si agregamos is_goal (bp) y preguntamos ? search([mo]), el procedimiento responde yes
- Pero no retorna el camino.

Extracción de Caminos

- Implementaremos el predicado search([[mo]],Path): verdadero si Path es un camino de mo a un nodo objetivo.
- Idea: la frontera es una lista de caminnos, los últimos nodos de los caminos son los nodos a expandir.

Extracción de Caminos en Búsqueda (cont.)

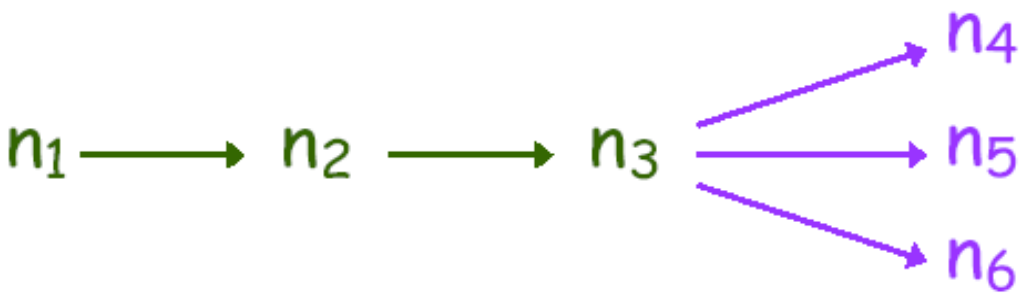
Assume a path is represented as a list of nodes in reverse order: so the path mo → al → eif is represented as the list: [eif, al, mo]

```
search( F, [Node | RestP] ) :-
    select( [Node | RestP] , F, RemF),
    is_goal(Node).

search( F, Path ) :-
    select( [Node | RestP] , F, RemF ),
    nbs(Node, NbList),
    extendpath(NbList, [Node | RestP], NewPaths ),
    add_to_frontier(RemF, NewPaths, NewF),
    search( NewF, Path ).
```

Extracción de Caminos en Búsqueda Genérica

- We call search from start node *mo* using
  search([[mo]], Path).
  - initial frontier consists of a length one *path* (not node)
- Only new predicate needed is *extendpath*
  - basic idea: given a path [n_3, n_2, n_1] and neighbors of n_3 specified by nbs(n_3, [n_4, n_5, n_6]); it produces 3 new paths organized in a list:
    [[n_4, n_3, n_2, n_1] , [n_5, n_3, n_2, n_1] , [n_6, n_3, n_2, n_1]]



Implementación de extendpath

```
extendpath(RNbs, Path, [[Nb | Path] | RNPaths]) :-
extendpath([Nb | RNbs], Path, RNPaths)

extendpath([],_,[]).
```

Búsqueda Primero en Profundidad con Extracción de Caminos

```
search( F, [Node | RestP] ) :-
    select([Node | RestP], F, RemF),
    is_goal(Node).

search( F, Path ) :-
    select( [Node | RestP], F, RemF ),
    nbs(Node, NbList),
    extendpath(NbList, [Node | RestP], NewPaths ),
    add_to_frontier(RemF, NewPaths, NewF),
    search( NewF, Path ).

select(Path, [Path|RestPaths], RestPaths).

add_tf (RemF, Paths, NewF) :-
                append(Paths, RemF, NewF).
```