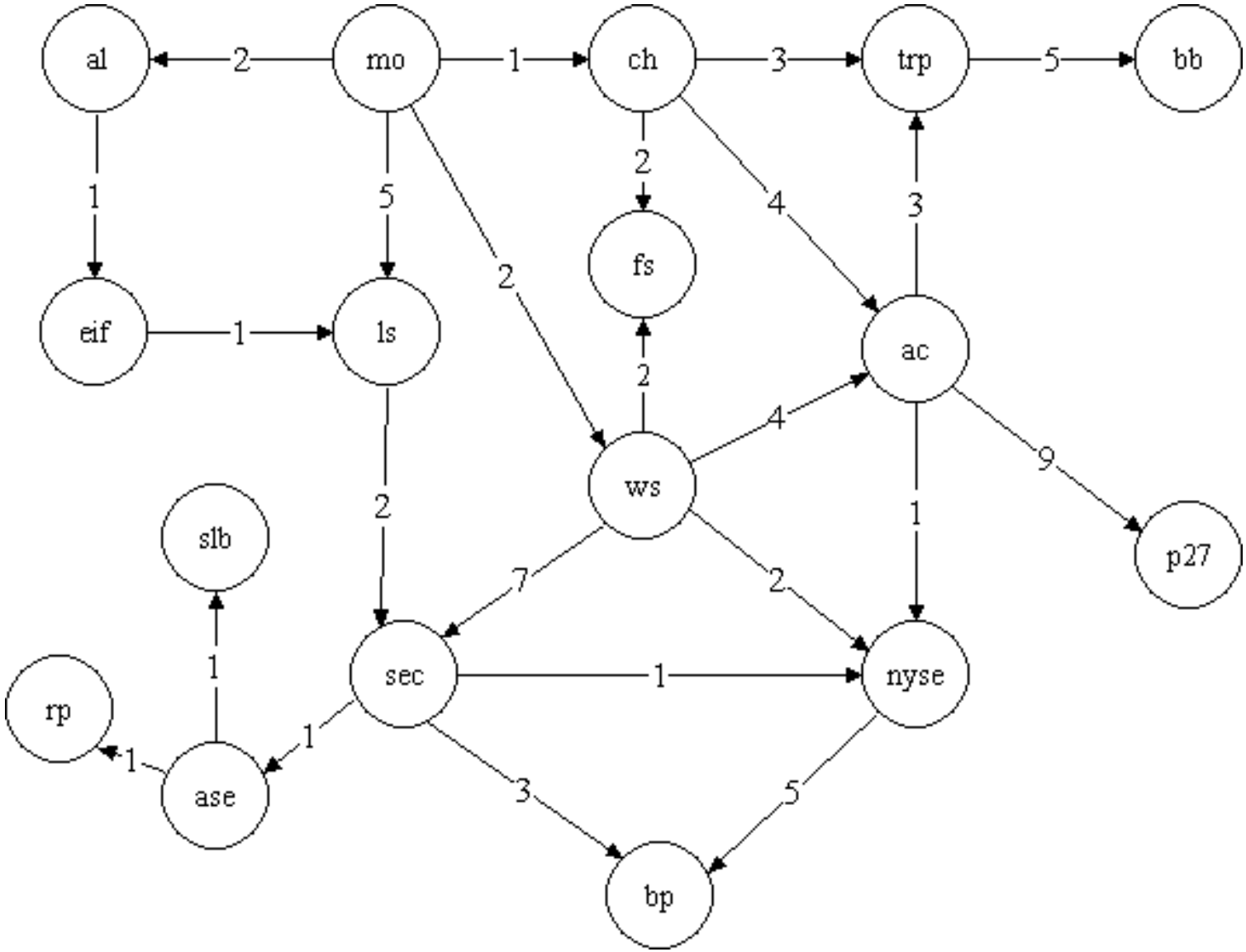
Solución de Problemas Mediante Búsqueda (2)

Carlos Hurtado
Depto de Ciencias de la Computación,
Universidad de Chile

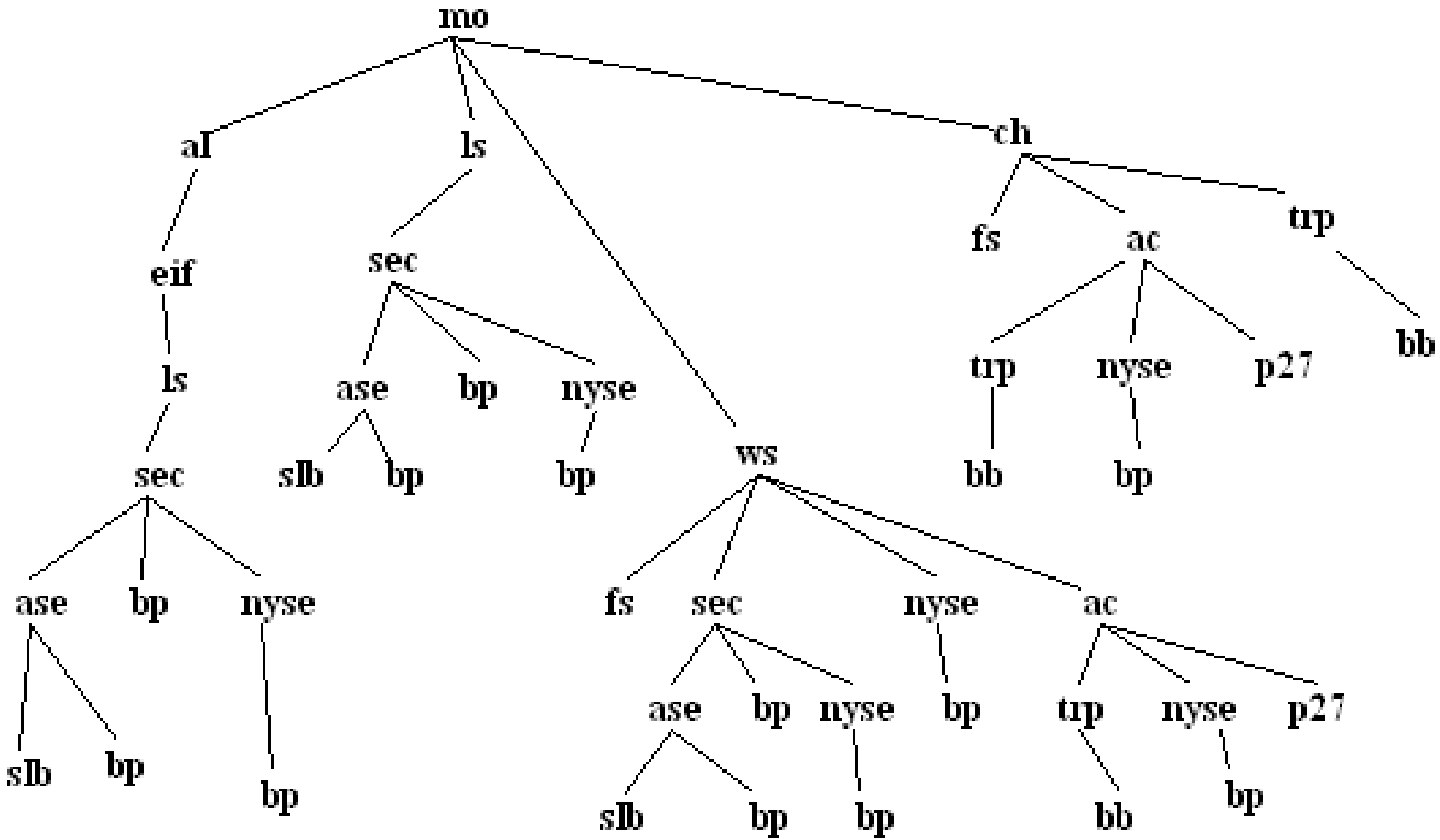
Manhattan Bike Curier (Acíclico) Ref. Curso IA U. of Toronto



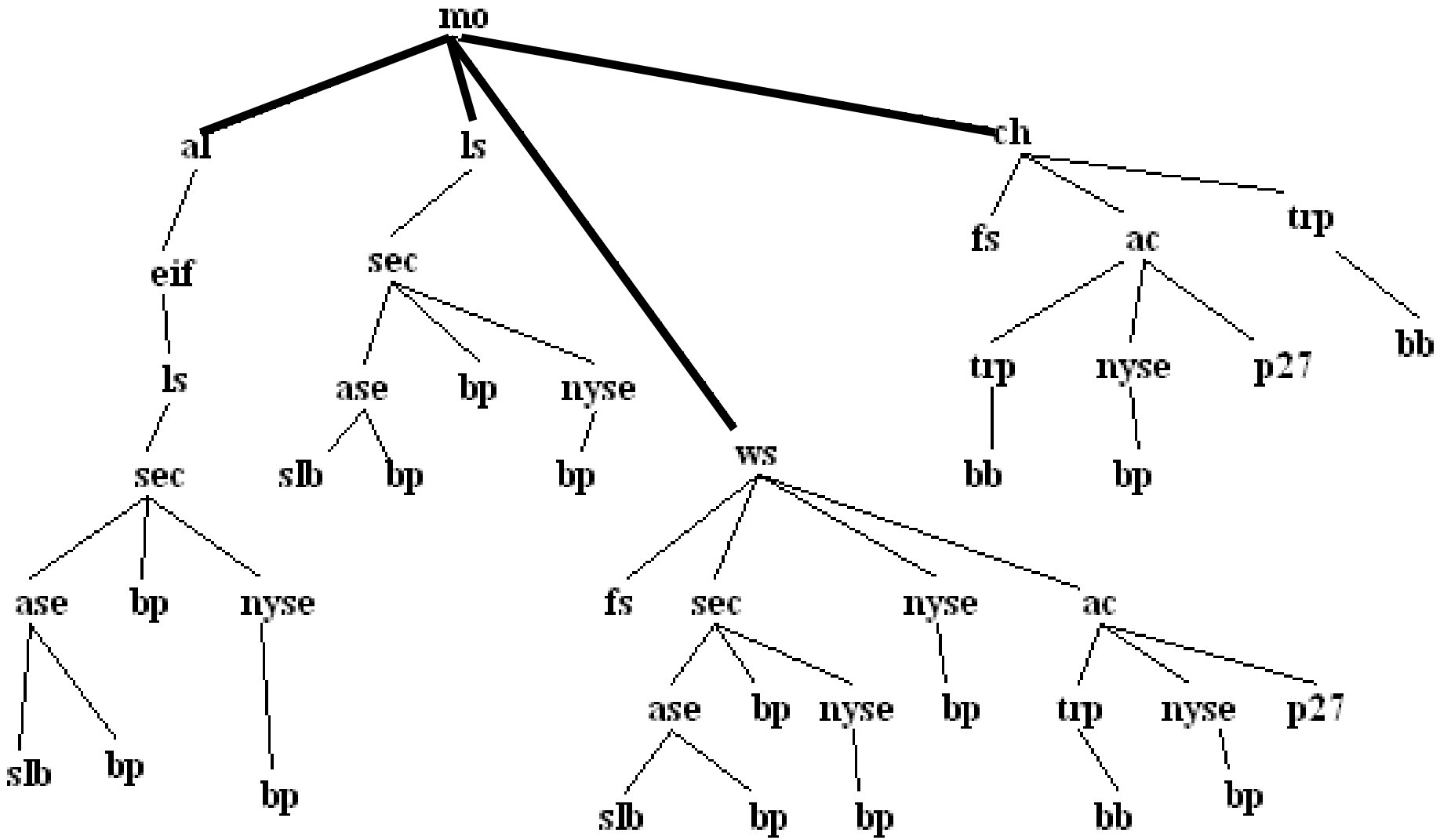
Algoritmo Genérico de Búsqueda

- 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 retornar solución
  - Agregar al final de FRONTERA nodos P apuntados por 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 algún esquema

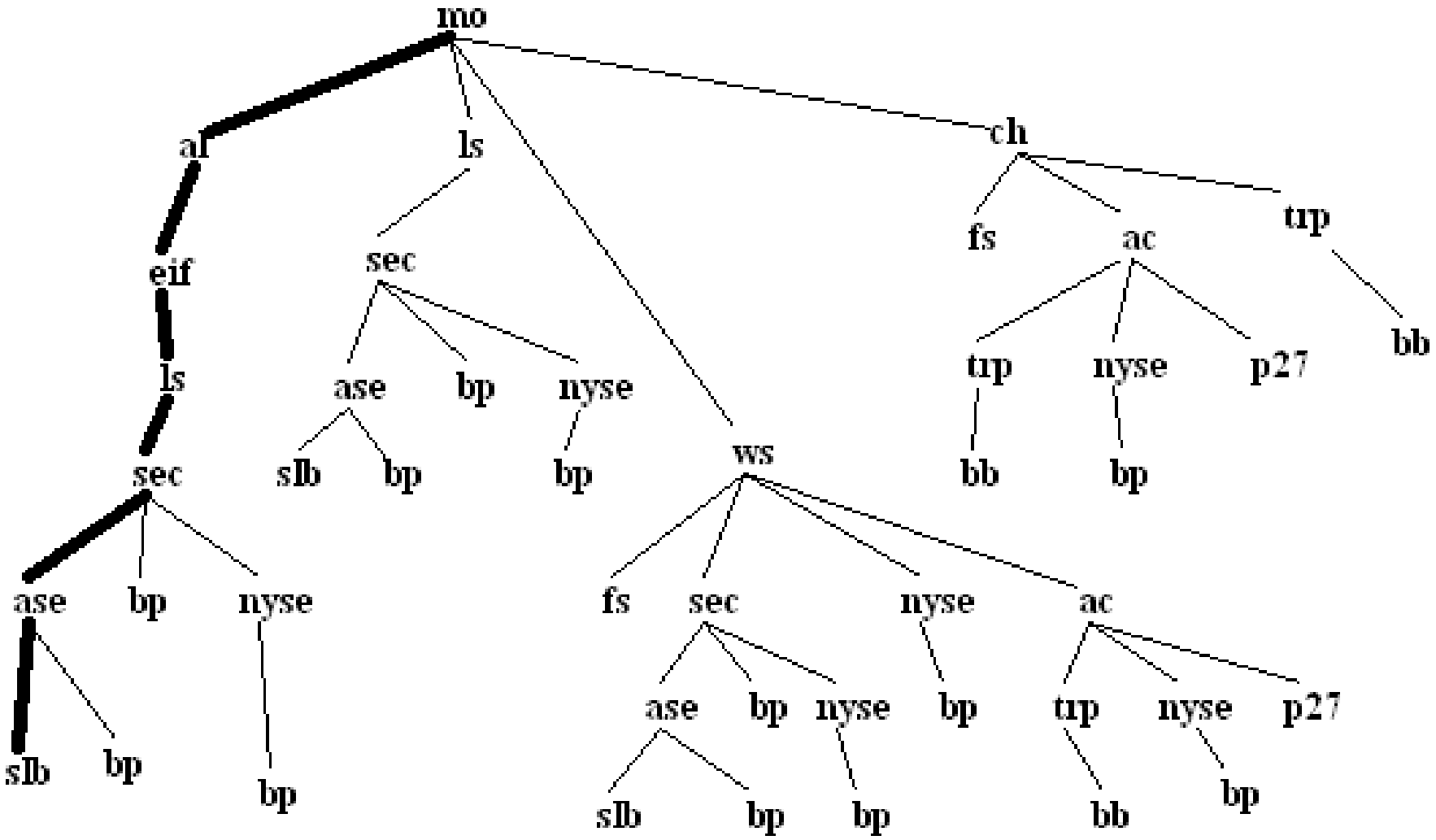
Arbol de Búsqueda (s = mo)



Búsqueda Primero en Anchura



Búsqueda Primero en Profundidad



Estrategias Básicas de Búsqueda

- Búsqueda Primero en Anchura (BPA)
  - Algoritmo genérico + política FIFO (First In First Out) de manejo de FRONTERA.
  - Arbol de búsqueda se genera a lo ancho.
  - Garantiza que el camino encontrado es el de menor longitud en número de arcos.
- Búsqueda Primero en Profundidad (BPP)
  - Algoritmo genérico + política LIFO (Last In First Out) de manejo de FRONTERA.
  - Arbol de búsqueda se genera en profundidad.
  - No es necesario almacenar el árbol completo en memoria (sólo el camino parcial calculado).
  - Si encuentra el objetivo, no garantiza nada sobre el camino retornado.

Búsqueda Primero en Anchura

- 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 stop, entregar solución
  - Agregar **al final** de FRONTERA nodos P apuntados por 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

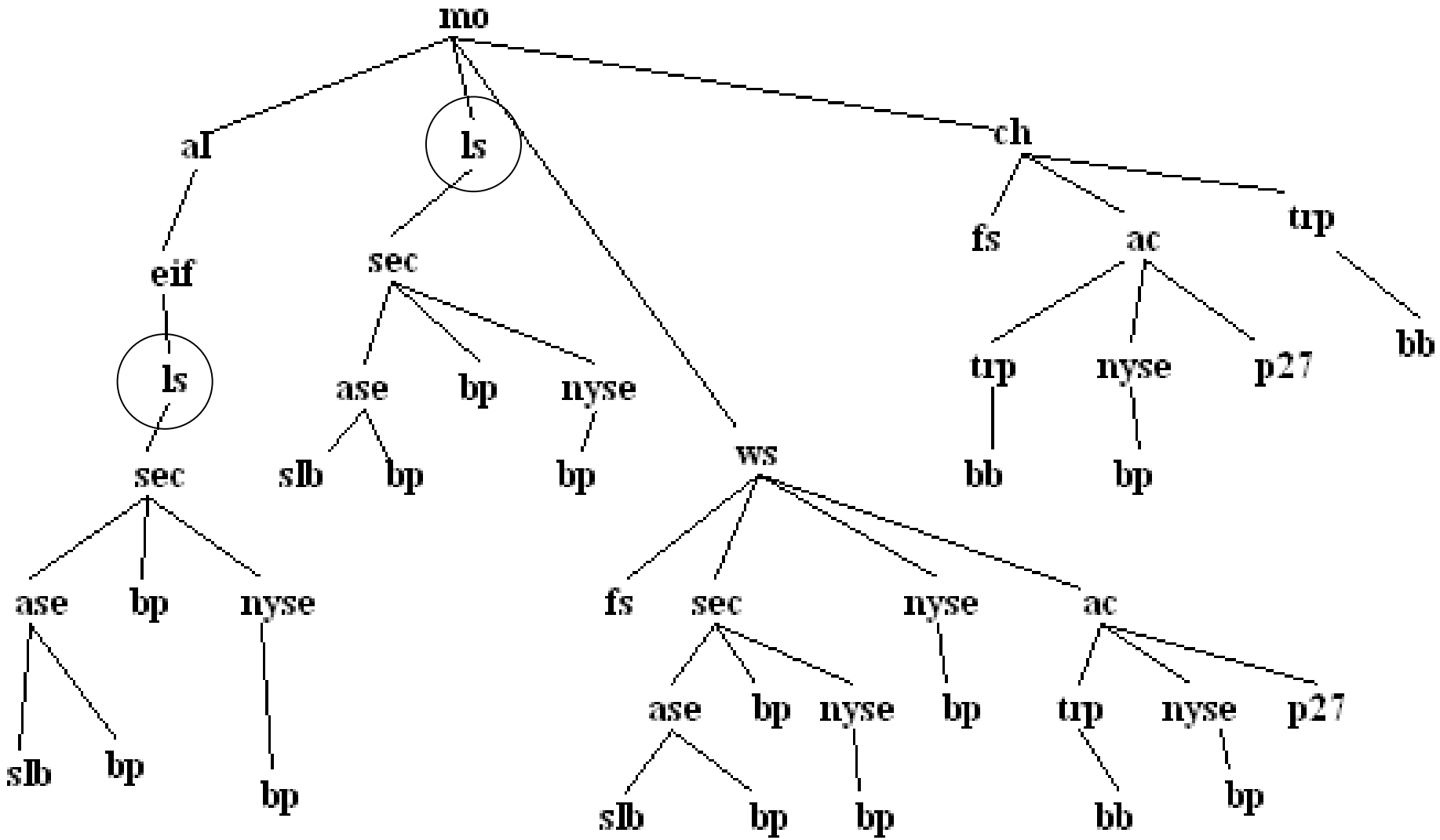
Búsqueda Primero en Profundidad

- 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
  - **Eliminar de Tr las hojas cuya etiqueta no está en FRONTERA (backtrack)**
  - Sacar primer nodo n de FRONTERA
  - Si n está en G stop, entregar solución
  - Agregar **al comienzo** de FRONTERA nodos P apuntados por 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

Ciclos

- BPA: si existe una solución, no producen problemas.
- BPP: pueden llevar a ejecuciones que no terminan.
- Soluciones:
  - Cota de profundidad.
  - Detección de ciclos.

Expansiones Redundantes



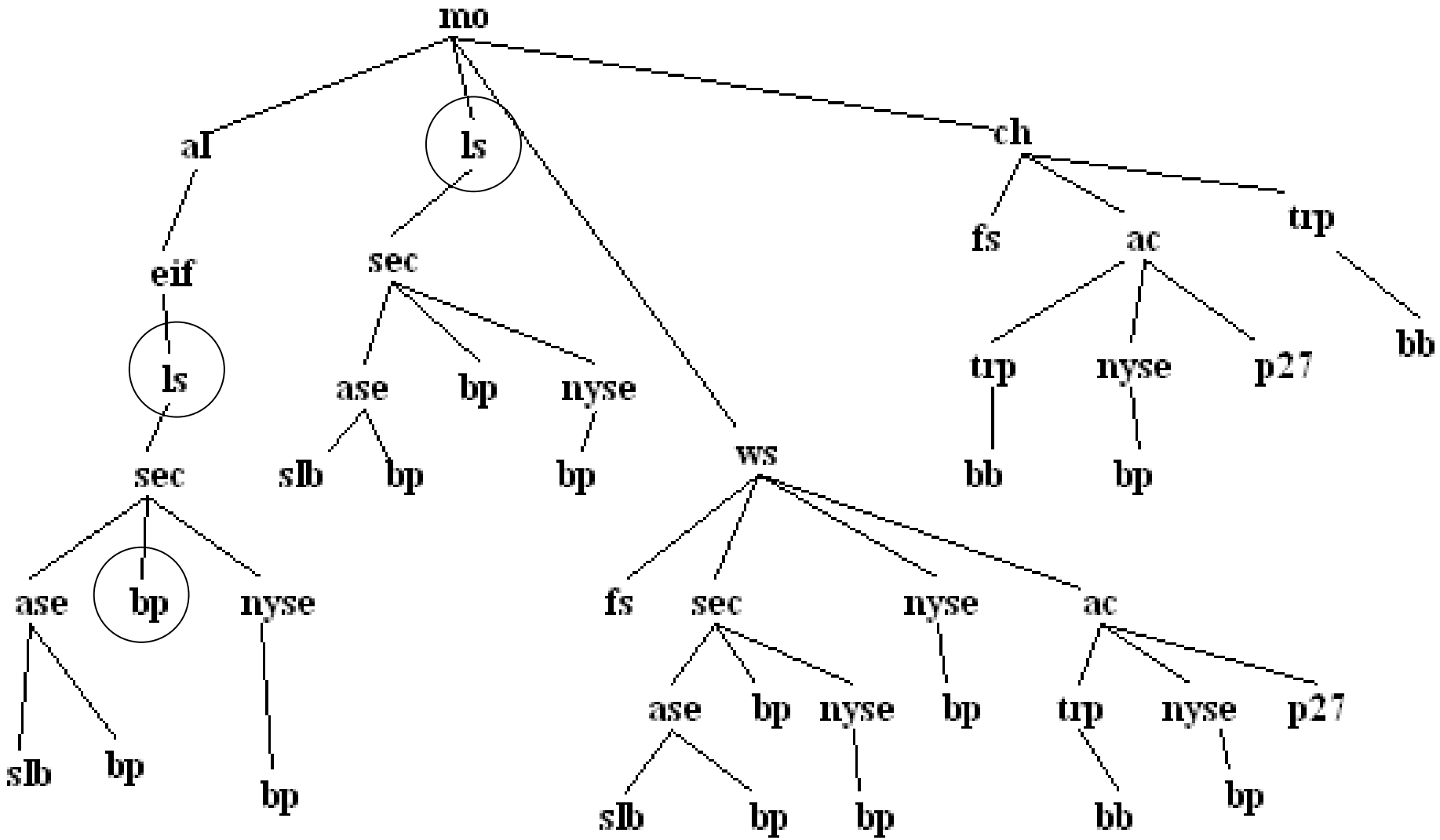
Expansiones Redundantes

- Problema clave de algoritmos búsquedas es evitar varias expansiones de un mismo nodo.
- Si no fuese necesario extraer el mejor camino (i.e., sólo decir si existe algún camino objetivo pero no cuál), simplemente podríamos usar una lista de nodos visitados para evitar expansiones redundantes.
- Es por esto que BPP y BPA sin extracción de caminos toman O(n^2) pasos.

Expansiones Redundantes

- Si necesitamos entregar el camino óptimo, evitar expansiones redundantes es mucho más complejo.
- Si llegamos al mismo nodo por segunda vez, podríamos eliminar los peores caminos.
- Es decir, mantener el mejor camino por cada nodo.
- El problema de esto es que al visitar un nodo por segunda vez, podríamos tener que modificar los mejores caminos de otros nodos.

Expansiones Redundantes



Costos de BPP y BPA

- Tiempo: número de operaciones de expansión.
- Memoria: tamaño máximo de memoria requerida
- Sea
  - L: largo del camino con menos arcos a un nodo objetivo.
  - B: factor de ramificación del grafo.
  - N: número de nodos en el grafo.
- BPA
  - Tiempo: O(B^L) (si no hay solución: O(B^N)).
  - Memoria: O(B^L)
- BPP (suponiendo que detectamos ciclos)
  - Tiempo: O(B^N)
  - Memoria: O(B x N)

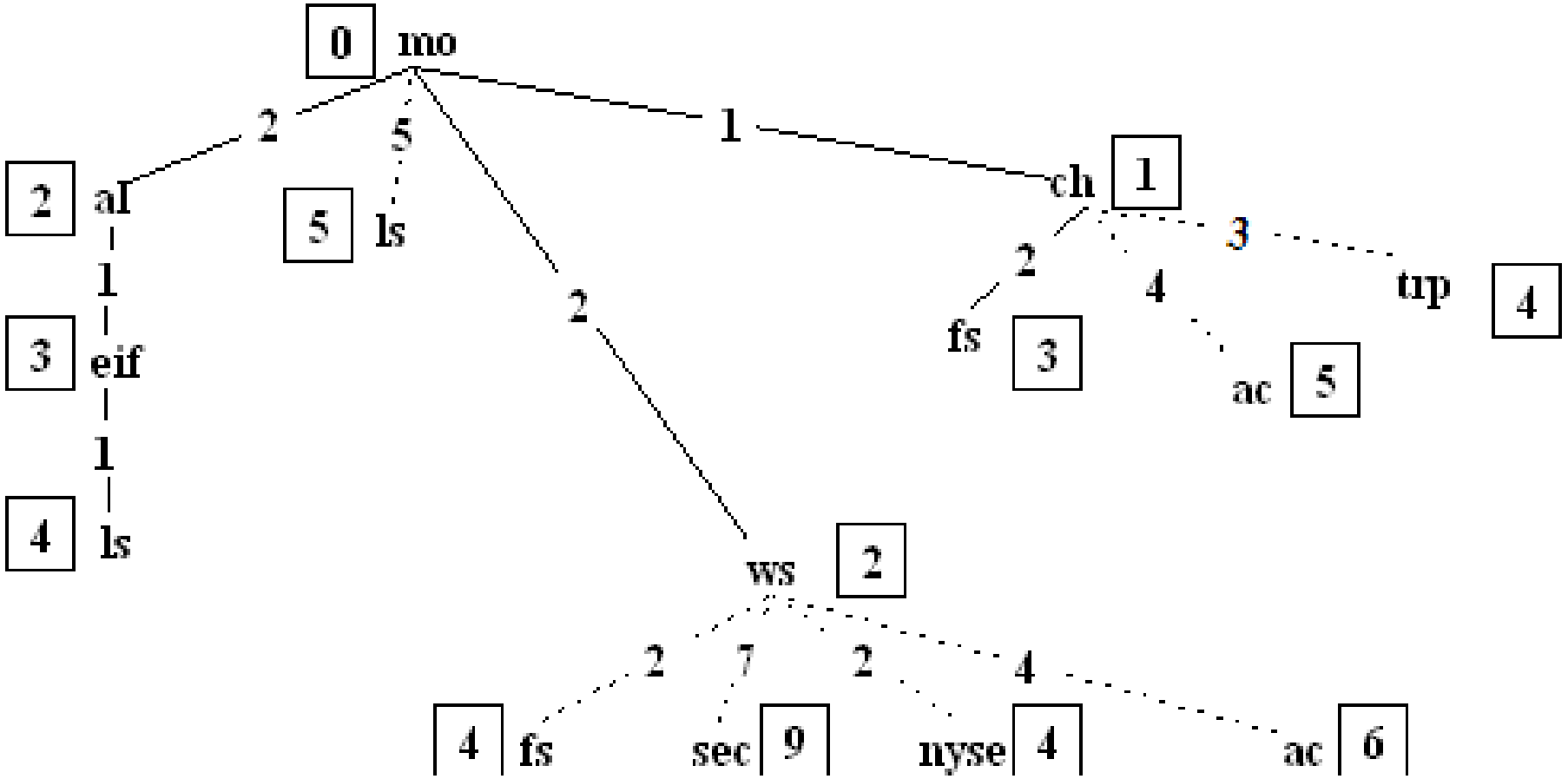
Búsqueda de Costo Uniforme

- Generalización de Búsqueda Primero en Anchura
- Arbol de búsqueda se genera a lo "ancho" pero en base a una función de costo.

Búsqueda de Costo Uniforme

- 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
  - **f(r): costo de un nodo r del árbol de búsqueda**
- While (FRONTERA no es vacío) do
  - Sacar primer nodo n de FRONTERA
  - Si n está en G stop, entregar solución
  - Agregar al final de FRONTERA nodos p en P apuntados por n y **calcular su valor f(p)**
  - Por cada nodo 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 f**

Búsqueda de Costo Uniforme



Búsqueda en Costo Uniforme

- Es el conocido algoritmo de Dijkstra para encontrar el camino más corto entre dos nodos.
- Costo en tiempo y espacio similar a búsqueda en anchura
- Con seguridad obtenemos el camino mejor (asumiendo costos positivos)
- ¿Qué sucede si tenemos costos negativos?

Búsqueda de Costo Uniforme y Expansiones Redundantes

- Es fácil demostrar lo siguiente:
  - La primera expansión de un nodo n produce (en el árbol de búsqueda) el camino óptimo desde el nodo inicial a n.
- Es decir, en expansiones redundantes de un nodo no mejoramos ningún otro mejor camino calculado.
- Podemos usar una lista de nodos visitados para vetar expansiones redundantes.
- Esta es la base del algoritmo de Dijkstra y explica que éste tome tiempo O(n^2).

"Desinformación¨ en Estrategias de Búsqueda

- Supongamos que buscamos un nodo *g1*, y corremos un algoritmo de búsqueda visto
- Ahora cambiamos el nodo buscado a *g2* y corremos el algoritmo nuevamente
- Para los algoritmos vistos (sin heurística) los dos árboles de búsqueda son similares hasta el primero objetivo encontrado (*g1* o *g2*)
- Las estrategias de búsqueda vistas son "desinformadas"
  - Proceso de búsqueda no está influenciado por el nodo objetivo

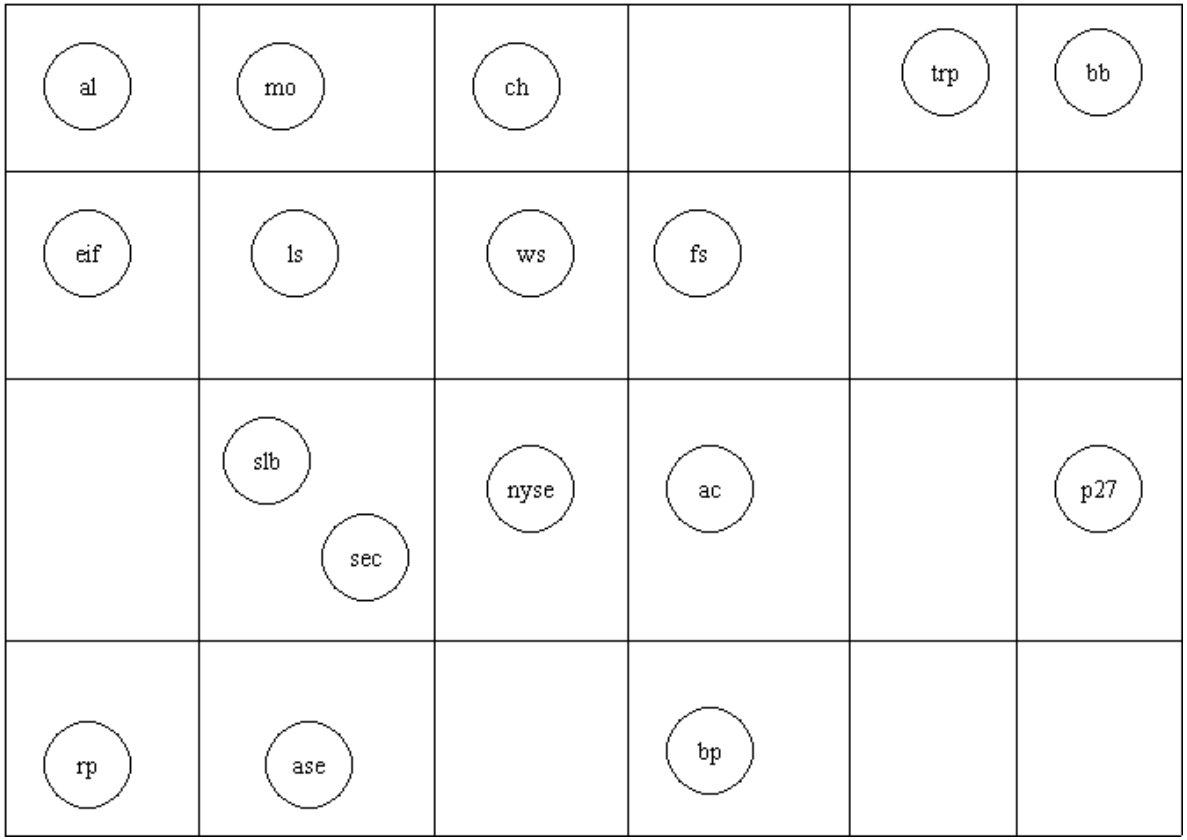
Heurística

- En general, regla hace más eficiente el proceso de resolución de un problema
- En el problema de búsqueda de caminos, una heurística es una función h' que estima el costo del camino más corto entre un nodo y el objetivo (costo del nodo).
- Es decir, h'(n) es una estimación del costo del nodo denotado h(n).

Propiedades de una buena Heurística

- Fácil de computar:
  - Si es costosa de computar entonces no ayuda a hacer más eficiente la búsqueda.
- Tiene buena precisión:
  - Descartar nodos malos
- No subestima el costo real de cada nodo
  - Si lo subestima podemos no encontrar la solución. Veremos esto más adelante.

Heuristica para MBC

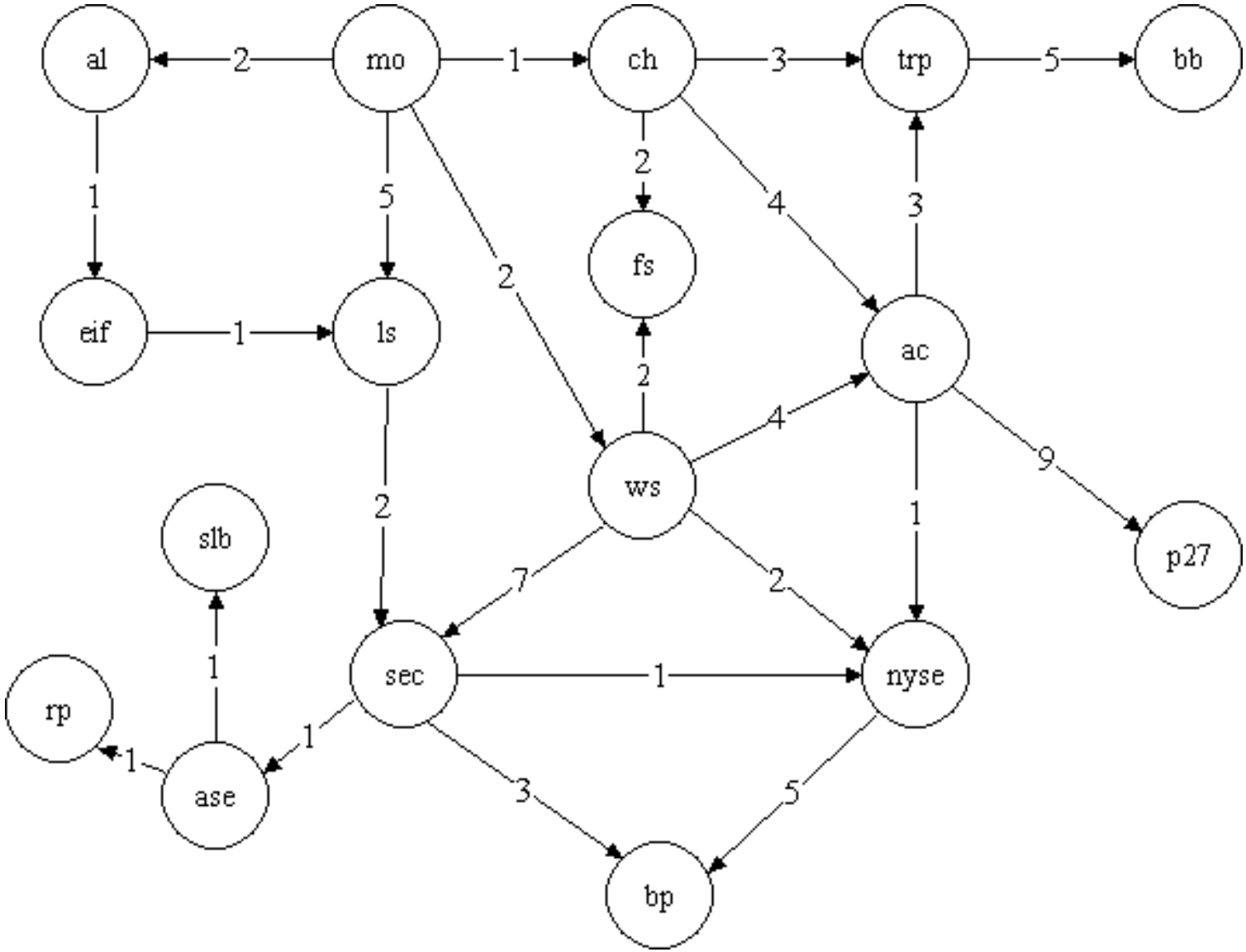


h'(n) = distancia de bloques entre n y el nodo objetivo

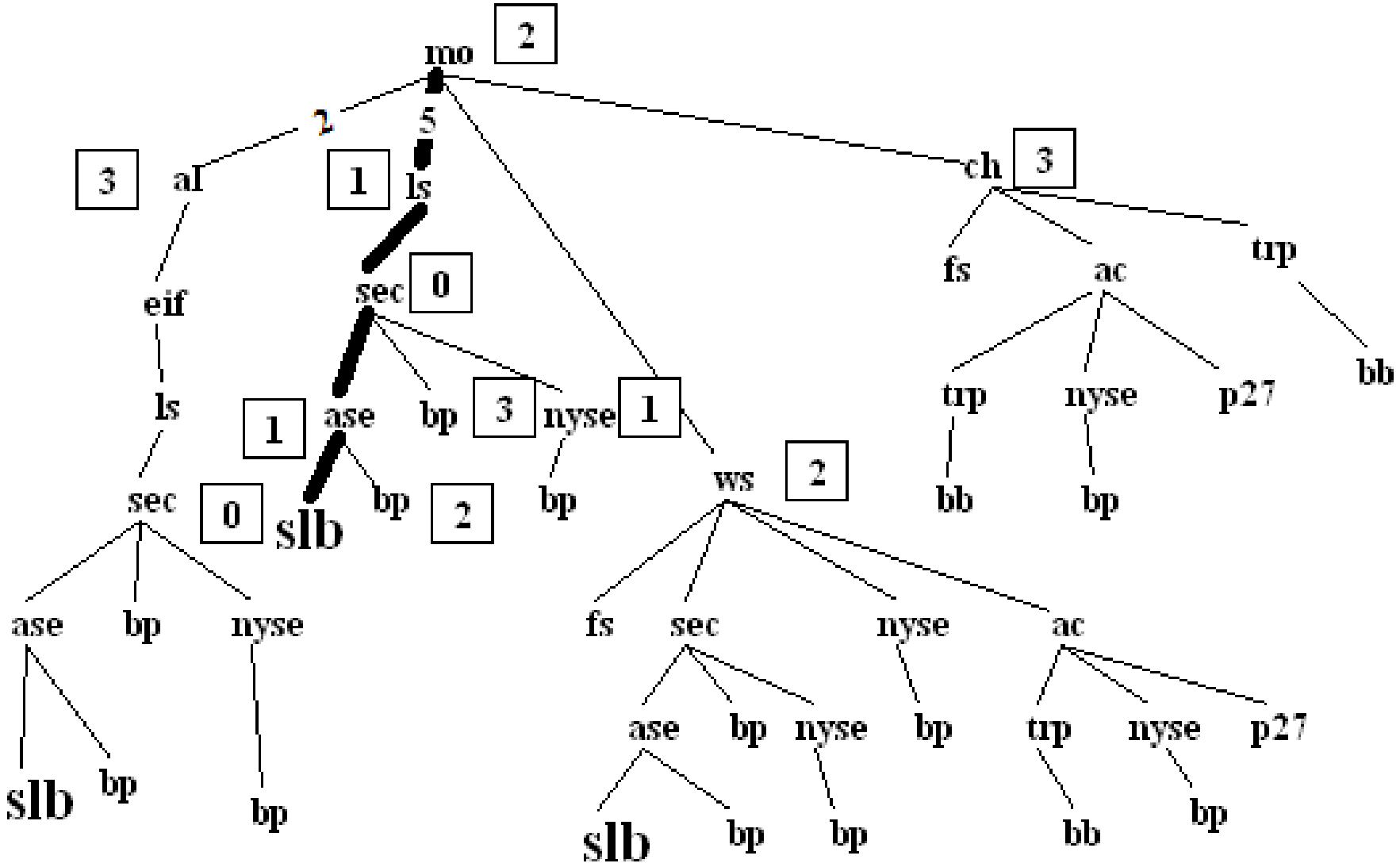
Busqueda Primero el Mejor (BPM)

- Similar a búsqueda en profundidad pero se ordena FRONTERA de menor a mayor valor de acuerdo a h'(n).
- Se exploran los caminos cuyos extremos parecen ser más cercanos al objetivo.

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



Búsqueda Primero el Mejor: mo - slb



Problemas con Búsqueda Primero el Mejor

- En el ejemplo anterior el algoritmo se acerca rápidamente al objetivo
  - No hay backtracking
- Pero no encuentra el camino de menor costo
- El algoritmo ignora los costos parciales de cada camino.
  - Tiene sentido sólo si buscamos el objetivo y no estamos interesados en el costo del camino

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 camino en la frontera está dado por:

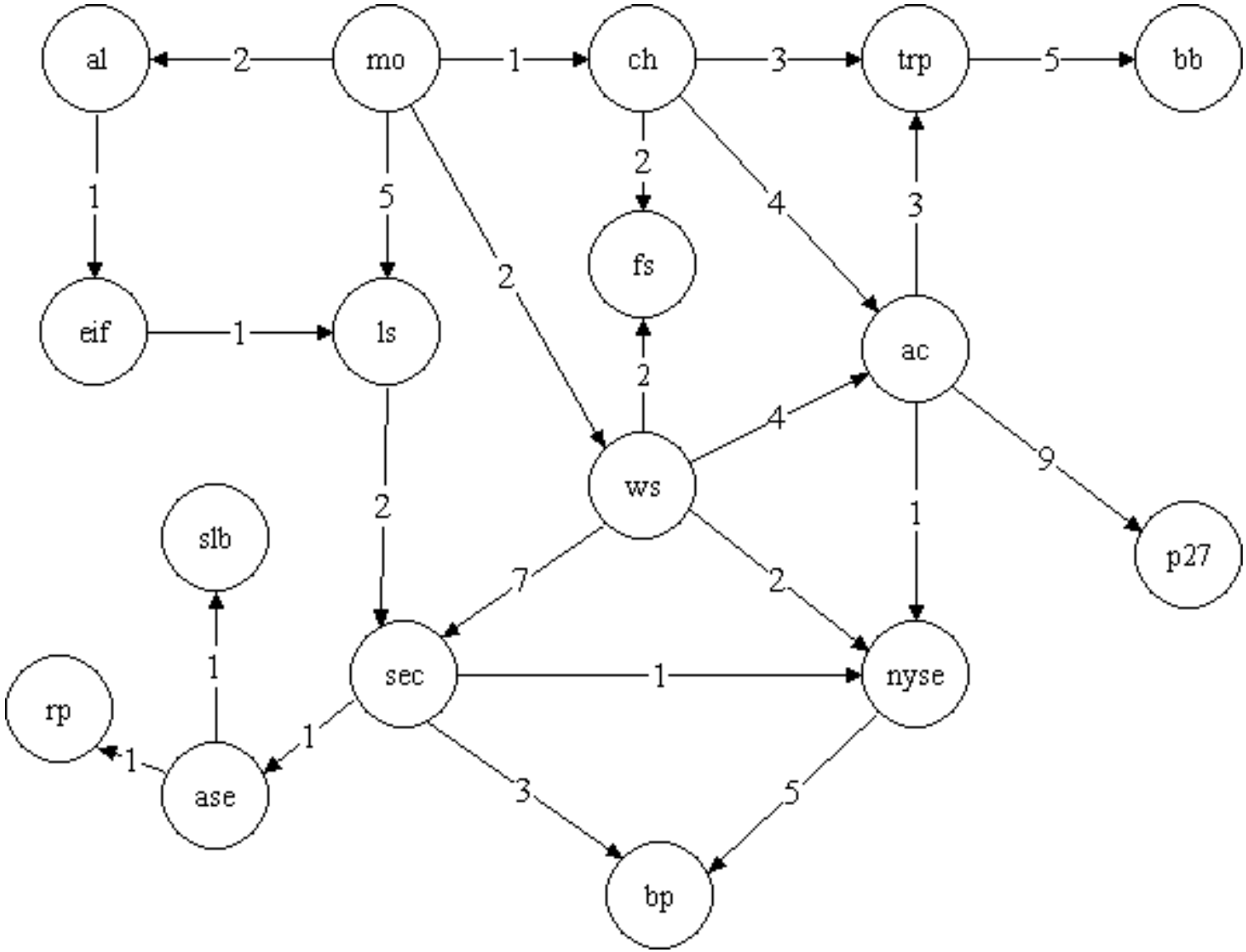
$$f(n) = g(n) + h'(n)$$

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

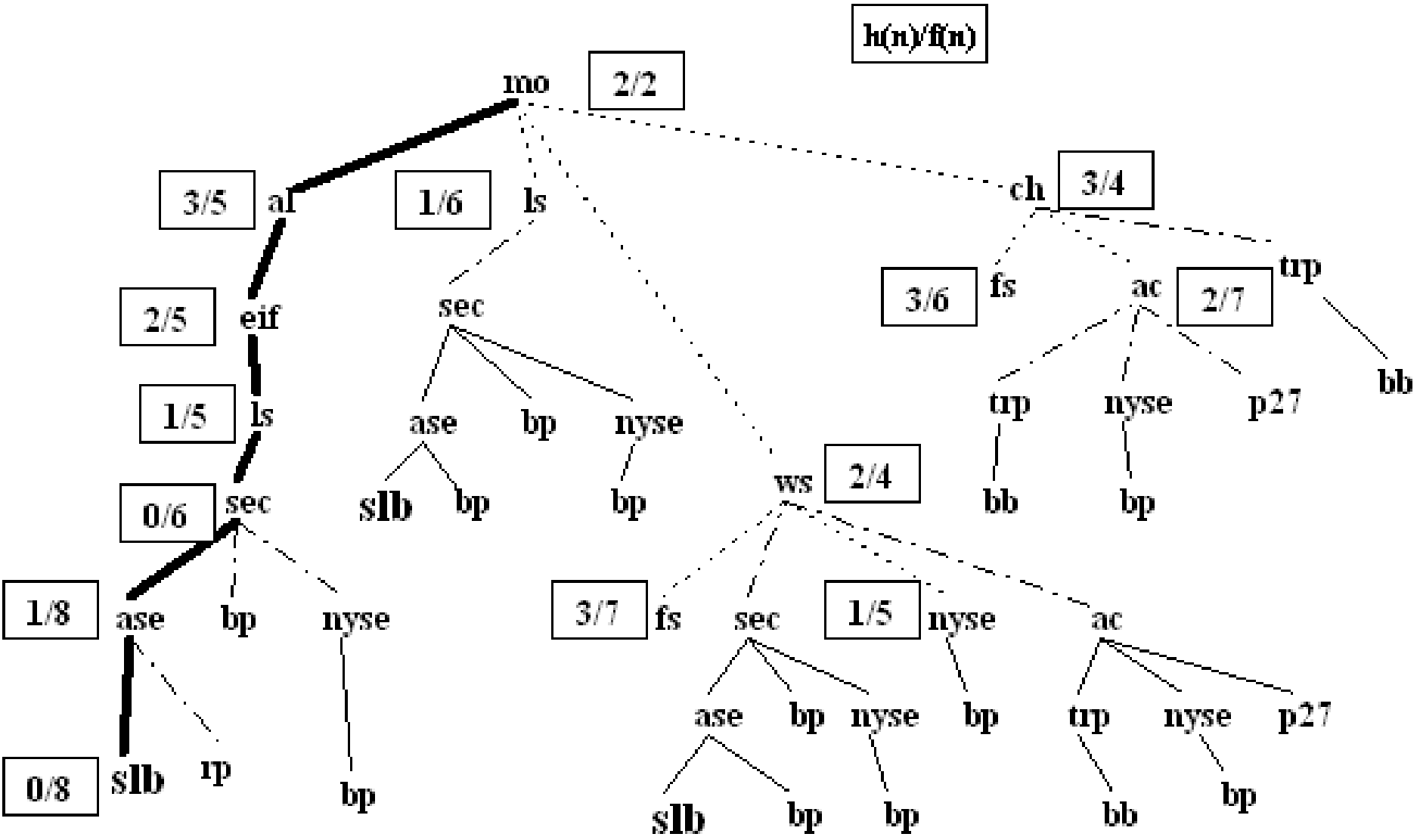
Algoritmo A*

- 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

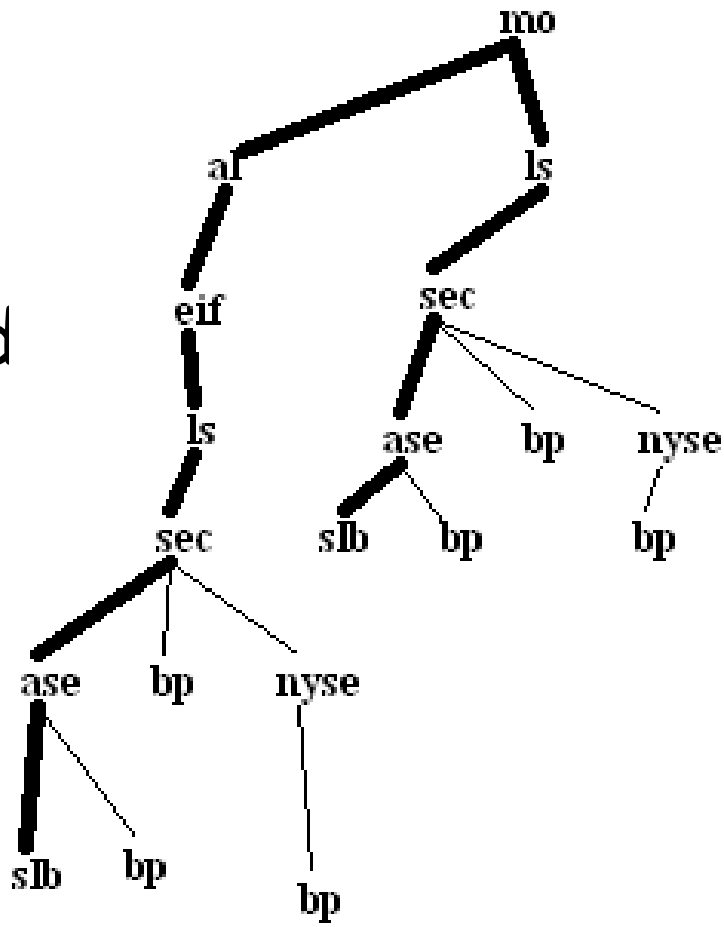
Análisis de A*

- En el ejemplo A* llega rápidamente al objetivo (slb)
  - Expande sólo 6 nodos que no llegan a la solución
- A* encuentra el camino de costo mínimo
- Combina lo mejor de
  - Búsqueda de Costo Uniforme (mejor camino) con
  - Búsqueda Primero Mejor (se dirige rápidamente al objetivo)

Propiedades de A*

- ¿Siempre A* encuentra el mejor camino?
  - No necesariamente:
  - Supongamos que h(al) =17
  - El nodo al no será expandid

  antes de del camino por ls



Heurística Admisible

- **Teorema**: Si el grafo contiene al menos un camino entre el nodo inicial y objetivos, entonces A* lo 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).
- La demostración consiste en probar dos condiciones:
  - (A) A* termina.
  - (B) El camino calculado por A* al momento de terminar es el camino de costo mínimo.

Heurística Admisible (cont.)

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

Esquema Demo condición (A)

- Supongamos que se cumple 1, 2 y 3.
- Sea p* el camino óptimo.
- En un número de iteraciones dada f(n), donde n es el nodo sacado de FRONTERA, aumenta.
- Esto por que los costos son positivos (2).
- Luego, por (1) eventualmente f(n) > costo(p*) y habremos alcanzado a un nodo objetivo.

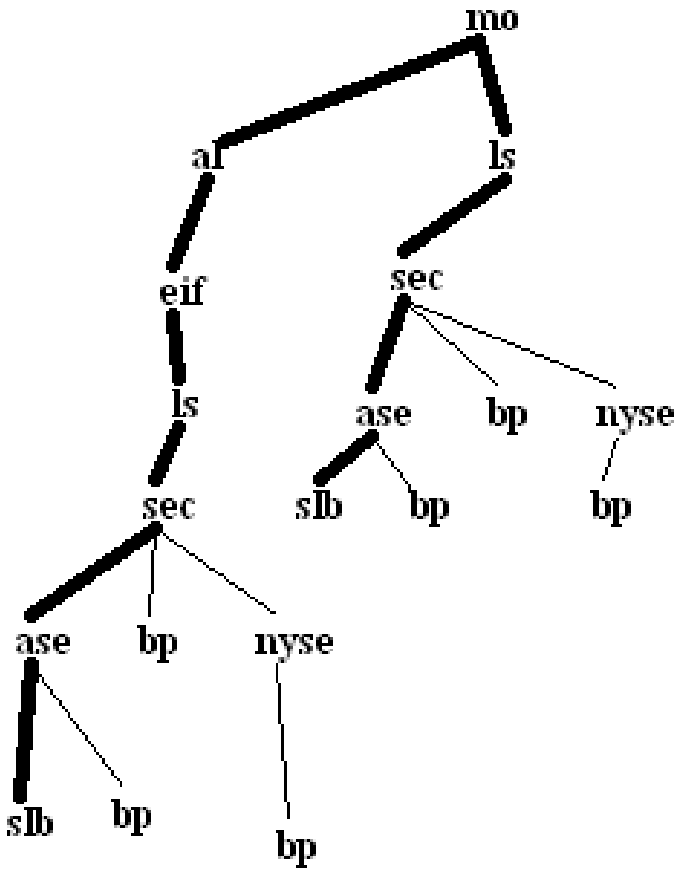
Esquema Demo Condición (B)

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



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 produce (en el árbol de búsqueda) el camino óptimo desde el nodo inicial a n.

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 h, menos nodos se expanden.

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, NbList, NewF),
               search( NewF ).
```

Define nodos objetivos

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, NbList, NewF),
               search( NewF ).
```

Define nodos objetivos

Define grafo

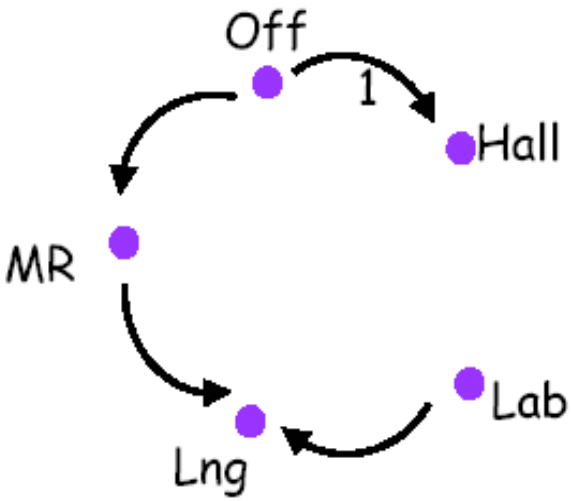
Búsqueda Genérica: Instanciación

```
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).
```

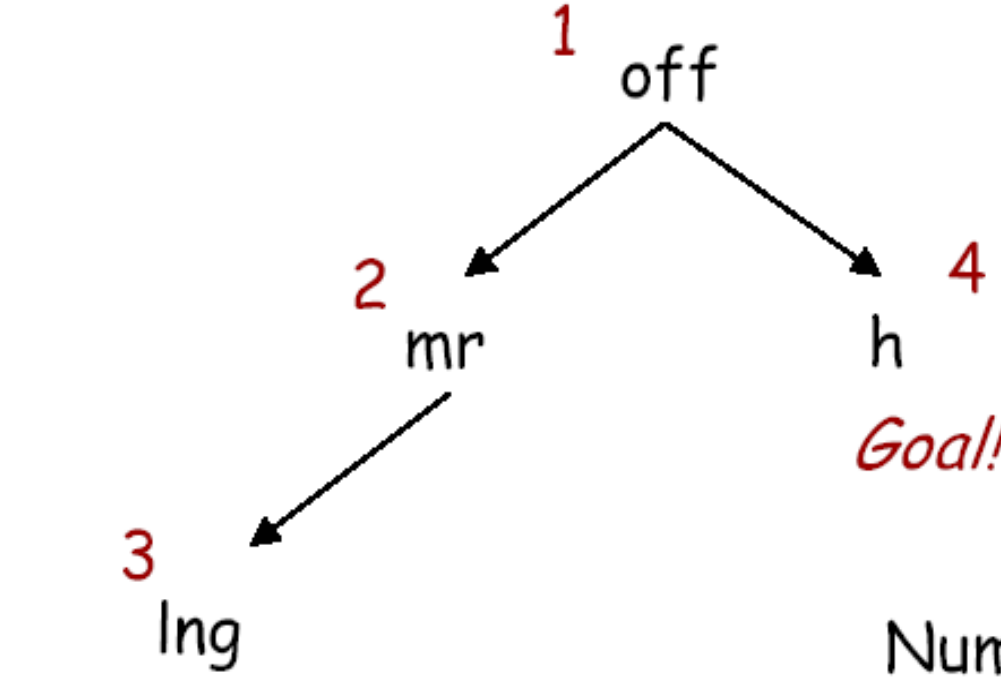
Start Node:

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)



Numbering reflects order in which nodes are examined (isgoal?)

What if we added an edge from lng to lab?

Extracción de Caminos en Búsqueda

```
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 ).
```

- Generic search procedure behaves as follows:
  - assert *isgoal(bp)* ; ask *?search([mo]).*
  - algorithm says *yes*.
  - but what path does courier take??
  - says *yes* if there is a path, but doesn't tell us what it is
- We need a *path extraction mechanism*
  - simplest thing to do is store paths on frontier
  - when you select a node from frontier, you also have a specific path to that node attached
- *search(F,Path) :* true if Path leads from the start node to a goal node.

Extracción de Caminos en Búsqueda (cont.)

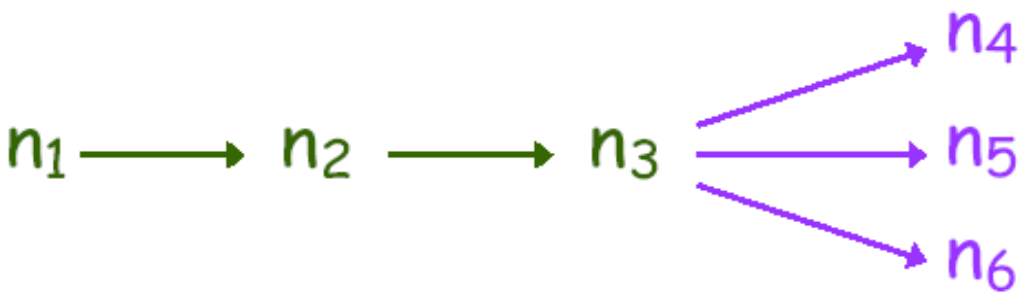
Assume a path is represented as a list of nodes in reverse order: so the path mo → al → eif is represented as the list: [eif, al, mo]

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