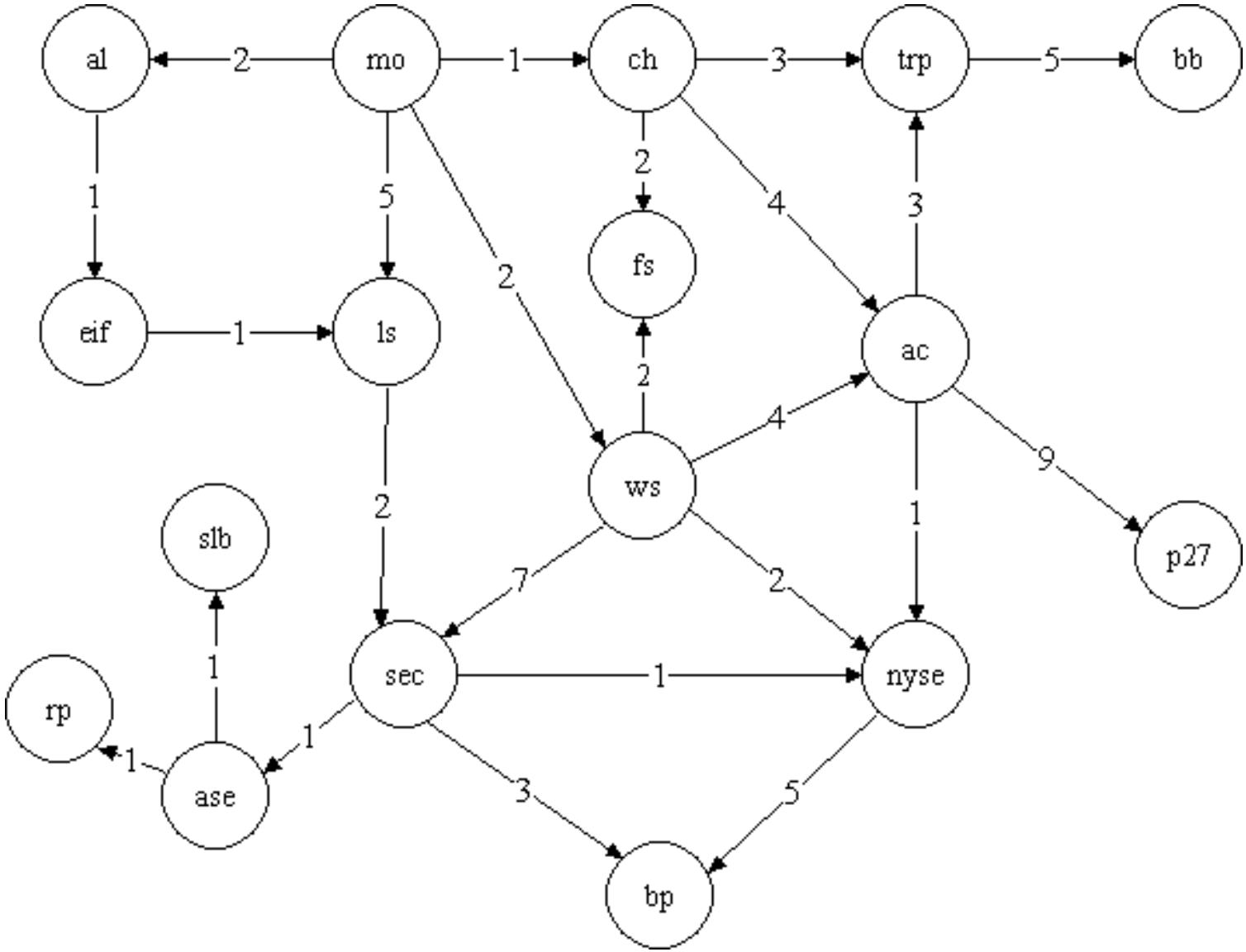
Solución de Problemas Mediante Búsqueda (2)

Carlos Hurtado

Depto de Ciencias de la Computación,

Universidad de Chile

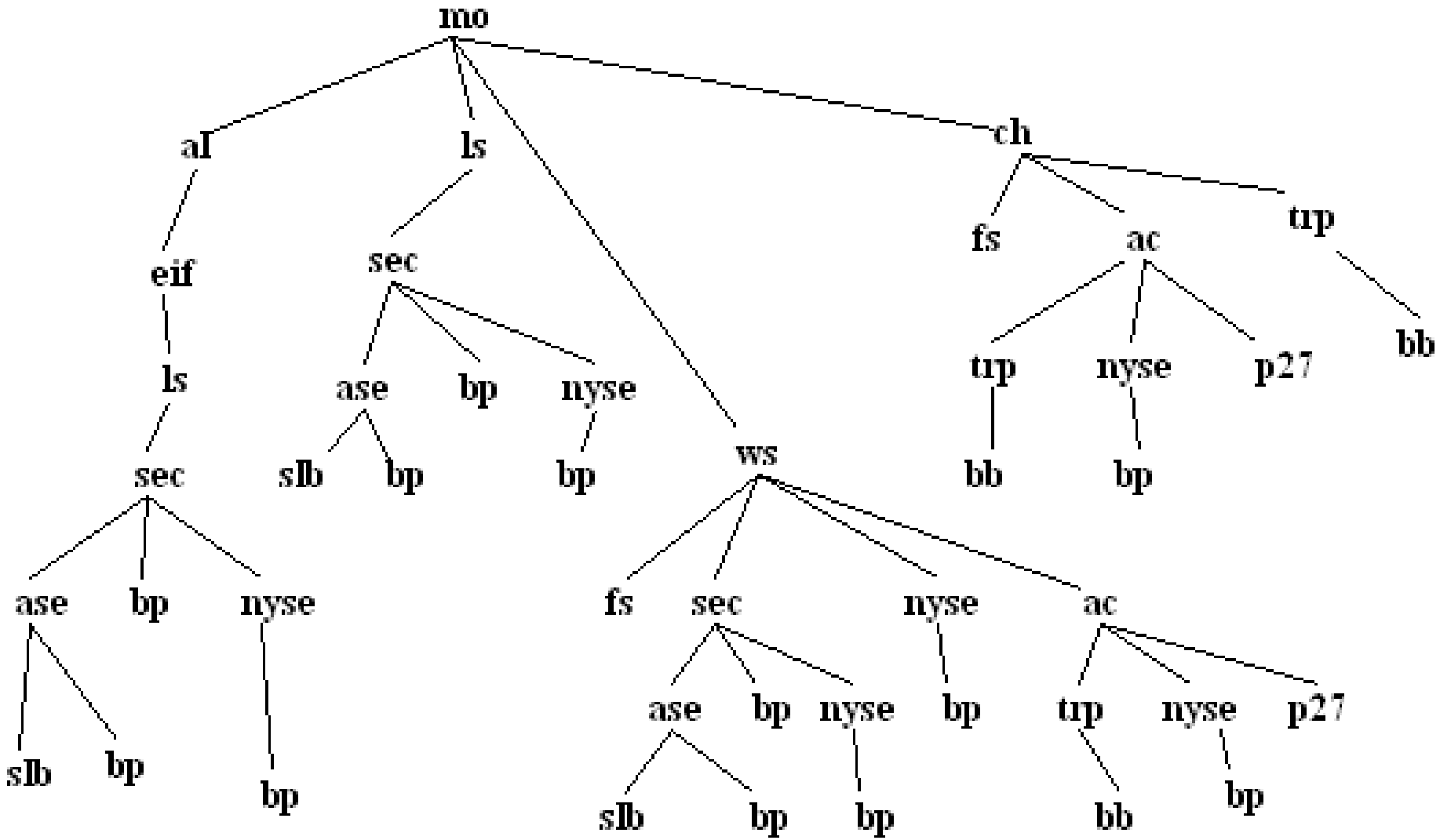
Manhattan Bike Curier (Acíclico) Ref. Curso IA U. of Toronto



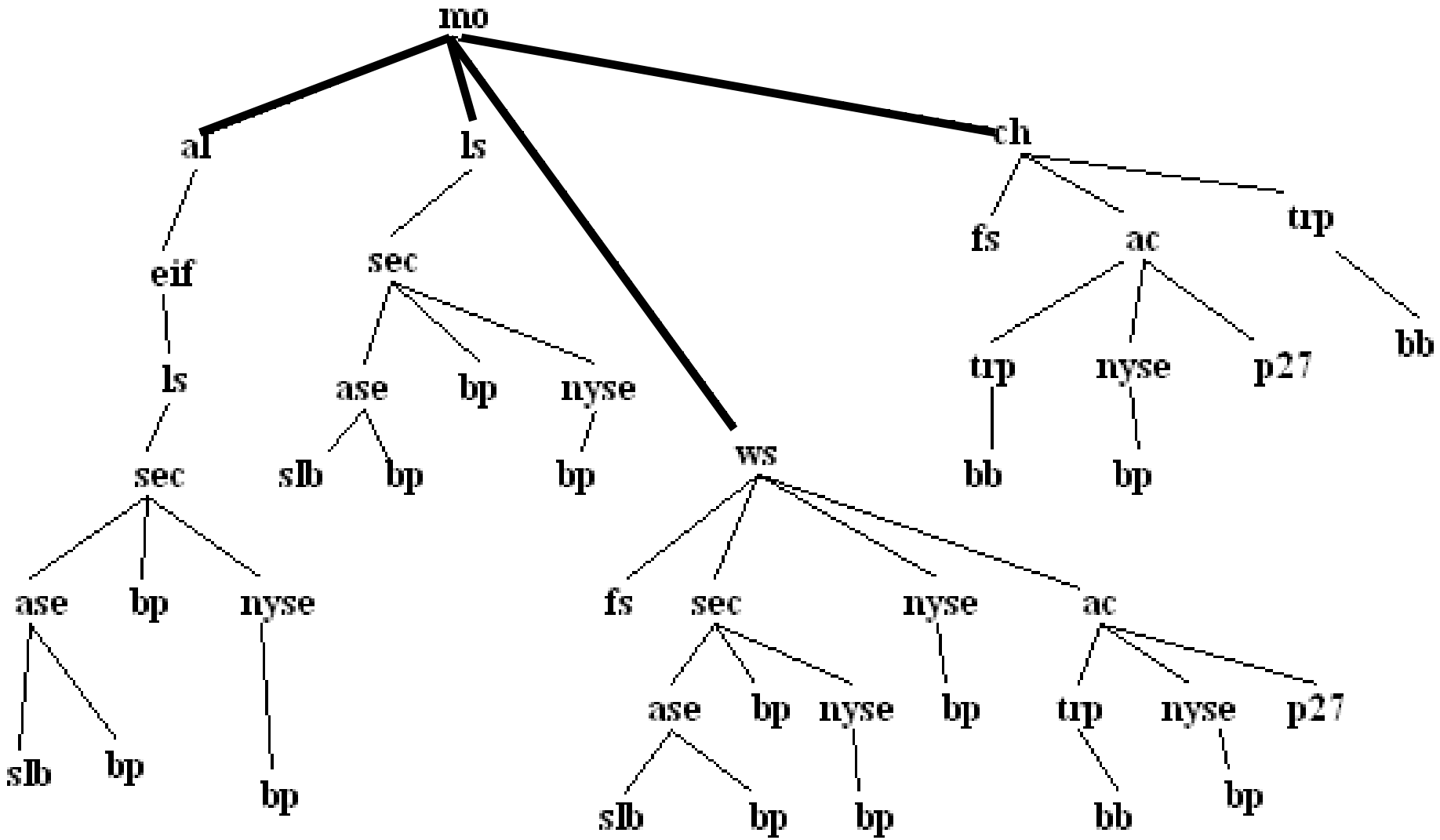
Algoritmo Genérico de Búsqueda

- 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

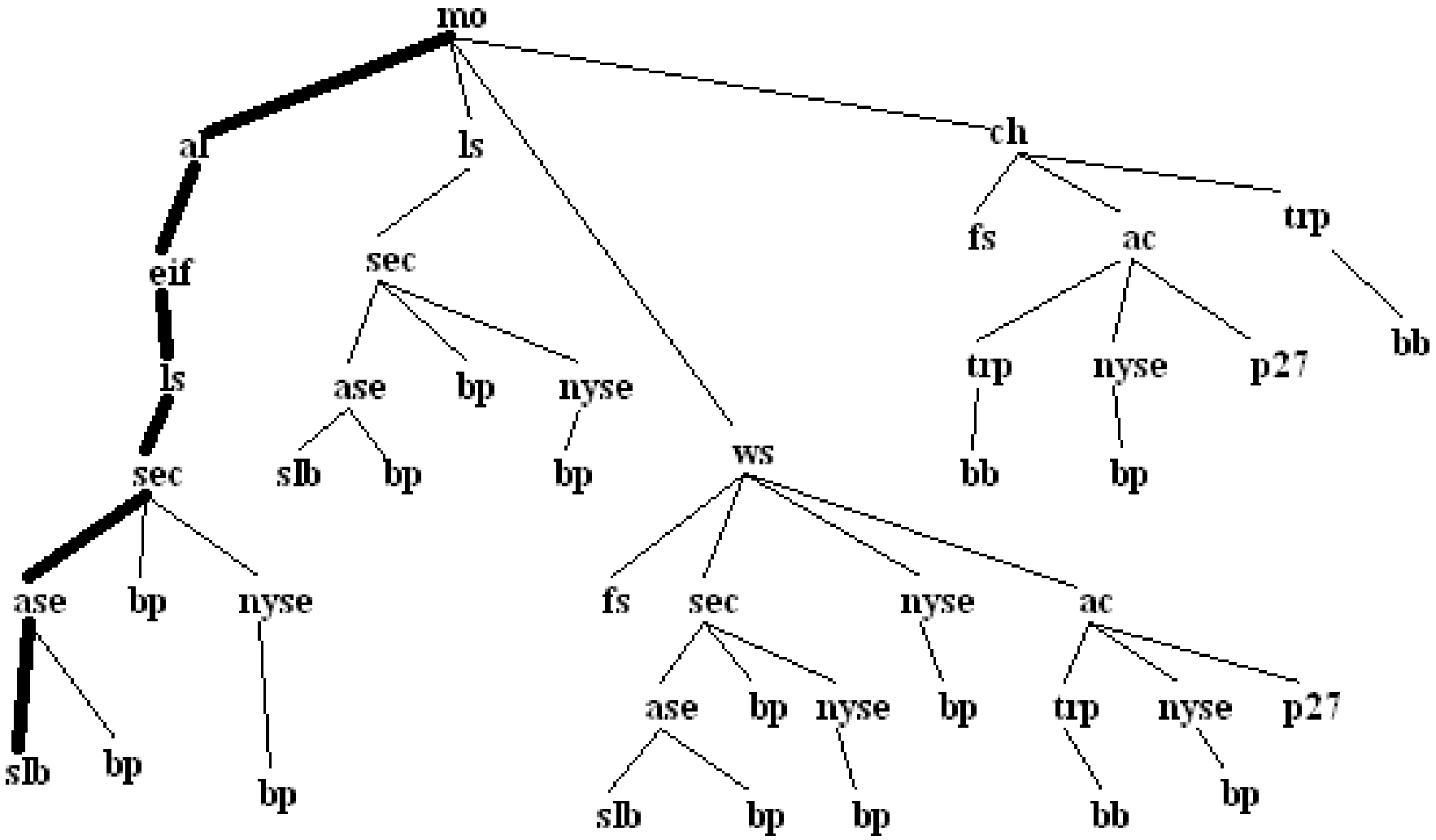
Arbol de Búsqueda (s = mo)



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

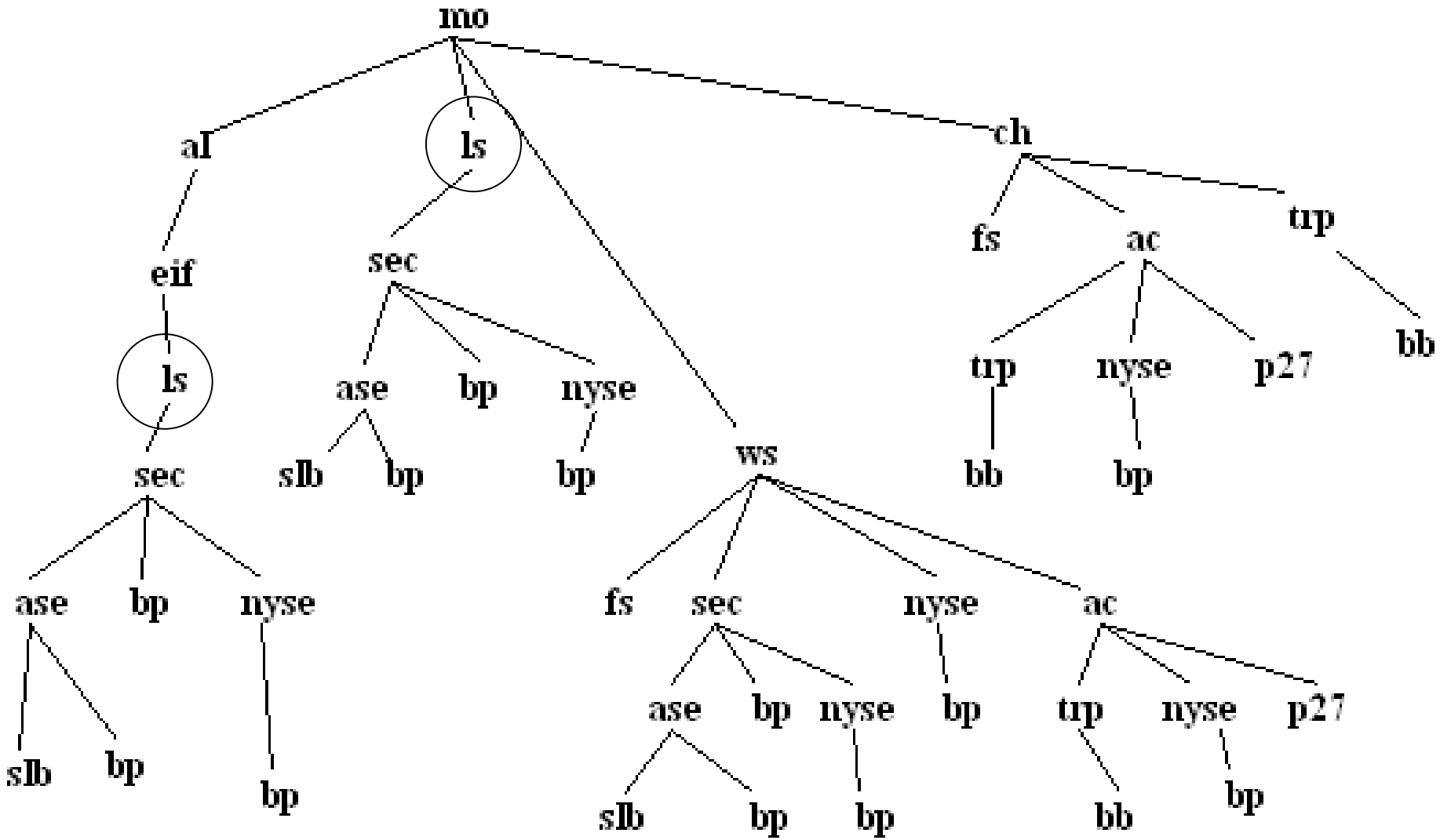
Ciclos

- Si no existen ciclos en el grafo de estados ambos algoritmos terminan.
- Si existen ciclos:
  - BPA puede no terminar, pero si existe una solución termina.
  - BPP: puede no terminar aunque exista una solución.
  - Soluciones:
    - Cota de profundidad.
    - Detección de ciclos.

Costos de BPP y BPA

- Tiempo: número de operaciones de expansión (cuando sacamos un nodo de la frontera).
- Memoria: tamaño máximo de memoria requerida
- Sea
  - B: factor de ramificación del grafo.
  - N: número de nodos en el grafo.
- Suponiendo que los algoritmos terminan:
  - BPA
    - Tiempo: O(B^N)
    - Memoria: O(B^N)
  - BPP
    - Tiempo: O(B^N)
    - Memoria: O(B x N)

Expansiones Redundantes



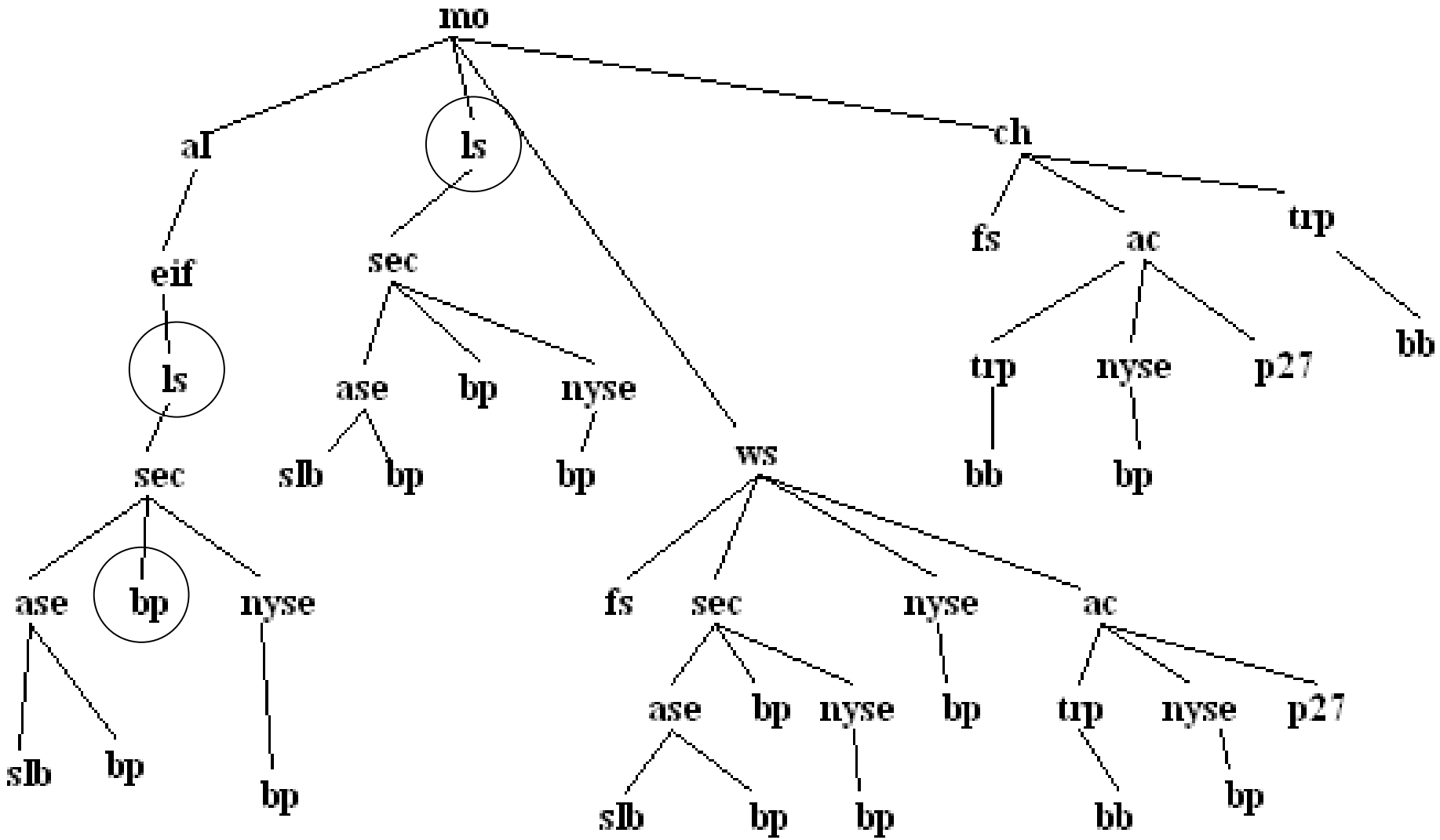
Expansiones Redundantes

- Problema central de algoritmos búsquedas es evitar expansiones redundantes.
- Si no es necesario extraer el mejor camino (sólo decir si existe algún camino a un objetivo) ésto es fácil.
  - En este caso basta tener 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.
- Solución básica:
  - Mantener en cada nodo el mejor camino desde el nodo de inicio.
  - Si llegamos al mismo nodo por segunda vez, eliminar el peor camino.
  - El problema de esto es que al visitar un nodo por segunda vez, podríamos necesitar modificar los mejores caminos de otros nodos.
- Más adelante veremos otras soluciones.

Expansiones Redundantes



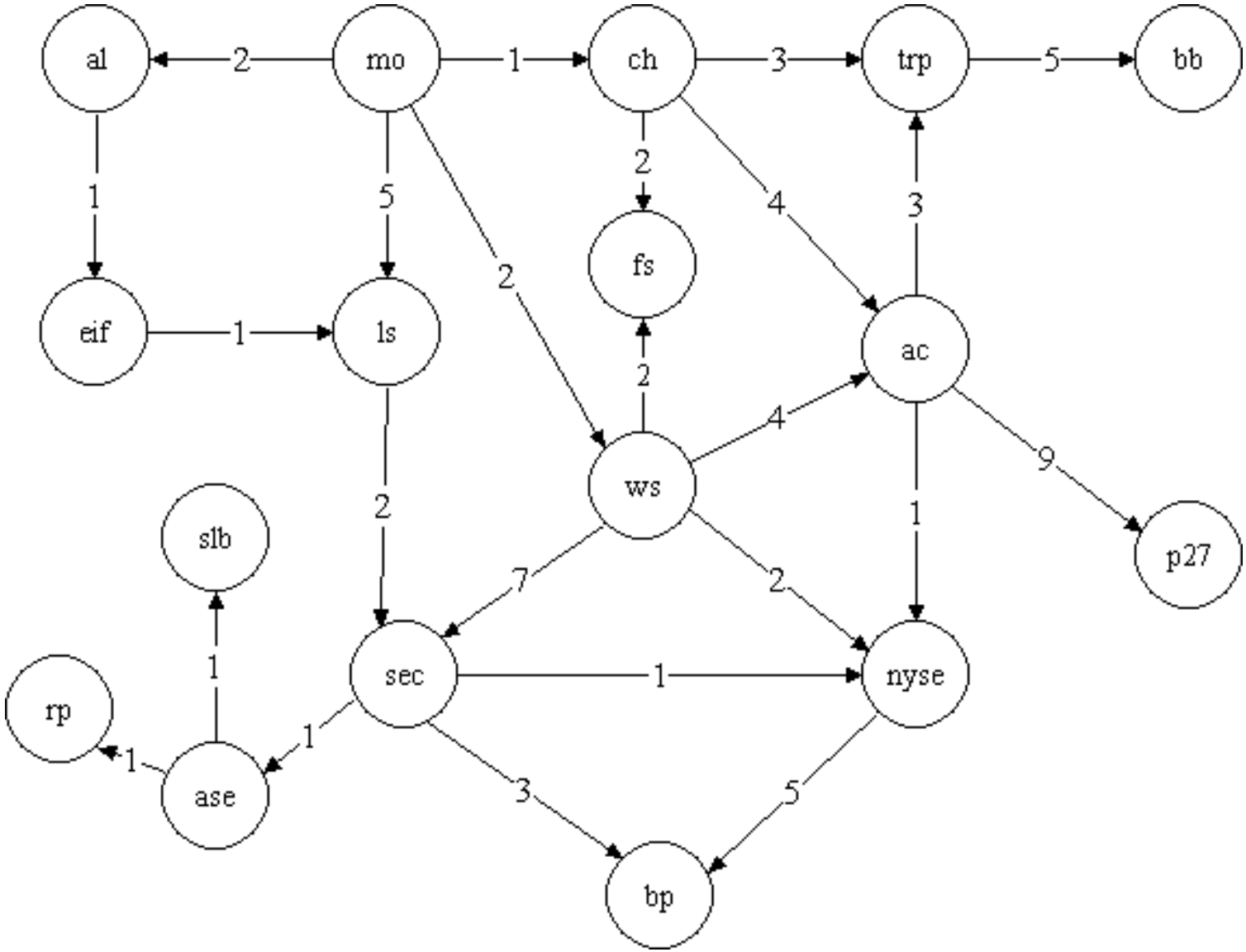
Búsqueda de Costo Uniforme

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

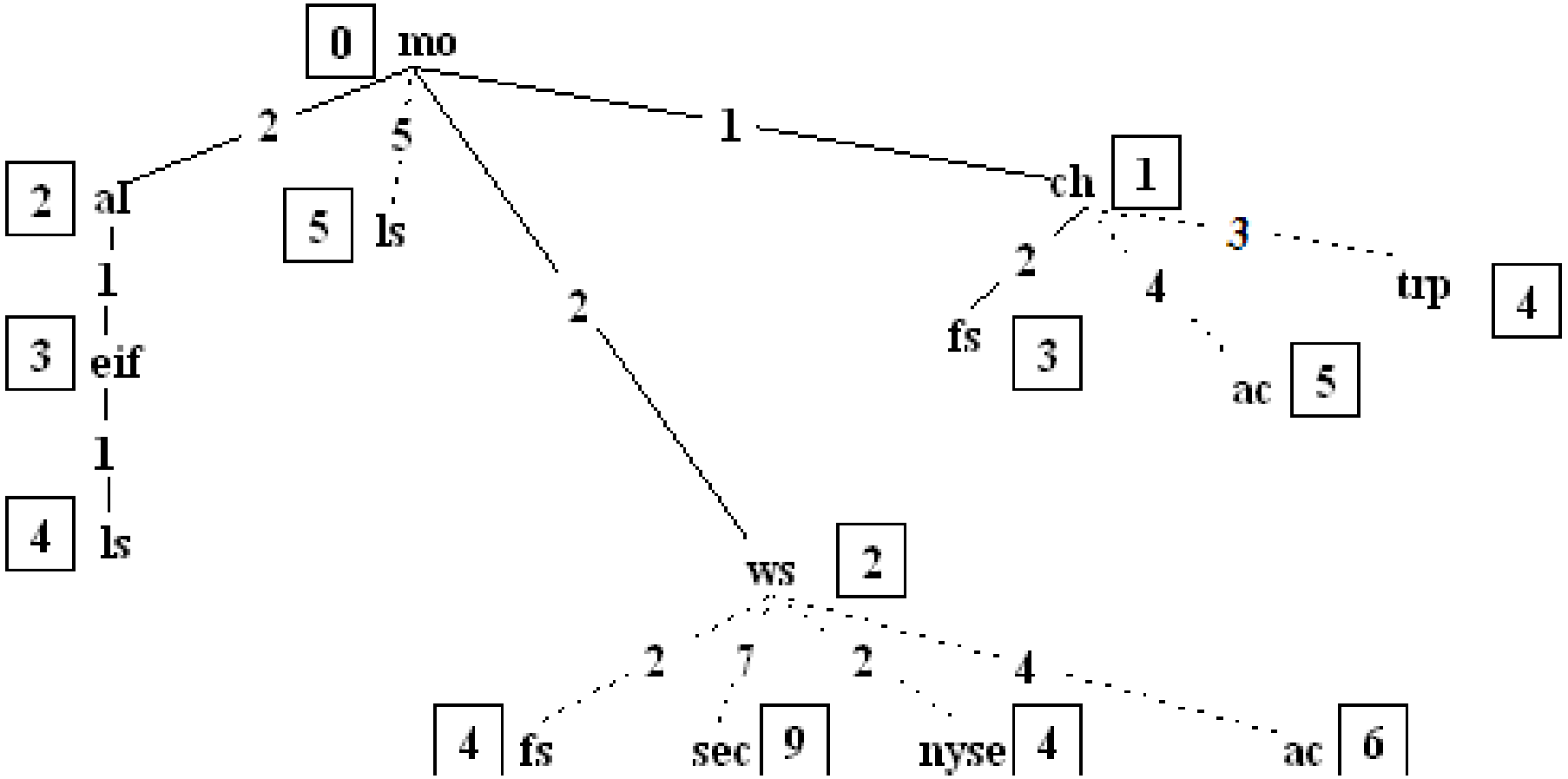
Búsqueda de Costo Uniforme

- 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**

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



Búsqueda de Costo Uniforme



Búsqueda en Costo Uniforme

- Es conocida como el 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 de costo mínimo (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 camino calculado.
- Podemos usar una lista de nodos visitados para evitar 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

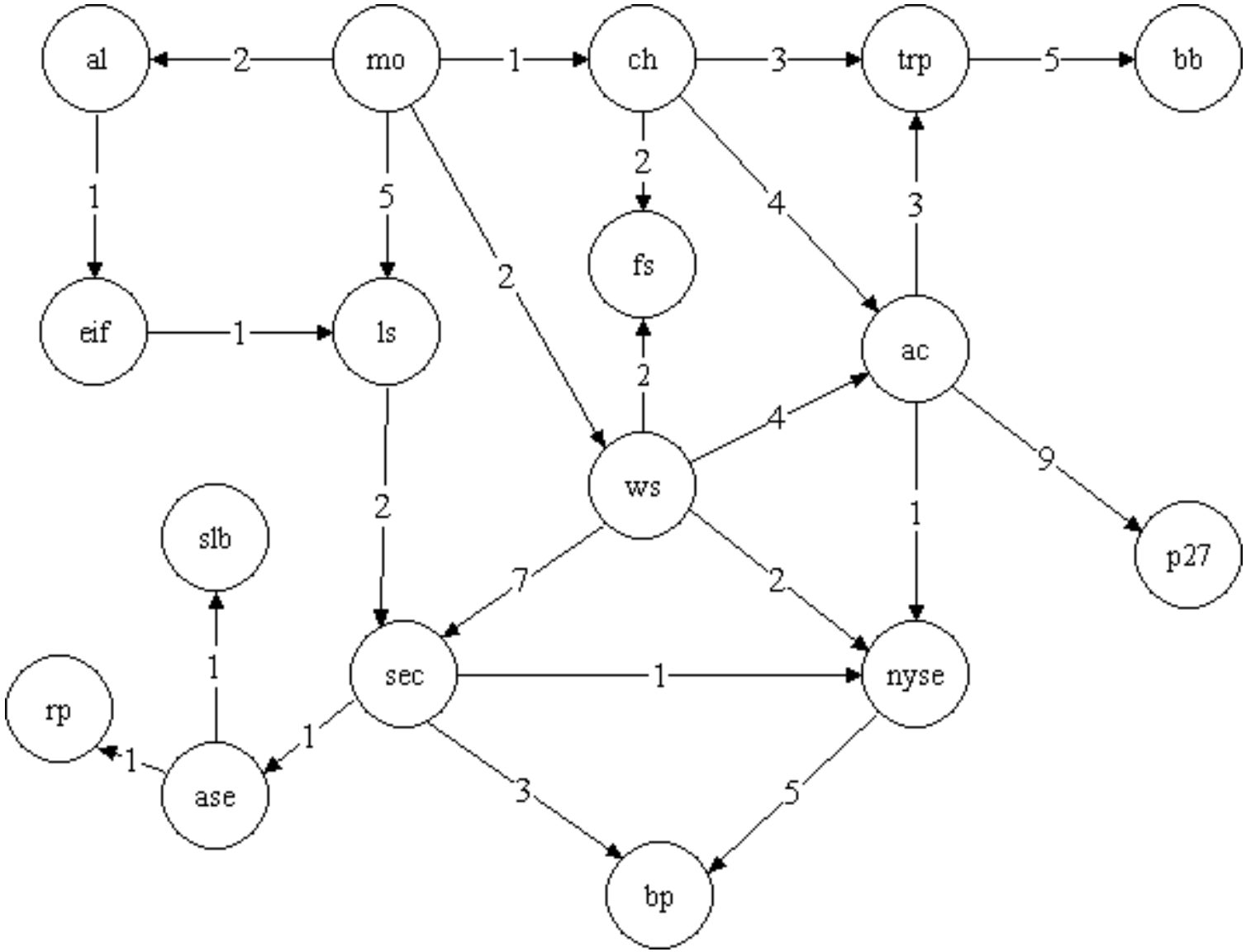
“Desinformación¨ en Estrategias de Búsqueda

- 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

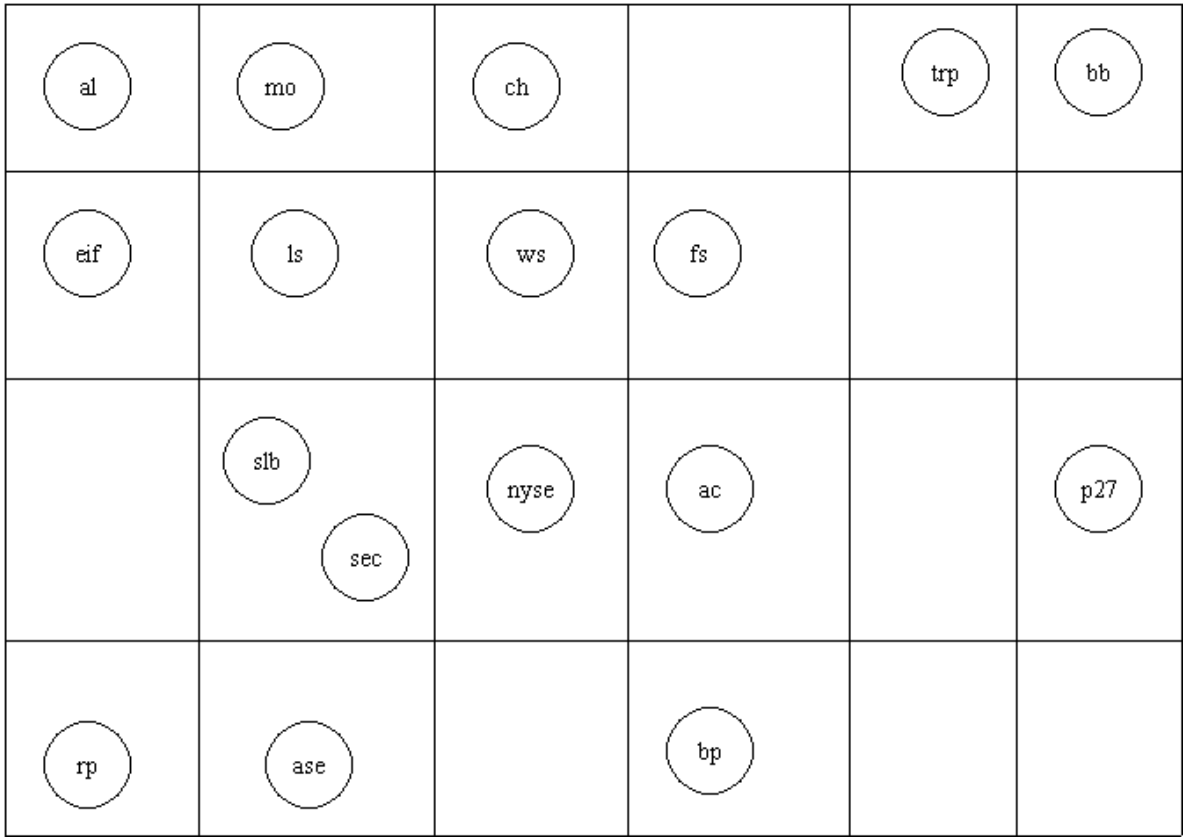
Heurística

- 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, que denotaremos h(n).

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



Heuristica para MBC

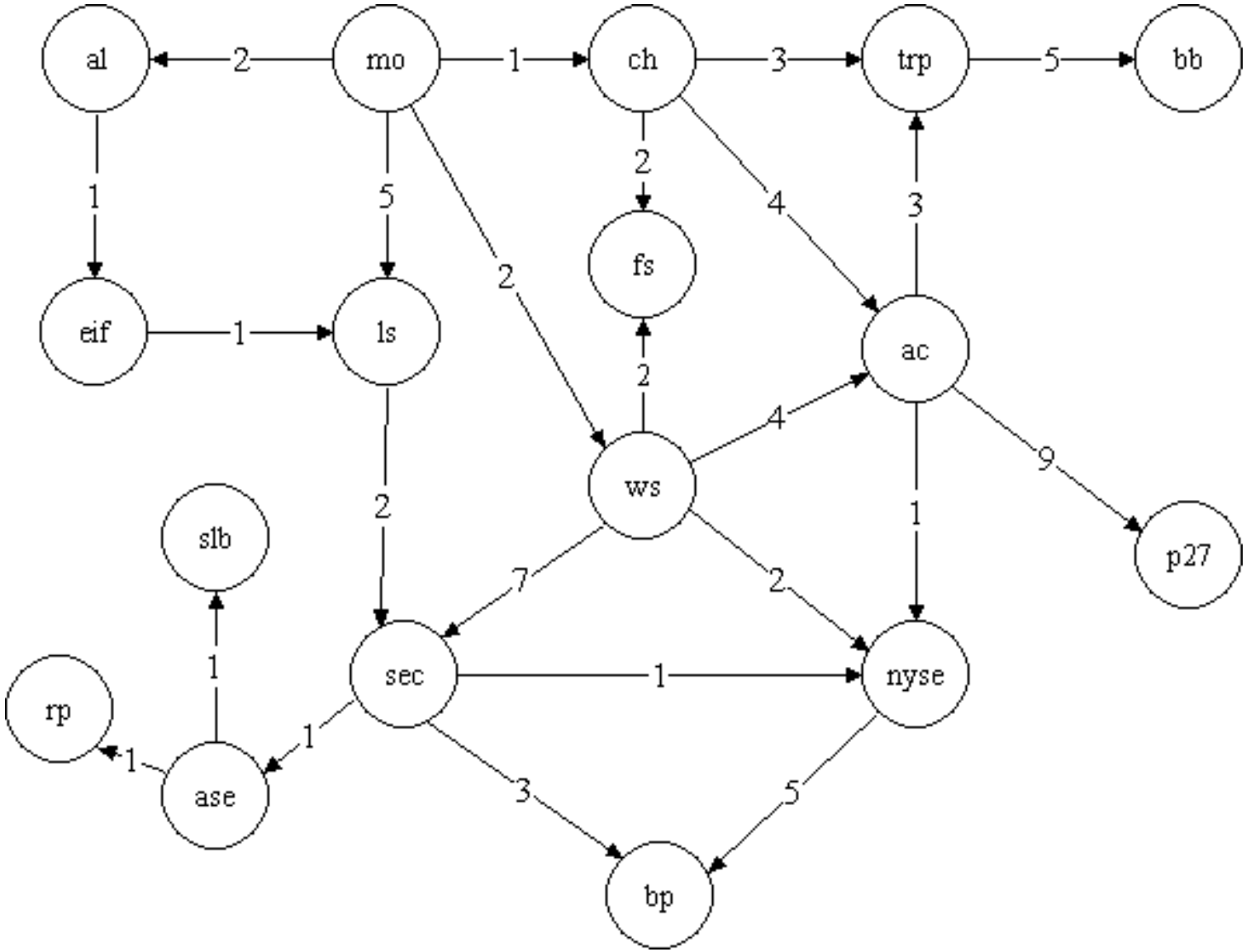


h’(n) = distancia de bloques entre n y el nodo objetivo

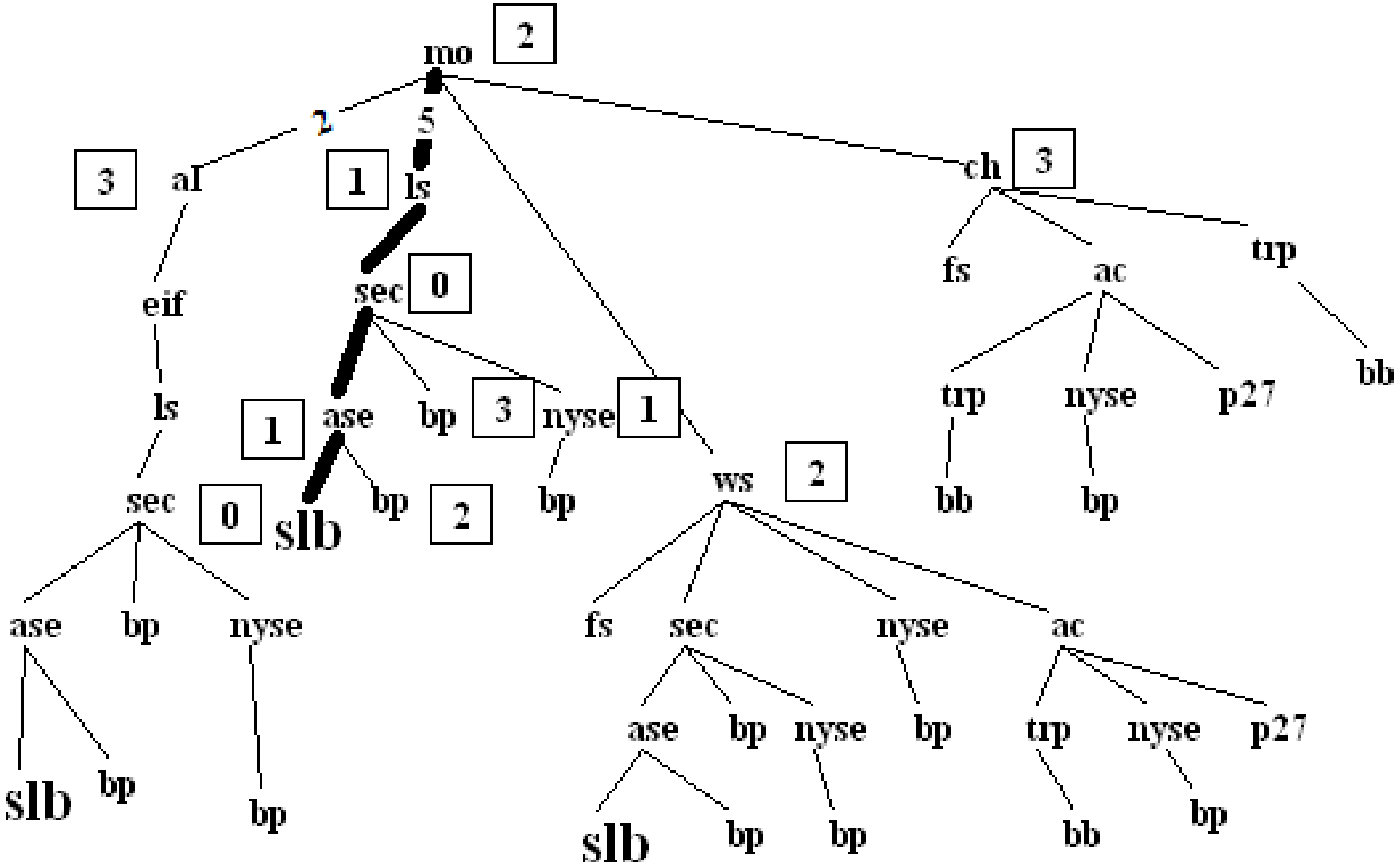
Busqueda Primero el Mejor (BPM)

- 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



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.

Problemas con Búsqueda Primero el Mejor

- En el ejemplo anterior el algoritmo se acerca rápidamente al objetivo
- Pero no encuentra el camino de menor costo
- El algoritmo ignora los costos parciales de cada camino.
  - Esto 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:

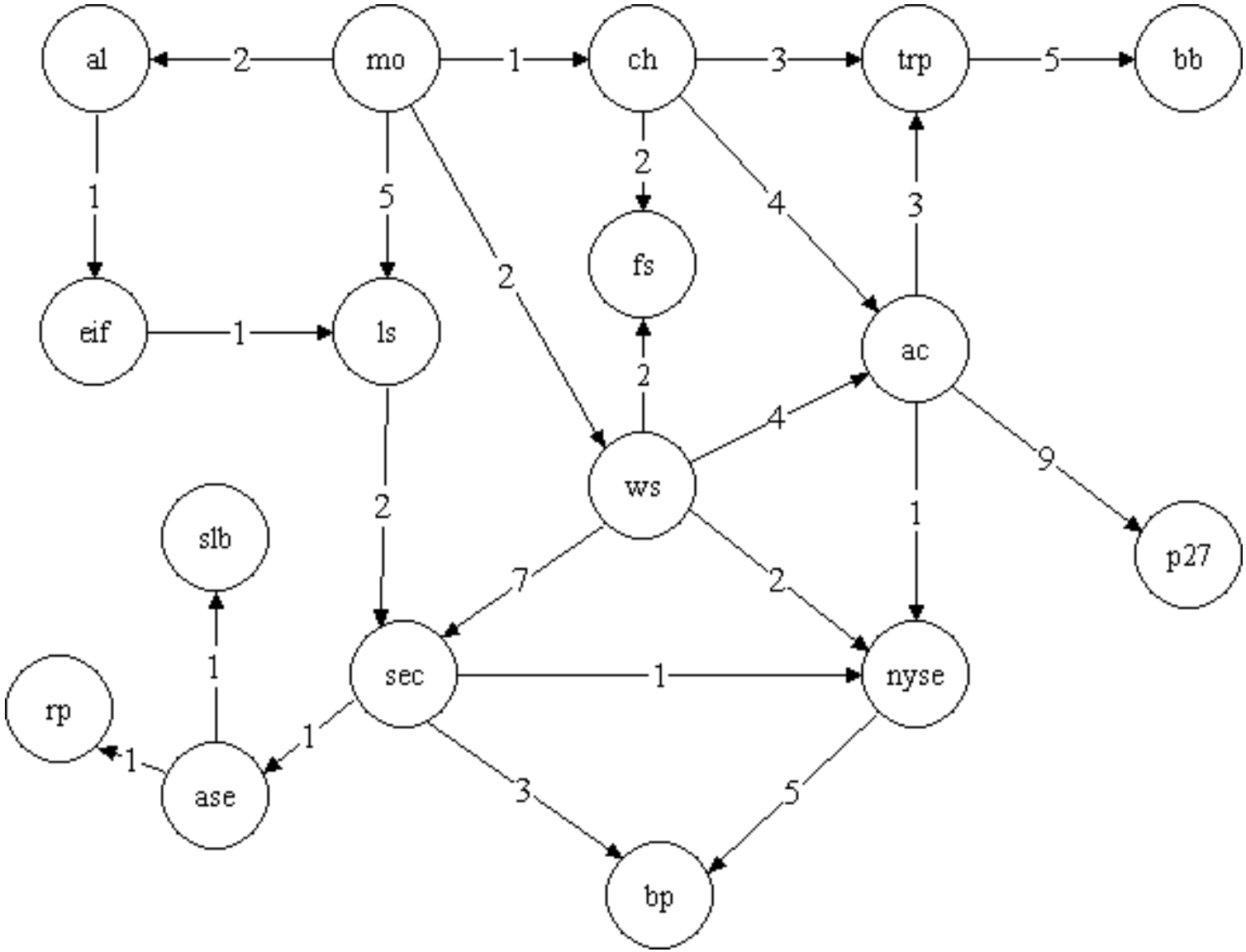
$$f(n) = g(n) + h'(n)$$

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

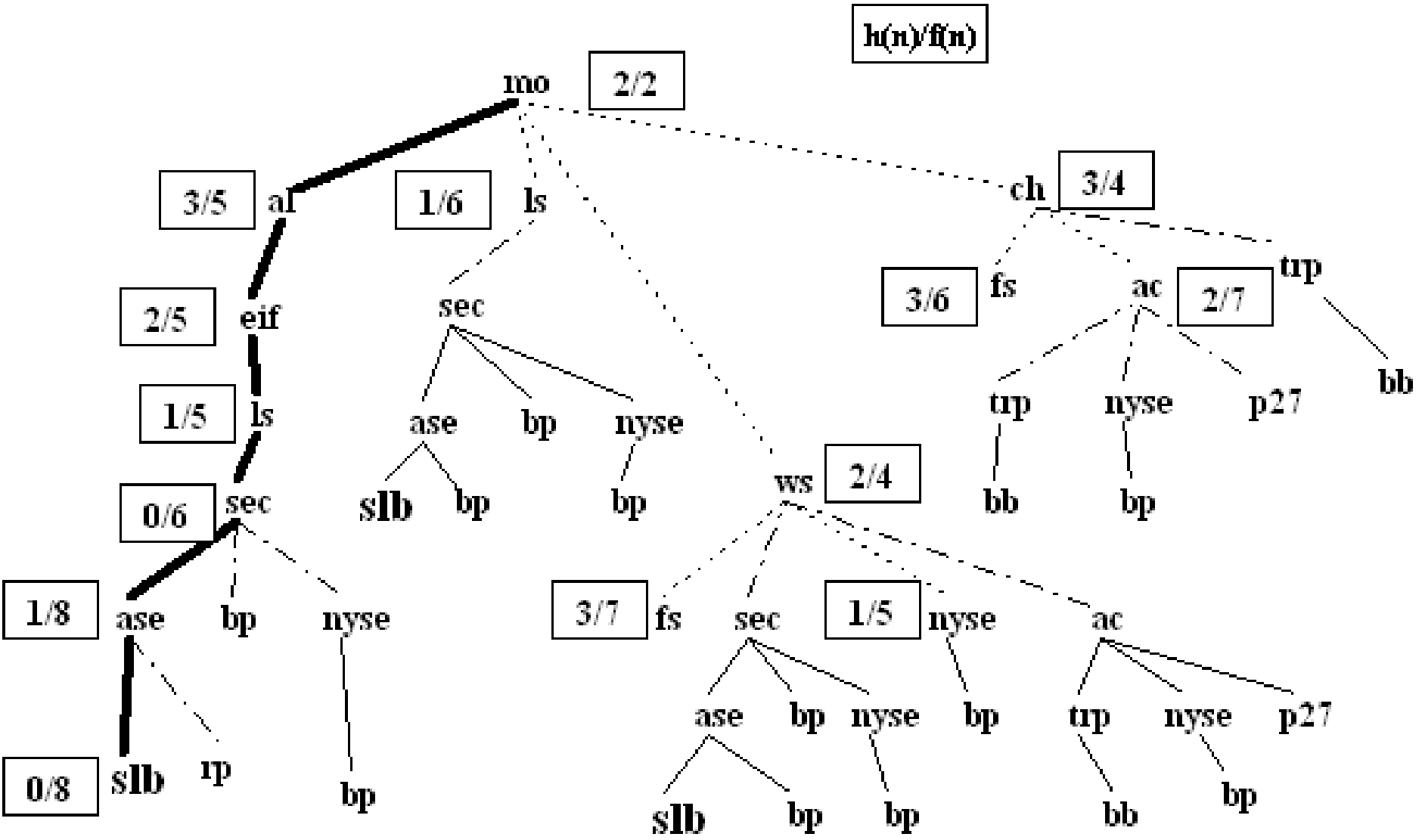
Algoritmo A*

- 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 7 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)