Chapter 7 : Advanced Frequent Pattern Mining

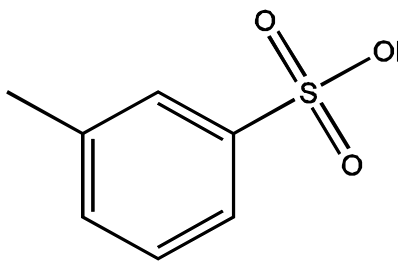
- Pattern Mining: A Road Map
- ~~Pattern Mining in Multi-Level, Multi-Dimensional Space~~
- Constraint-Based Frequent Pattern Mining
- ~~Mining High-Dimensional Data and Colossal Patterns~~
- ~~Mining Compressed or Approximate Patterns~~
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Graph Pattern Mining

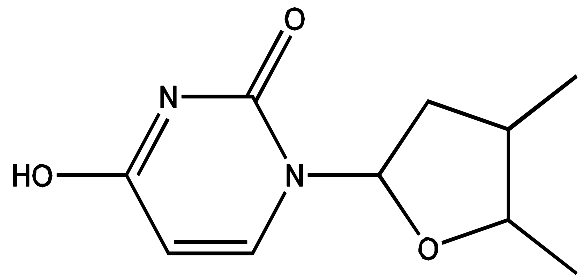
- *Frequent* subgraphs
  - A (sub)graph is ***frequent*** if its *support* (occurrence frequency) in a given dataset is no less than a *minimum support* threshold
- Applications of graph pattern mining
  - Mining biochemical structures
  - Program control flow analysis
  - Mining XML structures or Web communities
  - Building blocks for graph classification, clustering, compression, comparison, and correlation analysis

Example: Frequent Subgraphs

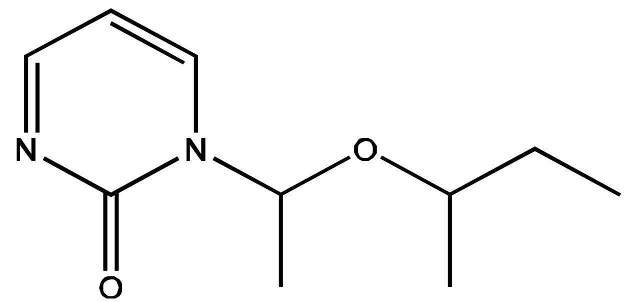
GRAPH DATASET



(A)



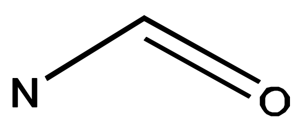
(B)



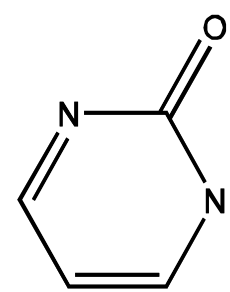
(C)

FREQUENT PATTERNS
(MIN SUPPORT IS 2)

(1)



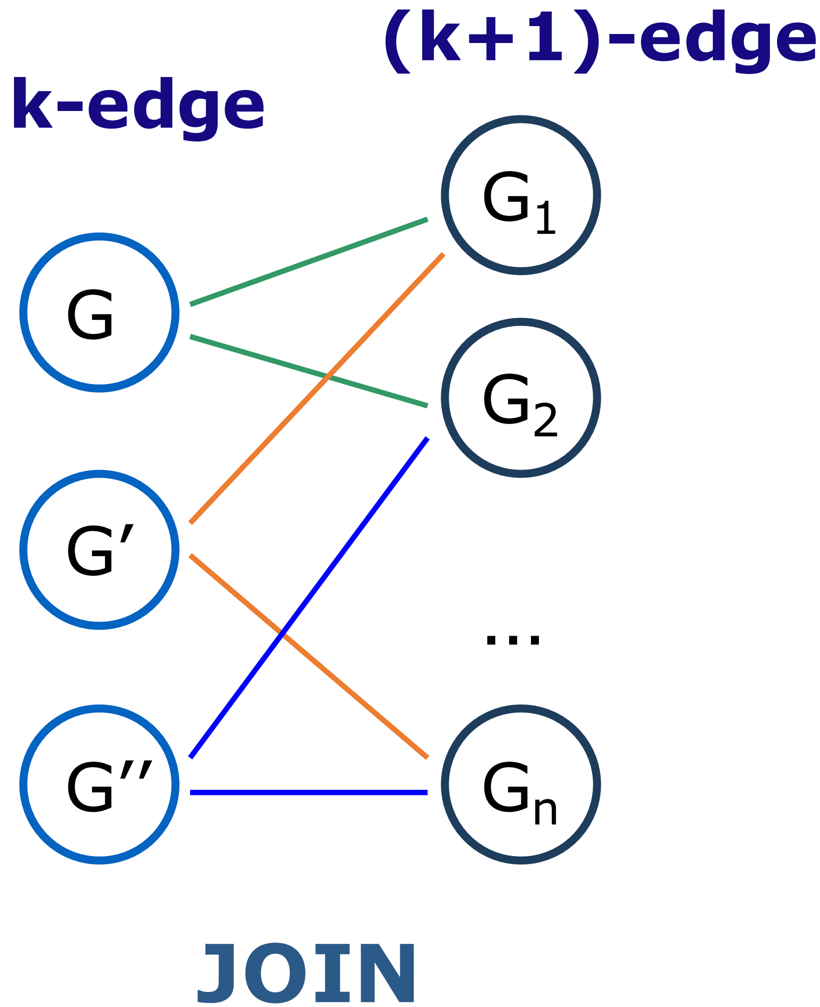
(2)



Properties of Graph Mining Algorithms

- Search order
  - breadth vs. depth
- Generation of candidate subgraphs
  - apriori vs. pattern growth
- Elimination of duplicate subgraphs
  - passive vs. active
- Support calculation
  - embedding store or not
- Discover order of patterns
  - path → tree → graph

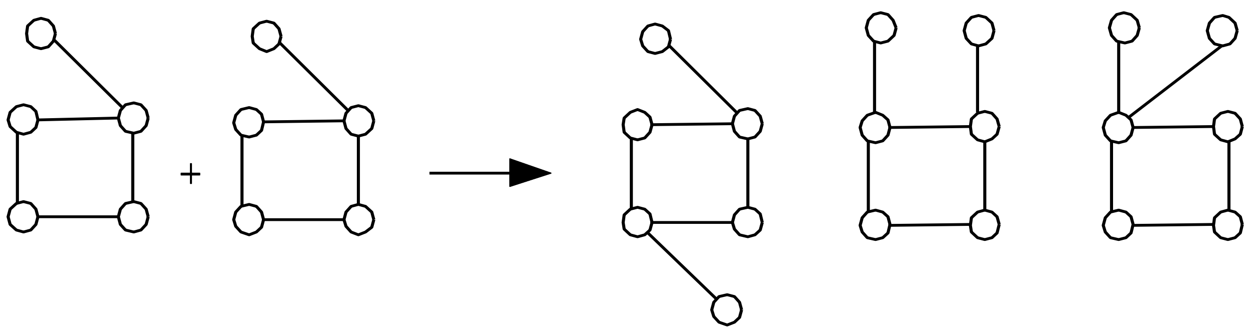
Apriori-Based Approach



Apriori-Based, Breadth-First Search

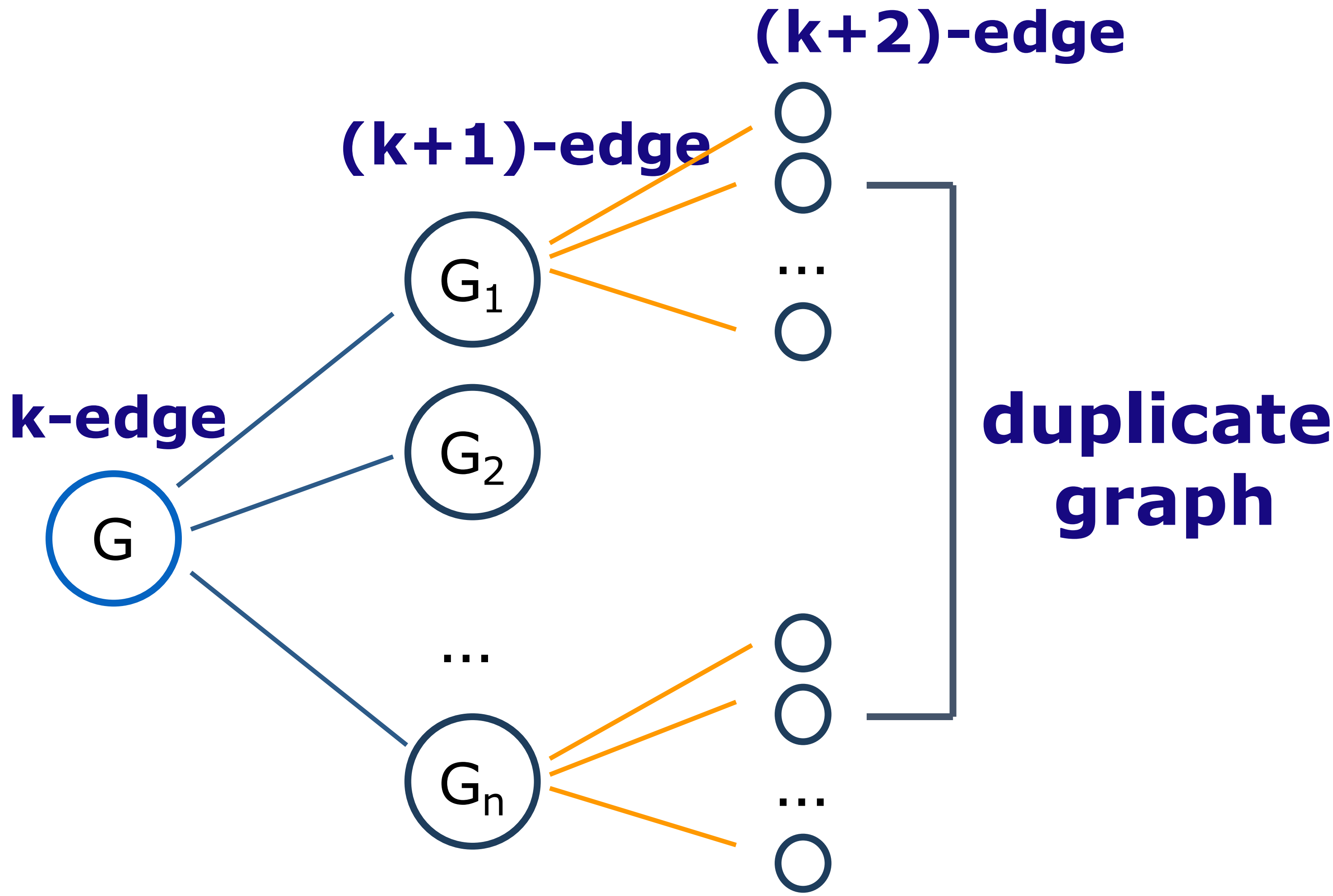
- Methodology: breadth-search, joining two graphs

- AGM (Inokuchi, et al. PKDD'00)
  - generates new graphs with one more node



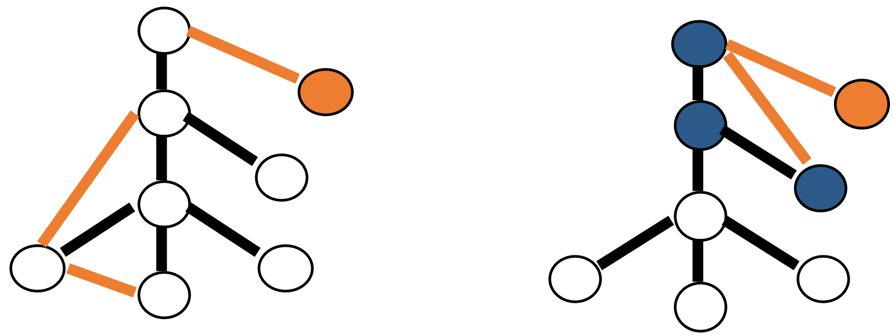
- FSG (Kuramochi and Karypis ICDM'01)
  - generates new graphs with one more edge

Pattern Growth Method



GSPAN (Yan and Han ICDM'02)

Right-Most Extension



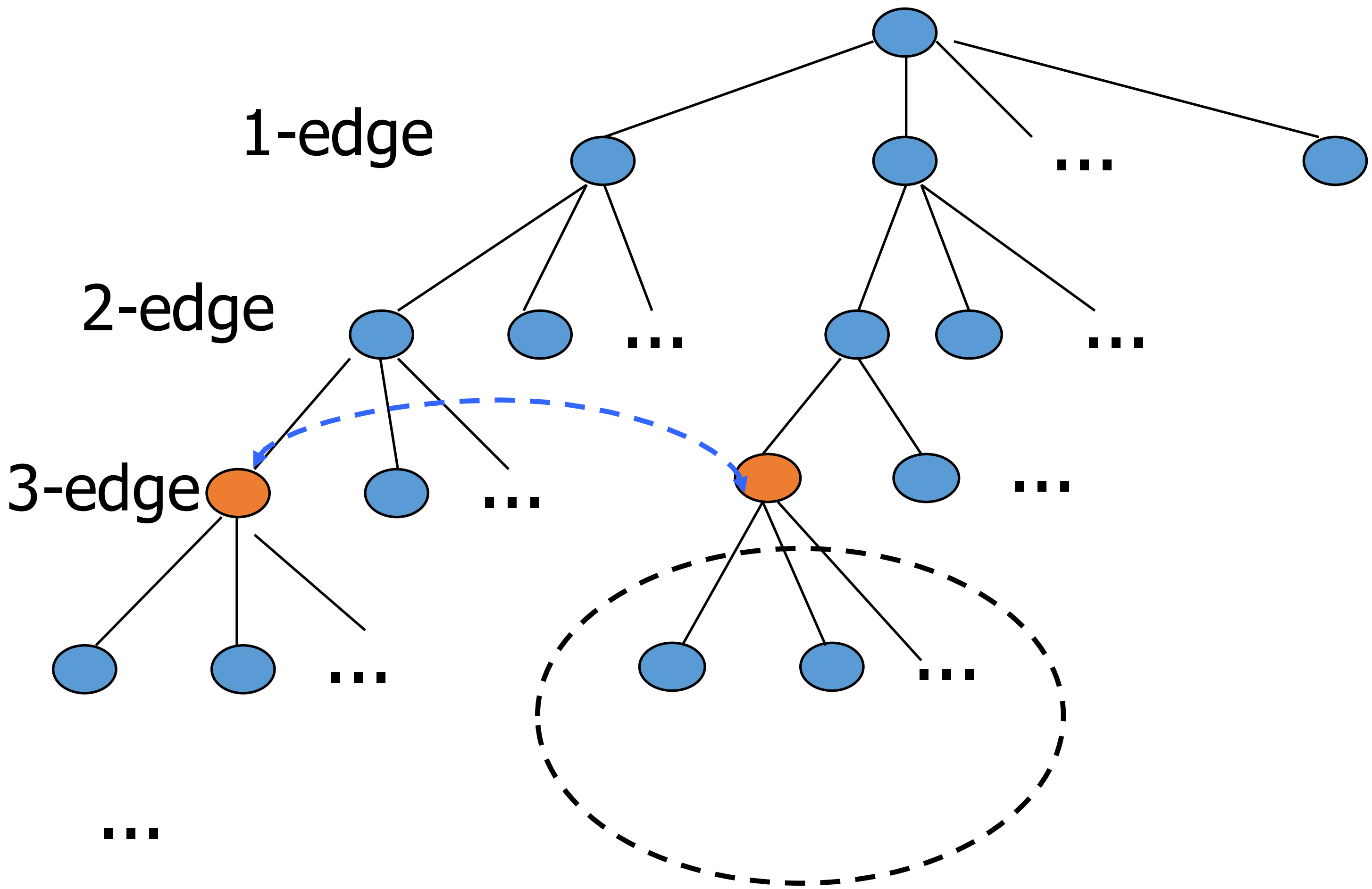
Theorem: Completeness

The Enumeration of Graphs using Right-most Extension is COMPLETE

Lexicographic Ordering in Graph

- It can tell us whether the graph has been discovered.
- And more, the most important, if a graph has been discovered, all its children nodes in the hierarchy must have been discovered.

Lexicographic Ordering in Graph

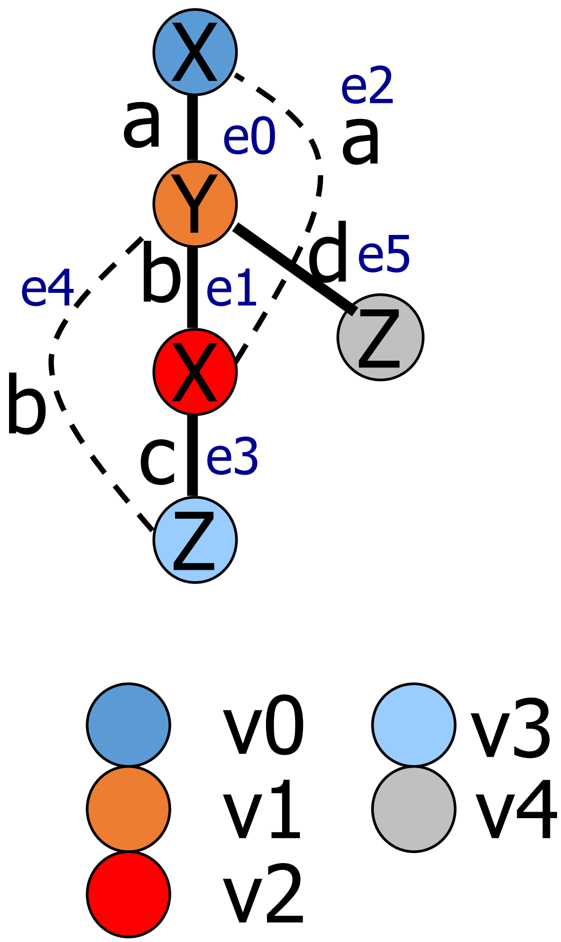


DFS code and Minimum DFS code

- Edge: $(v_i, v_j, l(v_i), l(v_j), l(v_i,v_j))$
  - v_i: node id by DFS
  - $l(v_i)$: node value (e.g. Oxygen, Carbon, Person name)
  - $l(v_i, v_j)$: edge value (e.g. covalent bond, relationship)

- Graph: sequence of edges of following order:
  - Extend one *new* node, add the *forward edge* that connect one node from old nodes to the new node.
  - Add all *backward edges* that connect this new node to other old nodes
  - repeat this procedure.

DFS code



DFS ID

Node value

Edge value

e0: (0,1,x,y,a)

e1: (1,2,y,x,b)

e2: (2,0,x,x,a)

e3: (2,3,x,z,c)

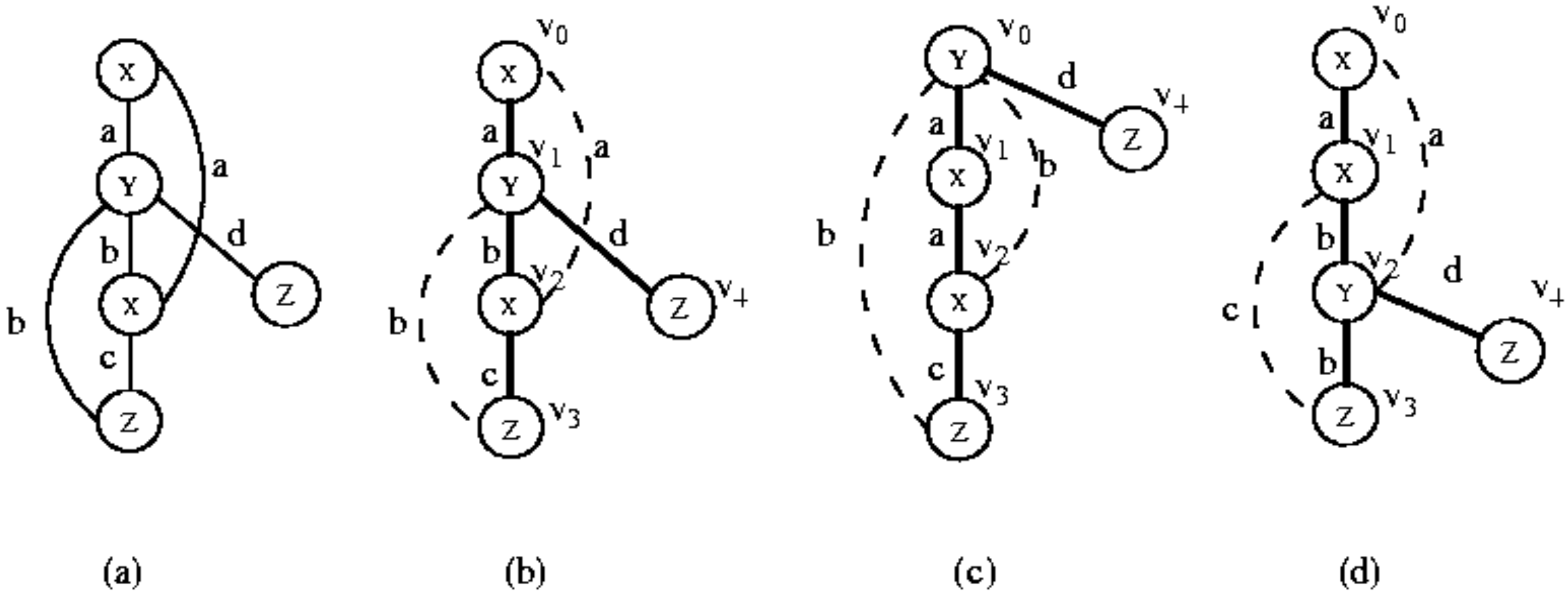
e4: (3,1,z,y,b)

e5: (1,4,y,z,d)

Forward expansion

Backward Linking

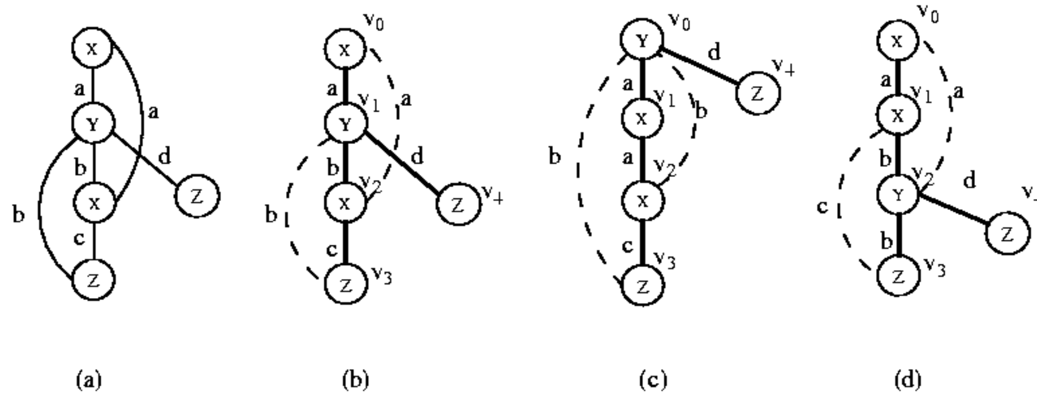
DFS code and Minimum DFS code

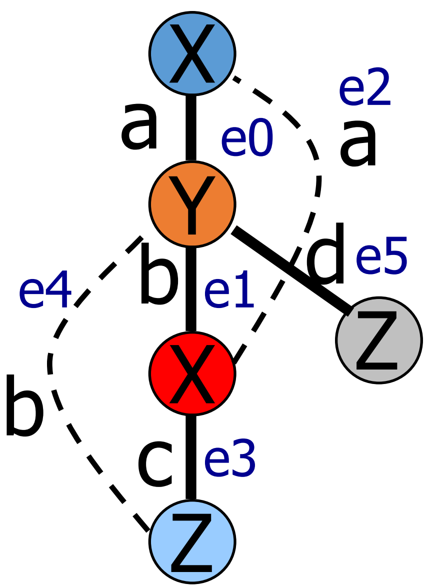


- For same graph, it might have different DFS codes due to its order of nodes and edges visited.
  - (b), (c), (d) are isomorphic graphs of (a)
- Use minimum DFS code to cancel the representation uncertainty.

Minimum DFS code

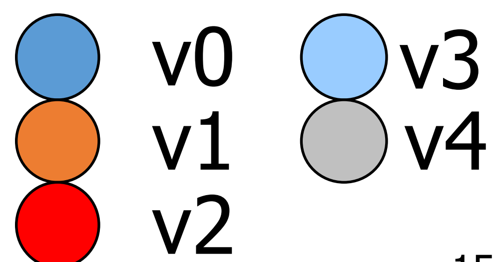
Each Graph may have lots of DFS code (why?):
one smallest lexicographic one is its Minimum DFS Code





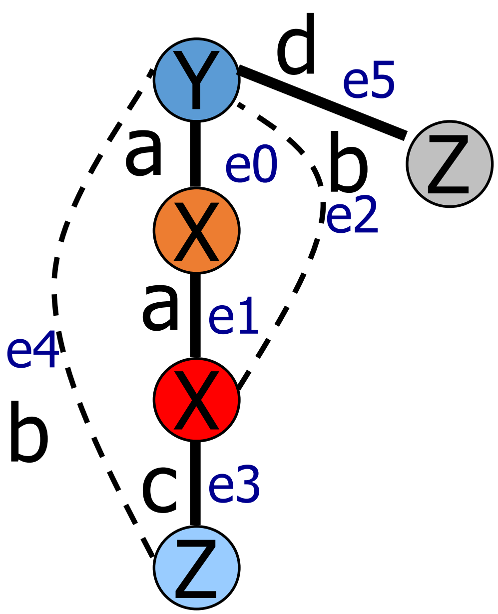
(B)

Edge no.	(B)	(C)	**(D)**
0	(0,1,x,y,a)	(0,1,y,x,a)	(0,1,x,x,a)
1	(1,2,y,x,b)	(1,2,x,x,a)	(1,2,x,y,a)
2	(2,0,x,x,a)	(2,0,x,y,b)	(2,0,y,x,b)
3	(2,3,x,z,c)	(2,3,x,z,c)	(2,3,y,z,b)
4	(3,1,z,y,b)	(3,0,z,y,b)	(3,0,z,x,c)
5	(1,4,y,z,d)	(0,4,y,z,d)	(2,4,y,z,d)



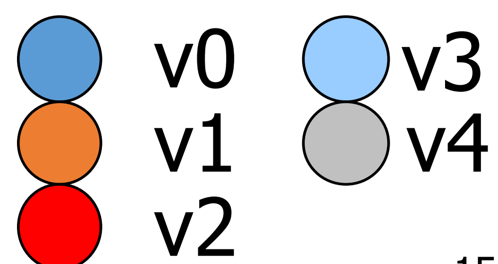
Minimum DFS code

Each Graph may have lots of DFS code (why?):
one smallest lexicographic one is its Minimum DFS Code



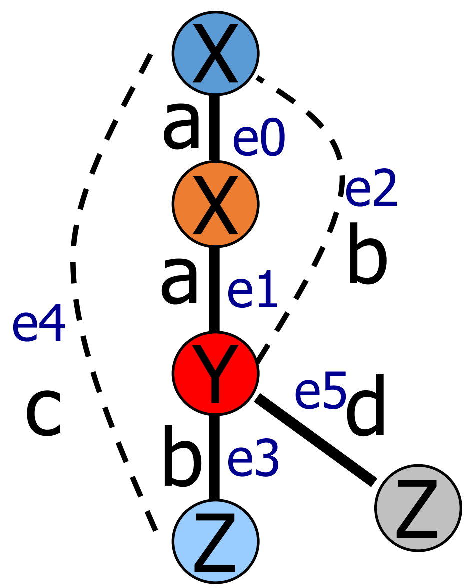
(C)

Edge no.	(B)	(C)	**(D)**
0	(0,1,x,y,a)	(0,1,y,x,a)	(0,1,x,x,a)
1	(1,2,y,x,b)	(1,2,x,x,a)	(1,2,x,y,a)
2	(2,0,x,x,a)	(2,0,x,y,b)	(2,0,y,x,b)
3	(2,3,x,z,c)	(2,3,x,z,c)	(2,3,y,z,b)
4	(3,1,z,y,b)	(3,0,z,y,b)	(3,0,z,x,c)
5	(1,4,y,z,d)	(0,4,y,z,d)	(2,4,y,z,d)



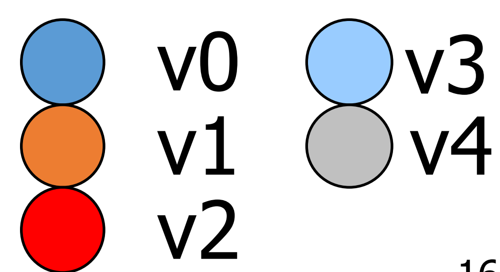
Minimum DFS code

Each Graph may have lots of DFS code (why?):
one smallest lexicographic one is its Minimum DFS Code



(D)

Edge no.	(B)	(C)	**(D)**
0	(0,1,x,y,a)	(0,1,y,x,a)	(0,1,x,x,a)
1	(1,2,y,x,b)	(1,2,x,x,a)	(1,2,x,y,a)
2	(2,0,x,x,a)	(2,0,x,y,b)	(2,0,y,x,b)
3	(2,3,x,z,c)	(2,3,x,z,c)	(2,3,y,z,b)
4	(3,1,z,y,b)	(3,0,z,y,b)	(3,0,z,x,c)
5	(1,4,y,z,d)	(0,4,y,z,d)	(2,4,y,z,d)



Minimum DFS Code

DFS code:
- List of tuples

Lexicographic order:
Compare tuples in sequence alphabetically

C > B > D

Minimum DFS: D

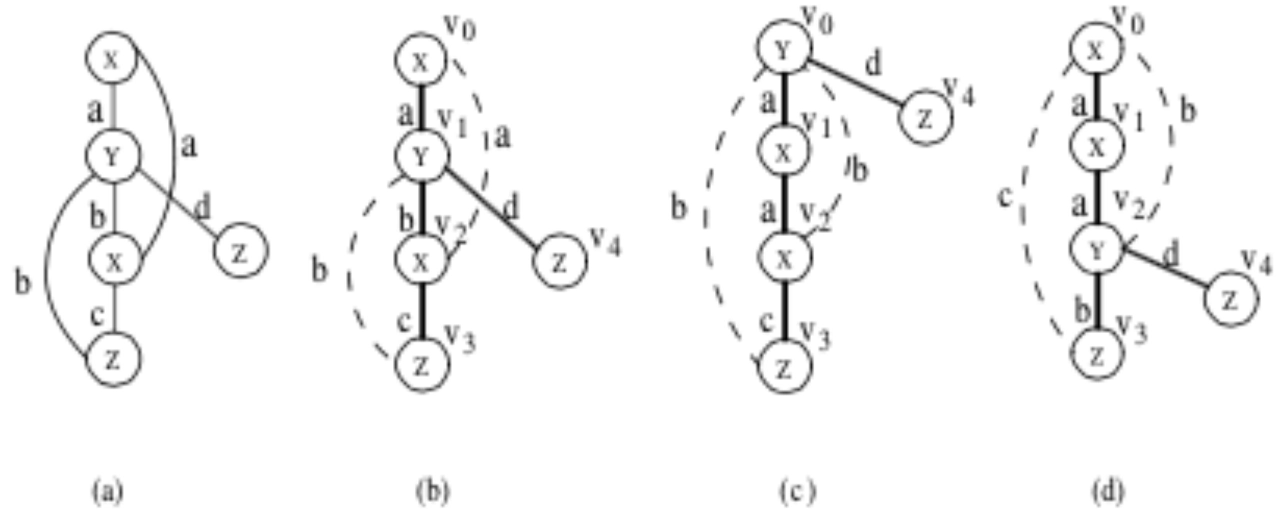


Figure 1. Depth-First Search Tree

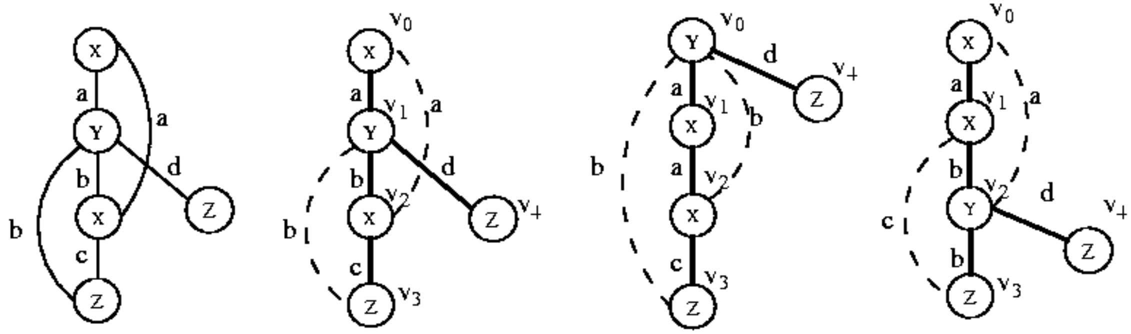
edge	(Fig 1b) α	(Fig 1c) β	(Fig 1d) γ
0	$(0, 1, X, a, Y)$	$(0, 1, Y, a, X)$	$(0, 1, X, a, X)$
1	$(1, 2, Y, b, X)$	$(1, 2, X, a, X)$	$(1, 2, X, a, Y)$
2	$(2, 0, X, a, X)$	$(2, 0, X, b, Y)$	$(2, 0, Y, b, X)$
3	$(2, 3, X, c, Z)$	$(2, 3, X, c, Z)$	$(2, 3, Y, b, Z)$
4	$(3, 1, Z, b, Y)$	$(3, 0, Z, b, Y)$	$(3, 0, Z, c, X)$
5	$(1, 4, Y, d, Z)$	$(0, 4, Y, d, Z)$	$(2, 4, Y, d, Z)$

Table 1. DFS codes for Fig. 1(b)-(d)

How to Generate Minimum DFS code

- Given node labels ordered (e.g., alphabetically)
- Selecting the minimum DFS code
  - select the nodes with the minimum label as the candidate roots
  - construct the DFS spanning tree from each root, always visit edges with smaller labels first; and if two edges with same label, visit the one whose second node has the smaller label
  - insert the backward edges

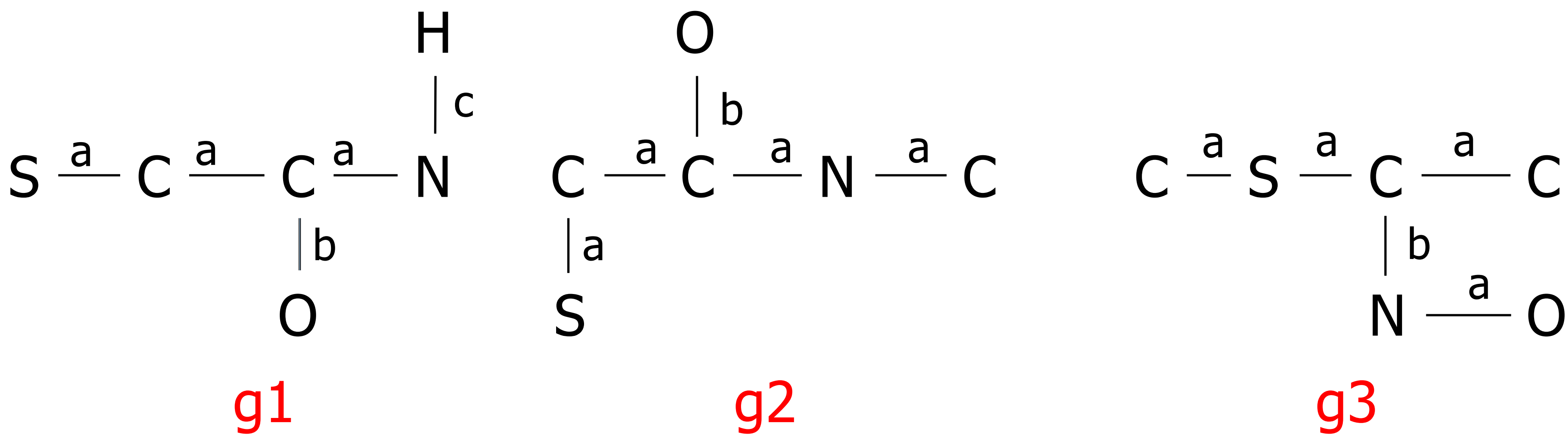
Forward/Backward Edge Set



(a) (b) (c) (d)

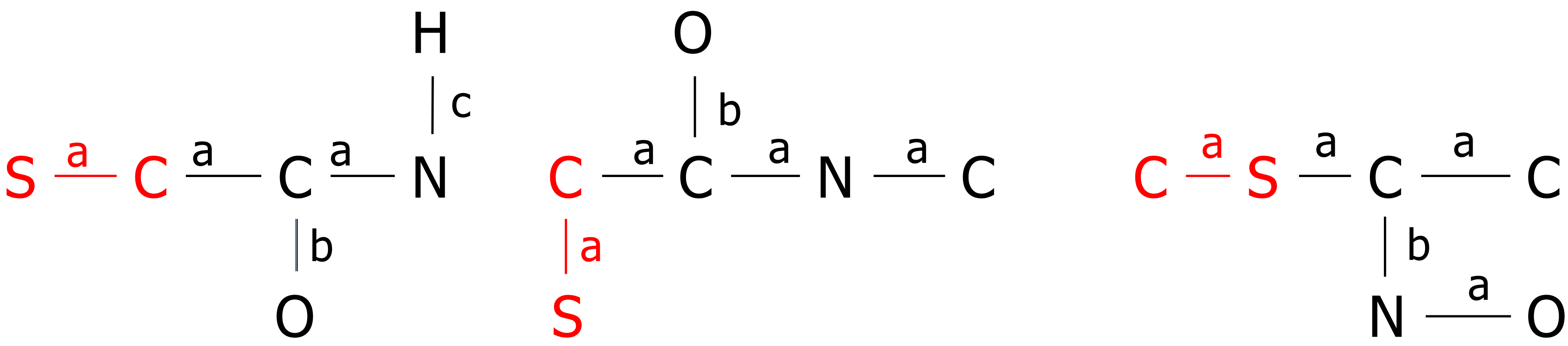
Forward Edge and Backward Edge. Given G_T, the forward edge (*tree edge* [3]) set contains all the edges in the DFS tree, and the backward edge (*back edge* [3]) set contains all the edges which are not in the DFS tree. For simplicity, (i, j) is an ordered pair to represent an edge. If $i < j$, it is a forward edge; otherwise, a backward edge. A linear order, $\prec_T$ is built among all the edges in G by the following rules (assume $e_1 = (i_1, j_1), e_2 = (i_2, j_2)$): (i) if $i_1 = i_2$ and $j_1 < j_2$, $e_1 \prec_T e_2$; (ii) if $i_1 < j_1$ and $j_1 = i_2$, $e_1 \prec_T e_2$; and (iii) if $e_1 \prec_T e_2$ and $e_2 \prec_T e_3$, $e_1 \prec_T e_3$.

Naïve graph pattern mining



min_sup = 2 -> Patterns?

Naïve graph pattern mining

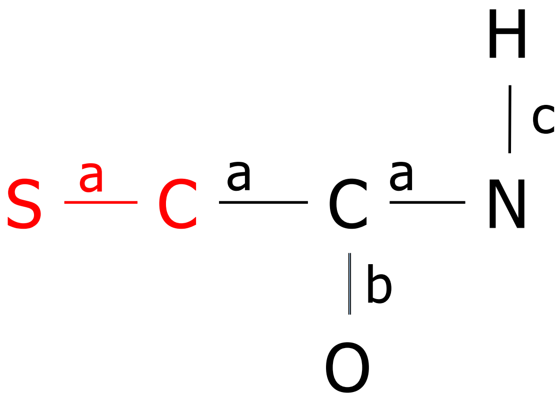


g1 g2 g3

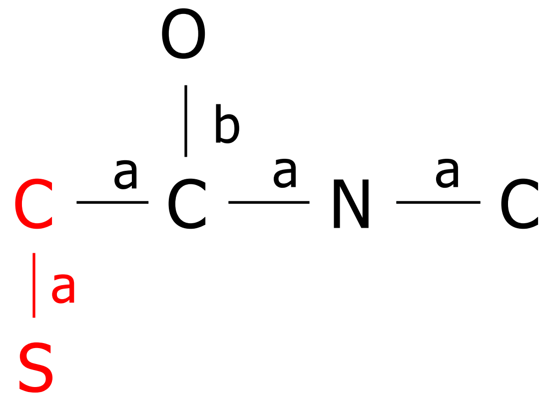
min_sup = 2 -> Patterns?

Sup: 3

Naïve graph pattern mining

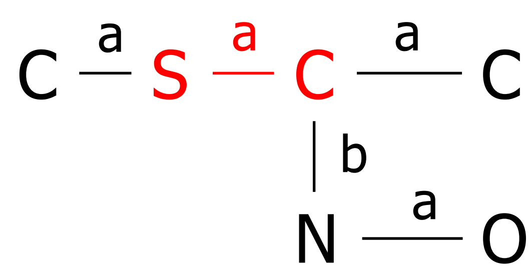


g1



g2

Alternatively,



g3

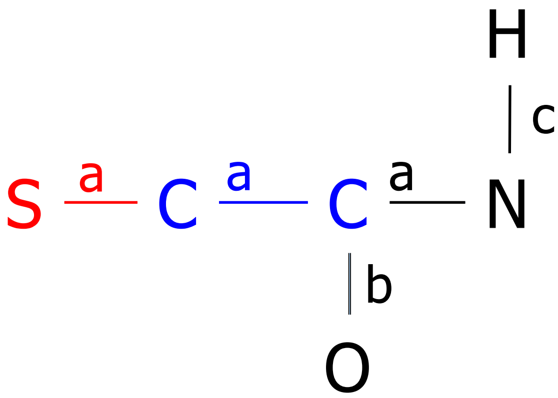
Only number of transaction matters,
Not occurrences

min_sup = 2 -> Patterns?

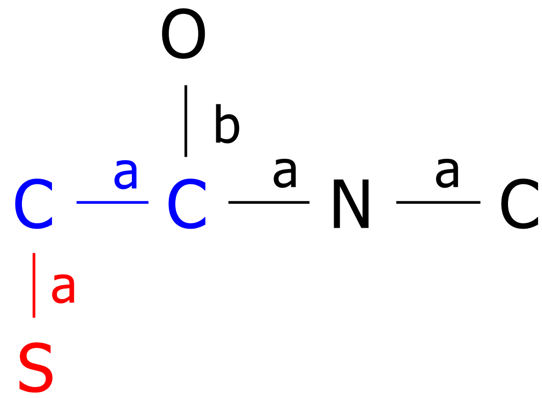
Sup: 3

Naïve graph pattern mining

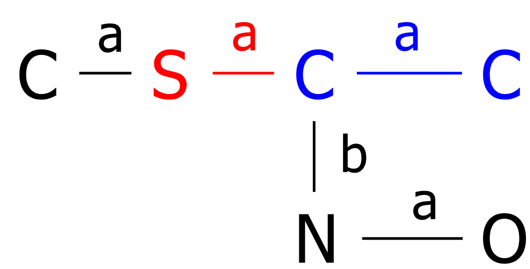
Alternatively,



g1



g2



g3

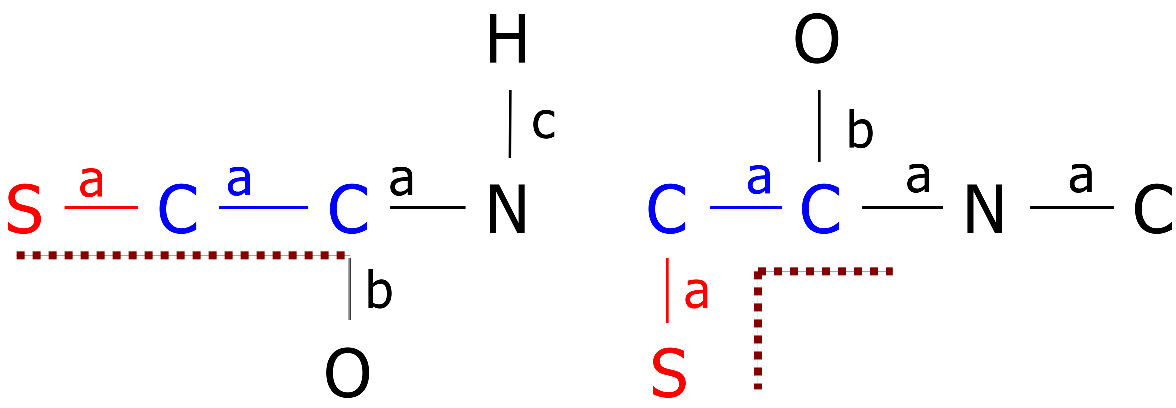
min_sup = 2 -> Patterns?

$S \overset{a}{—} C$ Sup: 3

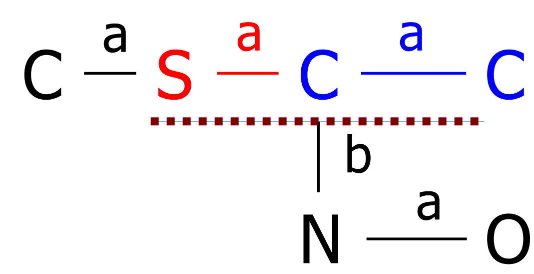
$C \overset{a}{—} C$ Sup: 3

Naïve graph pattern mining

Alternatively,



g1 g2



g3

min_sup = 2 -> Patterns?

$S \overset{a}{\text{—}} C$ Sup: 3

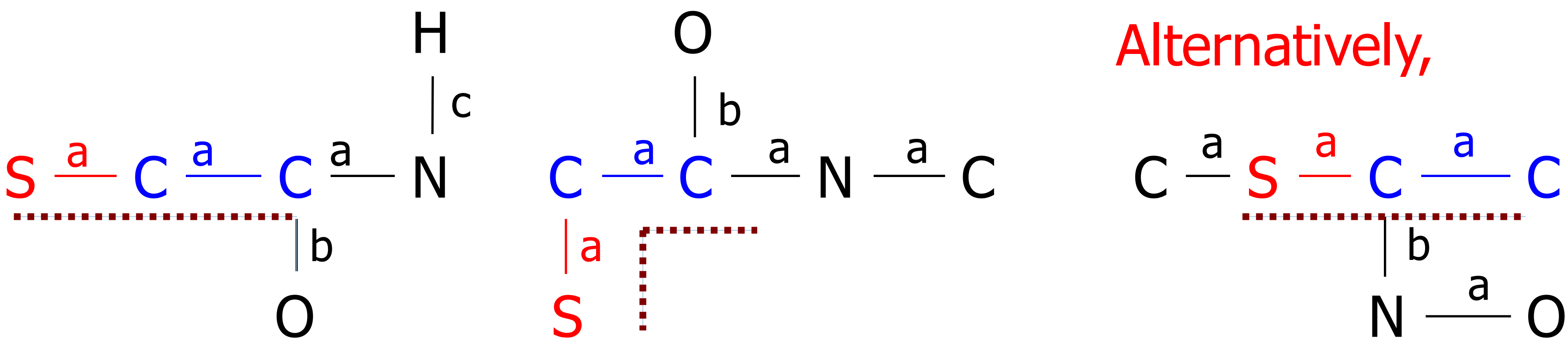
$C \overset{a}{\text{—}} C$ Sup: 3

Pattern growth →

$S \overset{a}{\text{—}} C \overset{a}{\text{—}} C$ Sup: 3

Naïve graph pattern mining

Alternatively,



g1 g2 g3

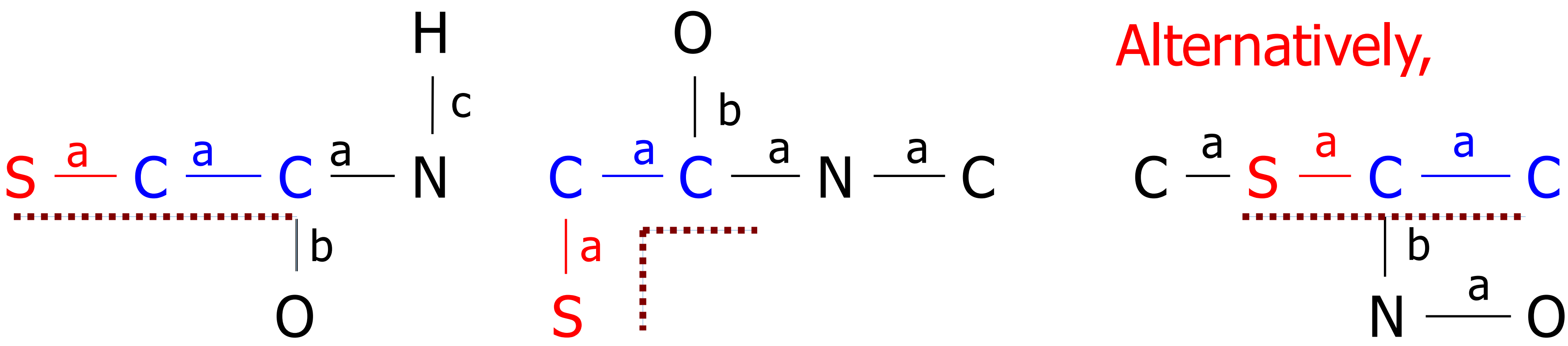
min_sup = 2 -> Patterns?

$S \overset{a}{-} C$ Sup: 3

$C \overset{a}{-} C$ Sup: 3

$\xrightarrow{\text{Pattern growth}}$ $S \overset{a}{-} C \overset{a}{-} C$ Sup: 3

Naïve graph pattern mining



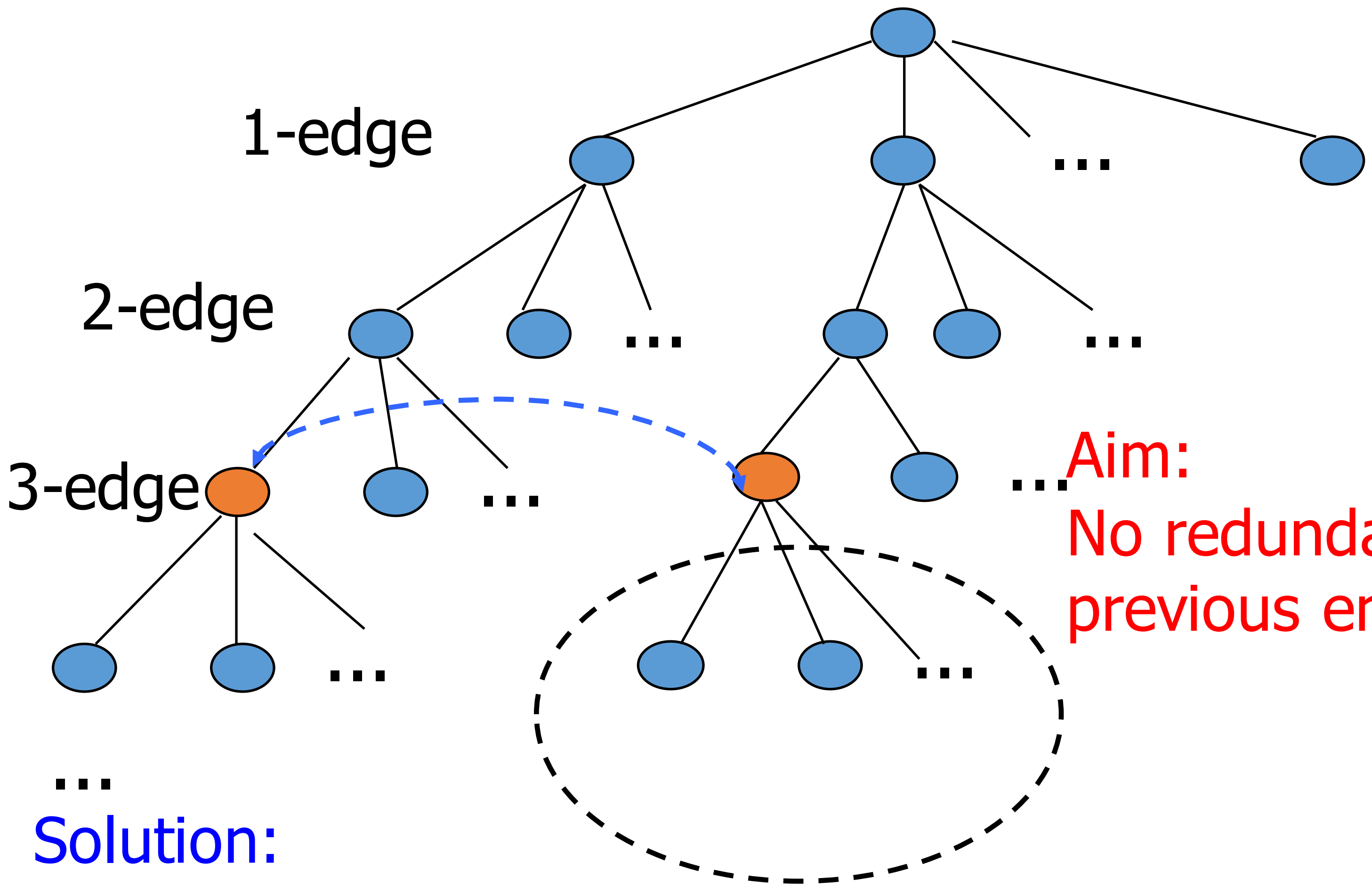
g1 g2 g3

min_sup = 2 -> Patterns?

- Generate candidate growth patterns
- Remove duplication and find frequent ones
- Repeat

generating the same pattern Multiple times is inefficient

Rightmost extension



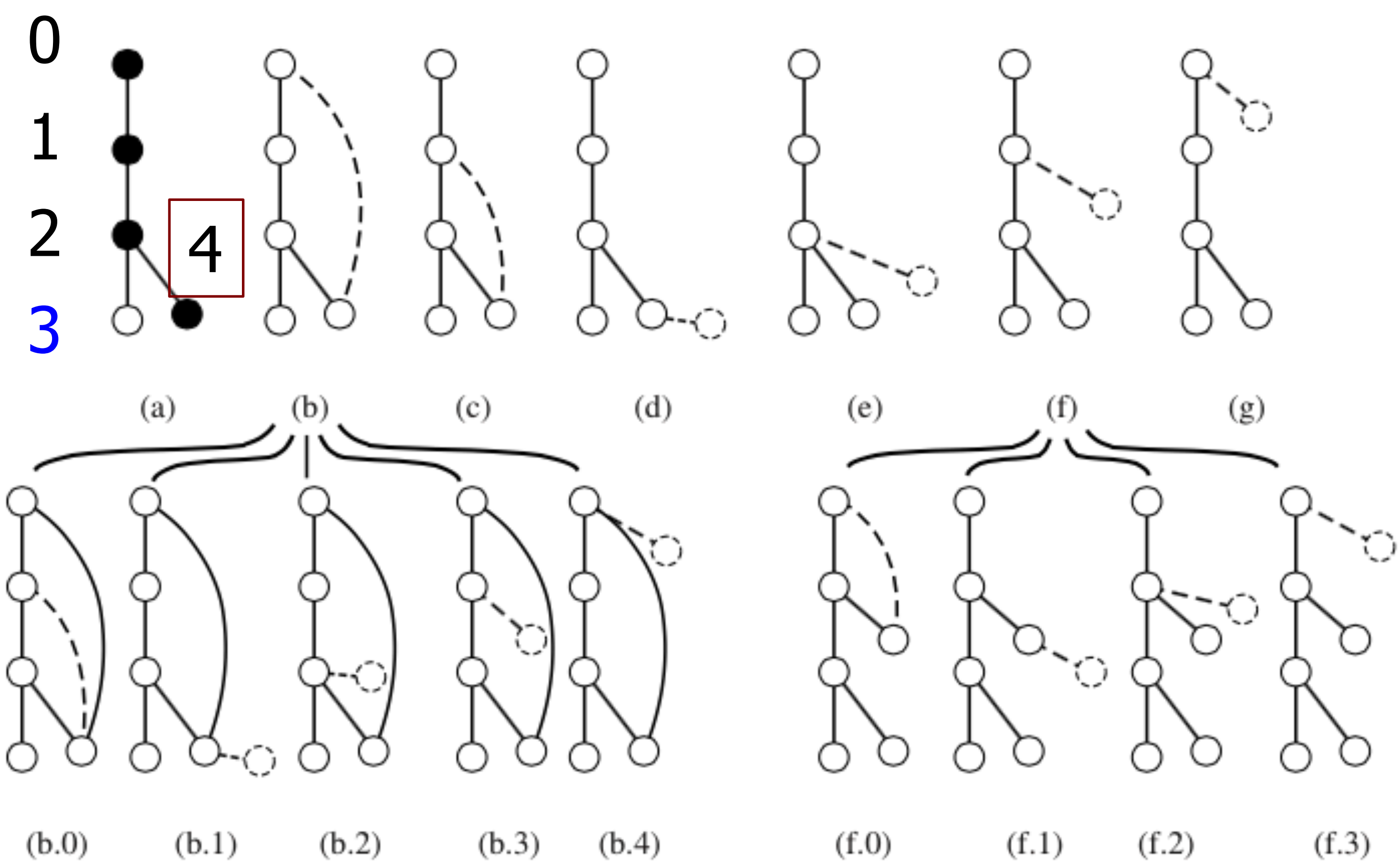
Aim:

No redundant work on previous encountered graph

Solution:

- Fix an order for pattern tree growth: right-most extension
- Test identical patterns: minimum DFS code

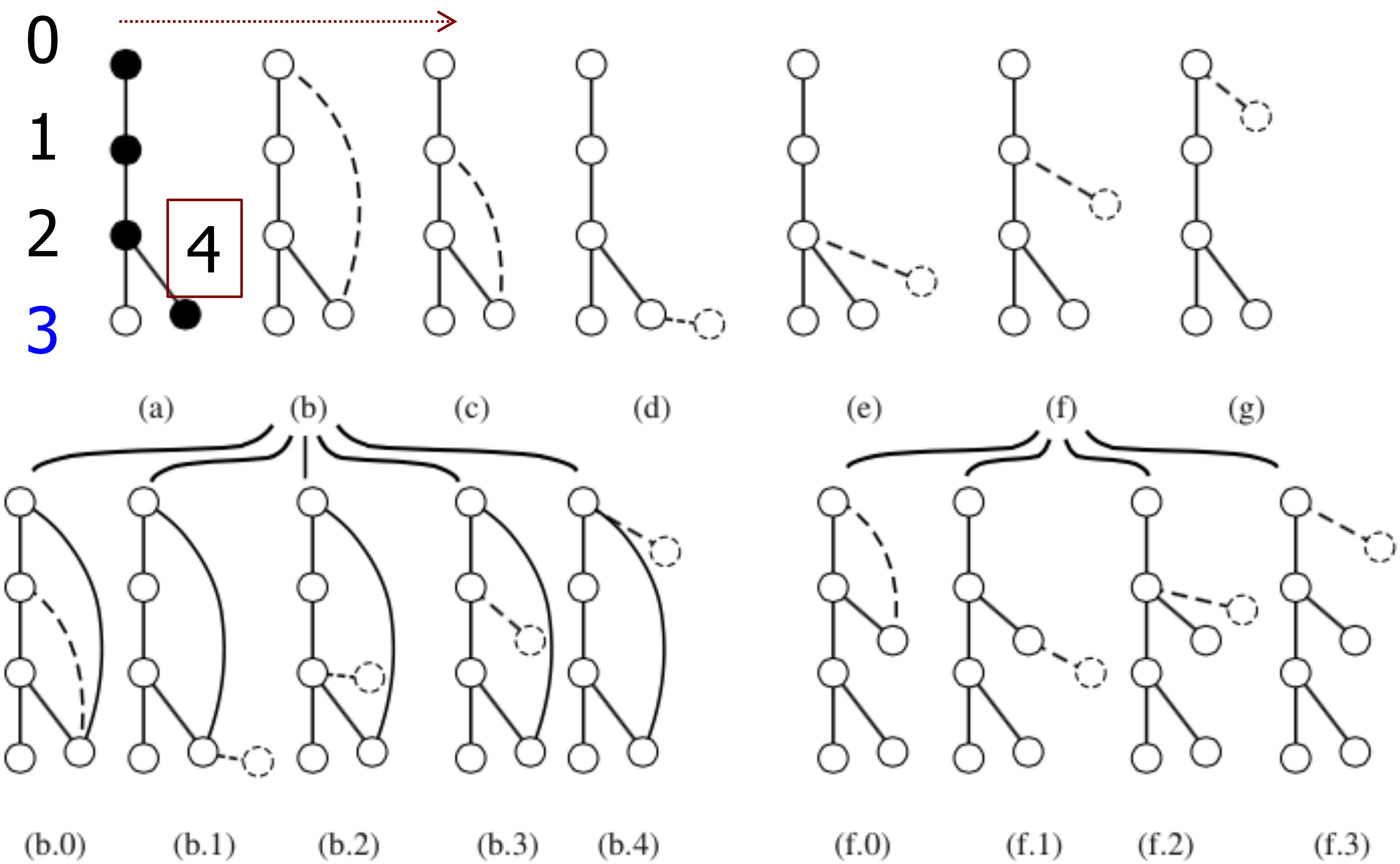
Rightmost Extension to Grow Patterns



Rightmost Path: 0, 1, 2, 4
Rightmost Node: 4

Rightmost Extension to Grow Patterns

First, link backward edges from current RM node to old nodes *in order of nodes*

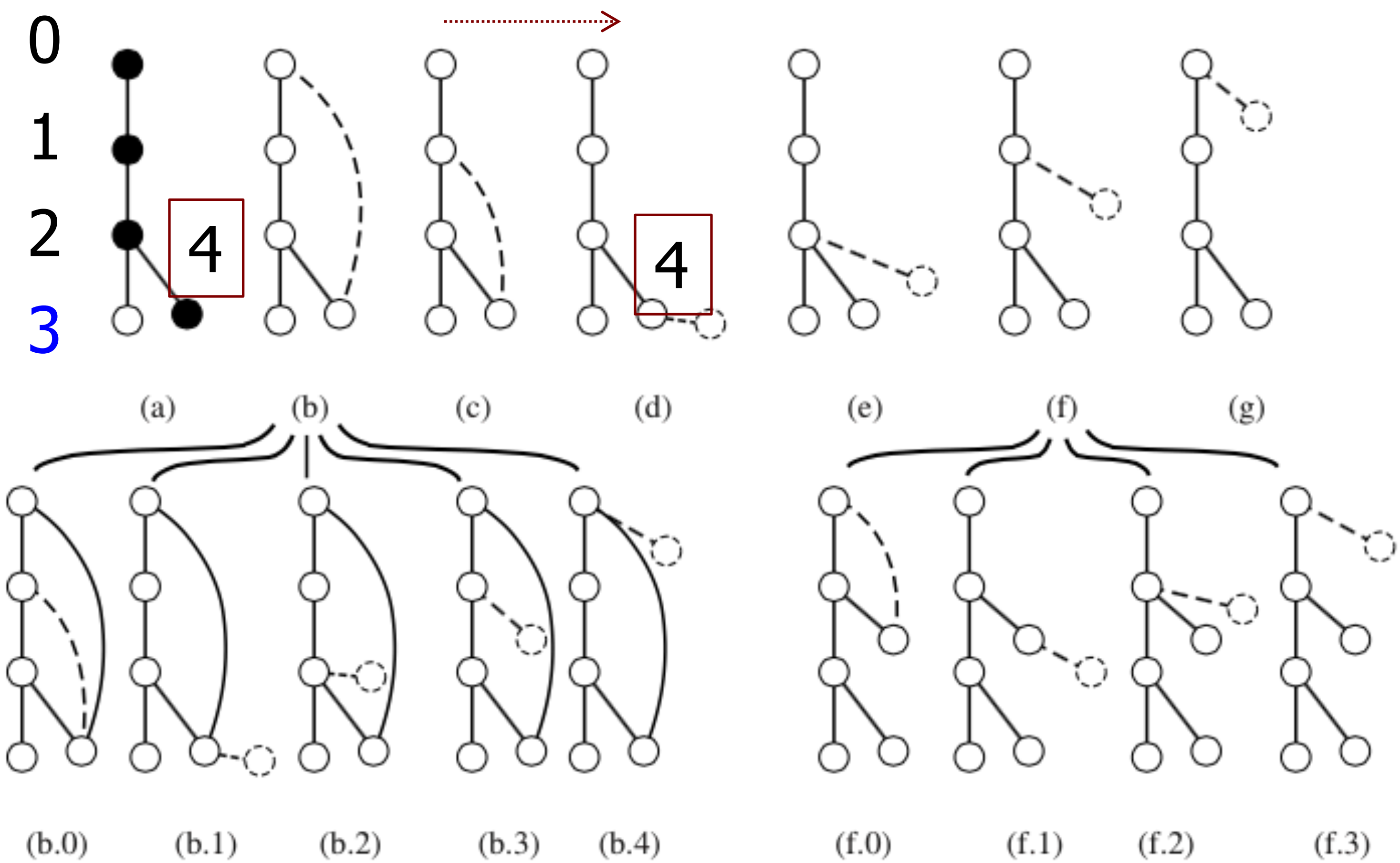


Rightmost Path: 0, 1, 2, 4

Rightmost Node: 4

Rightmost Extension to Grow Patterns

Then, extend forward edge from RM node

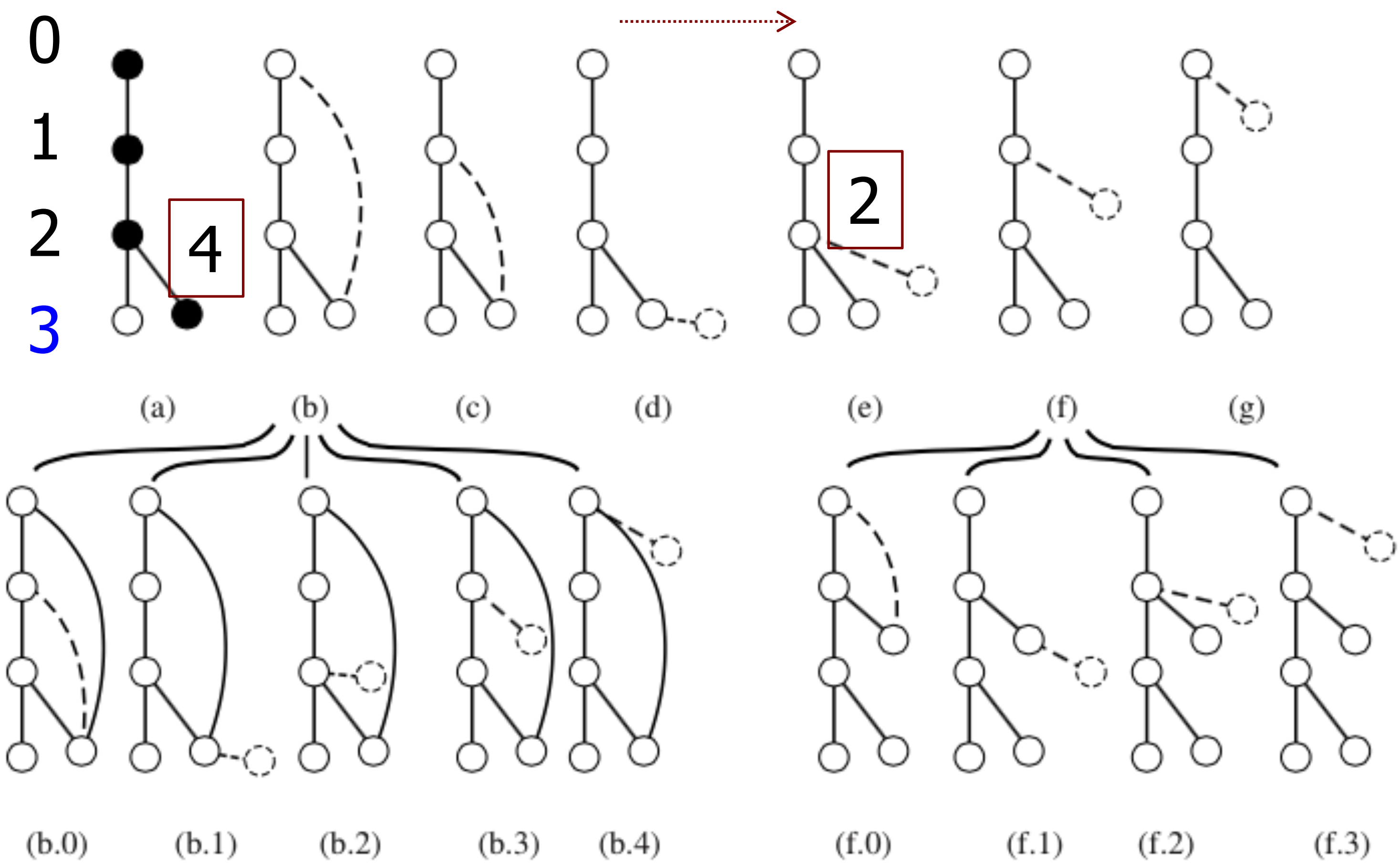


Rightmost Path: 0, 1, 2, 4

Rightmost Node: 4

Rightmost Extension to Grow Patterns

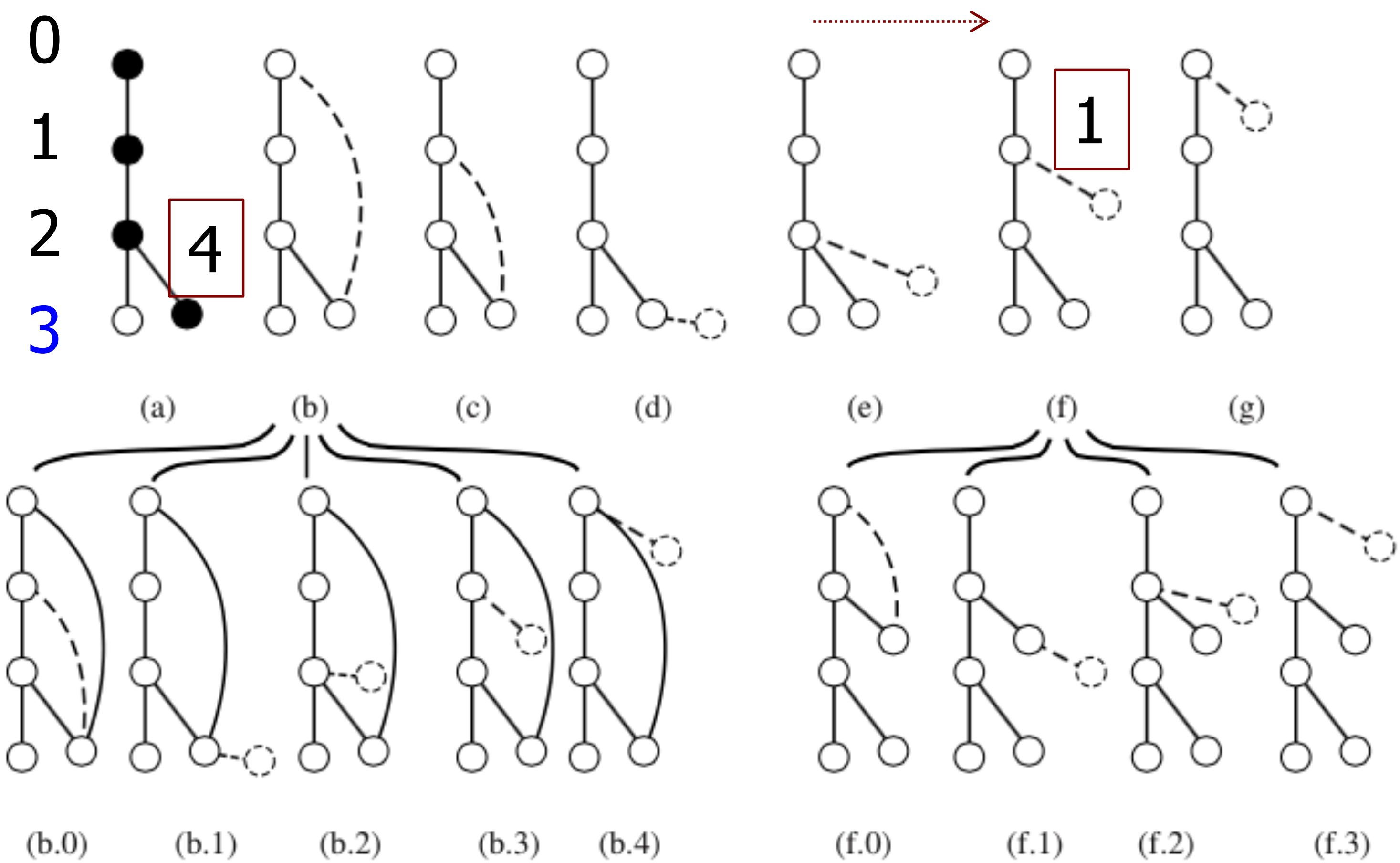
If not possible, use next available RM node, i.e., node 2



Rightmost Path: 0, 1, 2
Rightmost Node: 2

Rightmost Extension to Grow Patterns

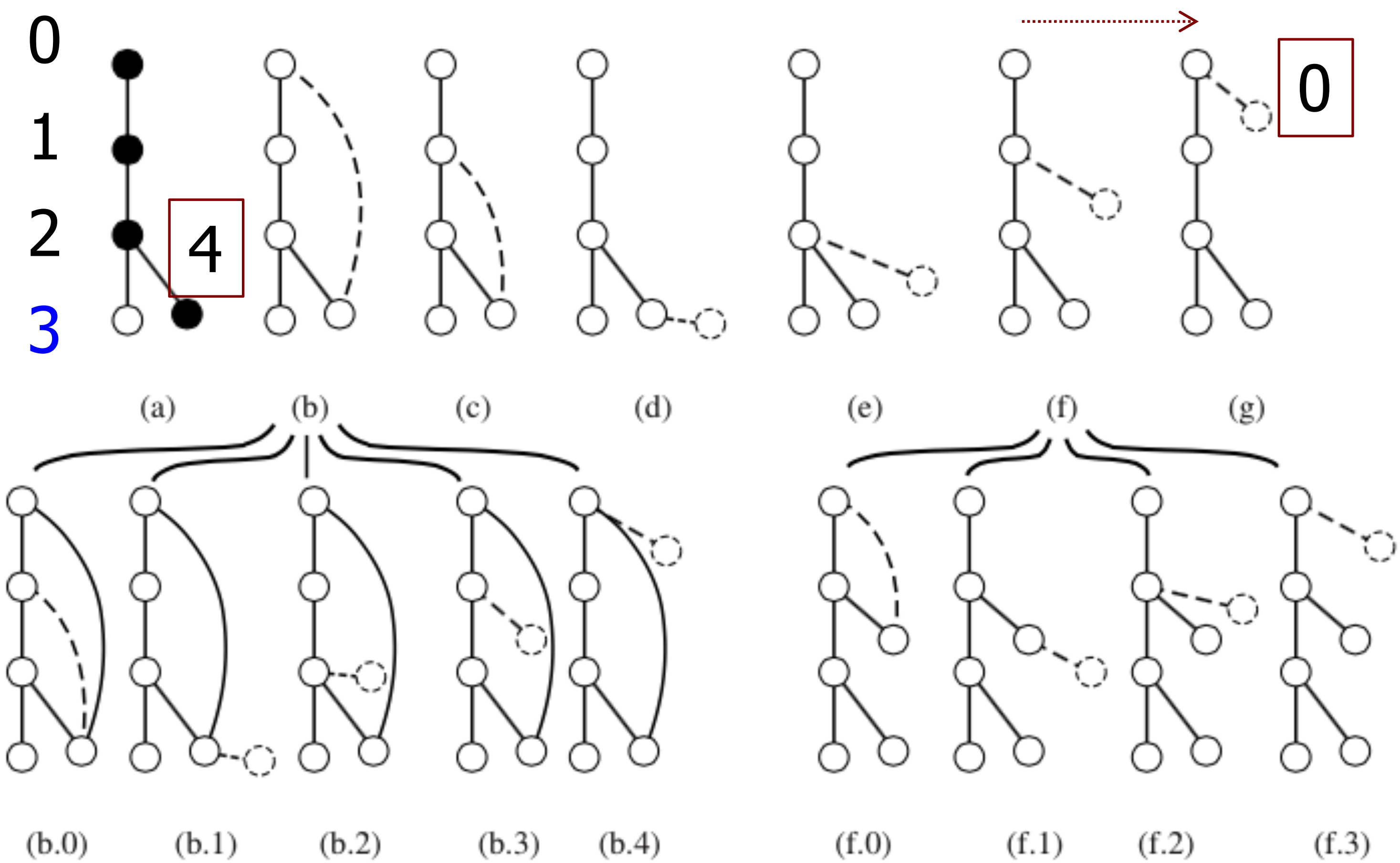
If not possible, use next available RM node, i.e., node 1



Rightmost Path: 0, 1
Rightmost Node: 1

Rightmost Extension to Grow Patterns

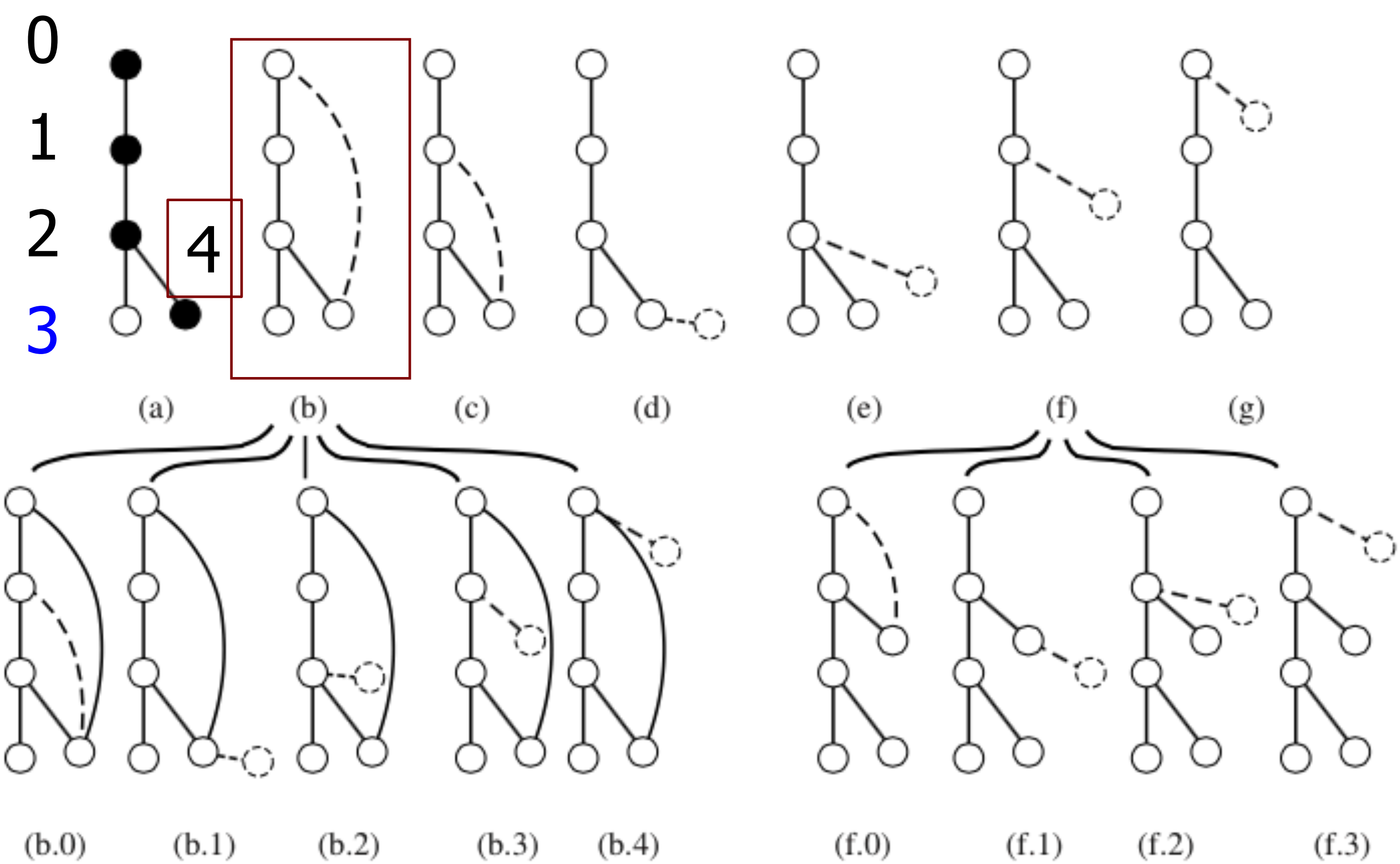
If not possible, use next available RM node, until extension terminates



Rightmost Path: 0
Rightmost Node: 0

Rightmost Extension to Grow Patterns

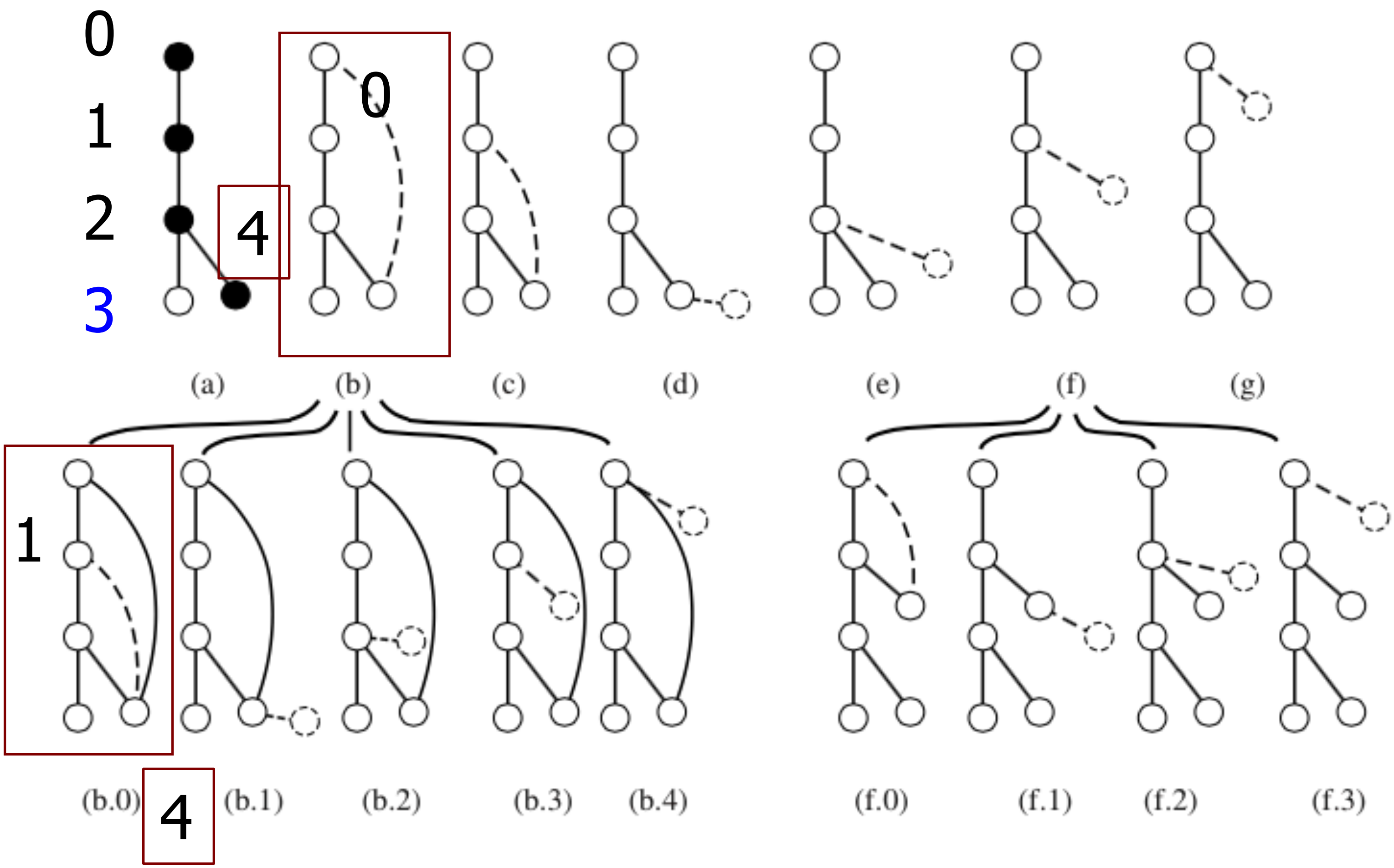
If backward edge 4->0 exists, add 4->0, and repeat



Rightmost Path: 0, 1, 2, 4
Rightmost Node: 4

Rightmost Extension to Grow Patterns

First, if backward 4->1 exists, link it

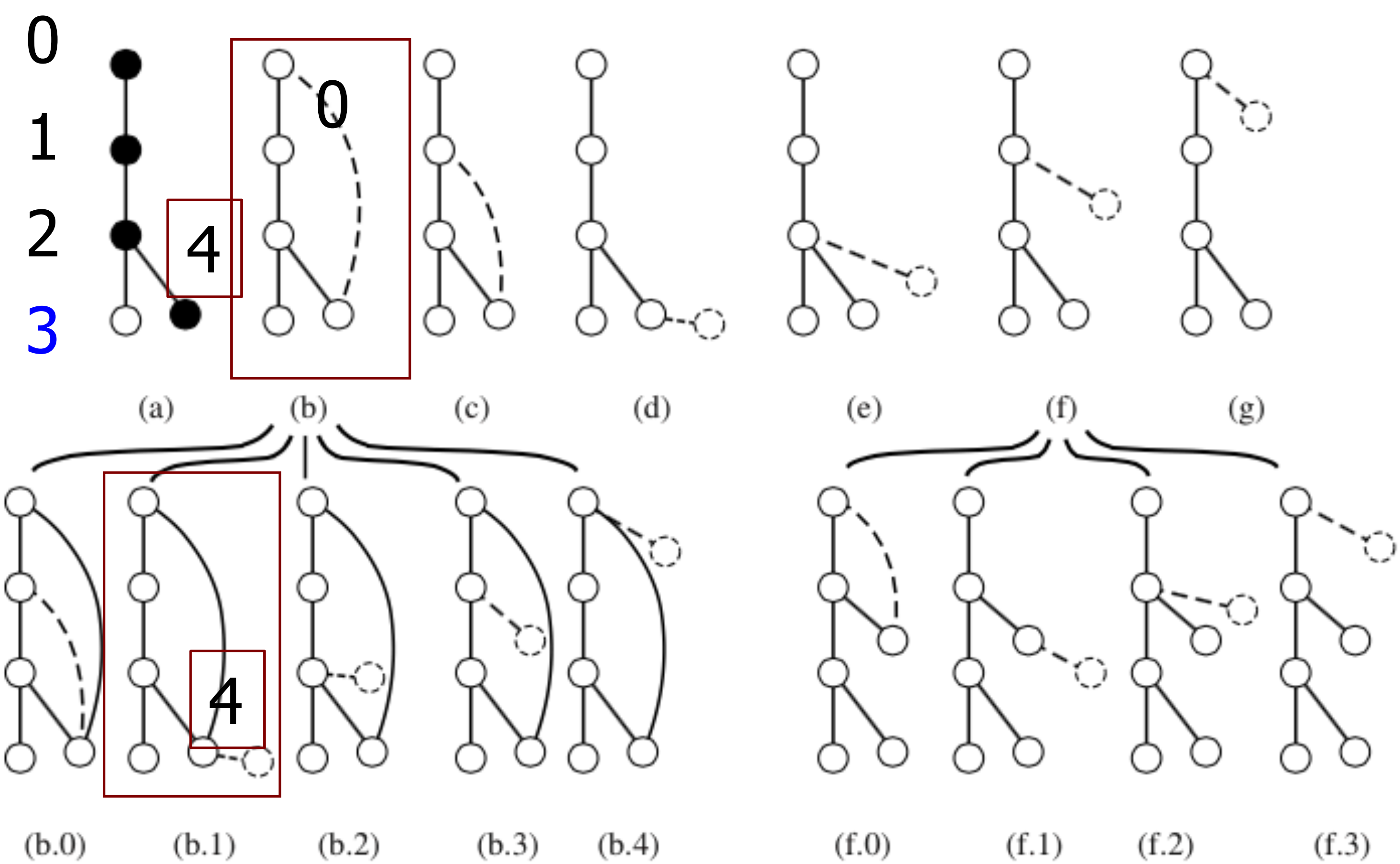


Rightmost Path: 0, 1, 2, 4

Rightmost Node: 4

Rightmost Extension to Grow Patterns

Otherwise, try extend RM Node 4,

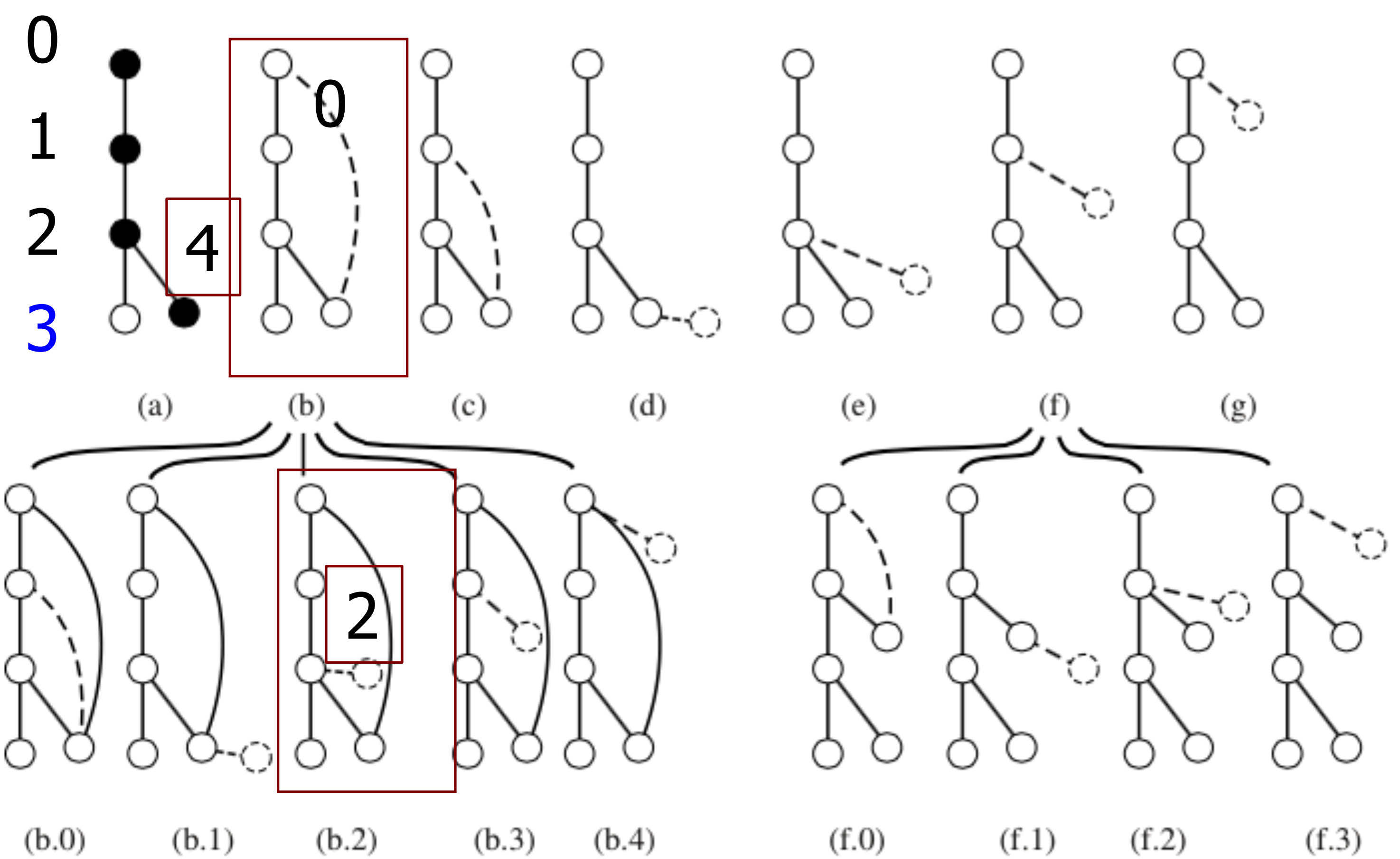


Rightmost Path: 0, 1, 2, 4

Rightmost Node: 4

Rightmost Extension to Grow Patterns

Then, use 2 as RM node, try forward extension

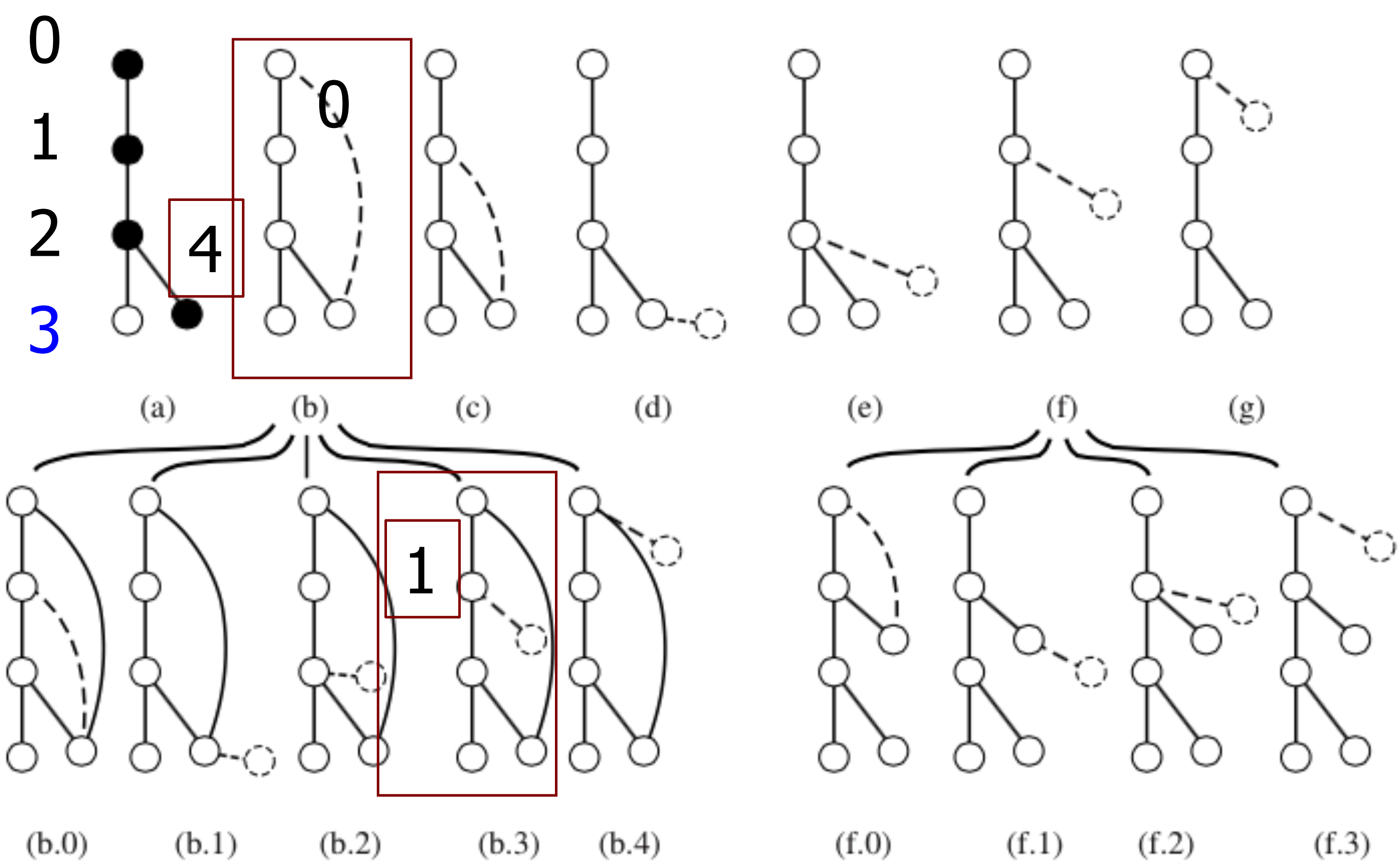


Rightmost Path: 0, 1, 2

Rightmost Node: 2

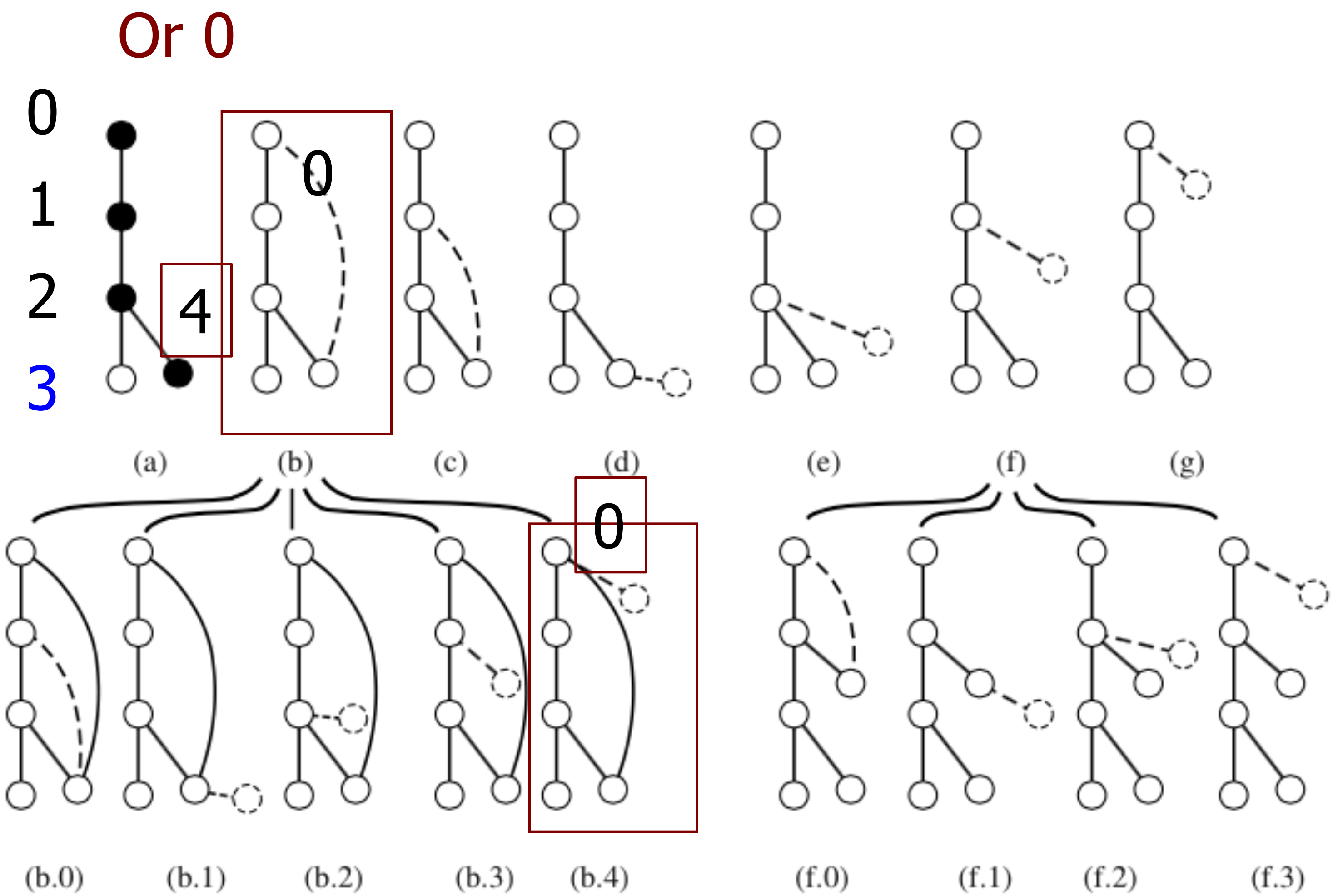
Rightmost Extension to Grow Patterns

Or 1 as RM Node



Rightmost Path: 0, 1
Rightmost Node: 1

Rightmost Extension to Grow Patterns

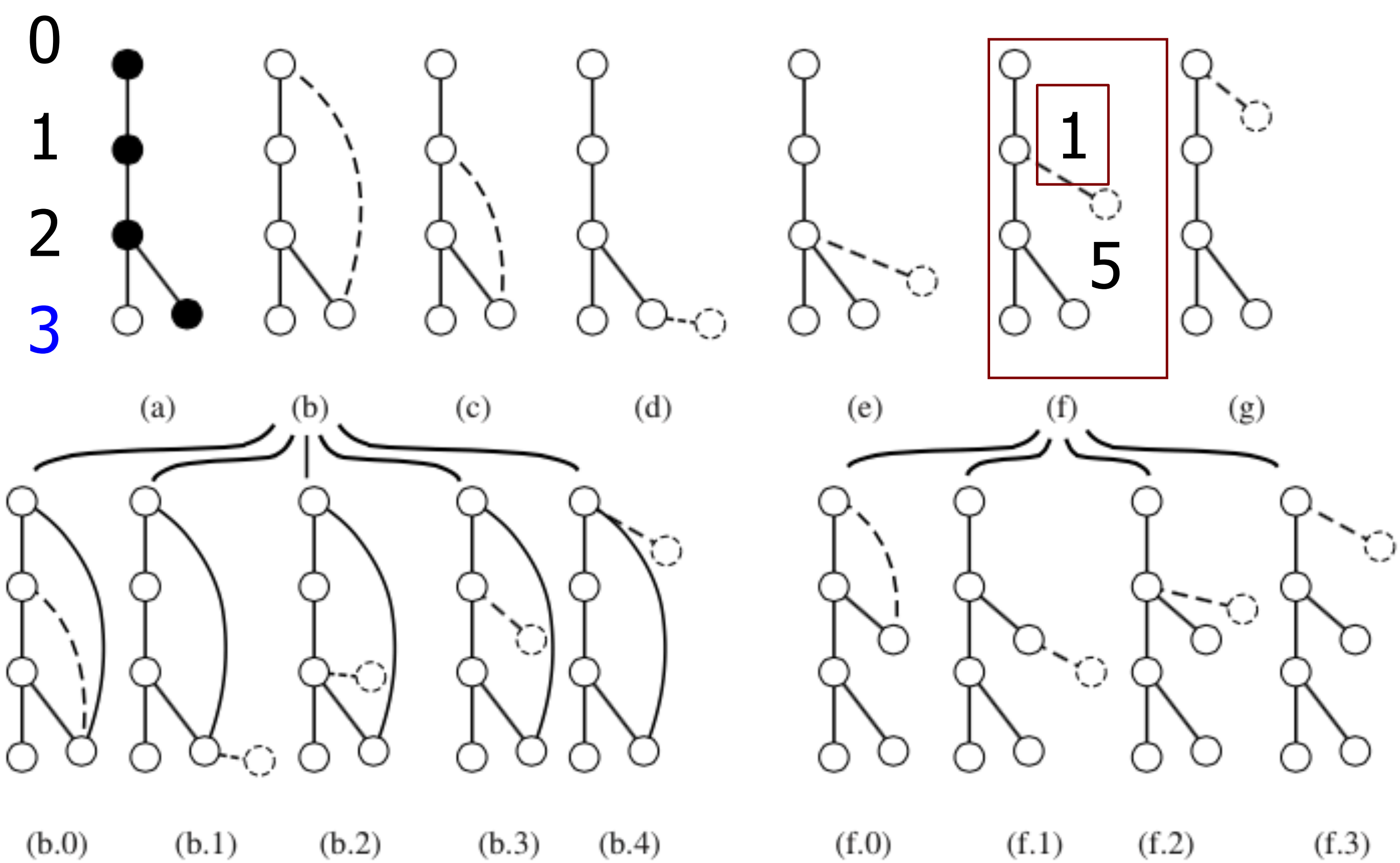


Rightmost Path: 0

Rightmost Node: 0

Rightmost Extension to Grow Patterns

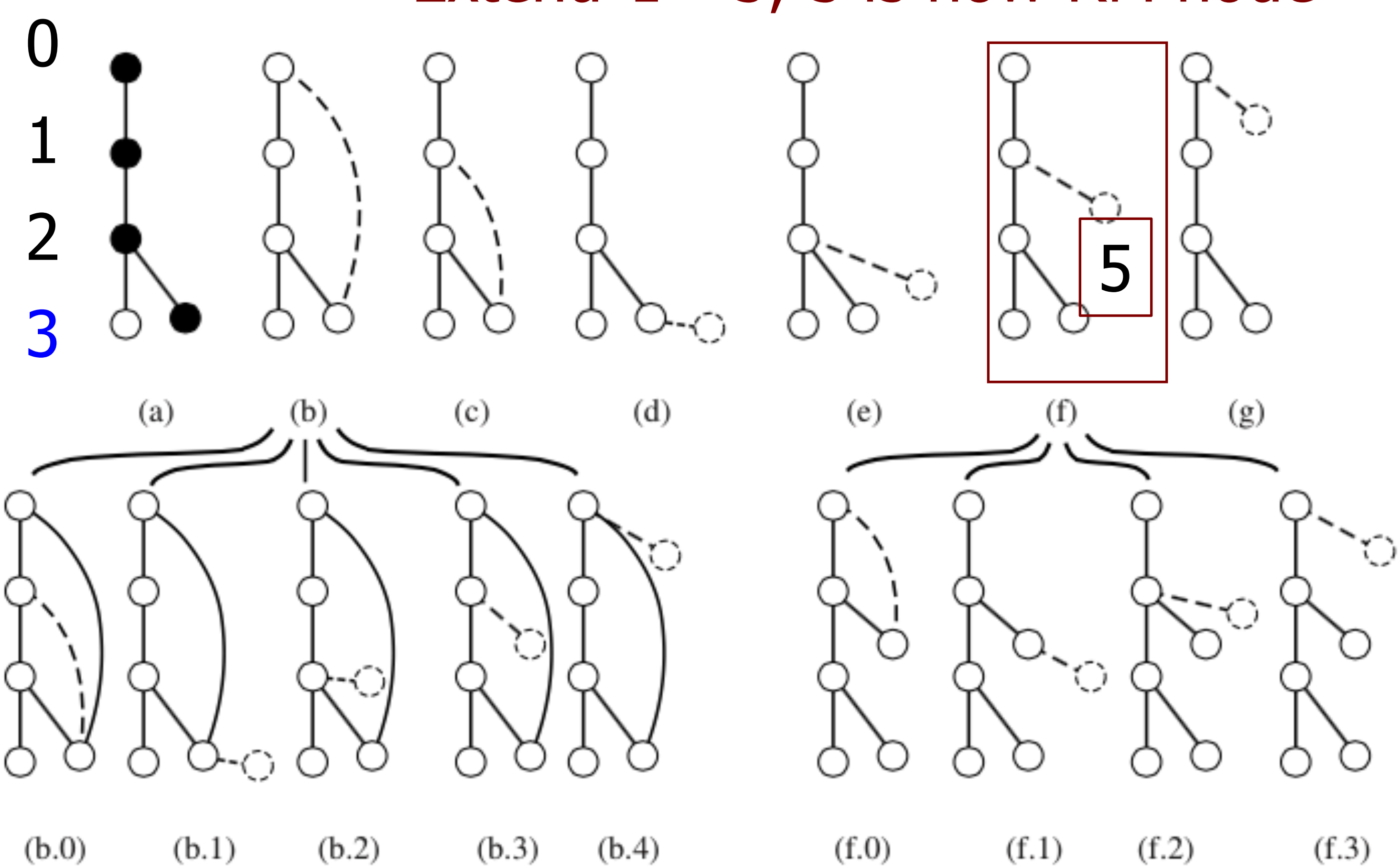
Another situation, RM node is 1, forward 1->5 exists



Rightmost Path: 0, 1
Rightmost Node: 1

Rightmost Extension to Grow Patterns

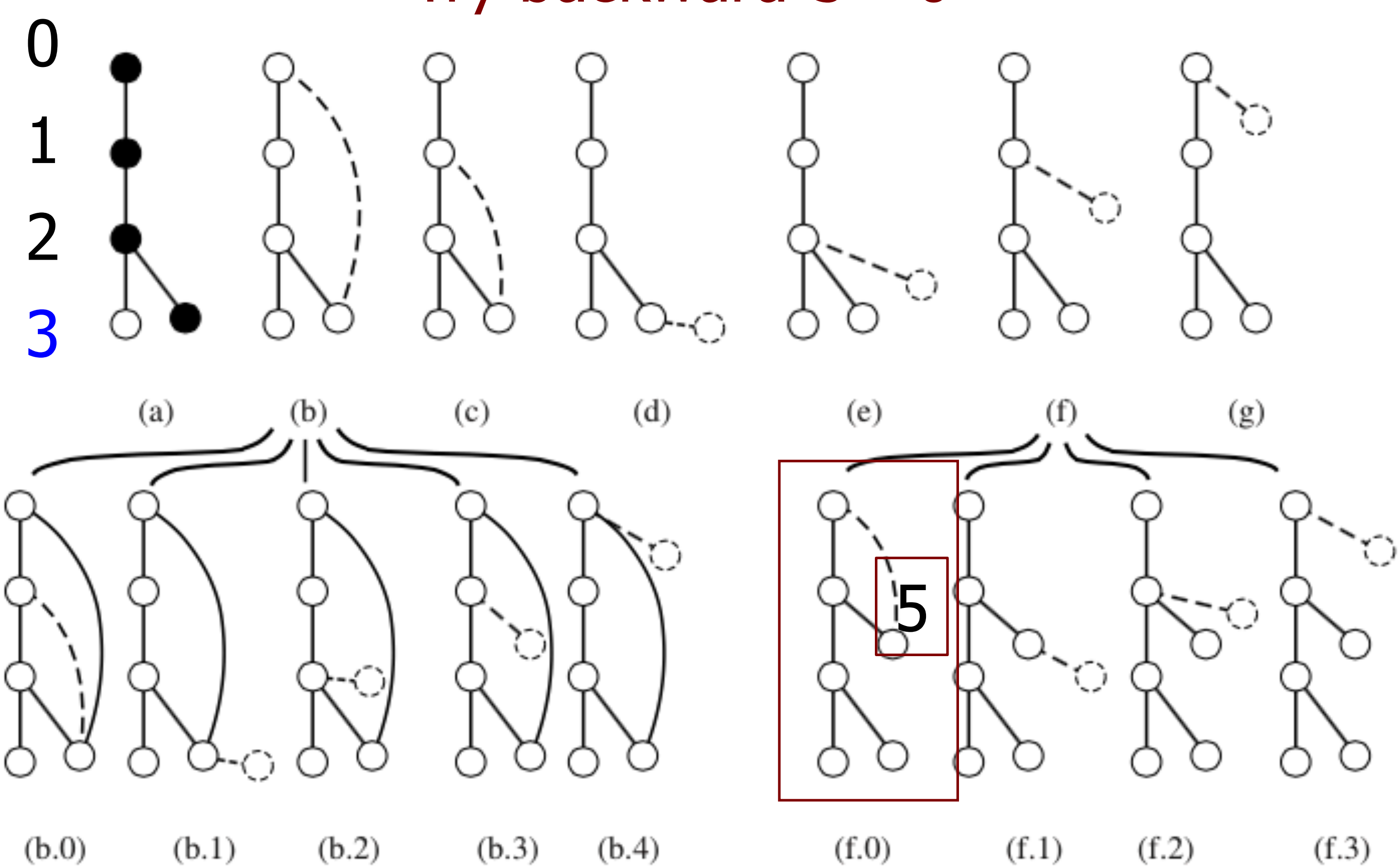
Extend 1->5, 5 is now RM node



Rightmost Path: 0, 1, 5
Rightmost Node: 5

Rightmost Extension to Grow Patterns

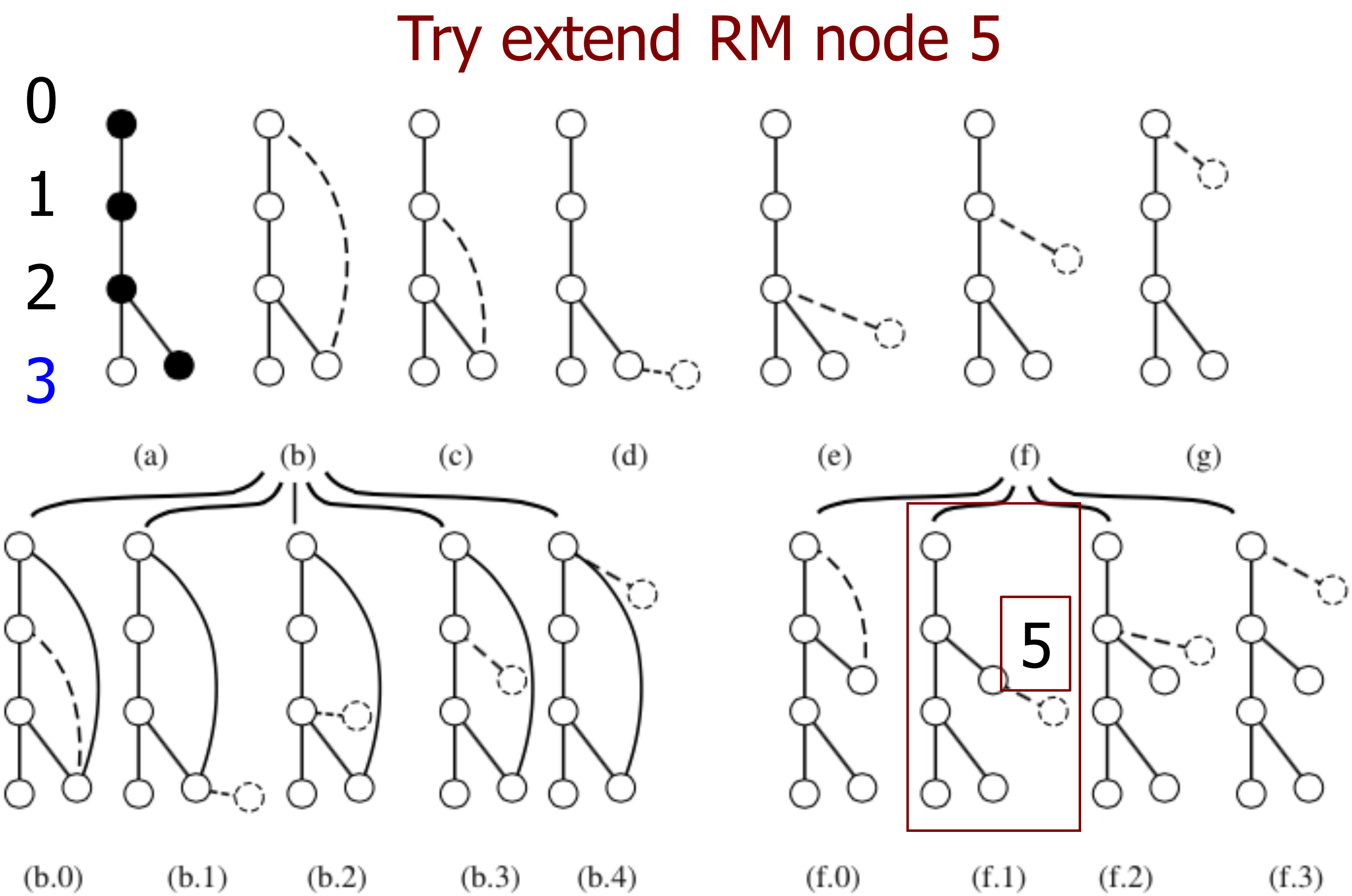
Try backward 5->0



Rightmost Path: 0, 1, 5

Rightmost Node: 5

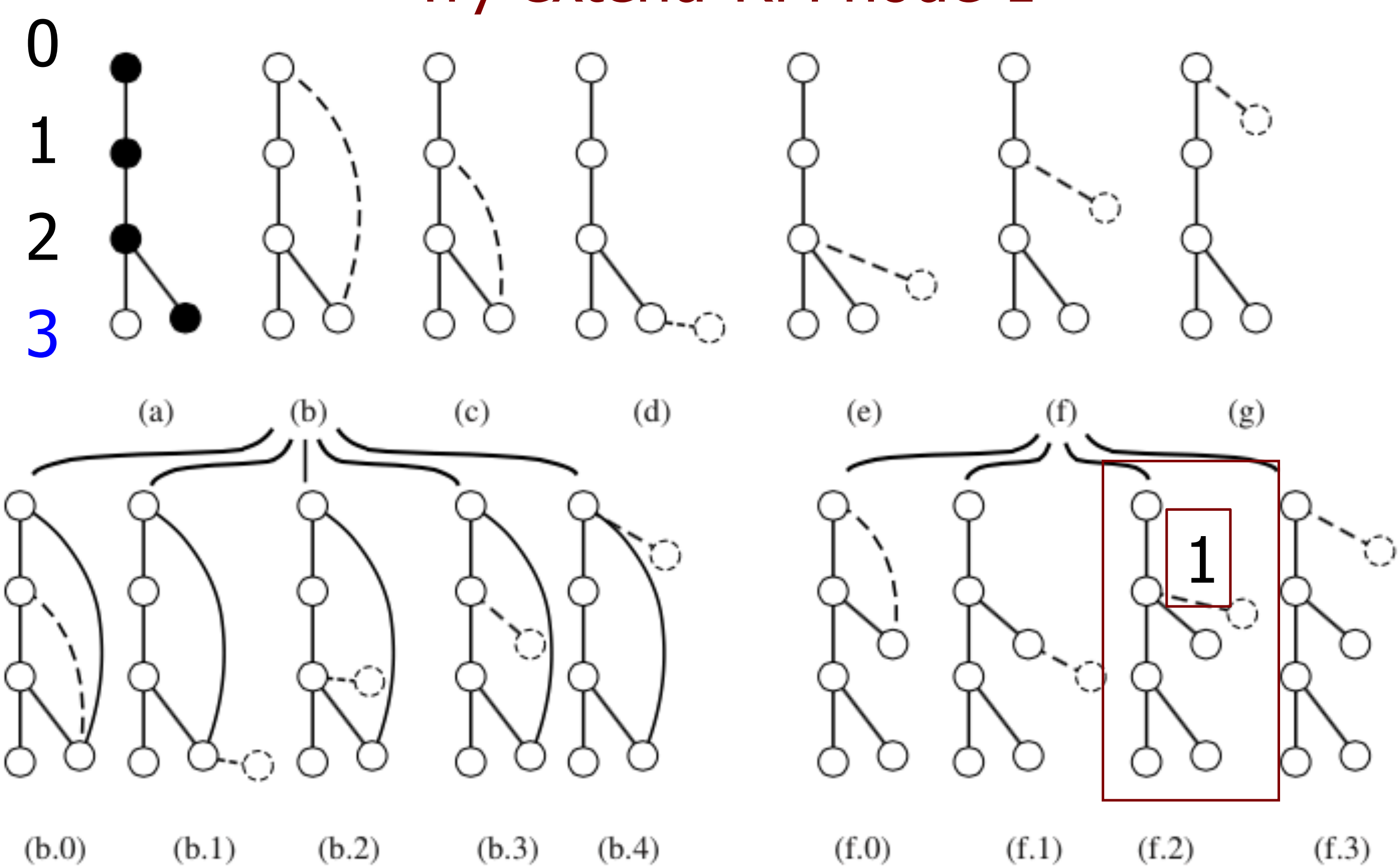
Rightmost Extension to Grow Patterns



Rightmost Path: 0, 1, 5
Rightmost Node: 5

Rightmost Extension to Grow Patterns

Try extend RM node 1

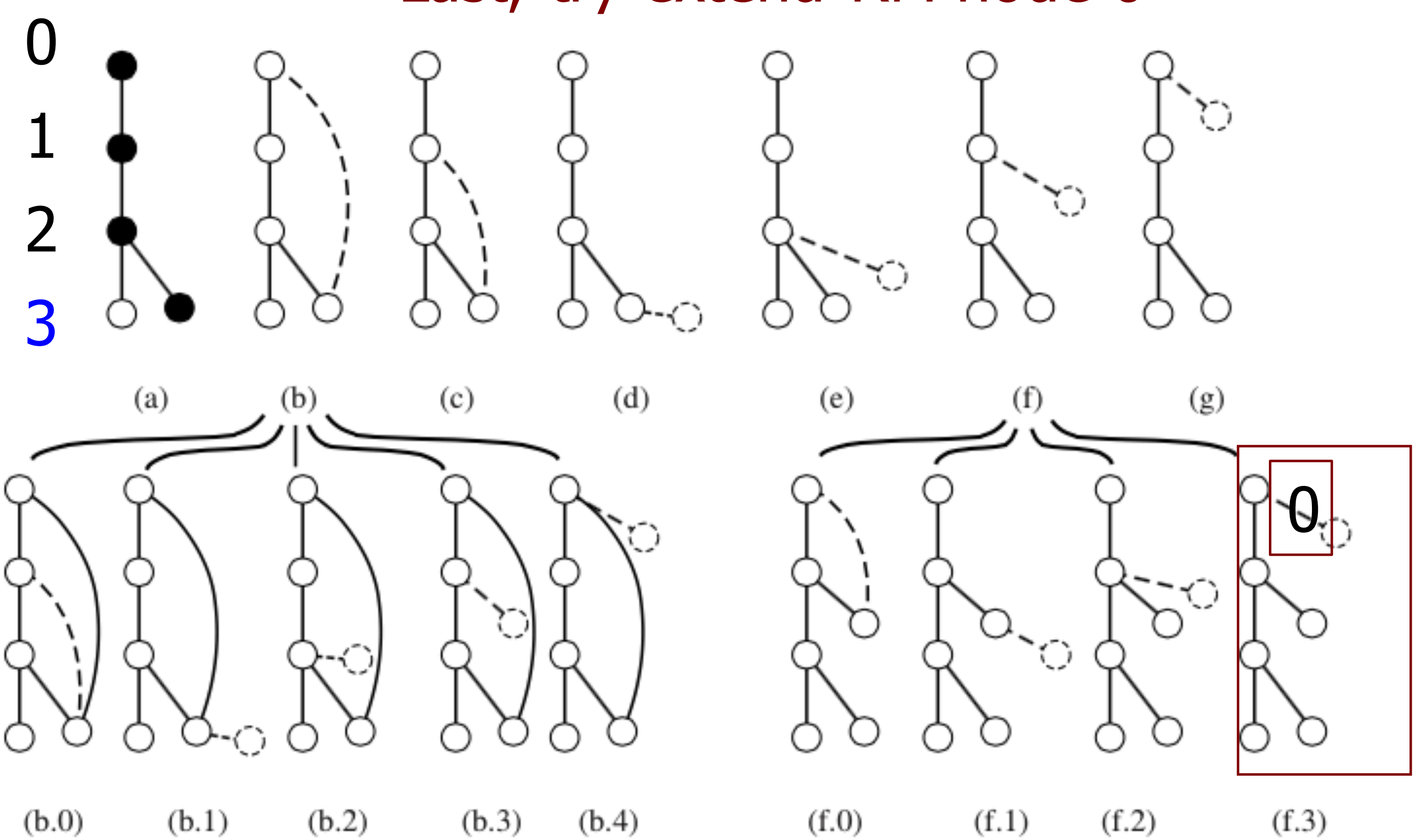


Rightmost Path: 0, 1

Rightmost Node: 1

Rightmost Extension to Grow Patterns

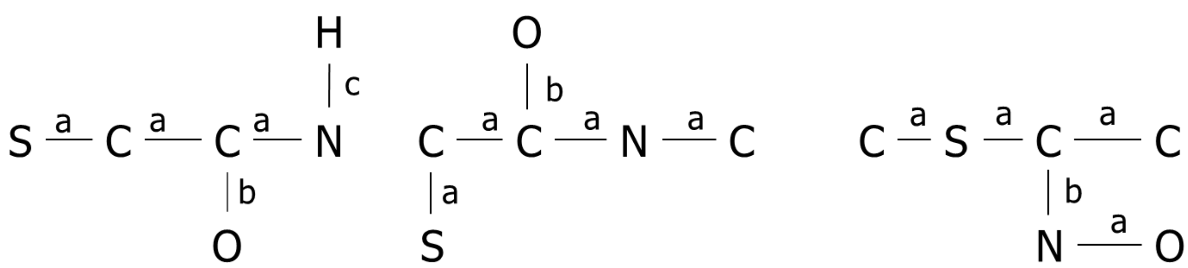
Last, try extend RM node 0



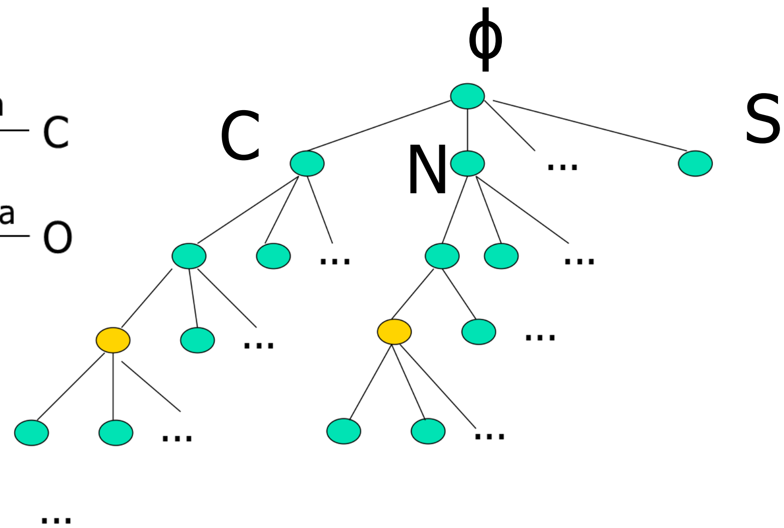
Rightmost Path: 0

Rightmost Node: 0

How to grow patterns to find out?



g1 g2 g3

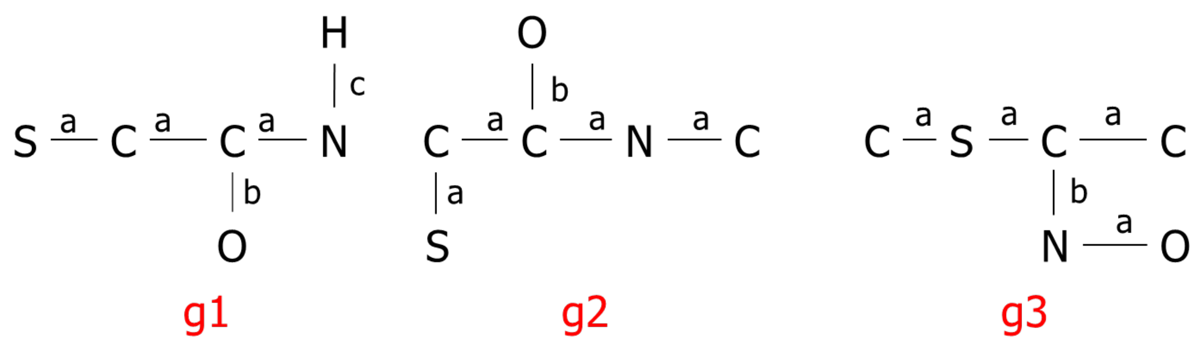


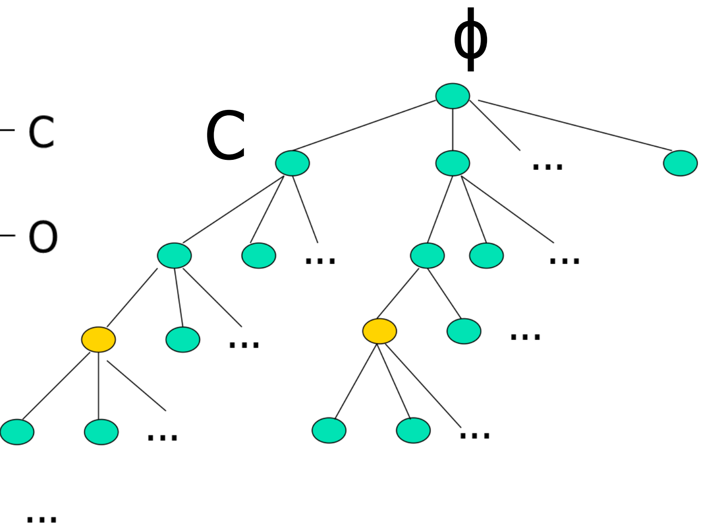
Frequency:

C: 3, N: 3, O: 3, S: 3, H: 1

Grow from C, N, O, S

How to grow patterns to find out?





Frequency:

C: 3, N: 3, O: 3, S: 3, H: 1

Grow from C (extension)

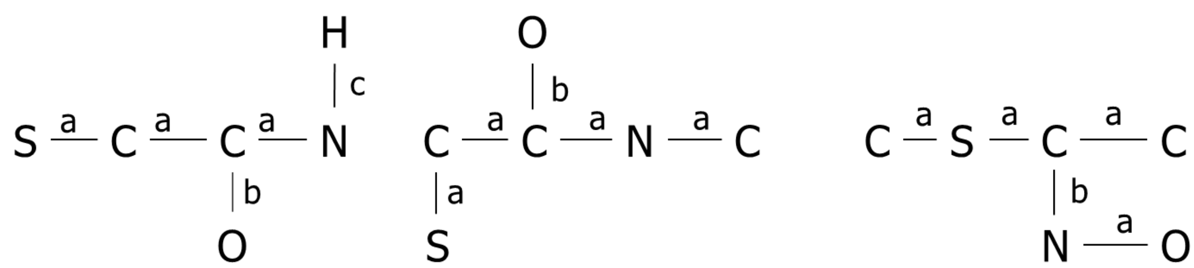
$C \overset{a}{\text{———}} C$ sup: 3

$C \overset{a}{\text{———}} N$ sup: 2

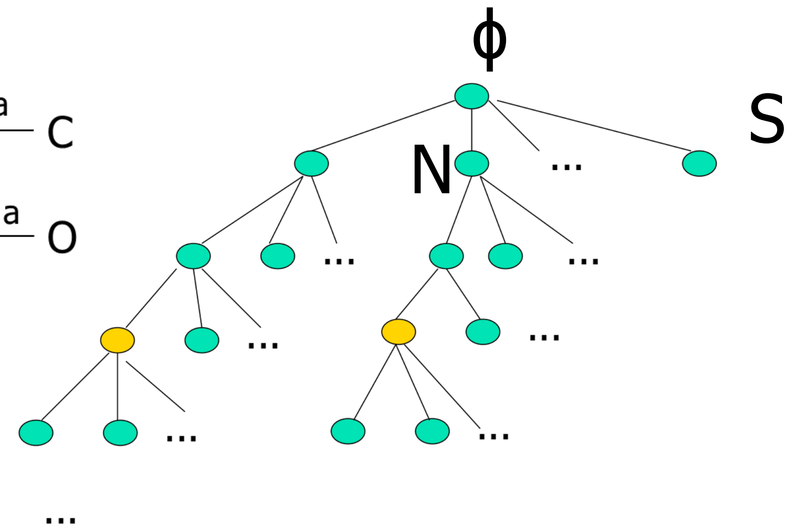
$C \overset{b}{\text{———}} O$ sup: 2

$C \overset{a}{\text{———}} S$ sup: 3

How to grow patterns to find out?



g1 g2 g3

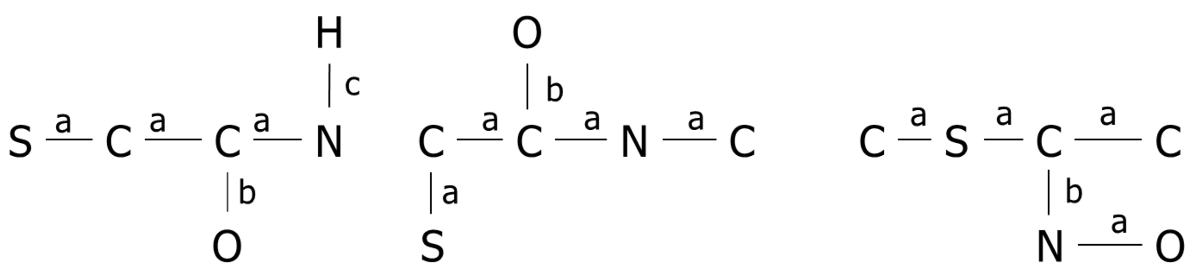


Frequency:

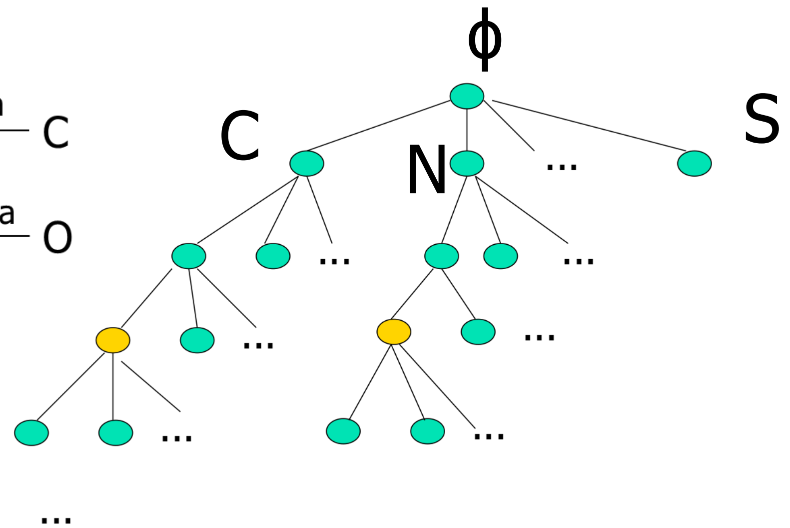
C: 3, N: 3, O: 3, S: 3, H: 1

Grow from N, S similarly (extension)

How to grow patterns to find out?



g1 g2 g3

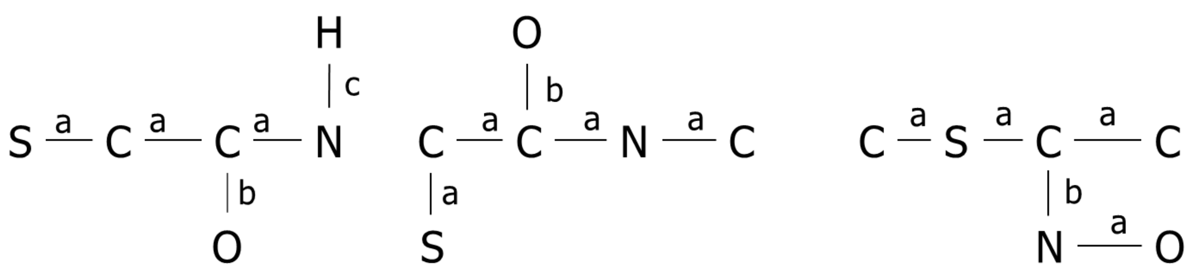


Frequency:

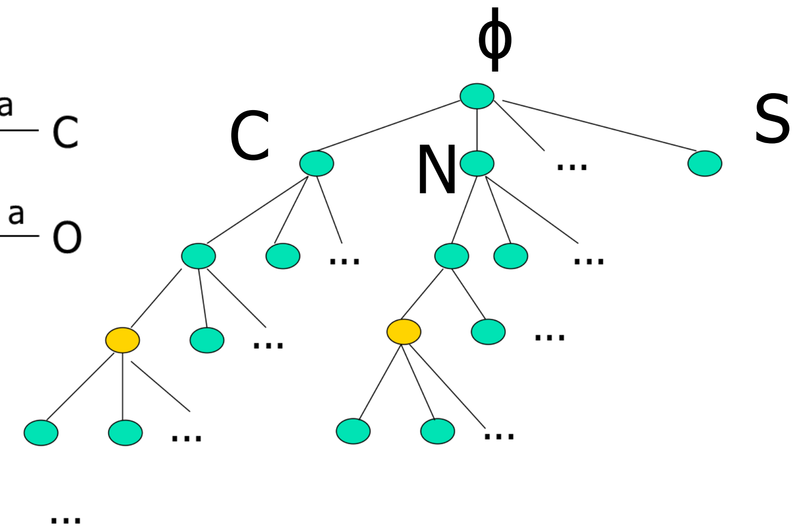
C: 3, N: 3, O: 3, S: 3, H: 1

Grow from $C \overset{a}{\text{———}} C$ (extension)

How to grow patterns to find out?



g1 g2 g3

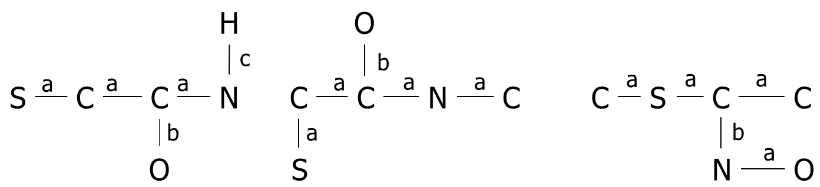


Frequency:

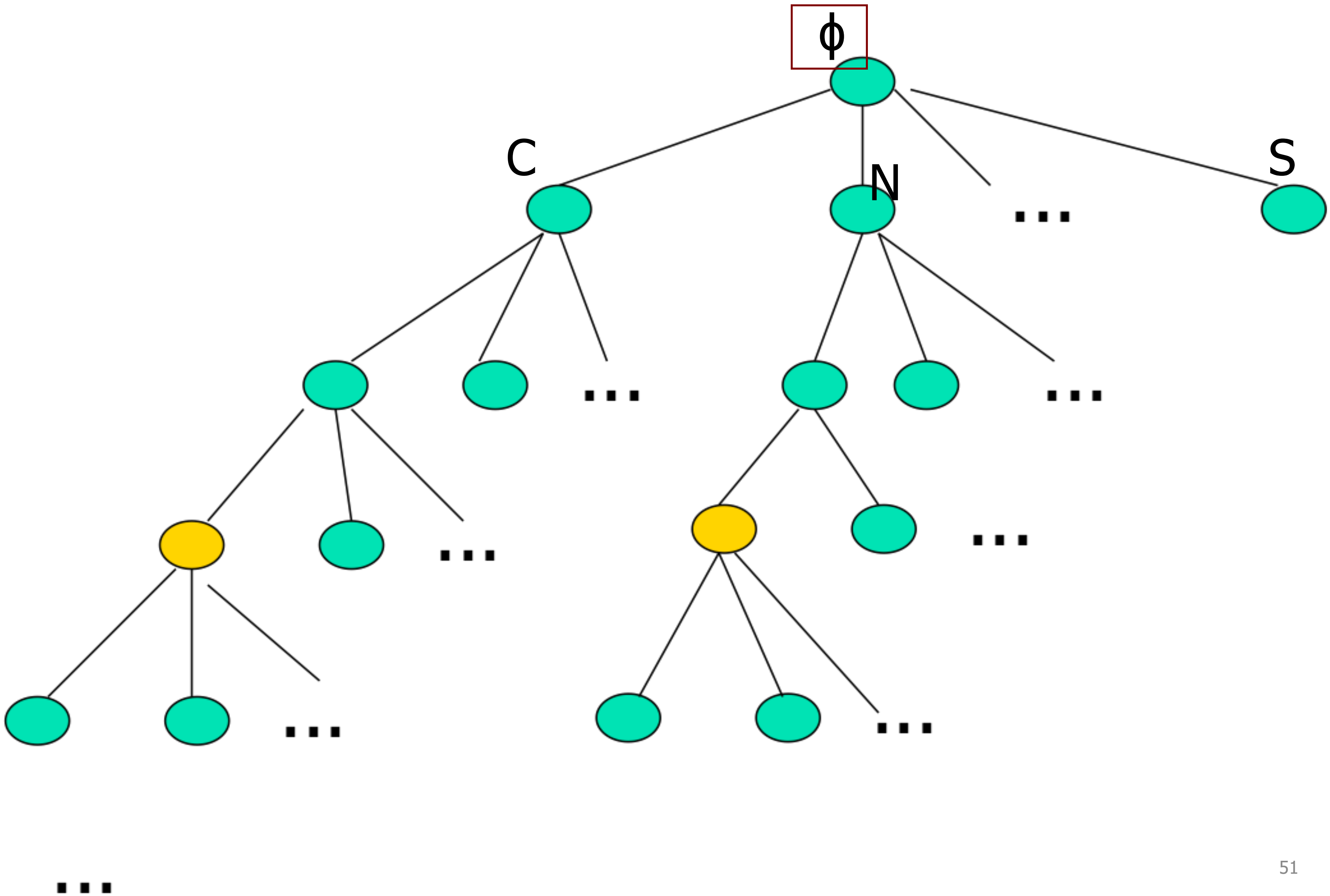
C: 3, N: 3, O: 3, S: 3, H: 1

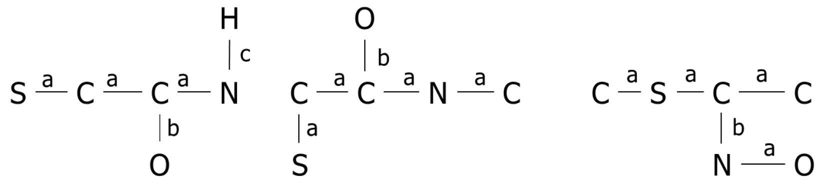
Grow from $C \overset{a}{\text{—}} N$ (extension)

Once $C \overset{a}{\text{—}} C \overset{a}{\text{—}} N$ sup: 2 is encountered, by minimum DFS code, it is identified redundant pattern, and thus removed.

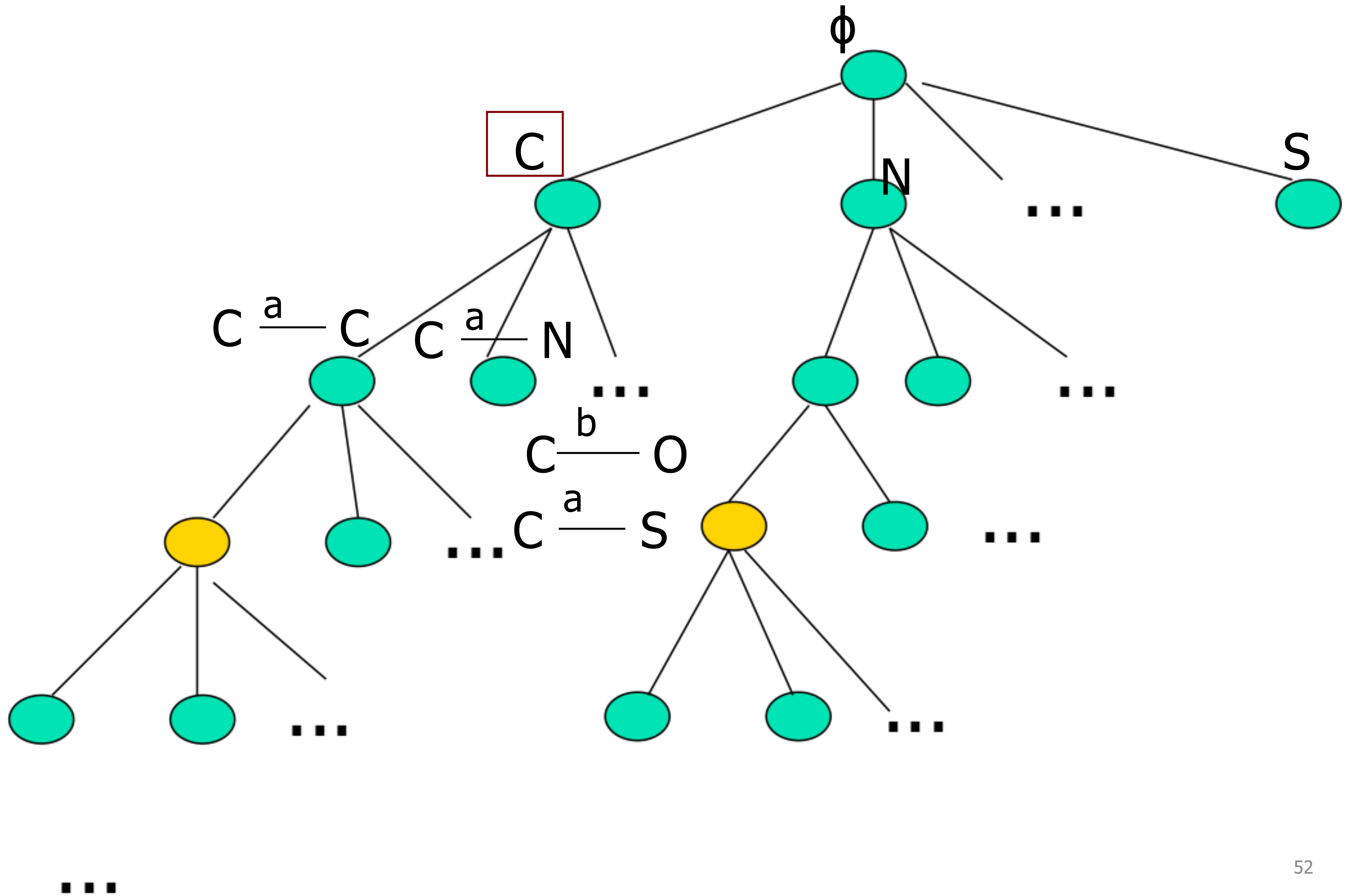


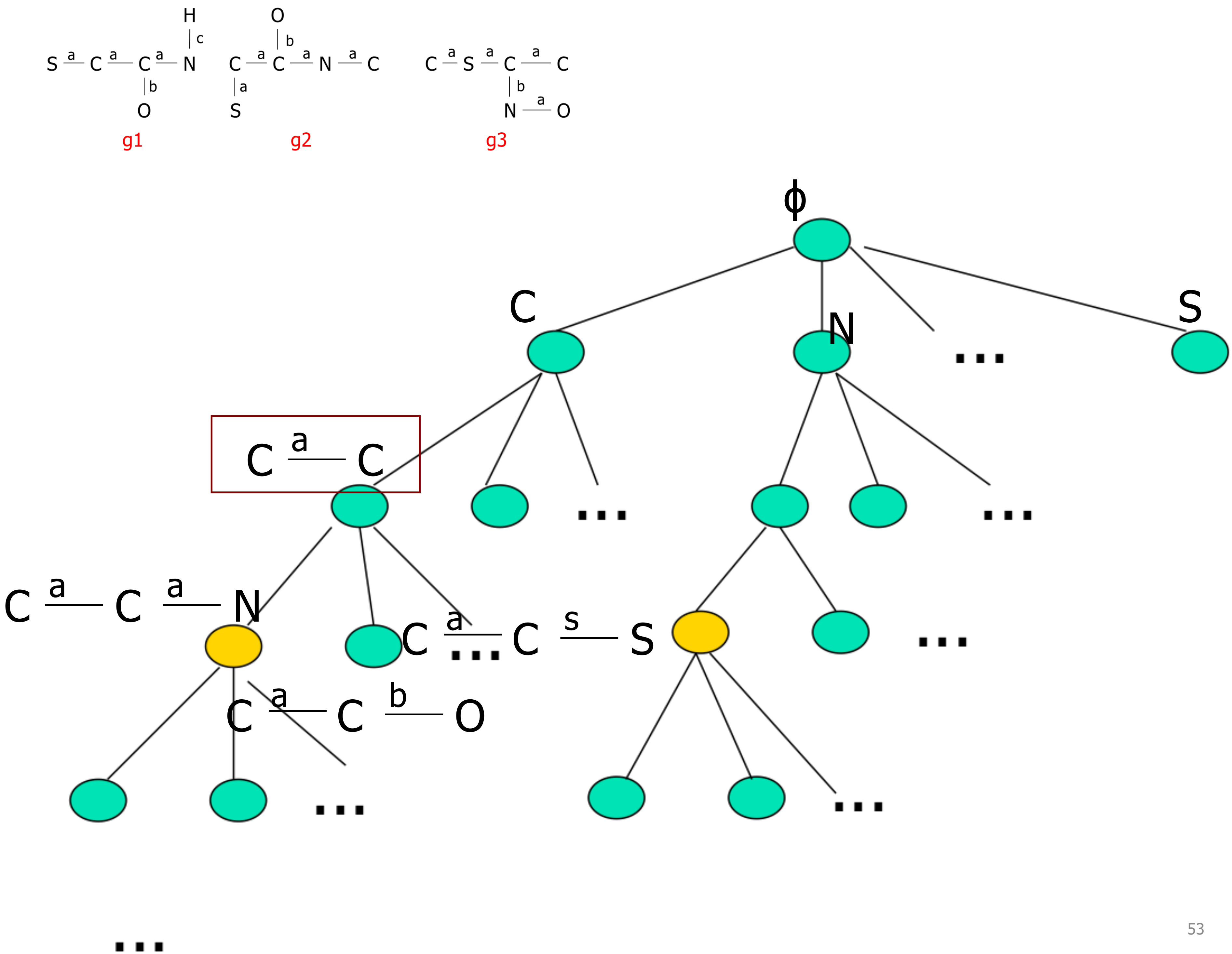
g1 g2 g3

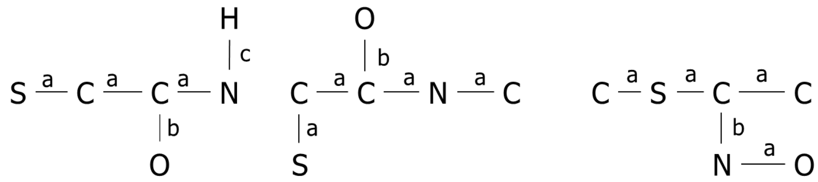




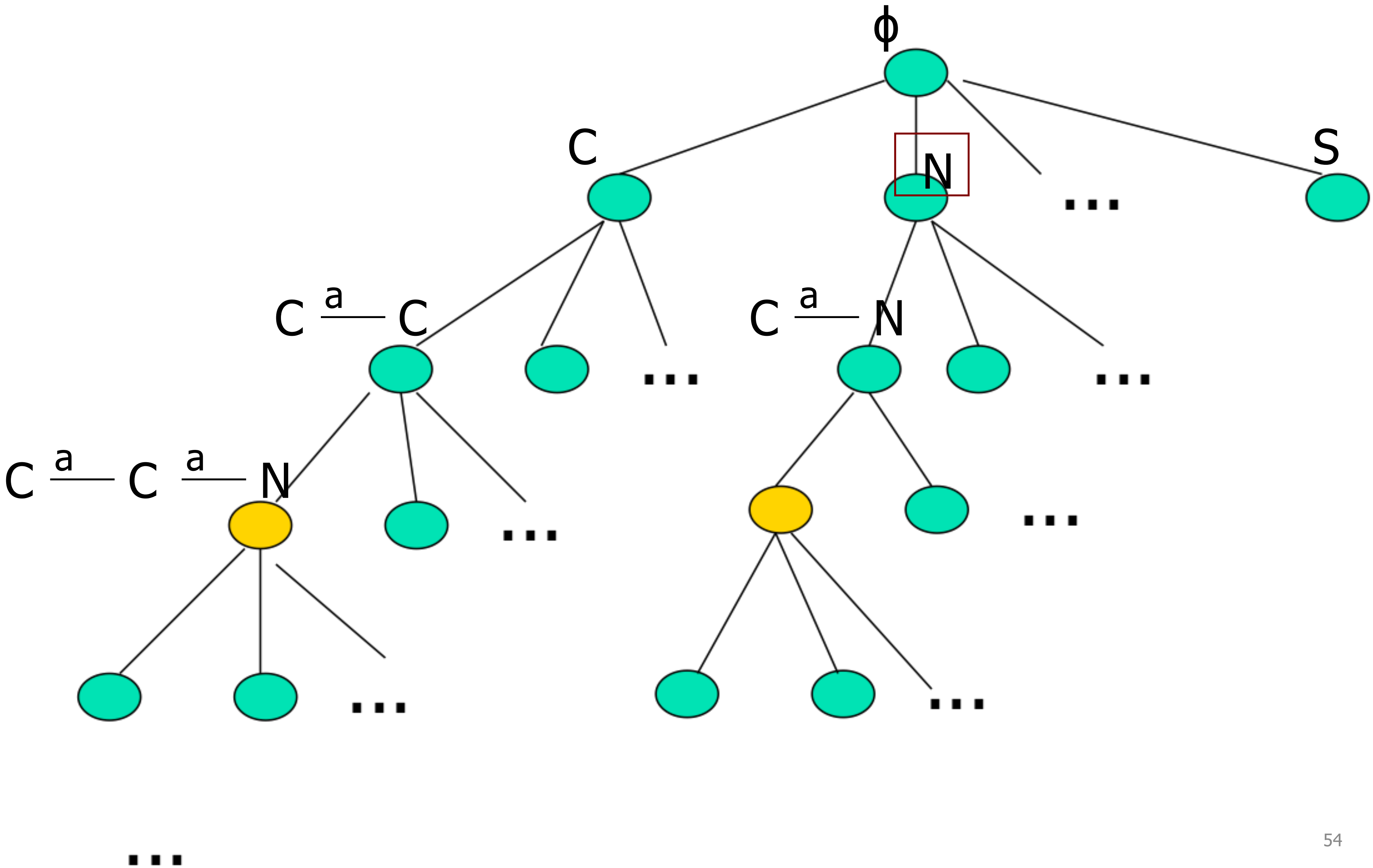
g1 g2 g3

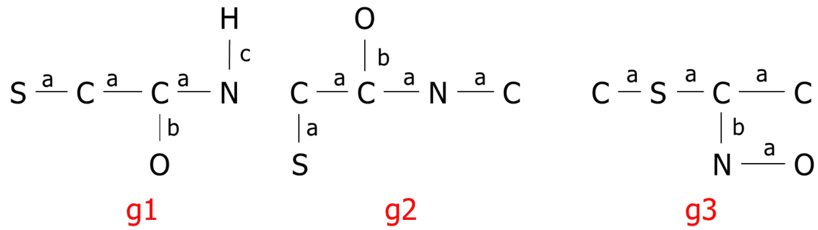


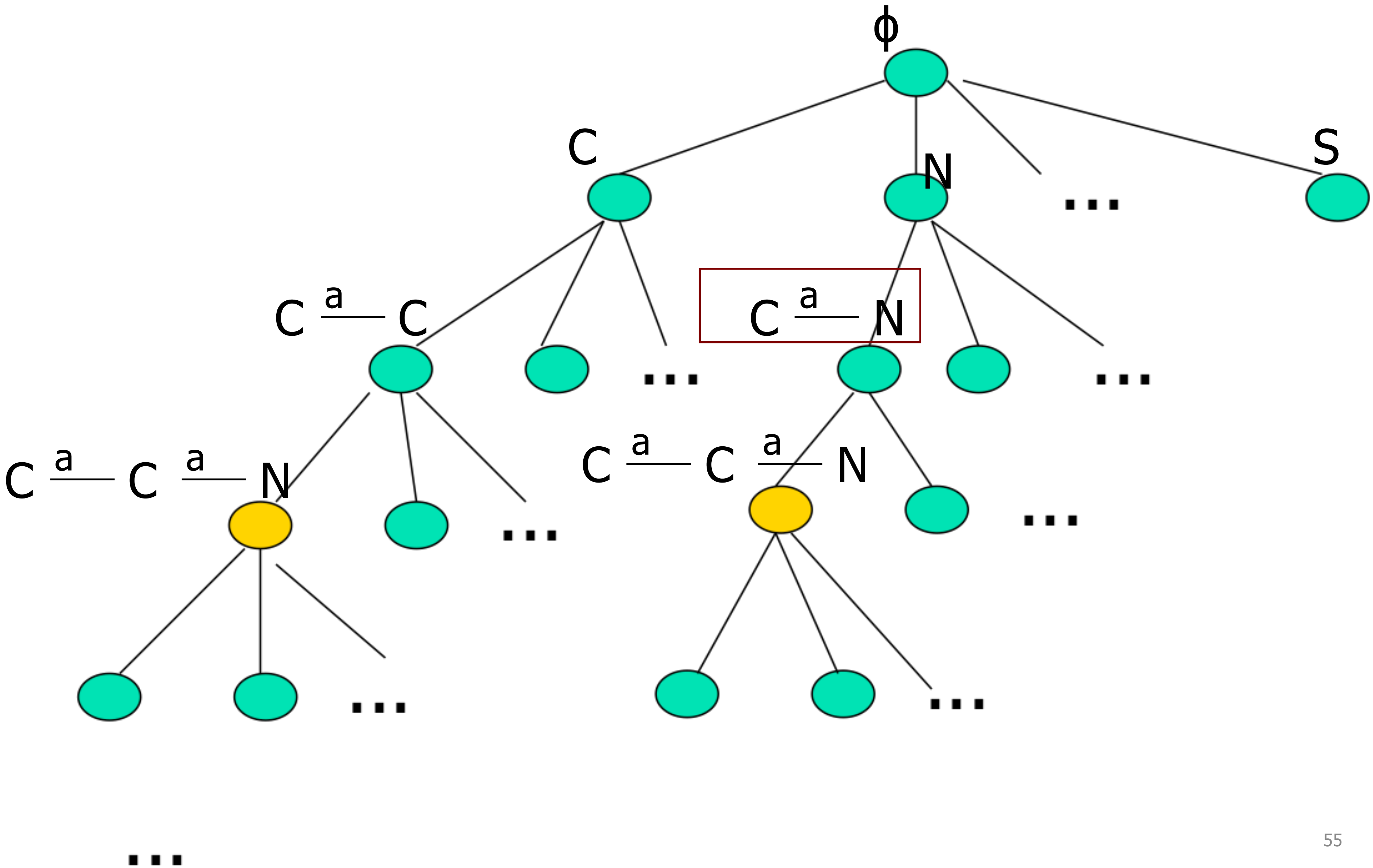

g1
g2
g3
ϕ
C
N
S
...
C a C
C a C a N
C a C s S
C a C b O
...

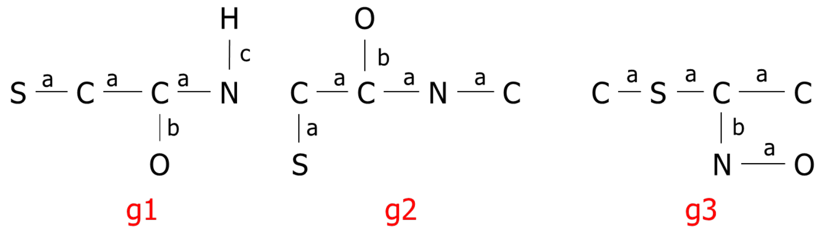


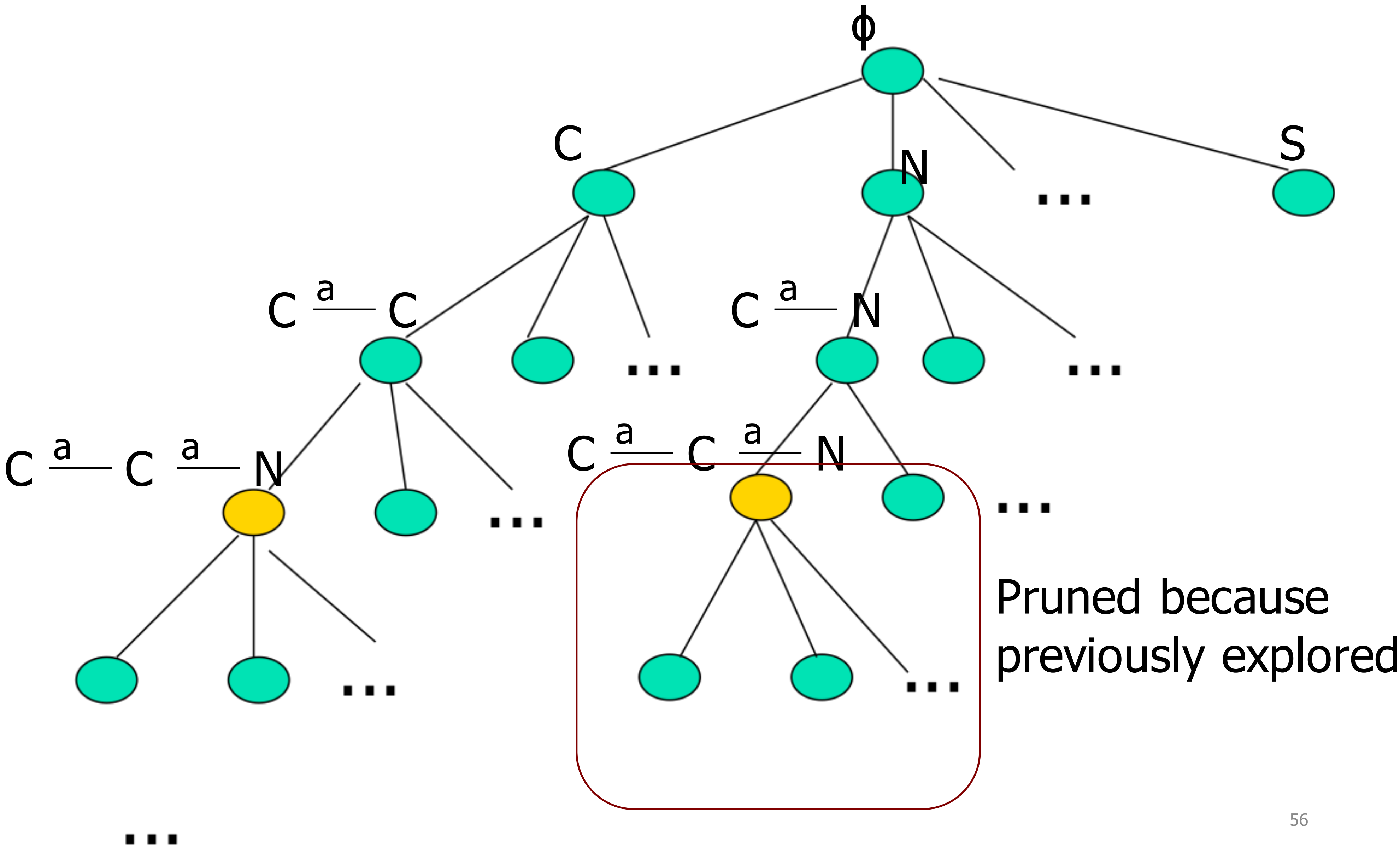
g1 g2 g3



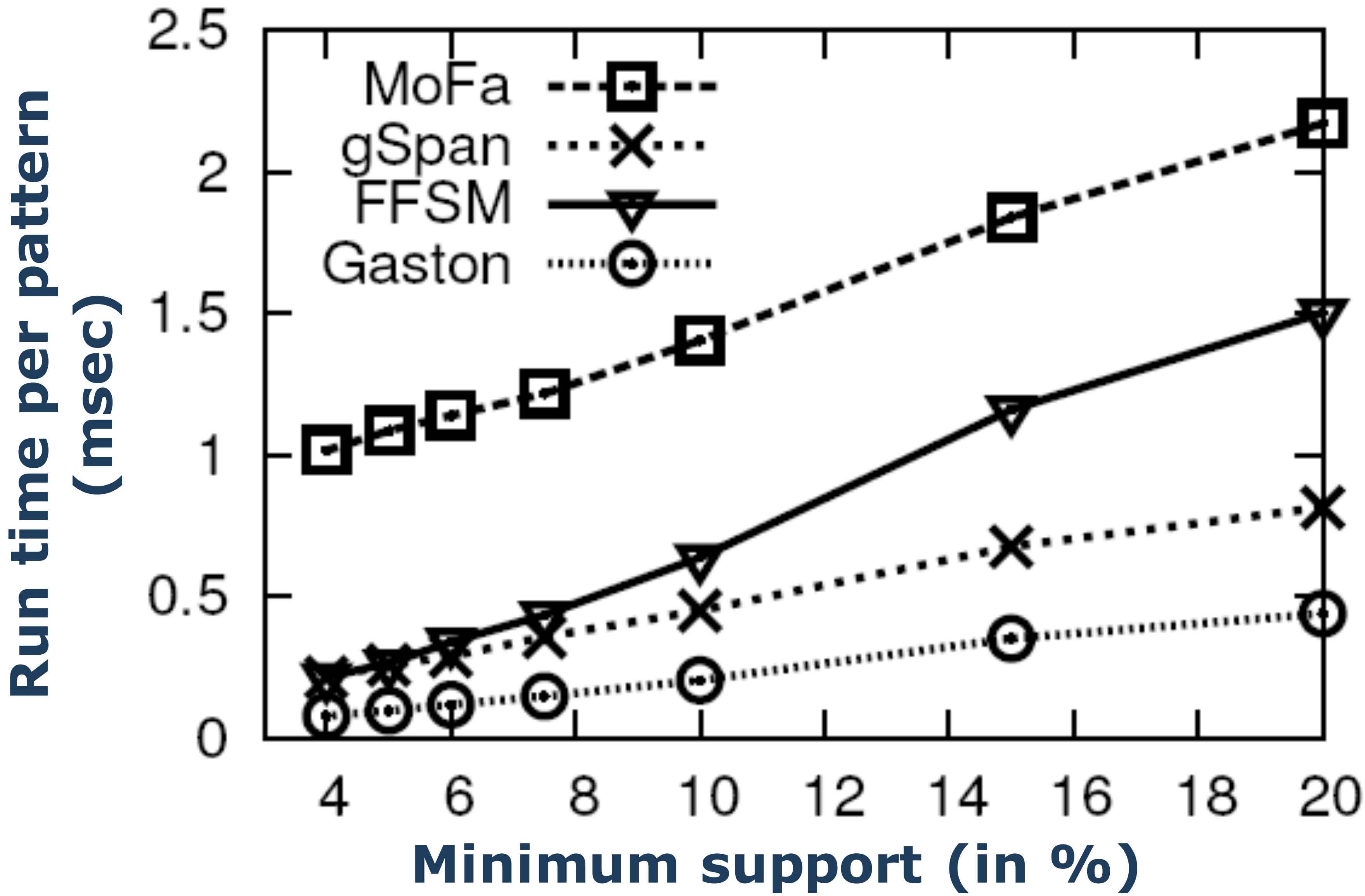

g1
g2
g3


ϕ
C
N
S
C a C
C a N
C a C a N
C a C a N

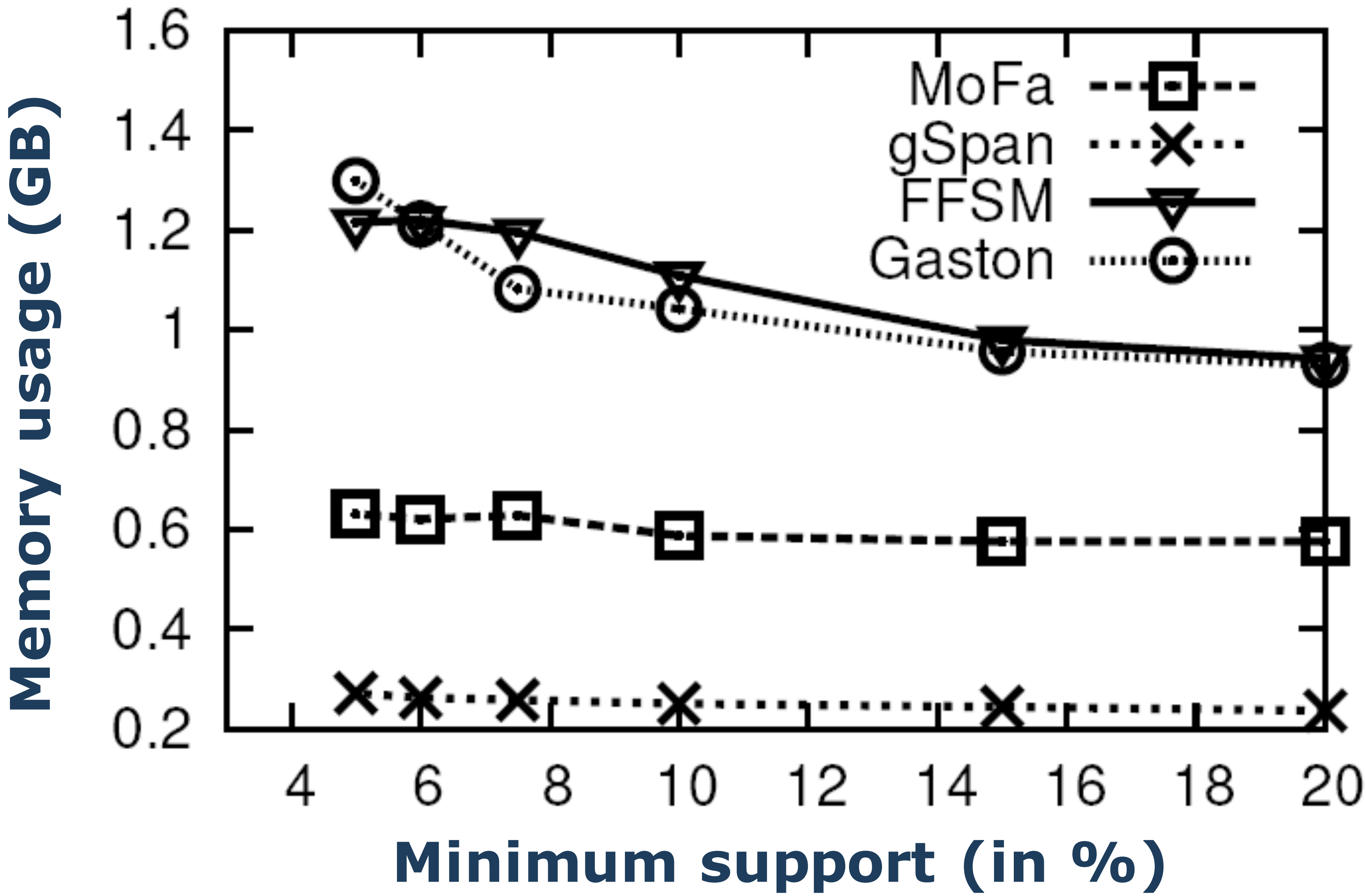

g1
g2
g3


ϕ
C
N
S
C a C
C a N
C a C a N
C a C a N
Pruned because previously explored

Performance (1): Run Time



Performance (2): Memory Usage



Chapter 7 : Advanced Frequent Pattern Mining

- Pattern Mining: A Road Map
- ~~Pattern Mining in Multi-Level, Multi-Dimensional Space~~
- Constraint-Based Frequent Pattern Mining
- ~~Mining High-Dimensional Data and Colossal Patterns~~
- ~~Mining Compressed or Approximate Patterns~~
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Summary

- Roadmap: Many aspects & extensions on pattern mining
- Mining patterns in multi-level, multi dimensional space
- Mining rare and negative patterns
- Constraint-based pattern mining
- Specialized methods for mining high-dimensional data and colossal patterns
- Mining compressed or approximate patterns
- Pattern exploration and understanding: Semantic annotation of frequent patterns