Data Mining
Frequent Pattern Mining (2)
Sequences and Trees

Ad Feelders

Universiteit Utrecht

Anti-Monotonicity

$X \subseteq Y \Rightarrow S(X) \geq S(Y)$

- Frequent Pattern Mining

1. Item Set Mining ✓
2. Sequence Mining ✓
3. Tree Mining ✓
4. Graph Mining ✗

Frequent Pattern Mining: the bigger picture

1. Item Set Mining: the patterns are *sets* of items, and an item set occurs in a transaction if it is a *subset* of the transaction.
2. Sequence Mining: the patterns are *sequences* of events, and an event sequence occurs in a data sequence if it is a *subsequence* of the data sequence.
3. Tree Mining: the patterns are *trees*, and a pattern tree occurs in a data tree if it is a *subtree* of the data tree.

Anti-monotonicity property:

$$P_1 \subseteq P_2 \Rightarrow s(P_1) \geq s(P_2),$$

where P_1 and P_2 are patterns, $\subseteq$ denotes a generic subpattern relation, and $s(\cdot)$ denotes support.

Sequence Mining

1. Alphabet Σ (set of labels).
2. Sequence $\mathbf{s} = s_1 s_2 \ldots s_n$ where $s_i \in \Sigma$.
3. Prefix: $\mathbf{s}[1 : i] = s_1 s_2 \ldots s_i$, $0 \leq i \leq n$ (initial segment).
4. Suffix: $\mathbf{s}[i : n] = s_i s_{i+1} \ldots s_n$, $1 \leq i \leq n + 1$ (final segment).

Subsequence

Let $\mathbf{s} = s_1 s_2 \ldots s_n$ and $\mathbf{r} = r_1 r_2 \ldots r_m$ be two sequences over Σ. We say $\mathbf{r}$ is a subsequence of $\mathbf{s}$, denoted $\mathbf{r} \subseteq \mathbf{s}$, if there exists a one-to-one mapping

$$\phi : [1, m] \rightarrow [1, n],$$

such that

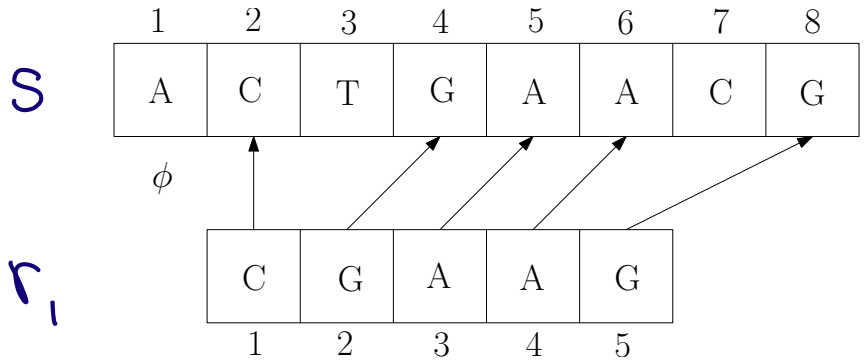
1. $\mathbf{r}[i] = \mathbf{s}[\phi(i)]$, and
2. $i < j \Rightarrow \phi(i) < \phi(j)$.

Each position in $\mathbf{r}$ is mapped to a position in $\mathbf{s}$ with the same label, and the order of positions in the sequence is preserved. There may however be gaps between $\phi(i)$ and $\phi(i+1)$ $(i = 1, \ldots, m-1)$.

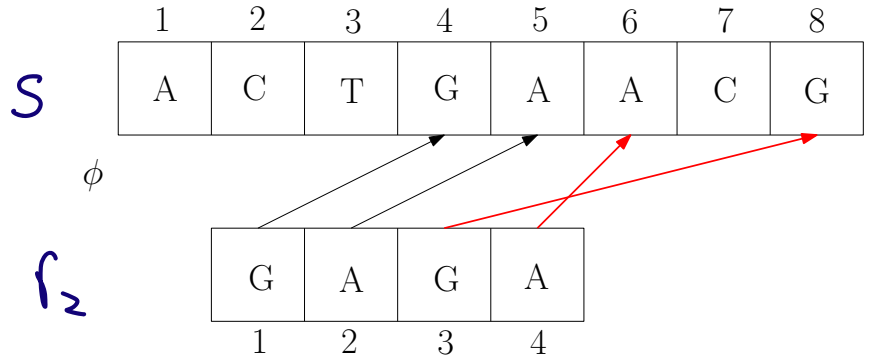
Subsequence: Example

Let $\Sigma = \{A, C, G, T\}$ and let $\mathbf{s} = ACTGAACG$.

1. $\mathbf{r}_1 = CGAAG$ is a subsequence of $\mathbf{s}$. The corresponding mapping is $\phi(1) = 2$, $\phi(2) = 4$, $\phi(3) = 5$, $\phi(4) = 6$, and $\phi(5) = 8$.



2. $\mathbf{r}_2 = GAGA$ is not a subsequence of $\mathbf{s}$.



Frequent Sequence Mining Task

Given a database $\mathbf{D} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$ of N sequences, and given some sequence $\mathbf{r}$, the *support* of $\mathbf{r}$ in the database $\mathbf{D}$ is defined as the total number of sequences in $\mathbf{D}$ that contain $\mathbf{r}$:

$$\sup(\mathbf{r}) = |\{\mathbf{s}_i \in \mathbf{D} : \mathbf{r} \subseteq \mathbf{s}_i\}|$$

Given a minimum support threshold minsup, compute

$$\mathcal{F}(\text{minsup}, \mathbf{D}) = \{\mathbf{r} \mid \sup(\mathbf{r}) \geq \text{minsup}\}$$

Anti-Monotonicity Property

For a database of sequences $\mathbf{D}$, and two sequences $\mathbf{r}_1$ and $\mathbf{r}_2$, we have

$$\mathbf{r}_1 \subseteq \mathbf{r}_2 \Rightarrow \sup(\mathbf{r}_1) \geq \sup(\mathbf{r}_2),$$

because $\forall \mathbf{s} \in \mathbf{D} : \mathbf{r}_2 \subseteq \mathbf{s} \Rightarrow \mathbf{r}_1 \subseteq \mathbf{s}$.

Hence, in a level-wise search for frequent sequences, there is no point in expanding infrequent ones.

if δ_1: A A B ← subsequence S

than δ_2: AB is also subsequence of S

GSP Algorithm

How to find next level sequences.

ex: r_1: AAB , r_2: AAA

4 pre-candidates r_{12}: AAAB r_{11}: AAAA

r_{21}: AABA r_{22}: AABB

1. Perform level-wise search.
2. Don't extend infrequent sequences.
3. Candidate generation for level $k+1$: take two frequent sequences $\mathbf{r}_a$ and $\mathbf{r}_b$ of length k with $\mathbf{r}_a[1:k-1] = \mathbf{r}_b[1:k-1]$ and generate pre-candidate $\mathbf{r}_{ab} = \mathbf{r}_a + \mathbf{r}_b[k]$. Pre-candidate $\mathbf{r}_{ab}$ becomes a candidate (has to be counted) if all its subsequences of length k are frequent. Note that we allow $\mathbf{r}_a = \mathbf{r}_b$.

Example Level-wise Search (minsup=3)

sid	Sequence
1	CAGAAGT
2	TGACAG
3	GAAGT

length 1

Candidate	Support	Frequent?
A	3	✓
C	2	✗
G	3	✓
T	3	✓

- C is not frequent, so it won't be used for candidate generation at the next level.

Example Level-wise Search (minsup=3)

match at first (k-1) position

sid	Sequence
1	CAGAAGT
2	TGACAG
3	GAAGT

Frequent	Support
A	3
G	3
T	3

Candidate	Support	Frequent?
AA	3	✓
AG	3	✓
AT	2	✗
GA	3	✓
GG	3	✓
GT	2	✗
TA	1	✗
TG	1	✗
TT	0	✗

Example Level-wise Search (minsup=3)

sid	Sequence
1	CAGAAGT
2	TGACAG
3	GAAGT

Frequent	Support
AA	3
AG	3
GA	3
GG	3

Candidate	Support	Frequent?
AAA	1	✗
AAG	3	✓
AGA	1	✗
AGG	1	✗
GAA	3	✓
GAG	3	✓
GGA	0	✗
GGG	0	✗

Example Level-wise Search (minsup=3)

sid	Sequence
1	CAGAAGT
2	TGACAG
3	GAAGT

combine with itself

Frequent	Support
AAG	3
GAA	3
GAG	3

Pre-candidate	Support	Frequent?
AAGG	-	infrequent subsequence AGG
GAAA	-	infrequent subsequence AAA
GAAG	3	✓
GAGA	-	infrequent subsequence GGA
GAGG	-	infrequent subsequence GGG

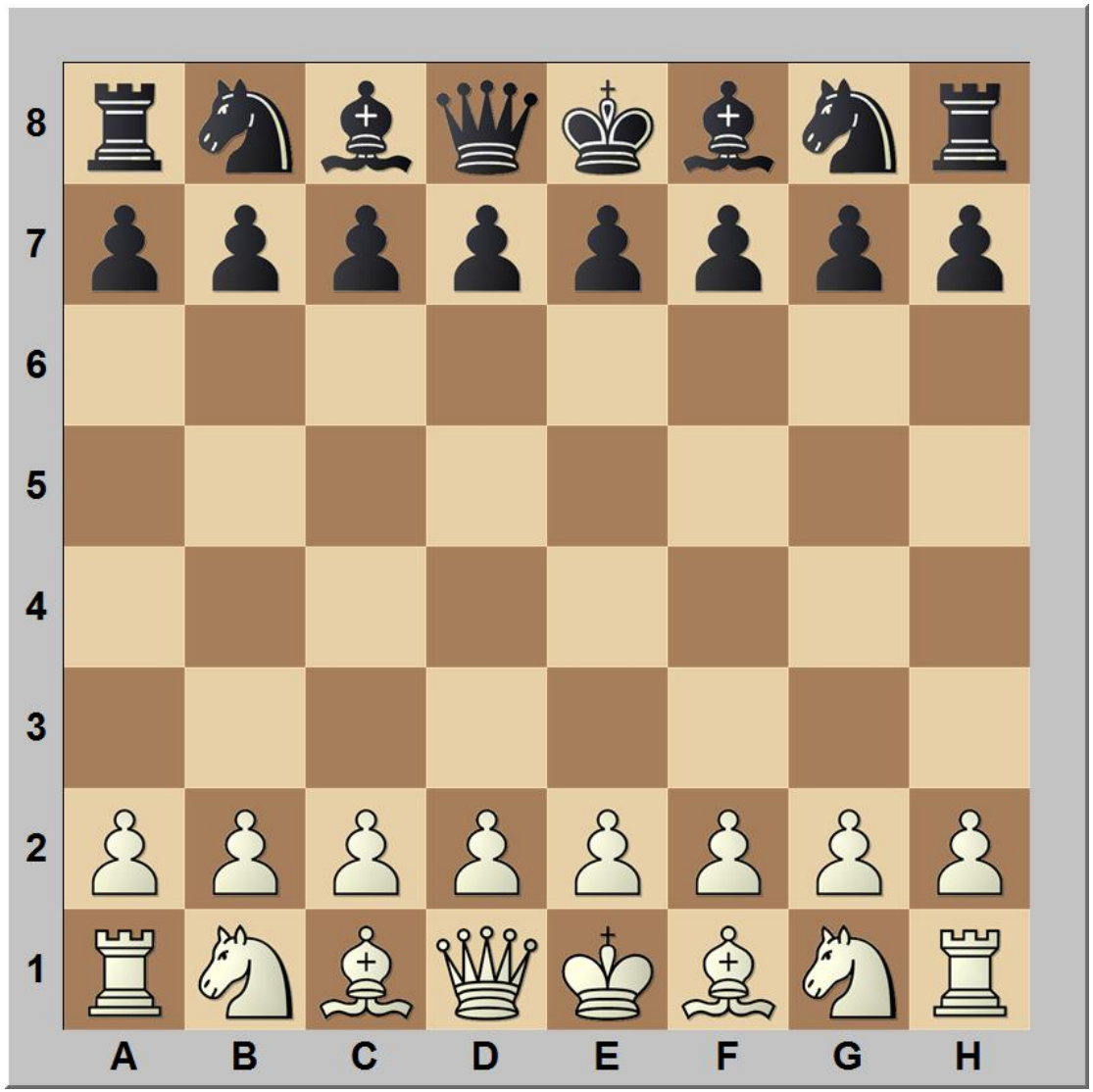
Level 5 pre-candidate GAAGG has infrequent subsequence GAGG.

Finding frequent movie sequences in Netflix data

Sequence of movie titles (frequency)
(1) *"Men in Black II", "Independence Day", "I, Robot"* (2,268)
(2) *"Pulp Fiction","Fight Club"* (7,406)
(3) *"Lord of the Rings: The Fellowship of the Ring", "Lord of the Rings: The Two Towers"* (19,303)
(4) *"The Patriot", "Men of Honor"* (28,710)
(5) *"Con Air", "The Rock"* (29,749)
(6) *"Pretty Woman", "Miss Congeniality"* (30,036)

From: KAUSTUBH BEEDKAR et al.,
Closing the Gap: Sequence Mining at Scale,
ACM Transactions on Database Systems, Vol. 40, No. 2, June 2015.

Finding frequent move sequences in chess games

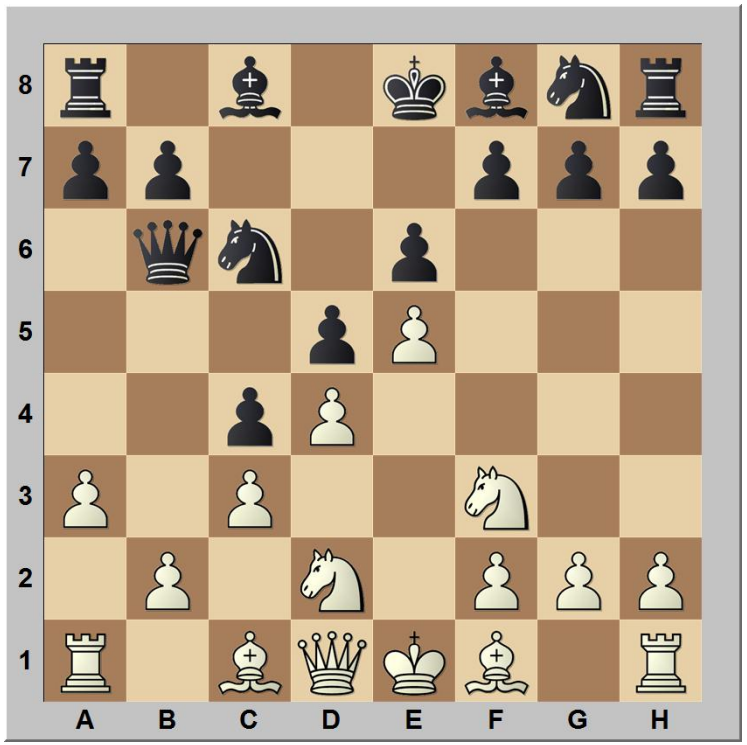


Chess game in PGN format

```
[Event "RUS-ch playoff 65th"]
[Site "Moscow"]
[Date "2012.08.13"]
[Round "4"]
[White "Svidler, Peter"]
[Black "Andreikin, Dmitry"]
[Result "0-1"]
[WhiteElo "2749"]
[BlackElo "2715"]

1. e4 e6 2. d4 d5 3. e5 c5 4. c3 Nc6 5. Nf3 Qb6 6. a3 c4 7. Nbd2 Bd7 8. g3 Na5
9. h4 Ne7 10. Bh3 h6 11. h5 Nc8 12. O-O Qc7 13. Ne1 Nb6 14. Qe2 O-O-O 15. Ng2
Be7 16. Rb1 Rdg8 17. f4 g6 18. Nf3 Kb8 19. Kh2 Nc6 20. Be3 Bd8 21. Bf2 Ne7 22.
g4 gxh5 23. gxh5 Nf5 24. Rg1 Ng7 25. Nd2 f5 26. exf6 Bxf6 27. Nf1 Nc8 28. Ng3
Nd6 29. Ne3 Bh4 30. Qf3 Be8 31. Bg4 Qf7 32. Rbf1 Bxg3+ 33. Bxg3 Ngf5 34. Re1
Ne4 35. Bxf5 exf5 36. Bh4 Nd2 37. Qe2 Qxh5 38. Qxh5 Bxh5 39. Bf6 Nf3+ 40. Kh1
Nxe1 41. Bxh8 Bf3+ 42. Kh2 Rxg1 43. Kxg1 Be4 0-1
```

Finding frequent move sequences in chess games



Typical plan could be `Be2/0-0/Re1/Rb1/Nf1`.

Definition (Node Labeled Graph)

A node labeled graph is a quadruple $G = (V, E, \Sigma, L)$ where:

1. V is the set of nodes,
2. E is the set of edges,
3. Σ is a set of labels, and
4. $L : V \rightarrow \Sigma$ is a labeling function that assigns labels from Σ to nodes in V.

Labeled Rooted Tree

Definition (Labeled Rooted Tree)

A labeled rooted tree $U = (V, E, \Sigma, L, v^r)$ is an acyclic undirected connected graph $G = (V, E, \Sigma, L)$ with a special node v^r called the root of the tree.

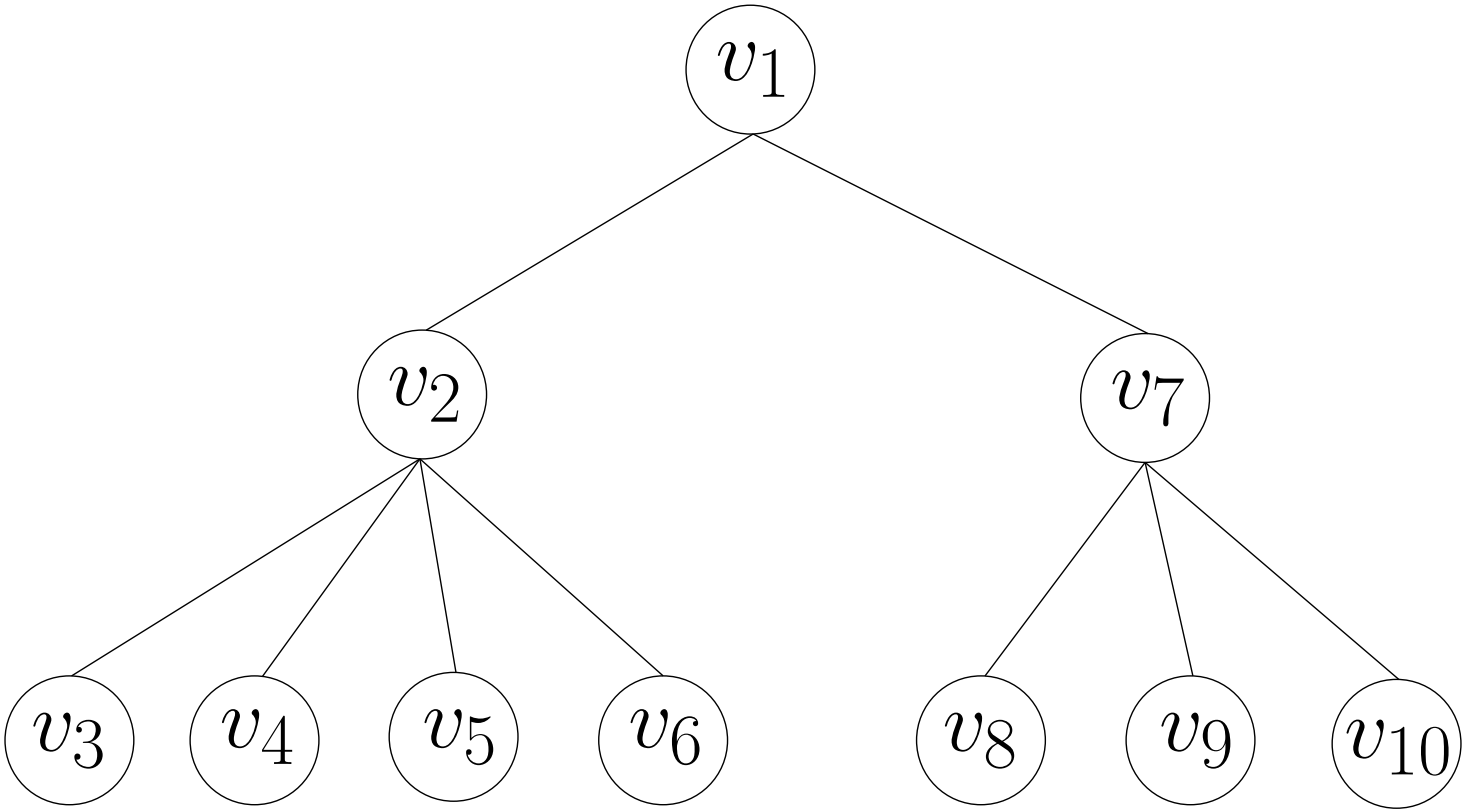
There exists exactly one path between the root node and any other node in V.

Labeled Rooted Ordered Tree

Definition (Labeled Rooted Ordered Tree)

A labeled rooted ordered tree $T = (V, E, \Sigma, L, v^r, \leq)$ is a labeled rooted tree $U = (V, E, \Sigma, L, v^r)$ where between all the siblings an order $\leq$ is defined. To every node in an ordered tree a preorder $(\text{pre}(v))$ number is assigned according to the depth-first preorder traversal of the tree.

Node Numbering according to Preorder Traversal



Tree Inclusion Relations

1. Induced subtree.
2. Embedded subtree.

Induced Subtree: definition

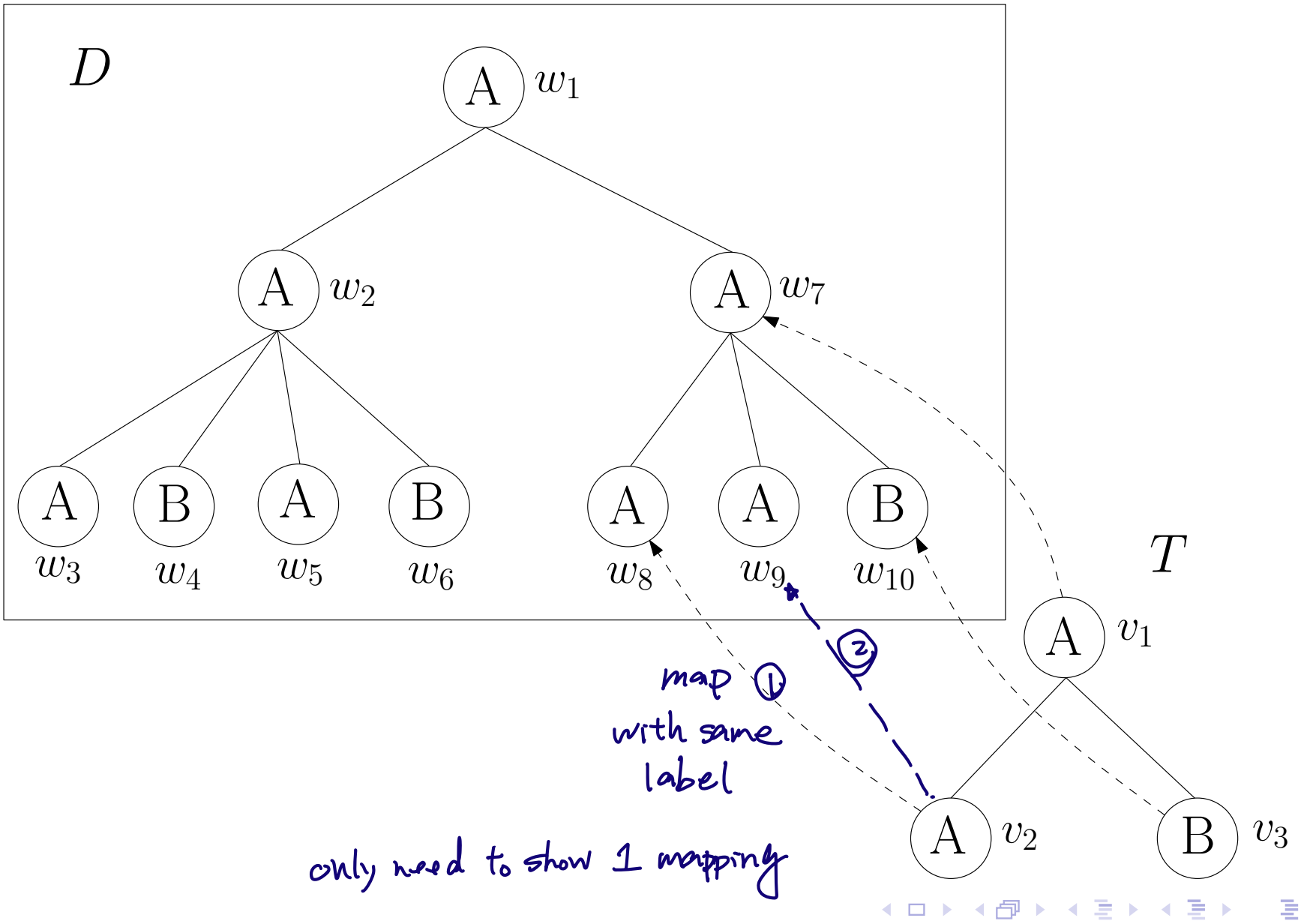
Let $\pi(v)$ denote the parent of node v.

Definition (Induced Subtree)

Given two ordered trees D and T, we call T an induced subtree of D if there exists an injective (one-to-one) matching function ϕ of V_T into V_D satisfying the following conditions:

1. ϕ preserves the labels: $L_T(v) = L_D(\phi(v))$.
2. ϕ preserves the left to right order between the nodes: $\text{pre}(v_i) < \text{pre}(v_j) \Leftrightarrow \text{pre}(\phi(v_i))) < \text{pre}(\phi(v_j))$.
3. ϕ preserves the parent-child relation: $v_i = \pi_T(v_j) \Leftrightarrow \phi(v_i) = \pi_D(\phi(v_j))$.

Induced Subtree: example



Induced Subtree: example

The matching function

1. $\phi(v_1) = w_7$
2. $\phi(v_2) = w_8$
3. $\phi(v_3) = w_{10}$

is one-to-one: each node in T is mapped to a different node in D.

Also verify that

1. $L_T(v_1) = L_D(w_7) = A$
2. $L_T(v_2) = L_D(w_8) = A$
3. $L_T(v_3) = L_D(w_{10}) = B$

We can verify that the other conditions are met as well, so T is an induced subtree of D.

Embedded Subtree: definition

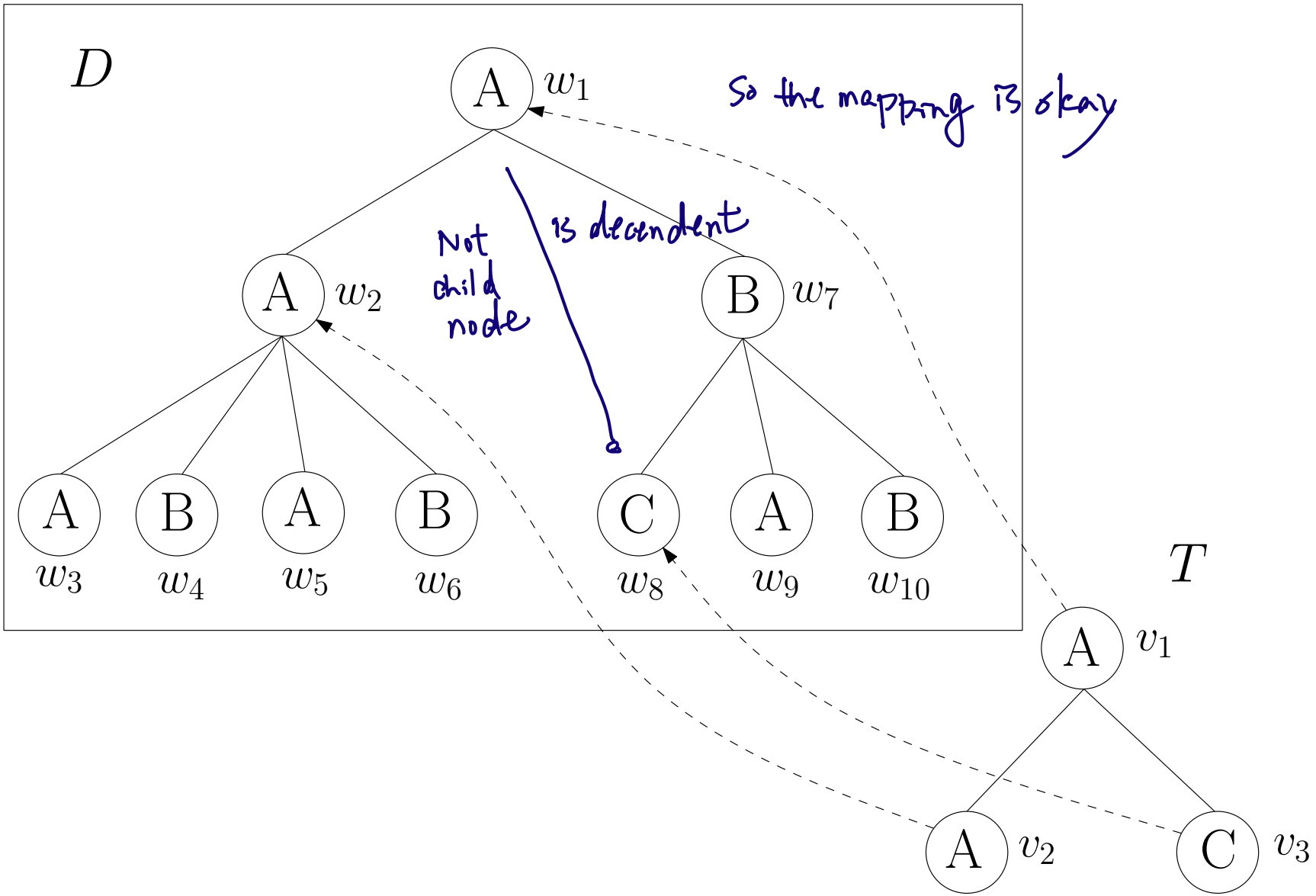
Let $\pi^*(v)$ denote the set of ancestors of v.

Definition (Embedded Subtree)

Given two ordered trees D and T, we call T an embedded subtree of D if there exists an injective matching function ϕ of V_T into V_D satisfying the following conditions:

1. ϕ preserves the labels: $L_T(v) = L_D(\phi(v))$.
2. ϕ preserves the left to right order between the nodes: $\text{pre}(v_i) < \text{pre}(v_j) \Leftrightarrow \text{pre}(\phi(v_i))) < \text{pre}(\phi(v_j))$.
3. ϕ preserves the ancestor-descendant relation: $v_i \in \pi^*_T(v_j) \Leftrightarrow \phi(v_i) \in \pi^*_D(\phi(v_j))$.

Embedded Subtree: example



The Frequent Tree Mining Task

Given a database of trees $D = \{d_1, d_2, \ldots, d_n\}$ and a tree inclusion relation $\subseteq$, we define the support of a tree T as

$$\sup(T, D) = |\{d \in D \mid T \subseteq d\}|$$

Given a minimum support threshold minsup, compute

$$\mathcal{F}(\text{minsup}, D) = \{T \mid \sup(T, D) \geq \text{minsup}\}$$

focus on induce subtree

Anti-Monotonicity Property

For a database of trees D, and two trees T_1 and T_2, we have

$$T_1 \subseteq T_2 \Rightarrow \sup(T_1, D) \geq \sup(T_2, D),$$

because $\forall d \in D : T_2 \subseteq d \Rightarrow T_1 \subseteq d$.

Hence, in a level-wise search for frequent trees, there is no point in expanding infrequent trees.

Example: Mining Frequent Induced Trees with FREQT

We must address two basic issues:

1. Generate candidate frequent trees:

   FREQT does this by adding a single node with a frequent label to a frequent tree. This is done by so-called *right-most extension*.

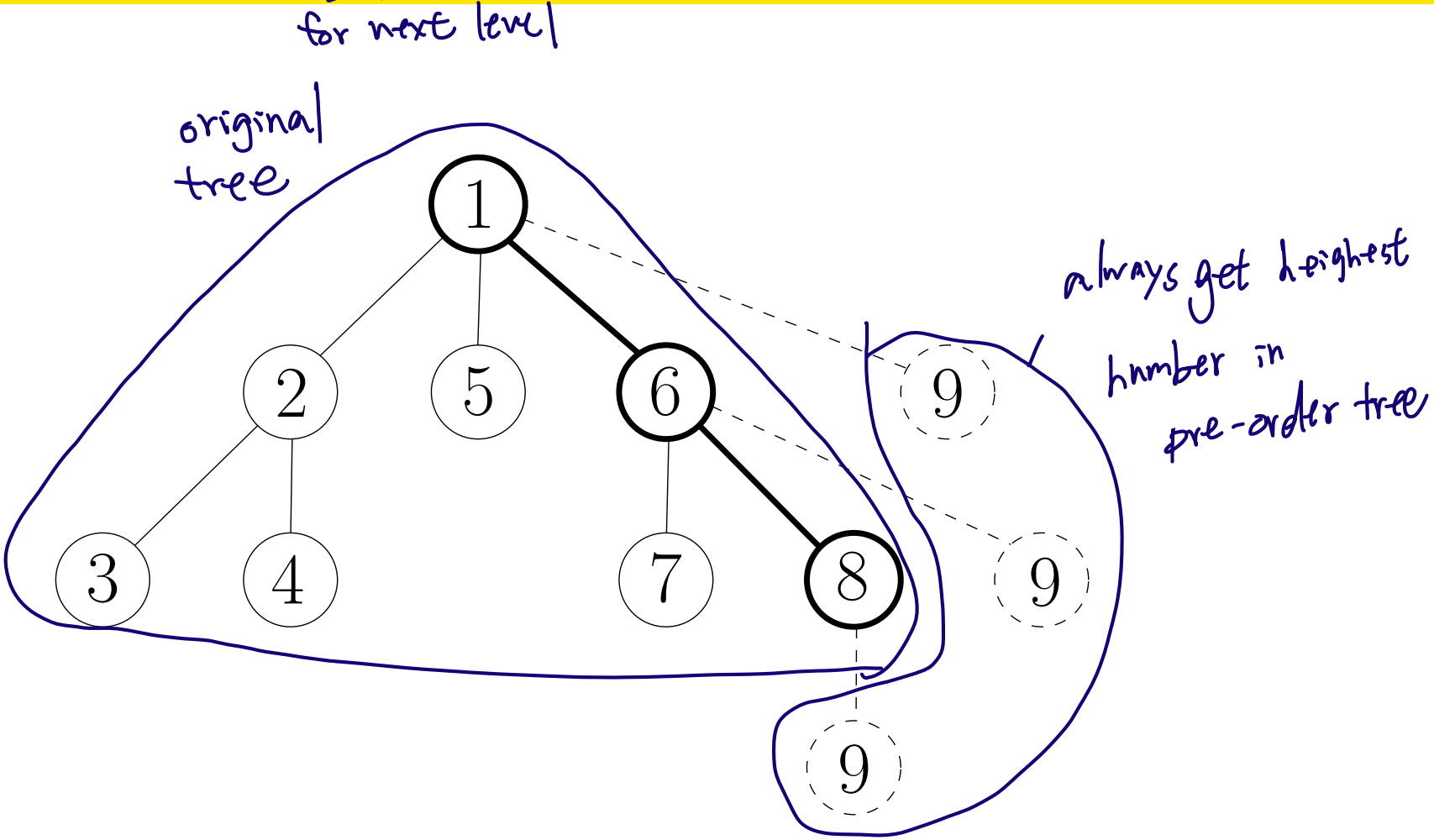
2. Record the occurrences of the candidate trees in the data trees, and determine whether they are frequent.

- Generating Candidates: Right-most Extension

Let T_k denote a tree of size k (a tree with k nodes).

- Consider the node numbering of T_k according to depth first pre-order traversal of the tree.
- The right-most branch of the tree is the path from the root node to the right-most leaf (i.e. the node with number k).
- To expand the tree T_k, it is only allowed to add a node as the right-most child of a node on the right-most branch of T_k. This node gets number $k+1$, as it is the last node in the traversal of T_{k+1}.

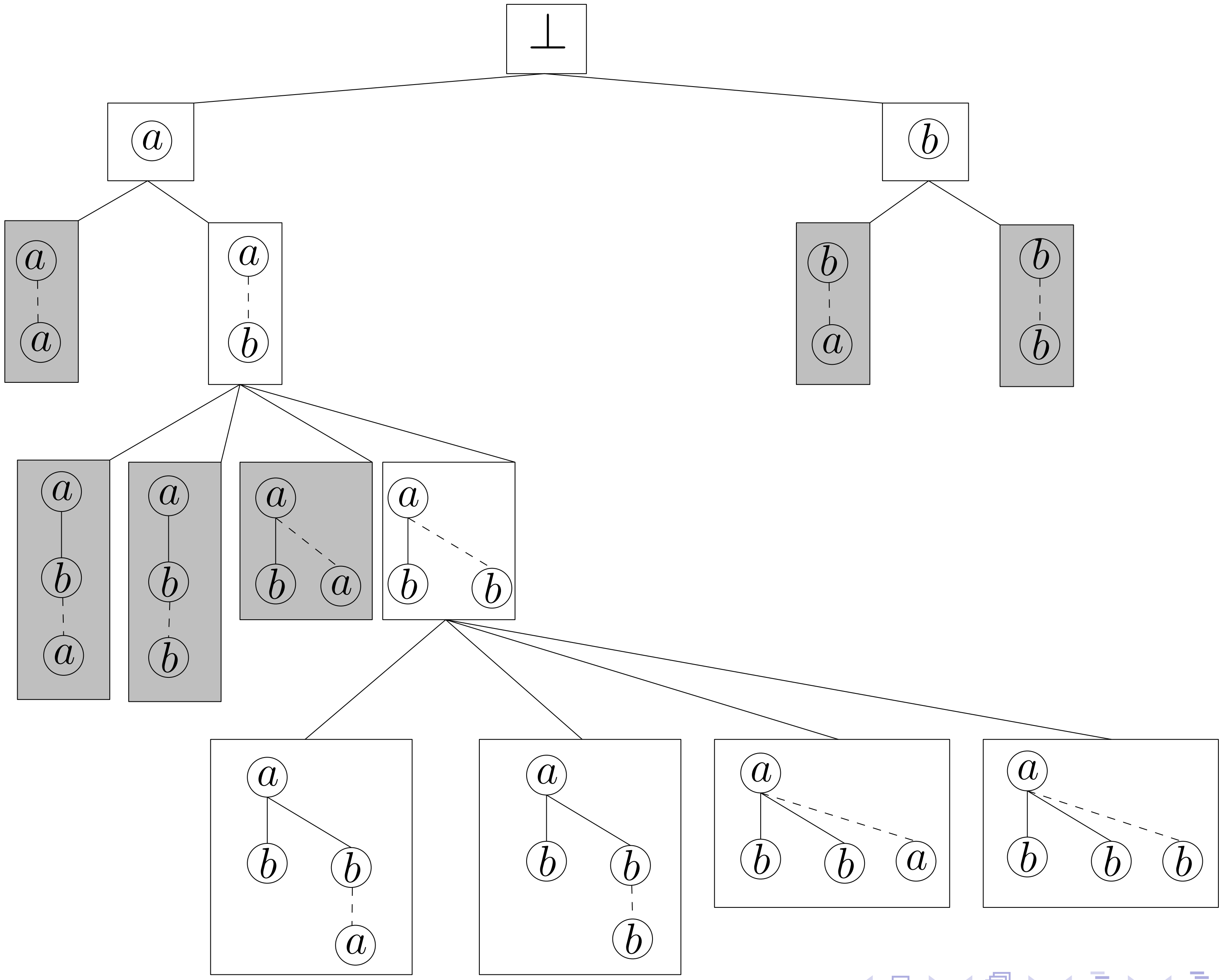
Generating Candidates: Right-most Extension



Right-most branch is fat.

Possible extensions are dashed.

Right-most Extension with label set $\Sigma = \{a, b\}$



Right-most Extension

can produce any tree and produce only once.

The right-most extension technique generates each tree at most once.

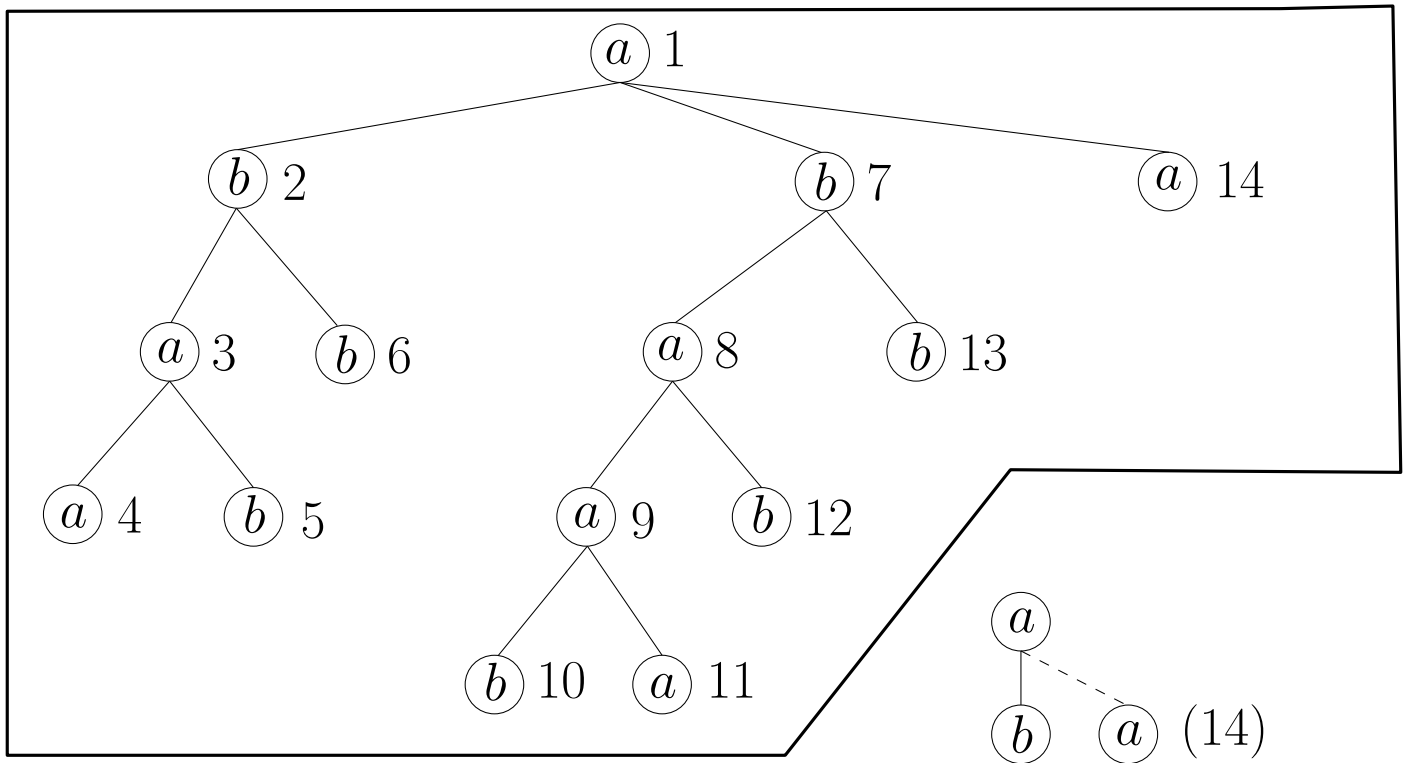
Consider any tree T_k. This tree only has one predecessor in the generation sequence, namely the tree T_{k-1} that is obtained by removing the right-most leaf of T_k (i.e. the node with number k in the depth first pre-order traversal).

Also, the right-most extension technique generates every possible tree, so each tree is generated *exactly* once.

Occurrence List

- To determine whether a pattern tree occurs in a data tree, an occurrence list is maintained that contains the list of nodes in the data tree to which the nodes in the pattern tree can be mapped.
- FREQT only stores the nodes of the data tree to which the right-most node in the pattern tree can be mapped.
- This is sufficient since only the nodes on the right-most branch are needed for future extension.

Right-most Occurrence List: Example



list all a's number

a (1,3,4,8,9,11,14)

a

b (2,7,5,12,10)

a

b

a (3,8)

a

b

a (14)

a

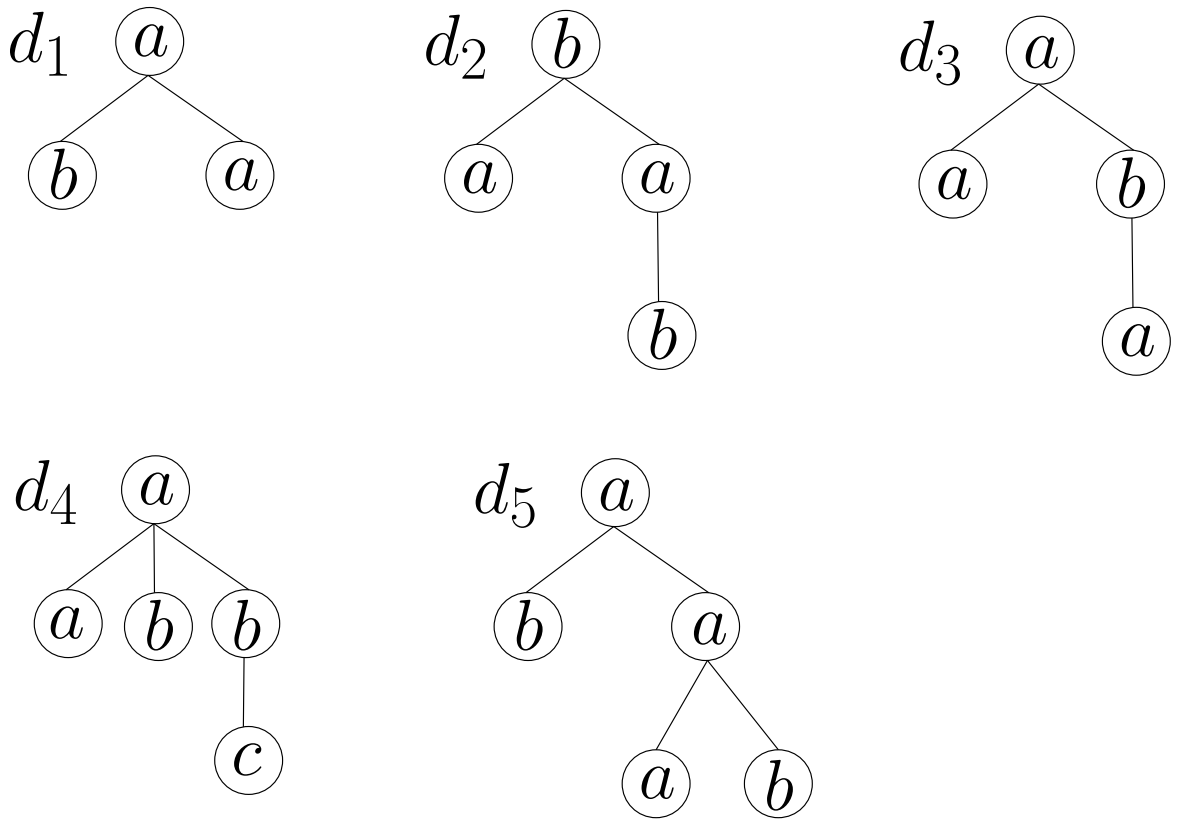
b

a

b (6,13)

Example

Consider the following database of labeled ordered trees:



Find all frequent induced subtrees with support at least 3.

Example: Level 1

At level 1 we have the following three candidates:

$$^{1}\textcircled{a} \quad ^{2}\textcircled{b} \quad ^{3}\textcircled{c}$$

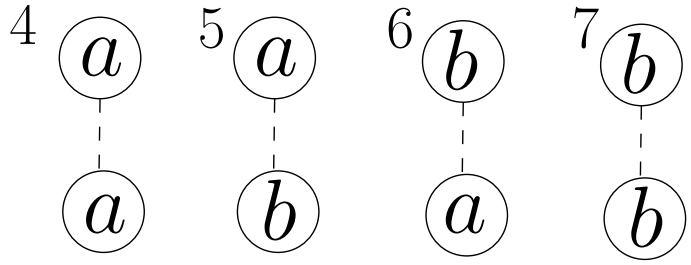
The right-most occurrence lists are:

data tree \ candidate	1	2	3
d_1	(1,3)	(2)	–
d_2	(2,3)	(1,4)	–
d_3	(1,2,4)	(3)	–
d_4	(1,2)	(3,4)	(5)
d_5	(1,3,4)	(2,5)	–
Support	5	5	1
Frequent?	✓	✓	✗

(min-support: 3)

Example: Level 2

At level 2 we have the following candidates:

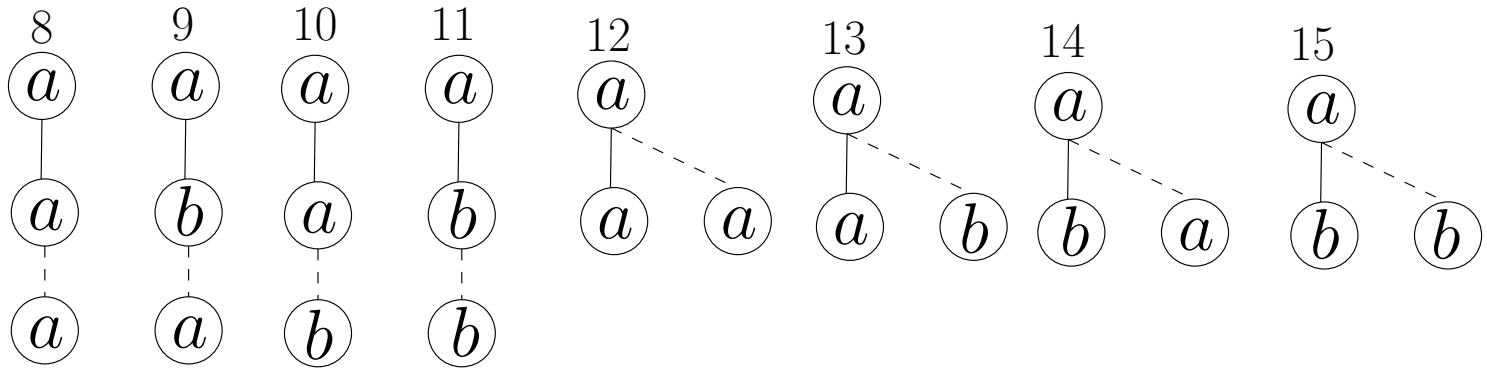


The RMO-lists are:

data tree \ candidate	4	5	6	7
d_1	(3)	(2)	–	–
d_2	–	(4)	(2,3)	–
d_3	(2)	(3)	(4)	–
d_4	(2)	(3,4)	–	–
d_5	(3,4)	(2,5)	–	–
Support	4	5	2	0
Frequent?	✓	✓	✗	✗

Example: Level 3

The level 3 candidates are:

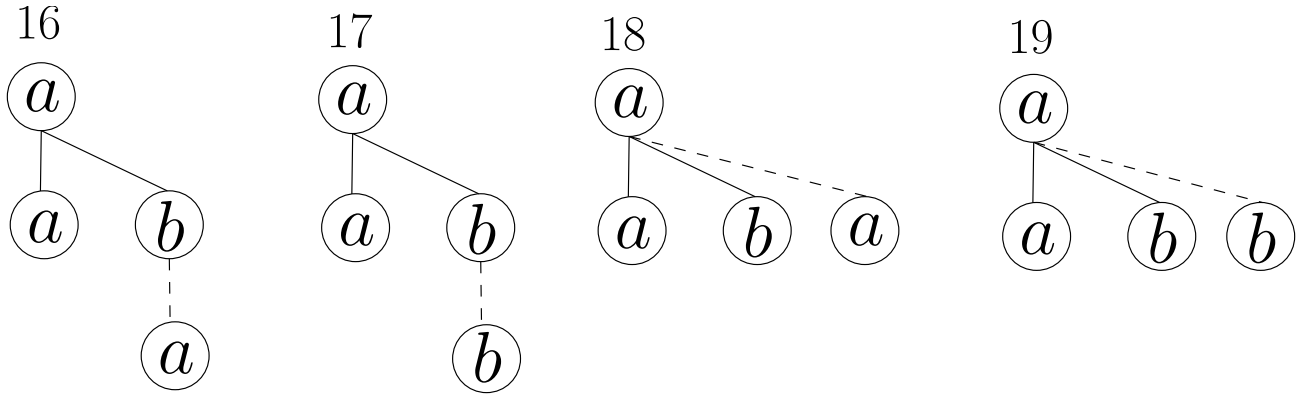


The RMO-lists are:

data tree \ candidate	8	9	10	11	12	13	14	15
d_1	–	–	–	–	–	–	(3)	–
d_2	–	–	–	–	–	–	–	–
d_3	–	(4)	–	–	–	(3)	–	–
d_4	–	–	–	–	–	(3,4)	–	(4)
d_5	(4)	–	(5)	–	–	(5)	(3)	–
Support	1	1	1	0	0	3	2	1
Frequent?	✗	✗	✗	✗	✗	✓	✗	✗

Example: Level 4

The level 4 candidates are:

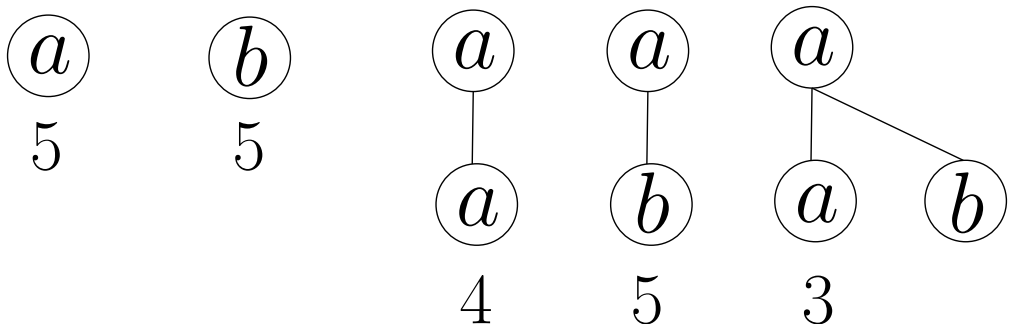


The RMO-lists are:

data tree \ candidate	16	17	18	19
d_1	–	–	–	–
d_2	–	–	–	–
d_3	(4)	–	–	–
d_4	–	–	–	(4)
d_5	–	–	–	–
Support	1	0	0	1
Frequent?	✗	✗	✗	✗

Example: final result

As the final result, the algorithm returns all frequent induced subtrees and their support:



Applications of frequent tree mining

- Mining usage patterns in Web logs.
- Mining frequent query patterns from XML queries.
- Classification of XML documents according to subtree structures.
- ...

Frequent Pattern Mining and Classification

Frequent pattern mining can also be used to extract features for classification tasks:

1. Find frequent patterns per class.
2. Define *discriminating* patterns, for example, as patterns that are frequent in one class but not in the other.
3. Use the presence/absence of such a discriminating pattern as a binary feature for constructing a classifier (e.g. classification tree!).
4. In this way we can include non-tabular data (sequences, trees, graphs) into an algorithm that requires a tabular data structure.

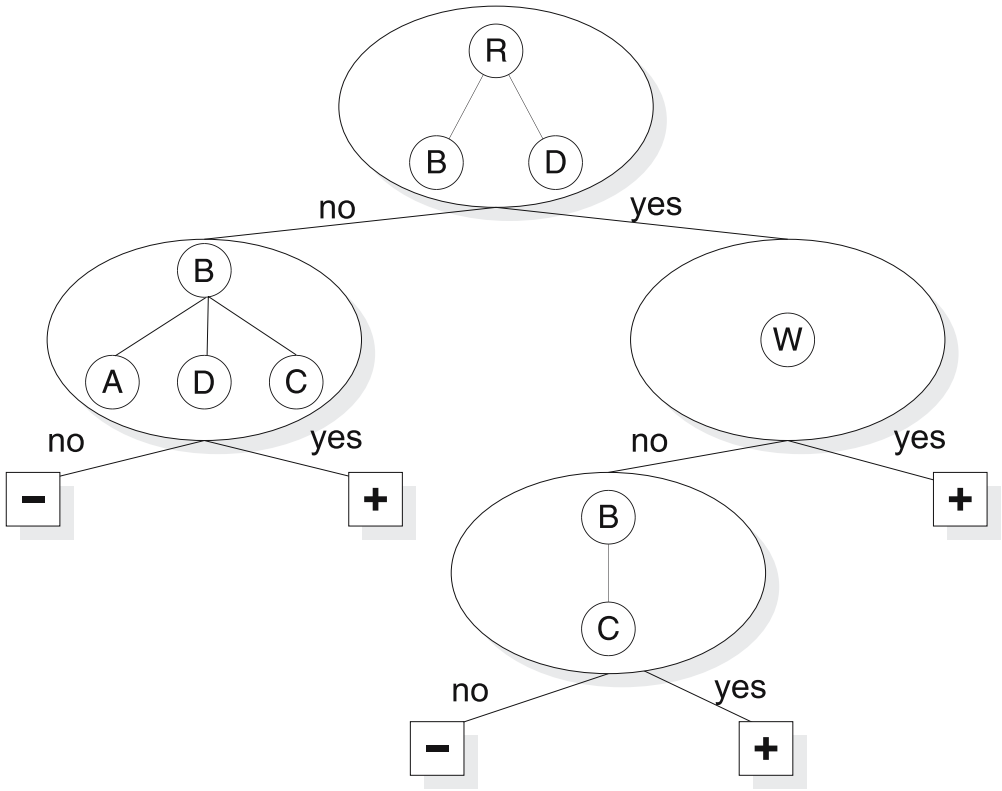


Fig. 4. A decision tree as produced by the TREE^2 algorithm