A Trie-based APRIORI Implementation for Mining Frequent Item Sequences

Ferenc Bodon

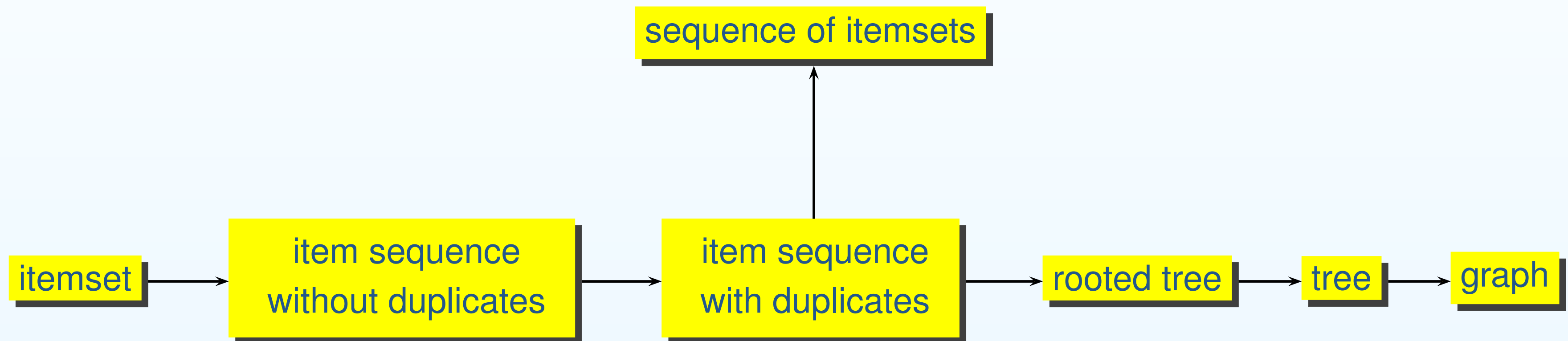
bodon@cs.bme.hu

Department of Computer Science and Information Theory,

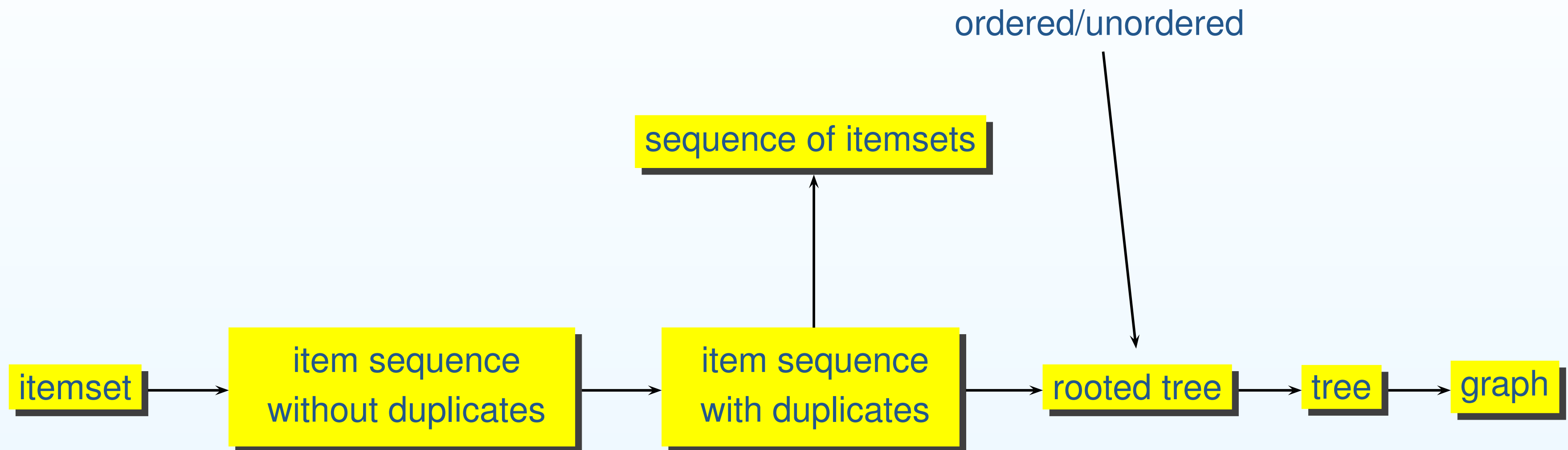
Budapest University of Technology and Economics

supervisor: Lajos Rónyai

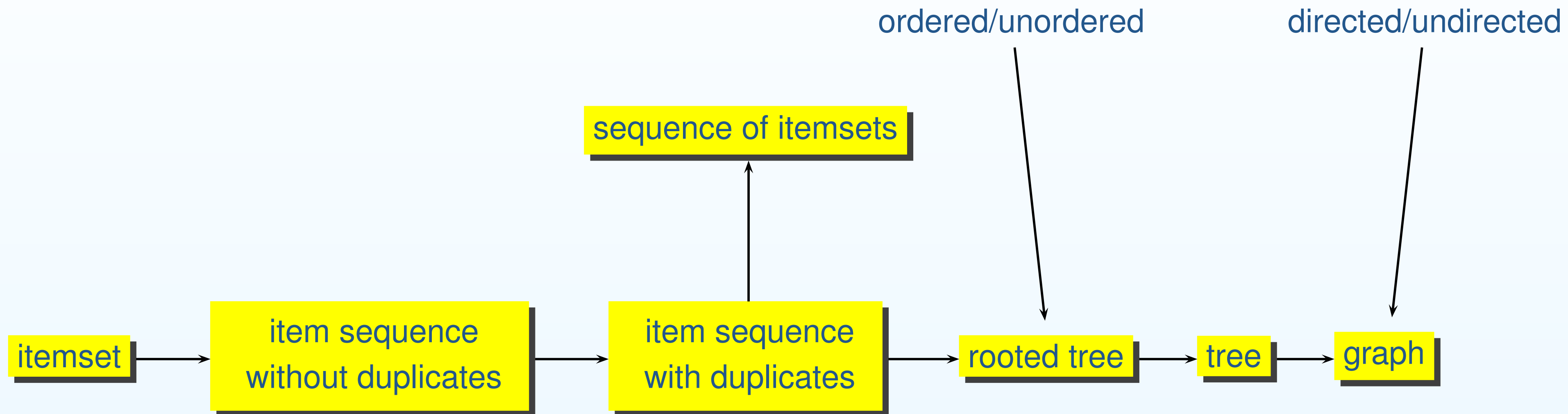
The hierarchy of the pattern types



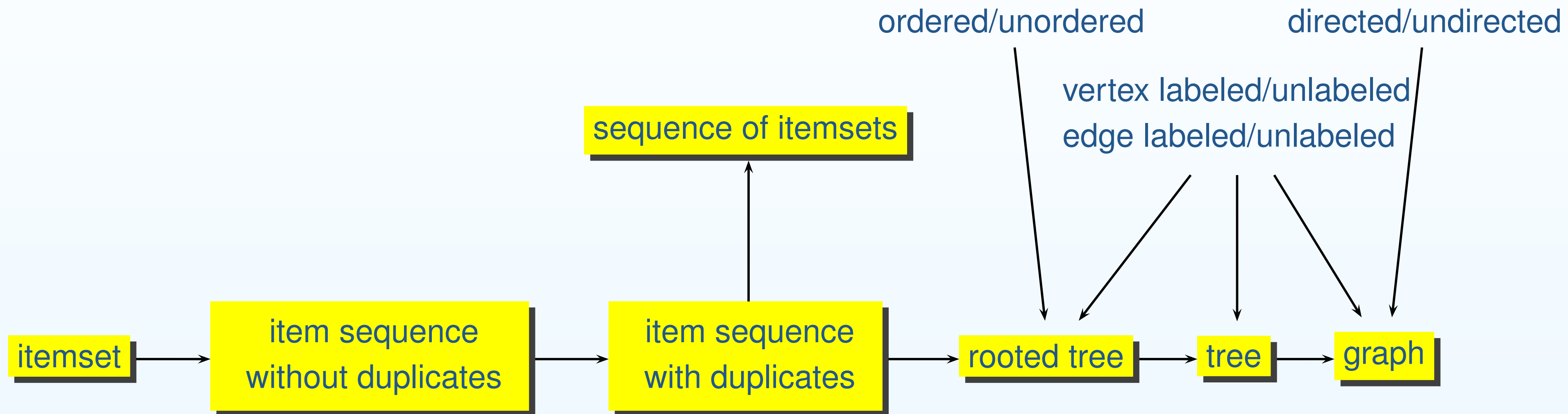
The hierarchy of the pattern types



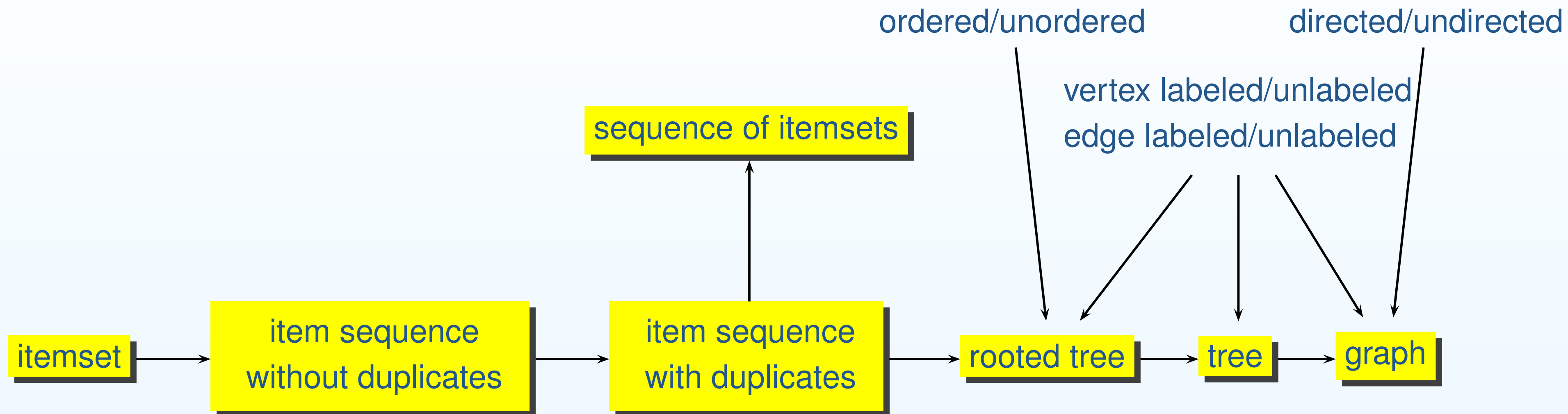
The hierarchy of the pattern types



The hierarchy of the pattern types



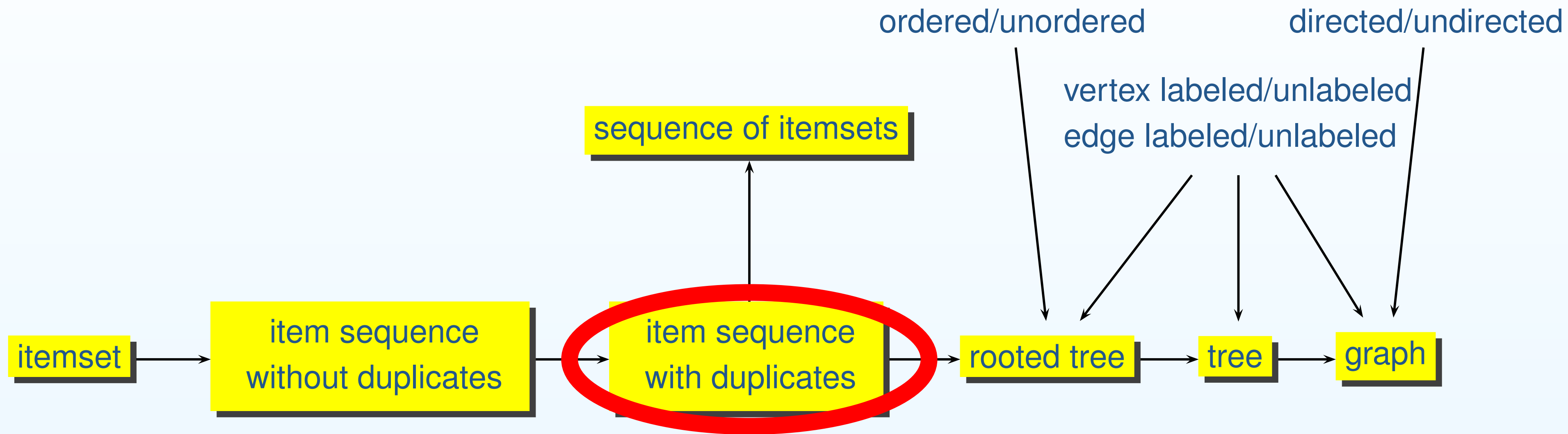
The hierarchy of the pattern types



At each generalization step we have to examine

- new problems, that come into play,
- applicability of the existing techniques at previous level.

The hierarchy of the pattern types



We provide an efficient, open-source, trie-based APRIORI implementation for mining frequent sequences of items in a transactional database

The background

We developed a FIM template library that includes

- a fast IO framework,
- some basic functions (e.g. fast subset enumerator),
- modularized Apriori, eclat, fp-growth, nonord-fp algorithms,
- tester classes,
- a benchmark environment.

In Apriori of FIM itemset are converted to item sequences, hence it is natural to extend FIM approach.

General Properties of the Trie

- The representation of the list of edges
  - label ordered list of (label, pointer) pairs,
  - tabular representation (also with offset-index trick),
  - hybrid solution

  is set by a template class
- no parent pointers are stored,
- dead-end branches are removed asap,
- classes that do
  - support counting,
  - candidate generation

  are set by template parameters.

Investigated techniques

1. Routing strategy at the nodes,
2. candidate generation,
3. equisupport extension,
4. filtered transaction caching.

Our environment makes it possible to examine each technique separately, and also to examine the effect on each other.

1/4. Routing Strategy

How to find the edge to follow in Apriori? Given a node with a list of n edges and a part of the filtered transaction (t'), find matching labels.

- search for corresponding item
  - if transaction is stored in a list: nt' comparisons,
  - indexarray solution,
- search for corresponding label
  - tabular representation of edge: t' comparisons,
  - binary search: $t' \log n$ comparisons,
  - linear search: $t'n$ comparisons,
  - intelligent linear search: $t'n/2$ comparisons,
- simultaneous traversal (merge)
  - first sort, then remove duplicates,
  - first remove duplicates, then sort.

1/4. Routing Strategy

- No single best routing strategy.
- The best routing strategy depends on
  - size of transaction,
  - the number of duplicates,
  - min_supp
  - $\ldots$
- Strategies with large overhead are not competitive at large support thresholds.
- Search corresponding item performed always good.
- It is more important how does the strategy suit to the features (prefetching, pipelining, etc.) of the modern processors than the worst case run-times.

2/4. Candidate Generation

Same as in FIM, there solutions can be applied

1. complete pruning with subset checks,
2. complete pruning with an intersection-based solution,
3. omit complete pruning.

2/4. Candidate Generation

Same as in FIM, there solutions can be applied

1. complete pruning with subset checks,
2. complete pruning with an intersection-based solution,
3. omit complete pruning.

Our expectation:

Complete pruning in frequent item sequence mining is more important than in FIM.

2/4. Candidate Generation

Same as in FIM, there solutions can be applied

1. complete pruning with subset checks,
2. complete pruning with an intersection-based solution,
3. omit complete pruning.

Our expectation:

Complete pruning in frequent item sequence mining is more important than in FIM.

Reasoning:

Support counting is slower, while determining if a candidate's subsequence is contained in a trie is achieved as fast as in FIM.

2/4. Candidate Generation

Same as in FIM, there solutions can be applied

1. complete pruning with subset checks,
2. complete pruning with an intersection-based solution,
3. omit complete pruning.

Our expectation:

Complete pruning in frequent item sequence mining is more important than in FIM.

Reasoning:

Support counting is slower, while determining if a candidate's subsequence is contained in a trie is achieved as fast as in FIM.

Experiments:

Support our hypothesis. Omitting complete pruning never resulted in a faster Apriori than intersection-based Apriori.

3/4. Equisupport Extension

Omitting equisupport extension is the most widely used technique in FIM.

Property 1. *Let* $X \subset Y \subseteq \mathcal{I}$. *If* $supp(X) = supp(Y)$ *then* $supp(Y \cup Z) = supp(X \cup Z)$ *for any* $Z \subseteq \mathcal{I} \setminus Y$.

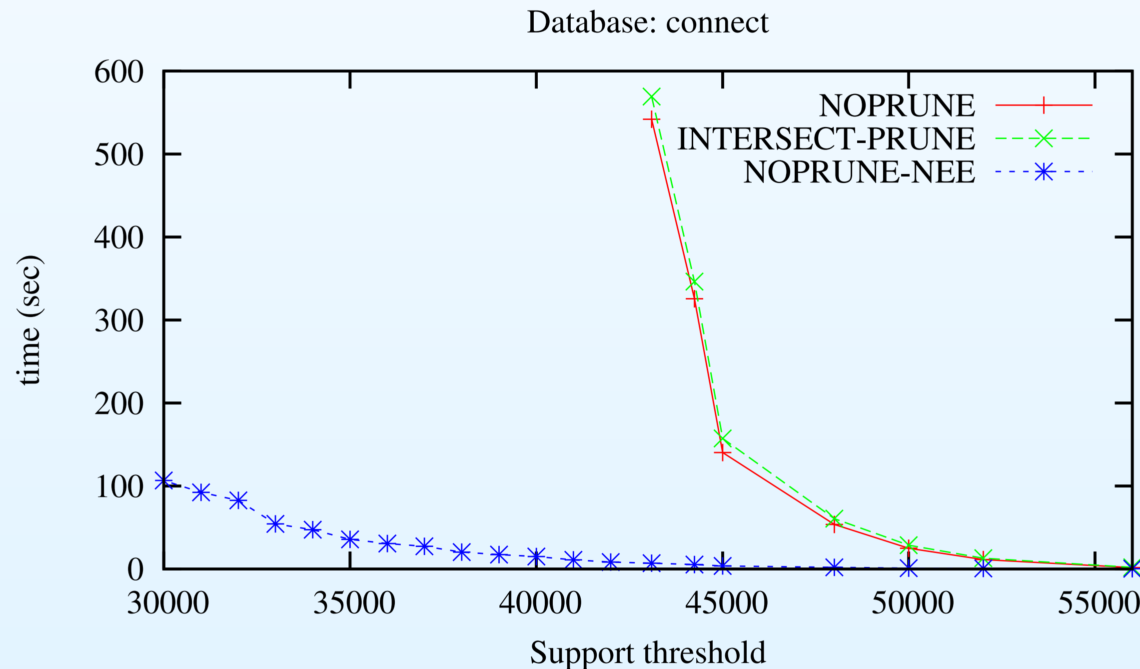
Usage: It is not necessary to determine the support of $Y \cup Z$.

3/4. Equisupport Extension

Omitting equisupport extension is the most widely used technique in FIM.

Property 1. *Let* $X \subset Y \subseteq \mathfrak{I}$. *If* $supp(X) = supp(Y)$ *then* $supp(Y \cup Z) = supp(X \cup Z)$ *for any* $Z \subseteq \mathfrak{I} \setminus Y$.

Usage: It is not necessary to determine the support of $Y \cup Z$.



3/4. Equisupport Extension

Omitting equisupport extension is the most widely used technique in FIM.

Property 1. *Let* $X \subset Y \subseteq \mathcal{I}$. *If* $supp(X) = supp(Y)$ *then* $supp(Y \cup Z) = supp(X \cup Z)$ *for any* $Z \subseteq \mathcal{I} \setminus Y$.

Usage: It is not necessary to determine the support of $Y \cup Z$.

Equisupport pruning in Apriori for FIM:

1. in infrequent candidate removal phase: $|X| = |Y| - 1$ and X is prefix of Y, then do not extend Y,
2. in candidate generation's subset check phase: $Y \cup Z$ is potential candidate, Y is non-prefix of $Y \cup Z$, X is prefix of Y then the support of $Y \cup Z$ is not determined.

3/4. Equisupport Extension

Omitting equisupport extension is the most widely used technique in FIM.

Property 1. *Let* $X \subset Y \subseteq \mathcal{I}$. *If* $supp(X) = supp(Y)$ *then* $supp(Y \cup Z) = supp(X \cup Z)$ *for any* $Z \subseteq \mathcal{I} \setminus Y$.

Usage: It is not necessary to determine the support of $Y \cup Z$.

Equisupport pruning in Apriori for FIM:

1. in infrequent candidate removal phase: $|X| = |Y| - 1$ and X is prefix of Y, then do not extend Y,
2. in candidate generation's subset check phase: $Y \cup Z$ is potential candidate, Y is non-prefix of $Y \cup Z$, X is prefix of Y then the support of $Y \cup Z$ is not determined.

Unfortunately, no equisupport extension pruning can be applied in frequent item sequence mining.

3/4. Equisupport Extension

Omitting equisupport extension is the most widely used technique in FIM.

Property 1. *Let* $X \subset Y \subseteq \mathcal{I}$. *If* $supp(X) = supp(Y)$ *then* $supp(Y \cup Z) = supp(X \cup Z)$ *for any* $Z \subseteq \mathcal{I} \setminus Y$.

Usage: It is not necessary to determine the support of $Y \cup Z$.

Equisupport pruning in Apriori for FIM:

1. in infrequent candidate removal phase: $|X| = |Y| - 1$ and X is prefix of Y, then do not extend Y,

Proof. By contradiction. Let the database $\mathcal{T}$ be $\{\langle A\rangle, \langle B, A\rangle\}$.
Then $supp(\langle\rangle) = supp(\langle A\rangle) = 2$ but
$supp(\langle B\rangle) = 1 \neq 0 = supp(\langle A, B\rangle)$. □

Unfortunately, no equisupport extension pruning can be applied in frequent item sequence mining.

4/4. Filtered transaction caching

Let t be a transaction.
filtered t: infrequent items are removed from *t*.
Collect the same filtered transaction and store them in memory.

Advantages:

- IO cost is reduced,

4/4. Filtered transaction caching

Let t be a transaction.
filtered t: infrequent items are removed from *t*.
Collect the same filtered transaction and store them in memory.

Advantages:

- IO cost is reduced,
- parsing costs are reduced,

4/4. Filtered transaction caching

Let t be a transaction.
filtered t: infrequent items are removed from *t*.
Collect the same filtered transaction and store them in memory.

Advantages:

- IO cost is reduced,
- parsing costs are reduced,
- the number of support count method calls is reduced.

4/4. Filtered transaction caching

Let t be a transaction.
filtered t: infrequent items are removed from *t*.
Collect the same filtered transaction and store them in memory.

Advantages:

- IO cost is reduced,
- parsing costs are reduced,
- the number of support count method calls is reduced.

Disadvantage:

- needs extra memory.
- requires CPU time to collect the same filtered transactions

Possible data structures: ordered list, trie, red-black tree, patricia tree

4/4. Filtered transaction caching

Expectation:

- Transaction caching is not such an effective technique as it is in FIM.

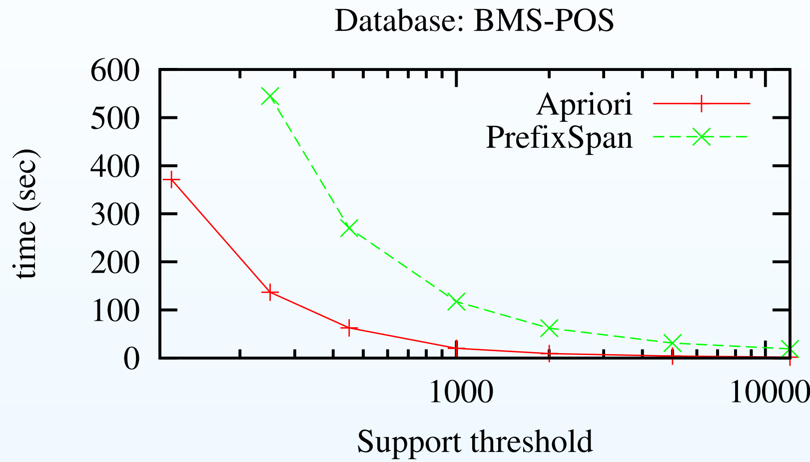
Reasoning:

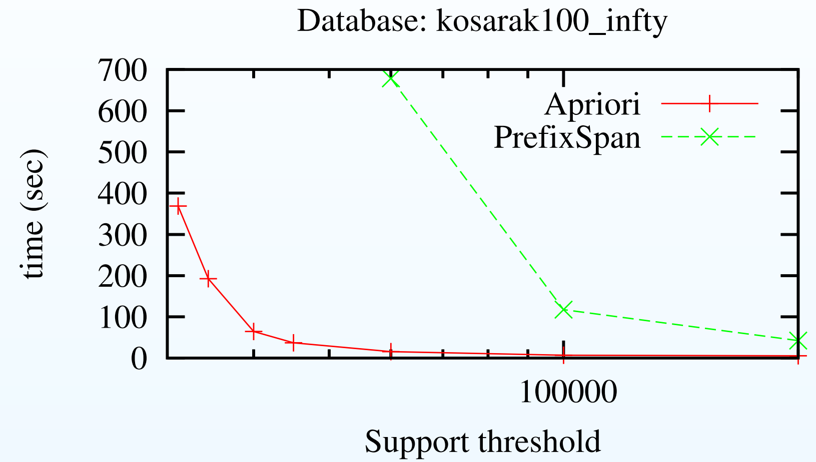
- For two item sequences to be equal, not just the items, but the indices of the same items have to be equal as well.

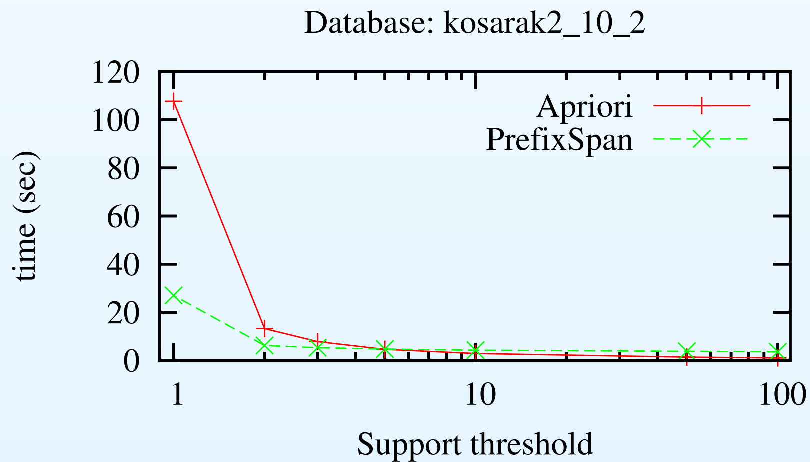
Experiments:

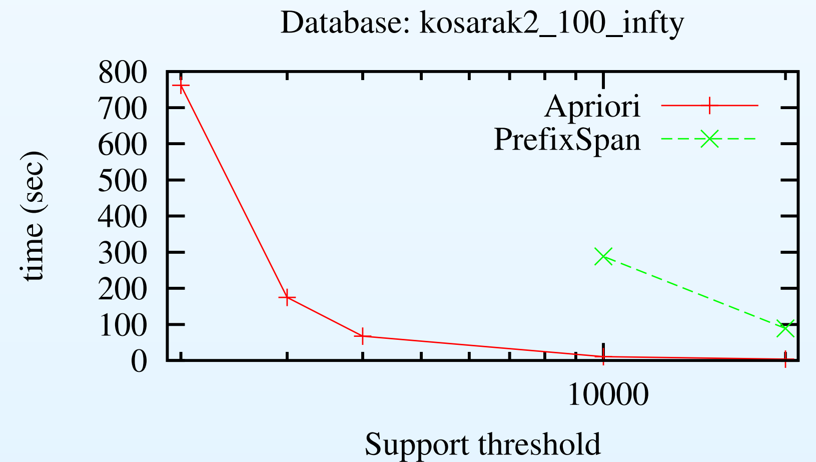
- never resulted in a faster algorithm,
- sometimes it significantly increases memory need.

Is our implementation competitive?





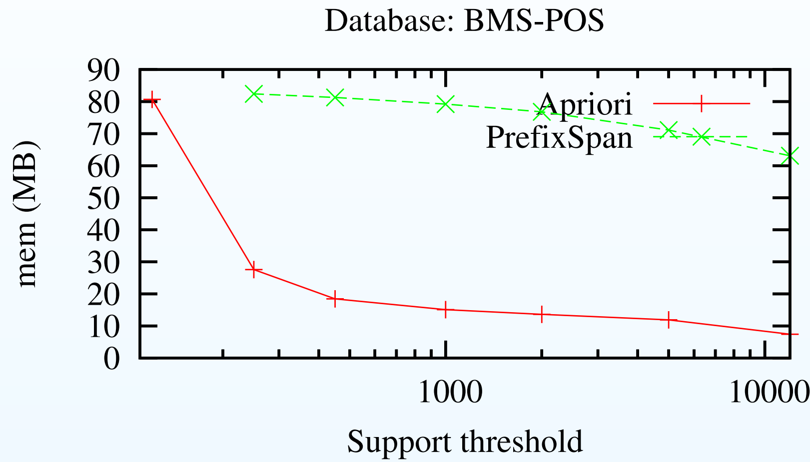


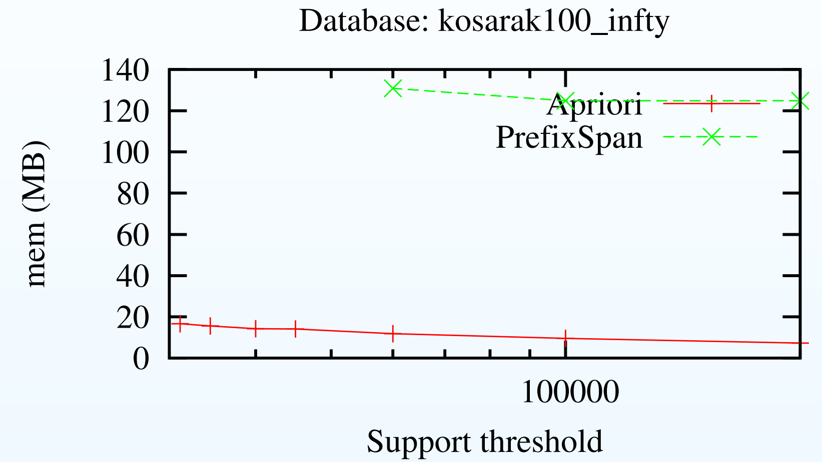


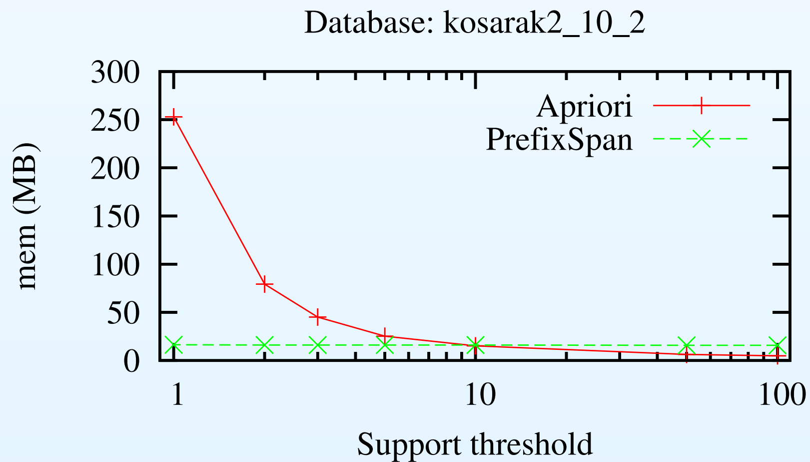
Run-times

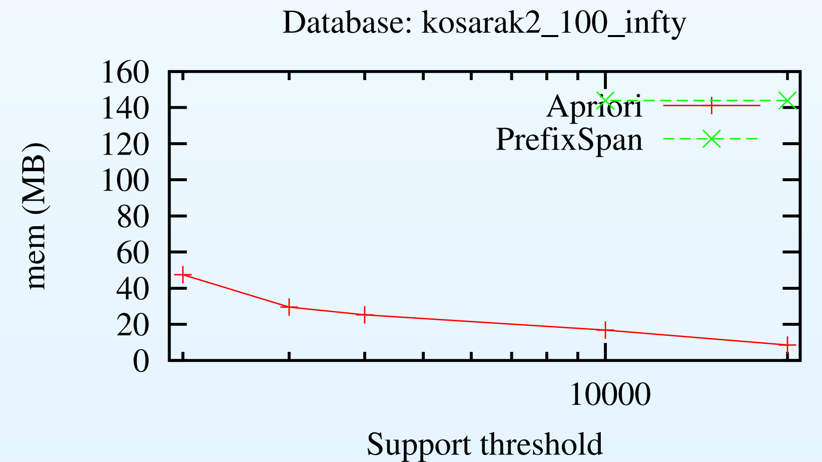
Apriori vs. prefixSpan by Taku Kudo

Is our implementation competitive?









Memory needs

Apriori vs. prefixSpan by Taku Kudo

Conclusion

When develping solutions for frequent item sequence mining it is useful to understand techniques used in FIM.

Thank you for your attention!