Frequent Pattern Mining: Exercises

Christian Borgelt

School of Computer Science
Otto-von-Guericke-University of Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany

christian@borgelt.net
http://www.borgelt.net/
http://www.borgelt.net/teach/fpm/

Frequent Item Set Mining: Notions

- How could information in the form of frequent item sets or association rules be used for **marketing purposes** etc.?
- What are other **potential applications** of frequent item set mining? (other than those mentioned on the motivations slide)
- What about the possible **multiplicity of items**?
  (a customer buys 3 cartons of milk, 5 apples, 2 cans of soup etc.)
  How can this be handled? Can it be handled by preprocessing?
- Why are the **transactions databases** represented as vectors/arrays?
  Why can't we use simple sets? What are alternative representations?
  What are advantages/disadvantages of alternative representations?
- Why is the **item base** often not given explicitly?
  How can it be obtained in such a case?
- How can absolute and relative **minimum support** be translated into each other?
  What additional information is needed for the transformation?

Frequent Item Set Mining: Support

- What does it mean that support is **anti-monotone** or **downward closed**?
  How can this be expressed as a formula?
- What relation of item set **covers** corresponds to the anti-monotonicity of support?
- What is the **apriori property**? How is it stated formally? How does it follow?
  What does the contraposition of the apriori property say? What does it suggest?
- Demonstrate that the support of an item set and one of its supersets can be equal!
  (That is, the support of the superset need not be actually smaller.)
  How can one describe the condition under which this holds?
  Can there be multiple different supersets that have the same support?
- Let I be an item set and a and b two items that are not in I.
  Let $s(I) = s(I \cup \{a\}) = s(I \cup \{b\})$. What is the support of $I \cup \{a, b\}$?
- Find an example where one can determine from the support of (several) item sets I_k that $\bigcup_k I_k$ must be frequent w.r.t. $s_{\min} \geq 3$ without accessing the transaction database (you may, however, use the size n of the transaction database).
  Do not use the solution/assumptions of the previous exercise!

Frequent Item Set Mining: Searching

- What is the size of the **search space** of frequent item set mining?
  How does it grow with the number of items?

- Why is an **exhaustive search** through all possible item sets not advisable?
  What is the (asymptotic) time complexity of an exhaustive search?
  Is the pruned/reduced search fundamentally more efficient?
  If yes, in what way? If no, why may it be useful nevertheless?

- How many subsets of size $k-1$ does an item set with k items have?
  Why is this a problem for the **candidate generation**?

- Show why any frequent item set of size $k+1$ can be formed in

  $$j = (k+1)k/2$$

  possible ways with the original version of the **Apriori** algorithm.
  Where does the denominator come from?

- In how many ways can an *in*frequent item set of size $k+1$ be generated?
  Does this number differ from the answer for frequent item sets? Why?

Frequent Item Set Mining: Searching

- What structure of the search space are we exploiting for the search?
  Why is this structure advantageous given the properties of support?
- What is a **Hasse diagram**? What does it represent?
  For what are we using Hasse diagrams?
- How is the Hasse diagram transformed to improve the search?
  What is the purpose of this transformation?
- What do we achieve by assigning **unique parent item sets**?
  Could we also assign **unique child item sets**?
- Are there any item sets that always have a unique parent item set,
  even in the unmodified Hasse diagram?
- How can we actually define unique parent item sets (concrete method)?
  Are there alternatives? How does this help in the search?
- After assigning **unique parent item sets**:
  In how many ways can an item set of size $k + 1$ be generated?

Frequent Item Set Mining: Searching

- What is the general scheme of searching with **unique parents**?
- What is a **canonical form** of an item set? How can it be defined?
  Is this concept actually needed for frequent item set mining?
  Why did we look at canonical forms for item sets anyway?
- What is the **prefix property**? How is it stated (two versions)?
  What advantage results from the prefix property?
- Describe how the **recursive search** can be simplified
  if the chosen canonical form has the prefix property!
- The **prefix property** is a necessary condition to ensure that the search
  by appending items to canonical code words is exhaustive.
  What is needed in addition to make it sufficient?
- Is there a **canonical extension rule** for frequent item set mining?
- What types of **pruning** the search (tree) can be distinguished?

Frequent Item Set Mining: Searching/Support

- How is **support** counted in the **Apriori** algorithm?
  What are the advantages/disadvantages of this counting scheme?
  What are the advantages of organizing the transactions as a prefix tree?

- How is **support** counted in the **Eclat** algorithm?
  What are the advantages/disadvantages of this counting scheme?

- Why does **Eclat** (usually) not exploit the apriori property fully?
  How could it be made to exploit the apriori property fully?

- **Item order**: Why is it advantageous in the **Eclat** algorithm
  to process the less frequent items first?

- How does the **occurrence deliver** scheme of **LCM** work?
  How is it related to the Eclat algorithm? How is it different?

- Why does **LCM** allow mining with memory proportional to the database size
  and **Eclat** does not, even though their processing schemes are similar?

Frequent Item Set Mining: Searching/Support

- How is support counted in the **SaM** algorithm?
  What does SaM have to ensure for the transaction array for this to work?
- Why can the merge step of the **SaM** algorithm be a problem?
- How does the **RElim** algorithm improve over **SaM**?
  What sorting principle is exploited?
- How is a k**-items machine** related to the **RElim** algorithm?
  What are the differences and the advantages/disadvantages?
- What is an **FP-tree**? How is it constructed from a transaction database?
  Why does it combine a horizontal and a vertical representation?
- How is support counted with an **FP-tree**?
- How is an **FP-tree** projected? What does the projection yield?
- How do we obtain the conditional transaction database for the second subproblem in the **FP-growth** algorithm?

Frequent Item Set Mining: Data Representation

- Construct horizontal and vertical representations of the matrix

	a	b	c	d	e	f	g
1:	1	0	0	1	0	1	0
2:	1	0	1	1	1	0	0
3:	0	1	0	1	0	0	0
4:	0	1	1	1	0	0	0
5:	0	1	1	0	0	0	0
6:	1	1	0	1	0	0	0
7:	0	1	0	1	1	0	0
8:	0	1	1	0	1	0	1
9:	0	0	1	1	0	1	0
10:	1	1	0	1	0	0	0

- Show that the diffset for an item (w.r.t. some conditional database), that is,

$$D(a \mid I) = K(I) - K(I \cup \{a\}),$$

can be both larger and smaller than its cover (depending on the database).

Frequent Item Set Mining: Data Representation

Construct the initial database representation for the **RElim algorithm** for this transaction database:

1:	$\{a, d, e\}$
2:	$\{b, c, d\}$
3:	$\{a, c, e\}$
4:	$\{a, c, d, e\}$
5:	$\{a, e\}$
6:	$\{a, c, d\}$
7:	$\{b, c\}$
8:	$\{a, c, d, e\}$
9:	$\{c, b, e\}$
10:	$\{a, d, e\}$

Construct the initial **FP-tree** for this transaction database:

1:	$\{a, d, e\}$
2:	$\{b, c, d\}$
3:	$\{a, c, e\}$
4:	$\{a, c, d, e\}$
5:	$\{a, e\}$
6:	$\{a, c, d\}$
7:	$\{b, c\}$
8:	$\{a, c, d, e\}$
9:	$\{c, b, e\}$
10:	$\{a, d, e\}$

You may skip the steps of finding the item frequencies and resorting the items in the transactions w.r.t. the frequency order (that is, you may use the alphabetical order).

Frequent Item Set Mining: Maximal and Closed

- Find the frequent / maximal / closed item sets for the following transaction vector and $s_{\min} = 3$ (you may or may not use some concrete algorithm):

1:	$a\ d\ f$	6:	$a\ c\ d\ e$
2:	$b\ d$	7:	$b\ c\ d$
3:	$b\ c$	8:	$a\ b\ d$
4:	$b\ d\ e$	9:	$b\ c\ e\ g$
5:	$c\ d\ f$	10:	$a\ b\ d$

- Find an example of a transaction database for which
  the number of maximal item sets goes down if the minimum support is reduced;
  or explain in some other way why this may happen!

- How are maximal and closed item sets related to each other?
  Can one be expressed in terms of the other?
  What information is preserved/lost with maximal/closed item sets? How?

- Is it possible that the number of all frequent or the number of closed frequent item sets goes down if the minimum support is reduced? If yes, give an example! If no, argue why it is not possible!

Frequent Item Set Mining: Maximal and Closed

- Define **closed item sets** with the help of the notion of a perfect extension.
- Characterize the set $M_T(1)$ (for an arbitrary transaction database T; that is, your characterization should work for all possible transaction databases T.)
- Suppose we have $\forall s_{\min} : F_T(s_{\min}) = C_T(s_{\min}) = M_T(s_{\min})$.
  What does the transaction database T look like?
- Suppose we have only $\forall s_{\min} : C_T(s_{\min}) = M_T(s_{\min})$.
  What does the transaction database T look like?
- Suppose we have only $\forall s_{\min} : C_T(s_{\min}) - \{\emptyset\} = M_T(s_{\min}) - \{\emptyset\}$.
  What does the transaction database T look like?
- How are closed item sets and closed sets of transaction identifiers related?
  What does it mean for a transaction index set that it is closed?
- How does the **Carpenter** algorithm exploit this relationship?
- What recursive relationship is the **IsTa** algorithm based on?

Frequent Item Set Mining: Evaluation

- Suppose the divide-and-conquer scheme for finding frequent item sets is extended with perfect extension pruning, so that subproblems are described by triples $S = (T, P, X)$.

  Is $P \cup X$ always a closed item set? Is it always a maximal item set?

- Find some other measure(s) on the partial order $(2^B, \subseteq)$ that is/are **anti-monotone** (or downward closed)!

- Show that the quotient of observed and expected frequency of an item set (expected w.r.t. an assumption of full independence) is **not anti-monotone**! Try to find counterexamples with $|I| \geq 1$ and with $|I| \geq 2$.

  Show that the reciprocal value of this quotient is also **not anti-monotone**!

- Show that if an item set I and an item a are independent, $\varrho_{\text{fi}}(I \cup \{a\}) = \varrho_{\text{fi}}(I)$. Show that, in contrast to this, $\varrho_{\text{ii}}(I \cup \{a\}) = 1$ in this case!

- Find an example where ϱ_{si} scores an item set I high, even though I is composed of two independent subsets!

Association Rule Induction

- What are **association rules**? With what measures are they evaluated?
- What is the **minimum support** of an association rule (two versions)?
  What is the **minimum confidence** of an association rule?
- How are association rules induced? What steps are needed?
  How is the induction related to frequent item set mining?
- With what **minimum support** do we have to find frequent item sets
  for association rule induction? Is it the minimum support of association rules?
  Or does it depend on the choice of the rule support definition? If yes, how?
- How are association rules formed and filtered
  once we have found the needed frequent item sets?
  With respect to what do we have to filter the association rules?
- Do we have to filter association rules w.r.t. the user-specified minimum support?
  If yes: always or only under certain conditions?

Association Rule Induction

- Show that the two different definitions of **rule support** lead to different results in the sense that there is no generally valid transformation of the one kind of minimum support into the other, so that the other yields an equivalent result!

- What relationships hold between the **confidence** values of different association rules formed from the same item set?
  How can we exploit these relationships in the generation of association rules?

- What are some measures other than minimum support and minimum confidence with which association rules may be evaluated?
  From what values are they usually computed?

- What is the **lift** value and how is it defined?
  What is the **information gain** of an association rule?
  What is the χ^2**-measure** of an association rule?

- Are association rules with **more than one item in the consequent** useful?
  What additional information do they provide?

Molecular Fragment Mining

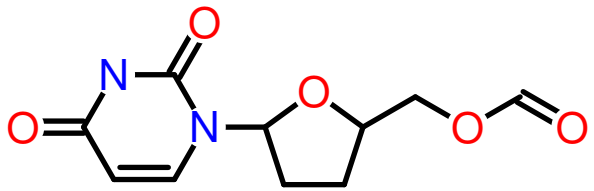
- Draw the molecule that is described by the following SMILES string:

  `S2c1c4c(ccc1N(c3c2cccc3)C(=O)C)cccc4`

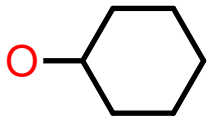
- Draw the molecule that is described by the following SLN string:

  `NH2C[1]:N:C(:C(:N:C:@1C(=O)OCH3)C(=O)C[2]:CH:CH:CH:CH:CH:@2)NH2`

- Construct a SMILES description for the following molecule:



- Find at least three different SMILES description of phenol, that is, of the molecule:



Molecular Fragment Mining

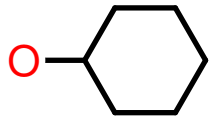
Draw the molecule that is described by the following SDfile:

```
4728
          1016 111283D                          4728

  9  9        0                1 V2000
   -0.0965    1.3884    0.0104 N   0  0  0  0  0  0  0  0  0
    1.8297   -0.2821   -0.0166 S   0  0  0  0  0  0  0  0  0
    1.0302    2.1371    0.0038 N   0  0  0  0  0  0  0  0  0
    0.1607    0.0664    0.0009 C   0  0  0  0  0  0  0  0  0
    2.1519    1.3924   -0.0108 C   0  0  0  0  0  0  0  0  0
    3.3971    1.9238   -0.0196 N   0  0  0  0  0  0  0  0  0
   -0.5873   -0.6698    0.0040 H   0  0  0  0  0  0  0  0  0
    3.5072    2.8871   -0.0156 H   0  0  0  0  0  0  0  0  0
    4.1746    1.3453   -0.0300 H   0  0  0  0  0  0  0  0  0
  1  4  2  0  0  0  0
  1  3  1  0  0  0  0
  2  4  1  0  0  0  0
  2  5  1  0  0  0  0
  3  5  2  0  0  0  0
  4  7  1  0  0  0  0
  5  6  1  0  0  0  0
  6  8  1  0  0  0  0
  6  9  1  0  0  0  0
M  END
$$$$
```

Frequent (Sub)Graph Mining

- How often does the fragment `C-C-C` occur in phenol, that is, in the molecule:



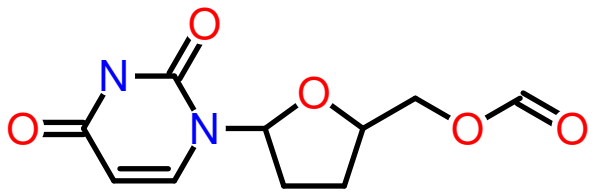
  In other words: How many **subgraph isomorphism**s exist?

- Does phenol (see above) possess an **automorphism** that is not the identity? How many different automorphisms does phenol possess?

- Construct the set of all **connected subgraphs** of phenol (see above)!

- Find all bonds in phenol (see above) that are **bridge**s!
  Are any of these bridges **proper bridge**s?

- Find all bonds in the following molecule that are **bridge**s! Which of these bridges are **proper bridge**s?



Frequent (Sub)Graph Mining

- Find all frequent (sub)graphs containing sulphur for a minimum support $s_{\min} = 3$ in the following graph database (SMILES format):

```
CCS(O)(O)N
CCS(O)(C)N
CS(O)(C)N
CCS(=N)N
CS(=N)N
CS(=N)O
```

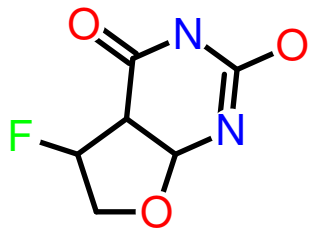
- Why is it more difficult to avoid **redundant search** when searching for frequent (sub)graphs than for frequent item sets? What are the main problems?

- What is the purpose of constructing **code words** of (sub)graphs?
  What is a **canonical code word**? What do we actually need?

- How do we assign a **unique parent (sub)graph**
  based on a canonical code words for a (sub)graph?
  What information from the canonical code word do we use?

Frequent (Sub)Graph Mining

- Why do we exclude the removal of proper bridges when assigning unique parents?
- How many different **spanning trees** does the following graph/molecule possess?



  Justify your answer!

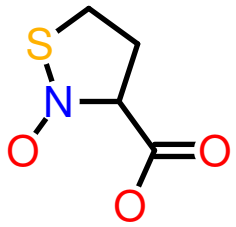
- Why is the number of possible code words usually (much) larger than the number of spanning trees? What additional choices does one have?
- Why does a coding scheme based on spanning trees in which edges closing cycles are listed after the spanning tree edges ensure that we can always the last edge? (In other words: why can we spare ourselves the check that this edge is not a proper bridge? What must the last edge rather be?)
- How many different (extended) **adjacency matrices** does the above graph/molecule possess?

Frequent (Sub)Graph Mining

- Check whether the code word

  `S 10-N 21-O 31-C 43-C 54-O 64=O 73-C 87-C 80-C`

  is the canonical code word, based on a depth-first search spanning tree, for the following graph/molecule:



  Use the order `S` $\prec$ `N` $\prec$ `O` $\prec$ `C` for the atoms and the order `-` $\prec$ `=` for the bonds!

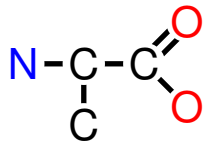
- Check whether the code word

  `S 0-N1 0-C2 1-O3 1-C4 2-C5 4-C5 4-C6 6-O7 6=O8`

  is the canonical code word, based on a breadth-first search spanning tree, for the above graph/molecule! Use the same orders as above!

Frequent (Sub)Graph Mining

- What are **rightmost path extensions**?
  What are **maximum source extensions**?
- Do rightmost path/maximum source extensions always yield canonical code words?
- Find the canonical code word, based on a depth-first search spanning tree, for clycin, that is, for



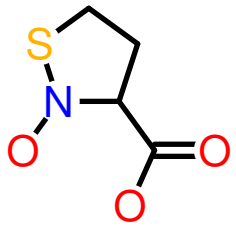
  Use the order `N` $\prec$ `O` $\prec$ `C` for the atoms and the order `-` $\prec$ `=` for the bonds!
- Which vertices/atoms of clycin are extendable, based on this canonical code word, by rightmost path extensions?
- Find the canonical code word, based on a breadth-first search spanning tree, for clycin (see above)! Use the same orders as above!
- Which vertices/atoms of clycin are extendable, based on this canonical code word, by maximum source extensions?

Frequent (Sub)Graph Mining

- Check whether the code word

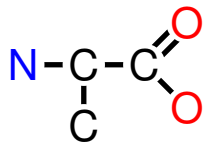
  `S 2 - 3 - N 4 - 5 - C 6 - O C 6 - 7 - C C 8 - 9 = O O`

  is the canonical code word, based on an (extended) adjacency matrix of which the upper triangle is read row by row, of the graph/molecule



  Use the order `S` $\prec$ `N` $\prec$ `O` $\prec$ `C` for the atoms and the order `-` $\prec$ `=` for the bonds!

- Find the canonical code word, based on an (extended) adjacency matrix of which the upper triangle is read row by row, for clycin, that is, for



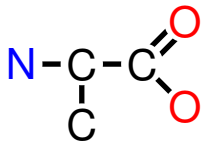
  Use the same orders of atoms and bonds as above!

Frequent (Sub)Graph Mining

- What is the purpose of the method of **vertex signatures**?
- How can we construct vertex signatures by iteratively splitting equivalence classes?
- What is an **automorphism** of a graph? What is the **orbit** of a vertex?
- Show that the automorphisms of a graph form a group (algebraic structure)!
- Why can we form a canonical code word very efficiently if we know that all vertex labels are pairwise different?
- Why can we form a canonical code word very efficiently if we know the orbits of all vertices?
- Can we always identify the orbits of the vertices of a graph by constructing vertex signatures? If yes, why can we be sure? If no: Are there special conditions under which we can be sure that we have identified the orbits? Are vertex signatures still helpful even if they do not identify all orbits?

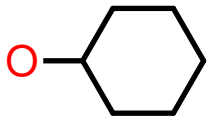
Frequent (Sub)Graph Mining

- Try to find unique labels for the vertices/atoms of clycin, that is, of



  using the method of vertex signatures! What does the resulting canonical code word look like (use the same orders as above)?

- Try to find unique labels for the vertices/atoms of phenol, that is, of

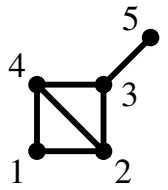


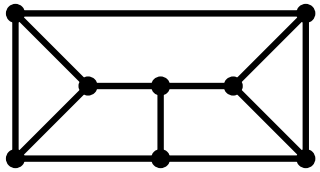
  using the method of vertex signatures. Why is it not possible to split all equivalence classes so that single element classes result?

- How is the canonical code word for phenol constructed? (That is, how can one deal with the fact that not all equivalence classes can be reduced to single elements?)

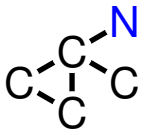
Frequent (Sub)Graph Mining

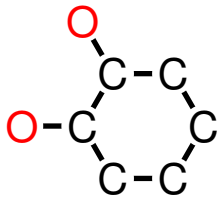
- Determine the orbits of the vertices for the following graphs:





- Determine the orbits of the vertices for the following graphs:



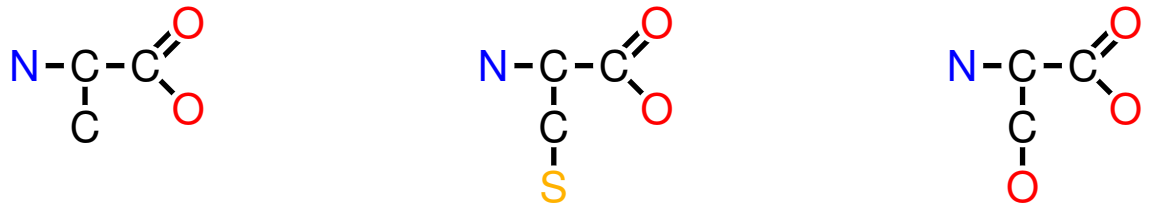


- Perfect extensions of item sets are defined by a very simple criterion. Why is the same criterion necessary, but not sufficient for graphs? What criterion is needed instead?

- Why do rings cause problems to perfect extensions of graphs? With what additional conditions can these problems be handled? Are these conditions necessary?

Frequent (Sub)Graph Mining

- Why is it easy to cut the search tree branches "to the right" of a perfect extension, but difficult to cut those "to the left" of a perfect extension?
- Find the perfect extensions of the fragment `C-C` and of the fragment `N-C` in the graph database that consists of the following three molecules:



- Why is it not a good idea to use the number of occurrences/subgraph isomorphisms as the support measure for mining frequent subgraphs of a single graph?
- How can we define support for mining frequent subgraphs in a single graph?
- What is the main disadvantage of support measures based on overlap graphs?
- Is there a support measure for finding item sets in a single graph that does not use an overlap graph? How is it defined? What are its advantages/disadvantages?