Data Mining 2: Frequent Pattern Mining Summer 2017
Christian Borgelt from 18.07.2017

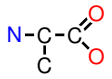
Exercise Sheet 6

Exercise 33 Canonical Code Words

a) What are rightmost path extensions? What are maximum source extensions?

b) Do rightmost path/maximum source extensions always yield canonical code words?

c) Find the canonical code word, based on a depth-first search spanning tree, for cyclin, that is, for the molecule:



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

d) Find the canonical code word, based on a breadth-first search spanning tree, for cyclin (see part c)! Use the same orders as in part c!

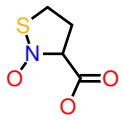
Exercise 34 Extendable Vertices

a) Which vertices/atoms of cyclin are extendable, based on the canonical code word from Exercise 33c), by rightmost path extensions?

b) Which vertices/atoms of cyclin are extendable, based on the canonical code word from Exercise 33d), by maximum source extensions?

Exercise 35 Canonical Code Words

a) Check whether the code word `S2-3-N4-5-C6-OC6-7-CC8-9=OO` 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!

b) Find the canonical code word, based on an (extended) adjacency matrix of which the upper triangle is read row by row, for cyclin (see Exercise 33c). Use the same orders of atoms and bonds as in part a!

Exercise 36 Vertex Signatures

a) What is the purpose of the method of vertex signatures? How can we construct vertex signatures by iteratively splitting equivalence classes?

b) What is an automorphism of a graph? What is the orbit of a vertex?

c) Show that the automorphisms of a graph form a group (algebraic structure)!

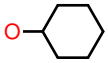
d) Why can we form a canonical code word very efficiently if we know that all vertex labels are pairwise different? Why can we form a canonical code word very efficiently if we know the orbits of all vertices?

e) 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?

Exercise 37 Vertex Signatures

a) Try to find unique labels for the vertices/atoms of cyclin (see Exercise 33c) using the method of vertex signatures! What does the resulting canonical code word look like (use the same orders as above)?

b) Try to find unique labels for the vertices/atoms of phenol, that is, of the graph/molecule shown on the right, using the method of vertex signatures.
Why is it not possible to reduce all equivalence classes to single elements?



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

Exercise 38 Orbits

a) Determine the orbits of the vertices for the following two graphs:

b) Determine the orbits of the vertices for the following two graphs:

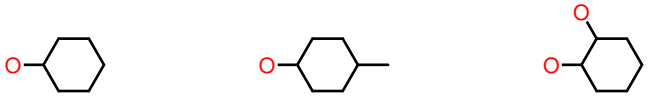
Exercise 39 Perfect Extensions

a) 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?

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

c) 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?

d) 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:



Exercise 40 Mining a Single Graph

a) 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?

b) How can we define support for mining frequent subgraphs in a single graph?

c) What is the main disadvantage of support measures based on overlap graphs?

d) Is there a support measure for finding frequent (sub)graphs in a single graph that does not use an overlap graph? How is it defined? What are its advantages/disadvantages?