Mining more complex patterns: frequent subgraphs

Christian Borgelt
(Jiří Kléma)

Department of Cybernetics,
Czech Technical University in Prague

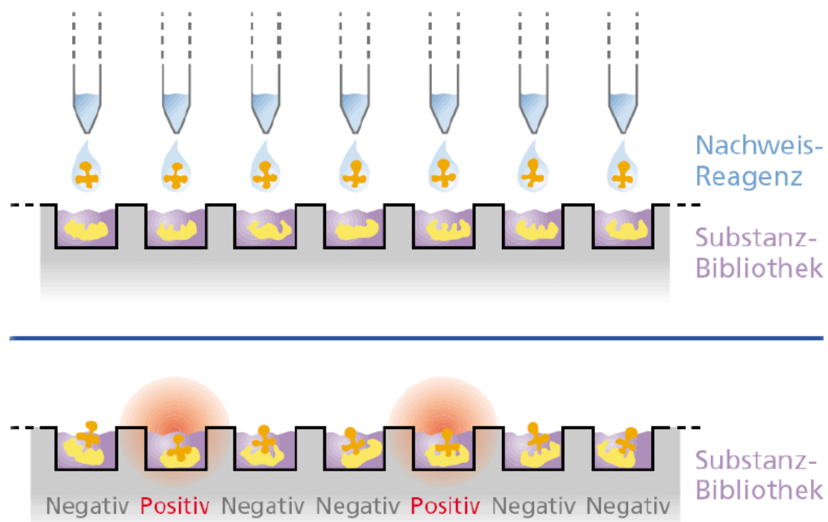
http://ida.felk.cvut.cz

Outline

- Motivation for frequent subgraph mining
  - applications, variance in tasks,
- necessary graph terms
  - isomorphic subgraph,
  - frequent, closed a maximal subgraph,
- subgraph space search
  - code words,
  - canonical code words,
  - how do they speed up search?
- summary
  - the issues covered,
  - the issues not covered
    (extensions for molecules, trees, single graph only, fragment repository).

Frequent subgraphs – illustration 1: molecular fragments

- acceleration of **drug development**,
- ex.: protection of human CEM cells against an HIV infection (public data),
  - high-throughput screening of chemical compounds (37,171 substances tested)
    - 325 confirmed active (100% protection against infection),
    - 877 moderately active (50-99% protection against infection),
    - others confirmed inactive (<50% protection against infection),
  - task: why some compounds active and others not?, where to aim future screening?

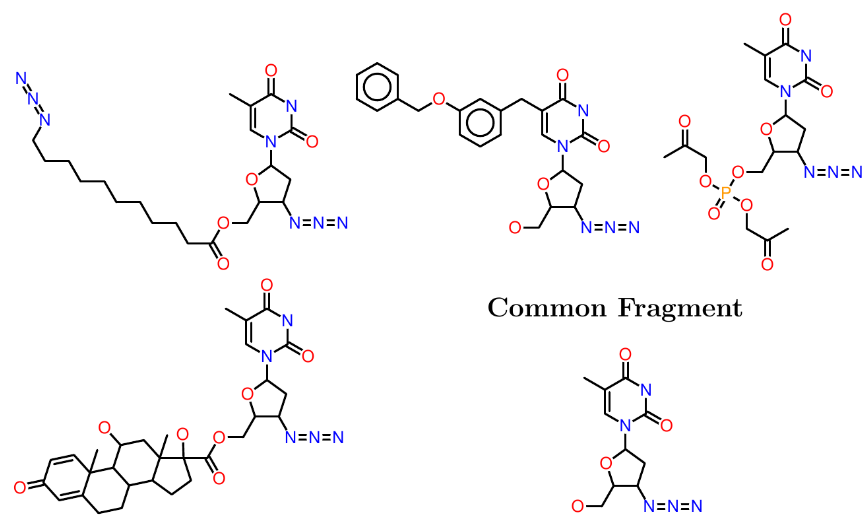


Frequent subgraphs – illustration 1: molecular fragments

- search for fragments common for the active substances
  - find molecular substructures that frequently appear in active substances,
  - frequent active patterns = subgraphs,
- search for discriminative patterns
  - we add the requirement that patterns appear only rarely in the inactive molecules,
  - where to aim future tests? what is the most promising pharmacophore, i.e., drug candidate?

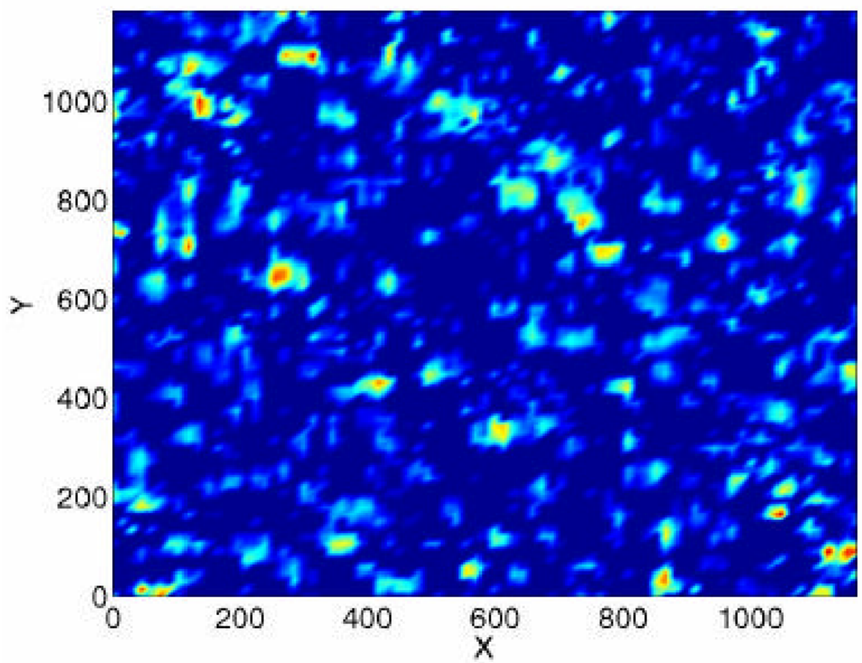
Excerpt from the NCI DTP HIV Antiviral Screen data set (SMILES format):

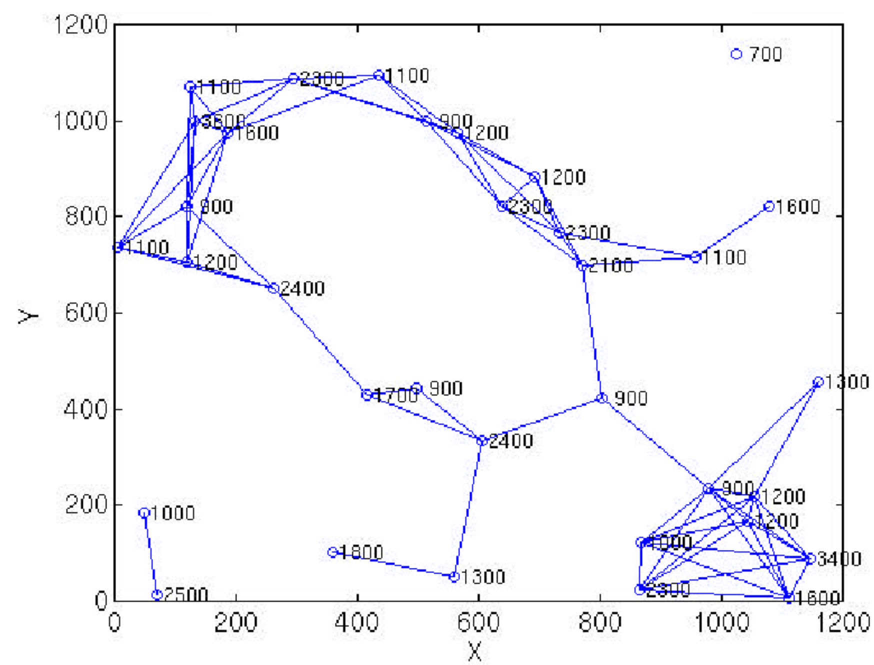
```
737,  0,CN(C)C1=[S+][Zn]2(S1)SC(=[S+]2)N(C)C
2018, 0,N#CC(=CC1=CC=CC=C1)C2=CC=CC=C2
19110,0,OC1=C2N=C(NC3=CC=CC=C3)SC2=NC=N1
20625,2,NC(=N)NC1=C(SSC2=C(NC(N)=N)C=CC=C2)C=CC=C1.OS(O)(=O)=O
22318,0,CCCCN(CCCC)C1=[S+][Cu]2(S1)SC(=[S+]2)N(CCCC)CCCC
24479,0,C[N+](C)(C)C1=CC2=C(NC3=CC=CC=C3S2)N=N1
50848,2,CC1=C2C=CC=CC2=N[C-](CSC3=CC=CC=C3)[N+]1=O
51342,0,OC1=C2C=NC(=NC2=C(O)N=N1)NC3=CC=C(Cl)C=C3
55721,0,NC1=NC(=C(N=O)C(=N1)O)NC2=CC(=C(Cl)C=C2)Cl
55917,0,O=C(N1CCCC[CH]1C2=CC=CN=C2)C3=CC=CC=C3
64054,2,CC1=C(SC[C-]2N=C3C=CC=CC3=C(C)[N+]2=O)C=CC=C1
64055,1,CC1=CC=CC(=C1)SC[C-]2N=C3C=CC=CC3=C(C)[N+]2=O
64057,2,CC1=C2C=CC=CC2=N[C-](CSC3=NC4=CC=CC=C4S3)[N+]1=O
66151,0,[O-][N+](=O)C1=CC2=C(C=NN=C2C=C1)N3CC3
...
```



Frequent subgraphs – illustration 2: gas and fluid dynamics

- measurements: size, velocity and locality of vortices (vortex = whirl = spinning motion),
- graph representation: vortex = vertex (node), proximity = edge length,
- often frequent patterns that e.g., appear shortly before anomalies
  – meteorology, aerodynamics, hydraulics.





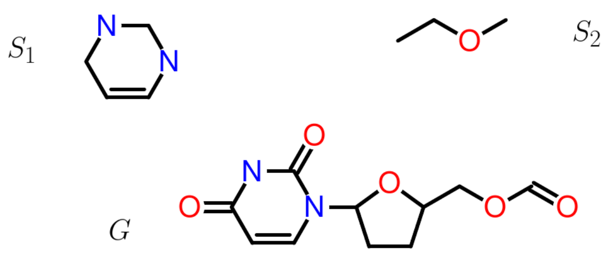
Graphs: basic terms

- Attribute (label) set $A = \{a_1, \ldots, a_m\}$,
  - attribute examples for molecules:
    chemical element, charge, bond type (single, double, triple, aromatic),
- labeled (attributed) graph is a triple $G = (V, E, \ell)$, where
  - V is the set of vertices,
  - $E \subseteq V \times V - \{(v, v) \mid v \in V\}$ is the set of edges, and
  - $\ell : V \cup E \rightarrow A$ assigns labels from the set A to vertices and edges,
  - G is undirected and simple (contains no loops, no multiple edges),
  - several vertices and edges may have the same attribute/label,
- **subgraph** $S \subseteq G$
  - informally: omit some vertices and their incident edges
    (full, induced subgraph),
  - when omitting more edges, it is a subgraph without the characteristic full,
  - proper subgraph $S \subset G$.

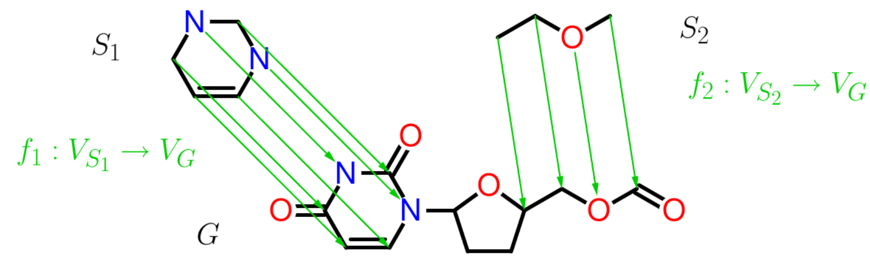
Graphs: basic terms

- connected component of a graph
  - connected subgraph, any larger subgraph that contains it is not connected,
- a vertex of a graph is called
  - isolated – it is not incident to any edge,
  - leaf – it is incident to exactly one edge,
- an edge of a graph is called
  - bridge – removing it increases the number of connected components of the graph,
  - proper bridge – if it is a bridge and not incident to a leaf (all other bridges are leaf bridges),
- graphs $S = (V_S, E_S, \ell_S)$ a $G = (V_G, E_G, \ell_G)$ are **isomorphic** ($S \equiv G$), iff
  - $\exists f : V_S \rightarrow V_G$ (bijection) such that:
    - $\ell_S(v) = \ell_G(f(v))$
    - $(x, y) \in E_S \Leftrightarrow (f(x), f(y)) \in E_G \quad \wedge \quad \ell_S((u, v)) = \ell_G((f(u), f(v)))$,
- graph S is an **isomorphic subgraph** of G (S occurs in G, $S \sqsubseteq G$), iff
  - f limited on injective functions $\forall v \in V_S$,
  - $S \sqsubseteq G \wedge G \sqsubseteq S \Leftrightarrow S \equiv G$.
- testing whether a subgraph isomorphism exists between given graphs S and G is NP-complete!

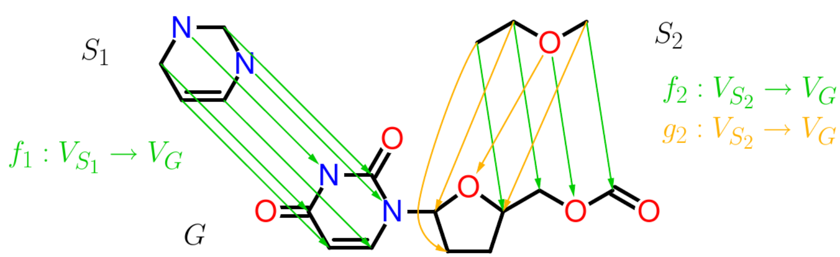
Subgraph Isomorphism: examples



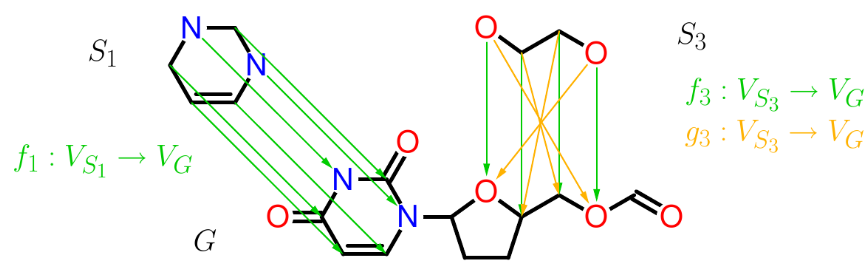
S_1 and S_2 subgraphs of G



isomorphisms f_1 and f_2 exist
(mapping preserves vertex and edge labels)



S_2 has more occurrences in G



S_3 possesses an automorphism
(S_3 non-identically maps to itself)
(S_3 has more occurrences at the same location in G)

Graphs: definition of support

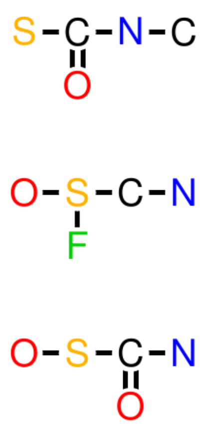
- G covers S iff
  - $S \sqsubseteq G$ (S is contained in G),
- G properly covers S (S is properly contained in G) iff
  - $S \sqsubset G \Leftrightarrow S \not\equiv G \wedge S \sqsubseteq G$,
- having a vector of graphs $\mathcal{G} = \{G_1, \dots, G_n\}$, the cover of S wrt $\mathcal{G}$ is
  - $K_{\mathcal{G}}(S) = \{k \in \{1, \dots, n\} \mid S \sqsubseteq G_k\}$,
  - the index set of the database graphs that cover S,
- (absolute) support S wrt $\mathcal{G}$ is a natural number
  - $s_{\mathcal{G}}(S) = |K_{\mathcal{G}}(S)|$,
  - the number of graphs that cover S
    (more occurrences in one graph are not concerned),
- the frequent subgraph (fragment) S wrt $\mathcal{G}$ is each subgraph that
  - $s_{\mathcal{G}}(S) \geq s_{min}$.

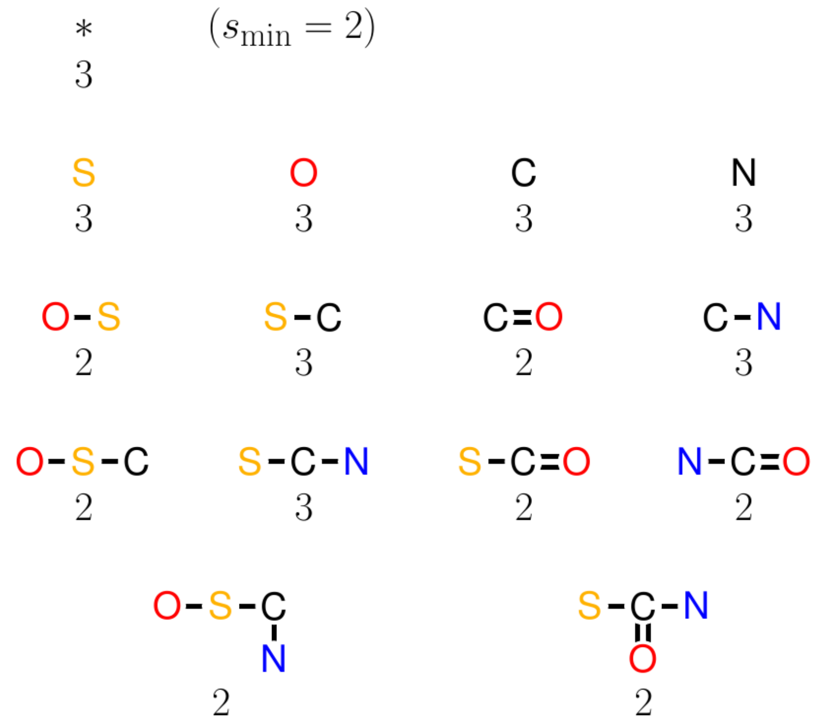
Frequent subgraph mining: definition

- given: graphs $\mathcal{G} = \{G_1, \ldots, G_n\}$ with labels $A = \{a_1, \ldots, a_m\}$ and minimum support s_{min}
- output: the set of frequent (sub)graphs with support meeting the minimum threshold
  - $F_{\mathcal{G}}(s_{\min}) = \{S \mid s_{\mathcal{G}}(S) \geq s_{\min}\}$,
- common constraint
  - connected subgraphs only,
- main problem
  - to avoid **redundancy** when searching
    - canonical representation of (sub)graphs,
    - partial order of (sub)graph space,
    - efficient pruning of the searched subgraph space,
    - fragment repository for processed graphs.
- APRIORI property generalized for graphs
  - **All subgraphs of a frequent (sub)graph are frequent.** (anti-monotone)
  - **No supergraph of an infrequent (sub)graph can be frequent.** (monotone)
  - $\forall S : \forall R \supseteq S : \quad s_{\mathcal{G}}(R) \leq s_{\mathcal{G}}(S)$.

Frequent subgraphs: example

- $\mathcal{G}$ contains three molecules, minimum support $s_{min} = 2$,
- 15 frequent subgraphs exist,
- empty graph is properly contained in all graphs by definition.



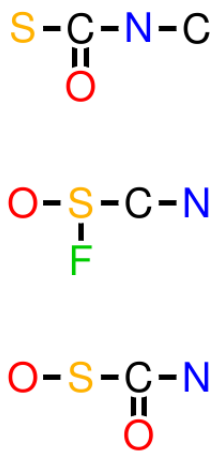


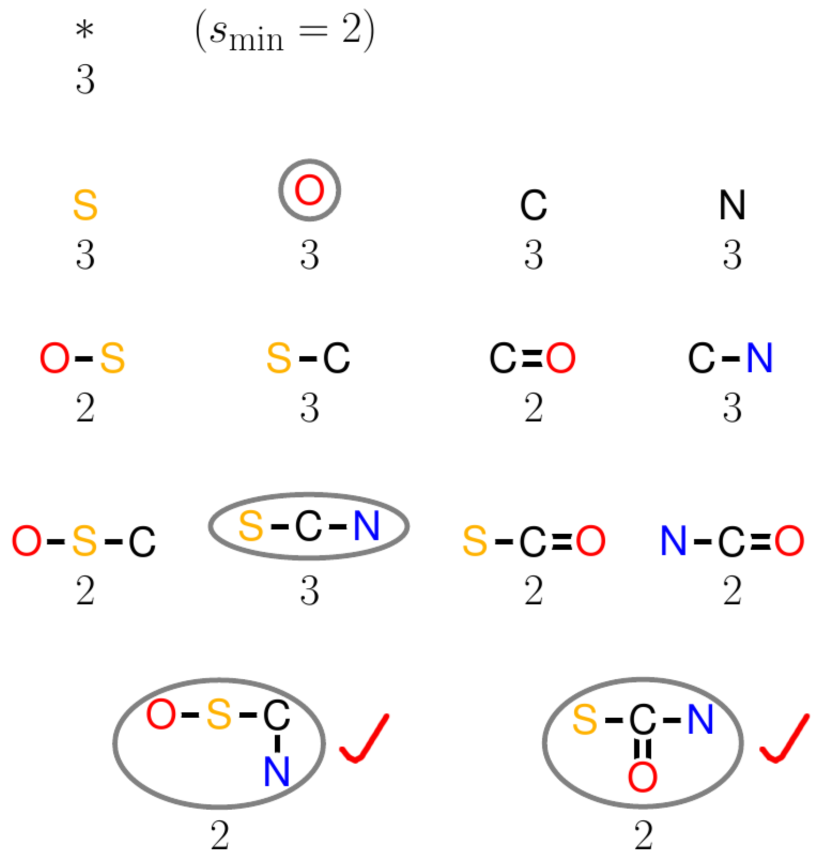
Types of frequent subgraphs – closed and maximal

- **maximal** subgraph
  - is frequent but none of its proper supergraphs is frequent, the set of maximal (sub)graphs: $M_{\mathcal{G}}(s_{\min}) = \{S \mid s_{\mathcal{G}}(S) \geq s_{\min} \land \forall R \supset S : s_{\mathcal{G}}(R) < s_{\min}\}$,
  - every frequent (sub)graph has a maximal supergraph,
  - no supergraph of a maximal (sub)graph is frequent,
  - $M_{\mathcal{G}}(s_{\min})$ (and their support) does not preserve knowledge of all support values
    - meaning support values of all frequent subgraphs,
- **closed** subgraph
  - is frequent but none of its proper supergraphs has the same support,
  - the set of closed (sub)graphs: $C_{\mathcal{G}}(s_{\min}) = \{S \mid s_{\mathcal{G}}(S) \geq s_{\min} \land \forall R \supset S : s_{\mathcal{G}}(R) < s_{\mathcal{G}}(S)\}$,
  - every frequent (sub)graph has a closed supergraph (with the identical support),
  - $C_{\mathcal{G}}(s_{\min})$ (and their support) preserves knowledge of all support values,
- relations among graph types
  - every maximal or closed subgraph is automatically frequent,
  - every maximal subgraph is also closed.

Closed and maximal subgraphs: example

- $\mathcal{G}$ contains three molecules, $s_{min} = 2$,
- 4 closed subgraphs exist, 2 of them are maximal too.

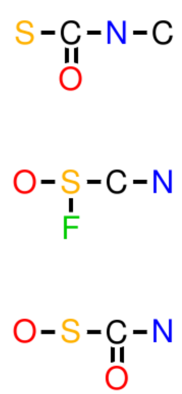


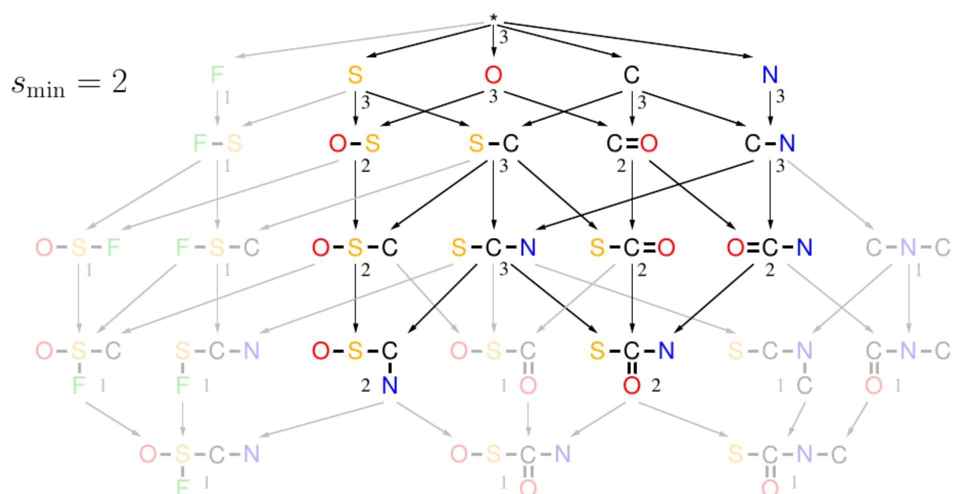


Partially ordered set of subgraphs and its search

- subgraph (isomorphism) relationship defines a partial order on subgraphs
  - Hasse diagram exists, the empty graph makes its infimum, no natural supremum exists,
  - diagram can be completely searched top-down from the empty graph,
  - branching factor is large, the depth-first search is usually preferable.
- the main problem
  - a (sub)graph can be grown in several different ways,
  - diagram must be turned into a tree – each subgraph has a unique parent.

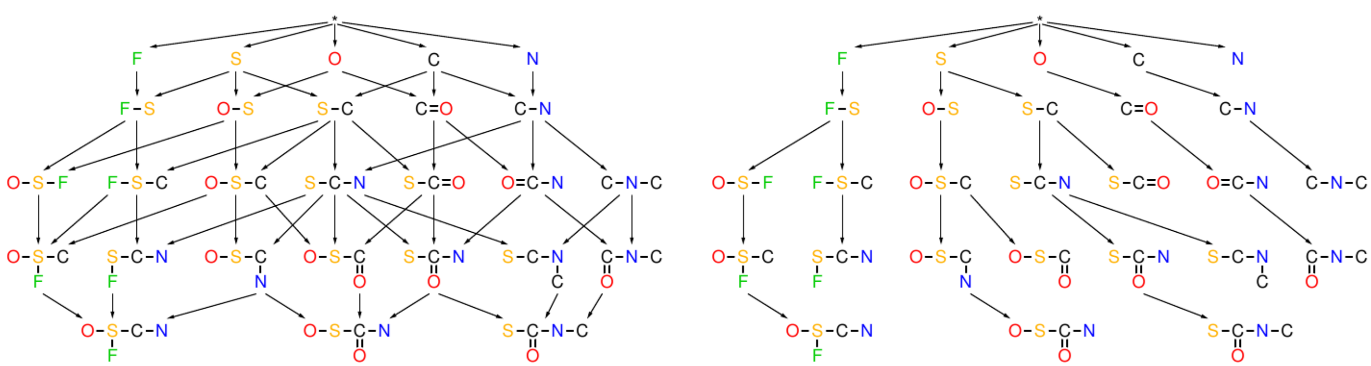
example molecules:





Partially ordered set of subgraphs and its search

- Searching for frequent (sub)graphs
  - (subgraphs with a unique parent),
- base loop
  - traverse all possible vertex attributes (their unique parent is the empty graph).
  - recursively process all vertex attributes that are frequent,
- Recursive processing for a given frequent (sub)graph S
  - generate all extensions R of S by an edge or by an edge and a vertex
    - $*$ edge addition $(u, v) \notin E_S$, $u \in V_S \vee v \in V_S$,
    - $*$ if $u \notin V_S \vee v \notin V_S$, the missing node is added too,
  - S must be the unique parent of R,
  - if R is frequent, further extend it, otherwise STOP.



Assigning unique parents

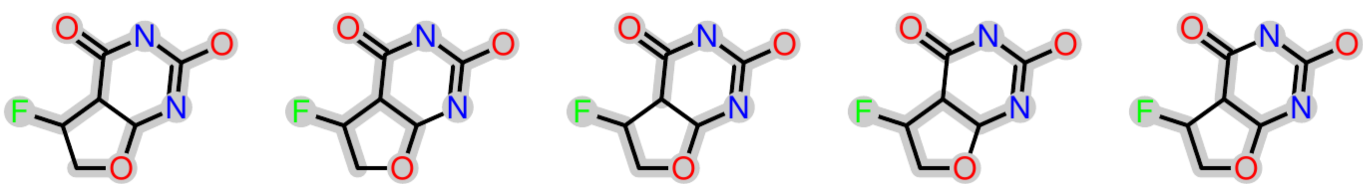
- How can we formally define the set of parents of subgraph S?
  - subgraphs that contain exactly one edge less than the subgraph S,
  - in other words, all the maximal proper subgraphs,
- **canonical (unique) parent** $p_c(S)$ of subgraph S
  - an order on the edges of the (sub)graph S must be given before,
  - let e^* be the last edge in the order that is not a proper bridge in S,
    - then $p_c(S)$ is the graph S without the edge e^*,
    - if e^* is a leaf bridge, we also have to remove the created isolated node,
  - if e^* is the only edge of S, we also need an order of the nodes,
- in order to define an order of the edges we will rely on a **canonical form of (sub)graphs**
  - each (sub)graph is described by a code word,
  - it unambiguously identifies the (sub)graph (up to automorphism = symmetries),
  - having multiple code words per graph
    - one of them is (lexicographically) singled out as the canonical code word.

Canonical graph representation

- Basic idea
  - the characters of the code word describe the edges of the graph,
  - vertex labels need not be unique, they must be endowed with unique labels (numbers),
- usual requirement on canonical form
  - prefix property – every prefix of a canonical code word is a canonical code word itself,
  - when the last edge e^* is removed, the canonical word of the canonical parent originates,
- assuming the prefix property holds, search algorithm takes the canonical word of a parent and
  - generates all possible extensions by an edge (and maybe a vertex),
  - checks whether the extended code words are the canonical code words,
  - consequence: easy and non-redundant access to children,
- the most common canonical forms
  - spanning tree,
  - adjacency matrix.

Canonical forms based on spanning trees

- Graph code word is created when constructing a **spanning tree** of the graph
  - numbering the vertices in the order in which they are visited,
  - describing each edge by the numbers of incident vertices, the edge and vertex labels,
  - listing the edge descriptions in the order in which the edges are visited (edges closing cycles may need special treatment),
- the most common ways of constructing a spanning tree are
  - search: depth-first $\times$ breath-first,
  - both approaches ask for their own way of code word construction,
- one graph may be described by a large number of code words
  - a graph has multiple spanning trees (initial vertex, branching options),
  - how to find the lexicographically smallest = canonical word quickly?
  - prefix property holds, edges listed in the order they are visited during the s.tree construction,
  - one only needs to verify that the extension is also canonical.



Canonical forms based on spanning trees

- A precedence order of labels is introduced
  - due to efficiency, frequency of labels shall be concerned,
  - vertex labels are recommended to be in ascending order,
- Regular expressions for code words
  - depth-first: $a\,(i_d\,\underline{i_s}\,b\,a)^m$,
    (exception: indices in decreasing order)
  - breadth-first: $a\,(i_s\,b\,a\,i_d)^m$ (or $a\,(i_s\,i_d\,b\,a)^m$),
  - meaning of symbols:

    n the number of vertices of the graph,

    m the number of edges of the graph,

    i_s index of the source vertex of an edge, $i_s \in \{0, \ldots, n-1\}$,

    i_d index of the destination vertex of an edge, $i_d \in \{0, \ldots, n-1\}$,

    a the attribute of a vertex,

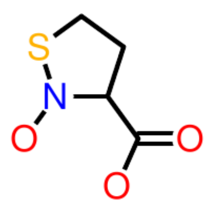
    b the attribute of an edge.

Canonical spanning tree: example

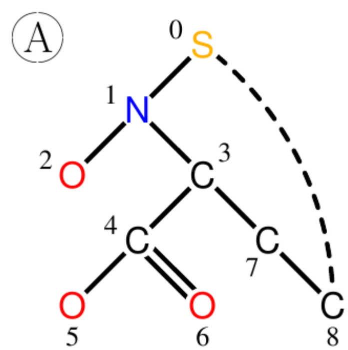
- Order of labels
  - elements (vertices): $S \prec N \prec O \prec C$, bonds (edges): $- \prec =$,
- Code words

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

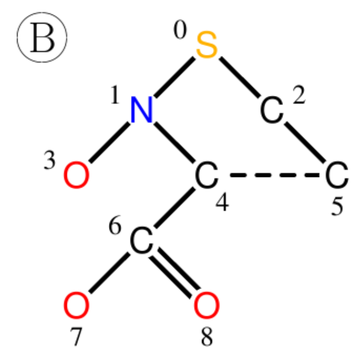
B: `S 0-N1 0-C2 1-O3 1-C4 2-C5 4-C5 4-C6 6-O7 6=O8`



example molecule



depth-first



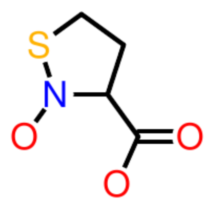
breadth-first

Recursive checking for canonical form

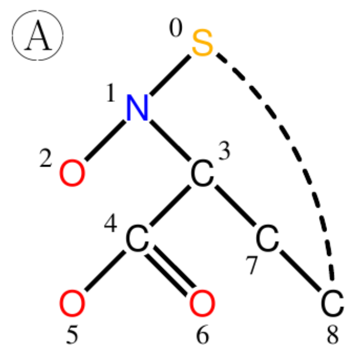
- traverse all vertices with a label no less than the current spanning tree root vertex,
- recursively add edges, compare the code word with the checked one (potentially canonical)
  - if the new edge description is larger, the edge can be skipped (backtrack),
  - if the new edge description is smaller, the checked code word is not canonical,
  - if the new edge description is equal, the rest of the code word is processed recursively.

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

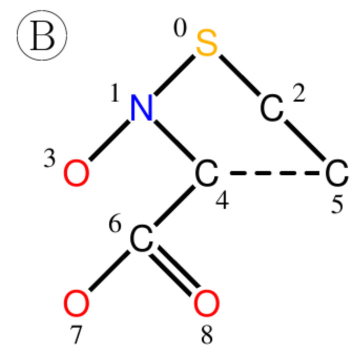
B: `S 0-N1 0-C2 1-O3 1-C4 2-C5 4-C5 4-C6 6-O7 6=O8`



example molecule



depth-first



breadth-first

Restricted extensions of canonical words

- Principle of recursive search of subgraph tree
  - generate all possible extensions of a given canonical code word (of a frequent parent),
  - extensions adds the description of an edge that extends the described (sub)graph,
  - prefix representation: the edge description added at the end of code word,
  - canonical form is checked, if met then proceed recursively, otherwise the word is discarded,
- how to verify efficiently whether a word is canonical?
  - in general, a lex. smaller word with the same root vertex needs to be found,
  - simple local rules can be found, the rules reject extensions locally = immediately
    - only certain vertices are extendable,
    - certain cycles cannot be closed,
    - they represent necessary canonicity conditions, not sufficient.

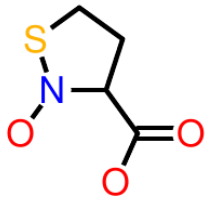
Restricted extensions of canonical words

- Depth-first: **rightmost path extension**
  - extendable vertices
    - must be on the rightmost path of the spanning tree
      (other vertices cannot be extended in the given search-tree branch),
    - if the source vertex of the new edge is not a leaf, the edge description must not precede the description of the downward edge on the path
      (the edge attribute must be no less than the edge attribute of the downward edge,
      if it is equal, the attribute of its destination vertex must be no less than the attribute of the downward edge's destination vertex),
  - edges closing cycles
    - must start at an extendable vertex,
    - must lead to the rightmost leaf
      (a subgraph has only one vertex meeting the condition),
    - the index of the source vertex must precede the index of the source vertex of any edge already incident to the rightmost leaf.

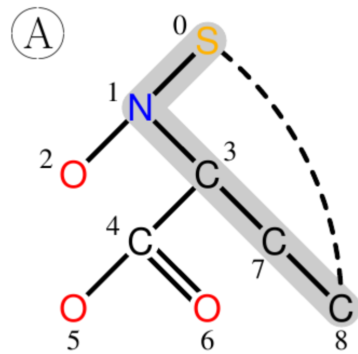
Restricted extensions of canonical words

- Breadth-first: **maximum source extension**
  - extendable vertices
    - cannot have a lower index than the maximum source index of edges already used,
    - if the source of the new edge is the one having the maximum source index, edge precedence must be checked (see depth-first option),
  - edges closing cycles
    - must start at an extendable vertex,
    - must lead forward, that is, to a vertex having a larger index than the extended vertex.

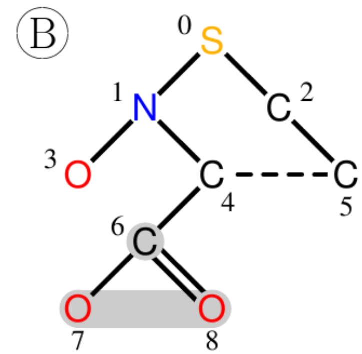
Restricted extensions: examples



example molecule



depth-first



breadth-first

- Extendability
  - vertices: ad A: 0, 1, 3, 7, 8, ad B: 6, 7, 8,
  - edges closing cycles: ad A: none, ad B: the edge between 7 and 8,
- Extension: attach a single bond carbon atom at the leftmost oxygen atom

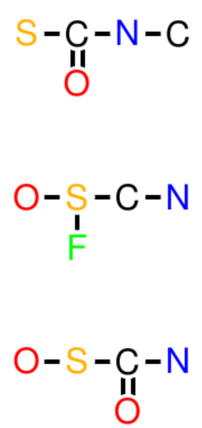
A: `S 10-N 21-O 31-C 43-C 54-O 64=O 73-C 87-C 80-C 92-C`
`S 10-N 21-O 32-C ···`

B: `S 0-N1 0-C2 1-O3 1-C4 2-C5 4-C5 4-C6 6-O7 6=O8 3-C9`
`S 0-N1 0-C2 1-O3 1-C4 2-C5 3-C6 ···`

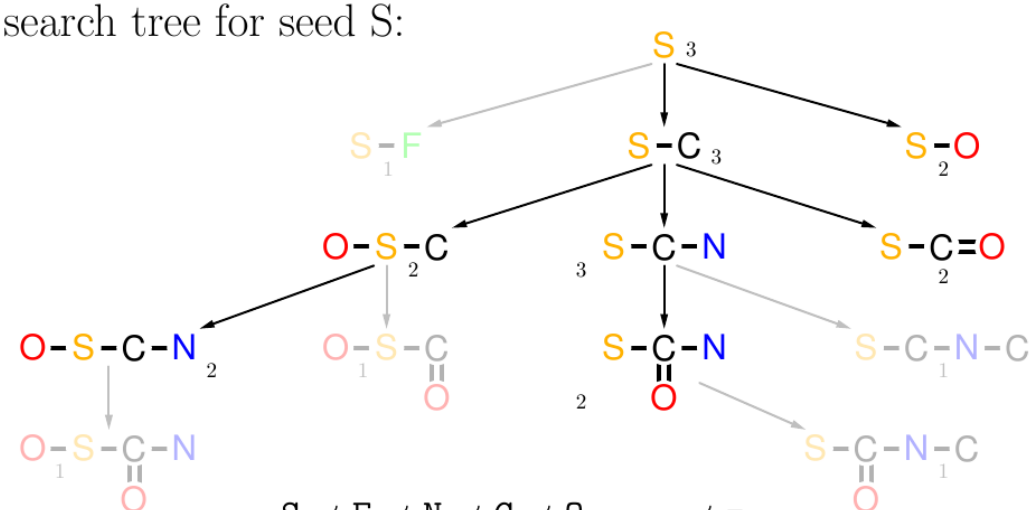
Frequent subgraphs with canonical form: example search tree

- Start with a single seed vertex,
- add an edge (and maybe a vertex) in each step (restricted extensions),
- determine the support and prune infrequent (sub)graphs (outside the code word space),
- check for canonical form and prune (sub)graphs with non-canonical code words.

example molecules:



search tree for seed S:

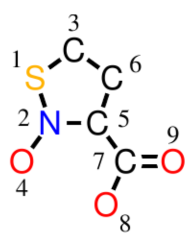


S ≺ F ≺ N ≺ C ≺ O   - ≺ =

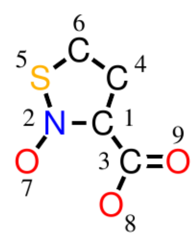
breadth-first search canonical form

Canonical forms based on adjacency matrices

- **Adjacency matrix**
  - common graph representation,
  - graph G with n vertices is captured by a $n \times n$ matrix $\mathbf{A} = (a_{ij})$,
  - $a_{ij} = 1 \Leftrightarrow$ an edge between the vertices with numbers i and j, 0 otherwise,
  - however not unique, different vertex numberings lead to different matrices.
- **Extended adjacency matrix**
  - for a labeled graph G (with vertex and edge attributes),
  - there is an additional column containing the vertex labels,
  - a_{ij} either contains the edge label or the special empty label $a_{ij} = \times$.



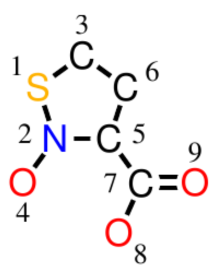
		1	2	3	4	5	6	7	8	9
1	S	×	–	–	×	×	×	×	×	×
2	N	–	×	×	–	–	×	×	×	×
3	C	–	×	×	×	×	–	×	×	×
4	O	×	–	×	×	×	×	×	×	×
5	C	×	–	×	×	×	–	–	×	×
6	C	×	×	–	×	–	×	×	×	×
7	C	×	×	×	×	–	×	×	–	=
8	O	×	×	×	×	×	–	–	×	×
9	O	×	×	×	×	×	×	=	×	×



		1	2	3	4	5	6	7	8	9
1	C	×	–	–	–	×	×	×	×	×
2	N	–	×	×	×	–	×	–	×	×
3	C	–	×	×	×	×	×	×	–	=
4	C	–	×	×	×	×	–	×	×	×
5	S	×	–	×	×	×	–	×	×	×
6	C	×	×	×	–	–	×	×	×	×
7	O	×	–	×	×	×	×	×	×	×
8	O	×	×	–	×	×	×	×	×	×
9	O	×	×	=	×	×	×	×	×	×

From adjacency matrices to code words

- by simply listing its elements row by row,
- the matrix is symmetric for undirected graphs – it suffices to list the elements of the upper triangle,
- condensed/reduced code word representation
  - only existing edges are listed,
  - column identifiers need to be added,
  - suitable for matrices.



Regular expression (non-terminals): $(a\ (\ i_c\ b\ )^*\ )^n$

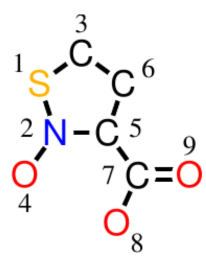
		1	2	3	4	5	6	7	8	9
1	S	×	−	−	×	×	×	×	×	×
2	N	−	×	×	−	−	×	×	×	×
3	C	−	×	×	×	×	−	×	×	×
4	O	×	−	×	×	×	×	×	×	×
5	C	×	−	×	×	×	−	−	×	×
6	C	×	×	−	×	−	×	×	×	×
7	C	×	×	×	×	−	×	×	−	=
8	O	×	×	×	×	×	−	−	×	×
9	O	×	×	×	×	×	×	=	×	×

code word:

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

Canonical extended adjacency matrices

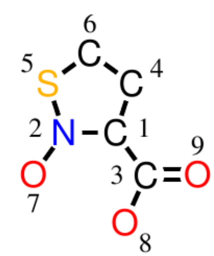
- the key issue is to find the canonical code word
  - it stems from lexicographical order of labels
    vertices: $S \prec N \prec O \prec C$, edges: $- \prec =$,
  - canonical code word is lexikographically smallest,
  - adjacency matrices allow for a much larger number of code words then spanning trees,
  - the row-wise listing restricted to the upper triangle has the advantage of prefix property.
- example of canonical and non-canonical code word



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

$<$

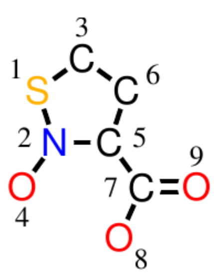
`C 2 - 3 - 4 - N 5 - 7 - C 8 - 9 = C 6 - S 6 - C O O O`



- trivial observations
  - one of the vertices with minimal label must have the index 1,
  - edges with different labels define the order of further vertices unambiguously,
  - the easiest construction with unique labels, backtracking needed otherwise.

Canonical extended adjacency matrices

- how to distinguish the vertices and edges with the same label?
  - let us introduce a **vertex signature** = local code word,
  - it captures the neighborhood structure of a vertex,
  - the structure is extended until no signature pair matches,
  - we iteratively split vertex **equivalence classes**.



vertex	signature
1	S
2	N
4	O
8	O
9	O
3	C
6	C
5	C
7	C

vertex	signature
1	S
2	N
4	O -
8	O -
9	O =
3	C --
6	C --
5	C ---
7	C --=

vertex	signature
1	S
2	N
4	O - N
8	O - C --=
9	O =
3	C -- S C --
6	C -- C -- C ---
5	C ---
7	C --=

Additional issues

- Fragment repository
  - canonical code words represent the dominant approach to redundancy reduction,
  - an alternative is to store already processed subgraphs, they are not processed again,
  - key efficiency issues: memory, fast access (hash),
- extensions for molecules
  - frequent molecular fragments processed en bloc,
  - ring mining, carbon chains and wildcard vertices,
- single graph only
  - distinct definition of support (more complex),
- trees
  - ordered $\times$ unordered, rooted $\times$ unrooted,
  - in general easier than unrestricted graphs.

Frequent subgraphs – summary

- Problem closely related to frequent itemset mining
  - APRIORI property,
  - however, to avoid redundancy during search gets more difficult,
    * larger branching factor,
    * itemsets have no internal structure,
  - non-trivial canonical graph representation
    * guarantees that subgraph support is counted at most once (additional necessary condition is parental support),
    * choice of representation related with choice of searching algorithm,
    * prefix property allows for early rejection of non-canonical candidates,
  - two canonical forms were introduced
    * spanning trees,
    * adjacency matrices,
- demo: Molecular Substructure Miner (MOSS),

Recommended reading, lecture resources

:: Reading

- Borgelt: **Frequent Pattern Mining.**
  - this lecture makes a selection of the graph part of Borgelt's course,
  - http://www.borgelt.net/teach/fpm/slides.html.
- Nijssen, Kok: **The Gaston Tool for Frequent Subgraph Mining.**
  - frequently used tool Gaston, application on molecular databases,
  - http://www.liacs.nl/~snijssen/gaston/index.html,
- Yan, Han: **gSpan: Graph-Based Substructure Pattern Mining.**
  - frequently applied tool gSpan,
  - http://www.cs.ucsb.edu/~xyan/software/gSpan.htm.