Supplement
Frequent Subgraph Mining
Molecular Fragment Mining

Molecular Fragment Mining: Overview

- **Motivation: Accelerating Drug Development**
  - Phases of Drug Development: Pre-clinical and Clinical
  - Data Gathering by High-Throughput Screening:
    Building Molecular Databases with Activity Information
- **Data Mining in Molecular Databases**
  - Finding Frequent and Discriminative Molecular Fragments
  - Avoiding Redundant Search with Canonical Forms of Graphs
  - Restricted Extensions (rightmost vs. maximum source)
  - Other Pruning Techniques: Equivalent Siblings, Perfect Extensions
  - Enhancements: Ring Extensions, Carbon Chains, Wildcard Atoms
- **Results on Databases of the National Cancer Institute**
- **Summary**

Accelerating Drug Development

- Developing a new drug can take **10 to 12 years**
  (from the choice of the target to the introduction into the market).
- In recent years the **duration** of the drug development processes **increased** continuously; at the same the **number** of substances under development **has gone down** drastically.
- Due to high investments pharmaceutical companies must secure their market position and competitiveness by only a **few, highly successful drugs.**
- As a consequence the chances for the development of drugs for target groups
  - with **rare diseases** or
  - with **special diseases in developing countries**

  are considerably reduced.
- A significant **reduction of the development time** could mitigate this trend or even reverse it.

(Source: Bundesministerium für Bildung und Forschung, Germany)

Phases of Drug Development

- **Discovery and Optimization of Candidate Substances**
  - High-Throughput Screening
  - Lead Discovery and Lead Optimization

- **Pre-clinical Test Series** (tests with animals, ca. 3 years)
  - Fundamental test w.r.t. effectiveness and side effects

- **Clinical Test Series** (tests with humans, ca. 4–6 years)
  - Phase 1: ca. 30–80 healthy humans
    Check for side effects
  - Phase 2: ca. 100–300 humans exhibiting the symptoms of the target disease
    Check for effectiveness
  - Phase 3: up to 3000 healthy and ill humans at least 3 years
    Detailed check of effectiveness and side effects

- **Official Acceptance as a Drug**

Drug Development: Acceleration Potential

- The length of the pre-clinical and clinical tests series can hardly be reduced, since they serve the purpose to ensure the safety of the patients.
- Therefore approaches to speed up the development process usually target the **pre-clinical phase** before the animal tests.
- In particular, it is tried to improve the search for new drug candidates (*lead discovery*) and their optimization (*lead optimization*).

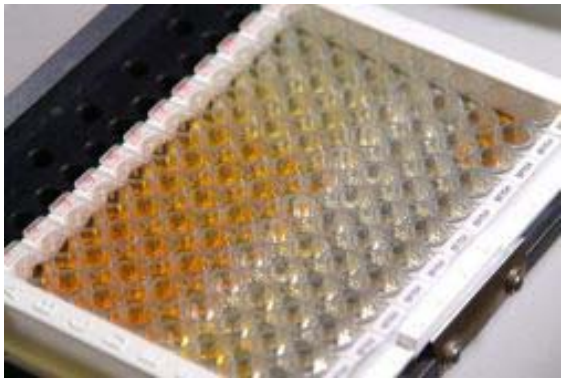
Here Intelligent Data Analysis and Data Mining can help.

One possible approach:

- With high-throughput screening a very large number of substances is tested automatically and their activity is determined.
- The resulting molecular databases are analyzed by trying to find **common substructures** of active substances.

High-Throughput Screening

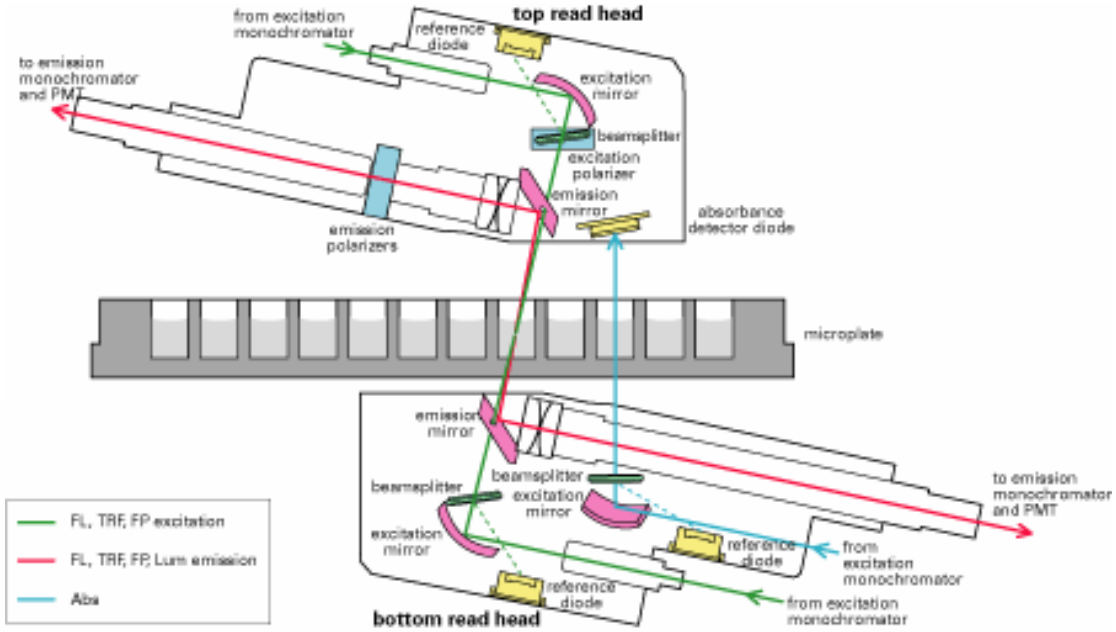
On so-called **micro-plates** proteins/cells are automatically combined with a large variety of chemical compounds.



www.elisa-tek.com
www.arrayit.com
© www.matrixtechcorp.com
www.thermo.com

High-Throughput Screening

The filled micro-plates are then evaluated in **spectrometers** (w.r.t. absorption, fluorescence, luminescence, polarization etc).



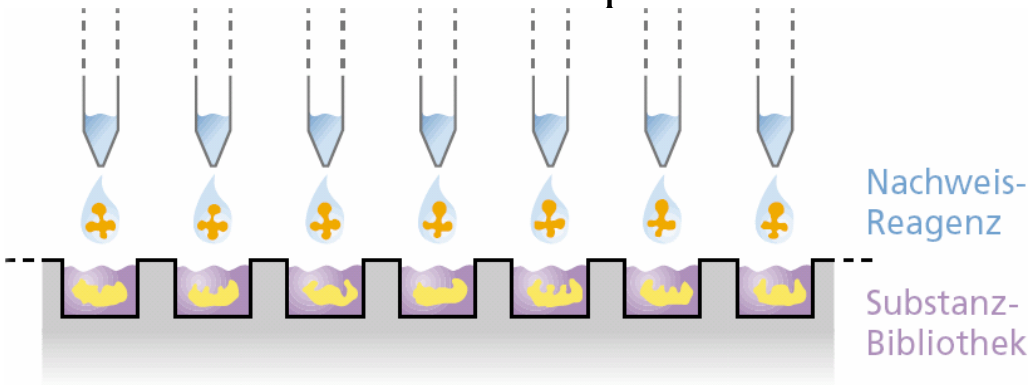
© www.moleculardevices.com www.biotek.com

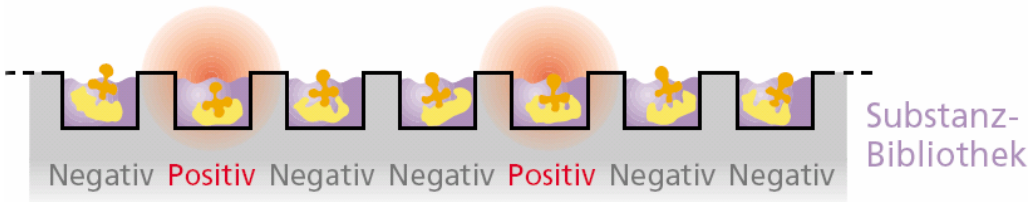
High-Throughput Screening

After the measurement the substances are classified as **active** or **inactive.**

By analyzing the results one

tries to understand the dependence between molecular structure and activity.





orithms are used:

- feature selection methods
- decision trees
- neural networks etc.

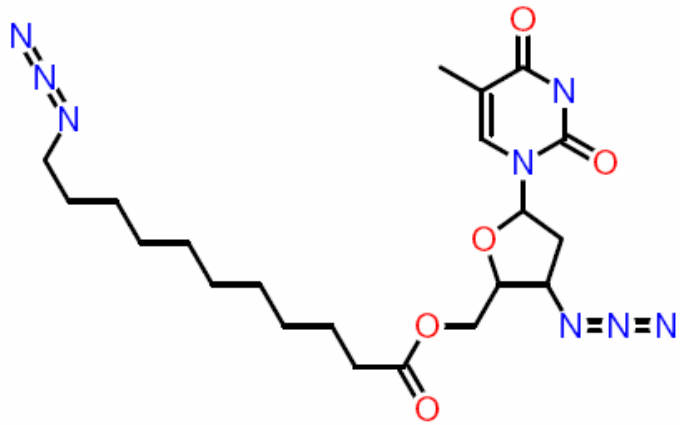
Figure © Christof Fattinger, Hoffmann-LaRoche, Basel

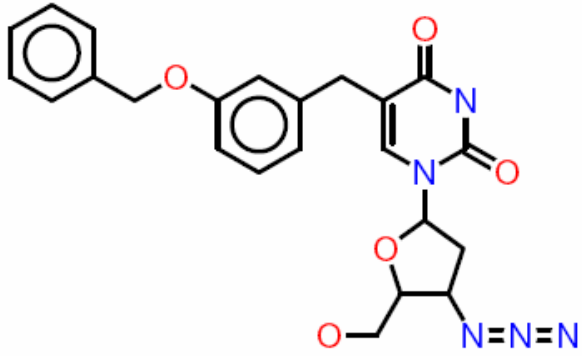
Example: NCI DTP HIV Antiviral Screen

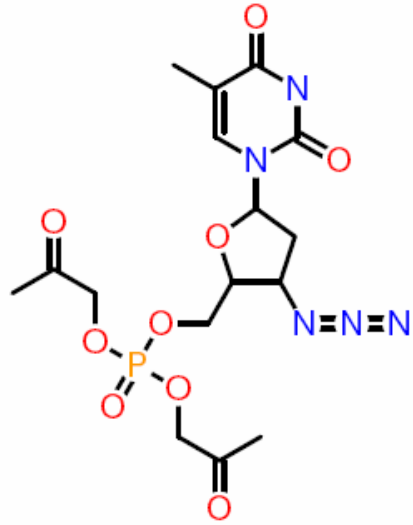
- Among other data sets, the National Cancer Institute (NCI) has made the **DTP HIV Antiviral Screen Data Set** publicly available.
- A large number of chemical compounds where tested whether they protect human CEM cells against an HIV-1 infection.
- Substances that provided 50% protection were retested.
- Substances that reproducibly provided 100% protection are listed as **"confirmed active"** (CA).
- Substances that reproducibly provided at least 50% protection are listed as **"moderately active"** (CM).
- All other substances are listed as **"confirmed inactive"** (CI).
- 325 **CA**, 877 **CM**, 35 969 **CI** (total: 37 171 substances)

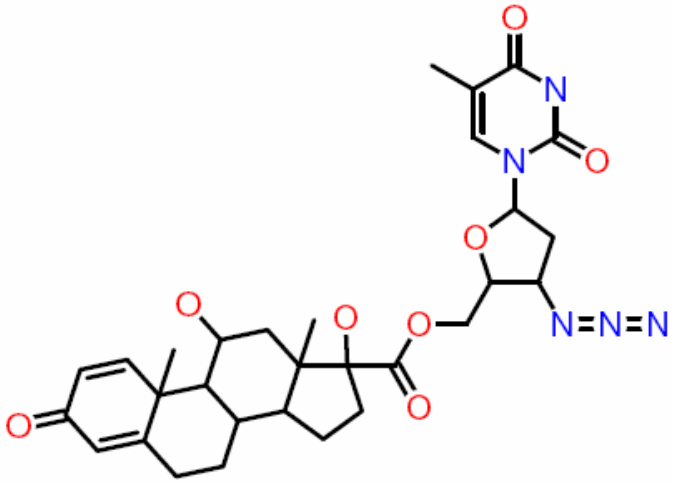
Finding Common Molecular Substructures

Some Molecules from NCI HIV Database

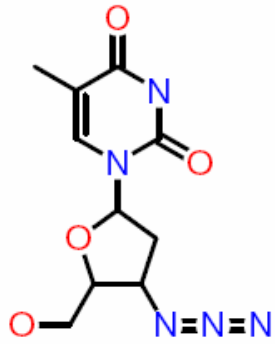








Common Fragment



Form of the Input Data

Excerpt from the DTP HIV Antiviral Screen Data Set:

```
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
```

identification number, activity (2: CA, 1: CM, 0: CI), molecule description in SMILES notation

Input Format: Smiles Notation and SLN

Smiles Notation: (z.B. Daylight, Inc.)

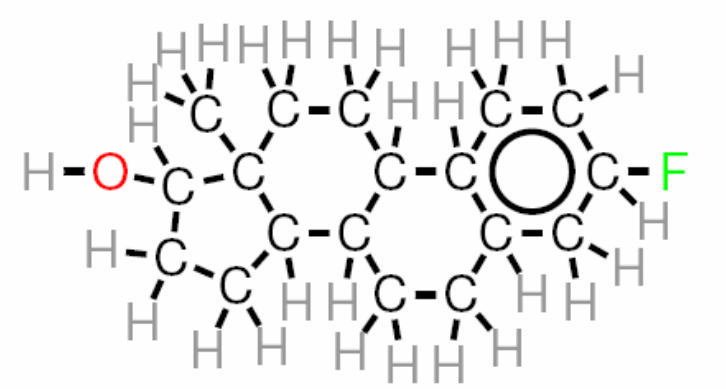
```
c1:c:c(:c:c2:c:1-C1-C(-C-C-2)-C2-C(-C-C-1)(-C(-C-C-2)-O)-C)-F
```

SLN (Sybyl Line Notation): (Tripos, Inc.)

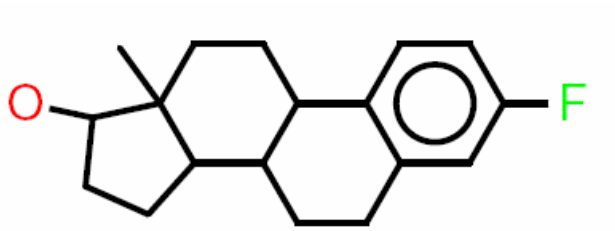
```
C[1]H:CH:C(:CH:C[8]:C:@1C[10]HCH(CH2CH2@8)C[20]HC(CH2CH2@10)
(CH(CH2CH2@20)OH)CH3)F
```

Represented Molecule:

Full Representation



Simplified Representation



Frequent Graph Mining: General Approach

- Finding frequent item sets means to find
  sets of items that are contained in many transactions.
- Finding frequent substructures means to find
  graph fragments that are contained in many graphs
  in a given database of attributed graphs (user specifies minimum support).
- But: Graph structure of nodes and edges has to be taken into account.
  → Search semi-lattice of graph structures instead of subset lattice.
- Commonly the search is restricted to **connected substructures.**

Molecule Representation:

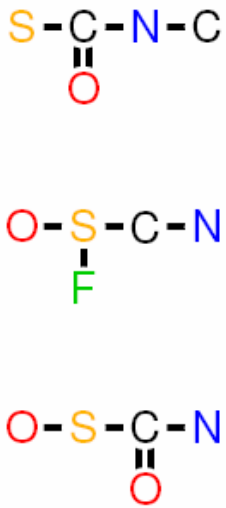
- A molecule can be represented as an **attributed undirected graph.**
- **Atom attributes:** atom type (chemical element), charge, aromatic ring flag
- **Bond attributes:** bond type (single, double, triple, aromatic)

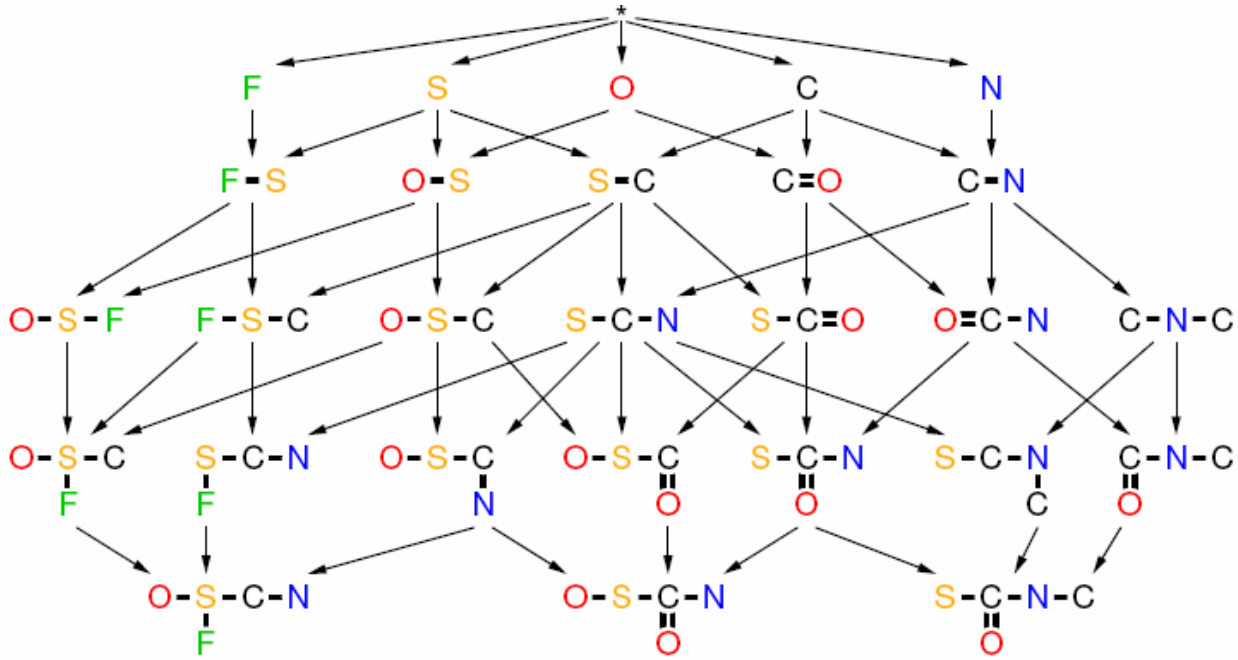
Semi-Lattice of Subgraphs

Semi-lattice ranging from the empty graph to the database graphs.

- The subgraph relationship defines a partial order on subgraphs.
- The empty graph is (formally) contained in all subgraphs.
- There is usually no unique largest graph (→ no complete lattice).

example molecules:



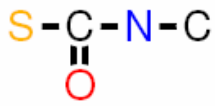


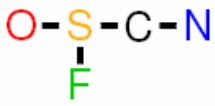
Frequent Subgraphs

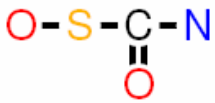
The frequent subgraphs form a sub-semi-lattice at the top.

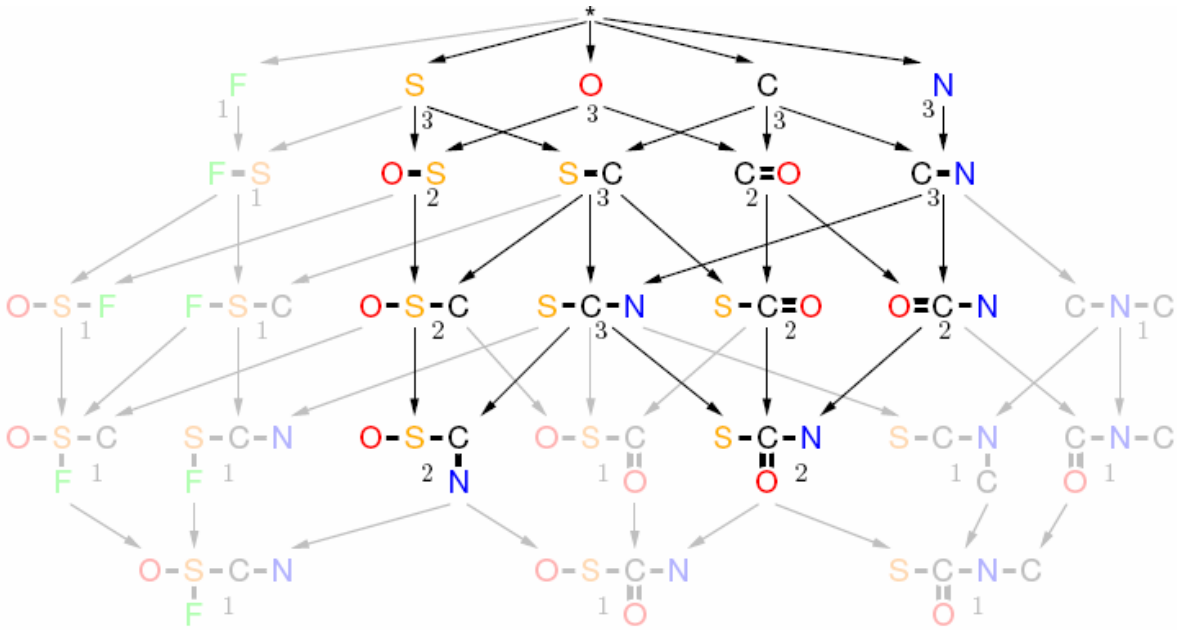
- The semi-lattice should be searched top-down.
- Standard search strategies: breadth-first and depth-first.
- Depth-first search is preferable, since the semi-lattice can be very wide.

example molecules:









Closed Fragments: Lossless Output Reduction

- A graph fragment is called **closed** if no supergraph of it has the same support. (That is, no extension of the fragment is possible in all containing graphs.)

- There are usually a lot fewer closed frequent fragments than frequent fragments.
  → The **output is considerably smaller.**

- **Confining oneself to closed fragments does not lose information:**

  All frequent fragments can be constructed from the closed ones and their support can be computed from the support of the closed fragments.
  - Any subgraph of a closed frequent fragment is frequent.
  - The support of a non-closed frequent fragment is the maximum of the support of the closed frequent fragments that are supergraphs of it.

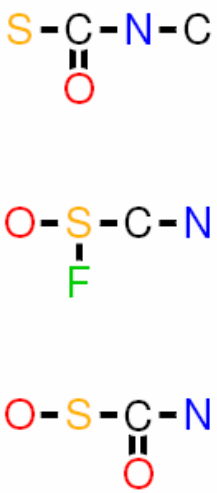
- One may even go further by restricting the search to **maximal subgraphs.** However, this is not lossless w.r.t. the subgraph support.

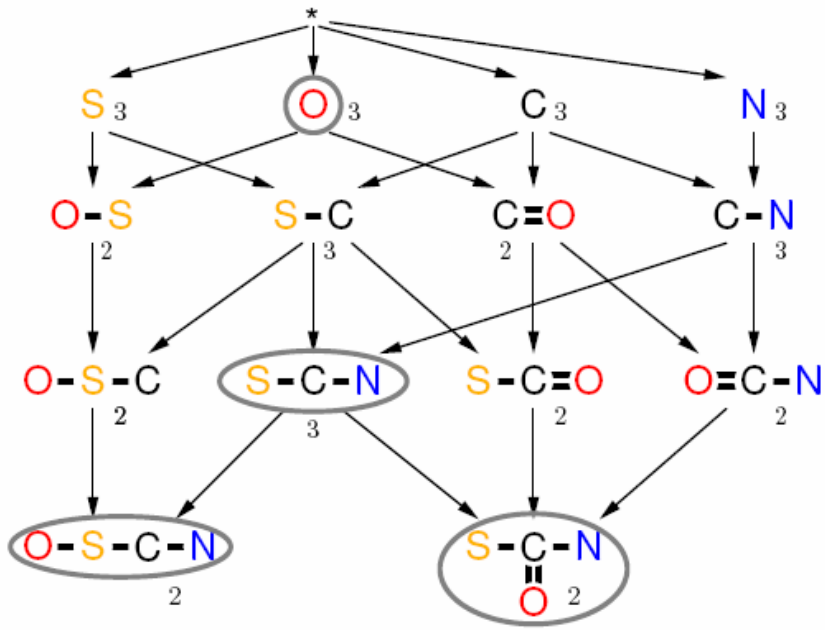
Closed Fragments: Lossless Output Reduction

Sub-semi-lattice of frequent subgraphs.

- Closed frequent subgraphs are encircled.
- There are 14 frequent fragments, but only 4 closed fragments.
- The two closed fragments at the bottom are also maximal.

example molecules:



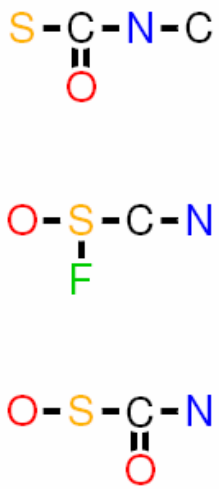


Basic Search Principle

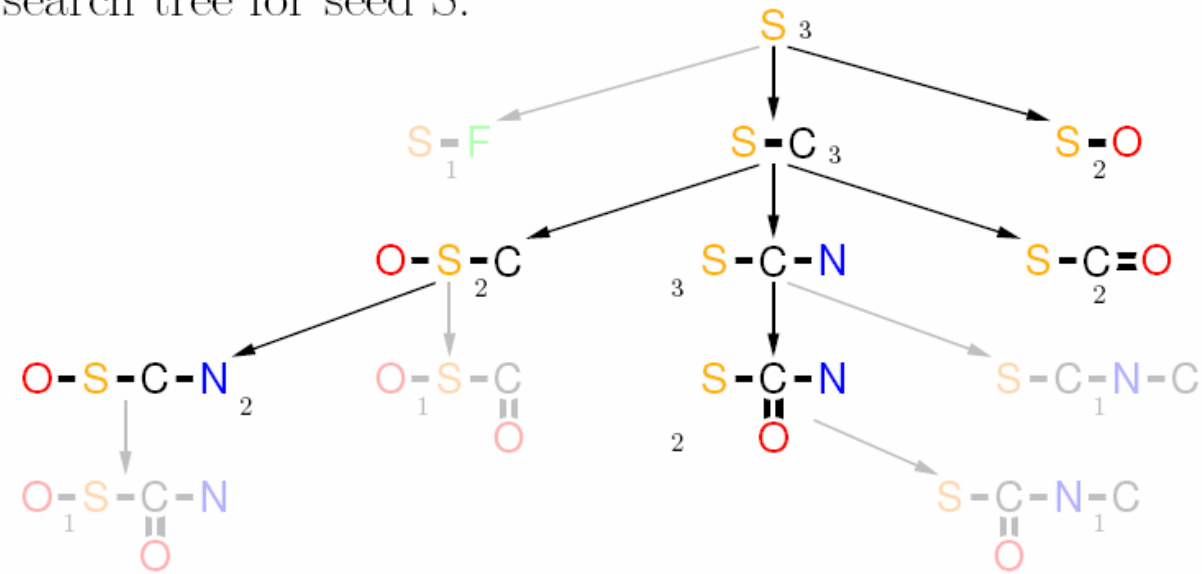
Grow subgraphs into the graphs of the given database.

- Start with a single node (seed node).
- Add an edge (and maybe a node) in each step.
- Determine the support and prune infrequent subgraphs.

example molecules:



search tree for seed S:



Pruning the Search

In applications the search trees tend to get very large, so we have to prune them.

- **Size Based Pruning:**
  - Prune the search tree if a certain depth is reached.
  - Restrict fragments to a certain size.
- **Support Based Pruning:**
  - No superstructure of an infrequent fragment can be frequent.
  - No counters for fragments having an infrequent substructure are needed.
- **Structural Pruning:**
  - Make sure that there is only one counter for each possible fragment.
  - Explains the unbalanced structure of the full search tree.
  - Can be achieved with *canonical forms of graphs.*

Canonical Forms of Graphs: General Idea

- Construct a **code word** that uniquely identifies an (attributed) graph up to isomorphism and symmetry (i.e. automorphism).
- **Basic idea:** The characters of the code word describe the edges of the graph.
- **Core problem:** Node and edge attributes can easily be incorporated into a code word, but how to describe the connection structure is not so obvious.
- The nodes of the graph must be numbered (endowed with unique labels), because we need to specify the source and the destination node of an edge.
- Each possible numbering of the nodes of the graph yields a code word, which is the concatenation of the sorted edge descriptions ("characters"). (Note that the graph can be reconstructed from such a code word.)
- The resulting list of code words is sorted lexicographically.
- The lexicographically smallest code word is the **canonical description**.

Canonical Forms: Constructing Spanning Trees

- For graph mining the canonical form should have the **prefix property:**
  Any prefix of a canonical code word is a canonical code word itself.
  (Because it guarantees that all possible graphs can be reached with it.)

- With a search restricted to connected substructures, we can ensure this by
  - systematically constructing a **spanning tree** of the graph,
    numbering the nodes in the order in which they are visited,
  - sorting the edge descriptions into the order in which the edges are added.

- The most common ways of constructing a spanning trees are
  - **depth-first search** → canonical form of gSpan
  - **breadth-first search** → canonical form of MoSS/MoFa

- An alternative way is to create all children of a node before proceeding in a depth-first manner (can be seen as a variant of depth-first search).

Canonical Forms: Edge Sorting Criteria

- The **edge description** consists of
  - the indices of the source and the destination atom (definition: the source of an edge is the node with the smaller index),
  - the attributes of the source and the destination atom,
  - the edge attribute.

- Sorting the edges into insertion order must be achieved by a **precedence order** on the describing elements of an edge.

- Order of individual elements (conjectures, but supported by experiments):
  - Node and edge attributes should be sorted according to their frequency.
  - Ascending order seems to be recommendable for the node attributes.

- **Simplification:** the source attribute is needed only for the first edge and thus can be split off from the list of edge descriptions.

Canonical Forms: Edge Sorting Criteria

- **Precedence Order for Depth-first Search:**
  - destination node index (ascending)
  - source node index (descending) ←
  - edge attribute (ascending)
  - destination node attribute (ascending)

- **Precedence Order for Breadth-first Search:**
  - source node index (ascending)
  - edge attribute (ascending)
  - destination node attribute (ascending)
  - destination node index (ascending)

- **Edges Closing Cycles:**

  Edges closing cycles may be distinguished from spanning tree edges, giving spanning tree edges absolute precedence over edges closing cycles.

Canonical Forms: Code Words

From these edge sorting criteria, the following code words result (regular expressions with non-terminal symbols):

- **Depth-First Search:** $a\,(i_d\,\underline{i_s}\,b\,a)^m$ (or $a\,(i_d\,b\,a\,\underline{i_s})^m$)
- **Breadth-First Search:** $a\,(i_s\,b\,a\,i_d)^m$ (or $a\,(i_s\,i_d\,b\,a)^m$)

where
- n the number of nodes of the graph,
- m the number of edges of the graph,
- i_s index of the source node of an edge, $i_s \in \{1, \ldots, n\}$,
- i_d index of the destination node of an edge, $i_d \in \{1, \ldots, n\}$,
- a the attribute of a node,
- b the attribute of an edge.

The order of the elements describing an edge reflects the precedence order.

That i_s in the depth-first search expression is underlined is meant as a reminder that the edge descriptions have to be sorted descendingly w.r.t. this value.

Searching with Canonical Forms

Principle of the Search Algorithm:

- **Base Loop:**
  - Traverse all possible node attributes, i.e.,
    the canonical code words of single node fragments.
  - Recursively process each code word that describes a frequent fragment.

- **Recursive Processing:**

  For a given (canonical) code word of a frequent fragment:
  - Generate all possible extensions by an edge (and a maybe a node).
    This is done by appending the edge description to the code word.
  - Check whether the extended code word is the **canonical form**
    of the fragment described by the extended code word
    (and whether the described fragment is frequent).

    If it is, process the extended code word recursively, otherwise discard it.

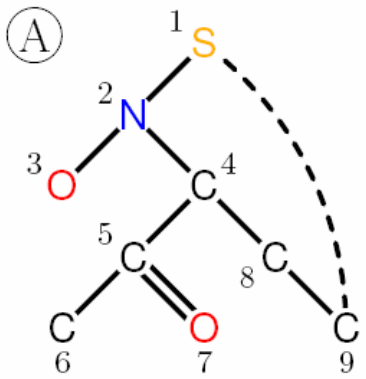
Checking for Canonical Form: Compare Prefixes

- **Base Loop:**
  - Traverse all nodes that have the same attribute as the current root node (first character of the code word; possible roots of spanning tree).

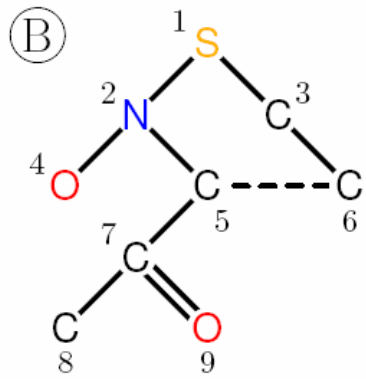
- **Recursive Processing:**
  - The recursive processing constructs alternative spanning trees and compare the code words resulting from it with the code word to check.
  - In each recursion step one edge is added to the spanning tree and its description is compared to the corresponding one in the code word to check.
  - If the new edge description is **larger**, the edge can be skipped (new code word is lexicographically larger).
  - If the new edge description is **smaller**, the code word is not canonical (new code word is lexicographically smaller).
  - If the new edge description is **equal**, the rest of the code word is processed recursively (code word prefixes are equal).

Canonical Forms: A Simple Example

example molecule



depth-first



breadth-first

Order of Elements: $\mathtt{S} \prec \mathtt{N} \prec \mathtt{O} \prec \mathtt{C}$ **Order of Bonds:** $- \prec =$

Code Words:

```
A: S 21-N 32-O 42-C 54-C 65-C 75=O 84-C 98-C 91-S
B: S 1-N2 1-C3 2-O4 2-C5 3-C6 5-C6 5-C7 7-C8 7=O9
```

(Reminder: in A the edges are sorted *descendingly* w.r.t. the second entry.)

Canonical Forms: Restricted Extensions

Principle of the Search Algorithm up to now:

- Generate all possible extensions of a given (frequent) fragment by an edge (and a maybe node).
- Check whether the extended fragment is in canonical form (and frequent).
  If it is, process the extended fragment recursively, otherwise discard it.

Straightforward Improvement:

- For some extensions of the given (frequent) fragment it is easy to see that they are not in canonical form.
- The trick is to check whether a spanning tree **rooted at the same node** yields a code word that is smaller than the one describing the fragment.
- This immediately rules out extensions of certain nodes in the fragment as well as certain edges closing cycles.

Canonical Forms: Restricted Extensions

Depth-First Search: Rightmost Extension

- **Extendable Nodes:**
  - Only nodes on the **rightmost path** of the spanning tree may be extended.
  - If the source node of the new edge is not a leaf, the edge description must not precede the description of the downward edge on the path.

    (That is, the edge attribute must be no less than the edge attribute of the downward edge, and if it is equal, the attribute of its destination node must be no less than the attribute of the downward edge's destination node.)

- **Edges Closing Cycles:**
  - Edges closing cycles must start at an extendable node.
  - They must lead to the rightmost leaf (node at end of rightmost path).
  - The index of the source node must precede the index of the source node of any edge already incident to the rightmost leaf.

Canonical Forms: Restricted Extensions

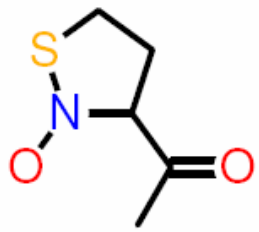
Breadth-First Search: Maximum Source Extension

- **Extendable Nodes:**
  - Only nodes having an index no less than the **maximum source index** of an edge already in the fragment may be extended.
  - If the source of the new edge is the one having the maximum source index, it may be extended only by edges whose descriptions do not precede the description of any downward edge already incident to this node.

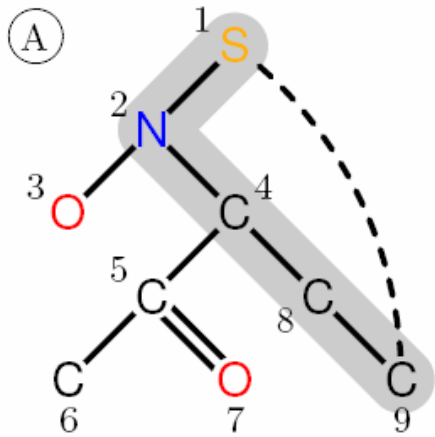
    (That is, the edge attribute must be no less, and if it is equal, the attribute of the destination node must be no less.)

- **Edges Closing Cycles:**
  - Edges closing cycles must start at an extendable node.
  - They must lead “forward”,
    that is, to a node having a larger index than the extended node.

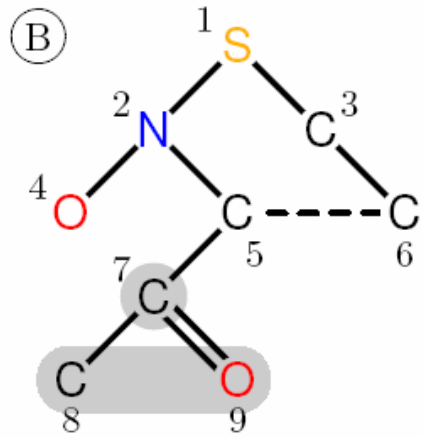
Restricted Extensions: A Simple Example



example molecule



depth-first



breadth-first

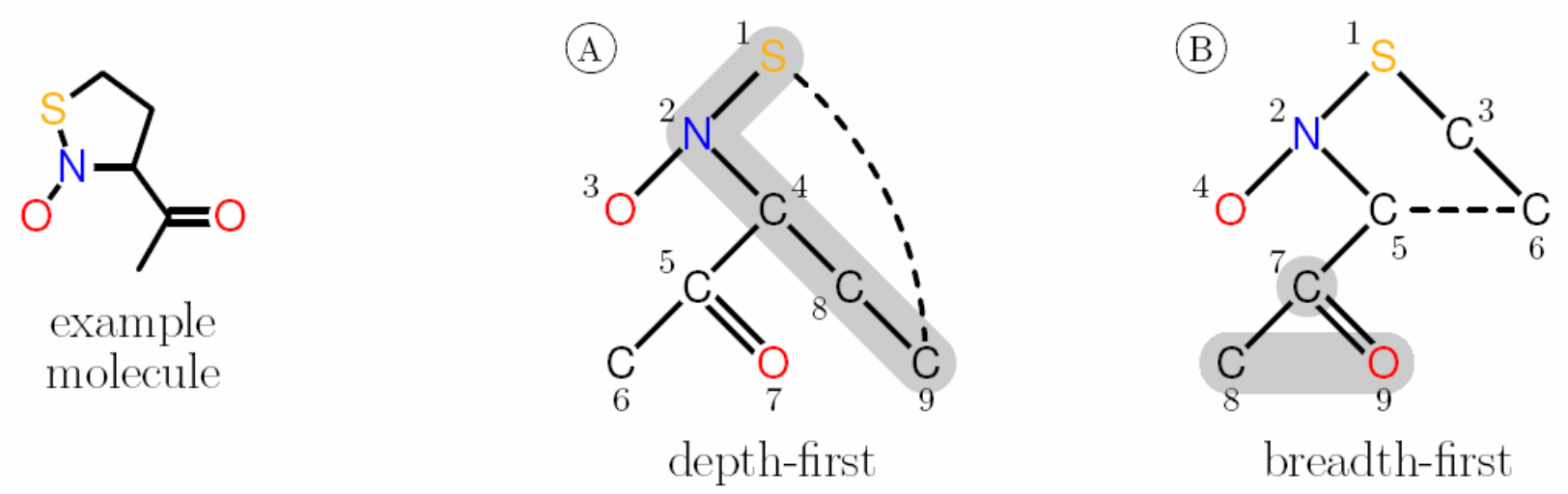
Extendable Nodes:

- A: nodes on the rightmost path, i.e., 1, 2, 4, 8, 9.
- B: nodes with an index no smaller than the maximum source, i.e., 7, 8, 9.

Edges Closing Cycles:

- A: none, because the existing cycle edge has minimum source.
- B: edge between nodes 8 and 9.

Restricted Extensions: A Simple Example

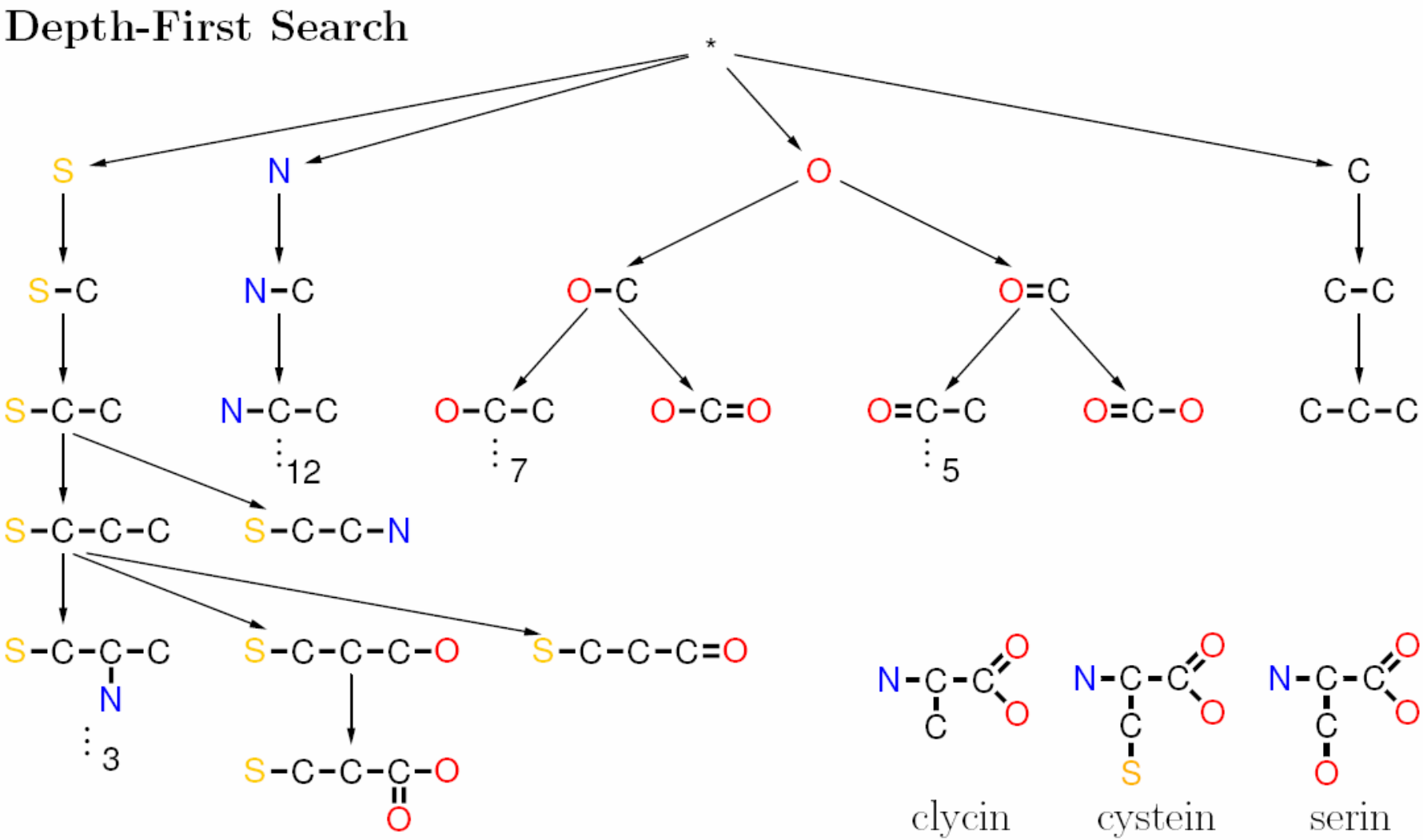


If other nodes are extended, a tree *with the same root* yields a smaller code word.

Examples:

```
A:  S 21-N 32-O 42-C 54-C 65-C 75=O 84-C 98-C 91-S 03-C
    S 21-N 32-C 43-C …
B:  S 1-N2 1-C3 2-O4 2-C5 3-C6 5-C6 5-C7 7-C8 7=O9 4-C0
    S 1-N2 1-C3 2-O4 2-C5 3-C6 4-C7 …
```

Searching without a Seed Atom



Canonical Forms: Comparison

Depth-First vs. Breadth-First Search Canonical Form

- With breadth-first search canonical form the extendable nodes are much easier to traverse, as they always have consecutive indices:
  One only has to store and update one number, namely the index of the maximum bond source, to describe the node range.

- Also the check for canonical form is slightly more complex (to program) for depth-first canonical form (maybe I did not find the best way, though).

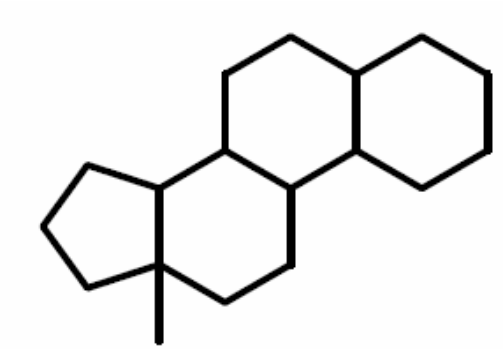
- The two canonical forms obviously lead to different branching factors, widths and depths of the search tree.
  However, it is not immediately clear, which form leads to the “better” (more efficient) structure of the search tree.

- The experimental results reported in the following indicate that it may depend on the data set which canonical form performs better.

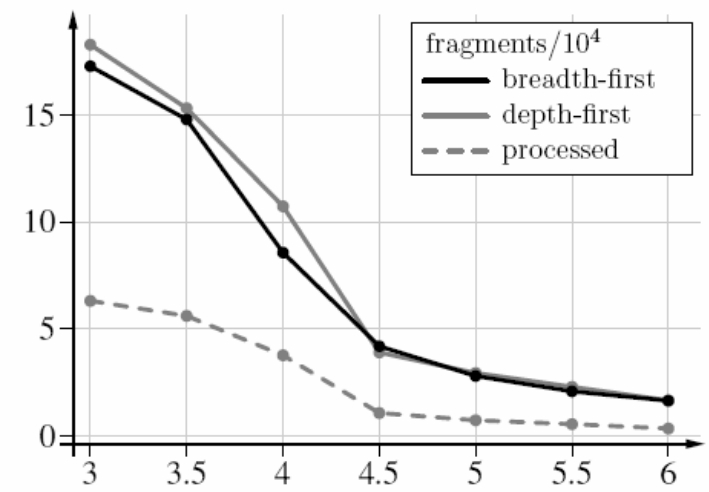
Experimental Results: Data Sets

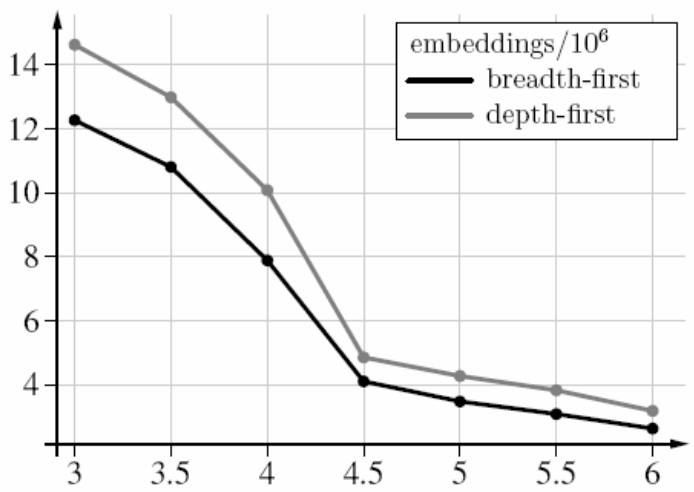
- **Index Chemicus — Subset of 1993**
  - 1293 molecules / 34431 atoms / 36594 bonds
  - Frequent fragments down to fairly low support values are trees (no rings).
  - Medium number of fragments and closed fragments.

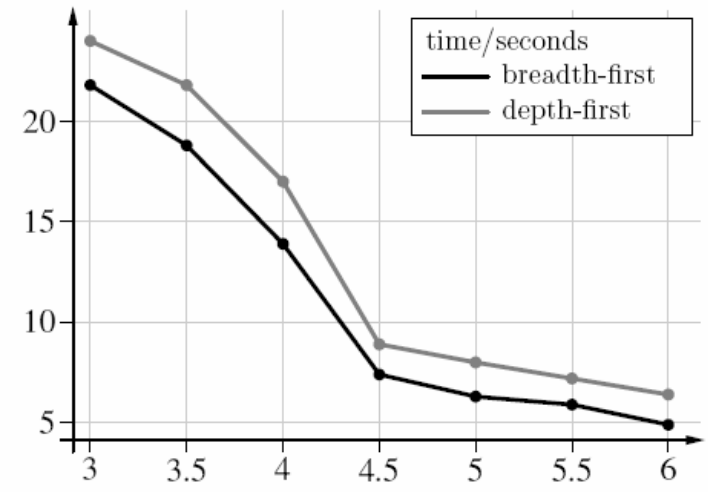
- **Steroids**
  - 17 molecules / 401 atoms / 456 bonds
  - A large part of the frequent fragments contain one or more rings.
  - Huge number of fragments, still large number of closed fragments.



Experimental Results: IC93 Data Set

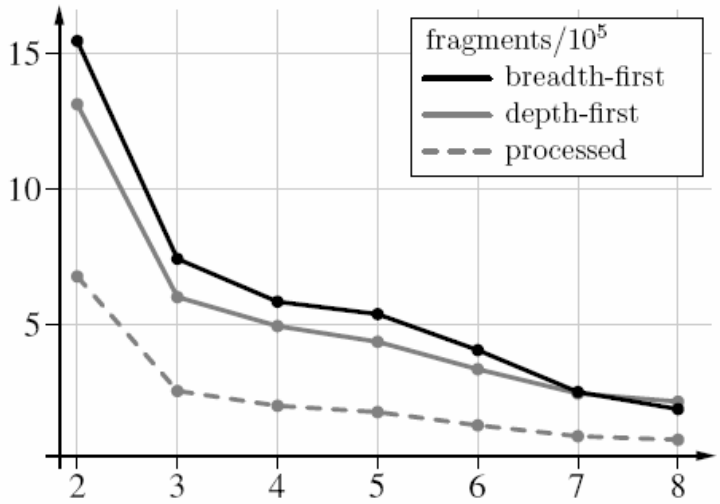


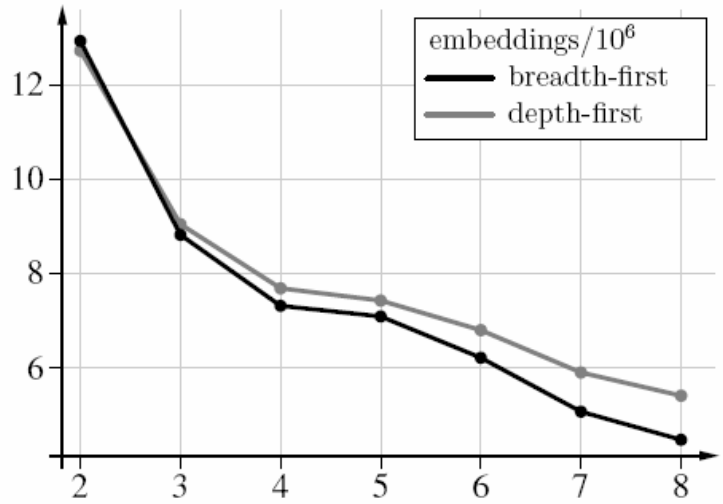


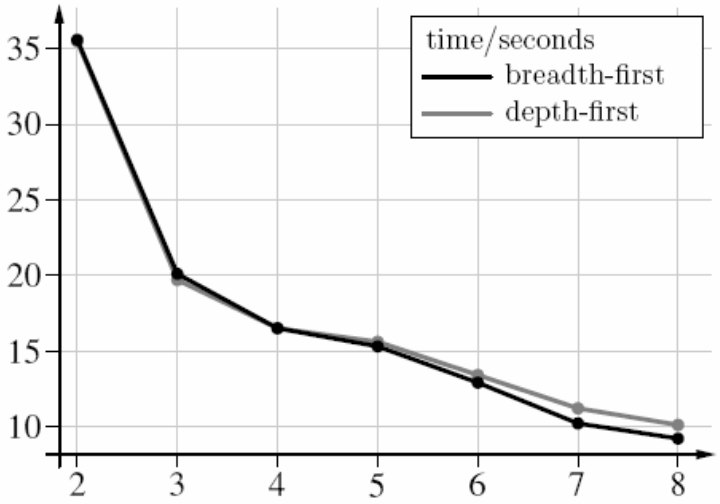


Experimental results on the IC93 data. The horizontal axis shows the minimal support in percent. The curves show the number of generated and processed fragments (top left), number of generated embeddings (top right), and the execution time in seconds (bottom left) for the two canonical forms/extension strategies.

Experimental Results: Steroids Data Set







Experimental results on the steroids data. The horizontal axis shows the absolute minimal support. The curves show the number of generated and processed fragments (top left), number of generated embeddings (top right), and the execution time in seconds (bottom left) for the two canonical forms/extension strategies.

Alternative Test: Equivalent Siblings

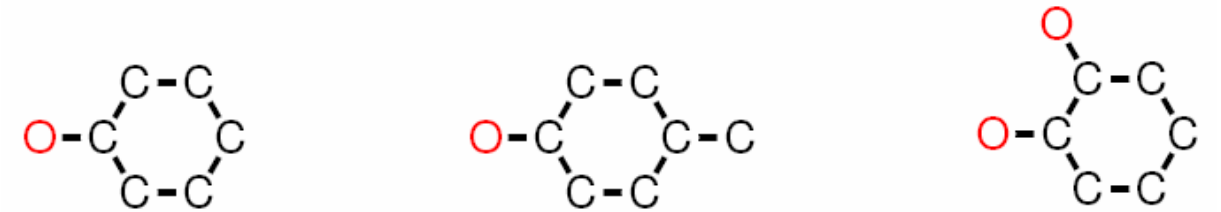
- **Basic Idea:**
  - If the fragment to extend exhibits a certain symmetry, several extensions may be equivalent (in the sense that they describe the same fragment).
  - At most one of these sibling extensions can be in canonical form, namely the one *least restricting future extensions* (smallest code word).
  - Identify equivalent siblings and keep only the maximally extendable one.

- **Test Procedure for Equivalence:**
  - Get any molecule into which two sibling fragments to compare can be embedded. (If there is no such molecule, the siblings are not equivalent.)
  - Mark any embedding of the first fragment in the molecule.
  - Traverse all embeddings of the second fragment into the molecule and check whether all bonds of an embedding are marked. If there is such an embedding, the two fragments are equivalent.

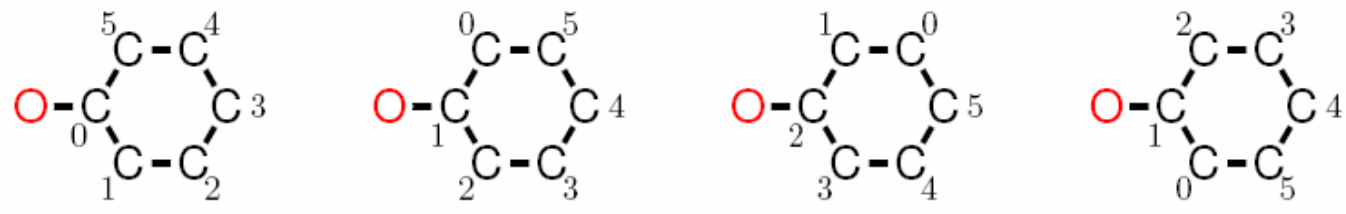
Alternative Test: Equivalent Siblings

**If siblings in the search tree are equivalent,
only the one with the least restrictions needs to be processed.**

Example: Mining phenol, p-cresol, and catechol.



Consider extensions of a benzene ring (twelve possible embeddings):



Only the fragment that **least restricts future extensions**
(i.e., that has the smallest code word) can be in canonical form.

Alternative Test: Equivalent Siblings

- **Test for Equivalent Siblings before Test for Canonical Form**
  - Traverse the sibling extensions and compare each pair.
  - Of two equivalent siblings remove the one that restricts future extensions more.

- **Advantages:**
  - Identifies some fragments that are non-canonical in a simple way.
  - Test of two siblings is at most linear in the number of bonds.

- **Disadvantages:**
  - Does not identify all non-canonical fragments, therefore a subsequent canonical form test is still needed.
  - Compares two sibling fragments, therefore it is quadratic in the number of siblings.

Alternative Test: Equivalent Siblings

The effectiveness of equivalent sibling pruning depends on the canonical form:

Mining the **IC93 data** with 4% minimal support

	depth-first	breadth-first
equivalent sibling pruning	156 (1.9%)	4195 (83.7%)
canonical form pruning	7988 (98.1%)	815 (16.3%)
total pruning	8144	5010
(closed) fragments found	2002	2002

Mining the **steroids data** with minimal support 6

	depth-first	breadth-first
equivalent sibling pruning	15327 (7.2%)	152562 (54.6%)
canonical form pruning	197449 (92.8%)	127026 (45.4%)
total pruning	212776	279588
(closed) fragments found	1420	1420

Alternative Test: Equivalent Siblings

Observations:

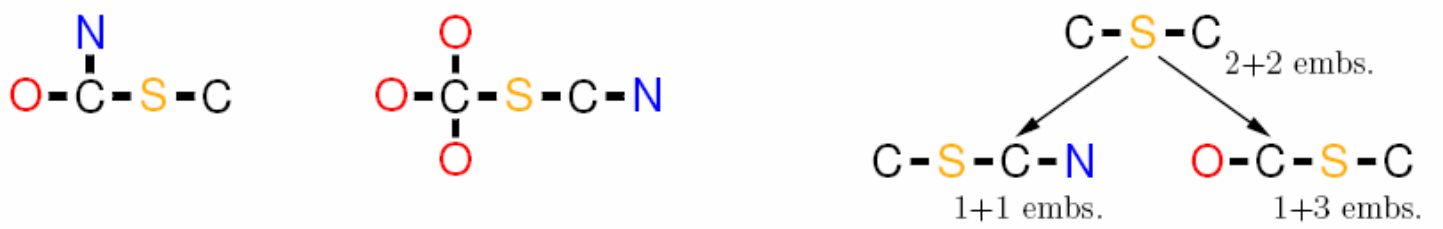
- Depth-first form generates more duplicate fragments the on IC93 data and fewer duplicate fragments on the steroids data (as seen before).
- There are only very few equivalent siblings with depth-first form on both the IC93 data and the steroids data.

  (Conjecture: equivalent siblings result from “rotated” tree branches, which are less likely to be siblings with depth-first form.)
- With breadth-first form a large part of the fragments that are not in canonical form can be filtered out with equivalent.
- On the test IC93 data no difference in speed could be observed, presumably because pruning takes only a small part of the total time.
- On the steroids data, however, equivalent sibling pruning yields a slight speed-up for breadth-first form (~ 5%).

Perfect Extensions

- An extension of a graph fragment is called **perfect**,
  if it can be applied to all embeddings of the fragment in exactly the same way.

- **Attention:** It may not be enough to compare the support and the number of embeddings of the graph fragment.
  (Even though perfect extensions must have the same support and an integer multiple of the number of embeddings of the base fragment.)



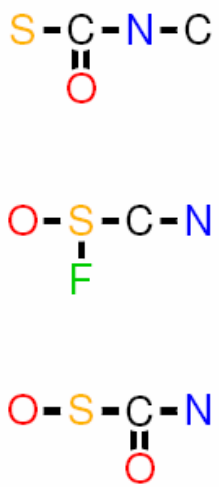
- **Consequence:** It may be necessary to check whether all embeddings of the base fragment lead to the same number of extended embeddings.

- **Additional problem:** Rings/cycles of different size can lead to problems.

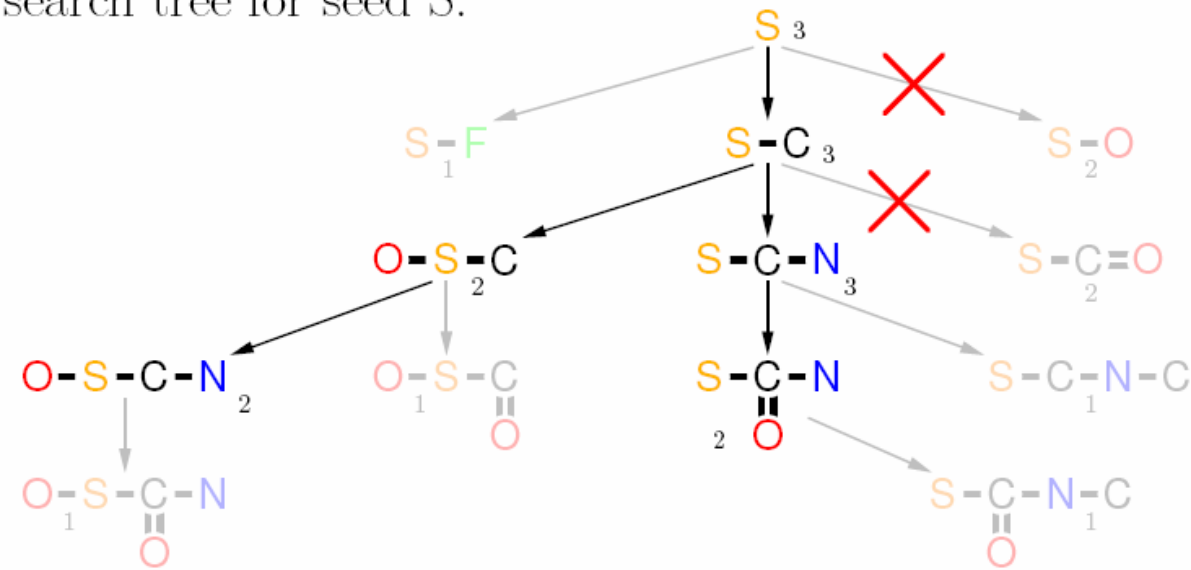
Partial Perfect Extension Pruning

- **Basic idea of perfect extension pruning:**
  First grow a fragment to the biggest common substructure.
- **Partial perfect extension pruning:** If the children of a search tree node are ordered lexicographically (w.r.t. their code word), no fragment in a subtree to the right of a perfect extension branch can be closed.

example molecules:

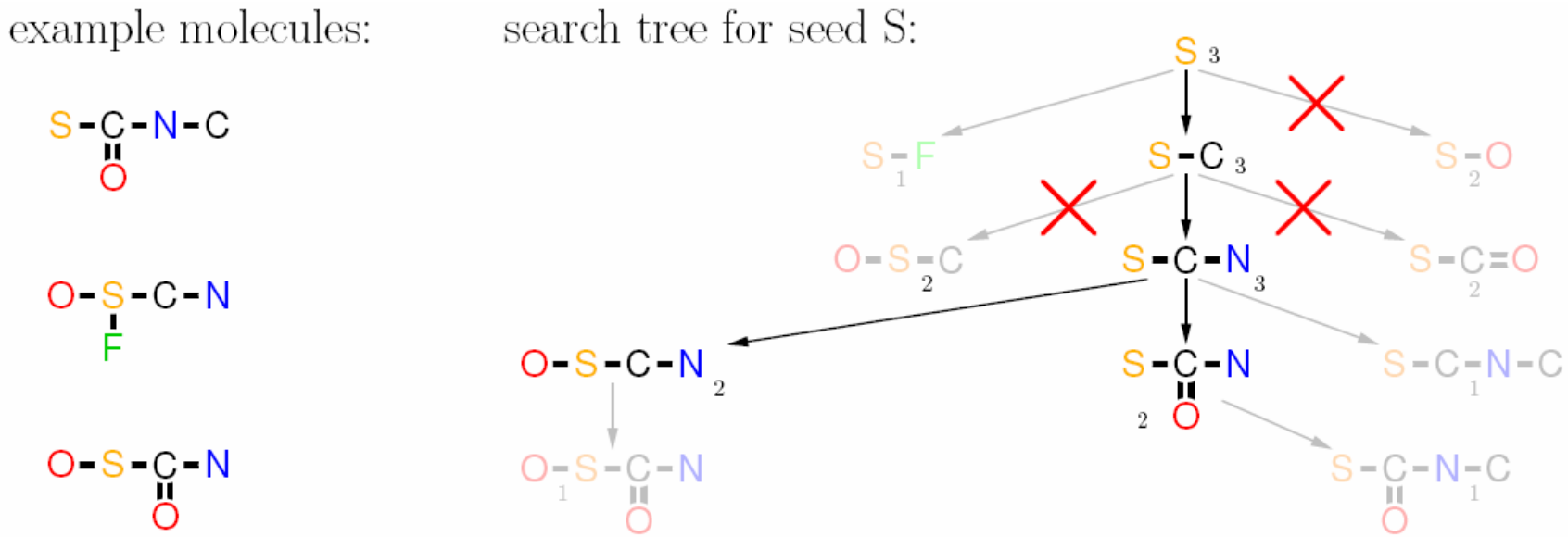


search tree for seed S:



Full Perfect Extension Pruning

- **Full perfect extension pruning:**
  Also prune the branches to the left of the perfect extension branch.
- **Problem:** This pruning method interferes with canonical form pruning, because the extensions in the left siblings cannot be repeated in the perfect extension branch (restricted extensions, “simple rules” for canonical form).

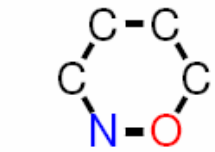


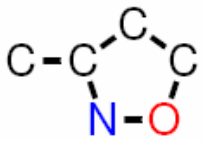
Reorganizing a Fragment's Code Word

- **Restricted extensions:**
  Not all extensions of a fragment are allowed by the canonical form.
  Some can be checked by simple rules (rightmost path/max. source extension).
- **Consequence:** In order to make canonical form pruning and full perfect extension pruning compatible, the restrictions on extensions must be mitigated.
- **Code word reorganization:** It must be possible to shift descriptions of new edges past descriptions of perfect extension edges in the code word.
  - The code word of a fragment consists of two parts:
    - a (possibly empty) **suffix** of perfect extension edges and
    - a **prefix** ending with a non-perfect extension edge.
  - A new edge description is usually appended at the end of the code word.
  - However, if the suffix is not empty, the new edge may be inserted into the suffix or even moved directly in front of the suffix.
    (Whichever possibility yields the lexicographically smallest code word.)

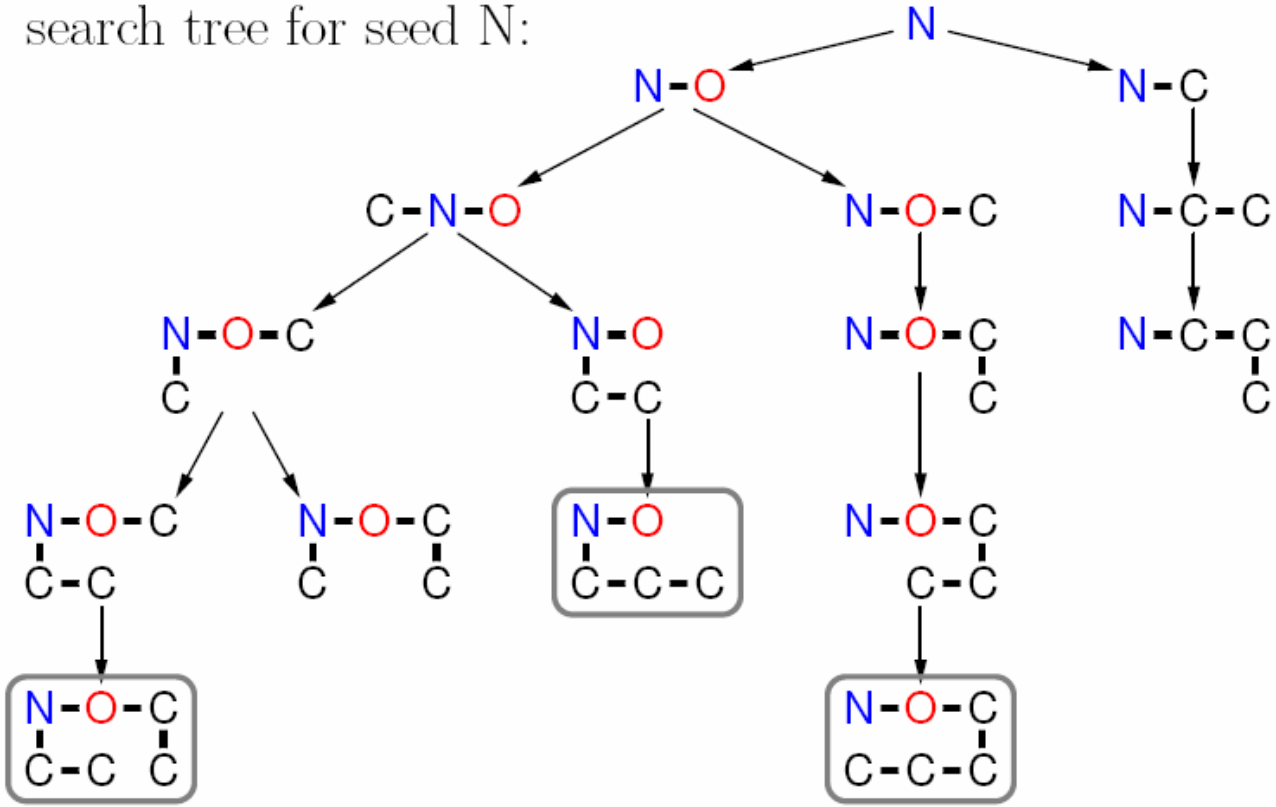
Perfect Extensions: Problems with Rings/Cycles

example molecules:



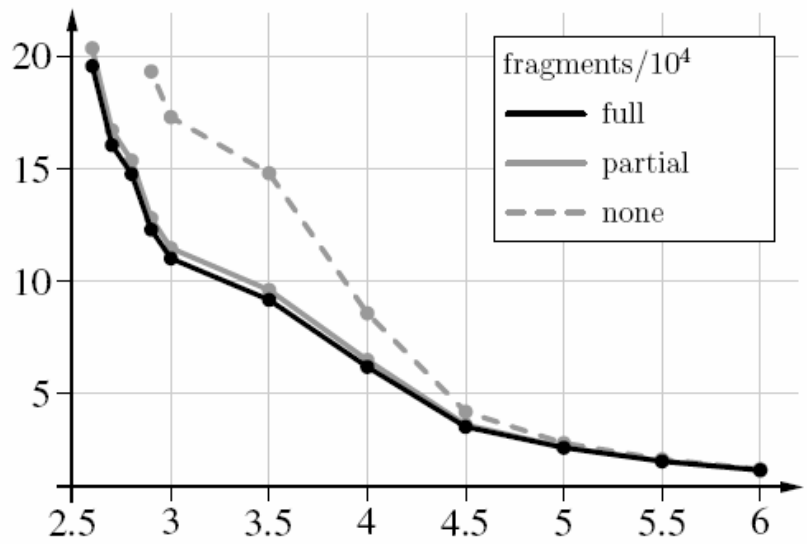


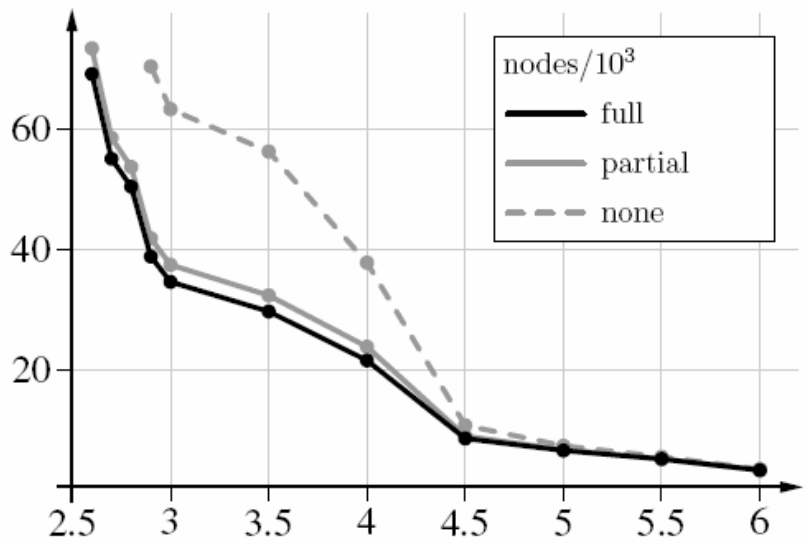
search tree for seed N:

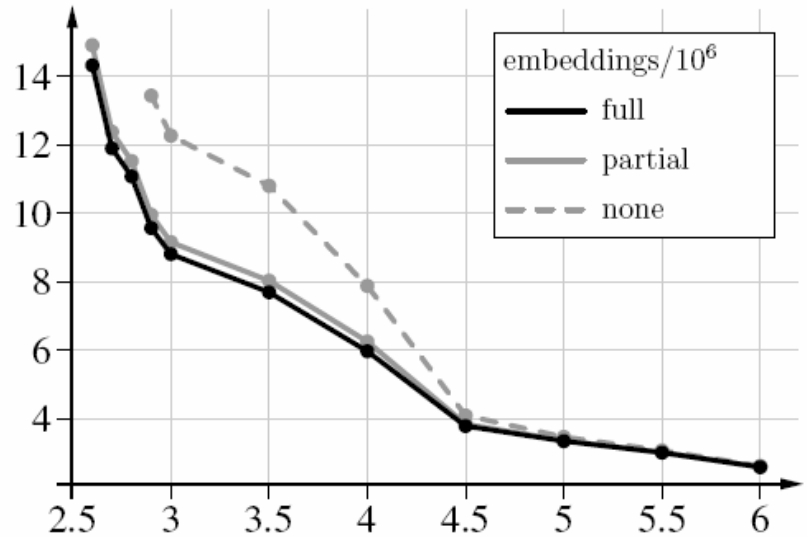


- **Problem:** Perfect extensions in rings may not allow for pruning.
- **Consequence: Additional constraint**
  Perfect extensions must be bridges or edges closing a ring/cycle.

Experimental Results: IC93 without Ring Mining

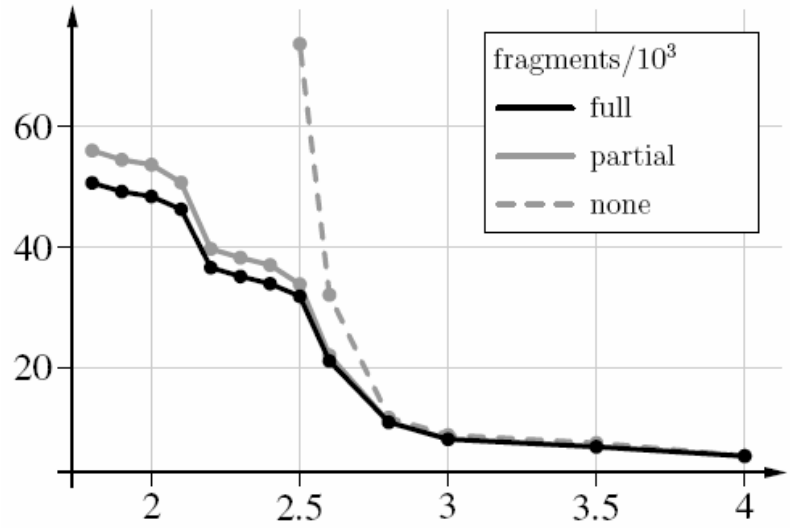


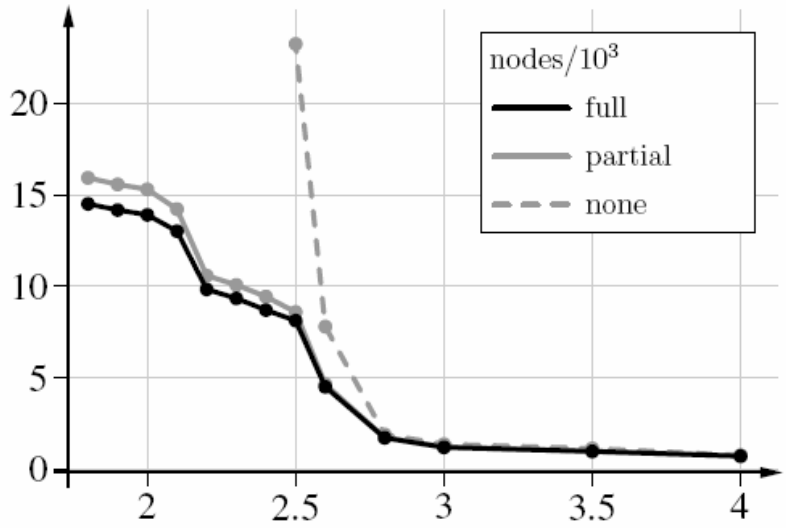


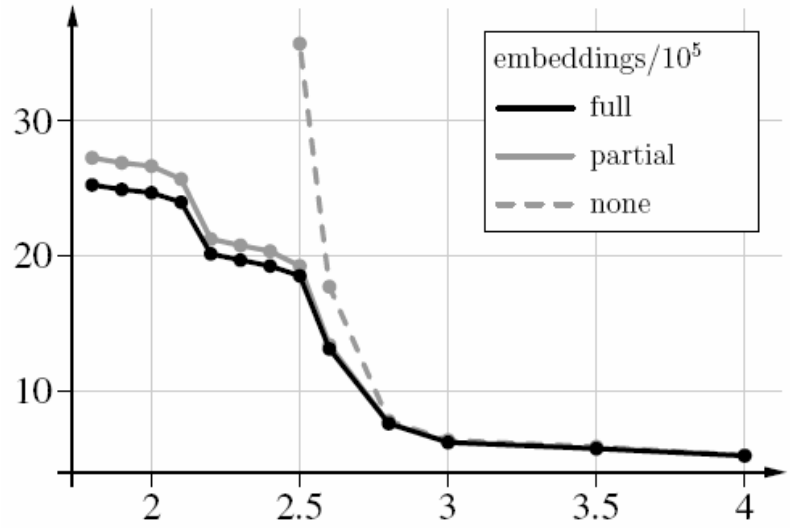


Experimental results on the IC93 data, obtained without ring ming (single bond extensions). The horizontal axis shows the minimal support in percent. The curves show the number of generated fragments (top left), the number of generated embeddings (bottom left), and the number of search tree nodes (top right) for the three different methods.

Experimental Results: IC93 with Ring Mining







Experimental results on the IC93 data, obtained with ring mining. The horizontal axis shows the minimal support in percent. The curves show the number of generated fragments (top left), the number of generated embeddings (bottom left), and the number of search tree nodes (top right) for the three different methods.

Extensions of the Search Algorithm

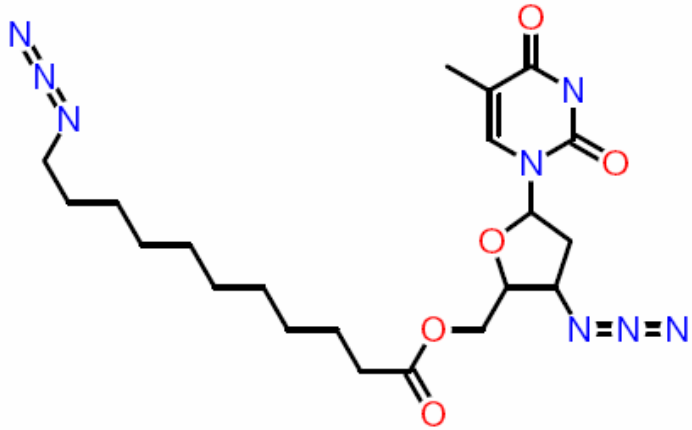
- **Rings**
  - Preprocessing: Find rings in the molecules and mark them.
  - In the search process: Add all atoms and bonds of a ring in one step.
  - Considerably improves efficiency and interpretability.

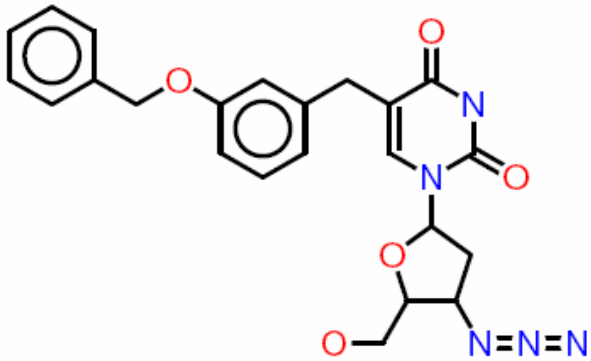
- **Carbon Chains**
  - Add a carbon chain in one step, ignoring its length.
  - Extensions by a carbon chain match regardless of the chain length.

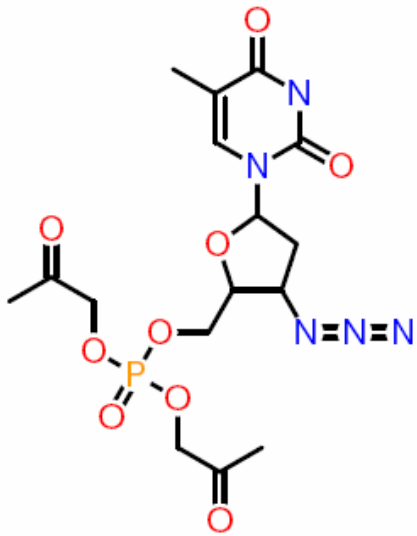
- **Wildcard Atoms**
  - Define classes of atoms that can be seen as equivalent.
  - Combine fragment extensions with equivalent atoms.
  - Infrequent fragments that differ only in a few atoms from frequent fragments can be found.

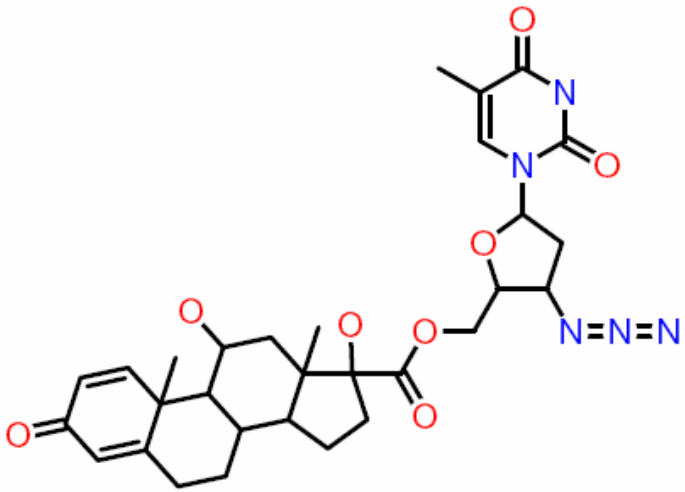
NCI DTP HIV Antiviral Screen: AZT

Some Molecules from NCI HIV Database

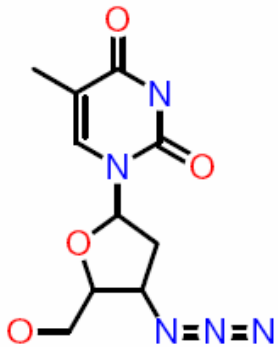








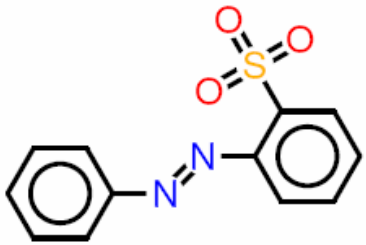
Common Fragment



NCI DTP HIV Antiviral Screen: Other Fragments

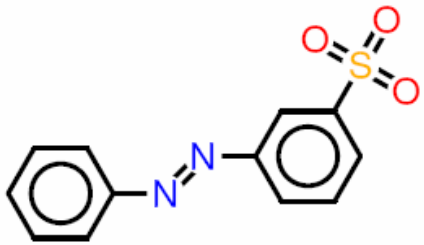
Fragment 1:

CA: 5.23%
CI/CM: 0.05%



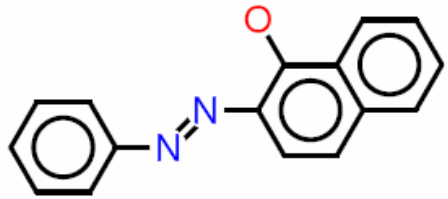
Fragment 2:

CA: 4.92%
CI/CM: 0.07%



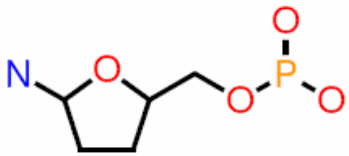
Fragment 3:

CA: 5.23%
CI/CM: 0.08%



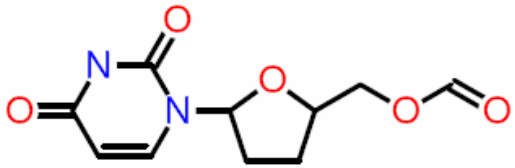
Fragment 4:

CA: 9.85%
CI/CM: 0.07%



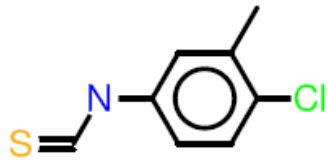
Fragment 5:

CA: 10.15%
CI/CM: 0.04%



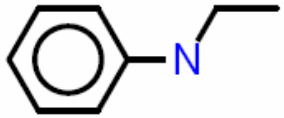
Fragment 6:

CA: 9.85%
CI/CM: 0.00%

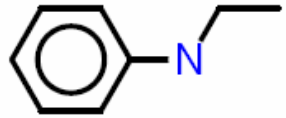


Experimental Results: Ring Extensions

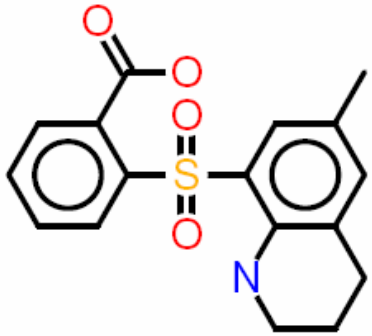
Improved Interpretability



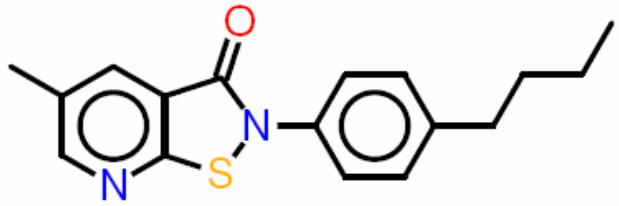
Fragment 3
basic algorithm
freq. in CA: 22.77%



Fragment 4
with ring extensions
freq. in CA: 20.00%



NSC #667948



NSC #698601

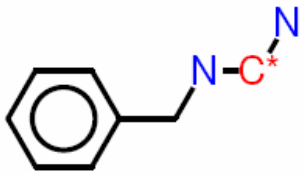
Compounds from the NCI cancer data set that contain Fragment 3 but not 4.

Experimental Results: Carbon Chains

- Technically: Add a carbon chain in one step, ignoring its length.
- Extension by a carbon chain match regardless of the chain length.
- Advantage: Fragments can represent carbon chains of varying length.

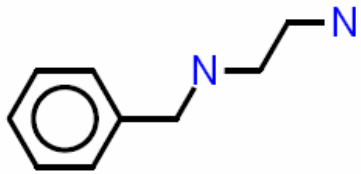
Example from the NCI Cancer Dataset:

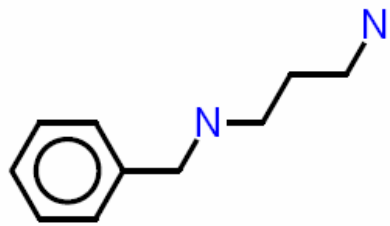
Fragment with Chain



freq. CA: 1.48%
freq. CI: 0.13%

Actual Structures

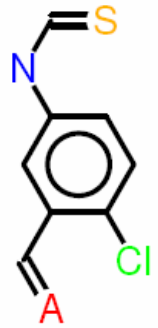




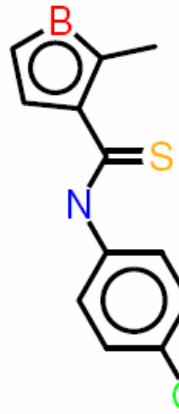
Experimental Results: Wildcard Atoms

- Define classes of atoms that can be considered as equivalent.
- Combine fragment extensions with equivalent atoms.
- Advantage: Infrequent fragments that differ only in a few atoms from frequent fragments can be found.

Examples from the NCI HIV Dataset:



	A=O	A=N
CA:	5.5%	3.7%
CI/CM:	0.0%	0.0%



	B=O	B=S
CA:	5.5%	0.01%
CI/CM:	0.0%	0.0%

Summary

- Frequent graph mining is closely related to frequent item set mining:
  Find frequent subgraphs instead of frequent subsets.
- A core problem of frequent graph mining is how to avoid redundant search.
  This problem is solved with the help of **canonical forms of graphs.**
  Different canonical forms lead to different behavior of the search algorithm.
- The restriction to **closed fragments** is a lossless reduction of the output.
  All frequent fragments can be reconstructed from the closed ones.
- A restriction to closed fragments allows for additional pruning strategies:
  partial and full **perfect extension pruning.**
- Extensions of the basic algorithm (particularly useful for molecules) include:
  Ring Mining, (Carbon) Chain Mining, and **Wildcard Nodes.**
- A **Java implementation** for molecular fragment mining is available at:
  `http://www.borgelt.net/moss.html`