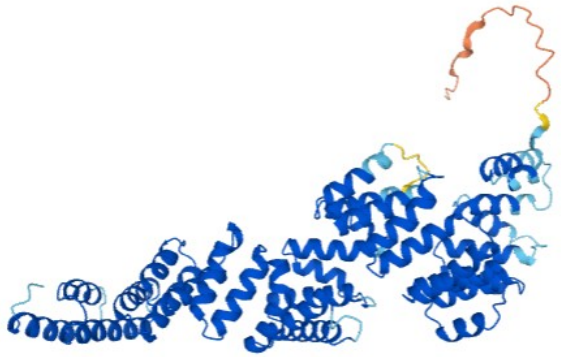
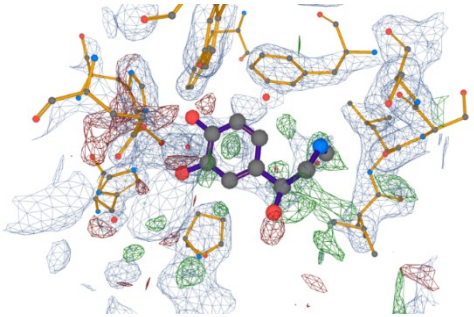
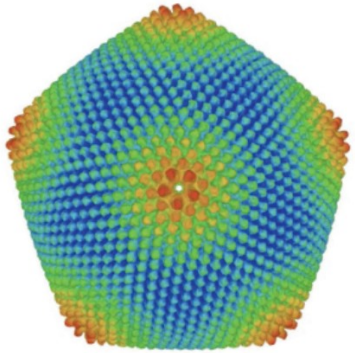
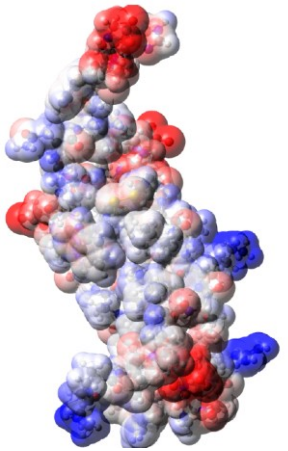
2D structure of a molecule
Isomorphism and canonical indexing



Radka Svobodová

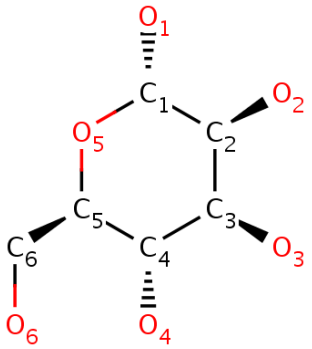


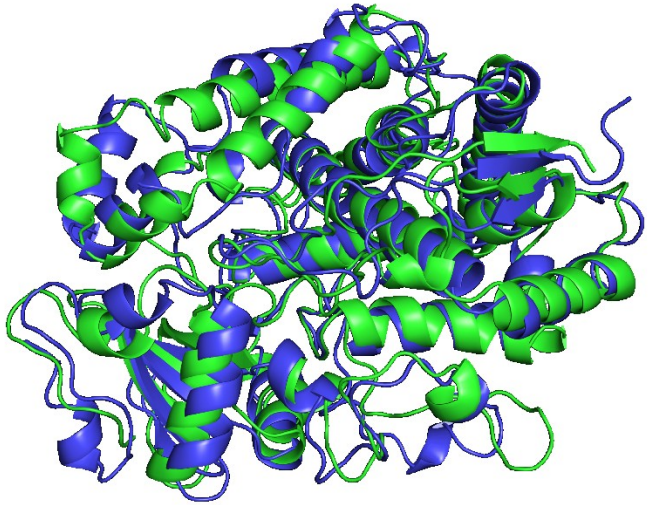




Content

- **Introduction:** concept of chemoinformatics, content of the subject, history of the field
- **Computer model of a molecule:** 1D, 2D and 3D structure, molecule representation using graph and matrix
- **2D structure (topology) of a molecule:**
  - writing a molecule using a string (SMILES, InChi, InChiKey)
  - **Molecular graphs:** Isomorphism and canonical indexing
- **3D structure (geometry) of the molecule:**
  - representation using Cartesian and internal coordinates, data formats, geometry comparison





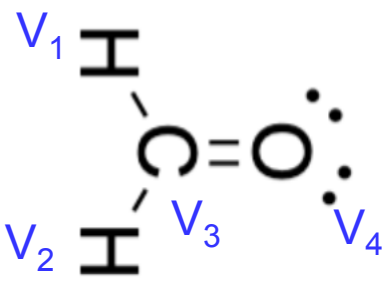
2D structure as a molecular graph

Definition of molecular graph:

$G = (V, E, L, \varphi, \beta)$

- V – set of nodes (atoms)
- E – multiset of edges (bonds)
- L – multiset of loops (free electron pairs)
- φ – function for naming of nodes by element symbols
- β – set of element symbols

Molecule:



Description of its mol. graph:

$V = \{v_1, v_2, v_3, v_4\}$

$E = \{\{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_4\}\}$

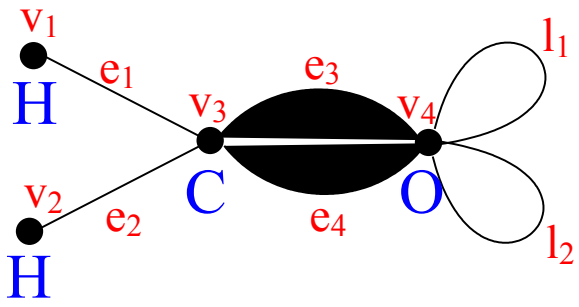
$L = \{\{v_4, v_4\}, \{v_4, v_4\}\}$

$\varphi(v_1) = H, \varphi(v_2) = H,$

$\varphi(v_3) = C, \varphi(v_4) = O$

$\beta = \{C, O, H\}$

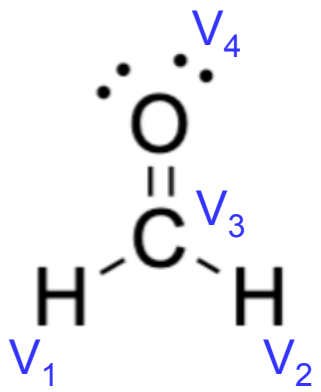
Figure of the molecular graph:



CEITEC

2D structure as a matrix

Molecule of formaldehyde:



Neighbourhood matrix of formaldehyde:

$$A = \begin{array}{c|cccc|c} & & & & & \\ & 0 & 0 & 1 & 0 & v_1 \\ & 0 & 0 & 1 & 0 & v_2 \\ & 1 & 1 & 0 & 2 & v_3 \\ & 0 & 0 & 2 & 4 & v_4 \\ & & & & & \\ \hline & v_1 & v_2 & v_3 & v_4 & \end{array}$$

Isomorfism of molecular graphs

- Two molecular graphs are isomorphic if they represent the same molecule.
- Two molecular graphs are isomorphic if they differ only in the indexing of the nodes.
- Two molecular graphs are isomorphic if:
  - Every node Ni of the graph G can be mapped to a node N´i of the graph G´.
  - Neighboring nodes of a node Ni can be mapped to neighboring nodes of a node N´i.

Isomorfism of molecular graphs

Graphs $G = (V, E, L, \varphi, \beta)$ and $G' = (V', E', L', \varphi', \beta)$ are isomorphic, if exist a bijective projection (permutation) $f: V \rightarrow V'$ with the following properties:

- if nodes $u, v \in V$ have n edges $\{u, v\}$ of graph G, then nodes $f(u)$, $f(v) \in V'$ have also n edges $\{f(u), f(v)\}$ in a graph G'
- if node $u \in V$ have n loops $\{u, u\}$ of graph G, then node $f(u) \in V'$ have n loops $\{f(u), f(u)\}$ graph G'
- the projection f maintain naming of the edges: $\varphi(u) = \varphi'(f(u))$ for each $u \in V$

Isomorphism and adjacency matrix

Molecular graphs determined by the neighborhood matrix:
$G = (V, A, \varphi, \beta)$ a $G' = (V', A', \varphi', \beta)$

Isomorphismus (=permutation) f: $V \rightarrow V'$

For a permutation of f, one can construct a permutation matrix P.

Note: The permutation matrix is formed from the unit matrix by permuting their row.

Then: $A = P^T A' P$

Time complexity

A problem is **NP-complete** if an algorithm with worst polynomial time complexity cannot be constructed for it.

Examples:

- Algorithms solvable in polynomial time belong, for example, to the complexity classes: $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, etc.
- NP-complete algorithms belong, for example, to the complexity classes: $O(2^n)$, $O(n!)$, $O(n^n)$, etc.

Isomorphism - time complexity

- The problem of isomorphism of general graphs is probably (not proven, but all indications are :-) among **NP-complete problems**.
- **How we can solve it?:**
  - Brute force
  - Improvements:
    - backtracking
    - Classes of edges
  - Limited classes of graphs:
    - planar graphs

Algorithms: brute force

- **Description:**

  For each permutation f: $V \rightarrow V'$

  test if it is an isomorfismus.

- **Complexity:**

  Sets V and V′ have the same number of nodes (n).

  => it exist n! permutations f: $V \rightarrow V'$

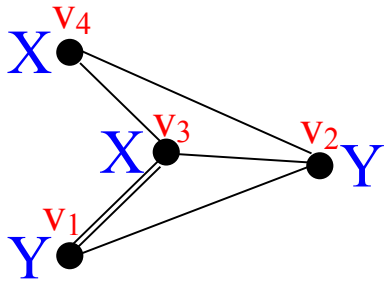
  => the algorithm has complexity in the class **O(n!)**

Algorithm: backtracking

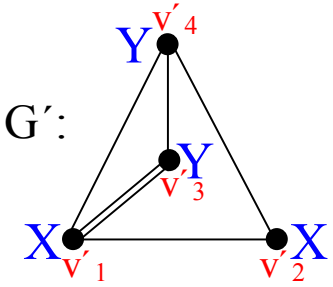
- Streamlining the brute force method
- It does not deal with all views. Adds an additional vertex only to partial views that satisfy the isomorphism condition.
- In the worst case, this method also has factorial complexity, but for common cases the computation is much shorter than using brute force.

Brute force X backtracking

Molecular graph G:



Molecular graph G´:



Brute force:

v_1, v_2, v_3, v_4

-> v´1, v´2, v´3, v´4;
-> v´1, v´3, v´2, v´4;
-> v´1, v´4, v´2, v´3;
-> v´1, v´2, v´4, v´3;
-> v´1, v´3, v´4, v´2;
-> v´1, v´4, v´3, v´2
-> v´2, v´1, v´3, v´4;
-> v´2, v´3, v´1, v´4;
-> v´2, v´4, v´1, v´3;
-> v´2, v´1, v´4, v´3;
-> v´2, v´3, v´4, v´1;
-> v´2, v´4, v´3, v´1
-> v´3, v´1, v´2, v´4;
-> v´3, v´2, v´1, v´4;
-> v´3, v´4, v´1, v´2;
-> v´3, v´1, v´4, v´2;
-> v´3, v´2, v´4, v´1;
-> v´3, v´4, v´2, v´1
-> v´4, v´1, v´2, v´3;
-> v´4, v´2, v´1, v´3;
-> v´4, v´3, v´1, v´2;
-> v´4, v´1, v´2, v´3;
-> v´4, v´2, v´3, v´1;
-> v´4, v´3, v´2, v´1

Bactracking:

v_1 -> v´1
v_1 -> v´2
v_1 -> v´3
v_1, v_2, v_3 -> v´3, v´1, v´4
v_1, v_2, v_3 -> v´3, v´4, v´1
v_1, v_2, v_3, v_4 -> v´3, v´4, v´1, v´2
v_1 -> v´4

Brute force:

24 iterations (4!)

Backtracking:

7 iterations

CEITEC

Algorithm: Division of nodes into classes

Classing criteria:

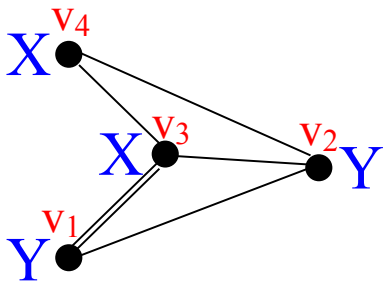
- atom rating
- degree of peak

Principle:

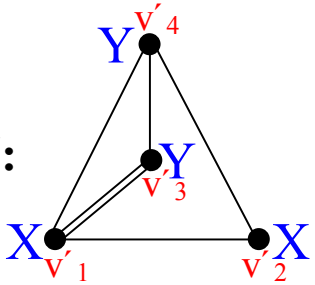
- Vertices of a graph G belonging to a certain class are mapped only to vertices of the graph G′ belonging to the same class.
- Note: A finer class partitioning can also be used:
  - In addition to the rank and degree of an atom, we take into account the rank and degree of its neighbors in the partitioning.
  - Divide the nodes according to the number of single, double and triple bonds that a given bond forms.

Example: Division of nodes into classes

Molecular graph G:



Molecular graph G´:



Classes:

$T_1 = \{v_4\}$ (name X, degree 2)
$T_2 = \{v_3\}$ (name X, degree 4)
$T_3 = \{v_1, v_2\}$ (name Y, degree 3)

$T'_1 = \{v'_2\}$ (name X, degree 2)
$T'_2 = \{v'_1\}$ (name X, degree 4)
$T'_3 = \{v'_3, v'_4\}$ (name Y, degree 3)

Principle:

We can project members of T_i only to members of T'_i

It exists only 2 possible projections:

$v_1, v_2, v_3, v_4 \rightarrow v'_3, v'_4, v'_1, v'_2$

~~$v_1, v_2, v_3, v_4 \rightarrow v'_4, v'_3, v'_1, v'_2$~~

CEITEC

Algorithm: planar graphs

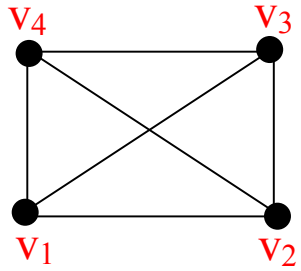
Definition: A graph is called planar if it is possible to create a planar drawing of it.

A graph drawing is a procedure that assigns to each vertex of the graph a point of the plane and assigns to each edge {u, v} a continuous simple arc that connects the points assigned to vertices u and v.

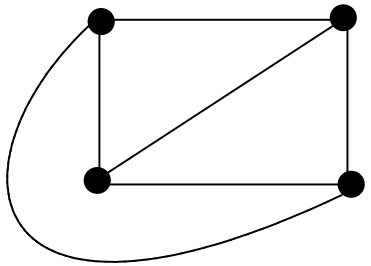
Planar drawing = A drawing that is made such that two arcs have at most 1 point in common. And only if this common point corresponds to the vertex from which both edges originate.

Planar graphs

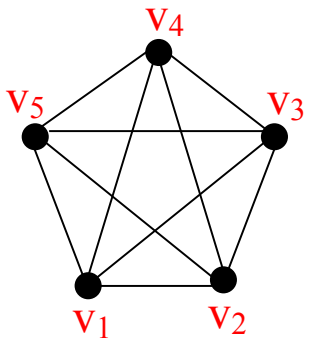
Planar graph G:



Planar drawing of graph G:

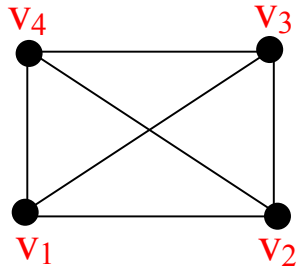


Graph G´:

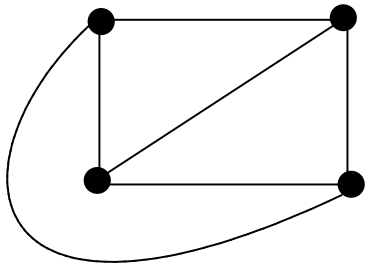


Planar graphs

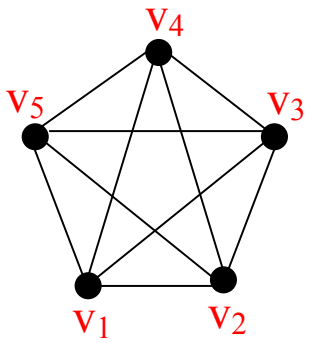
Planar graph G:



Planar drawing of graph G:



Graph G´:



For G´ does not exist any planar drawing

=>

G´ is not planar

Algorithm: planar graphs

The class of planar graphs is a subclass of general graphs.

In chemistry, a large fraction of molecules have planar graphs.

Exceptions: zeolites, fullerenes, some bioorganic substances (alkaloids, hormones, ...), more complex polymers and biopolymers.

Determine whether the graph is planar:

- An algorithm to find its planar drawing; O(n)
- Isomorphism of planar graphs:
  - Hopcroft and Tarjan's algorithm; O(n log n)

=> Isomorphism of planar graphs is not an NP-complete problem :-)

Isomorphismus: application in chemistry

Isomorphic graphs represent the same molecules.

Can be applied for finding of identical molecules in databases

Input: molecule graph

Output: information about the input molecule (or report that the molecule was not found

Procedure: search for isomorphism between the input molecule and database elements

Disadvantage: High time complexity. Finding a molecule of size N in a database with M molecules: O(M.N!)

Automorphism

= isomorphism of the graph on itself

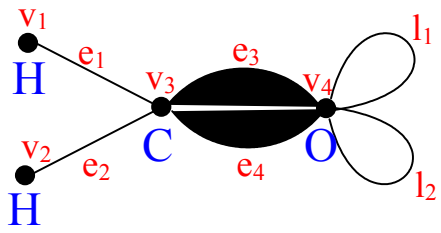
= symmetry operations on molecular graphs that preserve the graph topology (= do not change the adjacency matrix)

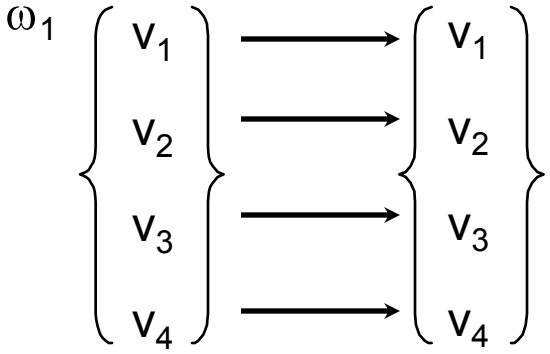
Topological equivalence:

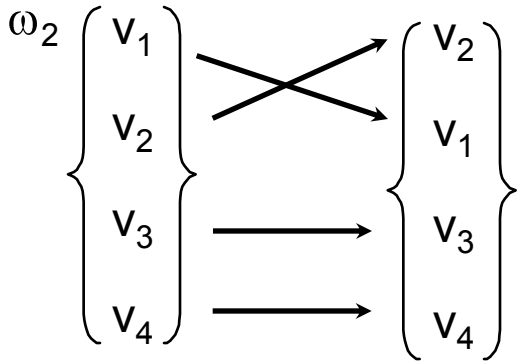
A pair of vertices v_i and v_i is topologically equivalent if there exists an automorphism ω such that one vertex maps to the other: $\omega(v_i) = v_j$

Authomorphism: examples

Molecule of formaldehyde has 2 authomorphisms:







Description of automorphisms (symetry operations):

ω_1 identity

ω_2 axis symmetry (the axis passes through the nodes v_3 and v_4)

Nodes v_1 and v_2 are **topologically equivalen**.

Automorphism and neighborhood matrix

Molecular graph determined by the neighborhood matrix:

$G = (V, A, \varphi, \beta)$

Automorphism (=permutation) ω: $V \rightarrow V$

The permutation ω is described by a matrix P.

Then holds:

$A = P^T A P$

Automorphism as a group

Set of all automorphisms:

$Aut = \{\omega_1, \omega_2, \ldots\}$, where $\omega: V \rightarrow V$

Operation of composition of projections:

$o: (\omega, \omega) \rightarrow \omega$

Grup of automorphisms:

$G_{aut} = (Aut, o)$

A group of automorphisms can be isomorphic to some point group of spatial symmetry.

For formaldehyde:

$G_{aut} = (\{\omega_1, \omega_2\}, o)$

Indexing of molecular graph nodes

Molecular graph: $G = (V, E, L, \varphi, \beta)$

Indexing the nodes of the molecular graph G:

bijection $\tau: V \rightarrow I$

I is an index set: $I = \{1, 2, \ldots, |V|\}$

Thus, each node is assigned a natural number (index).

Canonical indexing of molecular graph nodes

Canonical indexing = indexing that fulfills the following conditions:

For a molecular graph MG, indexing I can be generated algorithmically (using some alg_CI algorithm)

Consider arbitrary indexings I_1 and I_2. For the canonical indexings $CI_1 = alg\_CI(MG, I_1)$ and $CI_2 = alg\_CI(MG, I_2)$ must hold:

$i(CI_1) = i(CI_2) \quad \Leftrightarrow$

atom having index $i(CI_1)$ is topologically equivalent with atom having index $i(CI_2)$

Number of possible indexing

A molecular graph has n nodes => there are n! different ways of indexing this graph.

If |Aut| > 1 (there exist an automorphism other than identity),

then exit only n! / |Aut| possible indexing

Example - formaldehyde:

number of atoms: 4

number of authomorphisms: 2

number of indexing: 4! / 2 = 12

Canonical indexing and neighborhood matrix

For two canonically indexed molecular graphs $G = (V, A, \varphi, \beta)$ and $G' = (V', A', \varphi', \beta)$ hold:

$A = A'$ <=>

graphs represent the same molecule <=>

graphs are isomorphic

Canonical indexing - application

Searching databases of molecules

Database: contains canonically indexed molecules

Input: canonically indexed molecule

Procedure: compares the adjacency matrices of the input molecule and the molecules in the database

(Comparing matrices of size N has complexity $O(N^2)$.)

Advantage: Significantly less time complexity than if we search for isomorphism for each pair (input molecule, database molecule)

Finding a molecule of size N in a database with M molecules:

- Using isomorphism: O(M.N!)
- Using canonical indexing: $O(M.N^2)$

Canonical indexing - algorithms

The "brute force" solution:

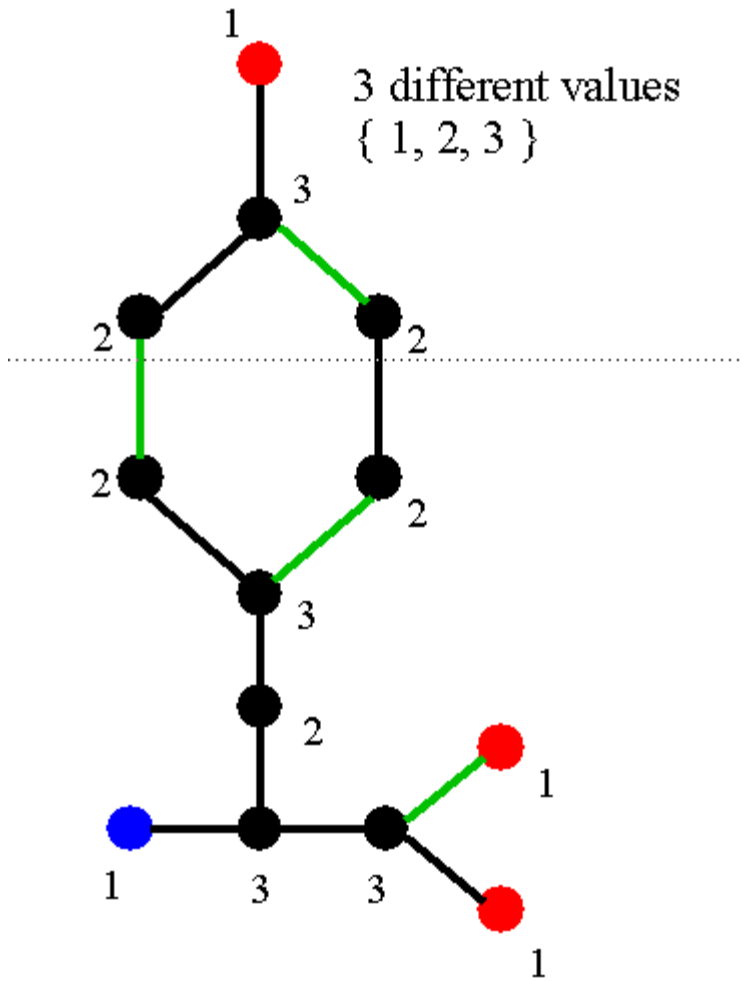
- a large number of such algorithms can be created
- For example: for each indexing, determine the numerical value that the linear notation of the neighborhood matrix takes.
- Then choose the indexing with the highest numerical value.
- advantage: no error cases
- disadvantage: complexity O(n!) => not used in practice

Canonical indexing - Morgan's algorithm

- First algorithm for canonical indexing (1965)
- Most of the others work on a similar principle
- Notes:
  - Morgan's algorithm is based only on the topology of the molecule, ignoring multiple bonds, loops, and the evaluation of vertices with chemical tags.
  - This is not real limitation:
    - We can determine these data from the topology.
    - For example: degree of node (atom) + atom bonding => number of multiple edges

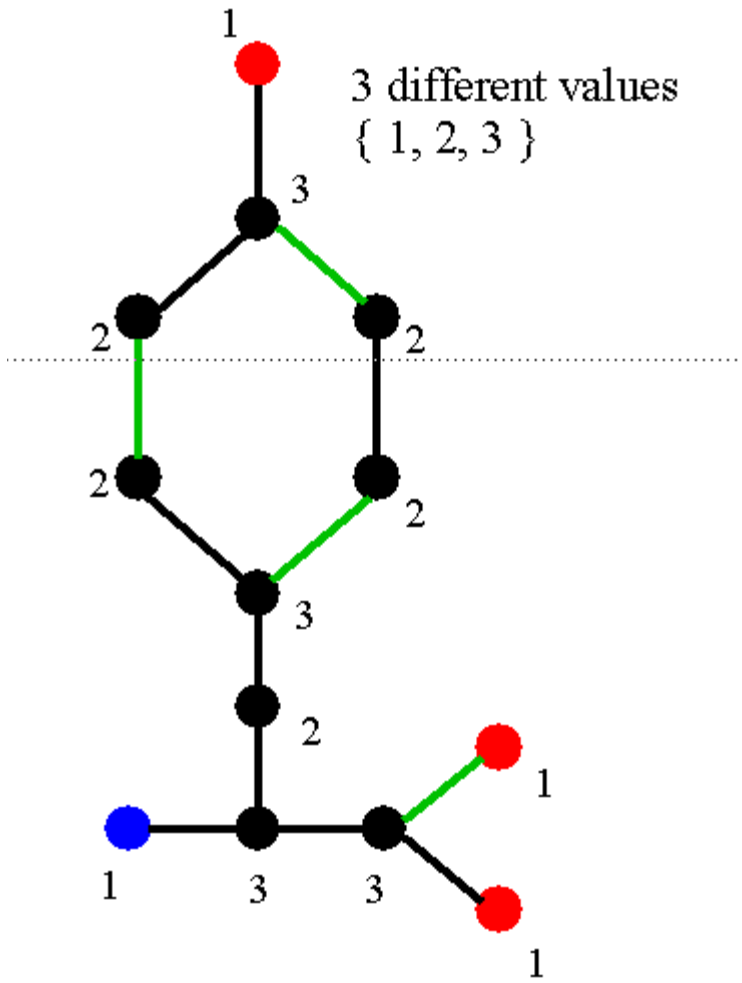
Morgan's algorithm

- Grade each node by its degree
- Determine the number of distinct values



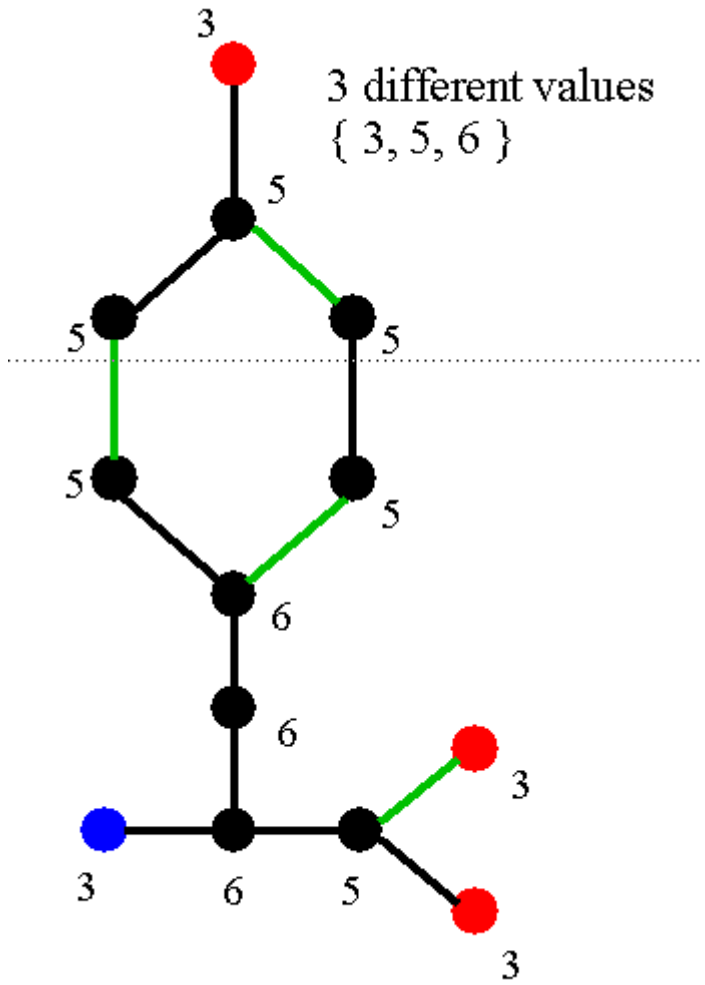
Morgan's algorithm

- Grade the node by the sum of the scores of the neighbouring nodes
- Determine the number of different values
- Repeat the above two points until the number of distinct values changes



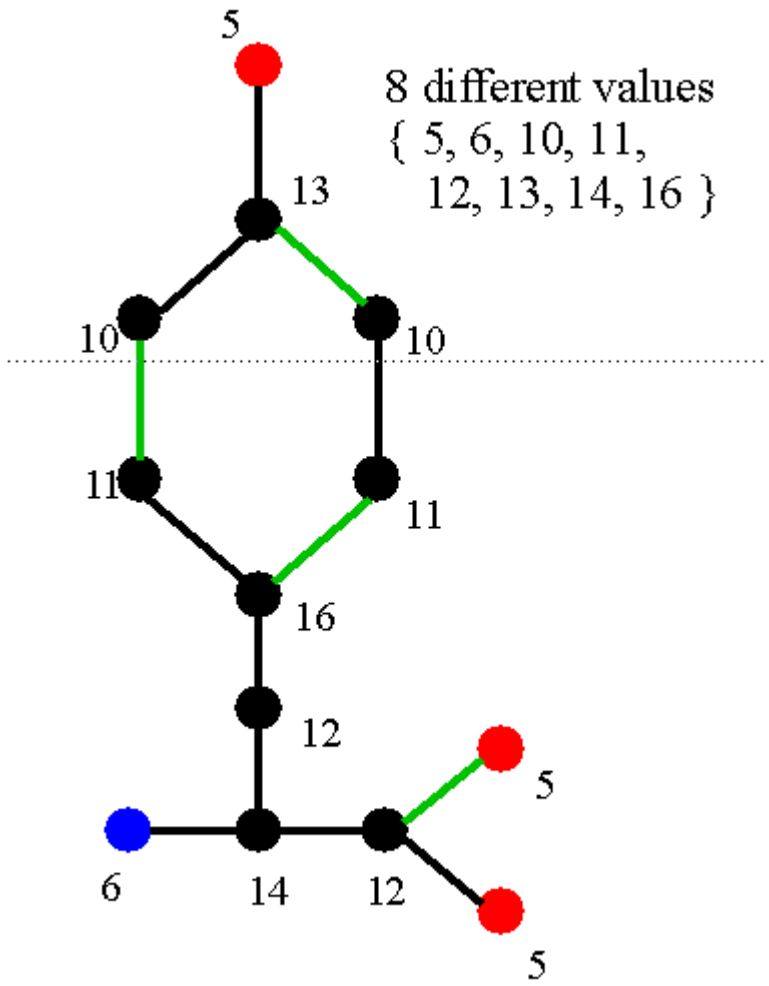
Morgan's algorithm

- Grade the node by the sum of the scores of the neighbouring nodes
- Determine the number of different values
- Repeat the above two points until the number of distinct values changes



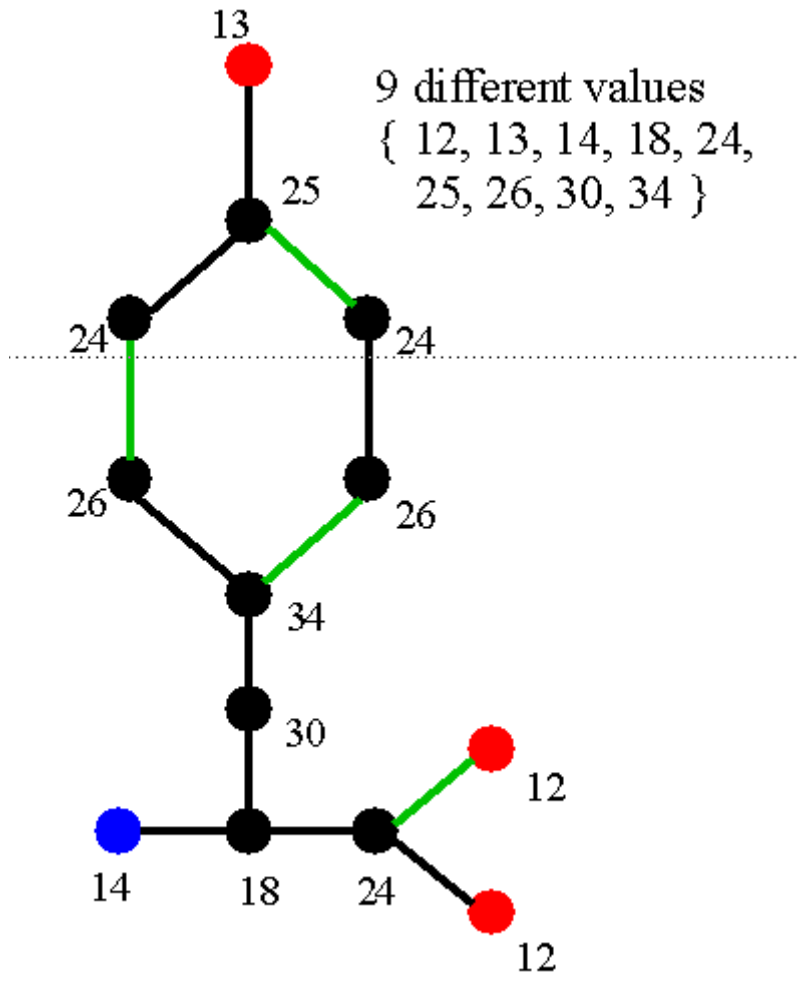
Morgan's algorithm

- Grade the node by the sum of the scores of the neighbouring nodes
- Determine the number of different values
- Repeat the above two points until the number of distinct values changes



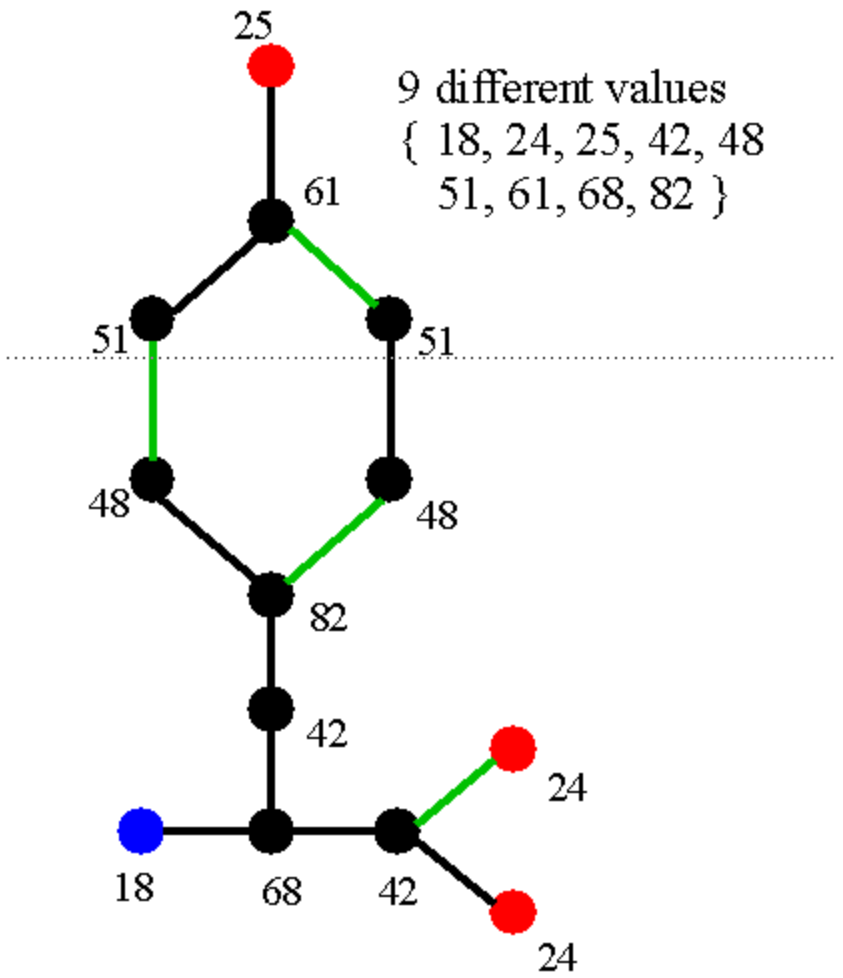
Morgan's algorithm

- Grade the node by the sum of the scores of the neighbouring nodes
- Determine the number of different values
- Repeat the above two points until the number of distinct values changes



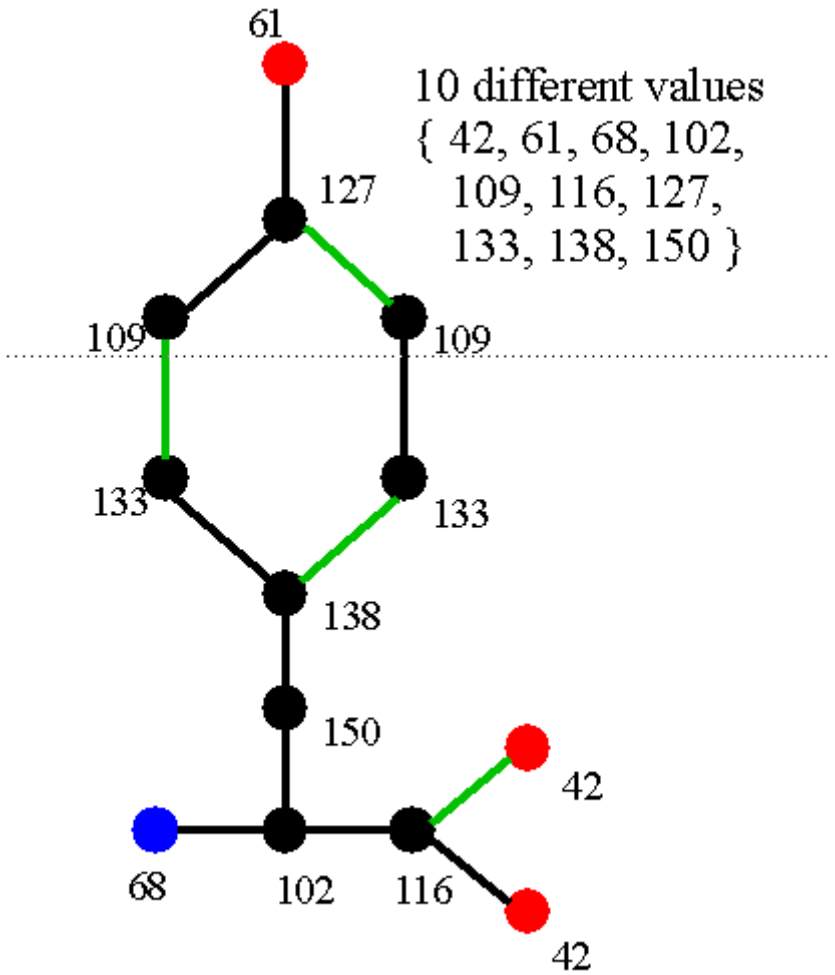
Morgan's algorithm

- Grade the node by the sum of the scores of the neighbouring nodes
- Determine the number of different values
- Repeat the above two points until the number of distinct values changes



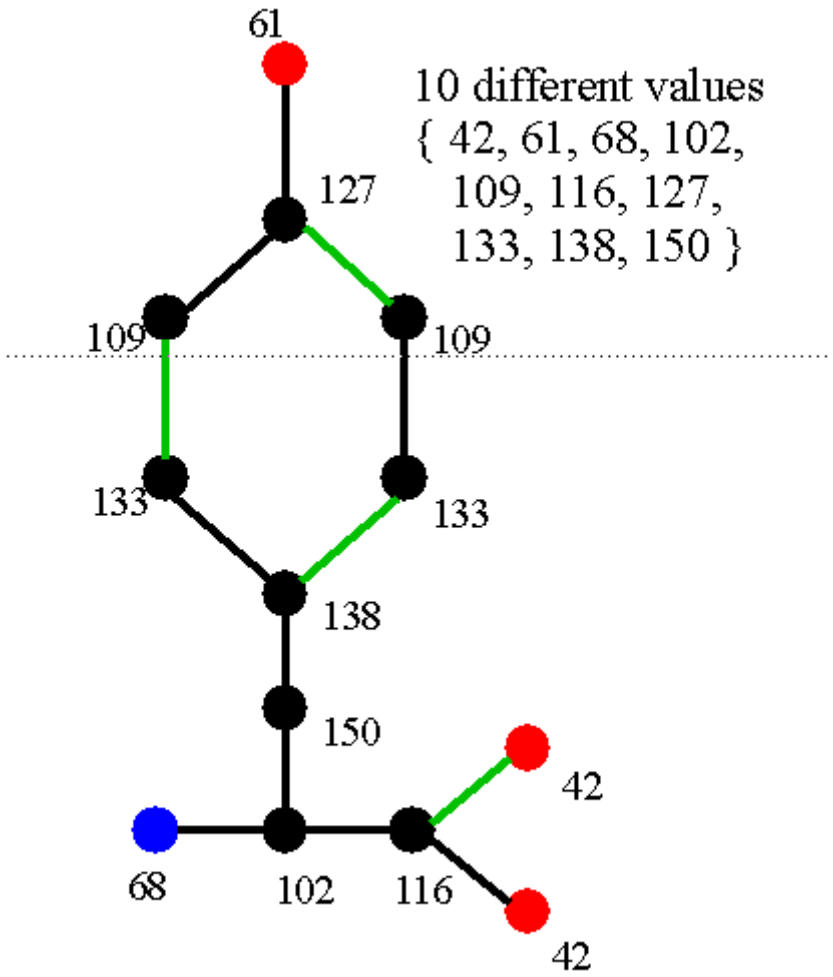
Morgan's algorithm

- Grade the node by the sum of the scores of the neighbouring nodes
- Determine the number of different values
- Repeat the above two points until the number of distinct values changes



Morgan's algorithm

- Most nodes have a different score
- Mark as 1 the node with the highest score
- Mark its neighbours in order of their scores

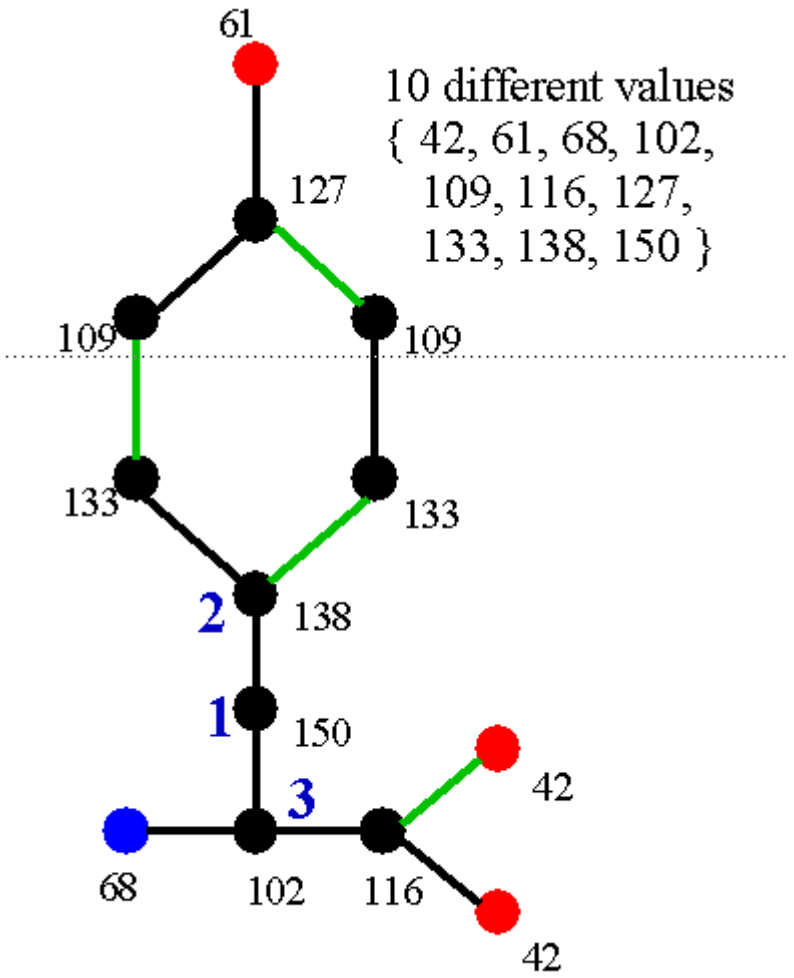


Morgan's algorithm

The remaining neighbours of node 2 have the same score

- choose the one that is connected by multiple edges (C=C is green)
- it is also possible to consider the mass of the atoms (for different ones)
- when the atoms are equi-valent, choose any

Continue until all atoms are marked

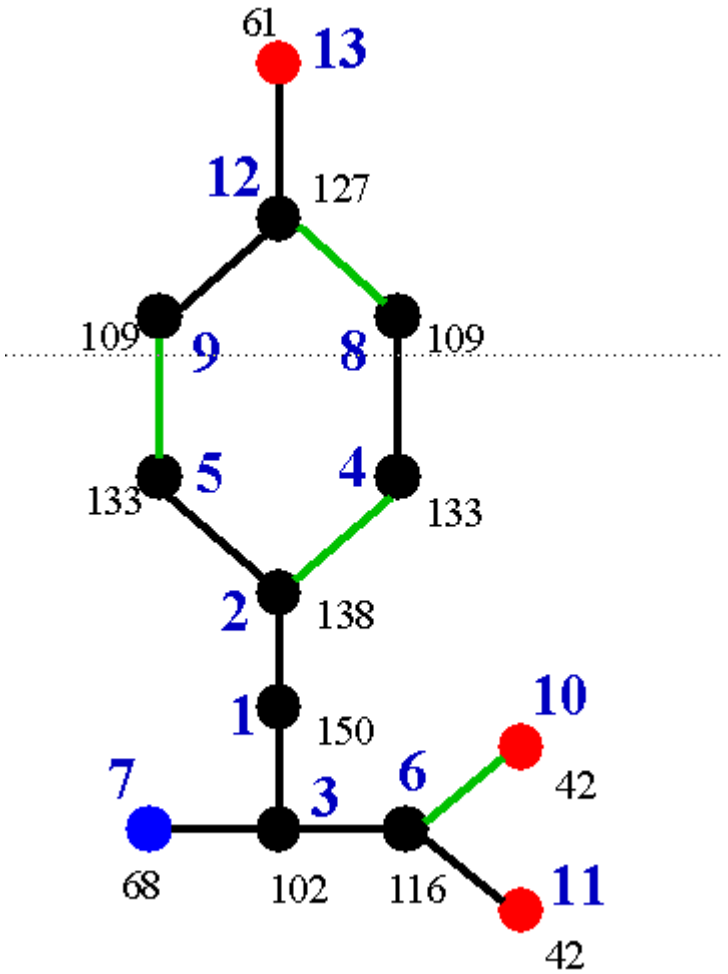


Morgan's algorithm

The remaining neighbours of node 2 have the same score

- choose the one that is connected by multiple edges (C=C is green)
- it is also possible to consider the mass of the atoms (for different ones)
- when the atoms are equi-valent, choose any

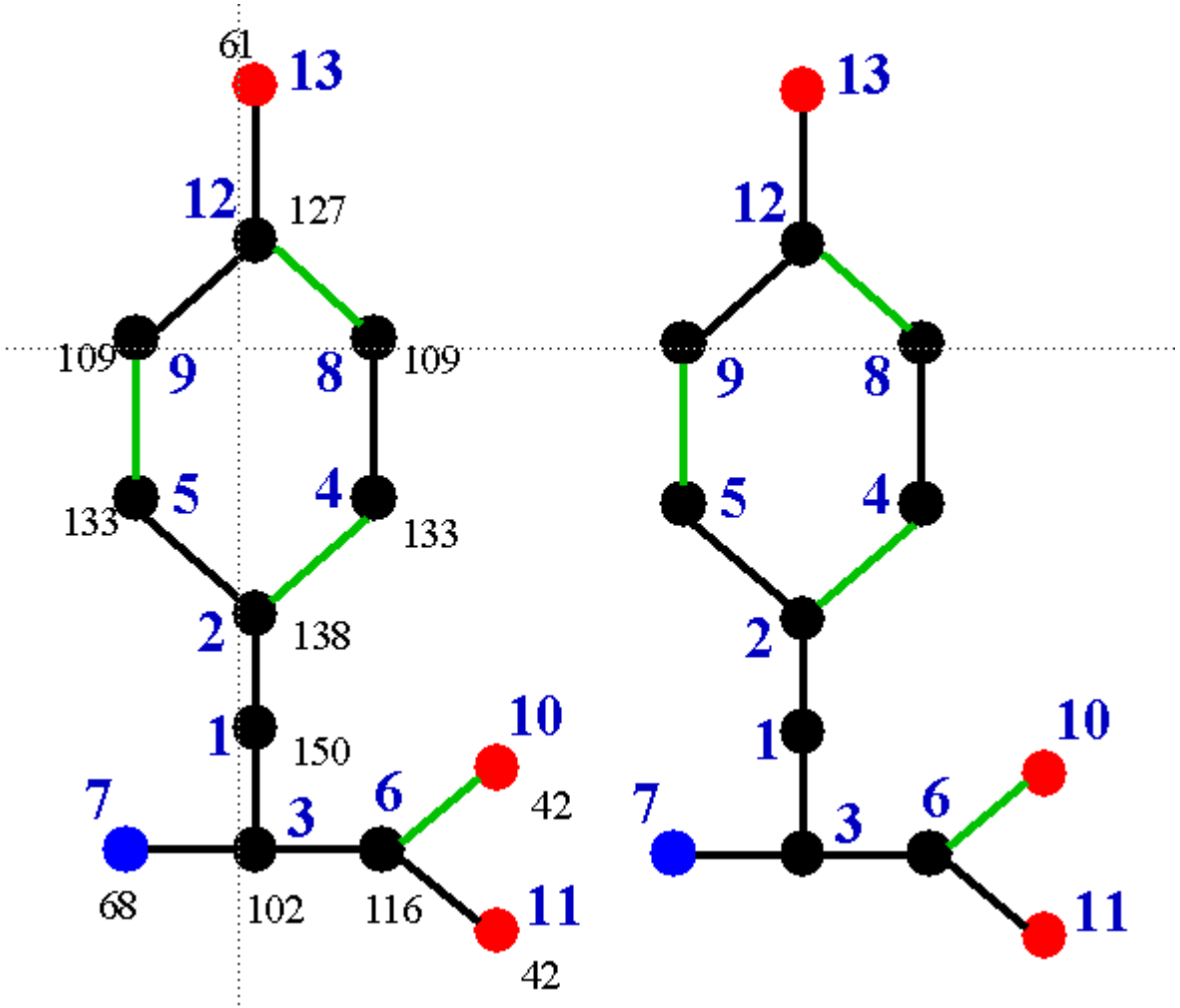
Continue until all atoms are marked



Morgan's algorithm

After finishing of the algorithm:

Canonically
indexed graph.



Morgan's algorithm - evaluation

Advantages of the algorithm:

- Low complexity of the algorithm: $O(n^2)$.

Disadvantages of the algorithm:

- The algorithm may in some cases index incorrectly (assign the same index to atoms that are not chemically equivalent)

Canonical indexing – example of other algorithms

Shelley-Munk algorithm:

- An extension of Morgan's original idea
- takes into account the properties of the neighbors of the atom being evaluated
- complexity: O(n2)

Thank you for your attention