Friedrich-Alexander-Universität
Naturwissenschaftliche Fakultät

43384 – Digital Alchemy

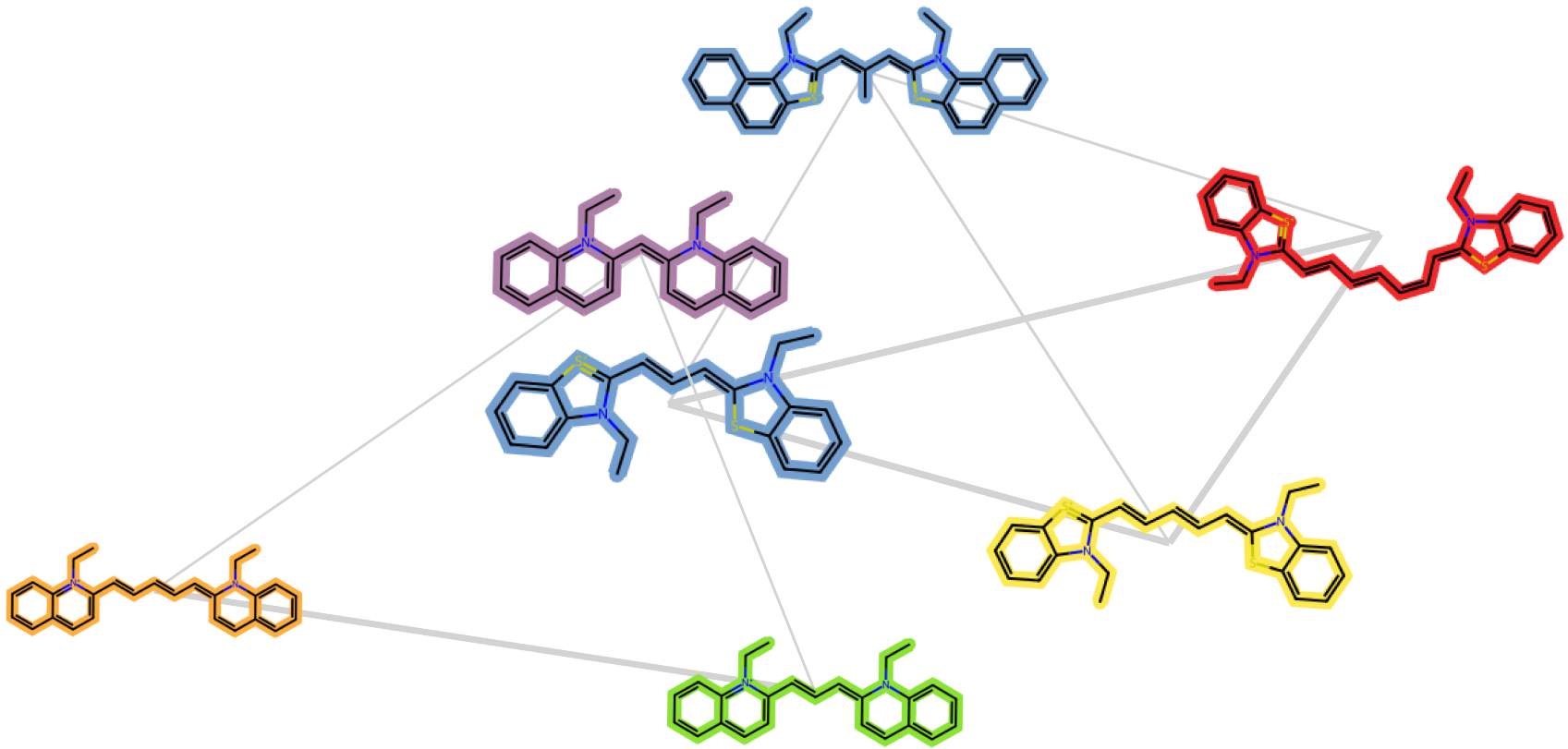
Unit 04 – Searching Chemical Structures

Prof. Dr. Carolin Müller

November 04, 2025

Searching Chemical Structures

Substructure and Similarity Relation of Molecules

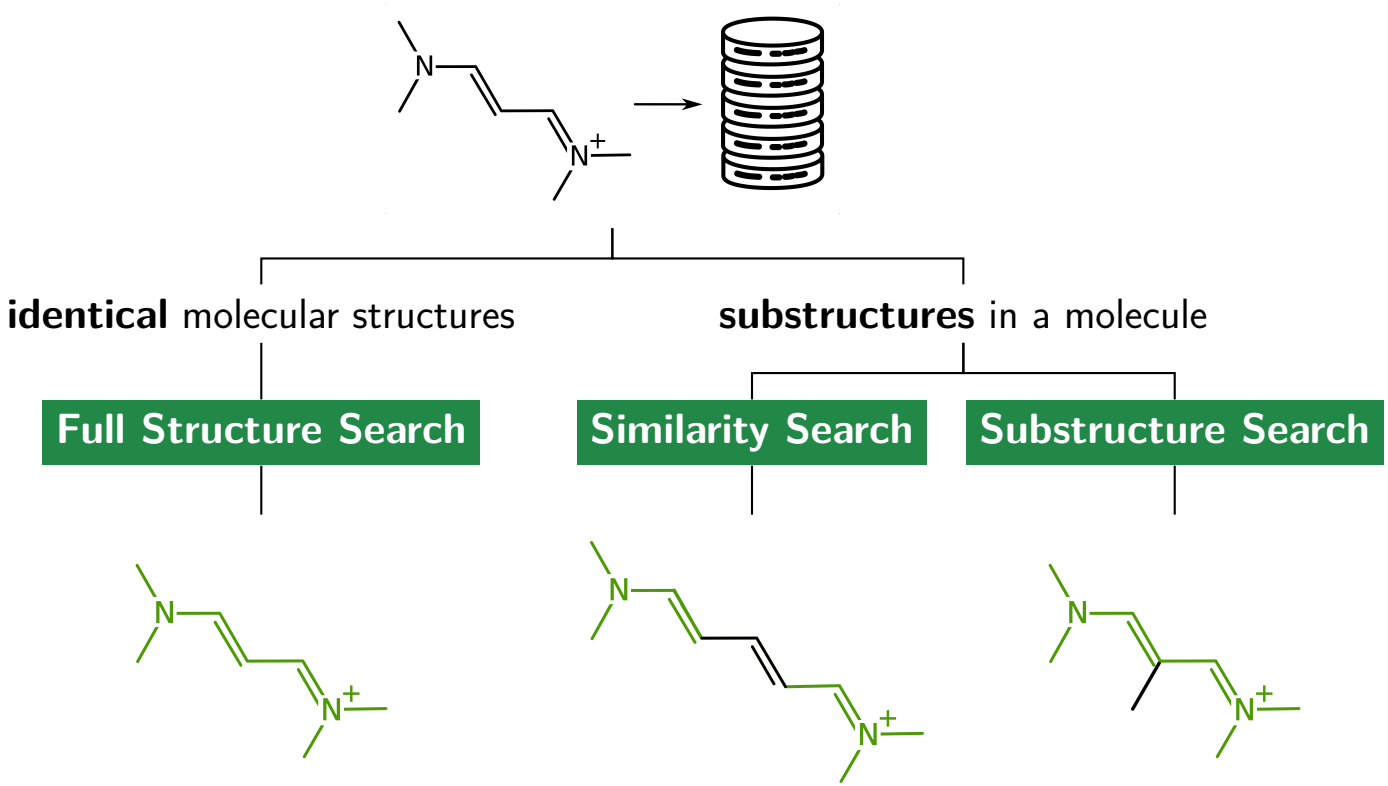


Searching Chemical Structures

Classical Algorithmic Chemoinformatics Problems

Searching Chemical Structures

Databases are indispensable tools in modern chemical research. Thus, the search of structural information in these databases is an important task, *e.g.*, for *molecular design* and *property prediction*.



Identification of Molecules

Classical Algorithmic Chemoinformatics Problems

- Identifying **identical** molecular structures

- Identifying **substructures** in a molecule

Identification of Molecules

Classical Algorithmic Chemoinformatics Problems

- Identifying **identical** molecular structures
  - **Graph isomorphism** problem: Two molecules are considered the same if they have the same molecular graph
- Identifying **substructures** in a molecule

Isomorphism

Two graphs, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ with the same number of elements are called isomorphic if they are mapped with a function $f : G_1(V_1, E_1) \rightarrow G_2(V_2, E_2)$ in such a way that any two adjacent vertices, v_x and v_y of G_1 are adjacent in G_2 if $f(v_x)$ and $f(v_y)$.

Two graphs are isomorphic (the same) if there is an edge-preserving one-to-one correspondence between their vertices.

Identification of Molecules

Classical Algorithmic Chemoinformatics Problems

- Identifying **identical** molecular structures
  - **Graph isomorphism** problem: Two molecules are considered the same if they have the same molecular graph
- Identifying **substructures** in a molecule
  - **Subgraph isomorphism** problem: A fragment is part of a molecule (graph G) if it corresponds to a subset of atoms and their corresponding bonds, *i.e.*, if the fragment graph F is a subgraph of G, *i.e.*, if F is isomorphic to a subgraph of G

Isomorphism

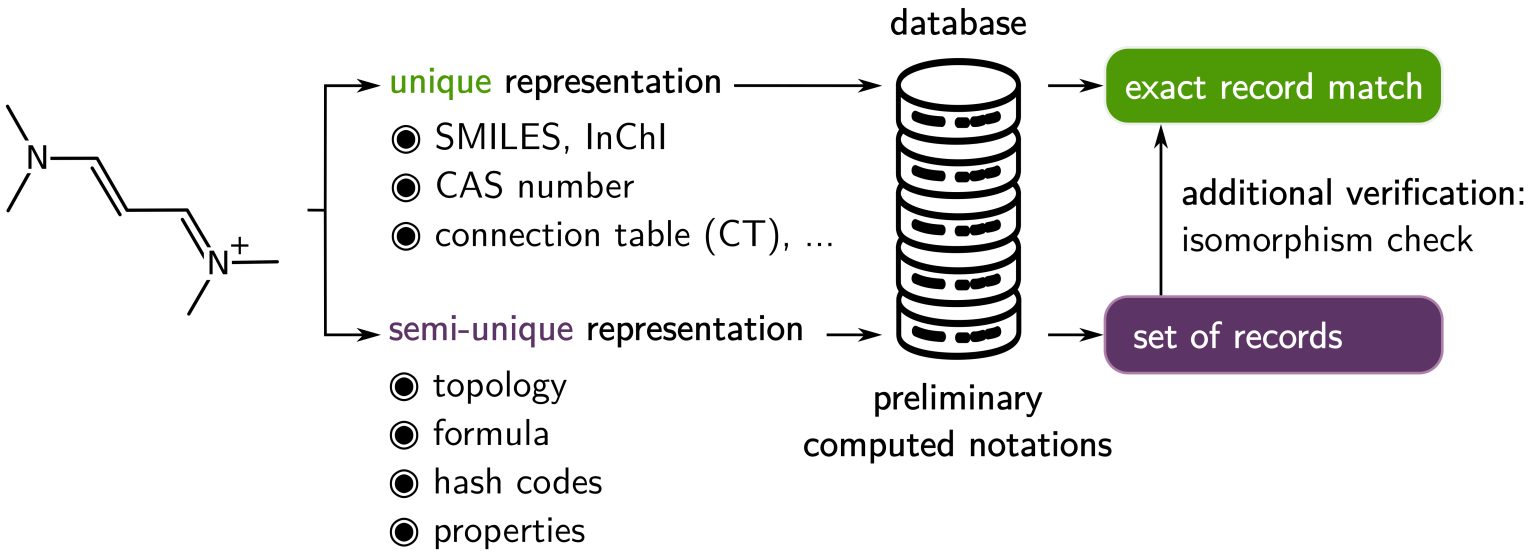
Two graphs, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ with the same number of elements are called isomorphic if they are mapped with a function $f : G_1(V_1, E_1) \rightarrow G_2(V_2, E_2)$ in such a way that any two adjacent vertices, v_x and v_y of G_1 are adjacent in G_2 if $f(v_x)$ and $f(v_y)$.

Two graphs are isomorphic (the same) if there is an edge-preserving one-to-one correspondence between their vertices.

Full Structure Search

Perception of complete chemical structures

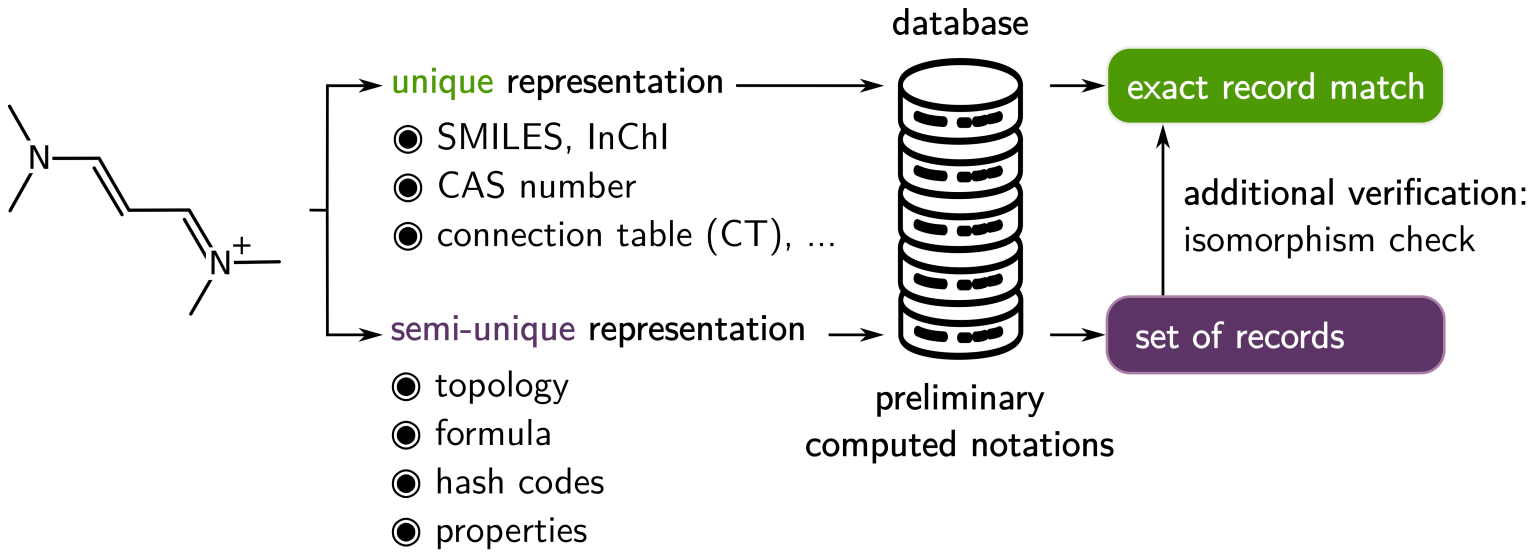
The perception of **complete structures** is dependent on their representation. Structures must be **represented uniquely** and usually compactly by an indexing key or may only be used for indexing purposes in order to retrieve the exact database record number.



Full Structure Search

Perception of complete chemical structures

The perception of **complete structures** is dependent on their representation. Structures must be **represented uniquely** and usually compactly by an indexing key or may only be used for indexing purposes in order to retrieve the exact database record number.



Problem: How to uniquely identify molecular structures?

a) Canonical Representations

Unique labeling of molecular structures

Problem: How to uniquely identify molecular structures?

Labeling of Molecules

In principal, a structure with n atoms can be labeled in $n!$ different ways
(*e.g.*, $n!$ different adjacency matrices or bond tables)

a) Canonical Representations

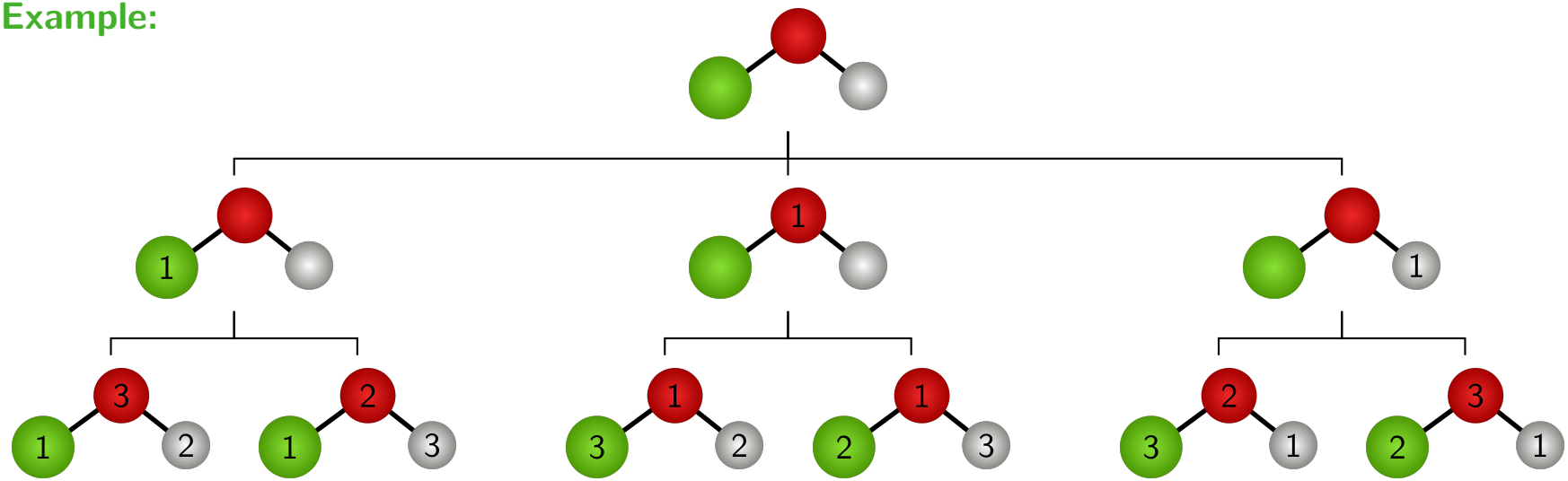
Unique labeling of molecular structures

Problem: How to uniquely identify molecular structures?

Labeling of Molecules

In principal, a structure with n atoms can be labeled in $n!$ different ways
(*e.g.*, $n!$ different adjacency matrices or bond tables)

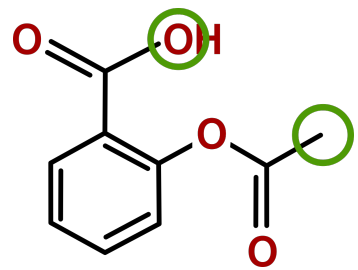
Example:



a) Canonical Line Notations

Idea of canonicalization

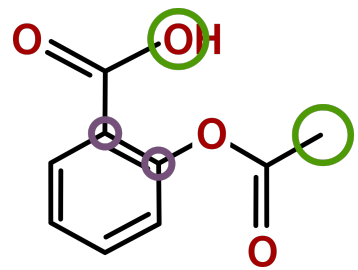
- Representations like Smiles and InChI depend on the order of traversal of atoms, *i.e.*
  - **What** is the first atom of the representation?



a) Canonical Line Notations

Idea of canonicalization

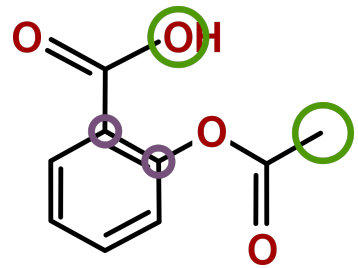
- Representations like Smiles and InChI depend on the order of traversal of atoms, *i.e.*
  - **What** is the first atom of the representation?
  - At each branch point: **Which** path to follow?



a) Canonical Line Notations

Idea of canonicalization

- Representations like Smiles and InChI depend on the order of traversal of atoms, *i.e.*
  - **What** is the first atom of the representation?
  - At each branch point: **Which** path to follow?
- Idea for canonicalization: Priorities based on topology

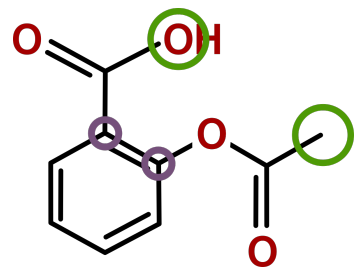


a) Canonical Line Notations

Idea of canonicalization

- Representations like Smiles and InChI depend on the order of traversal of atoms, *i.e.*
  - **What** is the first atom of the representation?
  - At each branch point: **Which** path to follow?

- Idea for canonicalization: Priorities based on topology
  - Assign initial invariants to atoms, encoding local information of the atoms
    - Number of neighbors
    - Atom type
    - Number of hydrogens

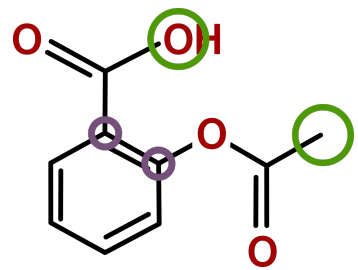


a) Canonical Line Notations

Idea of canonicalization

- Representations like Smiles and InChI depend on the order of traversal of atoms, *i.e.*
  - **What** is the first atom of the representation?
  - At each branch point: **Which** path to follow?

- Idea for canonicalization: Priorities based on topology
  - Assign initial invariants to atoms, encoding local information of the atoms
    - Number of neighbors
    - Atom type
    - Number of hydrogens
  - Update invariant based on neighboring invariant
  - Repeat until:

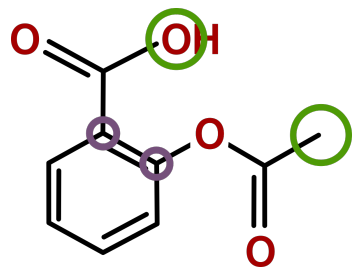


a) Canonical Line Notations

Idea of canonicalization

- Representations like Smiles and InChI depend on the order of traversal of atoms, *i.e.*
  - **What** is the first atom of the representation?
  - At each branch point: **Which** path to follow?

- Idea for canonicalization: Priorities based on topology
  - Assign initial invariants to atoms, encoding local information of the atoms
    - Number of neighbors
    - Atom type
    - Number of hydrogens
  - Update invariant based on neighboring invariant
  - Repeat until:
    - atoms are disambiguated
    - no more atoms can be differentiated

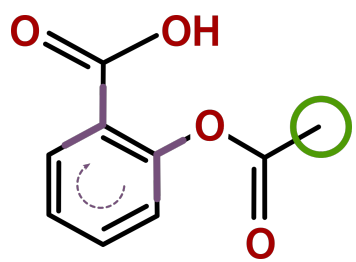


a) Canonical Line Notations

Idea of canonicalization

- Representations like Smiles and InChI depend on the order of traversal of atoms, *i.e.*
  - **What** is the first atom of the representation?
  - At each branch point: **Which** path to follow?

- Idea for canonicalization: Priorities based on topology
  - Assign initial invariants to atoms, encoding local information of the atoms
    - Number of neighbors
    - Atom type
    - Number of hydrogens
  - Update invariant based on neighboring invariant
  - Repeat until:
    - atoms are disambiguated
    - no more atoms can be differentiated
  - Priorities determine order of traversal



a) Canonical Line Notations

The Morgan Algorithm

Morgan Algorithm (1965)

Canonicalization algorithm to provide a biunique and invariant numbering of atoms in a molecular structure and to distingusih constitutionally equivalent atoms.

a) Canonical Line Notations

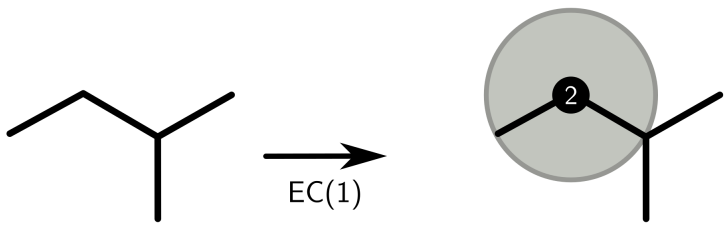
The Morgan Algorithm

Morgan Algorithm (1965)

Canonicalization algorithm to provide a biunique and invariant numbering of atoms in a molecular structure and to distingusih constitutionally equivalent atoms.

Steps of the algorithm:

1. Initialization
   - assign initial invariants through extended connectivities (ECs)



a) Canonical Line Notations

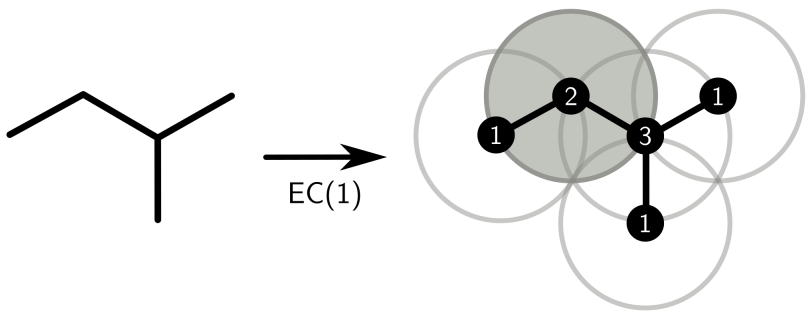
The Morgan Algorithm

Morgan Algorithm (1965)

Canonicalization algorithm to provide a biunique and invariant numbering of atoms in a molecular structure and to distingusih constitutionally equivalent atoms.

Steps of the algorithm:

1. Initialization
   - assign initial invariants through extended connectivities (ECs)



a) Canonical Line Notations

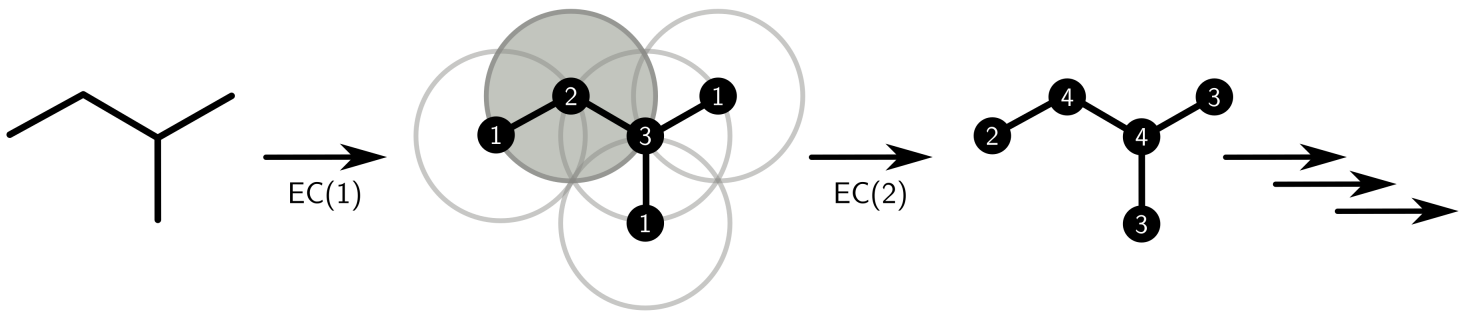
The Morgan Algorithm

Morgan Algorithm (1965)

Canonicalization algorithm to provide a biunique and invariant numbering of atoms in a molecular structure and to distingusih constitutionally equivalent atoms.

Steps of the algorithm:

1. Initialization
   - assign initial invariants through extended connectivities (ECs)
2. Relaxation process *via* ECs
   - sum EC values of directly connected non-hydrogen atoms of the former sphere



a) Canonical Line Notations

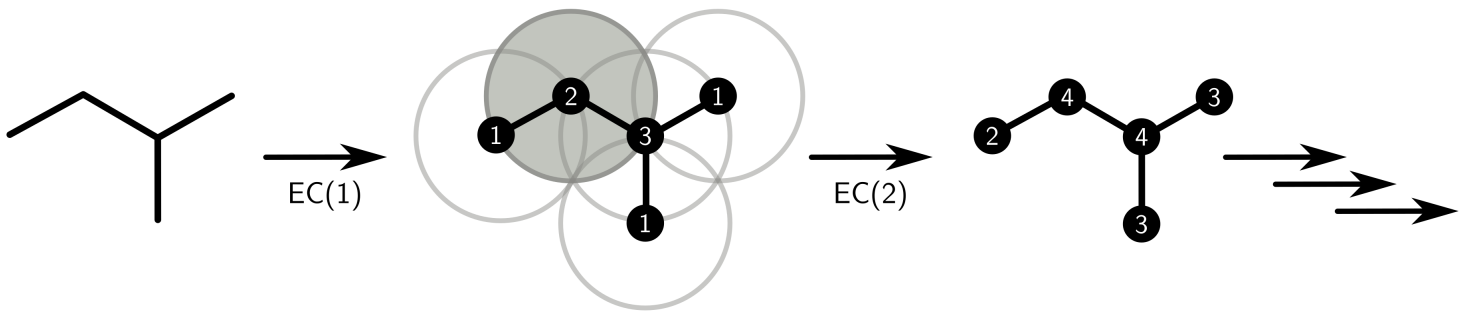
The Morgan Algorithm

Morgan Algorithm (1965)

Canonicalization algorithm to provide a biunique and invariant numbering of atoms in a molecular structure and to distingusih constitutionally equivalent atoms.

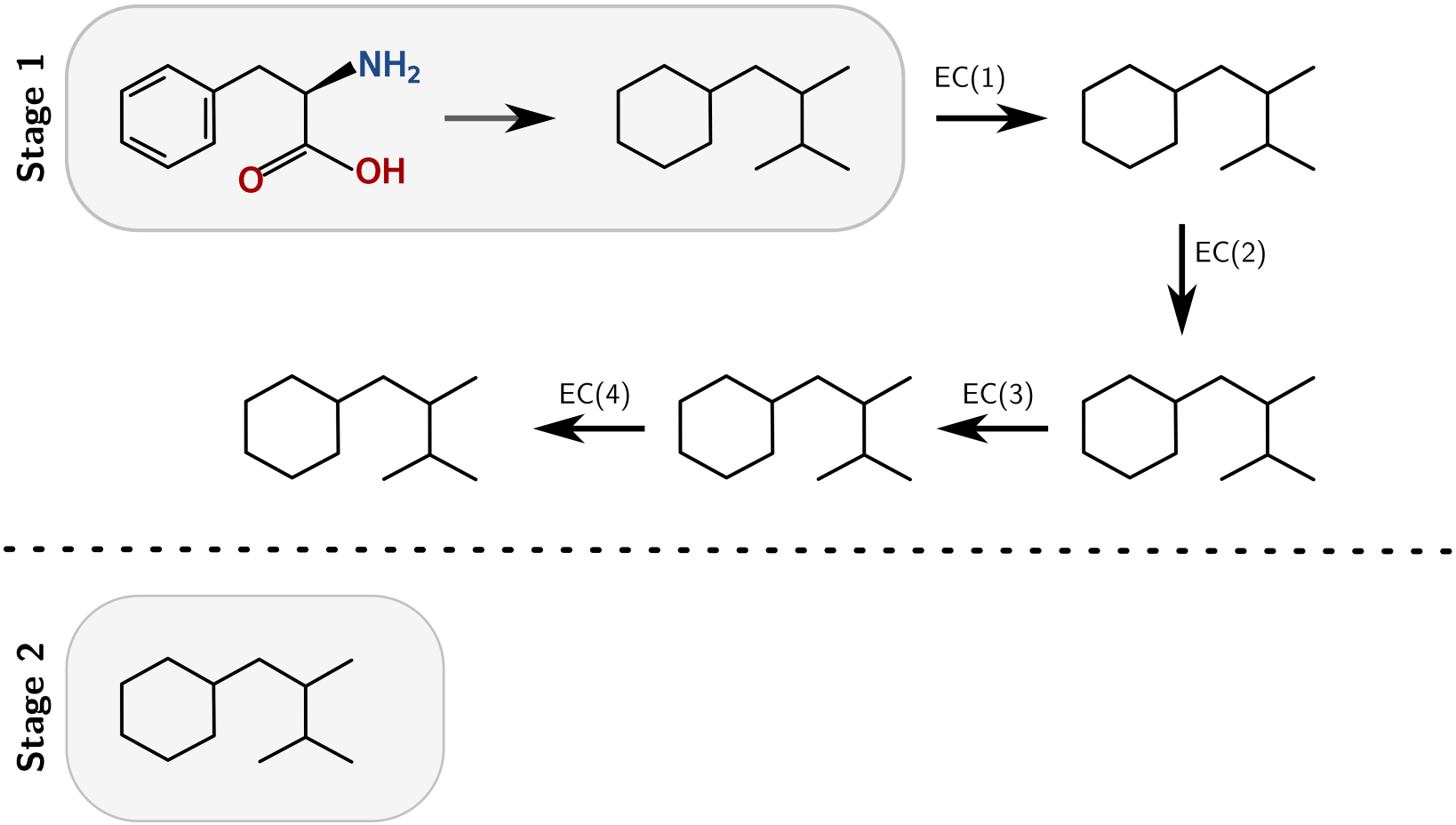
Steps of the algorithm:

1. Initialization
   - assign initial invariants through extended connectivities (ECs)
2. Relaxation process *via* ECs
   - sum EC values of directly connected non-hydrogen atoms of the former sphere
3. Assign invariant sequence of numbers to atoms



a) Canonical Line Notations

Morgan Algorithm – Example



b) Hash Codes

Molecular Bit Vectors (1D Descriptors)

Hash Code

A hash code is a value generated by applying a hash function to input data, typically producing a fixed-size integer.

RDKit Example:

```
1  from rdkit import Chem
2  hash(Chem.rdMolHash.MolHash(Chem.MolFromSmiles("CN(C)/C=C/C=[N+](C)C"),
3     Chem.rdMolHash.HashFunction.CanonicalSmiles))
4  > 5991625681167528425
```

b) Hash Codes

Molecular Bit Vectors (1D Descriptors)

Hash Code

A hash code is a value generated by applying a hash function to input data, typically producing a fixed-size integer.

Properties:

- **access of elements in a data table**: no serial search ($\mathcal{O}(n)$) but specification of likeliest location of a searched element in a hash table ($\mathcal{O}(1)$)

RDKit Example:

```
1  from rdkit import Chem
2  hash(Chem.rdMolHash.MolHash(Chem.MolFromSmiles("CN(C)/C=C/C=[N+](C)C"),
3     Chem.rdMolHash.HashFunction.CanonicalSmiles))
4  > 5991625681167528425
```

b) Hash Codes

Molecular Bit Vectors (1D Descriptors)

Hash Code

A hash code is a value generated by applying a hash function to input data, typically producing a fixed-size integer.

Properties:

- **access of elements in a data table**: no serial search ($\mathcal{O}(n)$) but specification of likeliest location of a searched element in a hash table ($\mathcal{O}(1)$)
- **collision**: hash table can store multiple elements of same hash code computed by hashing function
- **size**: 32- or 64-bit hash codes

RDKit Example:

```
1  from rdkit import Chem
2  hash(Chem.rdMolHash.MolHash(Chem.MolFromSmiles("CN(C)/C=C/C=[N+](C)C"),
3     Chem.rdMolHash.HashFunction.CanonicalSmiles))
4  > 5991625681167528425
```

b) Hash Codes

Molecular Bit Vectors (1D Descriptors)

Hash Code

A hash code is a value generated by applying a hash function to input data, typically producing a fixed-size integer.

Properties:

- **access of elements in a data table**: no serial search ($\mathcal{O}(n)$) but specification of likeliest location of a searched element in a hash table ($\mathcal{O}(1)$)
- **collision**: hash table can store multiple elements of same hash code computed by hashing function
- **size**: 32- or 64-bit hash codes

RDKit Example:

```
1  from rdkit import Chem
2  hash(Chem.rdMolHash.MolHash(Chem.MolFromSmiles("CN(C)/C=C/C=[N+](C)C"),
3     Chem.rdMolHash.HashFunction.CanonicalSmiles))
4  > 5991625681167528425
```

b) Hash Codes

Molecular Bit Vectors (1D Descriptors)

Hash Code

A hash code is a value generated by applying a hash function to input data, typically producing a fixed-size integer.

Properties:

- **access of elements in a data table**: no serial search ($\mathcal{O}(n)$) but specification of likeliest location of a searched element in a hash table ($\mathcal{O}(1)$)
- **collision**: hash table can store multiple elements of same hash code computed by hashing function
- **size**: 32- or 64-bit hash codes

Applications:

- internal usage in software apps

RDKit Example:

```
1  from rdkit import Chem
2  hash(Chem.rdMolHash.MolHash(Chem.MolFromSmiles("CN(C)/C=C/C=[N+](C)C"),
3     Chem.rdMolHash.HashFunction.CanonicalSmiles))
4  > 5991625681167528425
```

b) Hash Codes

Molecular Bit Vectors (1D Descriptors)

Hash Code

A hash code is a value generated by applying a hash function to input data, typically producing a fixed-size integer.

Properties:

- **access of elements in a data table**: no serial search ($\mathcal{O}(n)$) but specification of likeliest location of a searched element in a hash table ($\mathcal{O}(1)$)
- **collision**: hash table can store multiple elements of same hash code computed by hashing function
- **size**: 32- or 64-bit hash codes

Applications:

- internal usage in software apps
- fast location of molecules in databases (full structure search), by generating structure hash-codes from:
  - molecular topology (graphs)
  - connection tables (foundation of CAS registry system)
  - ensemble (merge hash codes of atoms into bond and molecular hash codes)

RDKit Example:

```
1  from rdkit import Chem
2  hash(Chem.rdMolHash.MolHash(Chem.MolFromSmiles("CN(C)/C=C/C=[N+](C)C"),
3     Chem.rdMolHash.HashFunction.CanonicalSmiles))
4  > 5991625681167528425
```

Substructure Search

Perception of structural fragments

Substructure Search

Process of identifying parts of a given structure that are equivalent to a specified query substructure.

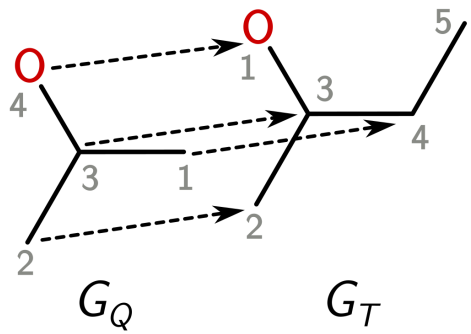
Substructure Search

Perception of structural fragments

Substructure Search

Process of identifying parts of a given structure that are equivalent to a specified query substructure.

In graph theoretical terms this refers to checking whether the query graph (G_Q) is isomorphic to a subgraph of the target graph (G_T). G_Q is isomorphic to a subgraph of G_T if all atoms of G_Q can be mapped onto a subset of atoms of G_T in such a way that the bonds (edges) of G_Q map the corresponding bonds in G_T.



Substructure Search

Perception of structural fragments

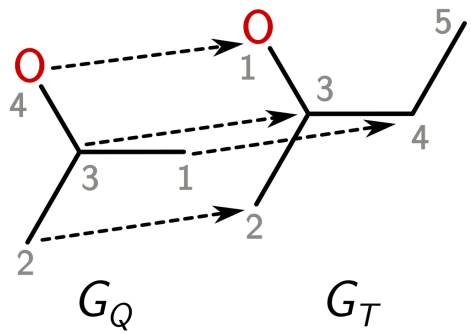
Substructure Search

Process of identifying parts of a given structure that are equivalent to a specified query substructure.

In graph theoretical terms this refers to checking whether the query graph (G_Q) is isomorphic to a subgraph of the target graph (G_T). G_Q is isomorphic to a subgraph of G_T if all atoms of G_Q can be mapped onto a subset of atoms of G_T in such a way that the bonds (edges) of G_Q map the corresponding bonds in G_T.

Properties:

- Mapping $M : G_Q \rightarrow G_T$ is represented by $M = (M_1, M_2, \ldots, M_n)$ with M_i is the number of the target graph atom mapped to the ith query graph atom ($i \rightarrow M_i$).



- mapping: (4,2,3,1)

Substructure Search

Perception of structural fragments

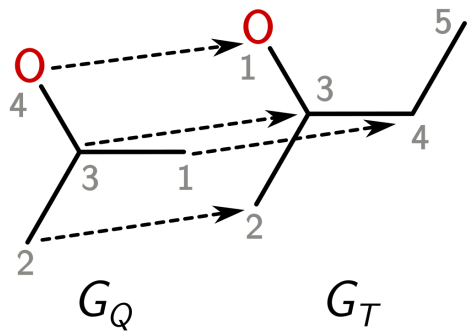
Substructure Search

Process of identifying parts of a given structure that are equivalent to a specified query substructure.

In graph theoretical terms this refers to checking whether the query graph (G_Q) is isomorphic to a subgraph of the target graph (G_T). G_Q is isomorphic to a subgraph of G_T if all atoms of G_Q can be mapped onto a subset of atoms of G_T in such a way that the bonds (edges) of G_Q map the corresponding bonds in G_T.

Properties:

- Mapping $M : G_Q \rightarrow G_T$ is represented by $M = (M_1, M_2, \ldots, M_n)$ with M_i is the number of the target graph atom mapped to the ith query graph atom ($i \rightarrow M_i$).
- If G_Q and G_T have n and m atoms, respectively, the number of all mappings is:
  $N_{maps} = \frac{m!}{(m-n)!}$



- mapping: (4,2,3,1)
- $N_{maps} = \frac{5!}{(5-4)!} = 120$

Substructure Search

Perception of structural fragments

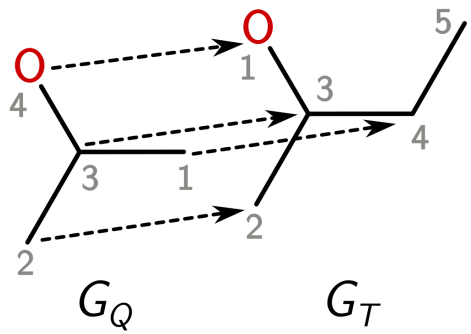
Substructure Search

Process of identifying parts of a given structure that are equivalent to a specified query substructure.

In graph theoretical terms this refers to checking whether the query graph (G_Q) is isomorphic to a subgraph of the target graph (G_T). G_Q is isomorphic to a subgraph of G_T if all atoms of G_Q can be mapped onto a subset of atoms of G_T in such a way that the bonds (edges) of G_Q map the corresponding bonds in G_T.

Properties:

- Mapping $M : G_Q \rightarrow G_T$ is represented by $M = (M_1, M_2, \ldots, M_n)$ with M_i is the number of the target graph atom mapped to the ith query graph atom ($i \rightarrow M_i$).
- If G_Q and G_T have n and m atoms, respectively, the number of all mappings is: $N_{maps} = \frac{m!}{(m-n)!}$
- search is non-polynomial-complete problem ($\mathcal{O}(2^n)$)

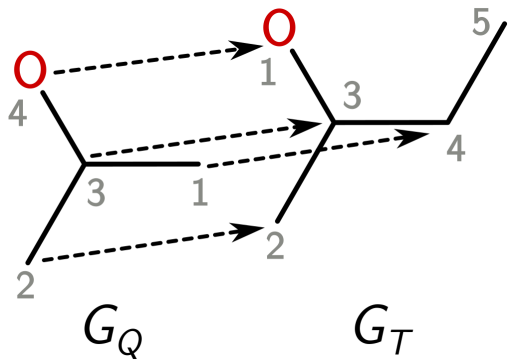


- mapping: (4,2,3,1)
- $N_{maps} = \frac{5!}{(5-4)!} = 120$

Substructure Search

Basic Idea

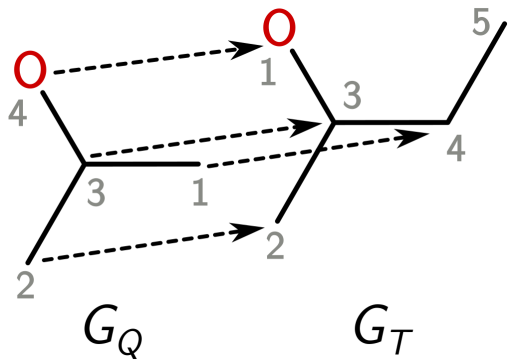
- **Given:** substructure G_Q of $n = 4$ atoms and molecule G_T of $m = 5$ atoms.



Substructure Search

Basic Idea

- **Given:** substructure G_Q of $n = 4$ atoms and molecule G_T of $m = 5$ atoms.

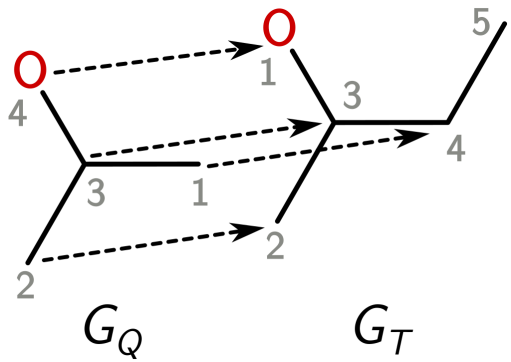


- **Task:** Find edge-preserving mapping between all atoms (vertices) of G_Q and some atoms of G_T, *i.e.*, so that if there is a bond between two atoms in G_Q there has to be a bond between the two corresponding atoms in G_T

Substructure Search

Basic Idea

- **Given:** substructure G_Q of $n = 4$ atoms and molecule G_T of $m = 5$ atoms.

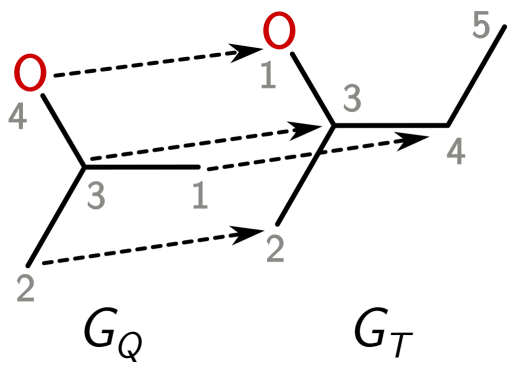


- **Task:** Find edge-preserving mapping between all atoms (vertices) of G_Q and some atoms of G_T, *i.e.*, so that if there is a bond between two atoms in G_Q there has to be a bond between the two corresponding atoms in G_T
- **Problem:** the number of possible mappings can be huge
  - e.g., with $n = 30$ and $m = 7$ there are circa 10 billion possible mappings

Substructure Search

Basic Idea

- **Given:** substructure G_Q of $n = 4$ atoms and molecule G_T of $m = 5$ atoms.



- **Task:** Find edge-preserving mapping between all atoms (vertices) of G_Q and some atoms of G_T, *i.e.*, so that if there is a bond between two atoms in G_Q there has to be a bond between the two corresponding atoms in G_T
- **Problem:** the number of possible mappings can be huge
  - e.g., with $n = 30$ and $m = 7$ there are circa 10 billion possible mappings
- **Approach:** *brute force* walk through all mappings is time-consuming and impracticable with $N_{at} > 10$
  - Standard algorithmic solution: **Backtracking**

Backtracking Algorithm

Basic Idea

Backtracking

Algorithm for fast search for isomorphism among all mappings ($G_q \to G_T$) with a search strategy that traverses a search tree in depth-first order. All searched mappings can be organized hierarchically as a tree. A search tree assigns atoms of G_Q to all possible atoms of G_T in a systematic way.

Backtracking Algorithm

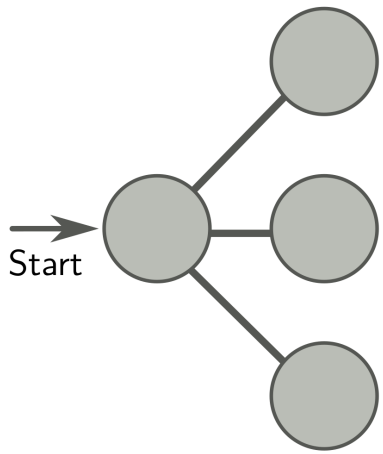
Basic Idea

Backtracking

Algorithm for fast search for isomorphism among all mappings ($G_q \rightarrow G_T$) with a search strategy that traverses a search tree in depth-first order. All searched mappings can be organized hierarchically as a tree. A search tree assigns atoms of G_Q to all possible atoms of G_T in a systematic way.

Procedure:

1. start at arbitrary query atom Q_1 mapped to a target atom T_1
2. neighbors ($Q_2, Q_3, \ldots$) are tried to be mapped to some neighbors of T_1 ($T_2, T_3, \ldots, T_m$)



Backtracking Algorithm

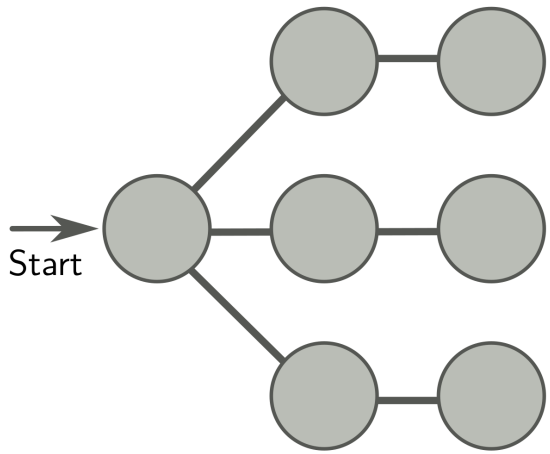
Basic Idea

Backtracking

Algorithm for fast search for isomorphism among all mappings ($G_q \to G_T$) with a search strategy that traverses a search tree in depth-first order. All searched mappings can be organized hierarchically as a tree. A search tree assigns atoms of G_Q to all possible atoms of G_T in a systematic way.

Procedure:

1. start at arbitrary query atom Q_1 mapped to a target atom T_1
2. neighbors ($Q_2, Q_3, \ldots$) are tried to be mapped to some neighbors of T_1 ($T_2, T_3, \ldots, T_m$)
   - if successful step 2 is repeated with the neighbors of $Q_2, Q_3, \ldots, Q_{n-1}$



Backtracking Algorithm

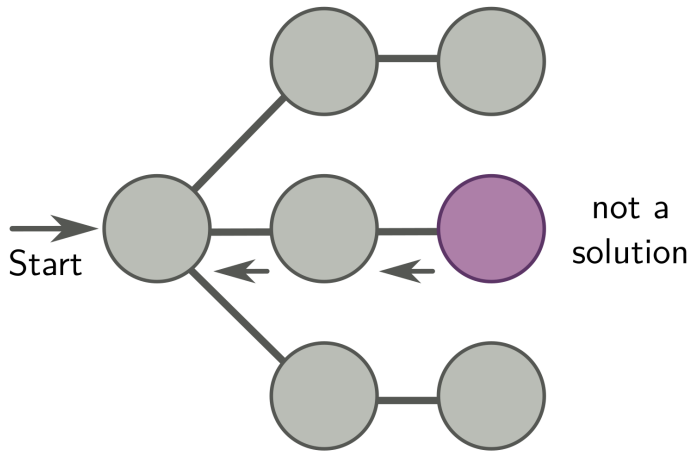
Basic Idea

Backtracking

Algorithm for fast search for isomorphism among all mappings ($G_q \to G_T$) with a search strategy that traverses a search tree in depth-first order. All searched mappings can be organized hierarchically as a tree. A search tree assigns atoms of G_Q to all possible atoms of G_T in a systematic way.

Procedure:

1. start at arbitrary query atom Q_1 mapped to a target atom T_1
2. neighbors ($Q_2, Q_3, \ldots$) are tried to be mapped to some neighbors of T_1 ($T_2, T_3, \ldots, T_m$)
   - if successful step 2 is repeated with the neighbors of $Q_2, Q_3, \ldots, Q_{n-1}$
   - if an atom Q cannot be mapped onto any target atom $T \to$ backtracking to last successful mapped atom $Q' \to$ mapping to different partner from target graph



Backtracking Algorithm

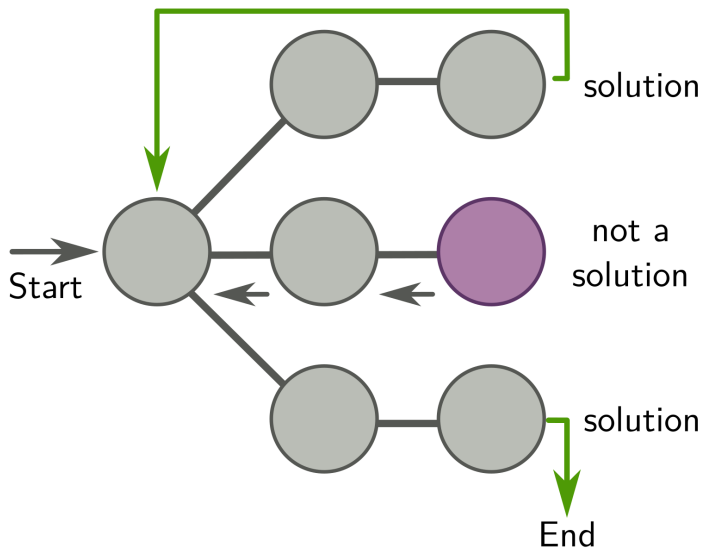
Basic Idea

Backtracking

Algorithm for fast search for isomorphism among all mappings ($G_q \to G_T$) with a search strategy that traverses a search tree in depth-first order. All searched mappings can be organized hierarchically as a tree. A search tree assigns atoms of G_Q to all possible atoms of G_T in a systematic way.

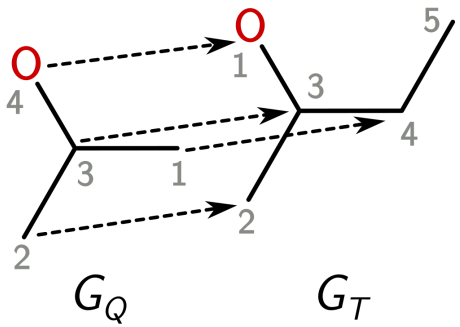
Procedure:

1. start at arbitrary query atom Q_1 mapped to a target atom T_1
2. neighbors ($Q_2, Q_3, \ldots$) are tried to be mapped to some neighbors of T_1 ($T_2, T_3, \ldots, T_m$)
   - if successful step 2 is repeated with the neighbors of $Q_2, Q_3, \ldots, Q_{n-1}$
   - if an atom Q cannot be mapped onto any target atom $T \to$ backtracking to last successful mapped atom $Q' \to$ mapping to different partner from target graph
3. process stops when isomorphism is found or at dead ends

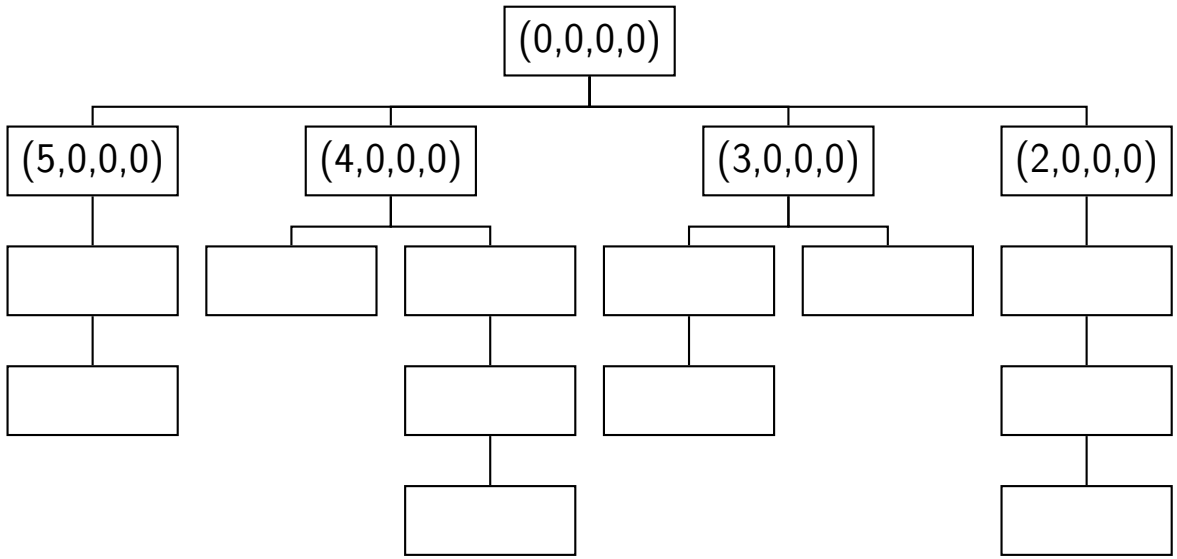


Backtracking Algorithm

Example

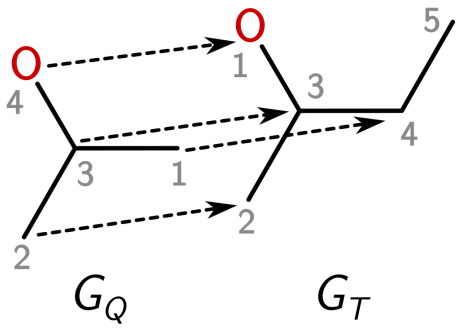


- **Given:** substructure G_Q of $n = 4$ atoms and molecule G_T of $m = 5$ atoms.
- **Task:** Find edge-preserving mapping between all atoms (vertices) of G_Q and some atoms of G_T ($G_Q \rightarrow G_T$)

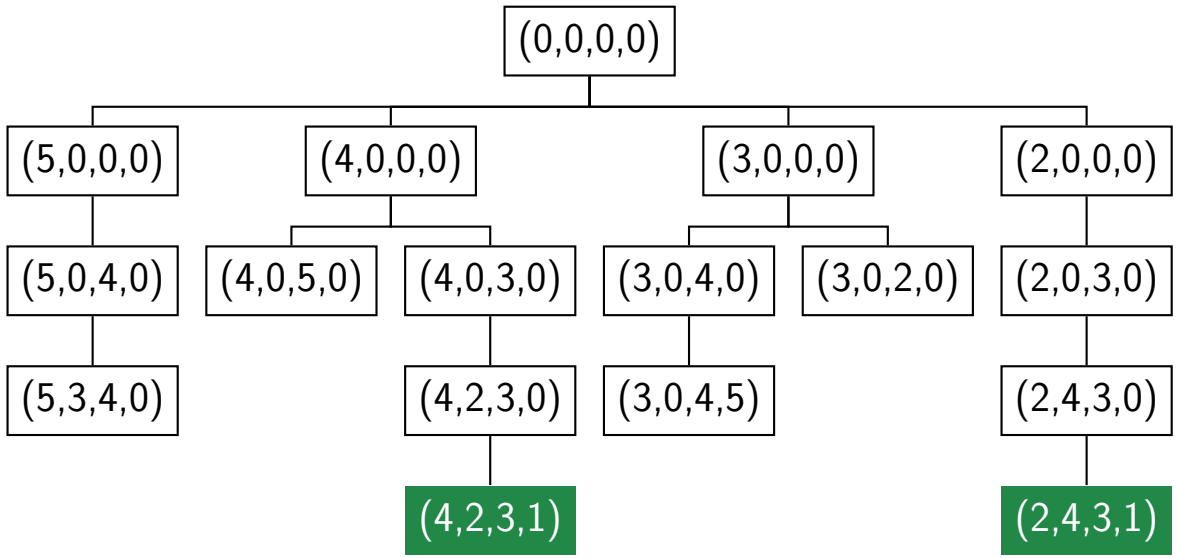


Backtracking Algorithm

Example



- **Given:** substructure G_Q of $n = 4$ atoms and molecule G_T of $m = 5$ atoms.
- **Task:** Find edge-preserving mapping between all atoms (vertices) of G_Q and some atoms of G_T ($G_Q \rightarrow G_T$)



- **Backtracking:** 16 instead of 120 mappings

Backtracking Algorithm

Summary and refined backtracking

- Backtracking:
  - depth-first algorithm has exponential order of computational complexity ($m \cdot 1.141^n$)
  - the earlier search tree traversal fails, the better
  - The order in which substructure atoms are assigned is arbitrary
  - However, starting with rare hetero-atoms might be good for efficiency because only few options are usually available for these

Backtracking Algorithm

Summary and refined backtracking

- Backtracking:
  - depth-first algorithm has exponential order of computational complexity ($m \cdot 1.141^n$)
  - the earlier search tree traversal fails, the better
  - The order in which substructure atoms are assigned is arbitrary
  - However, starting with rare hetero-atoms might be good for efficiency because only few options are usually available for these

- Refined backtracking: **Ullmann algorithm**
  - Backtracking only checks if currently mapped atom yields plausible assignment
  - idea:
    - In each step, keep track of all possible assignments for all substructure atoms
    - If no possible matches remain for any single substructure atom the branch can be skipped
    - Note, that the basic backtracking only check possibilities of the current atom to be assigned and not any later ones

Ullmann Algorithm

Feasibility Matrix for refined backtracking

Ullman Algorithm

The **Ullman algorithm** is a backtracking-based approach used to determine whether two graphs are isomorphic, efficiently exploring all possible mappings between their nodes while employing pruning strategies to minimize unnecessary exploration.

Ullmann Algorithm

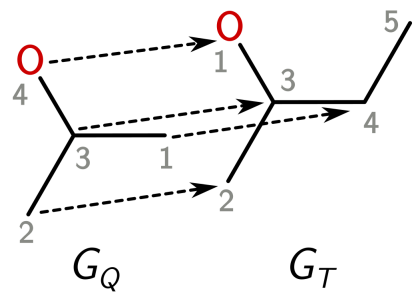
Feasibility Matrix for refined backtracking

Ullman Algorithm

The **Ullman algorithm** is a backtracking-based approach used to determine whether two graphs are isomorphic, efficiently exploring all possible mappings between their nodes while employing pruning strategies to minimize unnecessary exploration.

Procedure:

1. Initialization: Construct feasibility matrix FM
   - atom types must match
   - atom T in the molecule G_T must have at least as many neighbors as atom Q in G_Q



	1	2	3	4	5
1		x		x	x
2		x		x	x
3			x		
4	x				

Ullmann Algorithm

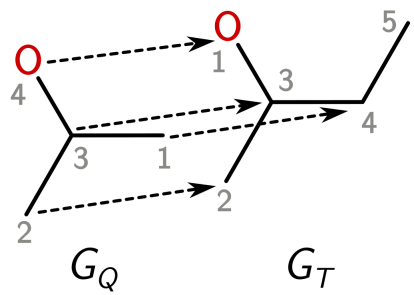
Feasibility Matrix for refined backtracking

Ullman Algorithm

The **Ullman algorithm** is a backtracking-based approach used to determine whether two graphs are isomorphic, efficiently exploring all possible mappings between their nodes while employing pruning strategies to minimize unnecessary exploration.

Procedure:

1. Initialization: Construct feasibility matrix FM
   - atom types must match
   - atom T in the molecule G_T must have at least as many neighbors as atom Q in G_Q
2. Refinement of FM: Check plausibility of each entry
   - if atom Q should be assigned to atom T it should also be possible to assign neighbors of Q to neighbors of T
   - Thus, for $M(Q,T) = x$ check for all neighbors j of Q if $M(j,k) = x$ where k is neighbor of T. If no neighbor T can be found eliminate possibility $M(Q,k)$



	1	2	3	4	5
1		x		**x**	x
2		x		**x**	x
3			x		
4	x				

Ullmann Algorithm

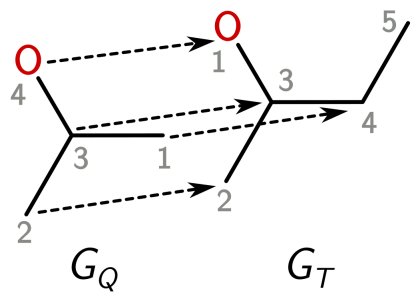
Feasibility Matrix for refined backtracking

Ullman Algorithm

The **Ullman algorithm** is a backtracking-based approach used to determine whether two graphs are isomorphic, efficiently exploring all possible mappings between their nodes while employing pruning strategies to minimize unnecessary exploration.

Procedure:

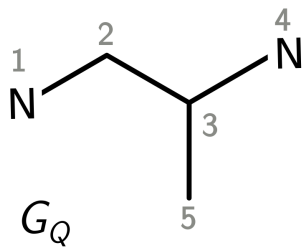
1. Initialization: Construct feasibility matrix FM
   - atom types must match
   - atom T in the molecule G_T must have at least as many neighbors as atom Q in G_Q
2. Refinement of FM: Check plausibility of each entry
   - if atom Q should be assigned to atom T it should also be possible to assign neighbors of Q to neighbors of T
   - Thus, for $M(Q,T) = x$ check for all neighbors j of Q if $M(j,k) = x$ where k is neighbor of T. If no neighbor T can be found eliminate possibility $M(Q,k)$
3. Perform standard backtracking

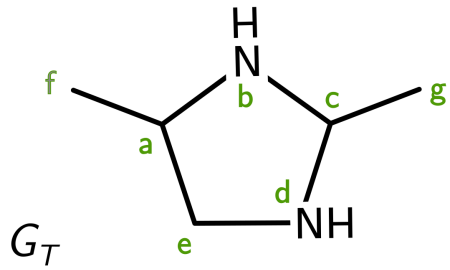


	1	2	3	4	5
1		x			x
2		x			x
3			x		
4	x				

Ullmann Algorithm

Example





- Initial feasibility matrix:

	a	b	c	d	e	f	g
1							
2							
3							
4							
5							

- Refined feasibility matrix:

	a	b	c	d	e	f	g
1							
2							
3							
4							
5							

Ullmann Algorithm

Summary

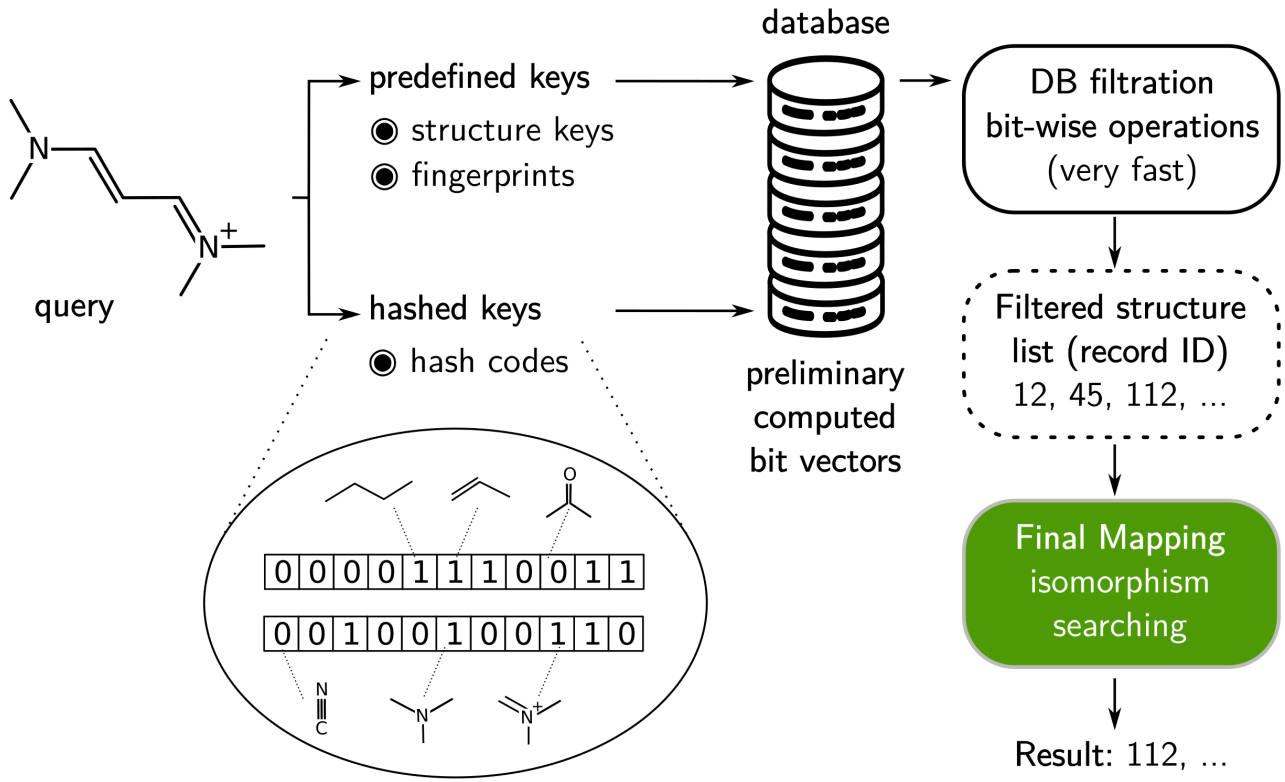
- Set up a feasibility matrix containing all possible assignments of substructure atoms to molecule atoms with respect to
  - atom type
  - bond order
- Refine the matrix
  - for each potential assignment check possible assignments of neighboring atoms
- Explore the search tree
  - choose assignment for current atom from feasibility matrix
  - update matrix & refine matrix
    - possibilities remain for every atom: continue exploration with next atom
    - no possibilities for at least one atom: choose different assignment & backtrack

Substructure Search

Screening

Screening

Screening is a process to filter structures that do not meet certain criteria or match specific patterns, namely predefined set of structural fragments (keys), reducing the set of structures being checked for isomorphism by means of backtracking.



c) Structural Keys

Perception of structural fragments

- idea: speed-up substructure search by filtering out molecules that do not contain a specified substructure

Structural Key

A structural key is a Boolean vector (represented by a bit vector) in which each element is *true* or *false* and denotes existence or absence of a corresponding structural feature.

Typical examples:

- absence/presence of certain elements
- existence of rings
- common functional groups

Procedure:

1. Construct key for searched substructure
2. reject molecules that do not contain fragments encoded in query key from further processing
3. isomorphism search on molecules that passed screening

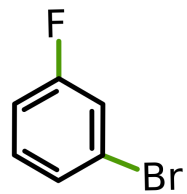
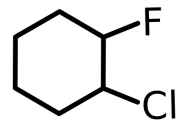
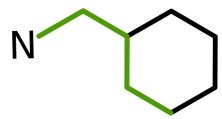
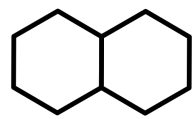
- approaches:
  - molecular formula (missing atoms)
  - structural keys
    - **M**olecular **ACC**ess **S**ystem (**MACCS**) keys: set of 166 keys (functional groups, rings, etc.)
    - **SMARTS**

c) Structural Keys

SMARTS

SMARTs (SMiles ARbitrary Target Specification)

Line notation used for the description of queries for substructure searching. While SMILES describe valid molecules, **SMARTS** describes a fragment that is searched against some other target molecules.

SMARTS	Match	No-match
`a[Cl,Br,F]` aromatic atom connected to a halogen		
`NC[C;R1;D3]` aliphatic carbon participating in 1 ring with 3 explicit connections		

d) Molecular Fingerprints

Molecular Bit Vectors (1D Descriptors)

Fingerprints (FPs)

A molecular **fingerprint** (FP) is a vectorized representation of molecules that captures precise details of the atomic configurations in them.

d) Molecular Fingerprints

Molecular Bit Vectors (1D Descriptors)

Fingerprints (FPs)

A molecular **fingerprint** (FP) is a vectorized representation of molecules that captures precise details of the atomic configurations in them.

- idea: bit/count vector calculated from a set of substructures (*e.g.* functional groups)
- typical size: 1k–4k bits

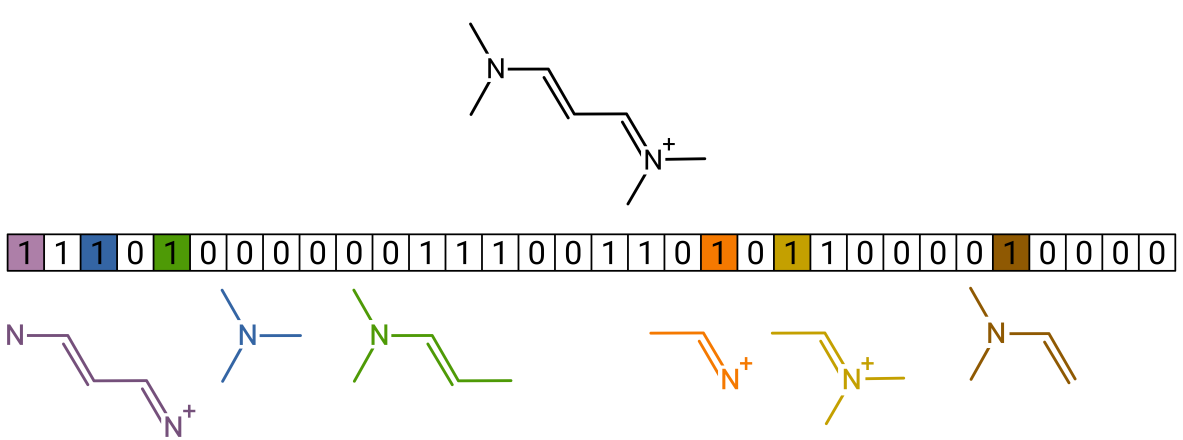
d) Molecular Fingerprints

Molecular Bit Vectors (1D Descriptors)

Fingerprints (FPs)

A molecular **fingerprint** (FP) is a vectorized representation of molecules that captures precise details of the atomic configurations in them.

- *E.g.*, bit vector where 1 indicates the presence of a structural feature and 0 its absence:



- idea: bit/count vector calculated from a set of substructures (*e.g.* functional groups)
- typical size: 1k–4k bits

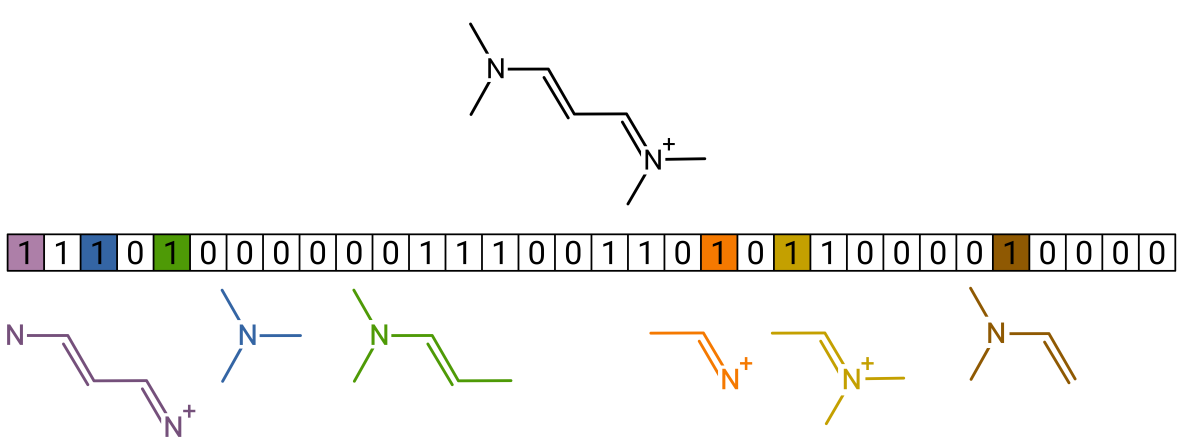
d) Molecular Fingerprints

Molecular Bit Vectors (1D Descriptors)

Fingerprints (FPs)

A molecular **fingerprint** (FP) is a vectorized representation of molecules that captures precise details of the atomic configurations in them.

- *E.g.*, bit vector where 1 indicates the presence of a structural feature and 0 its absence:



- idea: bit/count vector calculated from a set of substructures (*e.g.* functional groups)
- typical size: 1k–4k bits

- Examples (`rdkit` implementations):
  - RDKit FPs
  - Morgan/Circular FPs
  - Extended Connectivity FPs (ECFP)

1D Molecular Structure Descriptors

Morgan and Extended Connectivity Fingerprints

Idea

Morgan fingerprints (MFPs) are iteratively constructed by applying a hashing procedure to atom environments, considering their neighboring atoms up to a certain radius. **Extended Connectivity Fingerprints (ECFPs)** are a specific type of MFPs, where in every iteration features are hashed to generate identifiers for different substructures.

1D Molecular Structure Descriptors

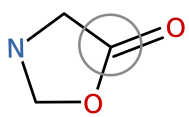
Morgan and Extended Connectivity Fingerprints

Idea

Morgan fingerprints (MFPs) are iteratively constructed by applying a hashing procedure to atom environments, considering their neighboring atoms up to a certain radius. **Extended Connectivity Fingerprints (ECFPs)** are a specific type of MFPs, where in every iteration features are hashed to generate identifiers for different substructures.

- **Initialization:** assignment of initial identifier to each atom

- Diameter 0 :



1D Molecular Structure Descriptors

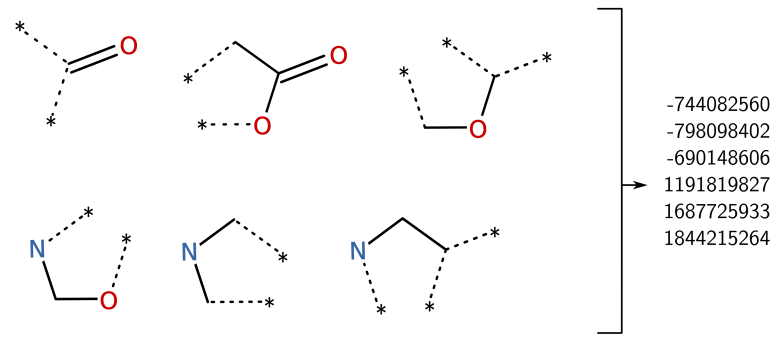
Morgan and Extended Connectivity Fingerprints

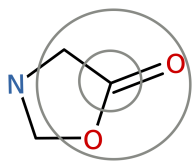
Idea

Morgan fingerprints (MFPs) are iteratively constructed by applying a hashing procedure to atom environments, considering their neighboring atoms up to a certain radius. **Extended Connectivity Fingerprints (ECFPs)** are a specific type of MFPs, where in every iteration features are hashed to generate identifiers for different substructures.

- **Initialization:** assignment of initial identifier to each atom
- **Iteration:** update of identifier of each atom by combining its own identifier with those of its neighboring atoms. This process is repeated up to a specified number of iterations (radius).

- Diameter 2:





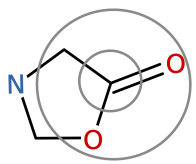
1D Molecular Structure Descriptors

Morgan and Extended Connectivity Fingerprints

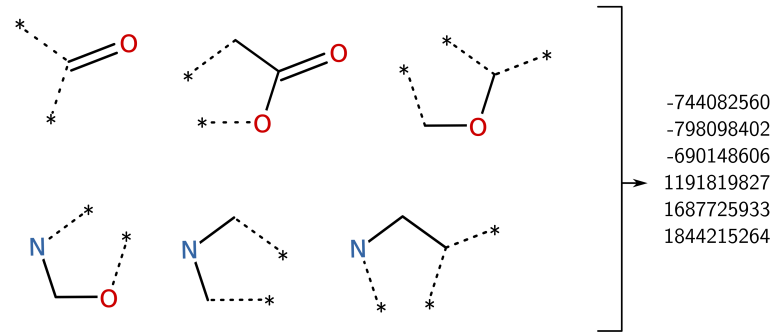
Idea

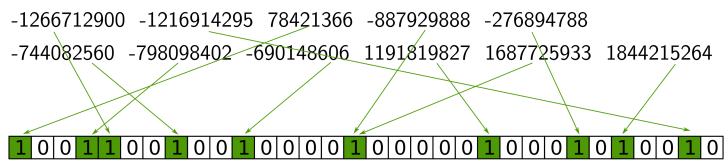
Morgan fingerprints (MFPs) are iteratively constructed by applying a hashing procedure to atom environments, considering their neighboring atoms up to a certain radius. **Extended Connectivity Fingerprints (ECFPs)** are a specific type of MFPs, where in every iteration features are hashed to generate identifiers for different substructures.

- **Initialization:** assignment of initial identifier to each atom
- **Iteration:** update of identifier of each atom by combining its own identifier with those of its neighboring atoms. This process is repeated up to a specified number of iterations (radius).
- **Final Hashing:** Hashing of final identifiers into a fixed-length bit vector (binary string, *i.e.*, molecular FP)



- Diameter 2:





Similarity Search

Perception of similar structures

- limits of substructure searching:
  - database structure must contain the entire query substructure (user must have a clear view of types of structure to be retrieved)
  - no control over size of produced output given a query (number of hits)
  - queries need to be engineered to obtain meaningful results (not too few or too many hits)
  - no ranking

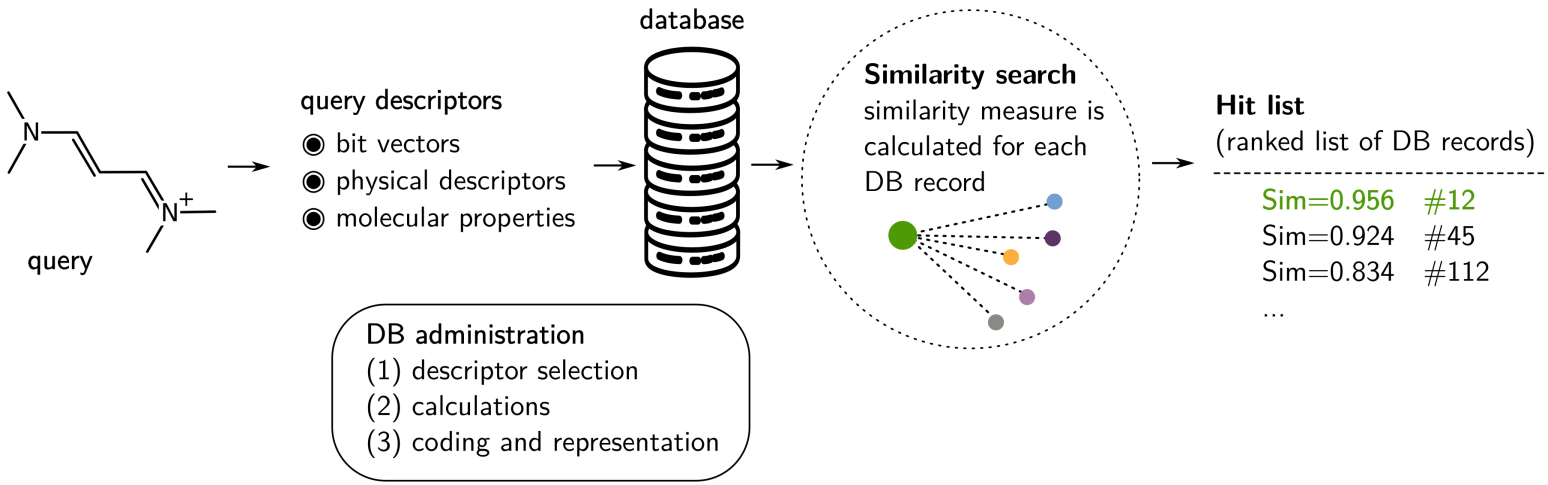
Similarity Search

Perception of similar structures

- limits of substructure searching:
  - database structure must contain the entire query substructure (user must have a clear view of types of structure to be retrieved)
  - no control over size of produced output given a query (number of hits)
  - queries need to be engineered to obtain meaningful results (not too few or too many hits)
  - no ranking

Similarity Search

Similarity search is an alternative and complement to substructure searching. Similarity searching retrieves objects that are similar to a query, sorted in order of their decreasing similarity.



Similarity Search

Similarity measures

Similarity

Similarity concept can be applied in different chemical spaces, e.g., in the vector, property, descriptor or structure space, which can be formalized as vectors ($\{X_1, X_2, \ldots, X_n\}$) in the following way:

$$V_A\ V_B \Leftrightarrow P_A\ P_B \Leftrightarrow D_A\ D_B \Leftrightarrow S_A\ S_B \Leftrightarrow V_A\ V_B$$

Similarity Search

Similarity measures

Similarity

Similarity concept can be applied in different chemical spaces, e.g., in the vector, property, descriptor or structure space, which can be formalized as vectors ($\{X_1, X_2, \ldots, X_n\}$) in the following way:

$$V_A\ V_B \Leftrightarrow P_A\ P_B \Leftrightarrow D_A\ D_B \Leftrightarrow S_A\ S_B \Leftrightarrow V_A\ V_B$$

- similarity of object A and B: $S_{AB} = matches/overlaps$ w.r.t. one/multiple characteristics $\{X_{jA}\}$ and $\{X_{jB}\}$
- identical objects: $S_{AB} = 1$ (or other maximum value)

Similarity Search

Similarity measures

Similarity

Similarity concept can be applied in different chemical spaces, e.g., in the vector, property, descriptor or structure space, which can be formalized as vectors ($\{X_1, X_2, \ldots, X_n\}$) in the following way:

$$V_A\ V_B \Leftrightarrow P_A\ P_B \Leftrightarrow D_A\ D_B \Leftrightarrow S_A\ S_B \Leftrightarrow V_A\ V_B$$

- similarity of object A and B: $S_{AB} = matches/overlaps$ w.r.t. one/multiple characteristics $\{X_{jA}\}$ and $\{X_{jB}\}$
- identical objects: $S_{AB} = 1$ (or other maximum value)
- **categories:**
  - structure space:
    - *Distance coefficients* – how far apart are objects A and B (Euclidian distance, RMSD)
    - *Association coefficients* – similarity based on number of characteristics common to both A and B (Tanimoto, Dice, cosine coefficient)

Similarity Search

Similarity measures

Similarity

Similarity concept can be applied in different chemical spaces, e.g., in the vector, property, descriptor or structure space, which can be formalized as vectors ($\{X_1, X_2, \ldots, X_n\}$) in the following way:

$$V_A\ V_B \Leftrightarrow P_A\ P_B \Leftrightarrow D_A\ D_B \Leftrightarrow S_A\ S_B \Leftrightarrow V_A\ V_B$$

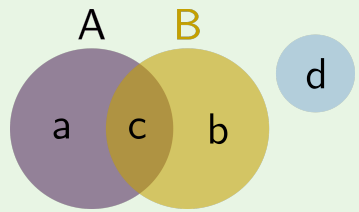
- similarity of object A and B: $S_{AB} = matches/overlaps$ w.r.t. one/multiple characteristics $\{X_{jA}\}$ and $\{X_{jB}\}$
- identical objects: $S_{AB} = 1$ (or other maximum value)
- **categories:**
  - structure space:
    - *Distance coefficients* – how far apart are objects A and B (Euclidian distance, RMSD)
    - *Association coefficients* – similarity based on number of characteristics common to both A and B (Tanimoto, Dice, cosine coefficient)
  - descriptor space:
    - *Correlation coefficients* – statistical analysis of the relationships between variables, *e.g.*, in the descriptor vectors D_A and D_B
    - *Probabilistic coefficients* – distribution of the frequencies of descriptor values over a particular database

Similarity Search

Association Coefficients

Simple matching coefficient

Lets consider objects A and B, where a is the number of features present in A and absent in B, b is the number of features absent in A but present in B, c is the number of features common to both objects, and d is the number of features absent in both.

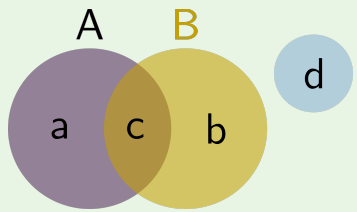


Similarity Search

Association Coefficients

Simple matching coefficient

Lets consider objects A and B, where a is the number of features present in A and absent in B, b is the number of features absent in A but present in B, c is the number of features common to both objects, and d is the number of features absent in both.



Similarity measure: all matches $c + d$ relative to all possibilities of matches and mismatches $(c + d) + (a + b)$, yields

$$S = \frac{c + d}{a + b + c + d},$$

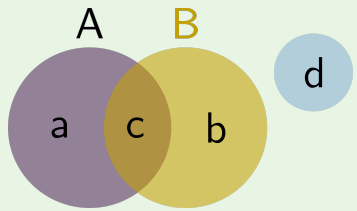
which is called simple matching coefficient (equal weight to matches and mismatches).

Similarity Search

Association Coefficients

Simple matching coefficient

Lets consider objects A and B, where a is the number of features present in A and absent in B, b is the number of features absent in A but present in B, c is the number of features common to both objects, and d is the number of features absent in both.



Similarity measure: all matches $c + d$ relative to all possibilities of matches and mismatches $(c + d) + (a + b)$, yields

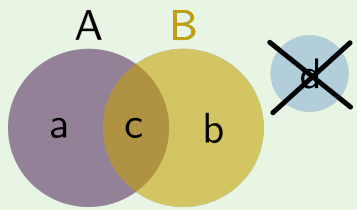
$$S = \frac{c + d}{a + b + c + d},$$

which is called simple matching coefficient (equal weight to matches and mismatches).

Tanimoto Similarity

Omitting d (features absent in both objects), when absent features convey no information, gives the Tanimoto similarity:

$$T = \frac{c}{a + b + c}$$



Similarity Searching

Practical course

- construct database comprising 3D-structures and absorption maxima (integer) of 19 cyanine dyes
- perform similarity search across this database
- visualize the molecular relations in form of a network