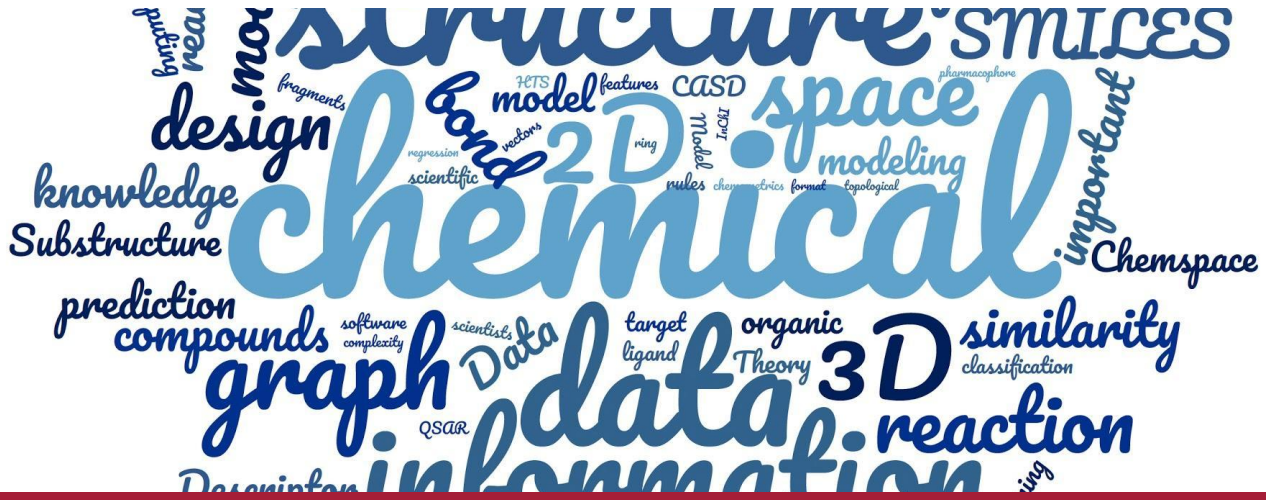
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Institute for Bioinformatics and Medical Informatics



BIO-4372 **Cheminformatics**

L04 Topological Structure Comparison

Part I: Exact and Substructure Methods

Winter Semester 2022-23

Philipp Thiel

- Exact structure searching
- Identity of molecules
- Molecule identity as a graph-based problem
- Efficient identity testing using line notations
  - Problem of uniqueness
  - Canonical labeling of molecular graphs

- Substructure searching
- Subgraph isomorphism problem
- Algorithmic approaches for substructure matching
- Efficient substructure searching on large databases

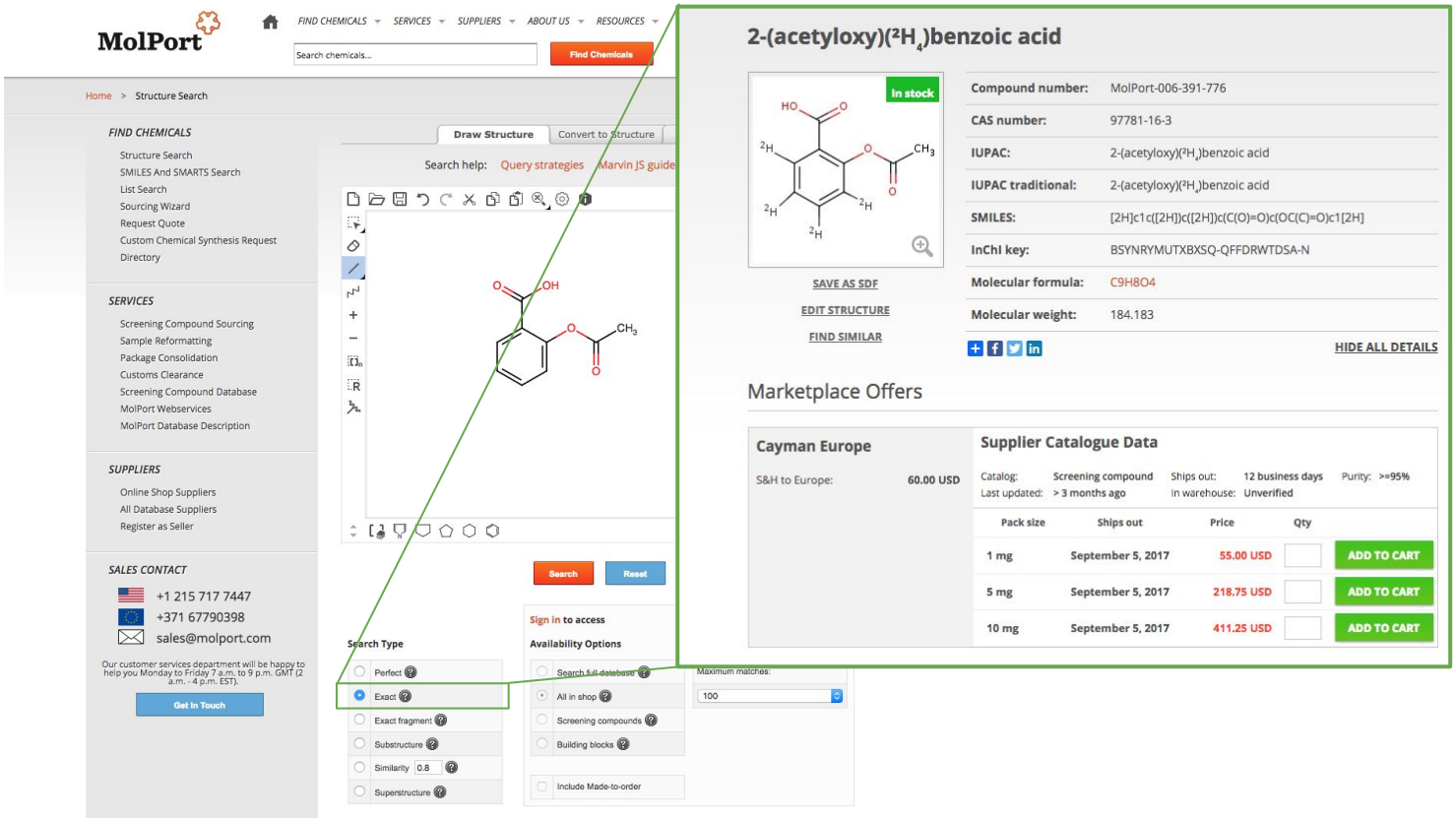
Motivation

- A lot of databases with chemical information are available
  - Cf. *L01 Introduction*
  - Fine chemical suppliers
  - General chemical information resources
  - Experimental data

- Scientists often require information on a specific molecule
- For example information on ...

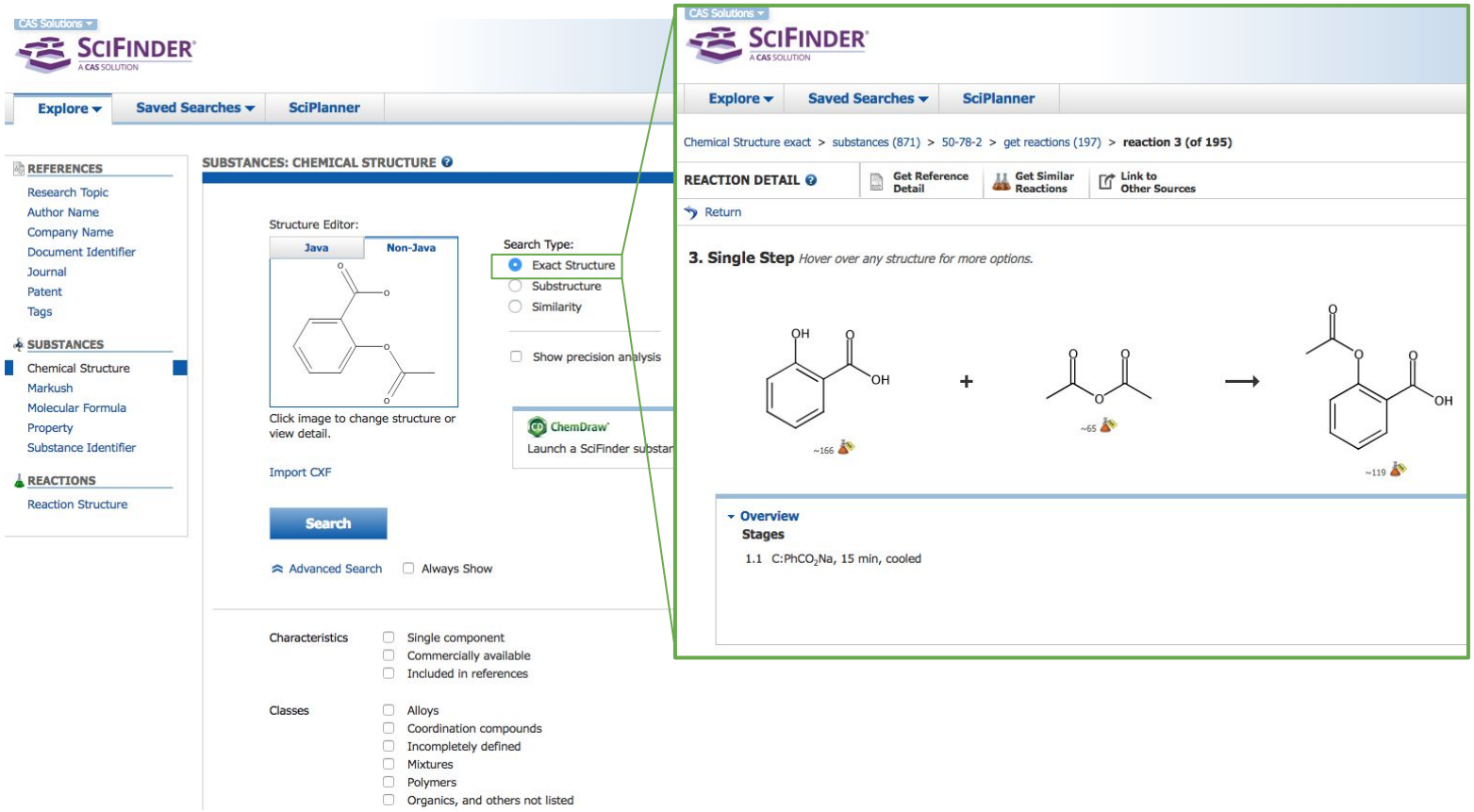
Motivation

- Availability and pricing



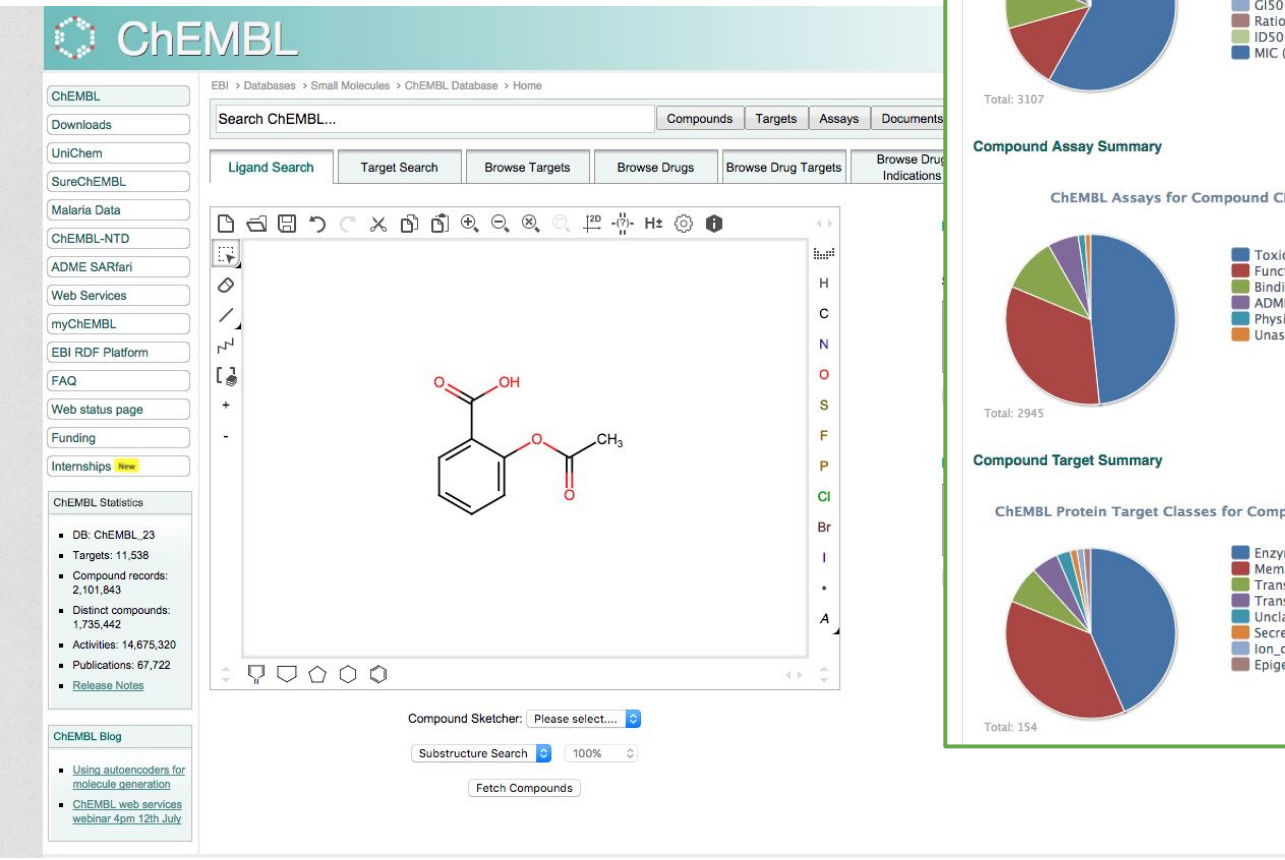
Motivation

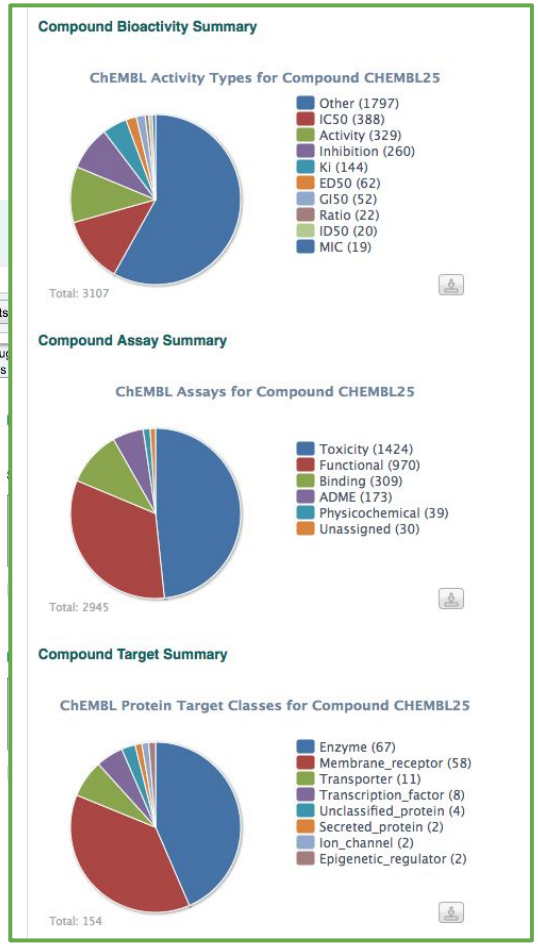
- Synthesis route(s)



Motivation

- Biological activity





Basic Problem

- Task to perform in all given examples:

  Exact Structure Searching

- Basic problem:

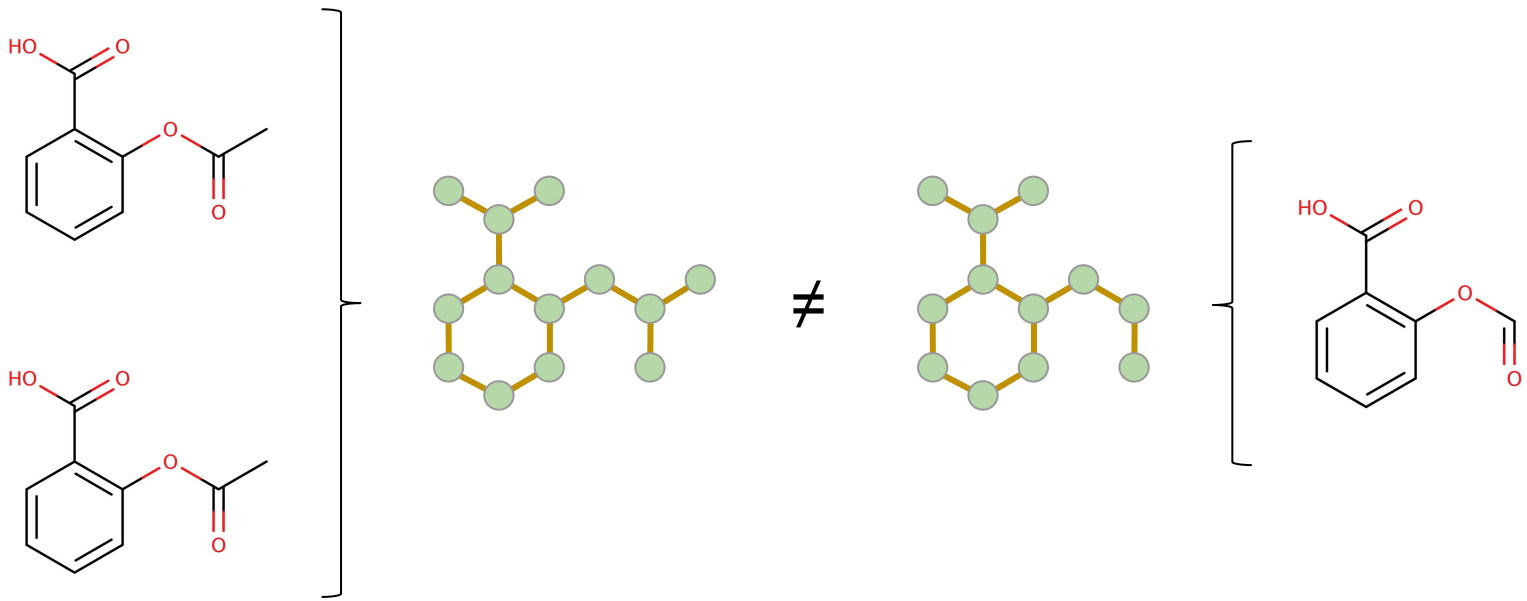
  Decide if two given molecules A and B are identical?

- Identity testing is very important
- It can be described using **molecular graph theory**
  - Problem rather complex
- We will also discuss more efficient approaches for this problem

Identity of Molecules

Graph-Based Identity

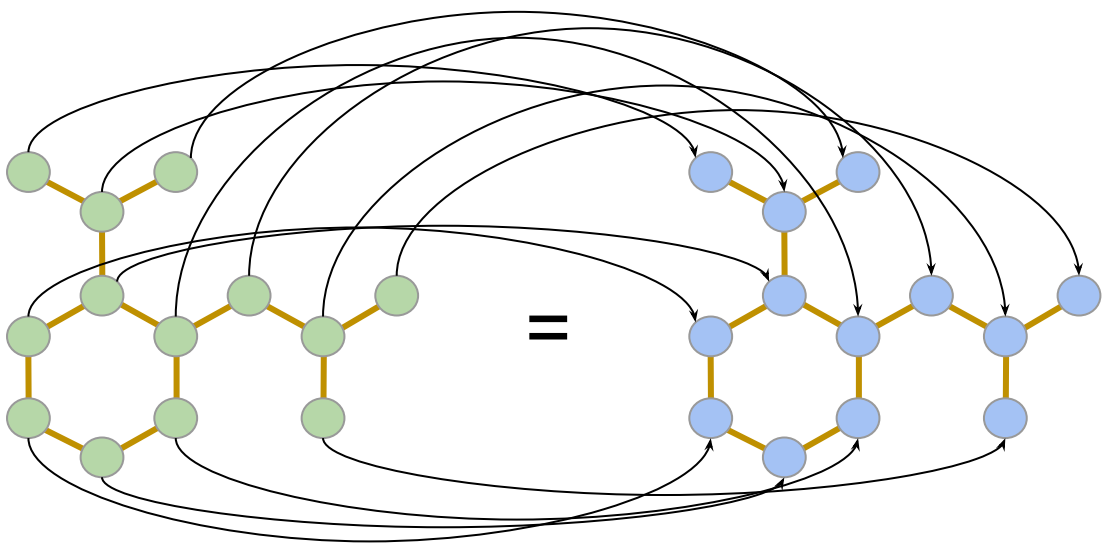
- If molecules *A* and *B* are identical
  ⇒ molecular graphs **A** and **B** are **isomorphic**



Identity of Molecules

Graph-Based Identity

- **Graph Isomorphism (GI)**
- Given: $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$ with $|V_1| = |V_2|$
- Problem: $\exists$ f: $V_1 \rightarrow V_2$, an edge-preserving bijection, such that: $(u,v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$?



Identity of Molecules

Graph-Based Identity

- **Graph Isomorphism (GI)**
- In general, one of the few problems in NP of which it is not yet known whether it is in P or NP-complete [1]
  - So far no proof of NP-completeness
  - So far no polynomial-time algorithm

- **Bounded node degree**: can be solved in **polynomial time** [2]
  - Thus, problem for molecular graphs in P

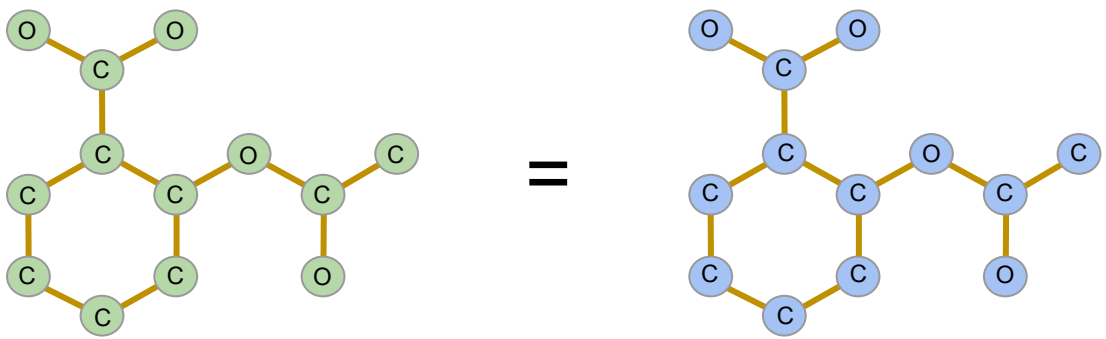
- GI alone **not sufficient for molecular graphs**
  - Node and edge labels have to be considered as well
  - Labeled Graph Isomorphism problem (LGI)

1. [GJ]
2. Luks E.M. (1982) *J. Computer System Sci.*, 25, 42

Identity of Molecules

Graph-Based Identity

- **Labeled Graph Isomorphism (LGI)**
- Given: $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$ with $|V_1| = |V_2|$ and a node labeling function $\mu: V_1 \cup V_2 \rightarrow \Sigma$
- G_1 is isomorphic to G_2 w.r.t to labels if $\exists$ bijection $f: V_1 \rightarrow V_2$ such that: $(u,v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$ and $\mu(u) = \mu(f(u)) \quad \forall \; u \in V_1$



Identity of Molecules

Graph-Based Identity

- **Labeled Graph Isomorphism (LGI)**
- An easier problem than GI
  - Labeling reduces number of possible node permutations
  - Testing for the numbers of equivalent labels is trivial
    - Comparison of elemental compositions: e.g. $C_6H_6 \neq C_6H_5O$
    - Can exclude existence of LGI

- For **larger structures** LGI is still a **hard problem**
  - Efficient implementations can take several seconds.

- Pairwise comparison of large databases ($>10^6$) infeasible
  - Often requires 10^{12} or more individual pair comparisons
  - **Thus, too slow**!

Identity of Molecules

Line Notations

- **Efficient** identity check possible with **line notations**
  - Simply by **string comparison**

- General problem of line notations: **not unique**
- Remember SMILES for ethanol:
  - CCO = OCC = CC[OH] = C1.C12.O2 = …

- Key requirement for string-based molecule comparison:

  Employed line notation is unique (= canonical)

Identity of Molecules

Line Notations

- Uniqueness achieved by computing a

  canonical numbering of the **molecular graph**
- Canonical numbering allows to generate unique line notations
- Example: Unique SMILES (USMILES)
  - USMILES for ethanol: CCO

- Most important algorithm: **Morgan Algorithm** [1]
  - Most applications use variants of it

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm
Overview

- Idea
  - **Numbering** based on **connectivity** (node degree)
    - Node degree alone insufficient to construct a unique numbering
  - So-called **extended connectivities** (EC) used
    - Include information of adjacent nodes

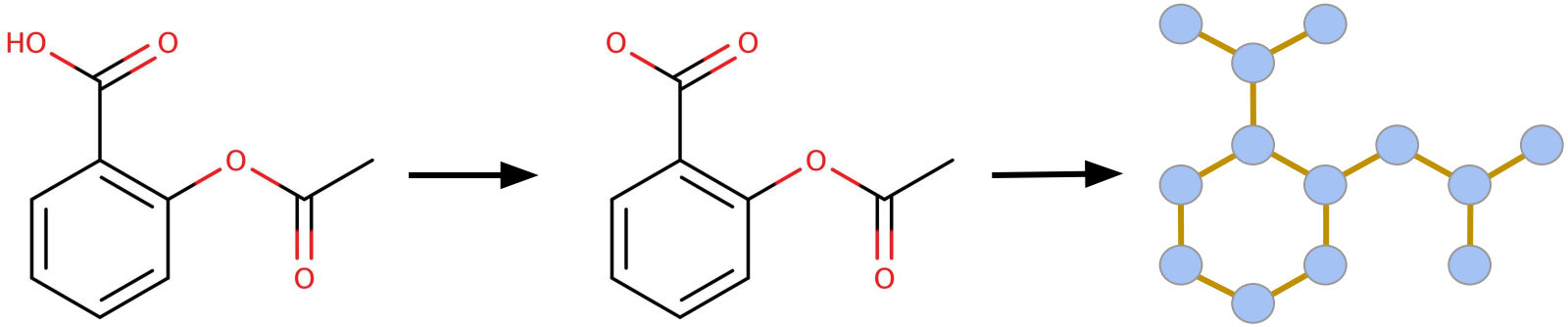
- Two-step algorithm
  1. **Relaxation**: node classification by iterative generation of EC labels
  2. **Canonical enumeration**: node numbering based on EC labels and resolving of remaining ambiguities

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Preprocessing

- Algorithm works on unlabeled **heavy atom graph**
  - Remove hydrogen atoms
  - Ignore atom types
  - Ignore bond orders



1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Relaxation

In : $G = (V, E)$ preprocessed molecular graph

Out: $G = (V, E)$ with EC node labels

$EC_0(v) = 1 \; \forall \; v \in V$;

$c = 1$;

$i = 1$;

while *true* **do**

  $\mathcal{C} = \emptyset$;

  foreach v **in** V **do**

    $EC_i(v) = \sum_{(v,u) \in E} EC_{i-1}(u)$;

    if $EC_i(v) \notin \mathcal{C}$ **then**

      $\mathcal{C} = \mathcal{C} \cup EC_i(v)$;

    end

  end

  if $|\mathcal{C}| \leq c$ **then**

    $EC = EC_{i-1}$;

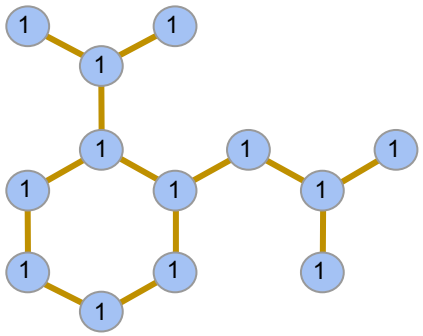
    break;

  end

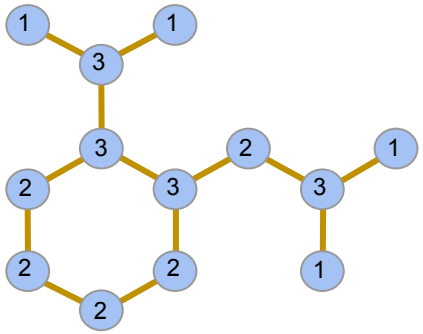
  $c = |\mathcal{C}|$;

  $i = i + 1$;

end



i = 1
c = 1



i = 2
c = 3

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Relaxation

In : $G = (V, E)$ preprocessed molecular graph

Out: $G = (V, E)$ with EC node labels

$EC_0(v) = 1 \; \forall \; v \in V$;

$c = 1$;

$i = 1$;

while *true* **do**

  $\mathcal{C} = \emptyset$;

  foreach v **in** V **do**

    $EC_i(v) = \sum_{(v,u) \in E} EC_{i-1}(u)$;

    if $EC_i(v) \notin \mathcal{C}$ **then**

      $\mathcal{C} = \mathcal{C} \cup EC_i(v)$;

    end

  end

  if $|\mathcal{C}| \leq c$ **then**

    $EC = EC_{i-1}$;

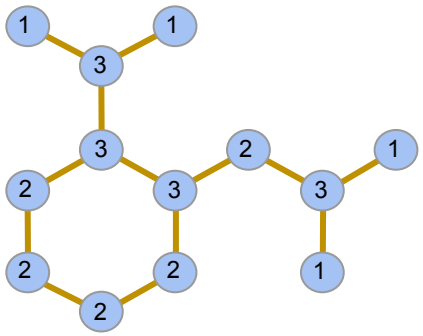
    break;

  end

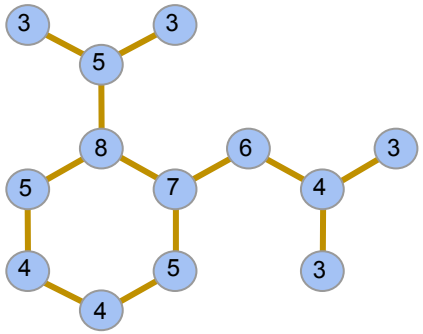
  $c = |\mathcal{C}|$;

  $i = i + 1$;

end



i = 2
c = 3



i = 3
c = 6

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Relaxation

In : $G = (V, E)$ preprocessed molecular graph

Out: $G = (V, E)$ with EC node labels

$EC_0(v) = 1 \; \forall \; v \in V$;

$c = 1$;

$i = 1$;

while *true* **do**

  $\mathcal{C} = \emptyset$;

  foreach v **in** V **do**

    $EC_i(v) = \sum_{(v,u) \in E} EC_{i-1}(u)$;

    if $EC_i(v) \notin \mathcal{C}$ **then**

      $\mathcal{C} = \mathcal{C} \cup EC_i(v)$;

    end

  end

  if $|\mathcal{C}| \leq c$ **then**

    $EC = EC_{i-1}$;

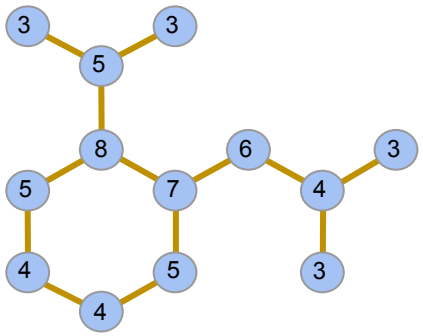
    break;

  end

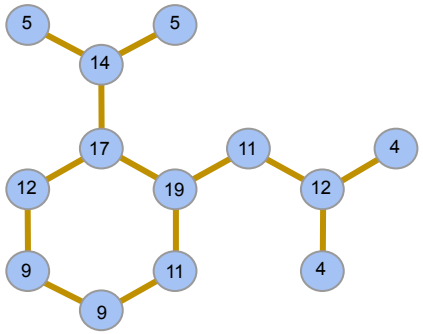
  $c = |\mathcal{C}|$;

  $i = i + 1$;

end



i = 3
c = 6



i = 4
c = 8

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Relaxation

In : $G = (V, E)$ preprocessed molecular graph

Out: $G = (V, E)$ with EC node labels

$EC_0(v) = 1 \; \forall \; v \in V$;

$c = 1$;

$i = 1$;

while *true* **do**

  $\mathcal{C} = \emptyset$;

  foreach v **in** V **do**

    $EC_i(v) = \sum_{(v,u) \in E} EC_{i-1}(u)$;

    if $EC_i(v) \notin \mathcal{C}$ **then**

      $\mathcal{C} = \mathcal{C} \cup EC_i(v)$;

    end

  end

  if $|\mathcal{C}| \leq c$ **then**

    $EC = EC_{i-1}$;

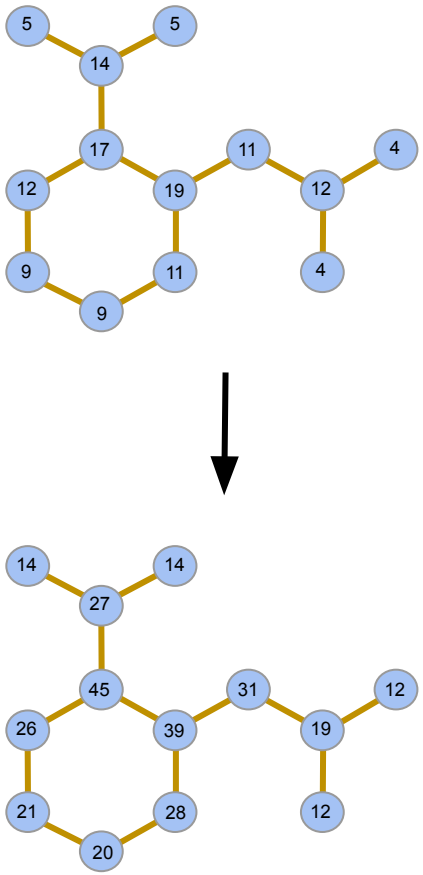
    break;

  end

  $c = |\mathcal{C}|$;

  $i = i + 1$;

end



i = 4
c = 8

i = 5
c = 11

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Relaxation

In : $G = (V, E)$ preprocessed molecular graph

Out: $G = (V, E)$ with EC node labels

$EC_0(v) = 1 \; \forall \; v \in V$;

$c = 1$;

$i = 1$;

while *true* **do**

  $\mathcal{C} = \emptyset$;

  foreach v **in** V **do**

    $EC_i(v) = \sum_{(v,u) \in E} EC_{i-1}(u)$;

    if $EC_i(v) \notin \mathcal{C}$ **then**

      $\mathcal{C} = \mathcal{C} \cup EC_i(v)$;

    end

  end

  if $|\mathcal{C}| \leq c$ **then**

    $EC = EC_{i-1}$;

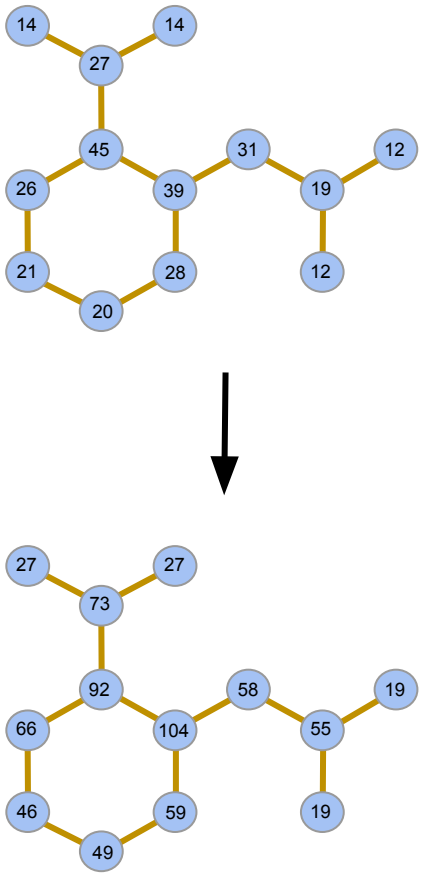
    break;

  end

  $c = |\mathcal{C}|$;

  $i = i + 1$;

end



i = 5
c = 11

i = 6
c = 11

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Relaxation

In : $G = (V, E)$ preprocessed molecular graph

Out: $G = (V, E)$ with EC node labels

$EC_0(v) = 1 \; \forall \; v \in V$;

$c = 1$;

$i = 1$;

while *true* **do**

  $\mathcal{C} = \emptyset$;

  foreach v **in** V **do**

    $EC_i(v) = \sum_{(v,u) \in E} EC_{i-1}(u)$;

    if $EC_i(v) \notin \mathcal{C}$ **then**

      $\mathcal{C} = \mathcal{C} \cup EC_i(v)$;

    end

  end

  if $|\mathcal{C}| \leq c$ **then**

    $EC = EC_{i-1}$;

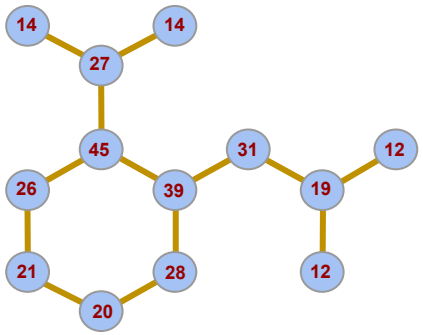
    break;

  end

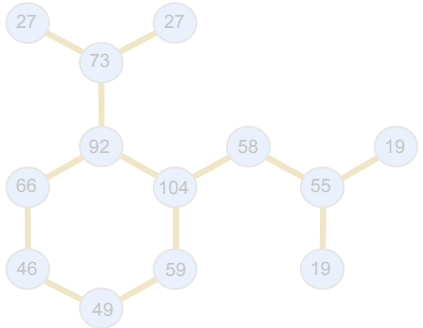
  $c = |\mathcal{C}|$;

  $i = i + 1$;

end



i = 5
c = 11



i = 6
c = 11

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$\boxed{label(v) = 0 \ \forall \ v \in V;}$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \ u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

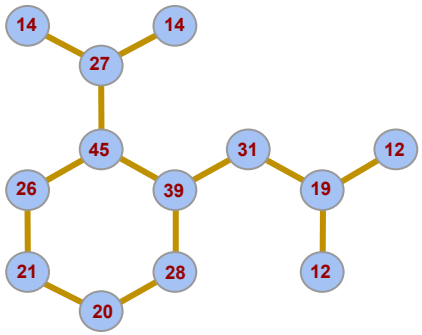
      `prioritizeAndLabel`(v, u);

    end

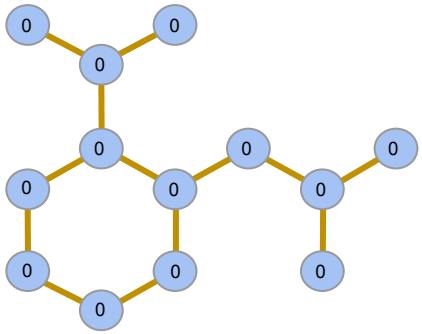
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

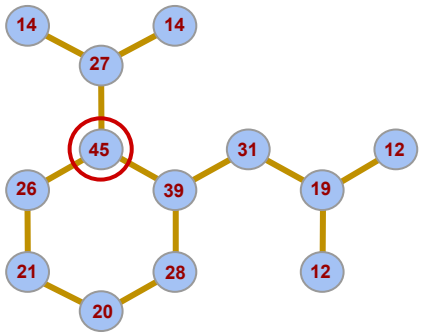
      `prioritizeAndLabel`(v, u);

    end

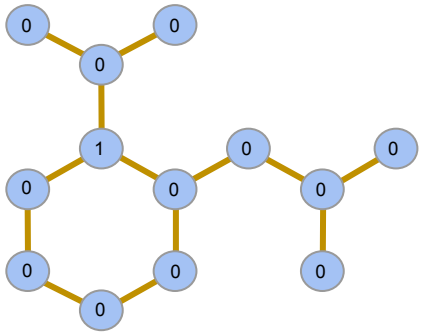
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V$;

$v_a = \text{argmax}_{v \in V} EC(v)$;

$label(v_a) = 1$;

$c = 2$;

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\}$;

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c$;

      $c = c + 1$;

    else

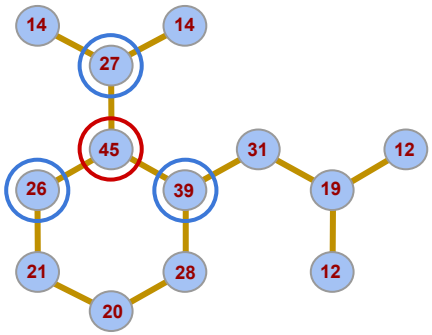
      `prioritizeAndLabel`(v, u);

    end

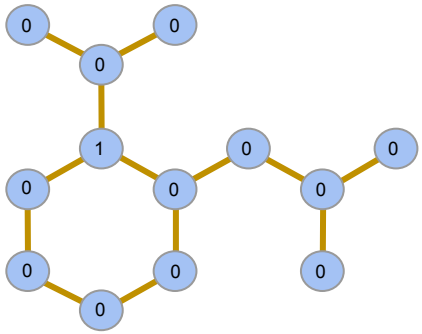
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1$;

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

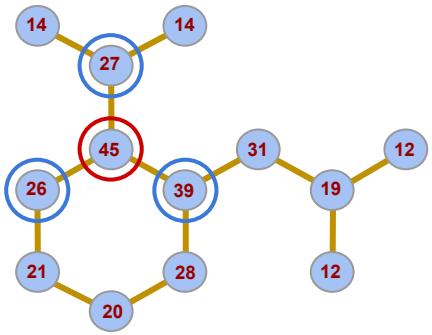
      `prioritizeAndLabel`(v, u);

    end

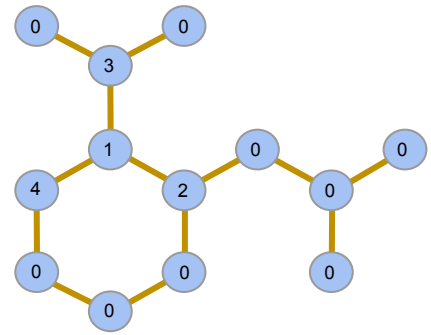
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V$;

$v_a = \text{argmax}_{v \in V} EC(v)$;

$label(v_a) = 1$;

$c = 2$;

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\}$;

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c$;

      $c = c + 1$;

    else

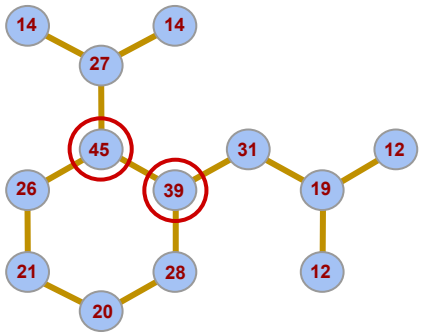
      `prioritizeAndLabel`(v, u);

    end

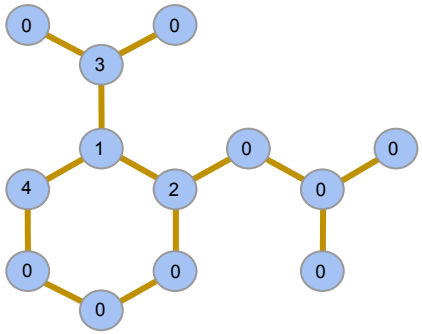
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1$;

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V$;

$v_a = \text{argmax}_{v \in V} EC(v)$;

$label(v_a) = 1$;

$c = 2$;

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\}$;

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c$;

      $c = c + 1$;

    else

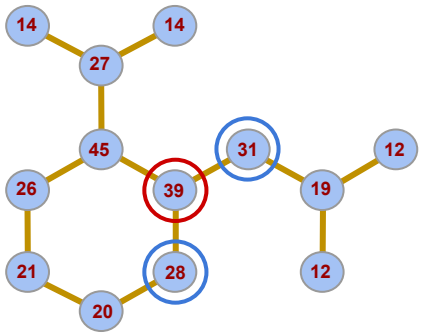
      `prioritizeAndLabel`(v, u);

    end

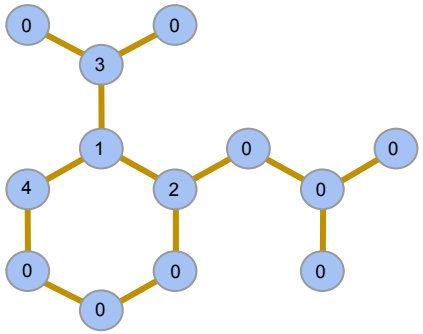
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1$;

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V$;

$v_a = \text{argmax}_{v \in V} EC(v)$;

$label(v_a) = 1$;

$c = 2$;

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\}$;

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c$;

      $c = c + 1$;

    else

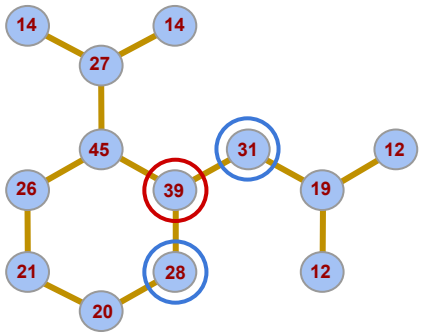
      `prioritizeAndLabel`(v, u);

    end

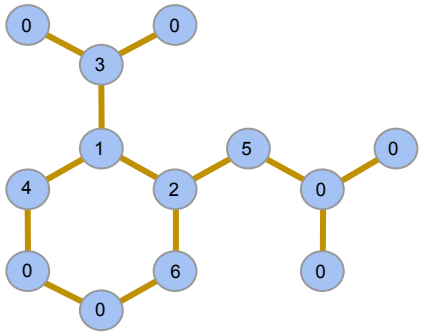
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1$;

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

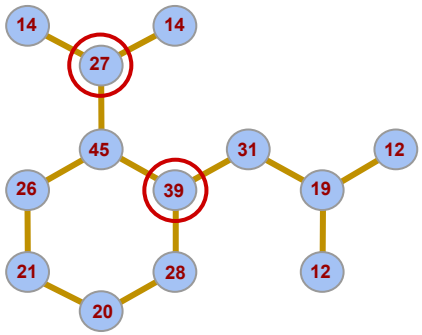
      `prioritizeAndLabel`(v, u);

    end

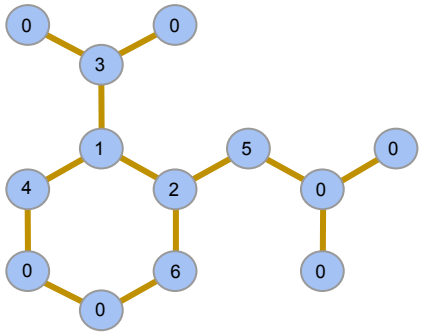
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V$;

$v_a = \text{argmax}_{v \in V} EC(v)$;

$label(v_a) = 1$;

$c = 2$;

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\}$;

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c$;

      $c = c + 1$;

    else

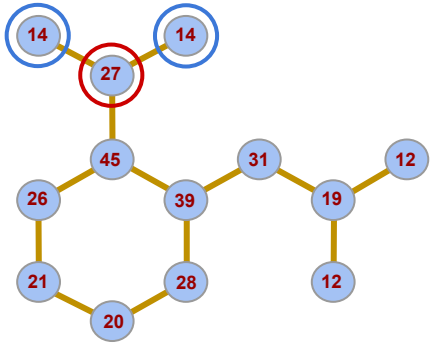
      `prioritizeAndLabel`(v, u);

    end

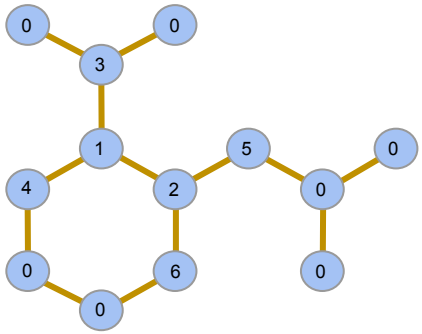
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1$;

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V$;

$v_a = \text{argmax}_{v \in V} EC(v)$;

$label(v_a) = 1$;

$c = 2$;

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\}$;

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c$;

      $c = c + 1$;

    else

      `prioritizeAndLabel`(v, u);

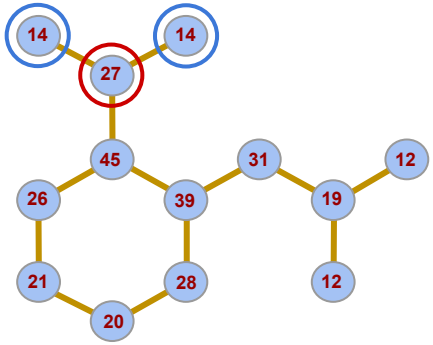
    end

  end

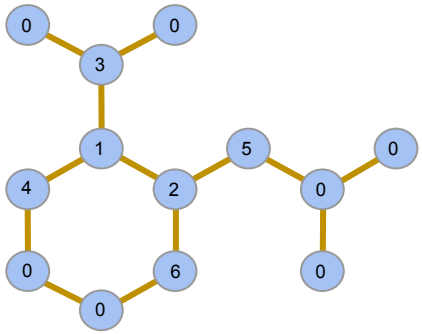
  $v_a = v \in V$ with $label(v) == label(v_a) + 1$;

end

Ambiguity to resolve



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

      `prioritizeAndLabel`(v, u);

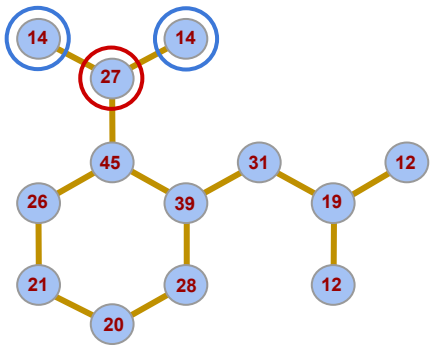
    end

  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end

Ambiguity to resolve



EC(v)

Resolving ambiguities

Implementation has to define rules to resolve such ambiguities. Rules can be like:

- *Atomic type priority*:
  C > N > P > …
- *Bond order priority*:
  single > double > triple > …
- …

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

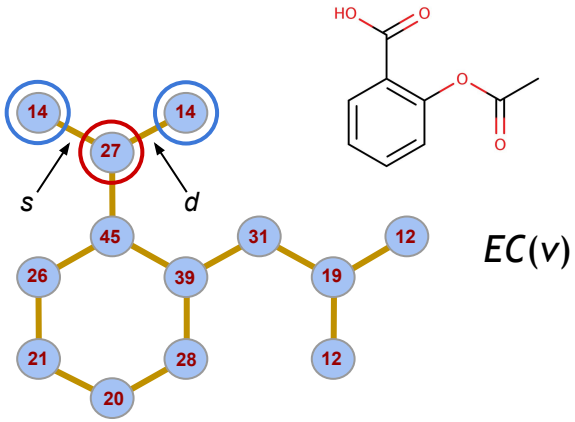
      `prioritizeAndLabel`(v, u); *Ambiguity to resolve*

    end

  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end



Resolving ambiguities

Implementation has to define rules to resolve such ambiguities. Rules can be like:

- *Atomic type priority*:
  C > N > P > …
- *Bond order priority*:
  single > double > triple > …
- …

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

      `prioritizeAndLabel`(v, u);

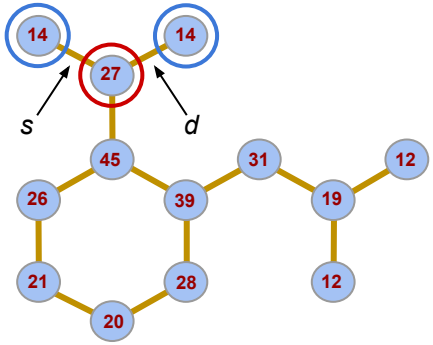
    end

  end

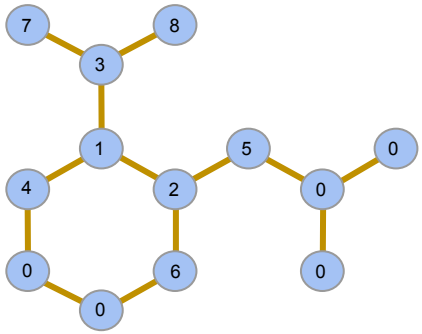
  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end

Ambiguity to resolve



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

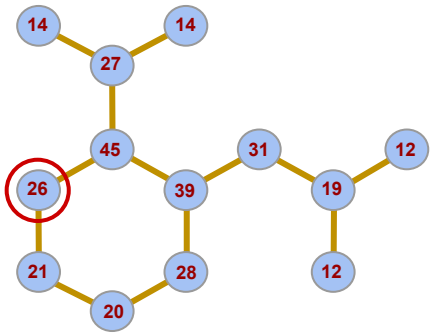
      `prioritizeAndLabel`(v, u);

    end

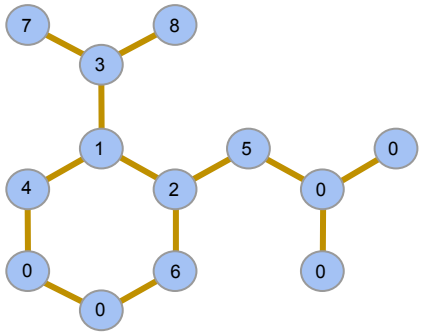
  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end



EC(*v*)



label(*v*)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Canonical Enumeration

In : $G = (V, E)$ with EC node labels

Out: $G = (V, E)$ with canonical node labels

$label(v) = 0 \; \forall \; v \in V;$

$v_a = \text{argmax}_{v \in V} EC(v);$

$label(v_a) = 1;$

$c = 2;$

while $\{v \in V \mid label(v) == 0\} \neq \emptyset$ **do**

  $\mathcal{N} = \{v \in V \mid (v, v_a) \in E \wedge label(v) == 0\};$

  foreach v **in** $\mathcal{N}$ *with decreasing EC label* **do**

    if $\nexists \; u \in \mathcal{N}$ *with* $EC(u) == EC(v)$ **then**

      $label(v) = c;$

      $c = c + 1;$

    else

      `prioritizeAndLabel`(v, u);

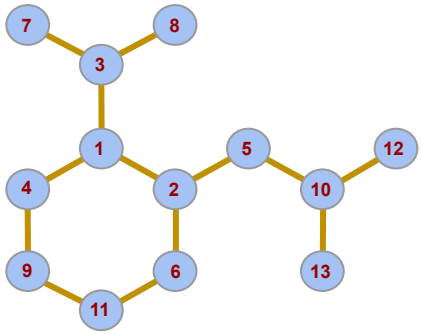
    end

  end

  $v_a = v \in V$ with $label(v) == label(v_a) + 1;$

end

EC(v)



label(v)

1. Morgan H.L. (1965) *J. Chem. Doc.*, 5, 107-13

Morgan Algorithm

Known Problems

- No provably unique enumeration!
- Relaxation step: oscillations
  - Oscillating values of *c* → algorithm does not terminate!
- Canonical enumeration step: ambiguities
  - Not all ambiguities can be resolved
- A few **improved variants** of the algorithm have been proposed.

  Problems **not entirely resolved** but **less likely**.

Morgan Algorithm

Use case: Canonical SMILES

- Also termed Unique SMILES (USMILES)
- David Weininger proposed an algorithm for unique SMILES [1]
  - Also based on Morgan's algorithm
  - Same problems
  - Since 1989 the algorithm had to be changed a few times
  - Algorithmic details of Daylight's USMILES not published

- Canonical SMILES guidelines:
  1. **Always use the same implementation**
  2. **Don't trust other people's canonical SMILES**

1. Weininger D. et al. (1989) *J. Chem. Inf. Comput. Sci.*, 29, 97-101

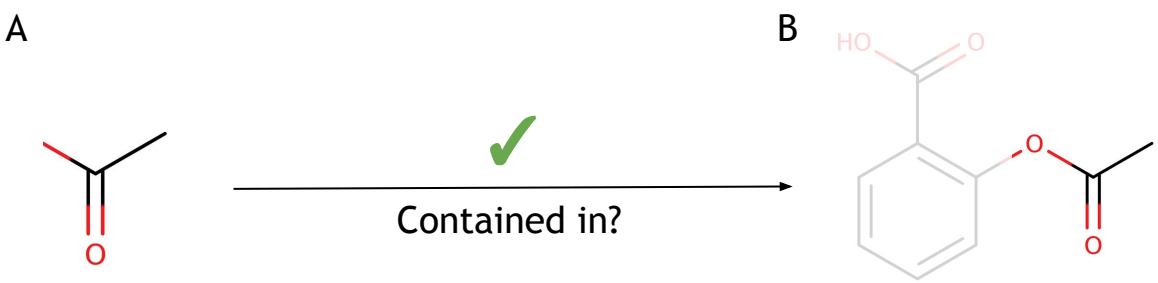
Morgan Algorithm

Closing Remarks

- **Generic canonical enumeration of a molecular graph**
- One of the most important algorithms for this task
- Mainly used to canonize line notations
  - SMILES
  - InChi
- Can also be used for tasks such as
  - Atom numbering in Ctab formats
  - Reaction mapping applications
  - Unique registration of stereoisomers

Motivation

- **Searching for substructures in molecules is more important**
- Basic question: **Is molecule A contained in molecule B?**



- Single problem instance: **substructure matching**
- Applied on a database: **substructure searching**

- Problem can be defined using molecular graph theory

  ⇒ **Subgraph isomorphism problem (SI)**

Motivation

- In order to perform substructure searching on molecular databases we require:

  1. **An algorithm for substructure matching**
  2. **An efficient screening procedure**

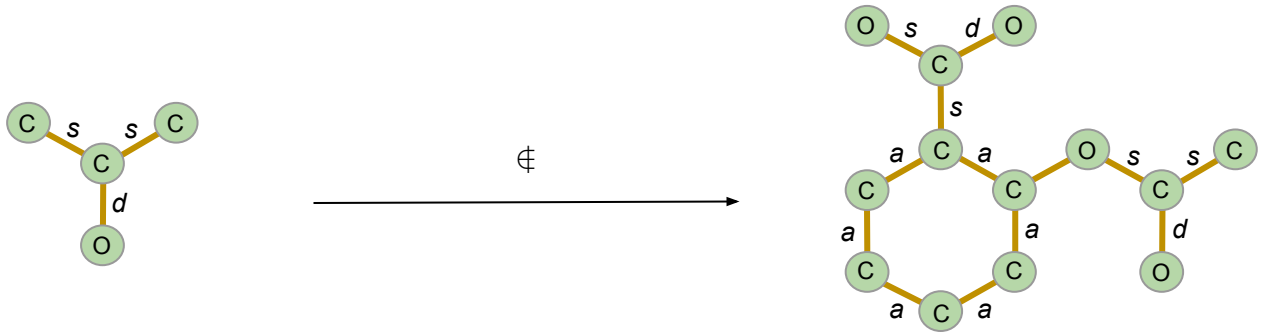
- We will discuss these steps in detail on the following slides

- Identification of interesting substructures to search for will be addressed in *L05 Topological Structure Comparison II*

Problem Definition

Labeled Subgraph Isomorphism

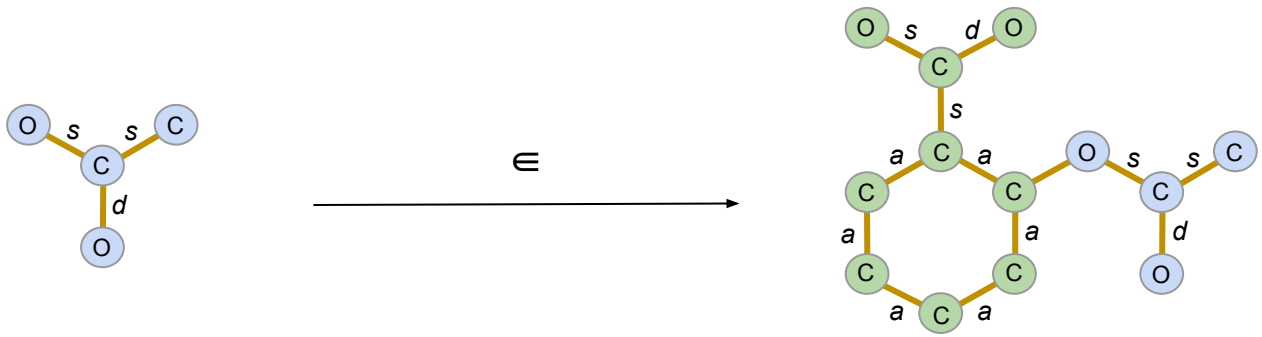
- **Given**: Molecular graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$. Functions $\mu: V_1 \cup V_2 \rightarrow A$ and $\gamma: E_1 \cup E_2 \rightarrow B$ assigning labels to nodes (atoms) and edges (bonds).
- **Problem**: $\exists$ f: $V_1 \rightarrow V_2$, an **injective function**, mapping each $v \in V_1$ to a unique vertex of V_2 such that
  if $(u,v) \in E_1 \Rightarrow (f(u), f(v)) \in E_2$ and
  $\mu(u) = \mu(f(u))$ and $\mu(v) = \mu(f(v))$ and
  $\gamma((u,v)) = \gamma((f(u),f(v)))$



Problem Definition

Labeled Subgraph Isomorphism

- **Given**: Molecular graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$. Functions $\mu: V_1 \cup V_2 \rightarrow A$ and $\gamma: E_1 \cup E_2 \rightarrow B$ assigning labels to nodes (atoms) and edges (bonds).
- **Problem**: $\exists$ f: $V_1 \rightarrow V_2$, an **injective function**, mapping each $v \in V_1$ to a unique vertex of V_2 such that
  if $(u,v) \in E_1 \Rightarrow (f(u), f(v)) \in E_2$ and
  $\mu(u) = \mu(f(u))$ and $\mu(v) = \mu(f(v))$ and
  $\gamma((u,v)) = \gamma((f(u),f(v)))$



Problem Definition

Labeled Subgraph Isomorphism

- Problem is **NP-complete**
- Algorithms scale exponentially with problem size
  - Number of nodes in the graphs
  - Number of atoms in molecules

- No such elegant solution as for identity testing using line notations
- Only a few algorithms have been described
- Existing approaches comprise **exact algorithms** and **heuristics**

- We first introduce **concepts** and sketch a **brute-force approach**

Subgraph Isomorphism

Overview

- **Given**: Labeled molecular graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ with $|V_1| = n$, $|V_2| = m \geq n$ represented by their corresponding **adjacency matrices** A_1 and A_2. G_1 is our query and G_2 our target molecule.

- **Problem**: Find all subgraphs of G_2 that are isomorphic to G_1

- **Definition:** An SI can be defined by a $n \times m$ **permutation matrix** P.

- **Idea:** Enumerate all possible permutation matrices P_i of dimension $n \times m$ and test every P_i if it defines an SI for G_1 and G_2

Subgraph Isomorphism

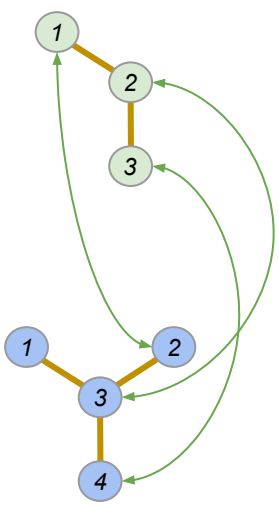
Permutation Matrix

- In a **permutation matrix** *P* rows correspond to nodes in G_1 and columns to nodes in G_2.
- *P* has the following properties:

$$\forall v_i \in V_1 : \sum_{v_j \in V_2} p_{ij} = 1$$

$$\forall v_j \in V_2 : \sum_{v_i \in V_1} p_{ij} \leq 1$$

G_1



G_2

$$A_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$P = \begin{array}{c} \\ v_{11} \\ v_{12} \\ v_{13} \end{array} \begin{array}{c} \begin{array}{cccc} v_{21} & v_{22} & v_{23} & v_{24} \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Subgraph Isomorphism

Subgraph Isomorphism Criterion

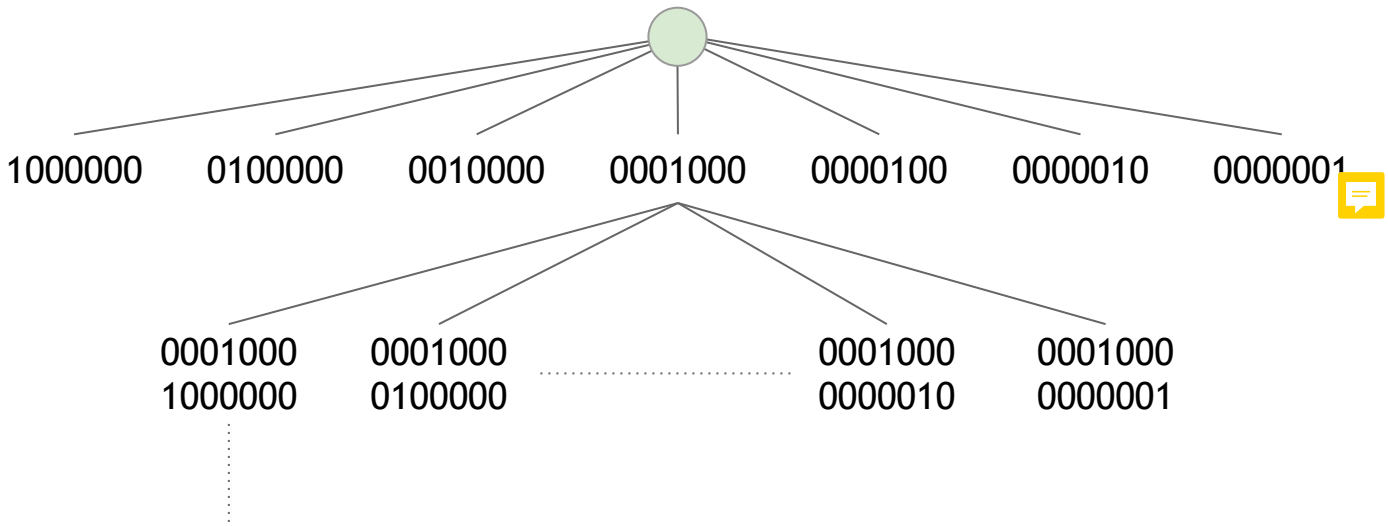
- Not every permutation matrix *P* defines an SI.
- **How to check if *P* defines an SI for G_1 and G_2?**
- **Algebraic check for subgraph isomorphisms exists!**
- If a given *P* defines an SI for G_1 and G_2 the following equation is satisfied:

$$A_1 = P(PA_2)^T$$

Subgraph Isomorphism

Brute-Force Enumeration Algorithm

- Given G_1 and G_2 as adjacency matrices A_1 and A_2
- Enumerate all P of dimension $n \times m$ and test for SI



$$P = \begin{matrix} 0001000 \\ 1000000 \\ 0100000 \\ 0010000 \end{matrix}$$

Leaf nodes are candidate permutation matrices to be tested

$$A_1 = P(PA_2)^T$$

Subgraph Isomorphism

More Efficient Approaches

- Brute-Force enumeration infeasible
  - Grows exponentially with input graph sizes
- Exact methods gain efficiency by
  - Search space reduction
  - Relaxations of the problem
- In cheminformatics two algorithms are mainly used
  - **Ullmann algorithm** (1976) [1]
  - VF2 algorithm (2004) [2]

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Cordella L.P. et al. (2004) *IEEE Trans. Pattern Anal. Mach. Intell.*, 26, 1367-72

Ullmann Algorithm

Overview

- Ullmann proposed an algorithm that works well in practice
- **Tree-traversal** with **backtracking**
- **Enumerates all substructure matches**

- Main ideas:
  - Start by initially excluding impossible subtrees
  - Recursively traverse remaining subtrees
  - Perform **refinement step** for every internal node to reduce the search space

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Overview

- Same conditions as for the brute-force approach:

- **Given**: Labeled molecular graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ with $|V_1| = n$, $|V_2| = m \geq n$ represented by their corresponding **adjacency matrices** A_1 and A_2. G_1 is our query and G_2 our target molecule.

- **Problem**: Find all subgraphs of G_2 that are isomorphic to G_1

- An SI can be defined by a $n \times m$ **permutation matrix** P.

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Compatibility Matrices

- A series of n **compatibility matrices** of size $n \times m$ is constructed
  - Binary matrices
  - $M^0, \ldots, M^{n-1}$ are inner nodes and **no permutation matrices**
  - Leafs M^n are **permutation matrices**
- M^n define valid SI for G_1 and G_2
- Questions:

1. **How to construct initial compatibility matrix M^0 ?**
2. **How to generate inner node compatibility matrices?**
3. **How to refine inner node compatibility matrices?**

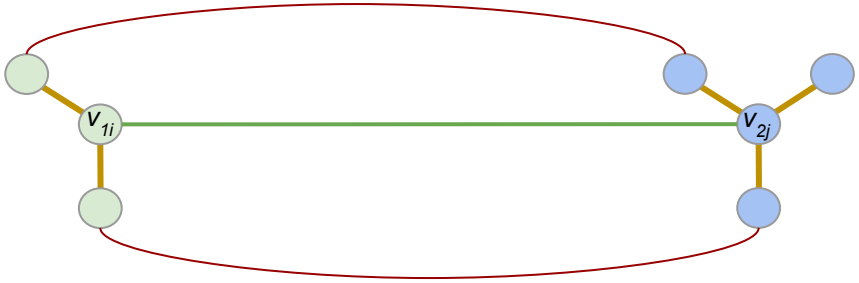
1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Compatibility Matrices: M^0

$$M^0_{ij} = \begin{cases} 1 & \mu(v_{1i}) == \mu(v_{2j}) \text{ and } deg(v_{1i}) \leq deg(v_{2j}) \\ 0 & \text{otherwise} \end{cases}$$

- Initial compatibility matrix M^0:
  - Label matching condition is obvious
  - **Target node degree must have at least degree of query node**



$deg(v_{1i}) = 2$ $deg(v_{2j}) = 3$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

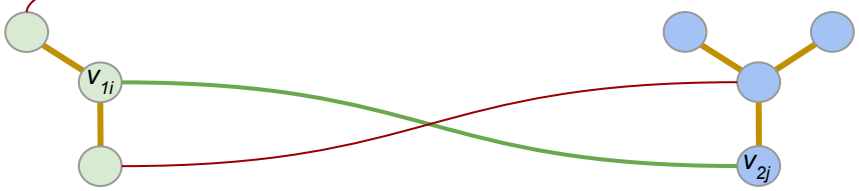
Ullmann Algorithm

Compatibility Matrices: M^0

$$M^0_{ij} = \begin{cases} 1 & \mu(v_{1i}) == \mu(v_{2j}) \text{ and } deg(v_{1i}) \leq deg(v_{2j}) \\ 0 & \text{otherwise} \end{cases}$$

- Initial compatibility matrix M^0:
  - Label matching condition is obvious
  - **Target node degree must have at least degree of query node**

no more matching options for this node
⇒ at least one neighbour of *v* unmatched -> excludes a match



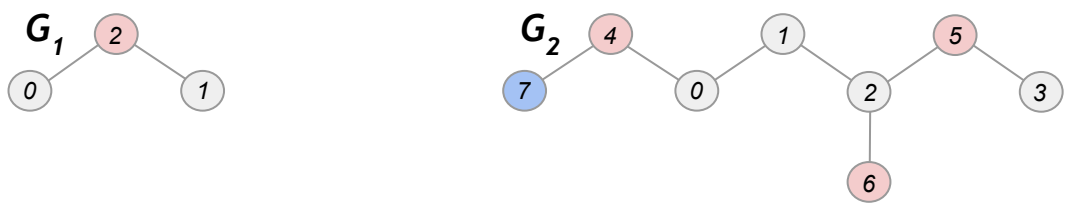
$deg(v_{1i}) = 2$ $deg(v_{2j}) = 1$

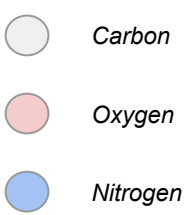
1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Compatibility Matrices: M^0

$$M^0_{ij} = \begin{cases} 1 & \mu(v_{1i}) == \mu(v_{2j}) \text{ and } deg(v_{1i}) \leq deg(v_{2j}) \\ 0 & \text{otherwise} \end{cases}$$





$$M^0 = \begin{array}{c} \\ v_{10} \\ v_{11} \\ v_{12} \end{array} \begin{array}{c} \begin{array}{cccccccc} v_{20} & v_{21} & v_{22} & v_{23} & v_{24} & v_{25} & v_{26} & v_{27} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Degree constraint!

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

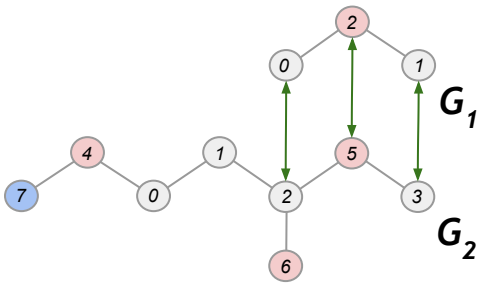
Recursive Ullmann

- Algorithm starts with M^0
- Row by row is processed and all possible subtrees are evaluated
- Internal nodes are **refined**
  - **Subtree pruning** for **search space reduction**
  - This effectively means: set as many as possible m_{ij} = 0 that are 1
- If current node is a leaf:
  1. **Print SI for G_1 and G_2**
  2. **Backtrack**
- If refinement invalidates current node:
  1. **Backtrack**

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Recursive Ullmann



11110000
11110000
00001100

10000000
01110000
00001100

01000000
10110000
00001100

00100000
11010000
00001100

00010000
11100000
00001100

n

00100000
00010000
00001100

P = 00100000
00010000
00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Recursive Ullmann: Refinement

- **Given**: compatibility matrix M and row index k
- Refine all rows of M with row index $i > k$
  1. Visit every mapped node pair ($m_{ij} = 1$)
  2. If node pair is **invalid** set $m_{ij} = 0$
- Repeat until M does not change anymore

- Question:

  When is an existing node mapping invalid?

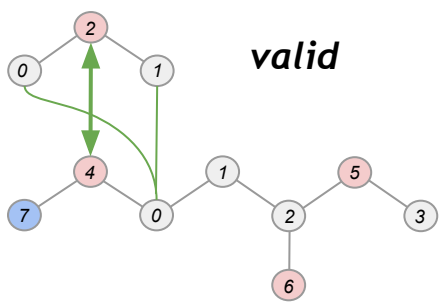
1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

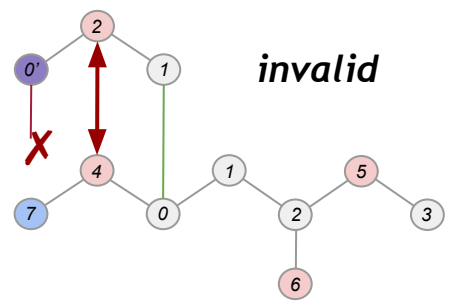
Ullmann Algorithm

Recursive Ullmann: Refinement

- For a valid m_{ij} mapping v_{1i} onto v_{2j} the following condition has to hold:

  All nodes adjacent to v_{1i} must have ≥ 1 valid mapping option.





- Thus, for all (v_{1i} , v_{1r}) we have to find at least one (v_{2j} , v_{2s}) with m_{rs} == 1 and $\gamma(v_{1i}, v_{1r}) == \gamma(v_{2j}, v_{2s})$.

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Recursive Ullmann: Further Ingredients

- Binary Vector **f** of length *m* (size of G_2)
  - f_i == 0: v_{2i} not mapped in a higher recursion
  - f_i == 1: v_{2i} already mapped in a higher recursion

- `filter(M,f,k,l)`
  - Set row *k* and column *l* of *M* to 0
  - Set $M_{kl} = f_l = 1$
  - Example: Selected M_{02}

f:	*00000000*		*00100000*
	11110000	→	00100000
M:	11110000		11010000
	00001100		00001100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

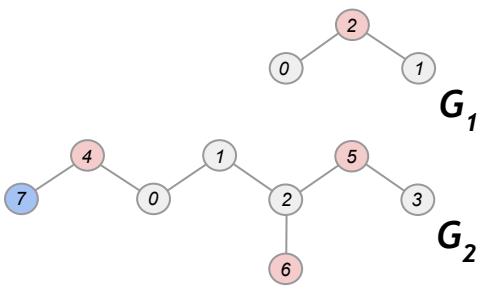
Recursive Ullmann: Further Ingredients

- `initializeCompatibilityMatrix(`A_1`,`A_2`)`
  - Function to generate M^0 from adjacency matrices
- `printSubgraphIsomorphism(`M`,`A_1`,`A_2`)`
  - Print SI defined by M for A_1 and A_2
- `createBackup(`M`,`**f**`)`
  - Store a copy of M and **f**
- `restoreBackup(`M`,`**f**`)`
  - Restore M and **f** from copy

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

k = -1

f_0 *00000000*

M_0
11110000
11110000
00001100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

In : A_1, A_2

Out: All substructure matches for G_1 and G_2

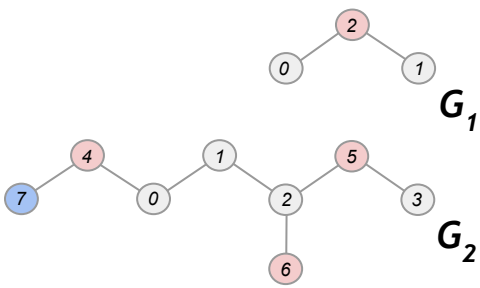
$M = \texttt{initilizeCompatibilityMatrix}(A_1, A_2);$

$k = -1;$

$\mathbf{f} = 0;$

Function `ullmann`$(M, k, \mathbf{f})$

```
    if k == n − 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m − 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```



k = -1

f'	00000000
M'	11110000 11110000 00001100

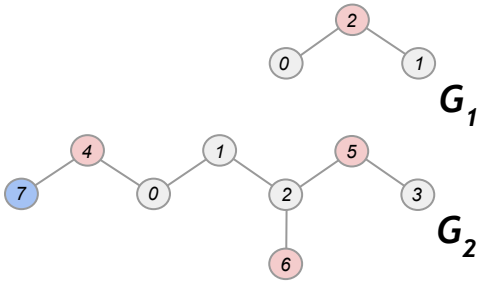
f	10000000
M	10000000 01110000 00001100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

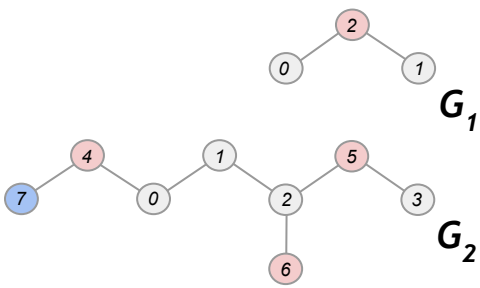
$$M \quad \begin{matrix} 10000000 \\ 01110000 \\ 00001100 \end{matrix}$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

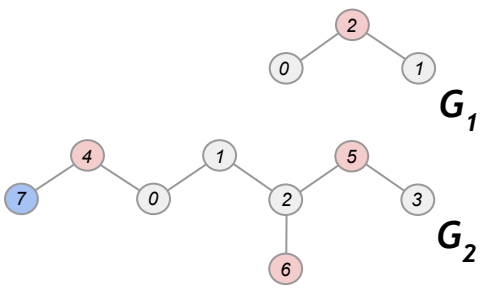
$$M \quad \left.\begin{matrix} 10000000 \\ 01110000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

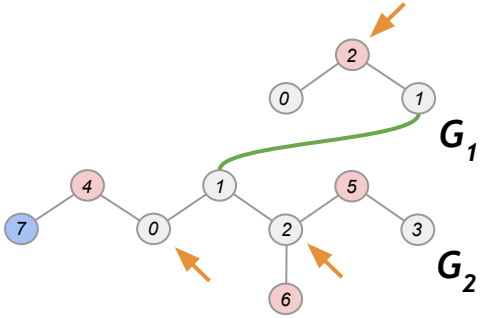
$$M \quad \begin{matrix} 10000000 \\ 01110000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

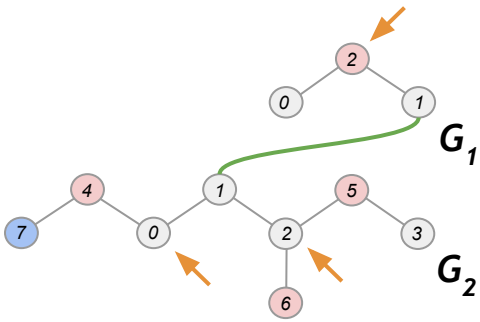
$$M \quad \begin{matrix} 10000000 \\ 01110000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

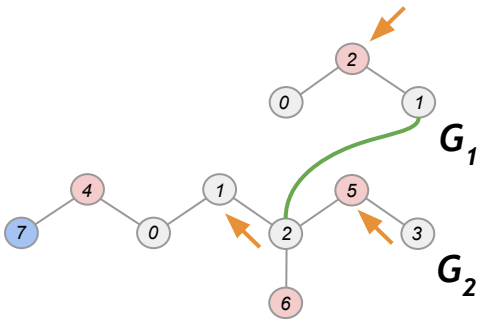
$$M \quad \begin{matrix} 10000000 \\ 00110000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

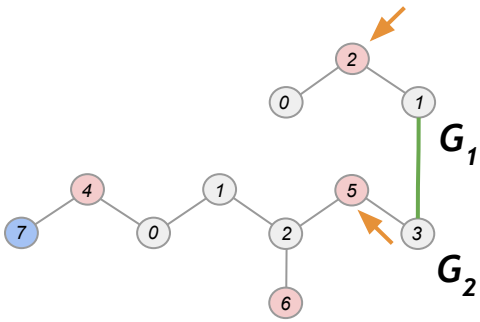
$$M \quad \left.\begin{matrix} 10000000 \\ 00110000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

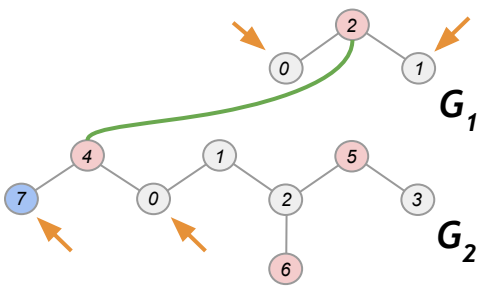
$$M \quad \left.\begin{matrix} 10000000 \\ 00110000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

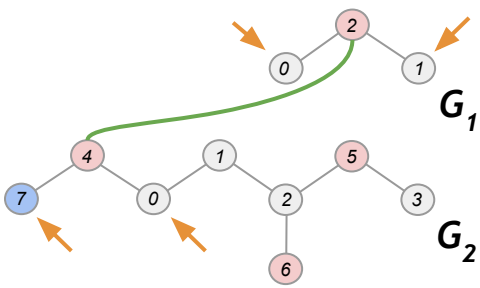
$$M \quad \left.\begin{matrix} 10000000 \\ 00110000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

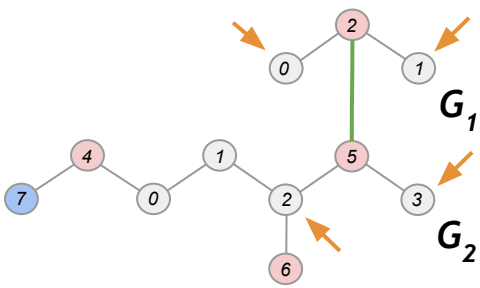
$$M \quad \left.\begin{matrix} 10000000 \\ 00110000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

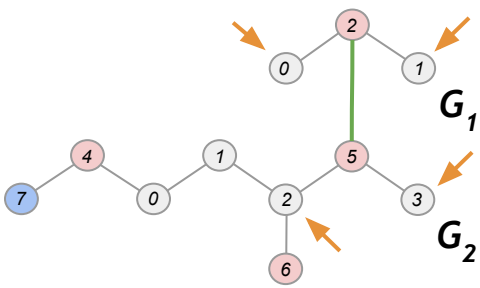
$$M \quad \left.\begin{matrix} 10000000 \\ 00110000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

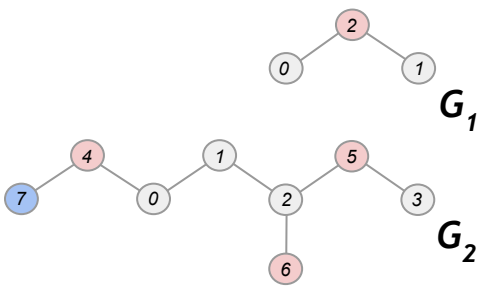
M changed = True

$$M \quad \left.\begin{matrix} 10000000 \\ 00110000 \\ 00000000 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

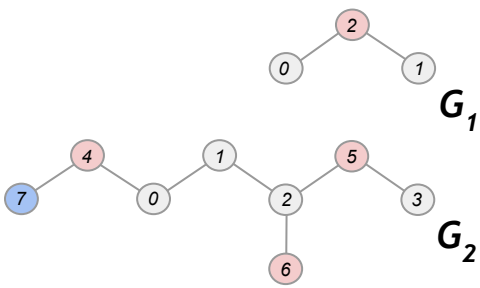
k = -1

f'	*00000000*
M'	11110000 11110000 00001100
f	*00000000*
M	11110000 11110000 00001100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

k = -1

f'	*00000000*
	11110000
M'	11110000
	00001100

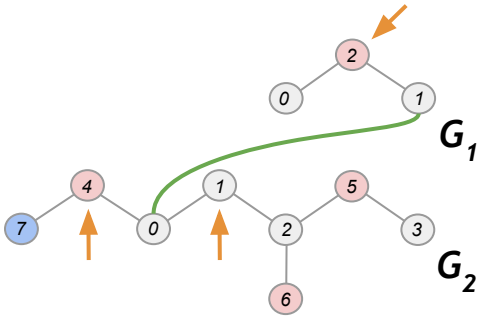
f	*01000000*
	01000000
M	10110000
	00001100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

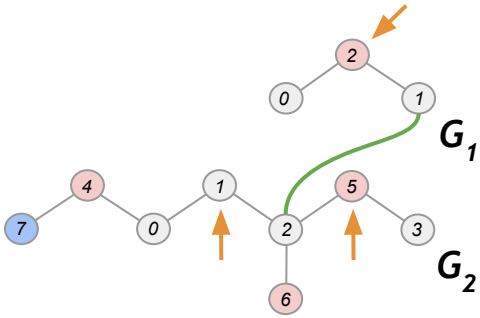
$$M \quad \left.\begin{matrix} 01000000 \\ 10110000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

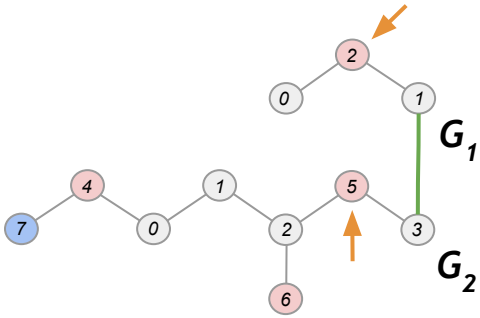
$$M \quad \begin{matrix} 01000000 \\ 10110000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

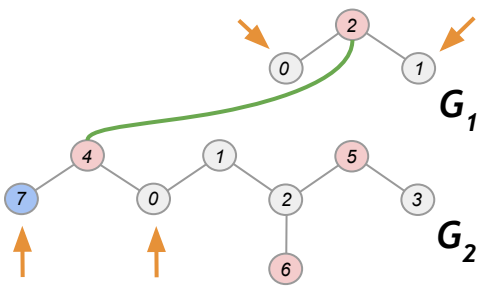
$$M \quad \begin{matrix} 01000000 \\ 10110000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

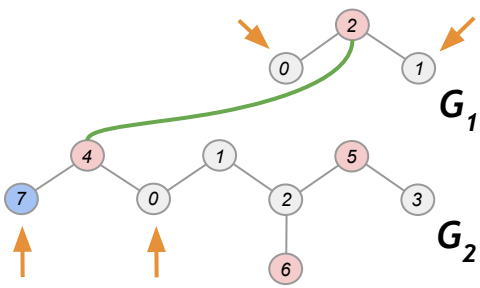
$$M \quad \begin{matrix} 01000000 \\ 10110000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

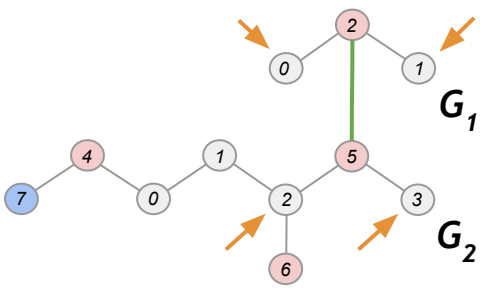
$$M \quad \left.\begin{matrix} 01000000 \\ 10110000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

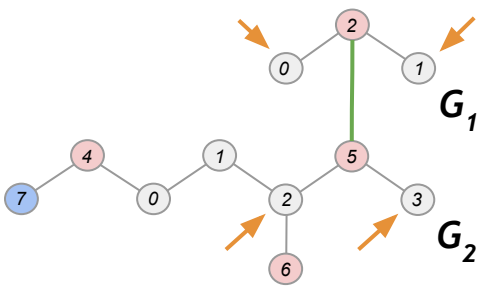
$$M \quad \left.\begin{matrix} 01000000 \\ 10110000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

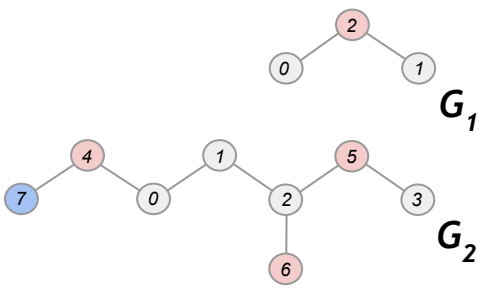
$$M \quad \left.\begin{matrix} 01000000 \\ 10110000 \\ 00000000 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```



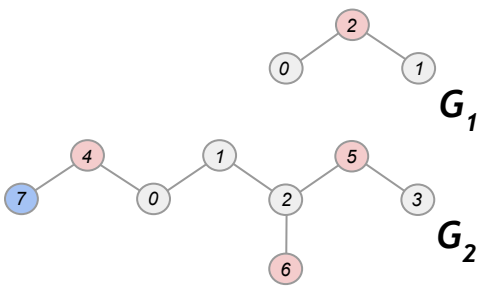
k = -1

f'	*00000000*
M'	11110000 11110000 00001100
f	*00000000*
M	11110000 11110000 00001100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

k = -1

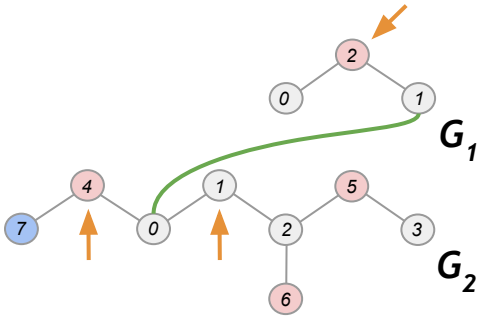
f'	*00000000*
M'	11110000 11110000 00001100
f	*00100000*
M	00100000 11010000 00001100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

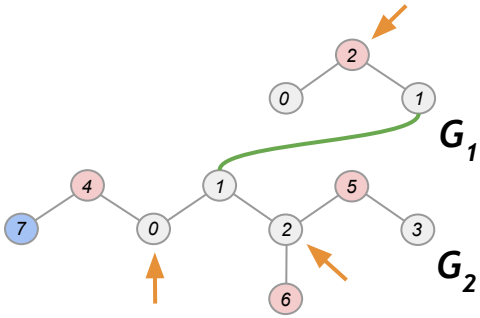
$$M \quad \begin{matrix} 00100000 \\ 11010000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

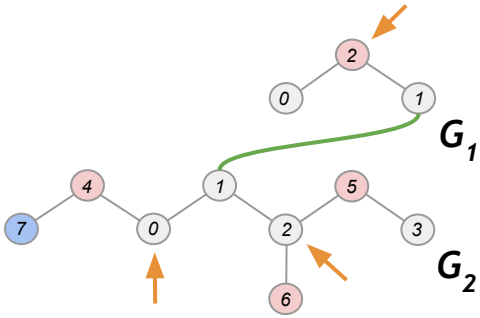
$$M \quad \left.\begin{matrix} 00100000 \\ 11010000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

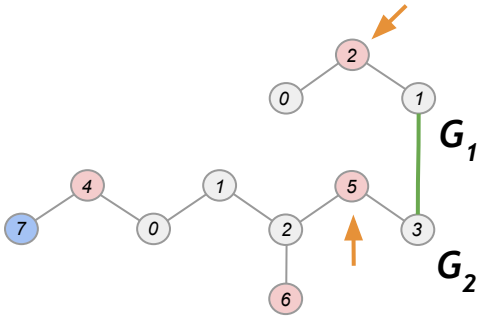
$$M \quad \begin{matrix} 00100000 \\ 10010000 \\ 00001100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

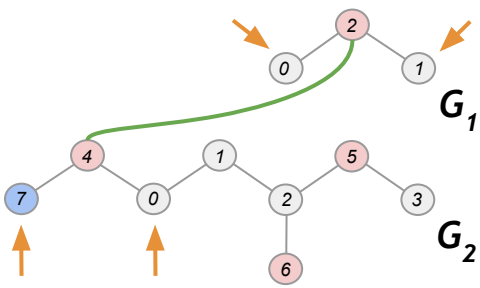
$$M \quad \left.\begin{matrix} 00100000 \\ 10010000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

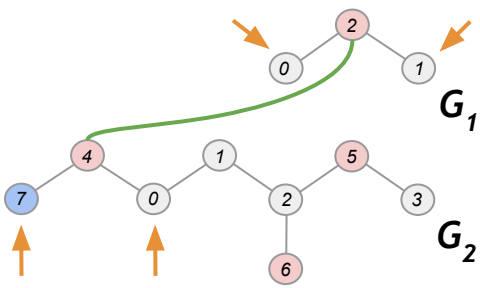
$$M \quad \left.\begin{matrix} 00100000 \\ 10010000 \\ 00001100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

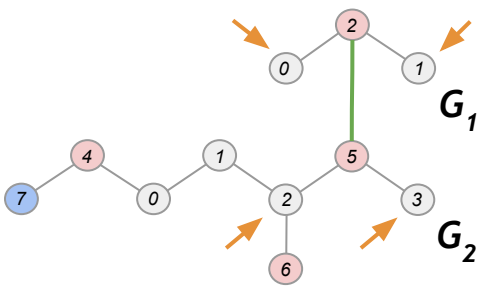
$$M \quad \left.\begin{matrix} 00100000 \\ 10010000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

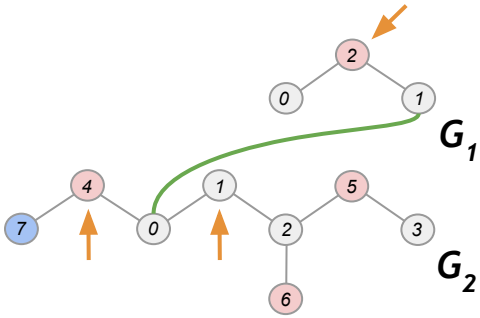
$$M \quad \left.\begin{matrix} 00100000 \\ 10010000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

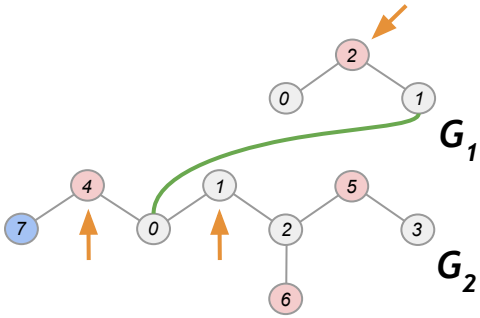
$$M \quad \begin{matrix} 00100000 \\ 10010000 \\ 00000100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

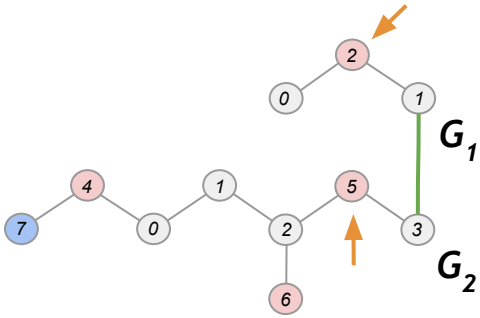
$$M \quad \left.\begin{matrix} 00100000 \\ 00010000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

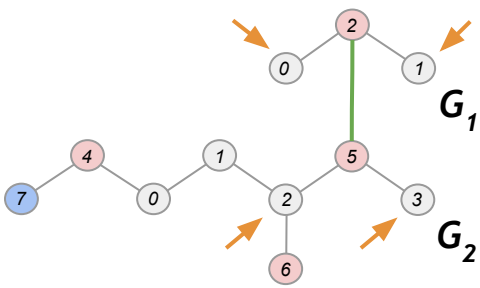
$$M \quad \begin{matrix} 00100000 \\ 00010000 \\ 00000100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = True

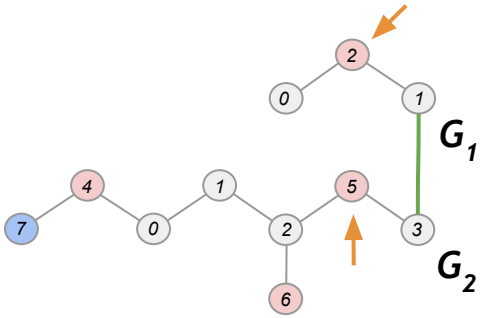
$$M \quad \left.\begin{matrix} 00100000 \\ 00010000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

M changed = False

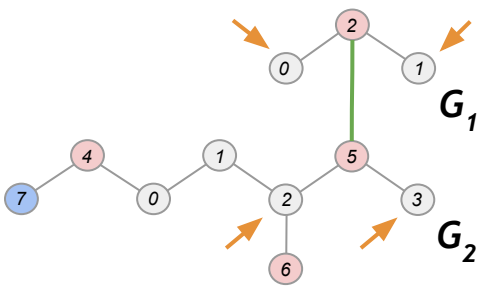
$$M \quad \left.\begin{matrix} 00100000 \\ 00010000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 0$

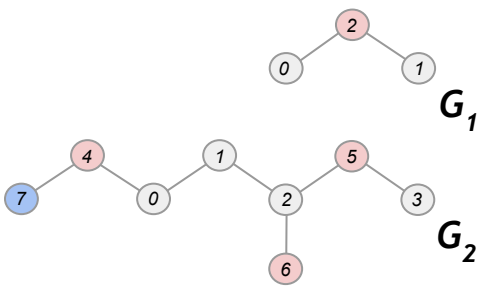
M changed = False

$$M \quad \left.\begin{matrix} 00100000 \\ 00010000 \\ 00000100 \end{matrix}\right\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

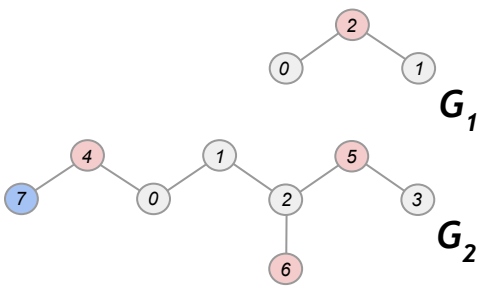
k = -1

f'	*00000000*
M'	11110000 11110000 00001100
f	*00100000*
M	00100000 00010000 00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

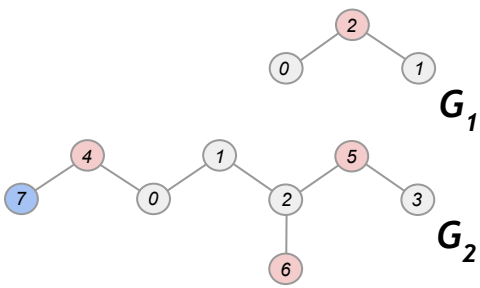
$k = 0$

f	*00100000*
	00100000
M	00010000
	00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

$k = 0$

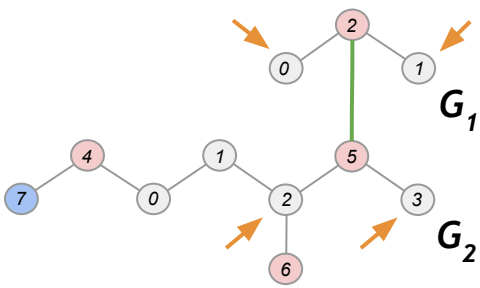
f'	*00100000*
M'	00100000 00010000 00000100
f	*00110000*
M	00100000 00010000 00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



k = 1

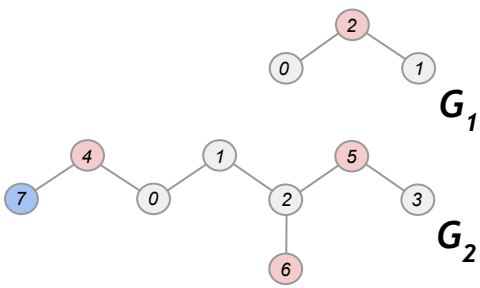
M changed = False

$$M \quad \begin{matrix} 00100000 \\ 00010000 \\ 00000100 \end{matrix} \Big\} > k$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

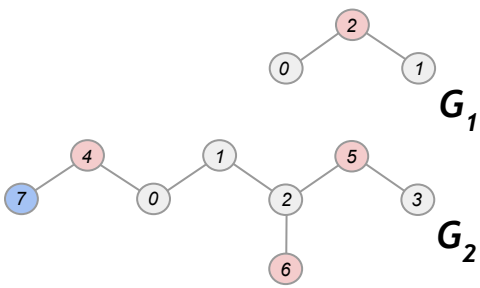
$k = 0$

f'	*00100000*
M'	00100000 00010000 00000100
f	*00110000*
M	00100000 00010000 00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

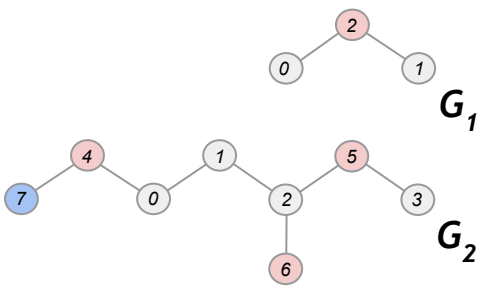
$k = 1$

f	*00110000*
	00100000
M	00010000
	00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

$k = 1$

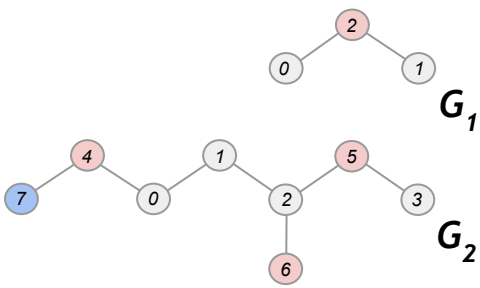
f'	*00110000*
M'	00100000 00010000 00000100
f	*00110100*
M	00100000 00010000 00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann

```
In  : M by reference, k
Out: True if all nodes of A1 can be mapped onto nodes of A2
Function refine(&M, k)
    repeat
        forall M_ij with i > k and M_ij == 1 do
            valid = True;
            forall v_1r adjacent to v_1i do
                found = False;
                forall v_2s adjacent to v_2j do
                    if M_rs == 1 and (v_1i, v_1r) == (v_2j, v_2s) then
                        found = True;
                        break;
                    end
                end
                if not found then
                    valid = False;
                    break;
                end
            end
            if not valid then
                M_ij = 0;
                if M_ih = 0 for 0 ≤ h ≤ m − 1 then
                    return False;
                end
            end
        end
    while M has been modified;
    return True;
end
```



$k = 2$

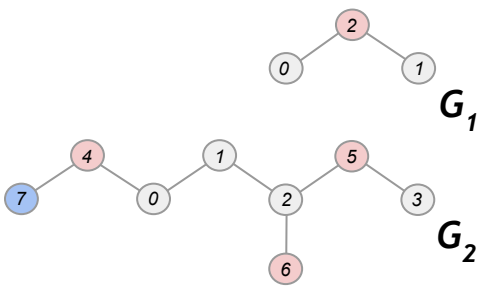
M changed = False

$$M \quad \begin{matrix} 00100000 \\ 00010000 \\ 00000100 \end{matrix}$$

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

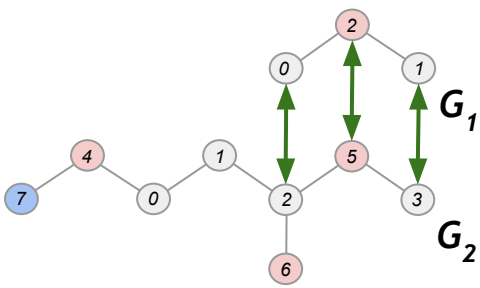
$k = 1$

f'	*00110000*
M'	00100000 00010000 00000100
f	*00110100*
M	00100000 00010000 00000100

1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Recursive Ullmann




```
In  : A1, A2
Out: All substructure matches for G1 and G2
M = initilizeCompatibilityMatrix(A1, A2);
k = -1;
f = 0;
Function ullmann(M, k, f)
    if k == n - 1 then
        printSubgraphIsomorphism(M, A1, A2);
    else
        for l = 0 to m - 1 with M_{k+1,l} == 1 and f_l == 0 do
            createBackup(M, f);
            filter(M, f, k + 1, l);
            if refine(M, k + 1) then
                Ullmann(M, k + 1, f);
            end
            restoreBackup(M, f);
        end
    end
end
```

$k = 2$

f	*00110100*
	00100000
M	00010000
	00000100

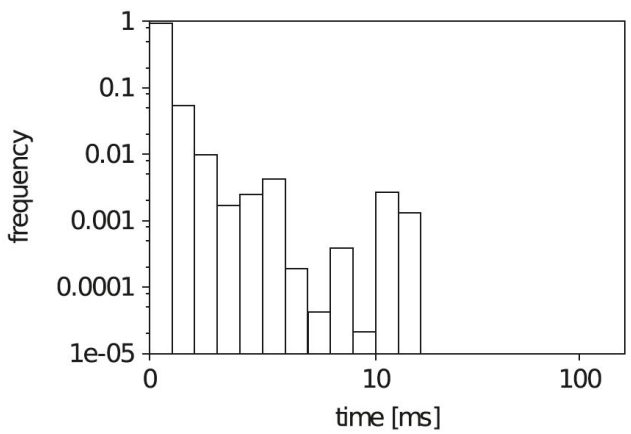
SI: (0,2), (1,3), (2,5)

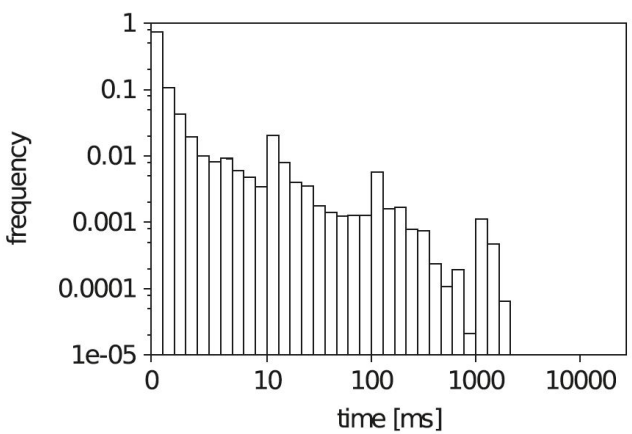
1. Ullmann J.R. (1976) *J. Assoc. Comput. Mach.*, 23, 31-42
2. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Ullmann Algorithm

Benchmark

- Ehrlich and Rarey benchmarked Ullmann and VF2
- Both algorithms have **average matching times below 1ms**
- VF2 has less outliers but sensitive to SMARTS formulation
- Ullmann has more outliers but is insensitive to SMARTS formulation
- VF2 (left) and Ullmann (right) for all occurrences matching:





1. Ehrlich H.C. and Rarey M. (2012) *J. Cheminf.*, 4, 13

Substructure Matching

Closing Remarks

- Efficiency acceptable for small data sets

- Substructure matching **too slow for database search**
  - On average 1 ms per structure
  - 100M (PubChem) structures take ~27 hours

- Thus, an efficient search / screening procedure required!

Efficient Database Searching

Overview

- Substructure search on databases has to be efficient
- Usually, a **two step search procedure** is applied [1]
- **Idea**: given a substructure query do
  1. **Fast elimination search** to remove impossible matchings
  2. Apply subgraph matching on remaining candidates
- First step ideally **eliminates vast majority** of molecules

1. Barnard J. (1993) *J. Chem. Inf. Comput. Sci.*, 33, 532-8

Fast Elimination Search

Overview

- **Problem**:

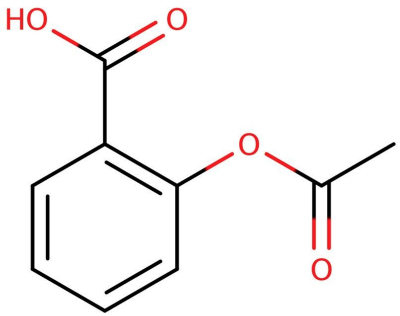
  Efficient identification of database structures

  that cannot contain query substructure.

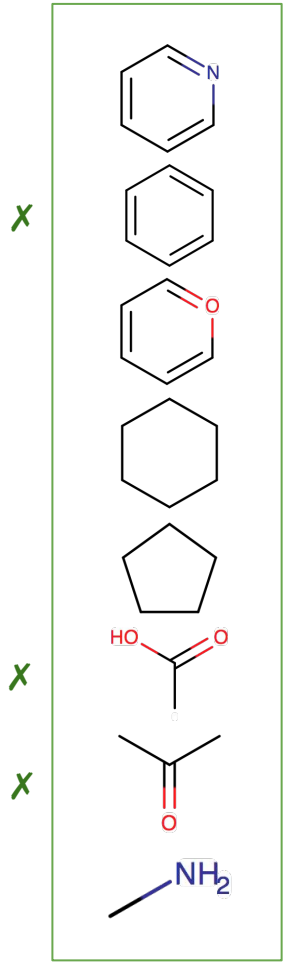
- **Idea**:
  - Use set of **predefined fragments**
    - I.e. substructures, also referred to as **features**
  - Enumerate **presence** of features in **every database molecule**
  - Enumerate **presence** of features in **query molecule**
  - Test every database molecule for **a missing query feature**

Fast Elimination Search

Example



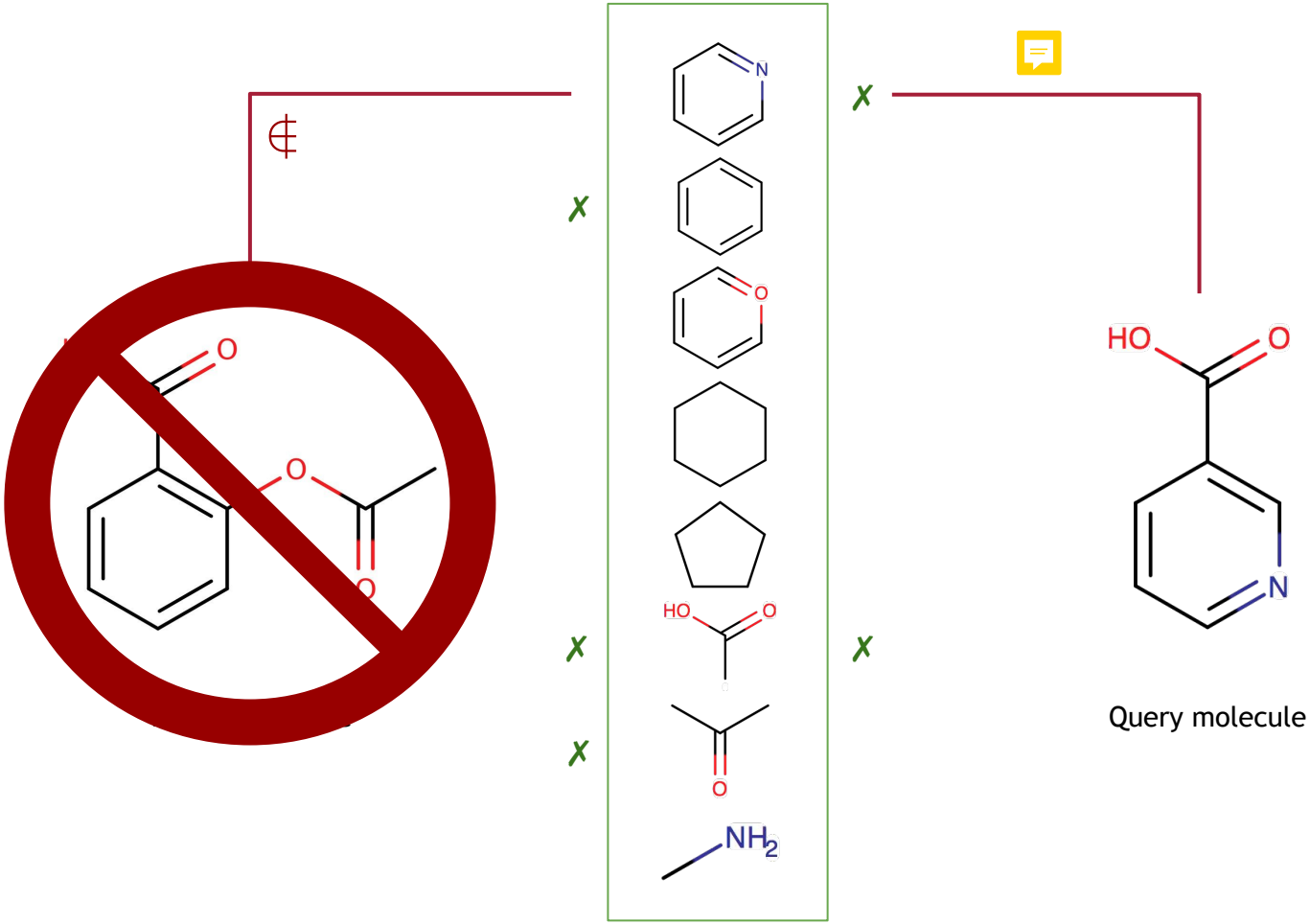
Target molecule



Feature list

Fast Elimination Search

Example



Feature list

Fast Elimination Search

Feature Lists

- **Feature lists**: collection of **predefined fragments**
  - Small substructures < 10 non-hydrogen atoms
  - Often functional groups

- Desired fragment properties
  - Independent of each other
  - No frequent fragments
    - Not discriminating
  - No infrequent fragments
    - Discriminating but by definition a rare case
    - Thus not very useful

Fast Elimination Search

Feature Lists

- Much work spent on generation of **effective fragment sets**
  - E.g. based on database analysis [1]
  - Only efficient on a certain applicability domain
    - Organic database feature list inefficient on inorganic database

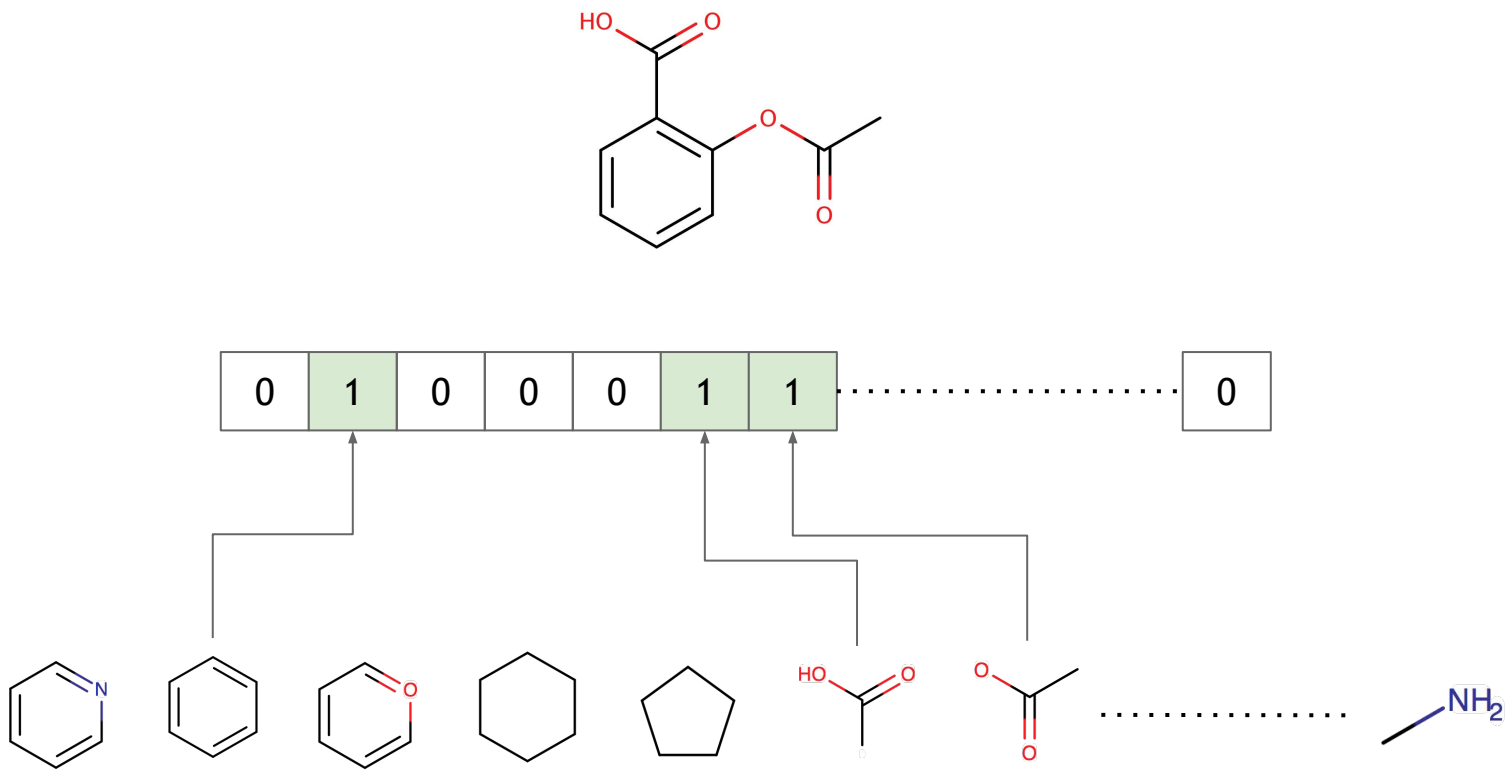
- Efficient encoding as **bitstrings** or **bitvectors**
  - Each position corresponds to one fragment
  - 0-bit := corresponding fragment is absent
  - 1-bit := corresponding fragment is present

- This special type is known as **structural keys**

1. Willett P. (1979), *J. Chem. Inf. Comput. Sci.*, 19, 159-62

Fast Elimination Search

Feature Lists: Example

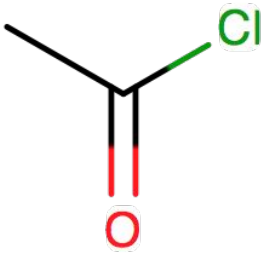


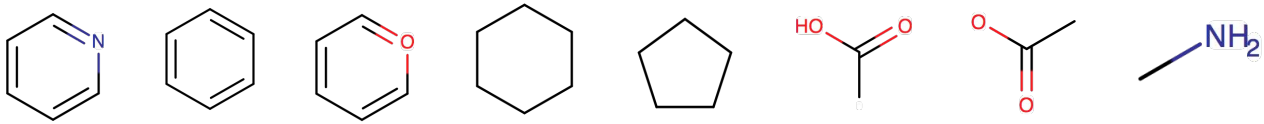
Fast Elimination Search

Closing Remarks

- Searching with bitvectors can be **implemented efficiently**
- A **compact representation** of a molecule
- Can be **precalculated** for all database molecules

- **Possible problem**:
  No fragment in feature list matching our query structure





Summary

- Molecular identity can be described as LGI: complex problem
- Identity check with line notations is fast: string comparison
- Canonical labeling of molecular graph required
- Morgan's graph canonization is the major algorithm
- Substructure search is more important
- Subgraph isomorphism: NP-complete
- Ullmann algorithm frequently used and rather efficient
- Large scale substructure search must be more efficient
- Fast elimination search using structural keys

Text Books:

- GJ — Garey M. and Johnson D.S., W. H. Freeman & Co., New York, 1979
  Computers and Intractability: A Guide to the Theory of NP-Completeness
- GE — Gasteiger J. and Engel T. (Eds.), 1st Ed., Wiley-VCH, 2003
  Chemoinformatics - A Textbook
- KA — Kerber A. et al.
  Mathematical Chemistry and Chemoinformatics, De Gruyter, 2014

Acknowledgments:

- 2D structure drawings were generated with ChemAxon **MarvinSketch**
  - `https://www.chemaxon.com/products/marvin/marvinsketch`
- 3D structures were generated with **BALLView**
  - `http://www.ball-project.org`
  - Hildebrandt A. et al. (2010) *BMC Bioinformatics*, 11, 531
  - Moll A. et al. (2006) *Bioinformatics*, 22, 365-6