b-it LSi
Life
Science
Informatics

LIMES
Life & Medical Sciences Institute

Algorithms in Chemoinformatics Canonical Representations and Substructure Searching

Martin Vogt
B-it Life Science Informatics
Rheinische Friedrich-Wilhelms-Universität Bonn

15 May 2019

UNIVERSITÄT BONN

Classical Algorithmic Chemoinformatics Problems

- Identifying identical molecular structures
  - Two molecules are considered the same if they have the same molecular graph

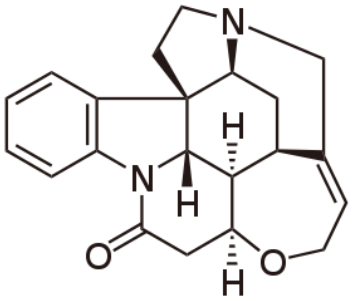
- Identifying substructures in a molecule
  - A (fragment) structure is part of a molecule if it corresponds to a subset of atoms and their corresponding bonds

Classical Algorithmic Chemoinformatics Problems

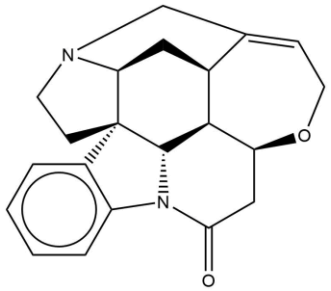
- Identifying identical molecular structures
  - Two molecules are considered the same if they have the same molecular graph
  - Mathematical formulation
    Graph isomorphism problem:
    Two graphs are isomorphic (the same) if there is an edge-preserving one-to-one correspondence between their vertices
- Identifying substructures in a molecule
  - A (fragment) structure is part of a molecule if it corresponds to a subset of atoms and their corresponding bonds
  - Mathematical formulation
    Subgraph isomorphism problem:
    A graph S is a subgraph of a graph G if S is isomorphic to a subgraph of G

Canonical Representations of Molecules

- "Old" problem of chemistry
- Problem: How to uniquely identify molecular structures
- Molecules are not "linear" but graphs
- "Naming" molecules in a unique way:
  - Systematically derive a name from its structure
  - Allows identification, cataloging of molecules, i.e. makes databases searchable
- Identifying molecules is "the same as" identifying isomorphic graphs
- Even for small molecules identity might not be obvious from inspection:



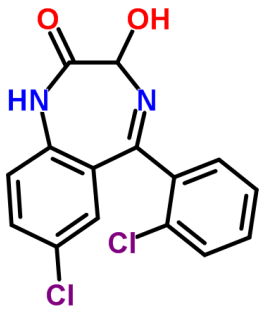
strychnin, source: en.wikipedia.org



strychnin, source: ChemDraw

Linear Representations

Structural graphs

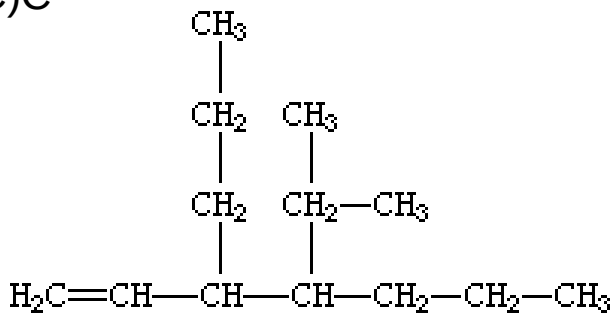


can be represented using linear representations

- Systematic name
  - 7-Chloro-5-(2-chlorophenyl)-3-hydroxy-1,3-dihydro-2H-1,4-benzodiazepin-2-one
- Smiles
  - `Clc3ccccc3C/2=N/C(O)C(=O)Nc1c\2cc(Cl)cc1`
- InChI
  - `InChI=1S/C15H10Cl2N2O2/c16-8-5-6-12-10(7-8)13(19-15(21)14(20)18-12)9-3-1-2-4-11(9)17/h1-7,15,21H,(H,18,20)`

Smiles Recap

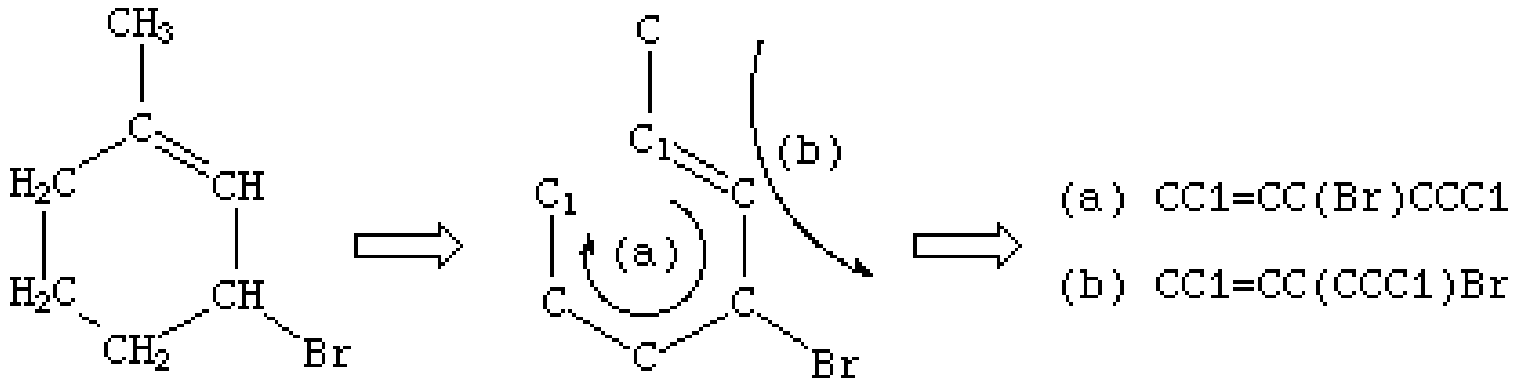
- Acyclic molecules:
  - SMILES is a linear representation of the atoms and bonds of a molecule.
  - Atoms are represented by their chemical symbol
    - B, C, N, O, S, P, F, Cl, Br, I
  - Hydrogens are implicit
  - Adjacent atoms in a SMILES representation are connected by a bond.
  - Special symbols can be used to indicate the order of the bond
    - -, =, #, :
  - Branches can be specified by enclosing them in parentheses
    - Isobutane: CC(C)C



C=CC(CCC)C(C(C)C)CCC

Smiles Recap

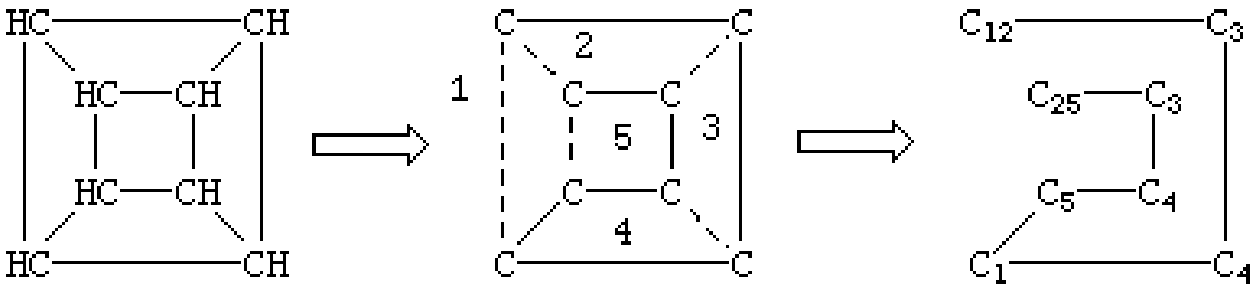
- Cyclic molecules
  - Cycles are “broken” by removing a bond
  - Broken bonds are marked by numerical indices
  - The resulting tree is linearized
  - Atoms of broken bonds are annotated with the corresponding indices



Images from: http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html

Smiles Recap

- Cyclic molecules
  - Cycles are “broken” by removing a bond
  - Broken bonds are marked by numerical indices
  - The resulting tree is linearized
  - Atoms of broken bonds are annotated with the corresponding indices



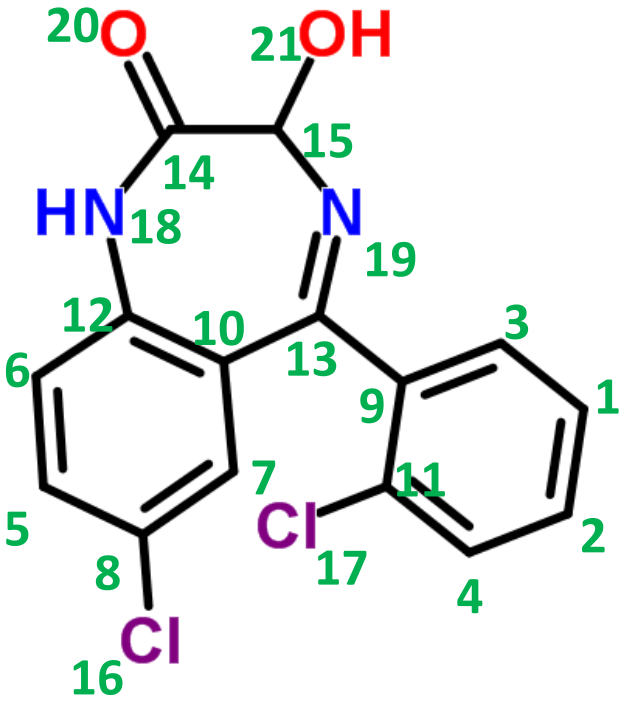
C12C3C4C1C5C4C3C25

Images from: http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html

InChI

- InChI describe molecules in layers:
  - molecular formula
  - atom connectivity (topology)
  - hydrogen locations (bond orders)
  - charges
  - stereochemistry
  - …
- Atoms are numbered according to molecular formula
- Connectivity is described similar to SMILES:
  - each atom is defined by its number
  - no “ring-closure” numbers, instead numbers can be reused

InChI

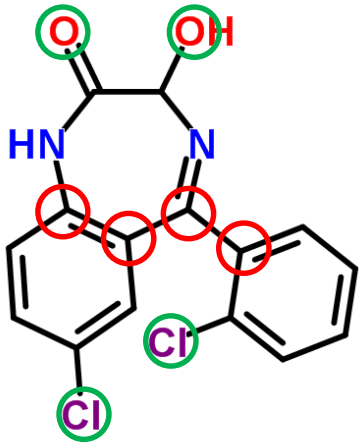


InChI=1S/C15H10Cl2N2O2/c16-8-5-6-12-10(7-8)13(19-15(21)14(20)18-12)9-3-1-2-4-11(9)17/h1-7,15,21H,(H,18,20)

- InChI=1S
- C15H10Cl2N2O2
- 16-8-5-6-12-10(7-8)13(19-15(21)14(20)18-12)9-3-1-2-4-11(9)17
- h1-7,15,21H,(H,18,20)

Canonical Representations

- Representations like Smiles and InChI depend on the order of traversal of atoms, i.e.
  - What is the first atom of the representation?
  - At each branch-point: Which path to follow first?
- Idea for canonicalization:
  Assign priorities to atoms based on the topology
  - Priorities determine order of traversal
  - Numbering of atoms in InChI depends on priorities
    - Although there is no direct correspondence



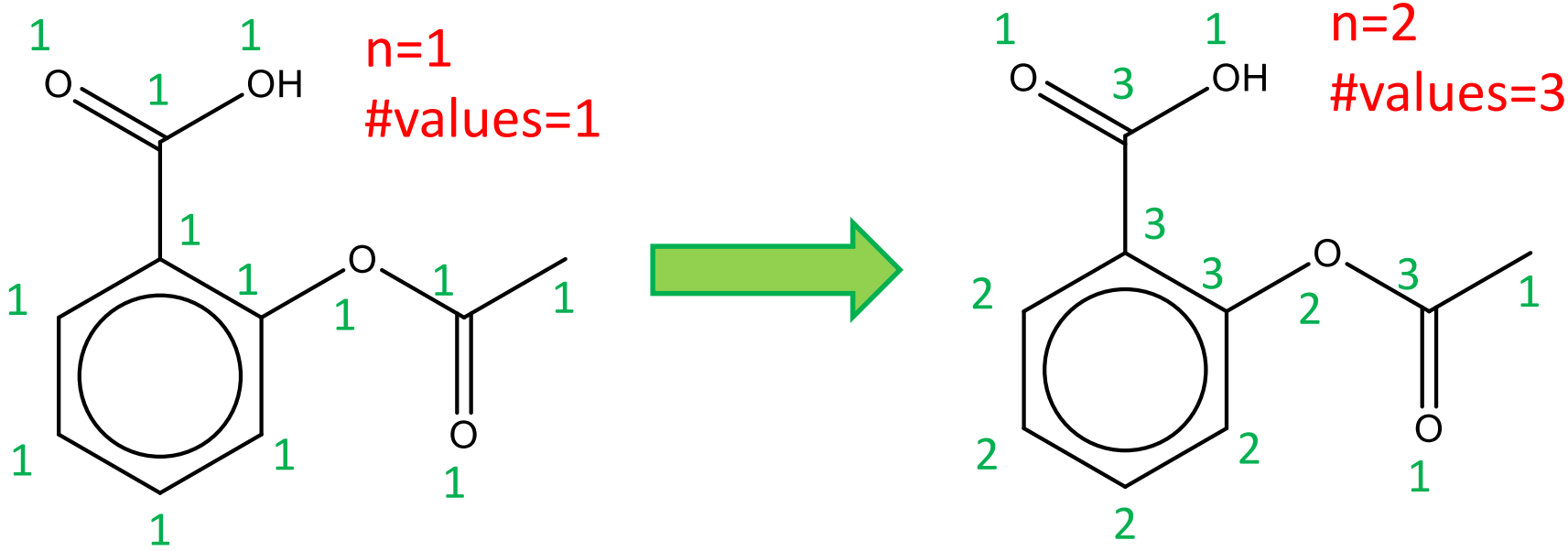
Canonical Labeling: Basic Idea

- Assign initial invariants to atoms, encoding local information of the atoms
  - Number of neighbors
  - Atom type
  - Number of hydrogens
  - …
- Update invariant based on neighboring invariant

- Repeat until atoms are disambiguated
- … or no more atoms can be differentiated

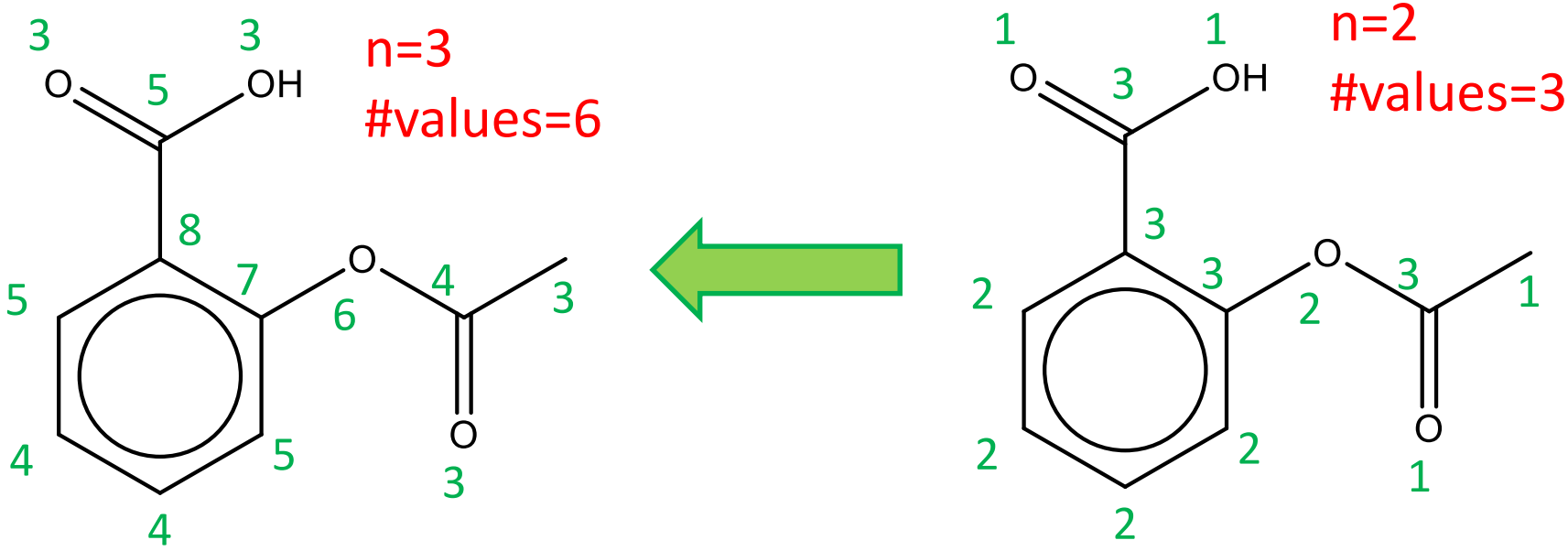
Morgan Algorithm (Simple)

1. Assign initial invariant of 1
2. New invariant: Sum of neighboring values
3. Determine number of values



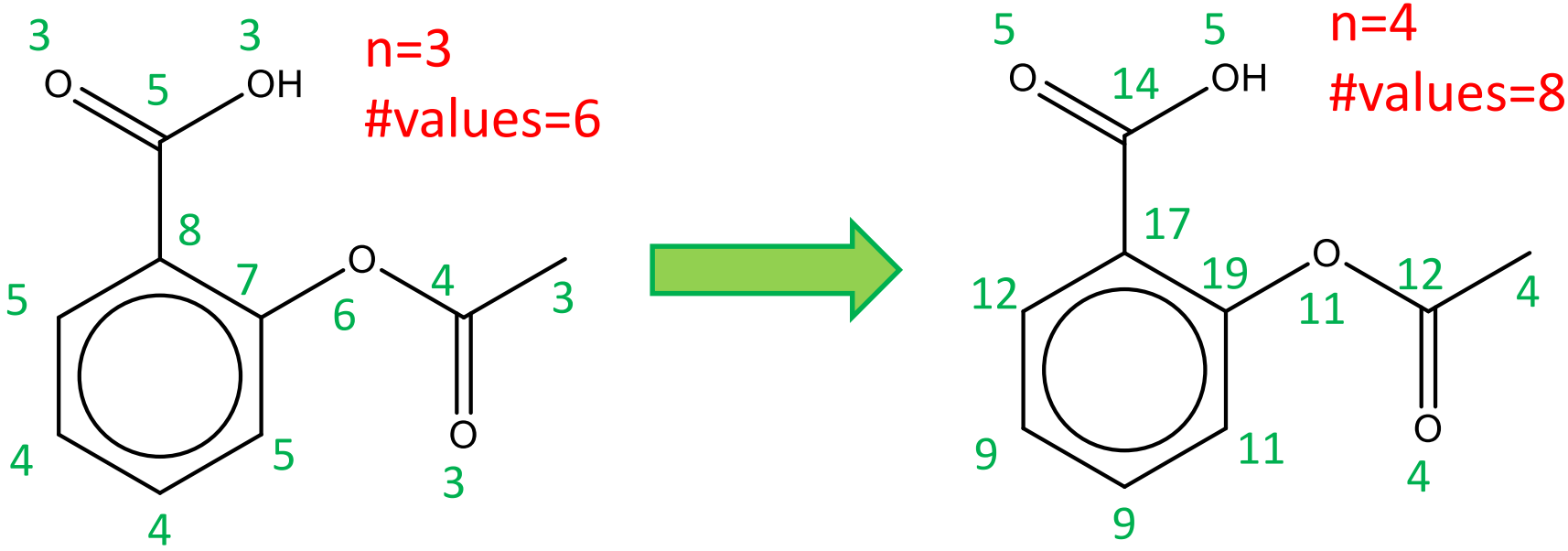
Morgan Algorithm (Simple)

- Repeat summing of neighboring values



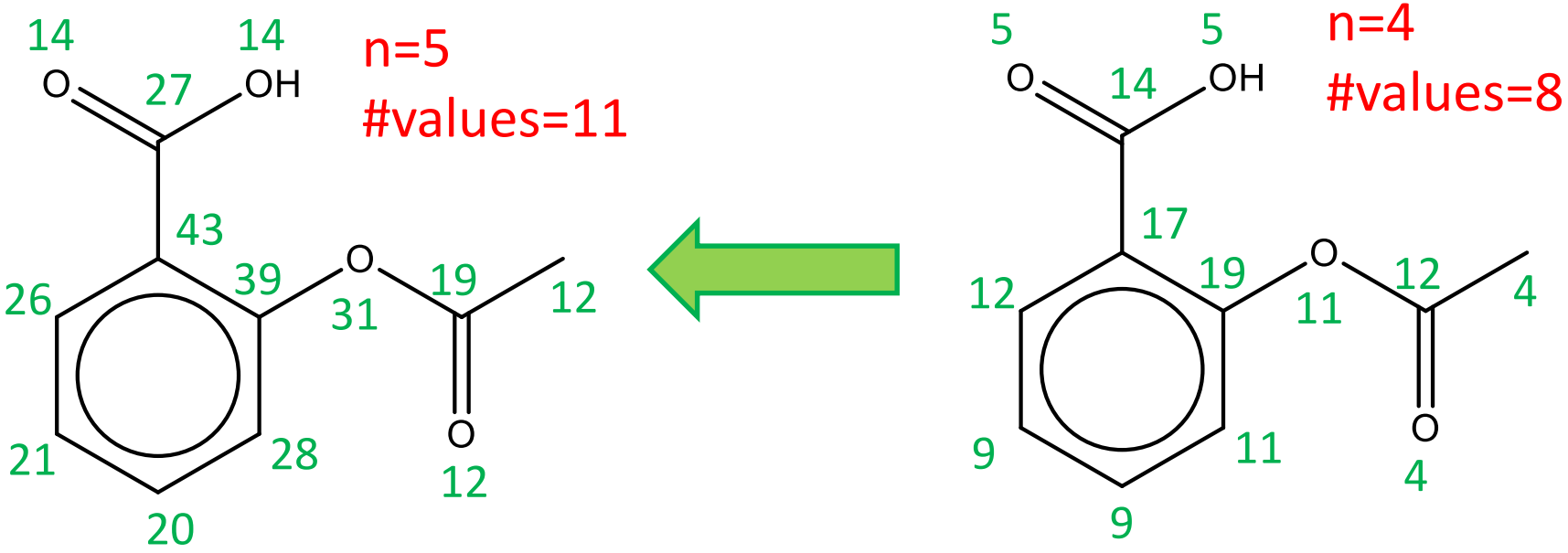
Morgan Algorithm (Simple)

- Repeat summing of neighboring values



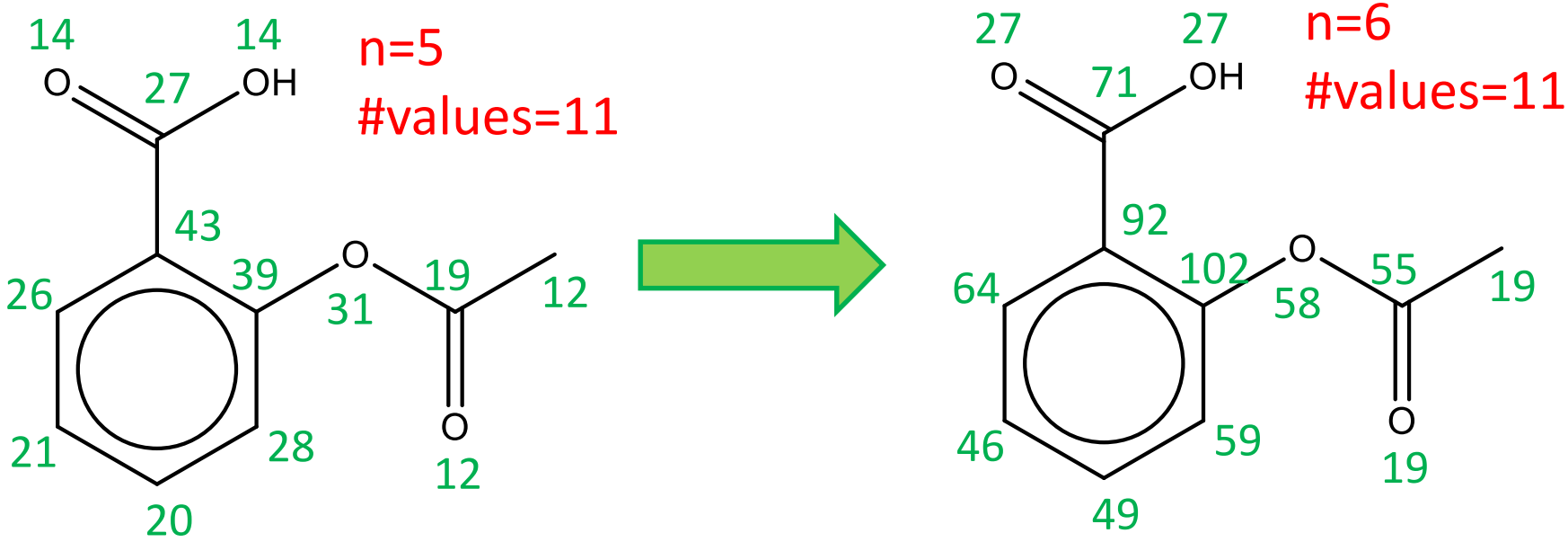
Morgan Algorithm (Simple)

- Repeat summing of neighboring values



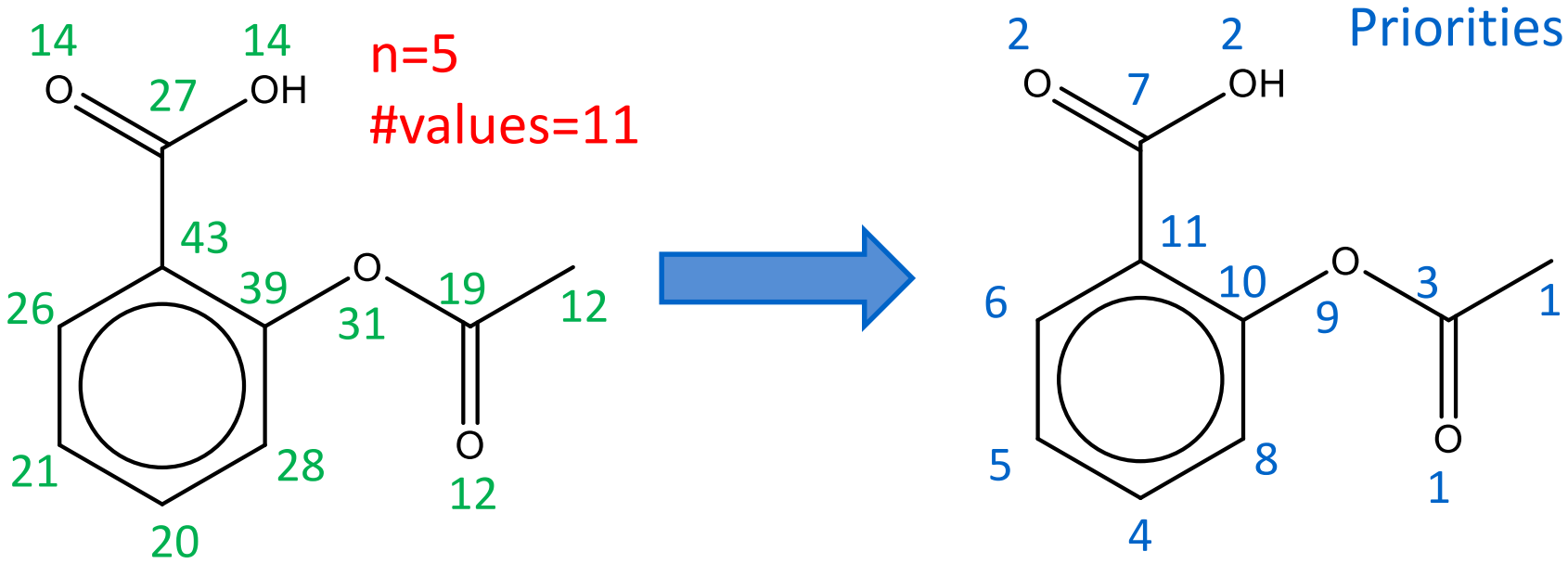
Morgan Algorithm (Simple)

- Repeat summing of neighboring values
- Until number of values does not increase anymore



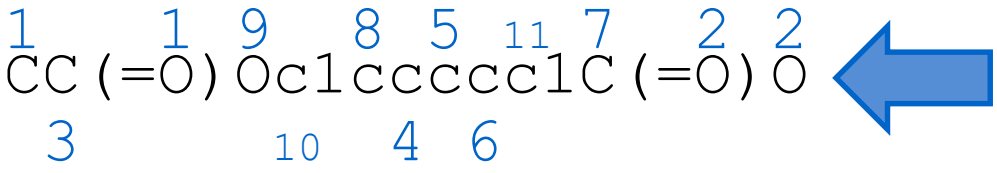
Morgan Algorithm (Simple)

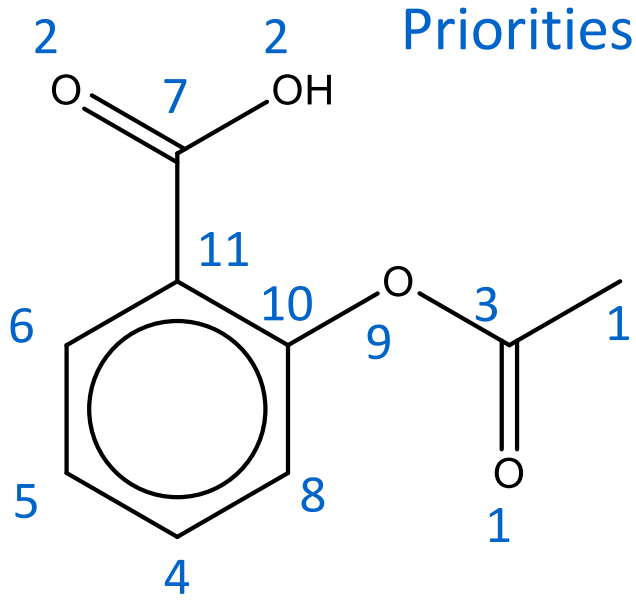
- Assign priorities according to invariants



Morgan Algorithm (Simple)

- Disambiguate ties by
  - atom type
  - bond order
- Construct Smiles according to invariants





Morgan Algorithm

- Invariants are very simple
- Not able to distinguish different atom types bond orders initially
- Numerical explosion
- Not all atoms can be distinguished sufficiently well…

Morgan Based Fingerprints…

- Morgan algorithm also contains the basic idea for some fingerprints
  - ECFP
  - Morgan fingerprints (RDKit)
- Initial invariants encode atom types, hydrogens, bonds
  - e.g. use Smarts patterns [CH2D1], [OH1D1]
- Each step combines the current invariant with those of the neighbors

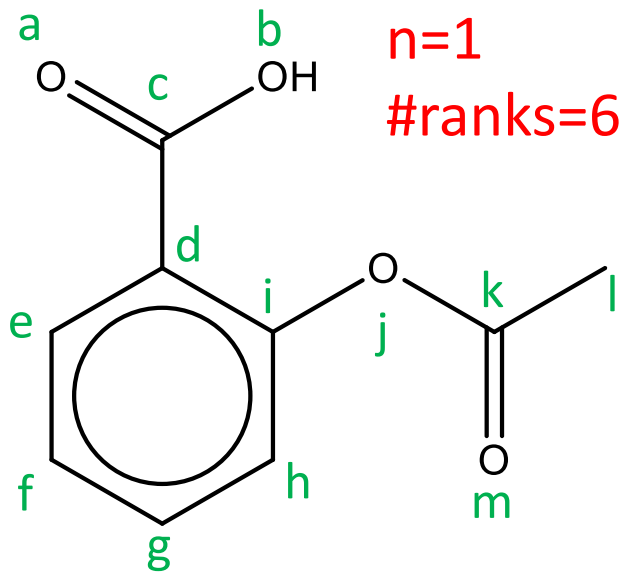
Cangen Algorithm

- Weininger et al.[1] proposed Cangen algorithm to address shortcomings:
  - Improved invariants
  - “Stable” prioritization
  - Avoids ambiguities from combining invariants
  - Resolves “symmetric” atoms

1. D Weininger D et al., JChemInfCompSci 29, 1989, 97--101

Cangen – Initial Invariants

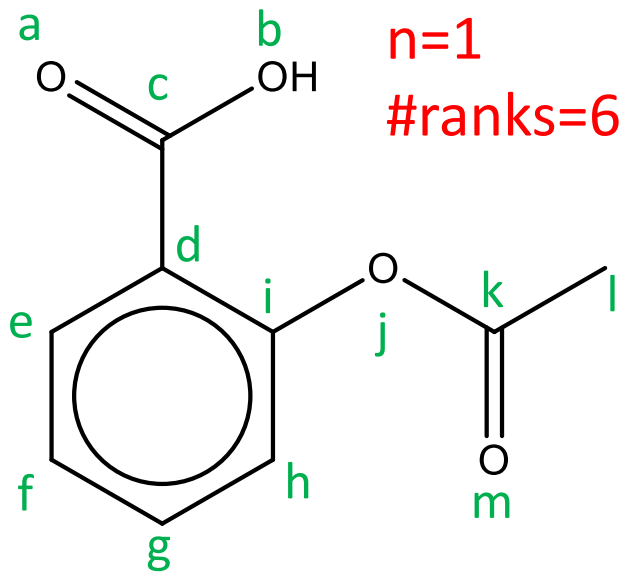
- Initial invariants encode atom type information
  - Number of neighbors
  - Sum of bond orders
  - Atom type
  - Charge
  - Number of attached hydrogens



Atom	#bds	Σ bds	At.Nr	Chg.	#H	
a	1	2	08	0	0	
b	1	1	08	0	1	
c	3	4	06	0	0	
d	3	4	06	0	0	
e	2	4	06	0	1	
f	2	4	06	0	1	
g	2	4	06	0	1	
h	2	4	06	0	1	
i	3	4	06	0	0	
j	2	2	06	0	0	
k	3	4	06	0	0	
l	1	1	06	0	3	
m	1	2	08	0	0	

Cangen – Initial Invariants

- Initial invariants are transformed to ranks



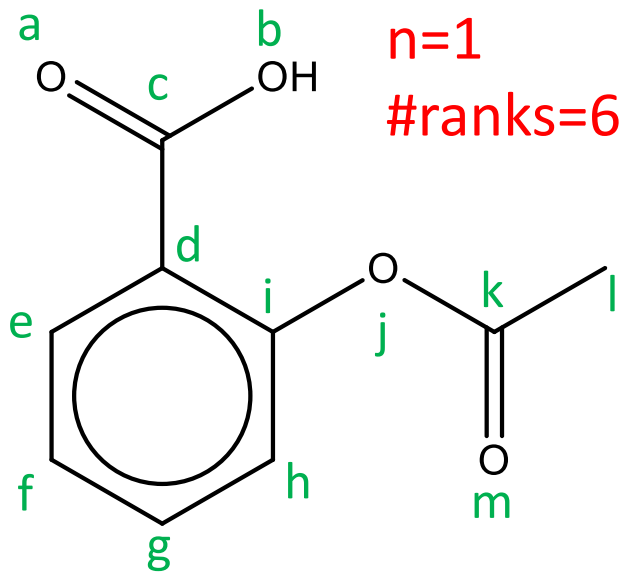
Atom	#bds	Σ bds	At.Nr	Chg.	#H	rank
a	1	2	08	0	0	**3**
b	1	1	08	0	1	**2**
c	3	4	06	0	0	**6**
d	3	4	06	0	0	**6**
e	2	3	06	0	1	**5**
f	2	3	06	0	1	**5**
g	2	3	06	0	1	**5**
h	2	3	06	0	1	**5**
i	3	4	06	0	0	**6**
j	2	2	06	0	0	**4**
k	3	4	06	0	0	**6**
l	1	1	06	0	3	**1**
m	1	2	08	0	0	**3**

Cangen – Update Rule for Invariants

- Rank is mapped to corresponding prime:

Rank	1	2	3	4	5	6	…
prime	2	3	5	7	11	13	…

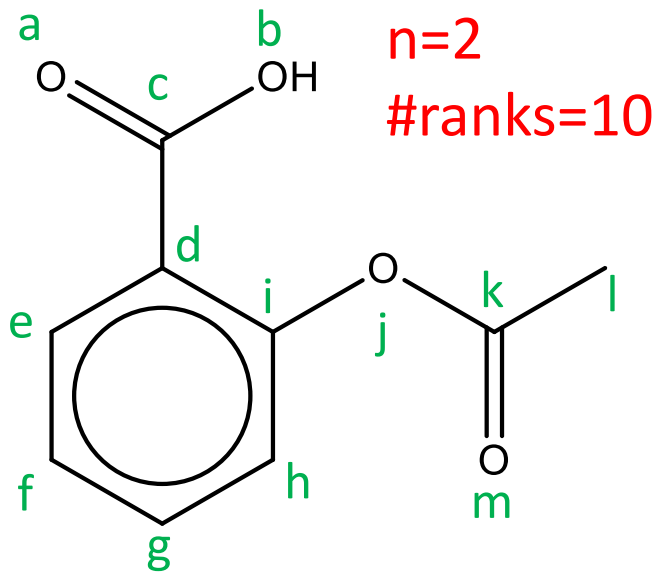
- New invariant:
  - primes of neighbors are multiplied



Atom	rank	prime	Nbors	New Inv.
a	3	5	c	13
b	2	3	c	13
c	6	13	a,b,d	195 = 5*3*13
d	6	13	c,e,i	1889 = 13*11*13
e	5	11	d,f	143 = 13*11
f	5	11	e,g	121 = 11*11
g	5	11	f,h	121 = 11*11
h	5	11	g,i	143 = 11*13
i	6	13	d,h,j	1001 = 13*11*7
j	4	7	i,k	169 = 13*13
k	6	13	j,l,m	70 = 7*2*5
l	1	2	k	13
m	3	5	k	13

Cangen – Update Rule for Invariants

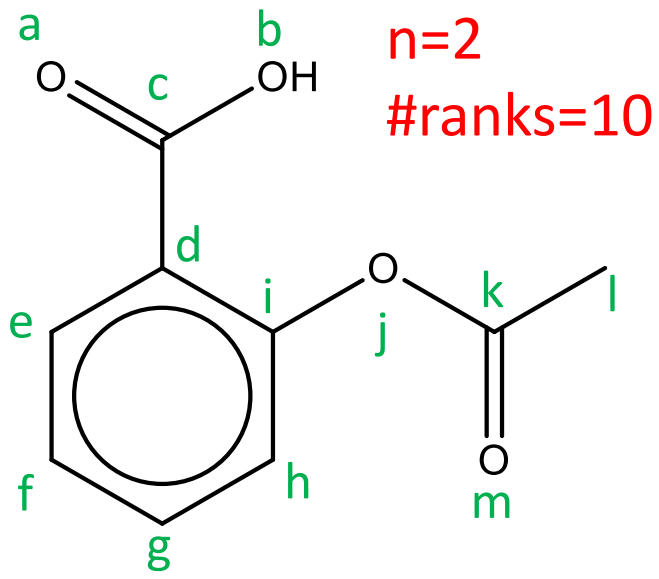
- New ranks are determined on the basis of:
  - Old ranks
  - New invariants



Atom	rank	New Inv.	(rk.,inv.)	New rank
a	3	13	(3,13)	**3**
b	2	13	(2,13)	**2**
c	6	195	(6,195)	**8**
d	6	1889	(6,1889)	**10**
e	5	143	(5,143)	**6**
f	5	121	(5,121)	**5**
g	5	121	(5,121)	**5**
h	5	143	(5,143)	**6**
i	6	1001	(6,1001)	**9**
j	4	169	(4,169)	**4**
k	6	70	(6,70)	**7**
l	1	13	(1,13)	**1**
m	3	13	(3,13)	**3**

Cangen – Iteration

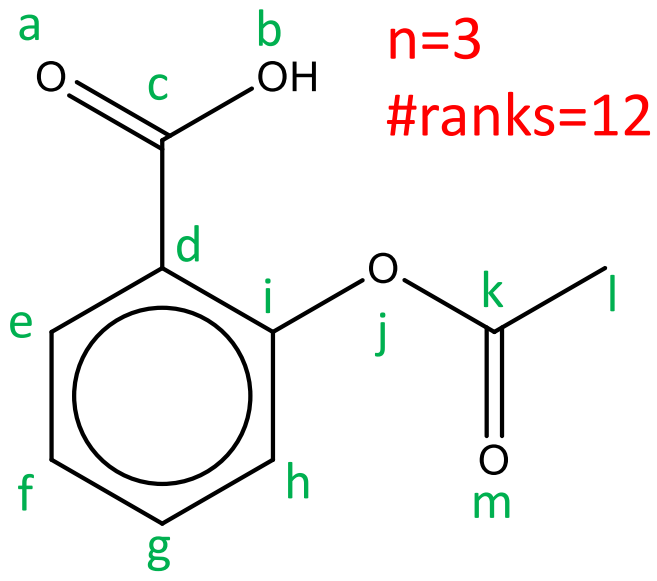
- Repeat until ranking is stable:
  - Calculate new invariants
  - Re-rank atoms



Atom	rank	prime	Nbors	New Inv.
a	**3**	5	c	19
b	**2**	3	c	-
c	**8**	19	a,b,d	-
d	**10**	29	c,e,i	-
e	**6**	13	d,f	319 = 29*11
f	**5**	11	e,g	143 = 13*11
g	**5**	11	f,h	143 = 11*13
h	**6**	13	g,i	243 = 11*23
i	**9**	23	d,h,j	-
j	**4**	7	i,k	-
k	**7**	17	j,l,m	-
l	**1**	2	k	-
m	**3**	5	k	17

Cangen – Iteration

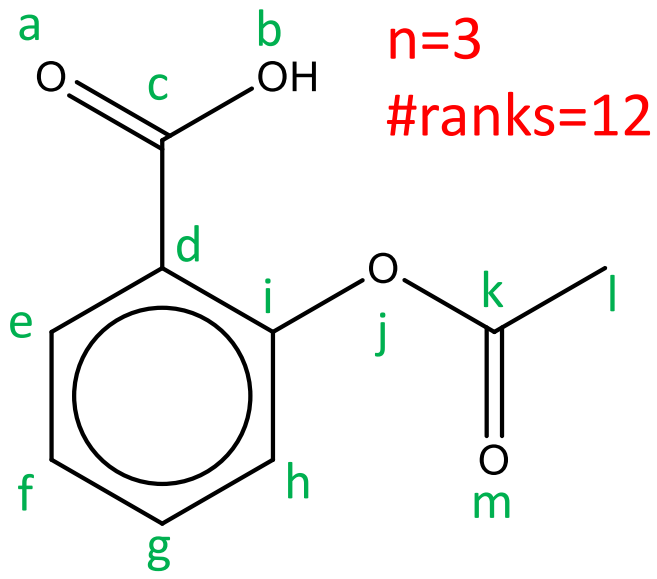
- Repeat until ranking is stable:
  - Calculate new invariants
  - Re-rank atoms



Atom	rank	New Inv.	(rk.,inv.)	New rank
a	3	19	(3,19)	4
b	2	-	(2,-)	2
c	8	-	(8,-)	10
d	10	-	(10,-)	12
e	6	319	(6,319)	8
f	5	143	(5,143)	6
g	5	143	(5,143)	6
h	6	243	(6,243)	7
i	9	-	(9,-)	11
j	4	-	(4,-)	5
k	7		(7,-)	9
l	1	-	(1,-)	1
m	3	17	(3,17)	3

Cangen – Iteration

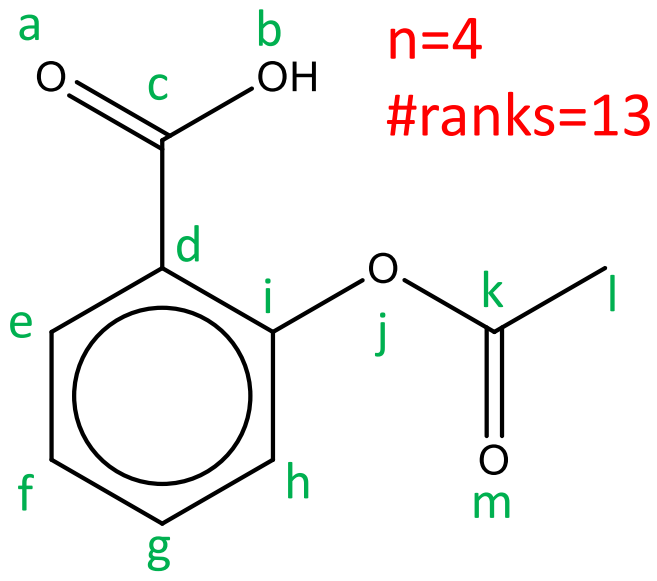
- Repeat until ranking is stable:
  - Calculate new invariants
  - Re-rank atoms



Atom	rank	prime	Nbors	New Inv.
a	**4**	7	c	-
b	**2**	3	c	-
c	**10**	29	a,b,d	-
d	**12**	37	c,e,i	-
e	**8**	19	d,f	-
f	**6**	13	e,g	249 = 19*13
g	**6**	13	f,h	221 = 13*17
h	**7**	17	g,i	-
i	**11**	31	d,h,j	-
j	**5**	11	i,k	-
k	**9**	23	j,l,m	
l	**1**	2	k	-
m	**3**	5	k	-

Cangen – Final Ranking

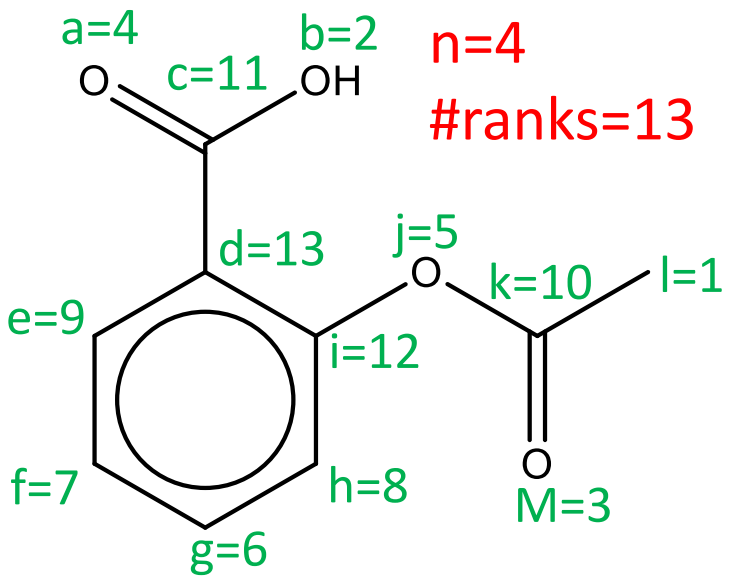
- Repeat until ranking is stable:
  - Calculate new invariants
  - Re-rank atoms
- Final ranking



Atom	rank	New Inv.	(rk.,inv.)	New rank
a	4	-	(4,-)	4
b	2	-	(2,-)	2
c	10	-	(10,-)	11
d	12	-	(12,-)	13
e	8	-	(8,-)	9
f	6	249	(6,249)	7
g	6	221	(6,221)	6
h	7	-	(7,-)	8
i	11	-	(11,-)	12
j	5	-	(5,-)	5
k	9		(9,-)	10
l	1	-	(1,-)	1
m	3	-	(3,-)	3

Cangen – Final Ranking

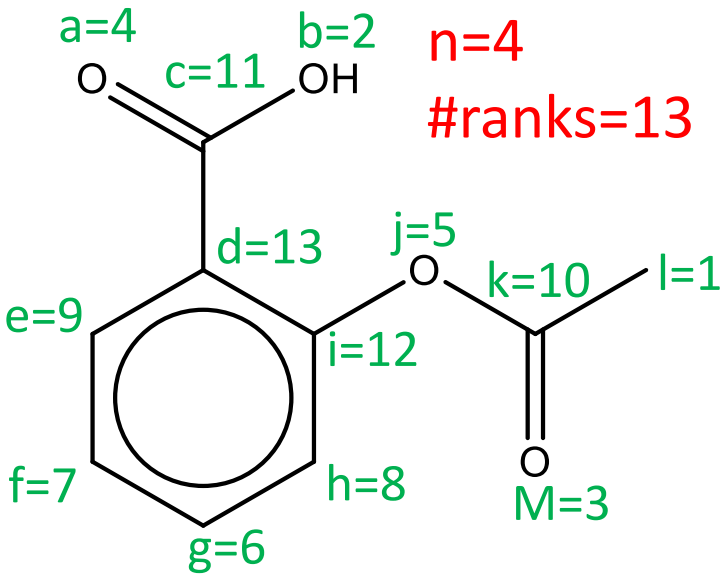
- Repeat until ranking is stable:
  - Calculate new invariants
  - Re-rank atoms
- Final ranking yields priorities

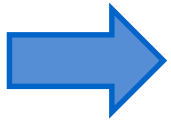


Atom	rank	New Inv.	(rk.,inv.)	New rank
a	4	-	(4,-)	4
b	2	-	(2,-)	2
c	10	-	(10,-)	11
d	12	-	(12,-)	13
e	8	-	(8,-)	9
f	6	249	(6,249)	7
g	6	221	(6,221)	6
h	7	-	(7,-)	8
i	11	-	(11,-)	12
j	5	-	(5,-)	5
k	9		(9,-)	10
l	1	-	(1,-)	1
m	3	-	(3,-)	3

Cangen – Generate Smiles

- Repeat until ranking is stable:
  - Calculate new invariants
  - Re-rank atoms
- Final ranking yields priorities
- Generate Smiles

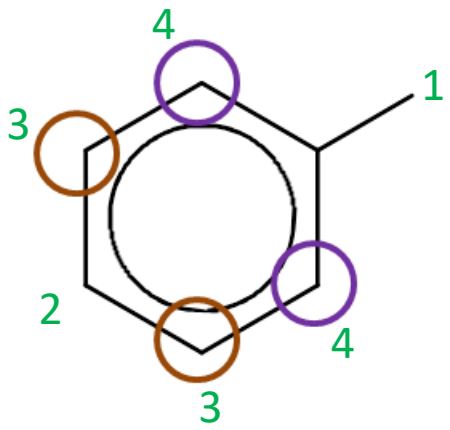




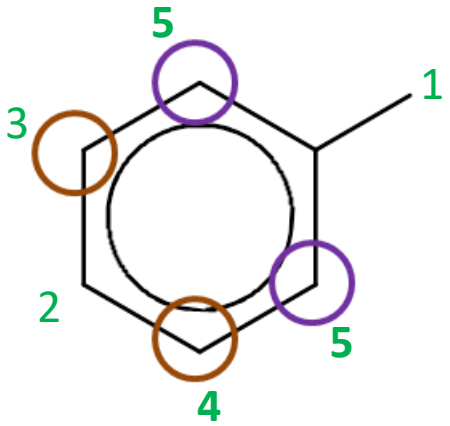
```
lk  m ji hgfed c a  b
CC(=O)Oc1ccccc1C(O)=O
```

Cangen – Ties

- In case of symmetric atoms not all atoms can be uniquely prioritized this way:



- In this case a single tie is broken to be able to continue the algorithm



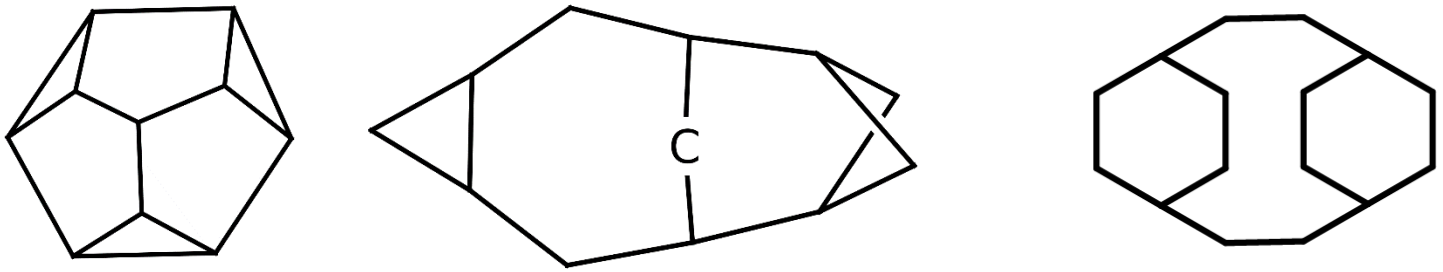
- If necessary, repeat breaking ties until all ambiguities are reolved

Remarks: Canonical Representations

- Limitations
  - Modifications are needed to handle stereochemistry
  - Different tautomers of a compound yield different Smiles
  - Canonicalization also depends on the aromaticity model
  - Different variations of the algorithms exist:
    - Canonical Smiles are only canonical with respect to a specific toolkit!
- InChI also uses a canonical atom labeling algorithm
  - Its algorithm is based on a mathematical group-theoretical formulation
  - The basic idea of iterative refinement of rankings by finer partitioning of the atoms into different ranks is similar
  - More complex
  - More robust
  - Standardized

Remarks: Canonical Representations

- Will CANGEN always provide unique representation?
  - The method works as long as the algorithm is able to distinguish between non-symmetric molecules.
  - The algorithm could fail if for each atom in a molecule there is another non-symmetric atom with the same local neighborhood.
  - This can happen



  - …although these molecules may be of little practical value

Images from: Carhart RE. JChemInfCompSci,1978,18,108--110

Remarks

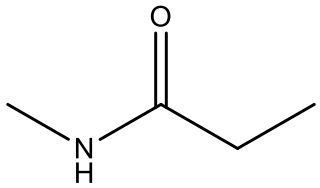
- Canonicalization of molecular representations allows easy identification of identical molecules
- It solves the “graph isomorphism” problem
  - It is believed no efficient (polynomial-time) algorithm exists for solving this problem
  - Thus any algorithm for this problem
    - either is inefficient at least in some instances
    - or fails to yield a unique canonical representation in some instances
- Cangen algorithm is a practical, efficient algorithm for generating canonical Smiles but it can fail for some obscure molecules
- InChI algorithm is practical, correct, and efficient (all but in a theoretical sense)

Substructure Searching

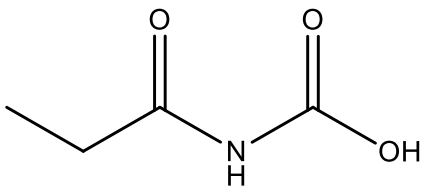
- Finding canonical identifiers effectively solves the graph isomorphism problem
- The corresponding problem to substructure searching is the subgraph isomorphism problem
- The subgraph isomorphism problem is believed to be harder than graph isomorphism
  - (The former is NP-complete whereas the latter is not)
- Algorithms rely on (but in a smart way) exploring the search space

Substructure Searching: Basic Idea

Given: substructure S of k atoms and molecule of n atoms



S, k=6



M, n=8

Find mapping between all atoms of S and some atoms of M so that:

if there is a bond between two atoms in S there has to be a bond between the two corresponding atoms in M

Problem: the number of possible mappings can be huge

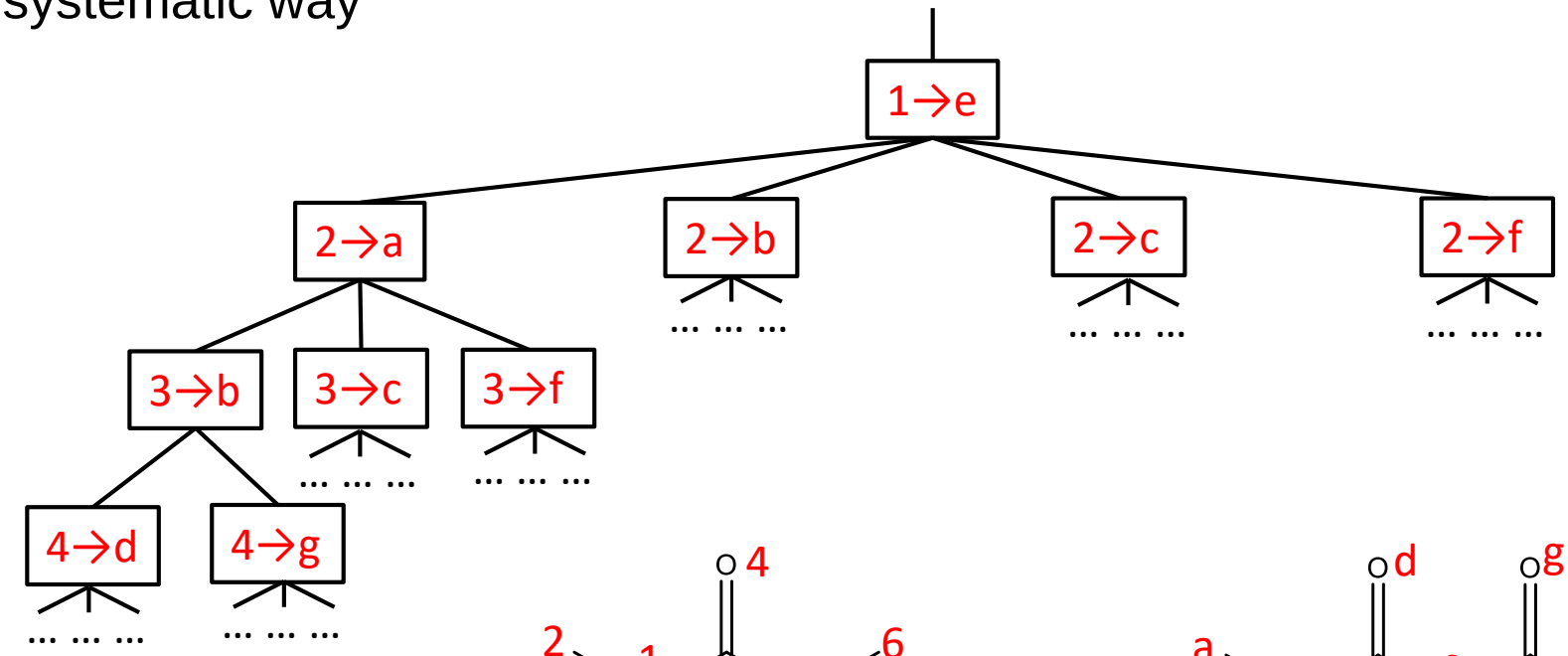
- e.g. n=30, k=7 there are ca. 10 billion ($10 \cdot 10^9$) possible mappings

Naively exploring the search space is too time expansive

- Standard algorithmic solution: Backtracking

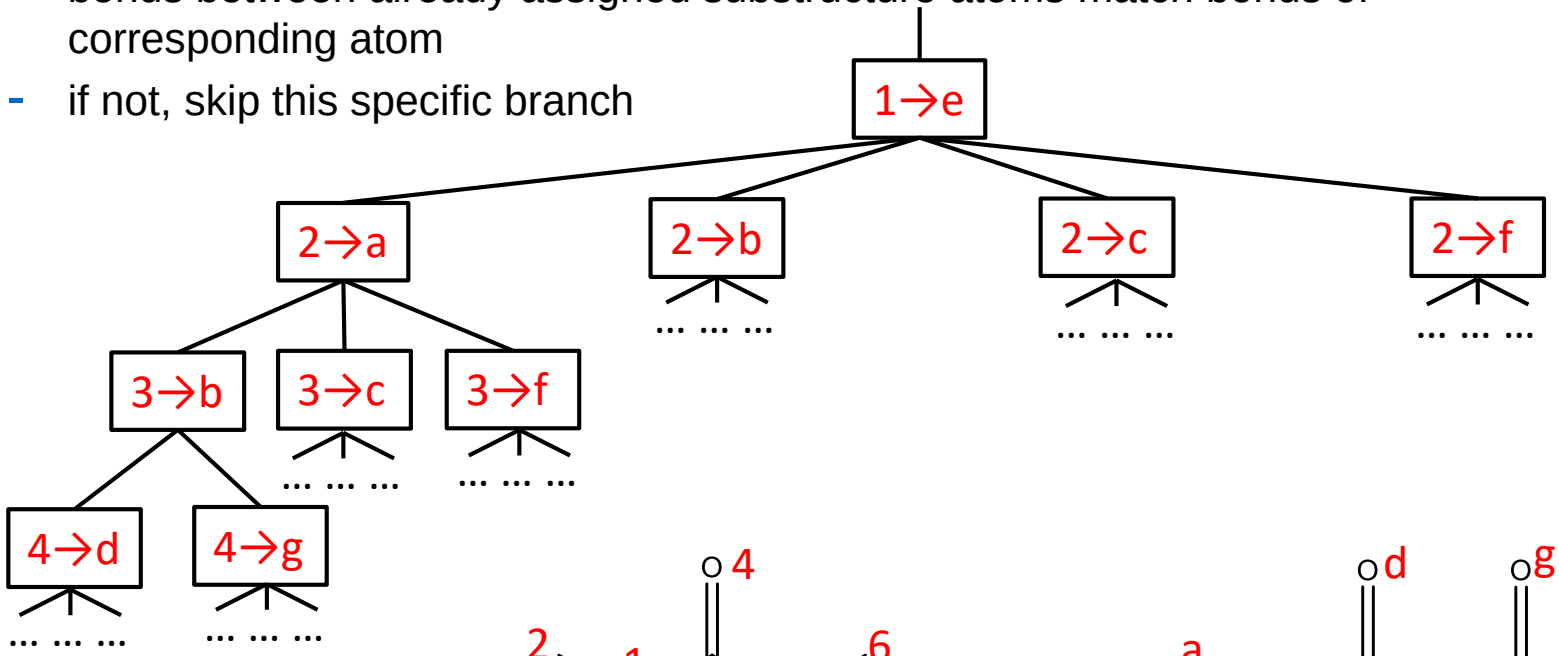
Backtracking

- Backtracking is a search strategy that traverses a search tree in depth-first order
- A search tree assigns atoms of S to all possible atoms of M in a systematic way



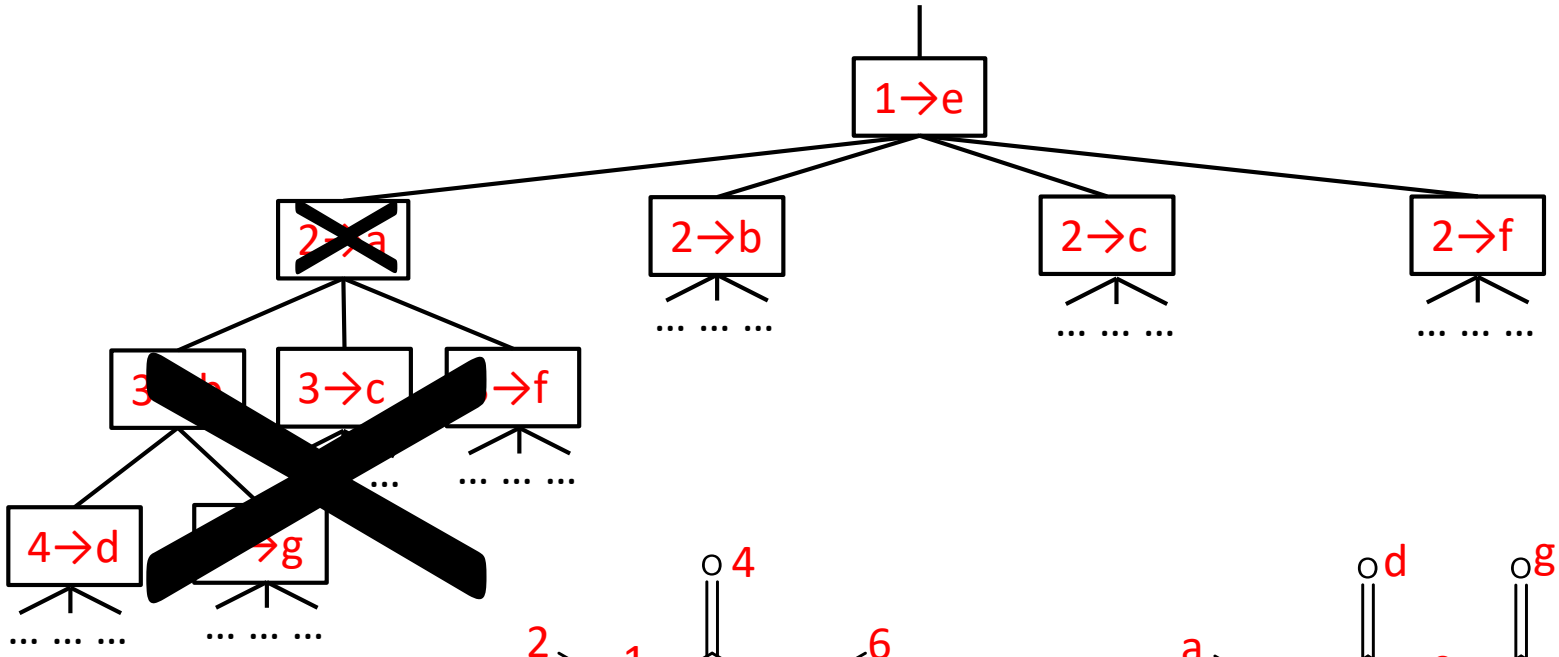
Backtracking

- At each step of traversing the tree check whether the assignment is plausible, i.e.
  - atom types match and
  - bonds between already assigned substructure atoms match bonds of corresponding atom
  - if not, skip this specific branch



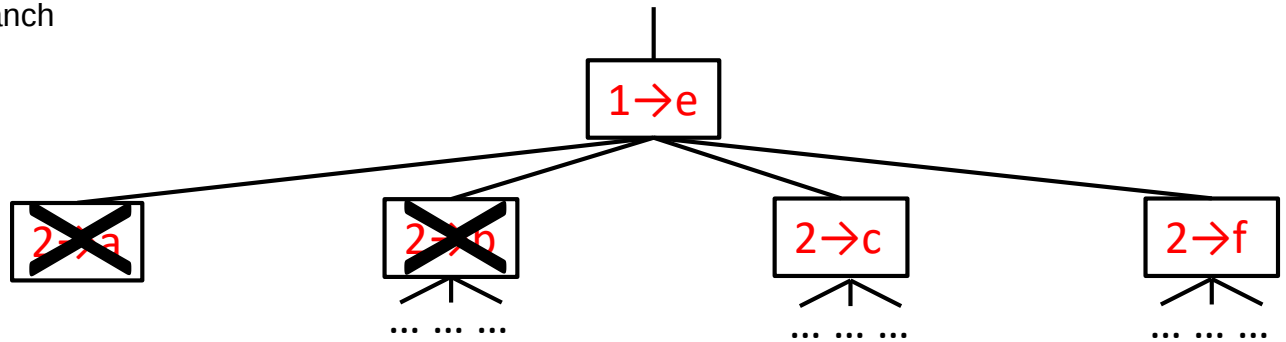
Backtracking

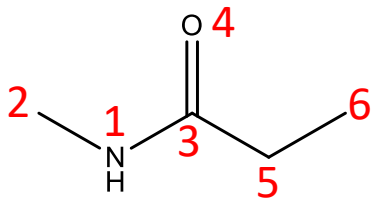
- Assign: 1→e
  - Assign: 2 →a
    - Assignment is not possible
    - Skip branch



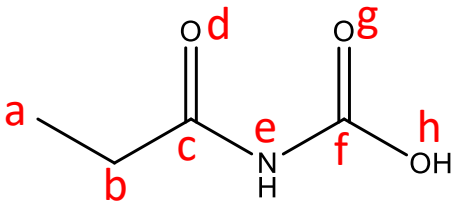
Backtracking

- Assign: 1→e
  - …
  - Assign: 2 →b
    - Assignment is not possible
    - Skip branch





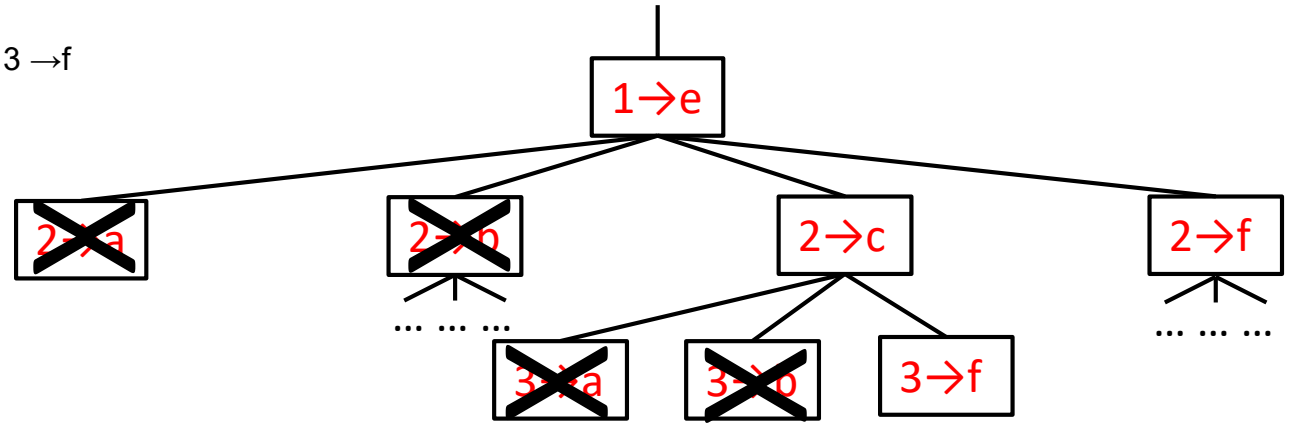
S, k=6

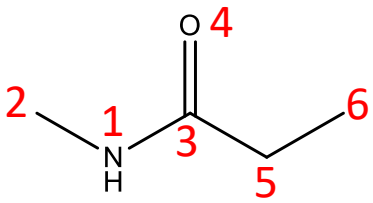


M, n=8

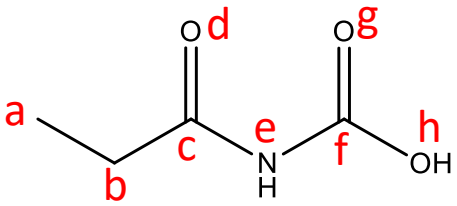
Backtracking

- Assign: 1→e
  - …
  - Assign: 2 →c
    - Assignment possible
    - …
    - Assign: 3 →f





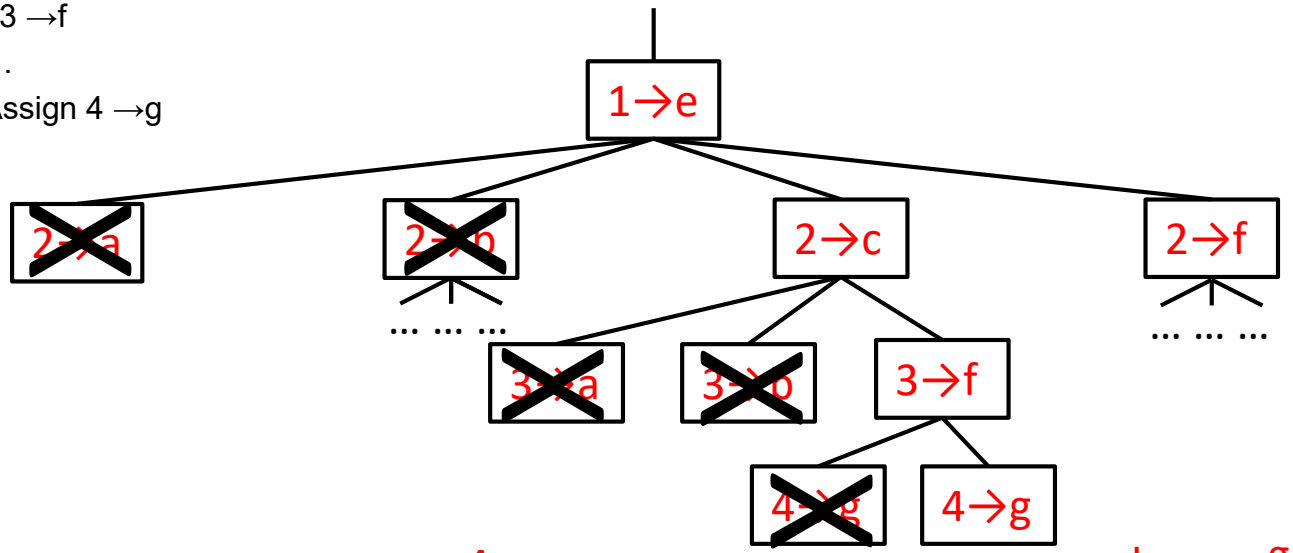
S, k=6

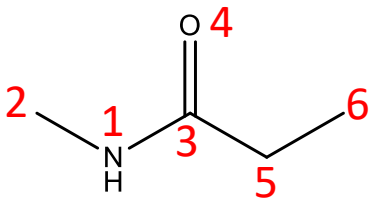


M, n=8

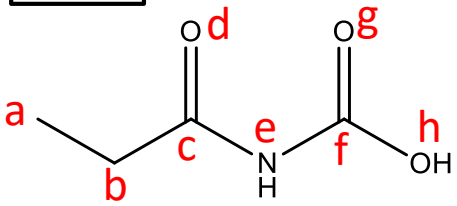
Backtracking

- Assign: 1→e
  - …
  - Assign: 2 →c
    - …
    - Assign: 3 →f
      - …
      - Assign 4 →g





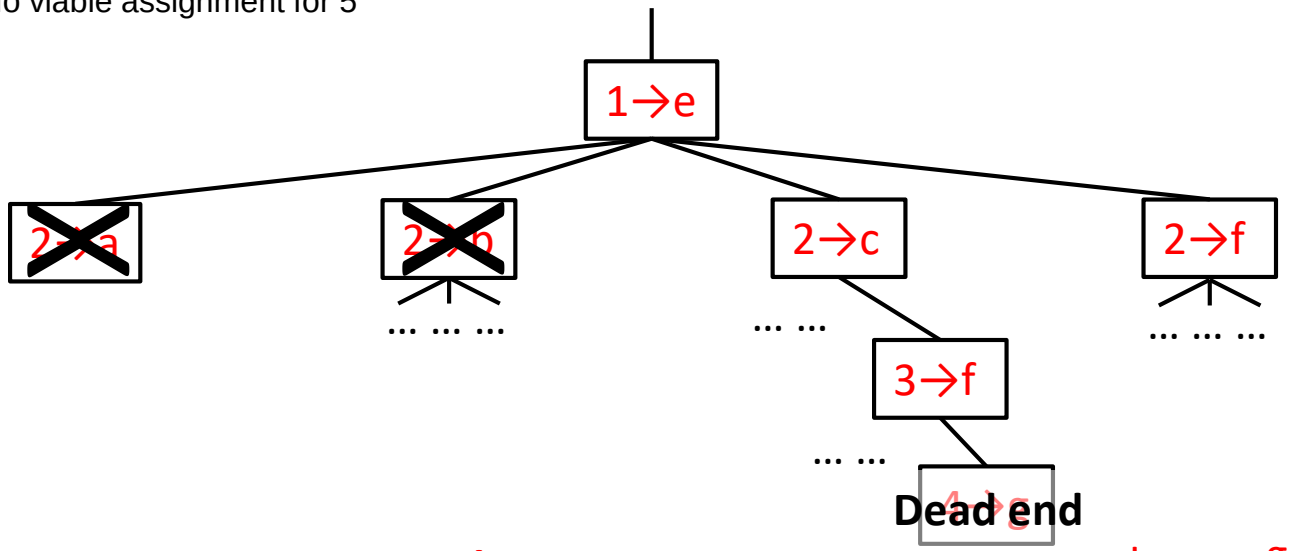
S, k=6



M, n=8

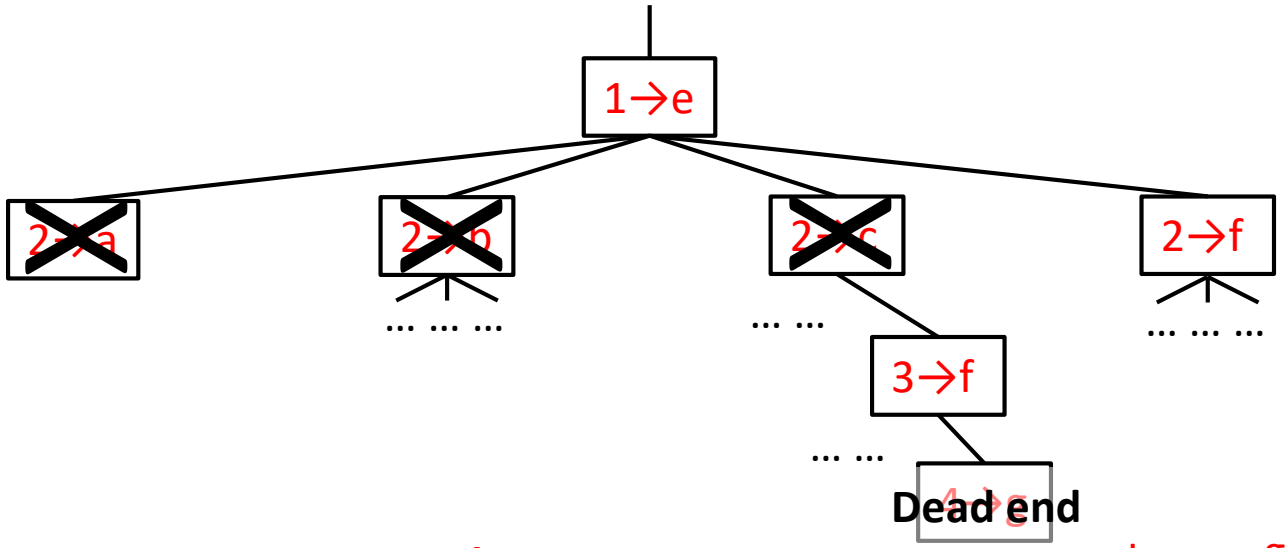
Backtracking

- Assign: 1→e
  - Assign: 2 →c
    - Assign: 3 →f
      - Assign 4 →g
      - No viable assignment for 5



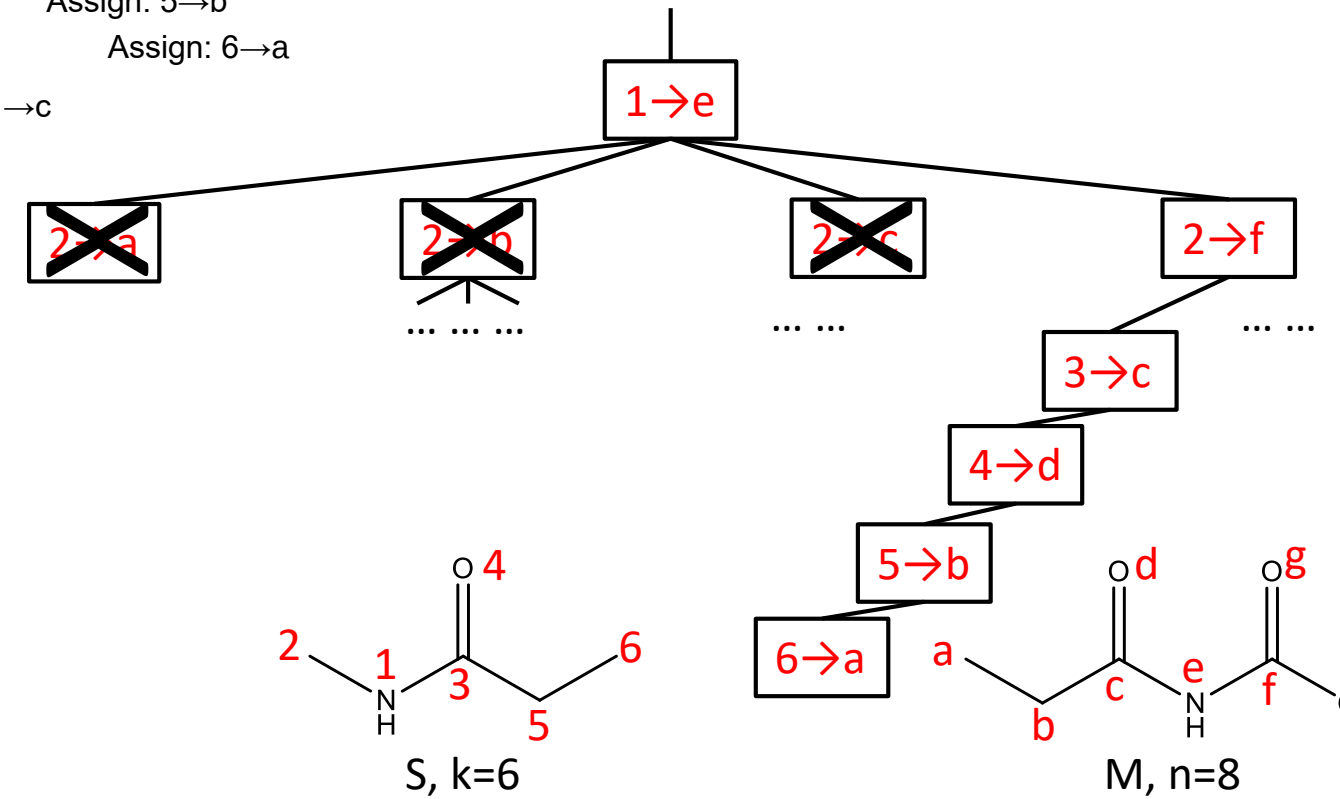
Backtracking

- Assign: 1→e
  - …
  - Assign: 2 →f



Backtracking

- Assign: 1→e
  - Assign: 2 →f
    - Assign: 3 →c
      - Assign: 4 →d
        - Assign: 5→b
          - Assign: 6→a
  - Assign: 3 →c



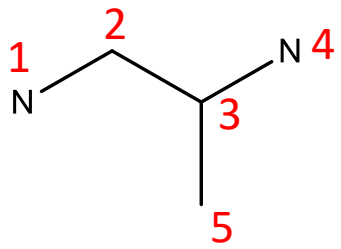
Ullmann Algorithm

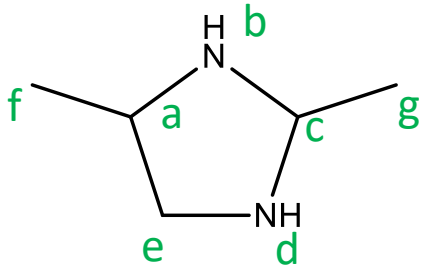
- For backtracking:
  - The earlier search tree traversal fails, the better
  - The order in which substructure atoms are assigned is arbitrary
  - However, starting with rare hetero-atoms might be good for efficiency because only few options are usually available for these
- Refined backtracking: Ullmann algorithm
  - Backtracking only checks if currently mapped atom yields plausible assignment
  - Idea:
    - In each step, keep track of all possible assignments for all substructure atoms
    - If no possible matches remain for any single substructure atom the branch can be skipped
    - Note, that the basic backtracking only check possibilities of the current atom to be assigned and not any later ones

Ullmann Algorithm: Feasibility Matrix

- Keep track of all possible matches for substructure atoms in a feasibility matrix
- Initially:
  - atom types must match
  - the corresponding atom in the molecule must have at least as many neighbors as the substructure atom
    - e.g., 3 cannot be assigned to e

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





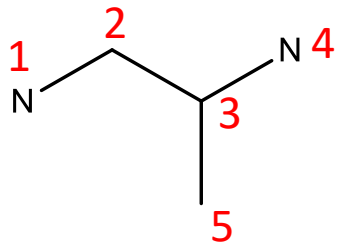
Ullmann Algorithm: Refinement

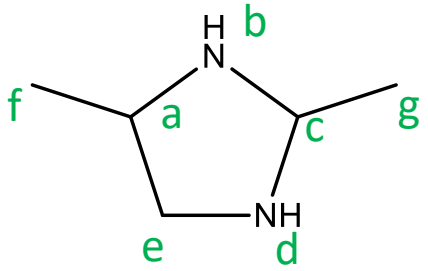
- Next, the matrix is refined by checking the plausibility of each entry:

> if atom s should be assigned to atom m it should also be possible to assign neighbors of s to neighbors of m

- Thus, for M(s,m)=x check for all neighbors n of s if M(n,k)=x where k is neighbor of m. If no neighbor m can be found eliminate possibility M(s,k)
- Continue checking entries until no more possibilities can be removed
- After that perform standard backtracking, in each step updating and refining M

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

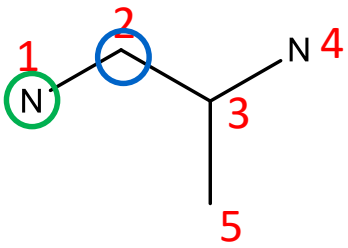


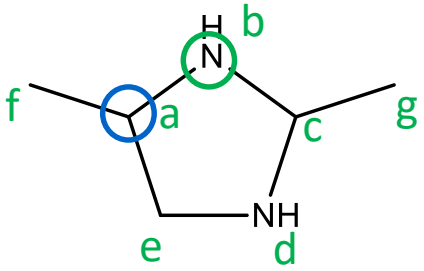


Ullmann Algorithm: Refinement

- The entries for atom 1 pass the test

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

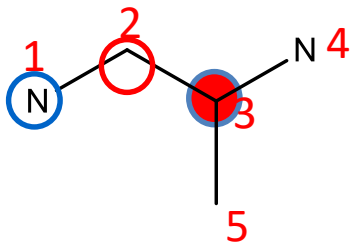


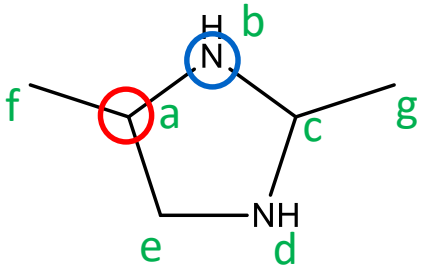


Ullmann Algorithm: Refinement

- The entries for atom 1 pass the test
- The possibility 2→a fails because although
  - 1→b is a possibility for one neighbor of 2
  - there is no possibility to map 3 to a neighbor of a

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

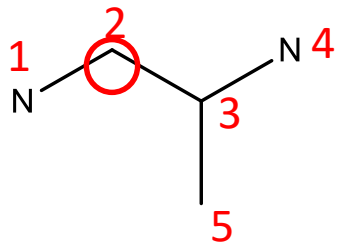


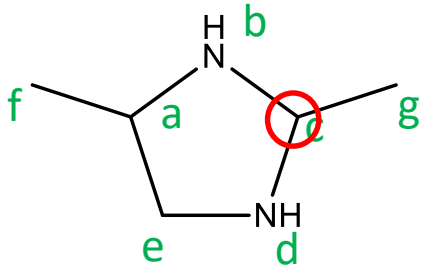


Ullmann Algorithm: Refinement

- The entries for atom 1 pass the test
- The possibility 2→a fails because although
  - 1→b is a possibility for one neighbor of 2
  - there is no possibility to map 3 to a neighbor of a
- The possibility 2→c fails for the same reason

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

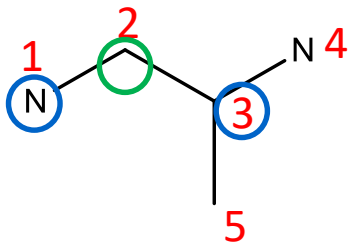


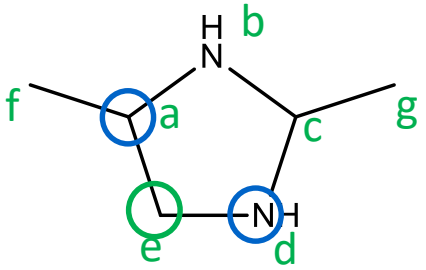


Ullmann Algorithm: Refinement

- The entries for atom 1 pass the test
- The possibility 2→a fails because although
  - 1→b is a possibility for one neighbor of 2
  - there is no possibility to map 3 to a neighbor of a
- The possibility 2→c fails for the same reason
- The possibility 2→e remains valid because
  - 1→d and
  - 3→a are possibilities for the neighbors

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

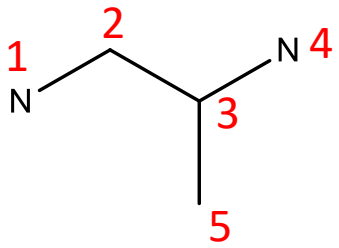


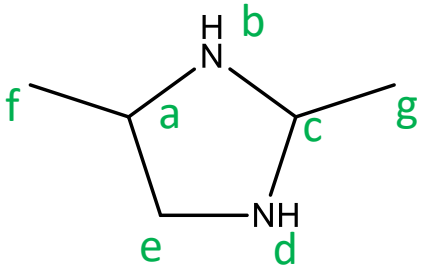


Ullmann Algorithm: Refinement

- Completing the matrix eliminates further possibilities

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

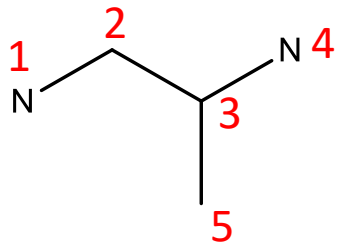


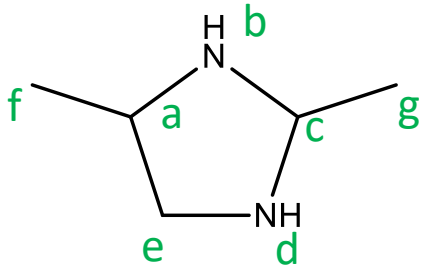


Ullmann Algorithm: Refinement

- Checking the matrix a second time eliminates another possibility

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

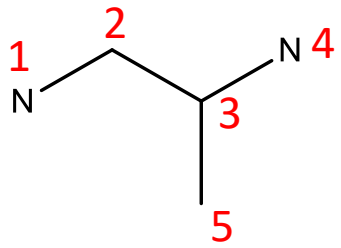


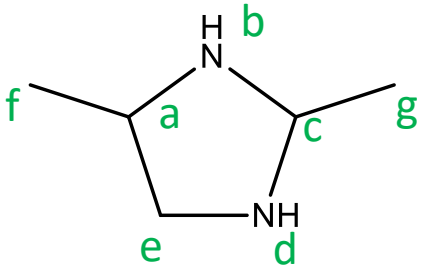


Ullmann Algorithm: Refinement

- Matrix after refinement
- A single assignment remains possible

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





Ullmann Algorithm: Summary

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

Ullmann Algorithm: Summary

- Ullmann is a backtracking algorithm
- Compared to “standard backtracking” considerable work is done in each step to check feasibility
- Each step requires much more computation time than the standard approach
- However, much better “pruning” of search tree makes it much more efficient

References

- Vogt M, de la Vega de León A & Bajorath J.
  Algorithmic Chemoinformatics
  in:
  Varnek A (Ed.)
  Tutorials in Chemoinformatics, 395-448
  John Wiley & Sons Ltd, Chichester, UK, 2017