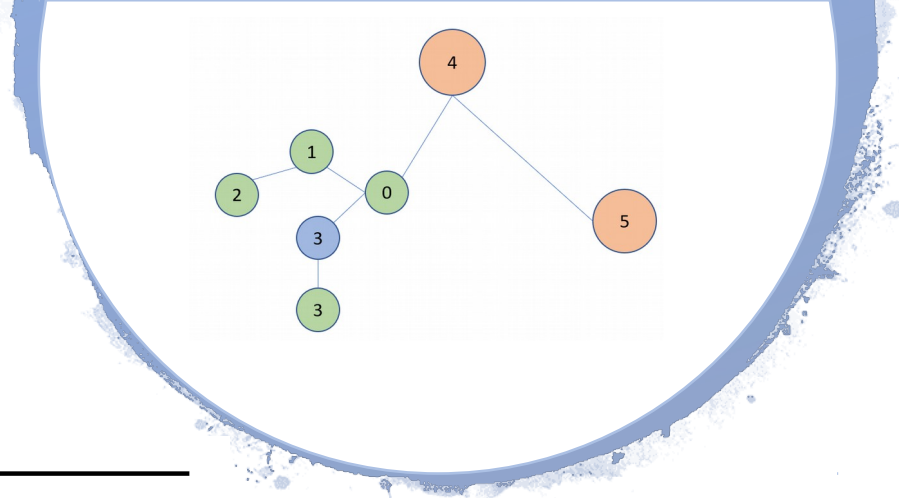
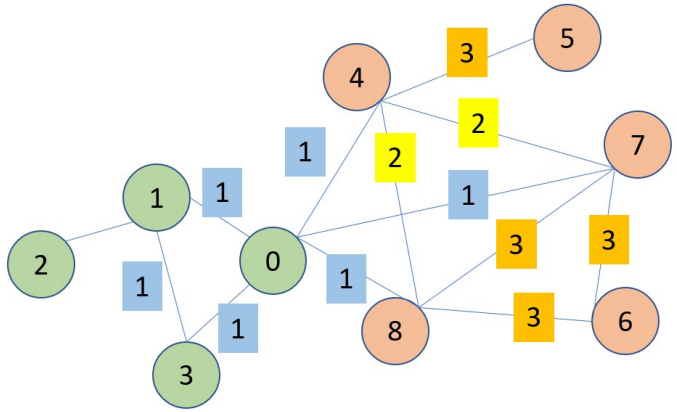


Junction Tree Variational Autoencoder for Molecular Graph Generation

Wengong Jin [1] Regina Barzilay [1] Tommi Jaakkola [1]

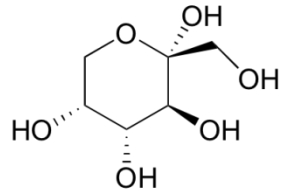
arXiv:1802.04364v2

Presented by: Rouzbeh Afrasiabi

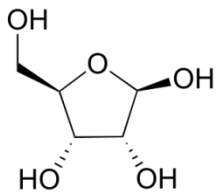


Organic chemistry

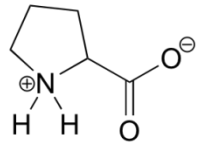
"Organic chemistry is the study of the structure, properties, composition, reactions, and preparation of carbon-containing compounds, which include not only hydrocarbons but also compounds with any number of other elements, including **hydrogen** (most compounds contain at least one carbon–hydrogen bond), **nitrogen**, **oxygen**, **halogens**, **phosphorus**, **silicon**, and **sulfur**. This branch of chemistry was originally limited to compounds produced by living organisms but has been broadened to include human-made substances such as plastics. " ACS



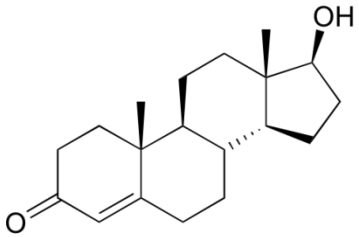
fructose



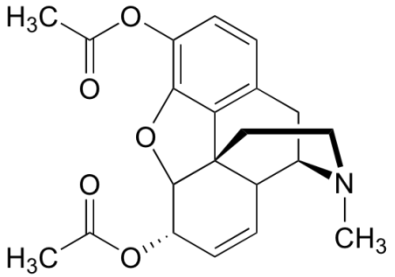
ribose



proline



testosterone



heroin

Why in silico generation of molecules?

"The purpose of a drug discovery and development project is to find new therapeutic agents that target a **key enzyme, protein-protein interaction, receptor-ligand, or protein-nucleic acid interaction** of relevance in a disease of interest in order to mitigate the course of the disease.

Finding compounds that can, for example, bind to the active site of an enzyme and inhibit its normal activity is merely the initial goal in most drug discovery projects. Typically, **tens of thousands or even hundreds of thousands** of different compounds must be screened in order to find a few promising compounds.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5765859/

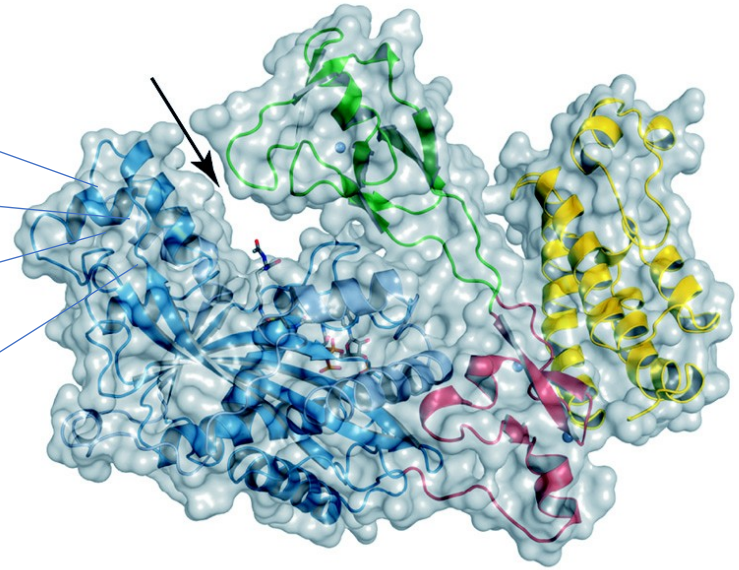
Experimentally screening large numbers of compounds to find
costly and **time consuming** process and is just the first step ir

" **Biological molecules have complex interactions and structure**

Dynamic structure

Can be specific or selective when binding

Not fully understood



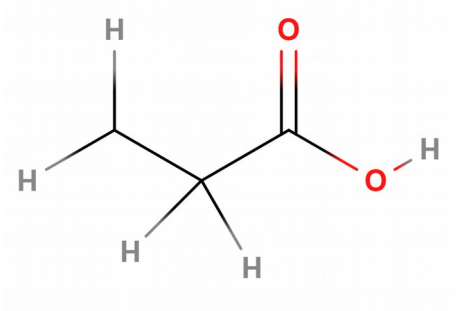
https://www.technology.org/2013/10/14/secret-life-cancer-related-protein-revealed-3d-structure/

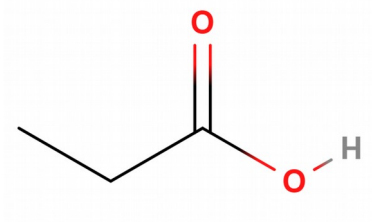
Methods of drawing/displaying organic molecules

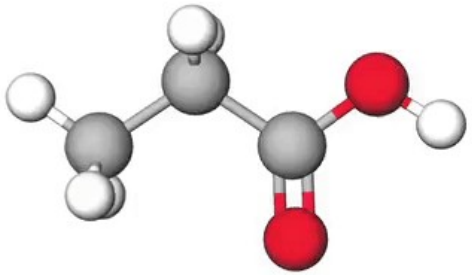
IUPAC **Molecular Formulae** **Displayed Formulae** **Skeletal Formulae** **3D structure**

propanoic acid
1-propanoic acid
2-methylacetic acid

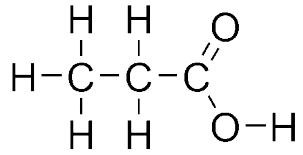
$C_3H_6O_2$





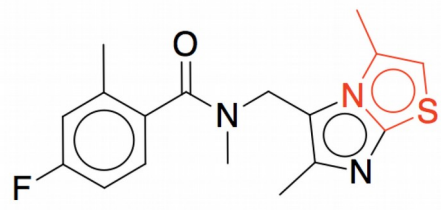


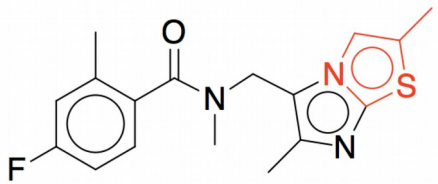
International **U**nion of **P**ure and **A**pplied **C**hemistry



SM😄S: Simplified Molecular-input line-entry system

CCC(=O)O





😭sadvantage of SMILES

Cc1cn2c(CN(C)C(=O)c3ccc(F)cc3C)c(C)nc2s1

Cc1cc(F)ccc1C(=O)N(C)Cc1c(C)nc2scc(C)n12

Identical molecules with different SMILES

Why identical molecules having different SMILES is a disadvantage?

Method	Reconstruction	Validity
CVAE	44.6%	0.7%
GVAE	53.7%	7.2%
SD-VAE[2]	76.2%	43.5%
GraphVAE	-	13.5%

First, the SMILES representation is **not designed to capture molecular similarity**. This prevents generative models like variational autoencoders from learning smooth molecular embeddings.

Second, essential chemical properties such as **molecule validity** are **easier to express on graphs** rather than linear SMILES representations.

What is the solution?

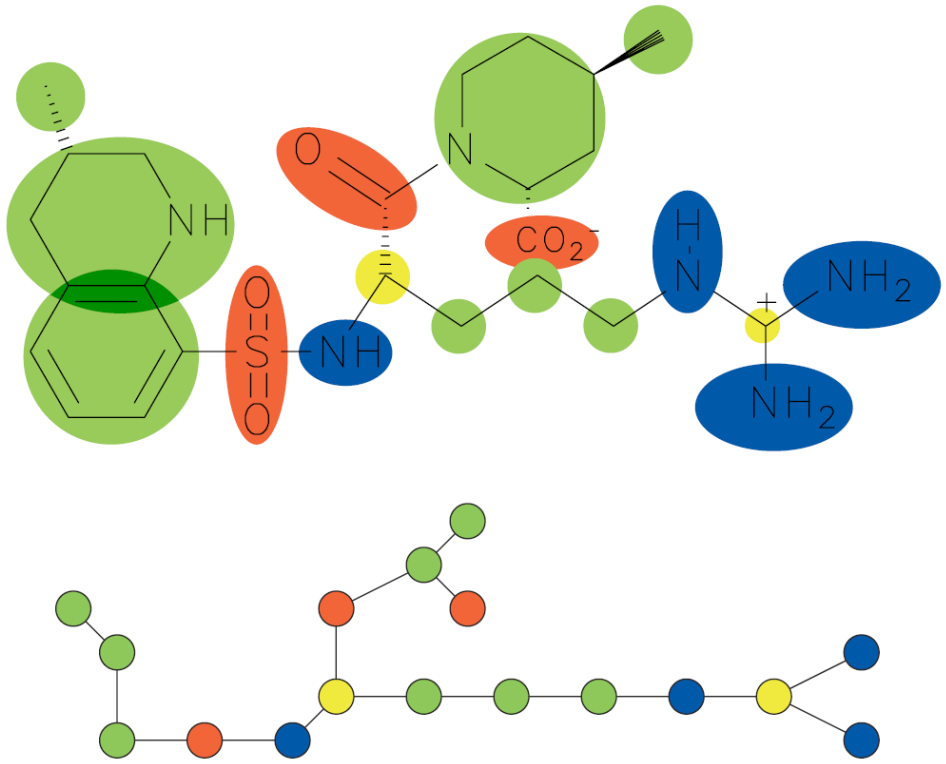
Representing molecules in a tree structured format??!!

But organic molecules contain cyclic structures!!

Feature trees: A new molecular similarity measure based on tree matching

Matthias Rarey[a,*] & J. Scott Dixon[b,**]

A feature tree represents hydrophobic fragments and functional groups of the molecule and the way these groups are linked together. Each node in the tree is labeled with a set of features representing chemical properties of the part of the molecule corresponding to the node. The comparison of feature trees is based on matching subtrees of two feature trees onto each other.

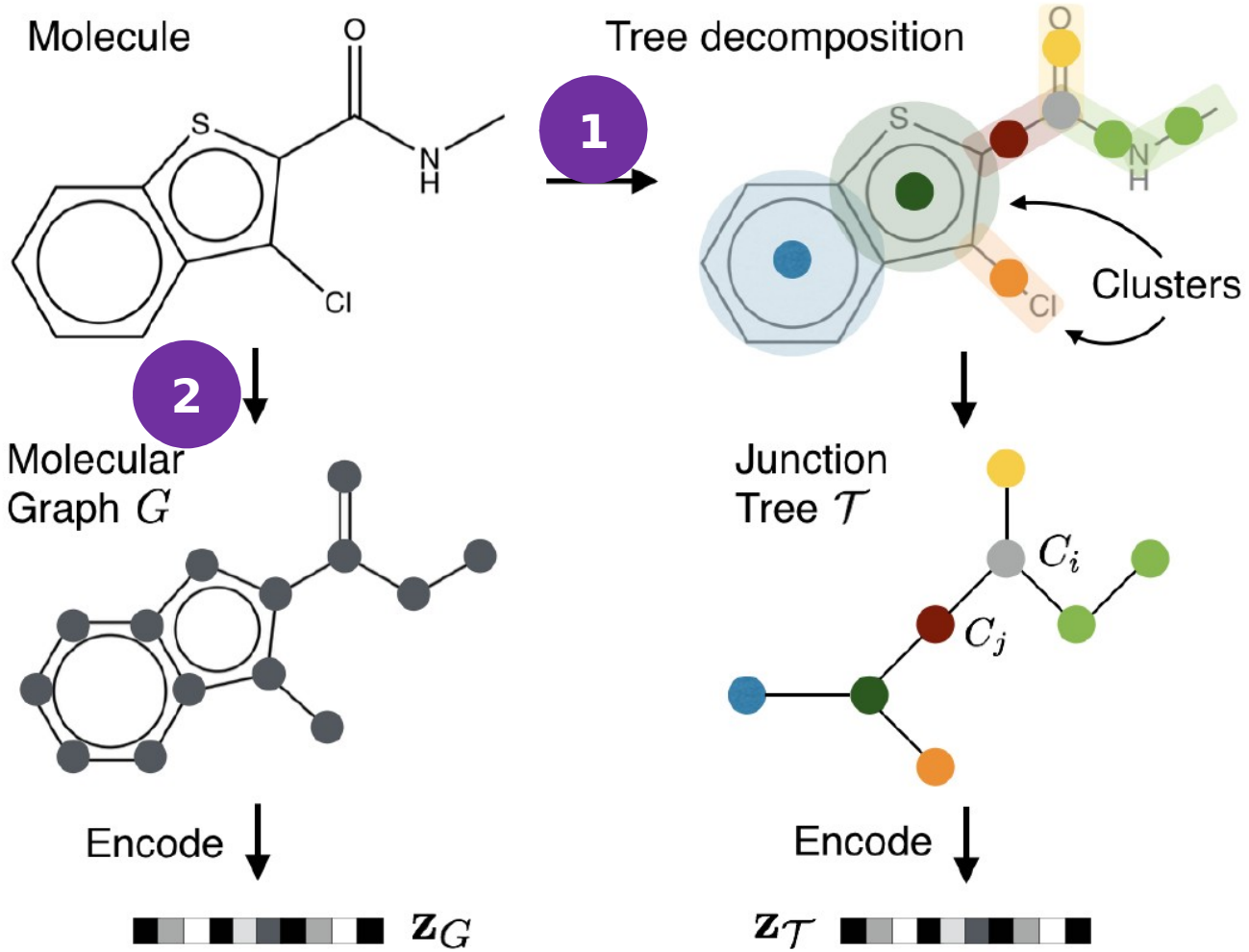


Journal of Computer-Aided Molecular Design, **12:** 471–490, 1998.

w about we use feature trees to train a VAE?

Good Idea!! Jin , Barzilay, Jaakkola

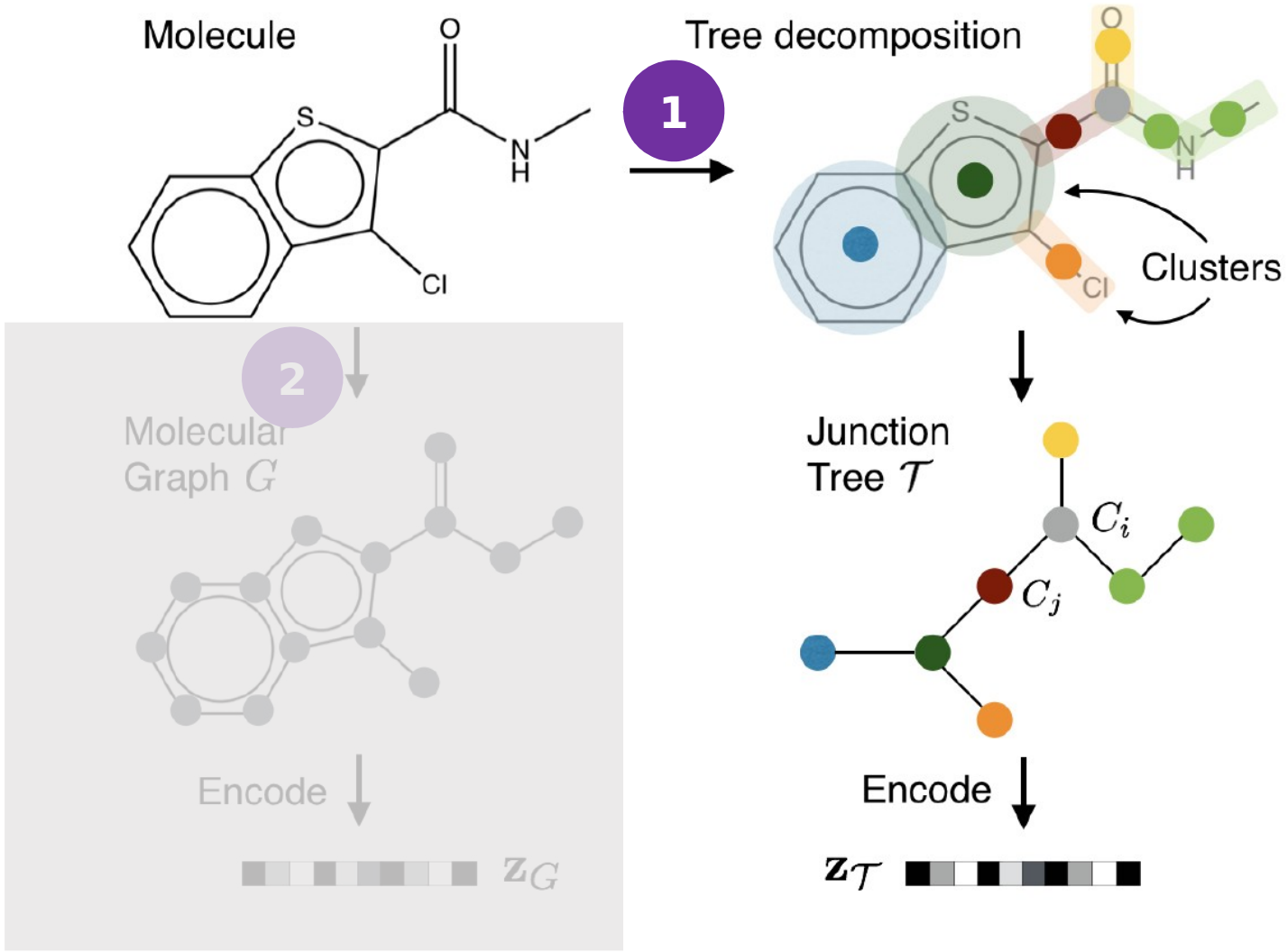
"Our junction tree variational autoencoder generates molecular graphs in two phases, by first generating a tree-structured scaffold over chemical substructures, and then combining them into a molecule with a graph message passing network. This approach allows us to incrementally expand molecules while maintaining chemical validity at every step. "



arXiv:1802.04364v2

https://github.com/wengong-jin/icml18-jtnn

Dataset :ZINC molecule dataset from Kusner et al. (2017)

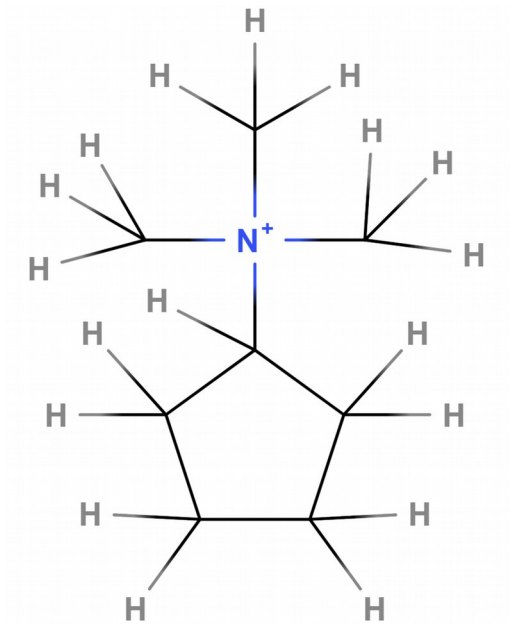

Molecule
O
S
N
H
Cl
1
Tree decomposition
O
S
N
H
Cl
Clusters
2
Molecular
Graph G
Encode
$\mathbf{z}_G$
Junction
Tree $\mathcal{T}$
C_i
C_j
Encode
$\mathbf{z}_\mathcal{T}$

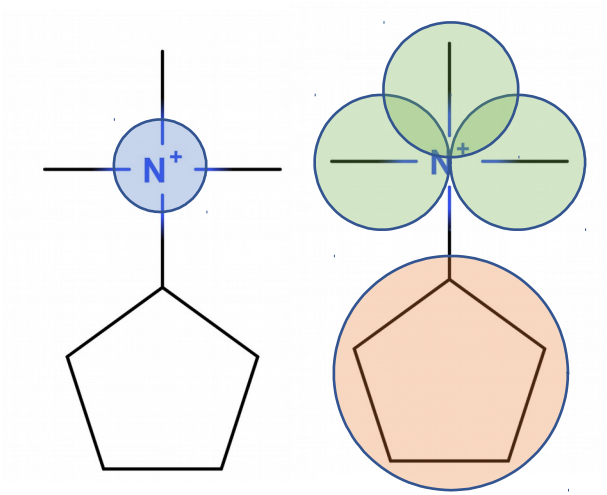
Tree Decomposition

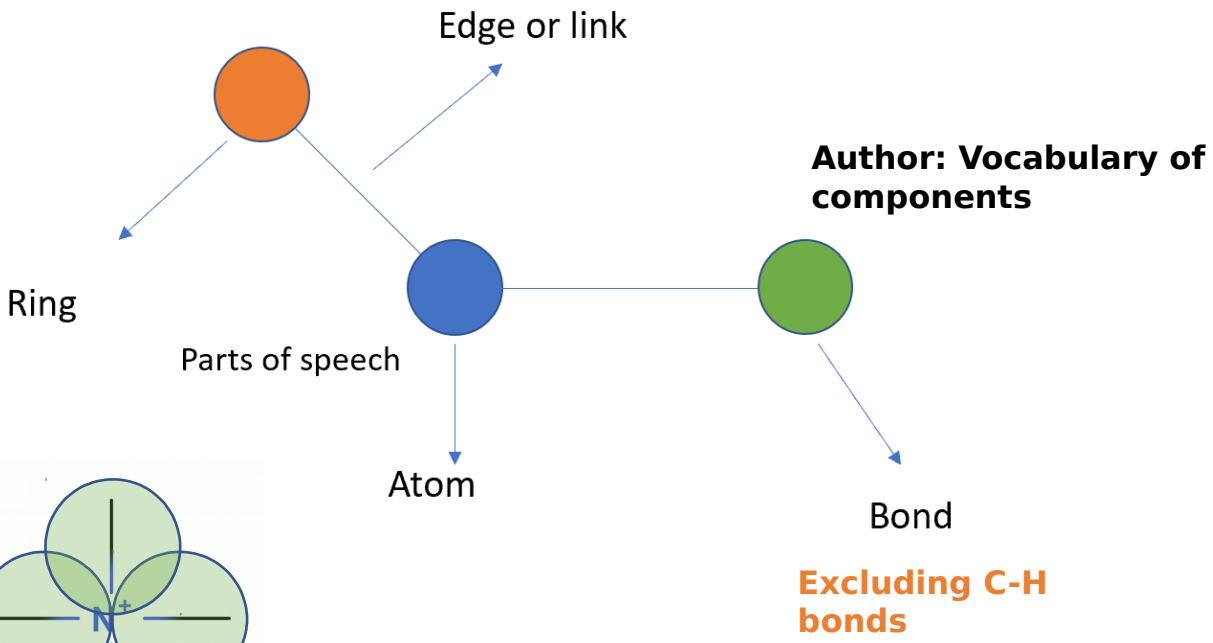
"If you change the rules on what controls you, you will change the rules on what you can control."

IN/ If PRP/ you VBP/ change DT/ the NNS/ rules IN/ on WP/ what VBZ/ controls PRP/ you ,/ , PRP/ you MD/ will VB/ change DT/ the NNS/ rules IN/ on WP/ what PRP/ you MD/ can VB/ control ./ .

Molecule

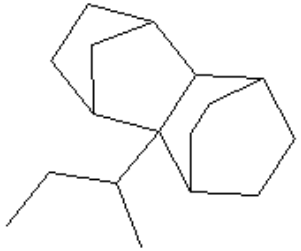


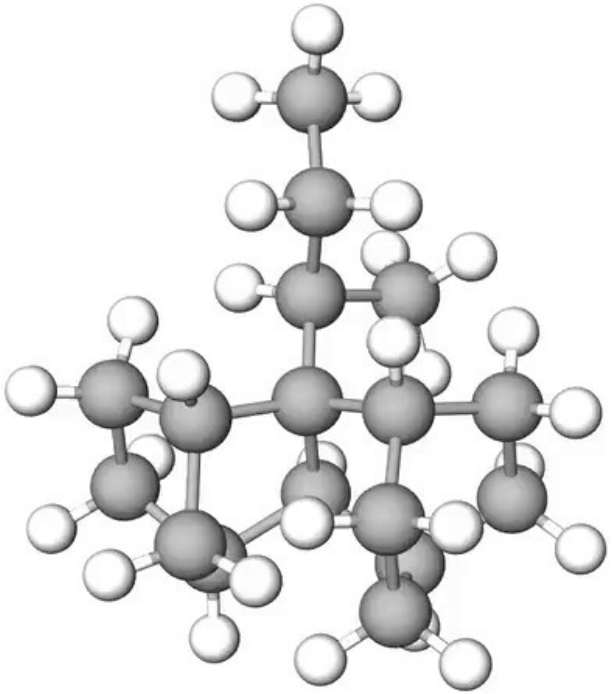


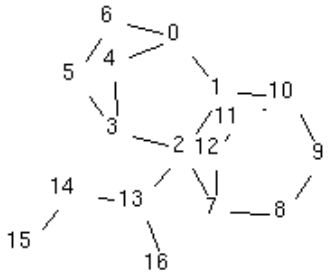


http://cogcomp.org/page/demo_view/pos

Molecule used in this presentation





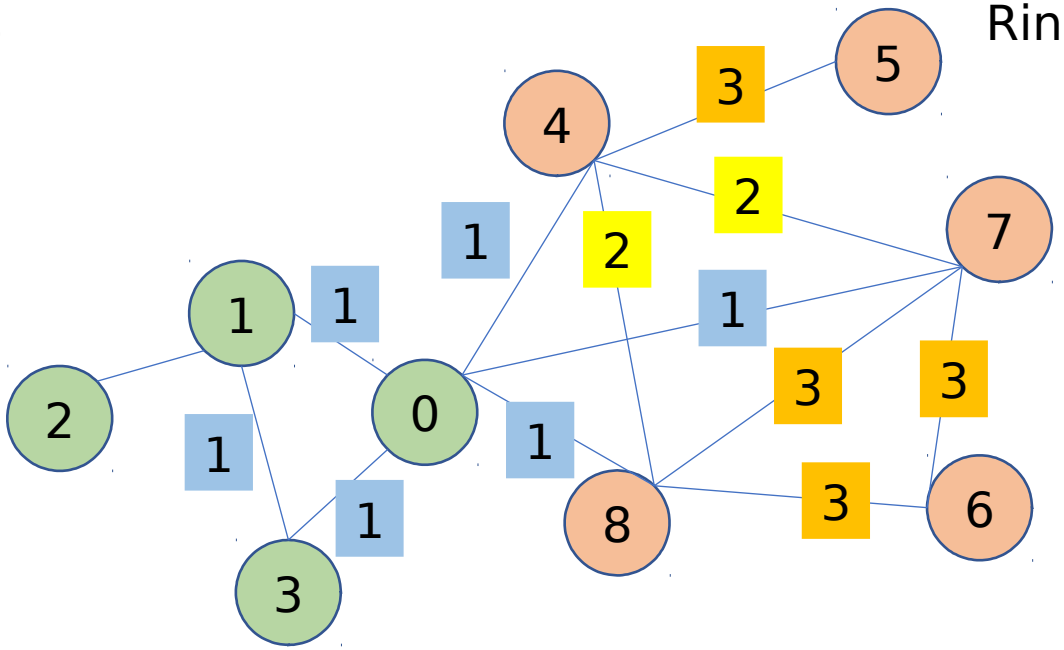


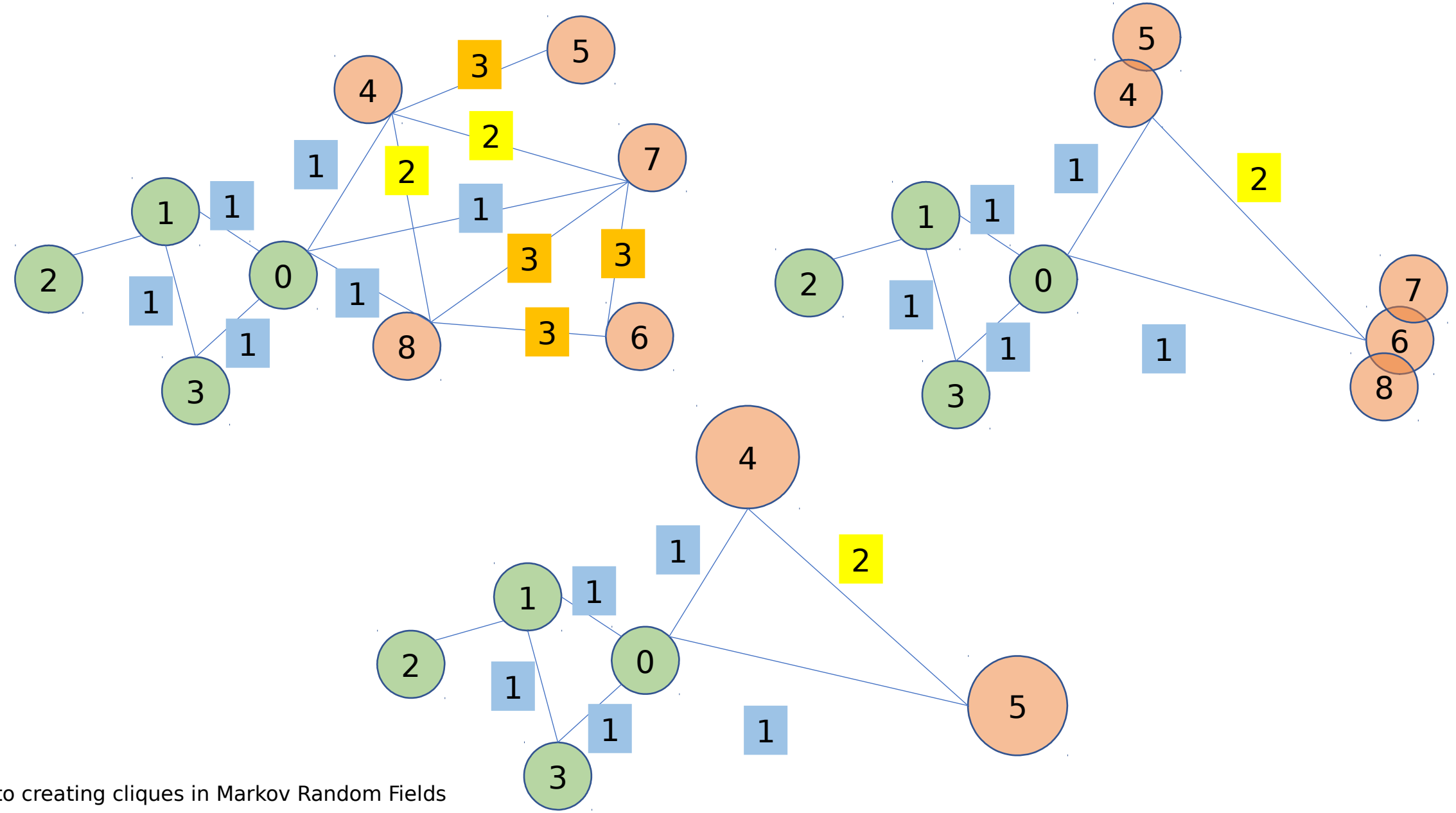
0 1 2 3 4 5 6 7 8

```
[[2, 13], [13, 14], [14, 15], [13, 16], [4, 3, 2, 1, 0], [4, 3, 5, 6, 0], [8, 9, 10, 11, 12, 7], [8, 9, 10, 1, 2, 7], [11, 12, 7, 2, 1, 10]]
```

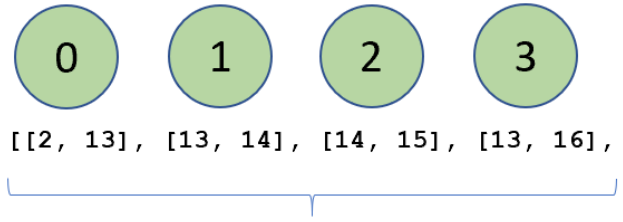
Bonds

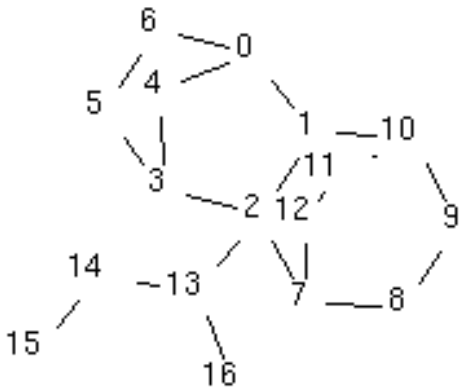
Rings

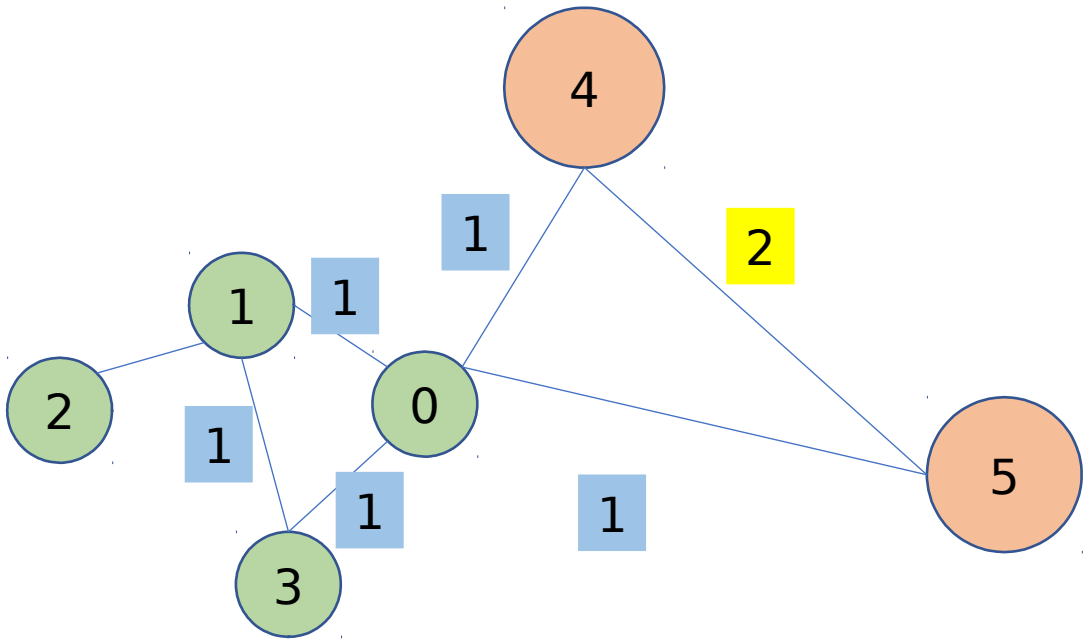


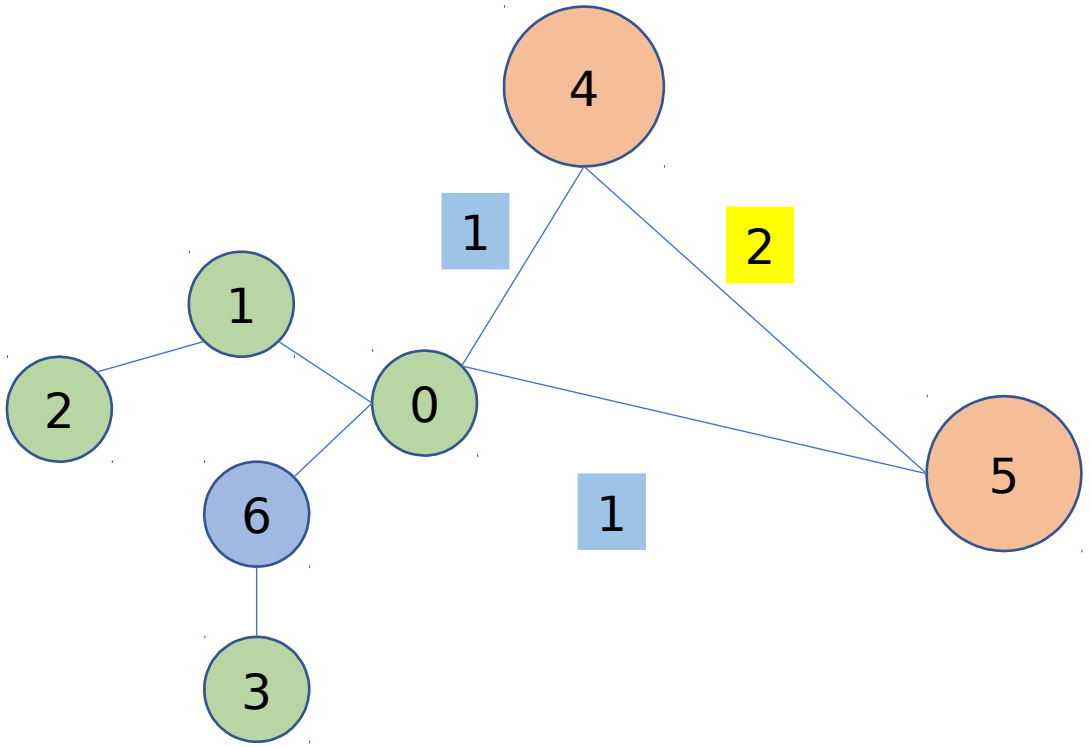


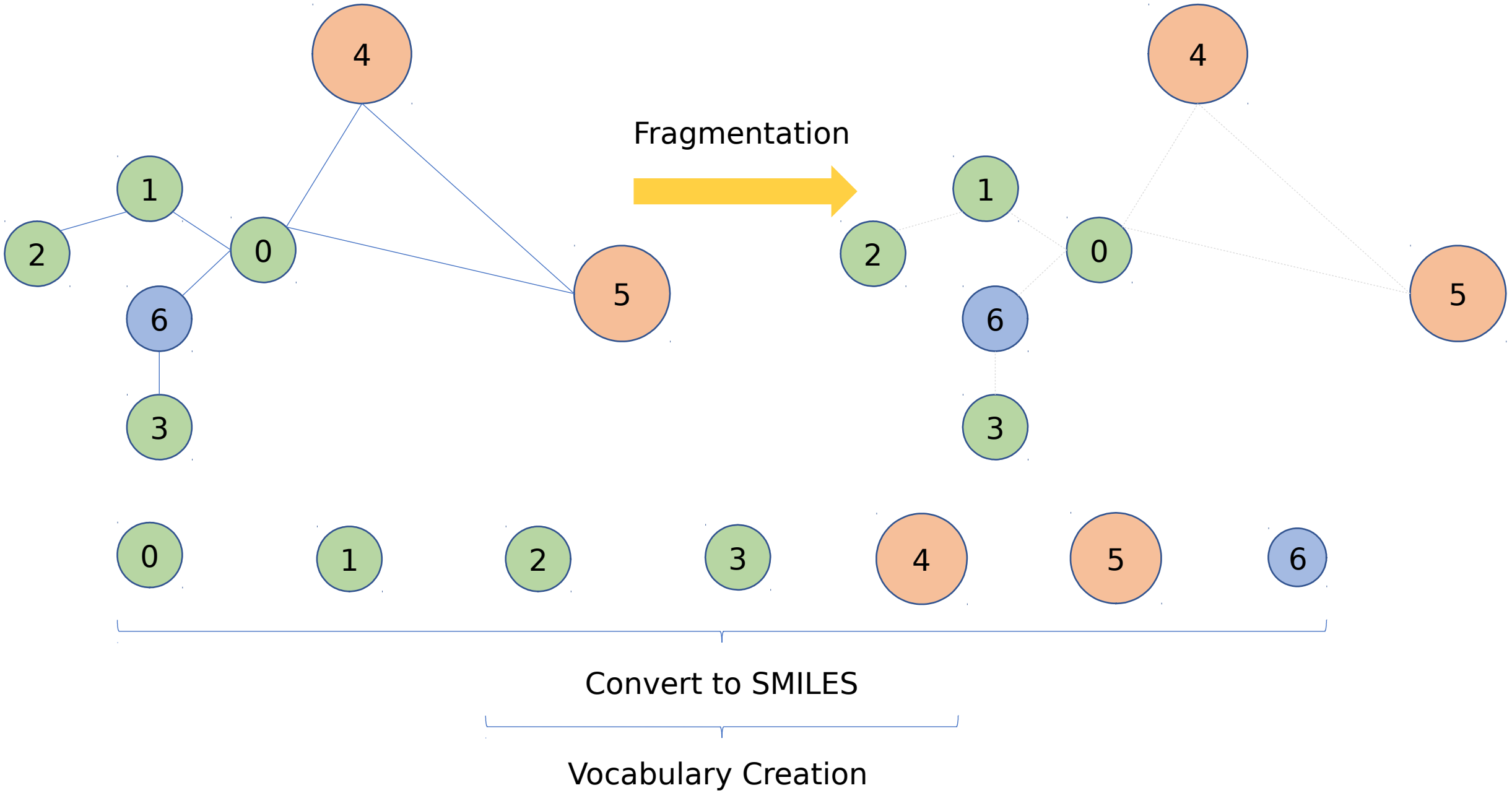
Similar to creating cliques in Markov Random Fields

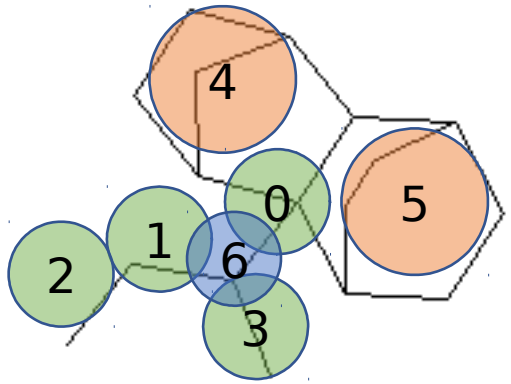

0
1
2
3
[[2, 13], [13, 14], [14, 15], [13, 16],
Bonds

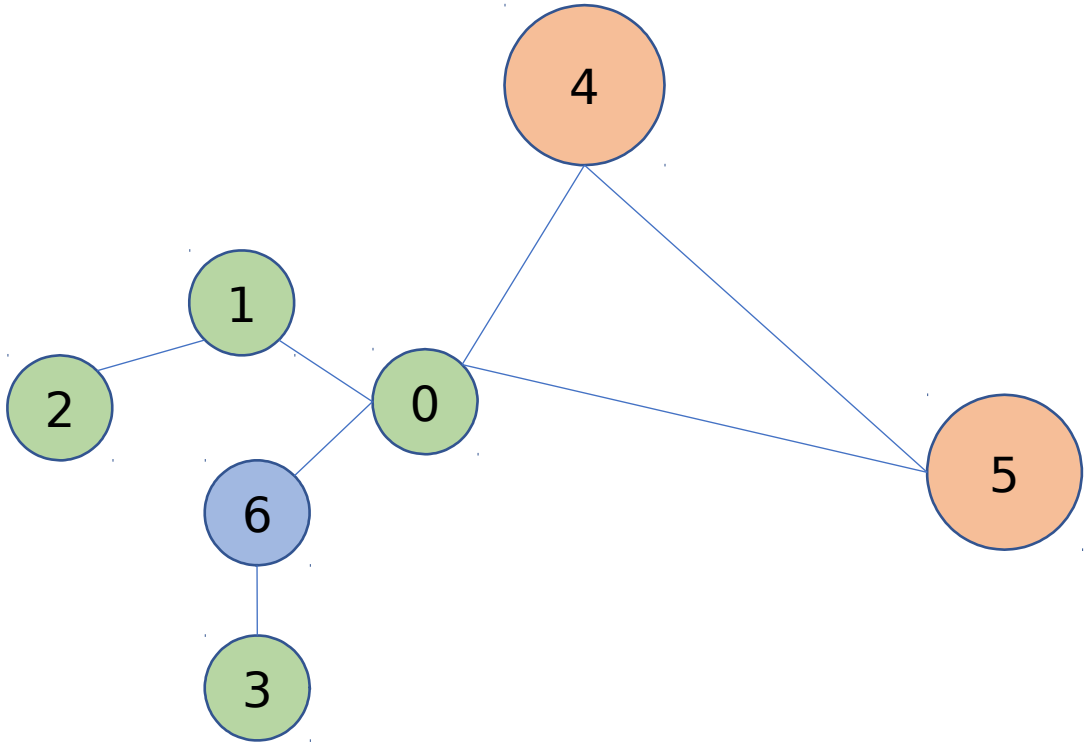

6
0
5
4
1
10
3
11
2
12
9
14
13
7
8
15
16

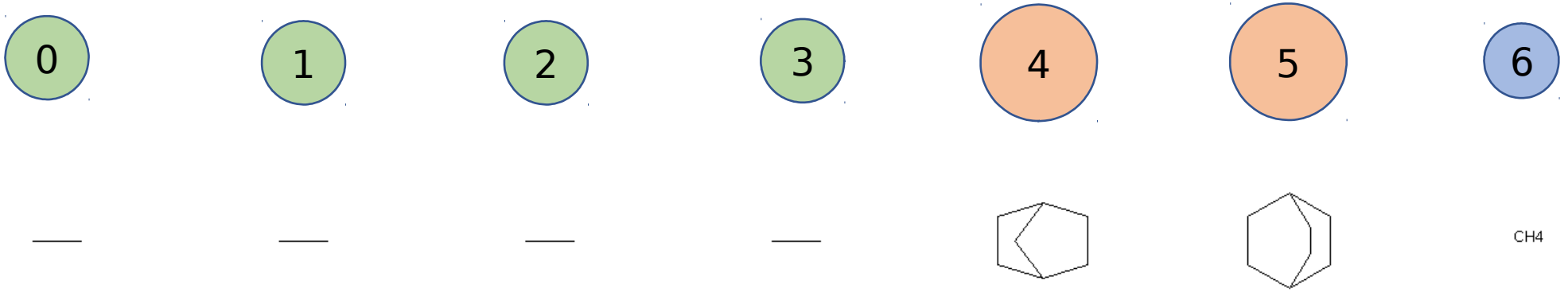

4
1
2
1
1
2
0
1
1
1
3
5


4
1
2
1
2
0
6
1
5
3


4
1
2
0
6
5
3
Fragmentation
4
1
2
0
6
5
3
0
1
2
3
4
5
6
Convert to SMILES
Vocabulary Creation

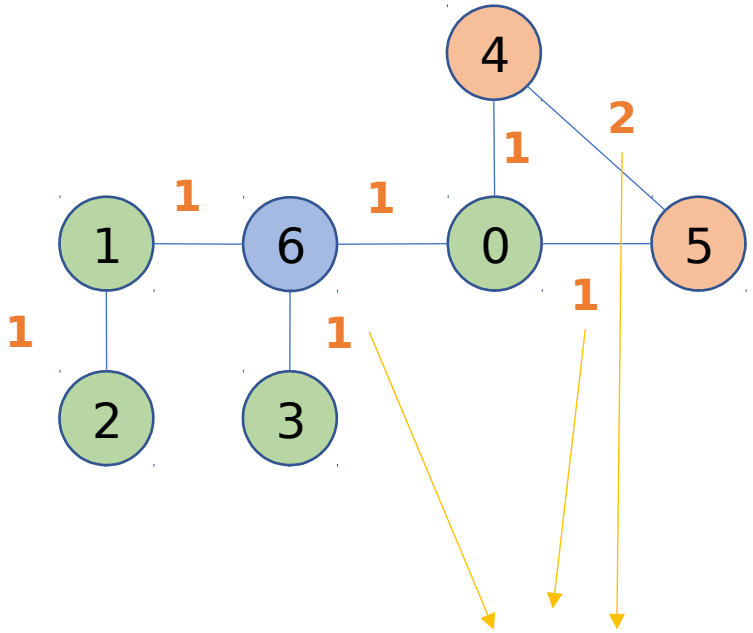




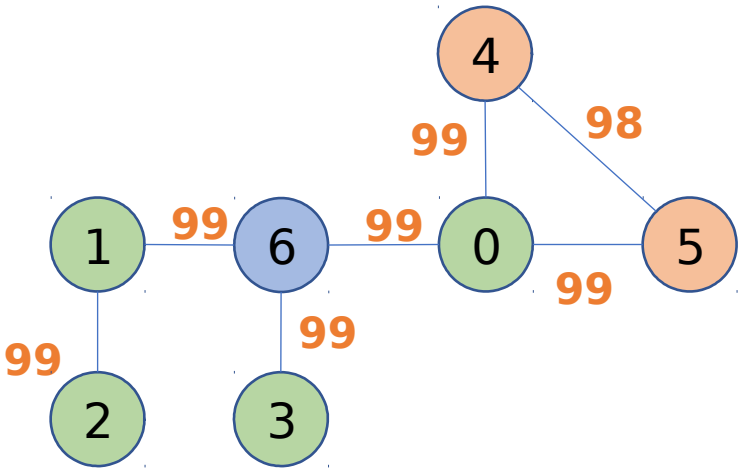


VOCAB	CC	CC	CC	CC	C1CC2CCC1C2	C1CC2CCC1CC2	C	SMILES
id	23846723	23846723	23846723	23846723	23846725	23846728	23846729	

Finding minimum spanning tree

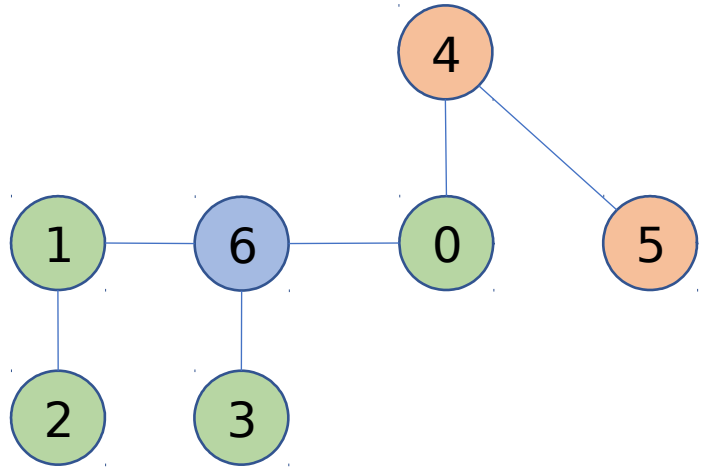


Common atoms between cliques (intersect)



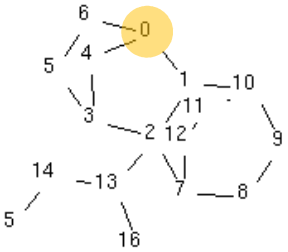
(0, 4)	99
(0, 5)	99
(0, 6)	99
(1, 2)	99
(1, 6)	99
(3, 6)	99
(4, 5)	98

Edges

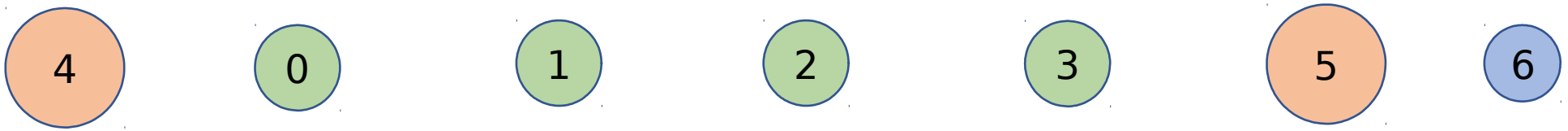


(0, 4)	99.0
(0, 6)	99.0
(1, 2)	99.0
(1, 6)	99.0
(3, 6)	99.0
(4, 5)	98.0

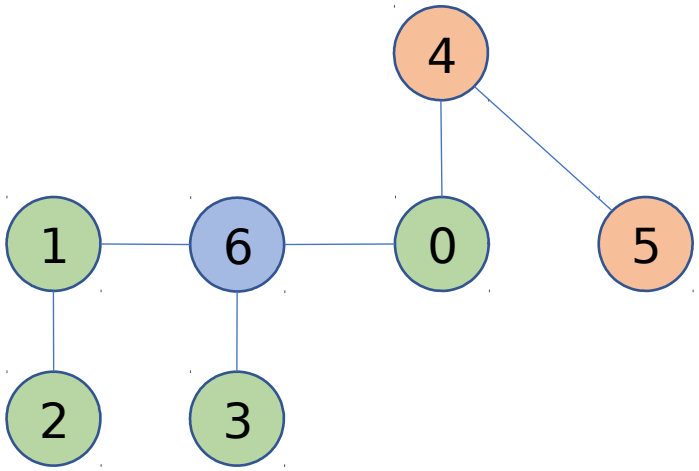
Node, Root, Leaf, singleton Assignment

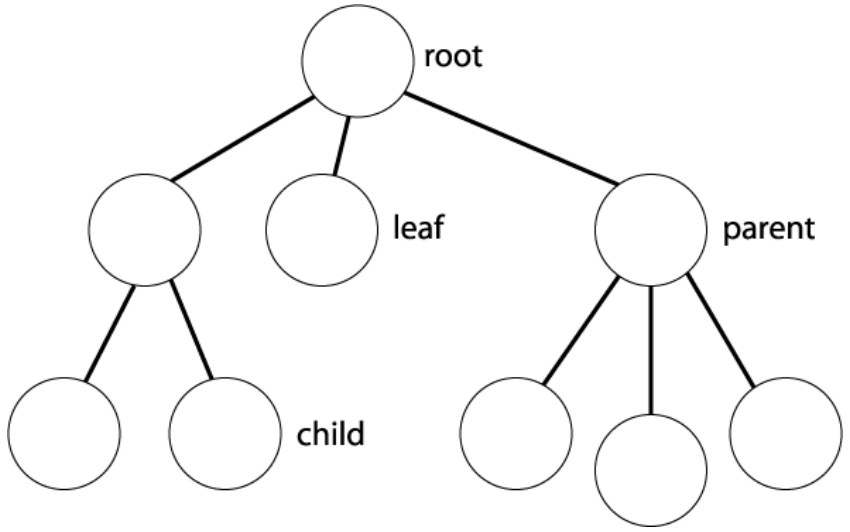


Root



Leaf Leaf Leaf Singleton





Source unknown

SMILES to Mol Tree

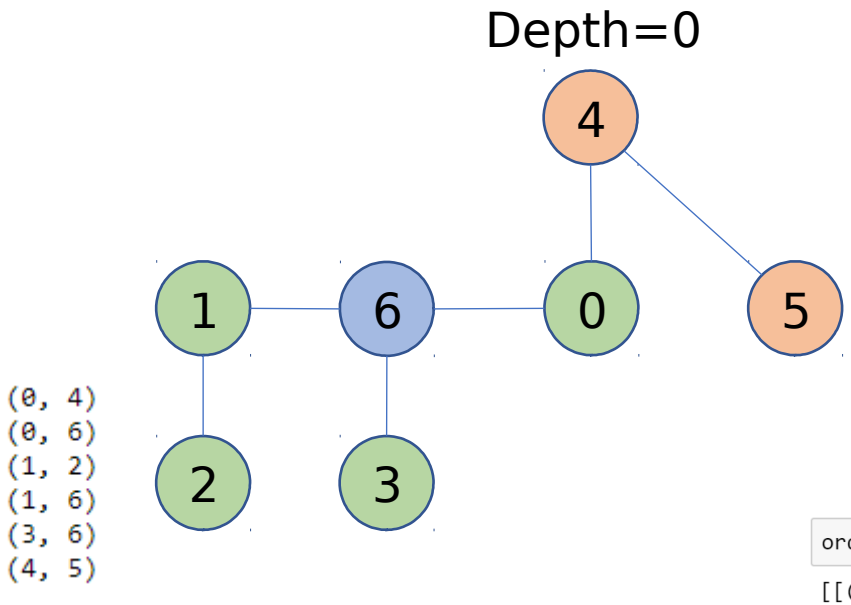
SMILES

1	C12C3C(C(C1)CC2)(C4CCC3CC4)C(CC)C
2	CCCCCCC1=NN2C(=N)/C(=C\c3cc(C)n(-c4ccc(C)cc4C)c3C)C(=O)N=C2S1
3	COCC[C@@H](C)C(=O)N(C)Cc1ccc(O)cc1
4	C=CCn1c(S[C@H](C)c2nc3sc(C)c(C)c3c(=O)[nH]2)nnc1C1CC1
5	C[NH+](C/C=C/c1ccco1)CCC(F)(F)F
6	COc1ccc(N2C(=O)C(=O)N(CN3CCC(c4nc5ccccc5s4)CC3)C2=O)cc1
7	Cc1ccc([C@@H](C)[NH2+][C@H](C)C(=O)Nc2ccccc2F)cc1
8	O=c1cc(C[NH2+]Cc2cccc(Cl)c2)nc(N2CCCC2)[nH]1
9	O=C(Cn1nc(C(=O)[O-])c2ccccc2c1=O)Nc1ccc2c(c1)C(=O)c1ccccc1C2=O
10	O=C(Nc1cccc(S(=O)(=O)N2CCCCC2)c1)c1cc(F)c(F)cc1Cl
11	CC(C)Cc1nnc(NC(=O)C(=O)NCCC2CCCCC2)s1
12	C[C@H](NC(=O)[C@@H](O)c1ccccc1)c1nnc2ccccn12
13	O=S(=O)(Nc1cc(F)ccc1F)c1ccc(Cl)cc1F
14	CSc1cc(C(=O)N2c3ccccc3NC(=O)C[C@@H]2C)ccn1
15	CCCN1C(=O)c2[nH]nc(-c3cc(Cl)ccc3O)c2[C@H]1c1ccc(C)cc1
16	CC[S@@](=O)[C@@H]1CCC[C@H](NC(=O)N(Cc2cccs2)C2CC2)C1
17	C[C@@H](c1ccco1)[NH+](Cc1ncc(-c2ccccc2)o1)C1CC1
18	COc1ccc(Cc2nnc(SCC(=O)N3CCC[C@@H](C)C3)o2)cc1
19	O=C(/C=C/c1ccc2c(c1)OCO2)NC[C@@H]1C[NH+]2CCN1CC2
20	COc1ccccc1/C=C/C=C(\C#N)C(=O)Nc1ccc(C(=O)N(C)C)cc1
21	Cc1cccc(NC(=S)N2CC[NH+](C)CC2)c1C
22	CNC(=O)c1cc(NC(=O)N2C[C@H]3CCC[NH+]3C[C@H]2C)ccc1F
23	CCCc1ccc([C@@H]([NH3+])C(OC)OC)cc1
24	[NH3+]C[C@H](c1cc(Cl)cs1)N1CC[C@@H]2CCCC[C@@H]2C1
25	CC(C)CN(CC(C)C)C(=O)NCC(C)(C)N1CCOCC1
26	COc1cc(C(=O)N[C@H]2C[C@H]2c2cccc(Cl)c2)cc(OC)c1OC
27	COCCNC(=O)c1cccc(N2CC[NH2+][C@@H](c3ccccc3)C2)n1
28	Cc1ccc(-c2cc(OCC(=O)[O-])nc(NCc3ccc4c(c3)OCO4)n2)cc1
29	COc1cccc(NC2CC[NH+](C[C@@H](O)CN3C[C@H](C)O[C@@H](C)C3)CC2)c1

Mol Tree

```
tree
```

	neighbors	node_edges	old_nodes	node_atoms	root	leaf	fragment_smiles	fragment_mols	clique_atoms	singletons	node	clique_nodes
0	[0, 5]	[[4, 0], [4, 5]]	4	[0, 1, 2, 3, 4, 5, 6]	1	0	C1CC2CCC1C2	<rdkit.Chem.rdchem.Mol object at 0x000002AC8CC...	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]	0	5	[0, 5, 4]
1	[2, 6]	[[1, 2], [1, 6]]	1	[13, 14]	0	0	CC	<rdkit.Chem.rdchem.Mol object at 0x000002AC8CC...	[13, 14, 15]	0	2	[2, 6, 1]
2	[1]	[[2, 1]]	2	[14, 15]	0	1	CC	<rdkit.Chem.rdchem.Mol object at 0x000002AC8CC...	[13, 14, 15]	0	3	[1, 2]
3	[6]	[[3, 6]]	3	[13, 16]	0	1	CC	<rdkit.Chem.rdchem.Mol object at 0x000002AC8CC...	[16, 13]	0	4	[6, 3]
4	[4, 6]	[[0, 4], [0, 6]]	0	[2, 13]	0	0	CC	<rdkit.Chem.rdchem.Mol object at 0x000002AC96D...	[0, 1, 2, 3, 4, 5, 6, 13]	0	1	[4, 6, 0]
5	[4]	[[5, 4]]	5	[1, 2, 7, 8, 9, 10, 11, 12]	0	1	C1CC2CCC1CC2	<rdkit.Chem.rdchem.Mol object at 0x000002AC8CC...	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]	0	6	[4, 5]
6	[0, 1, 3]	[[6, 0], [6, 1], [6, 3]]	6	[13]	0	0	C	<rdkit.Chem.rdchem.Mol object at 0x000002AC8CC...	[16, 2, 13, 14]	1	7	[0, 1, 3, 6]



Tree Traversal

order1

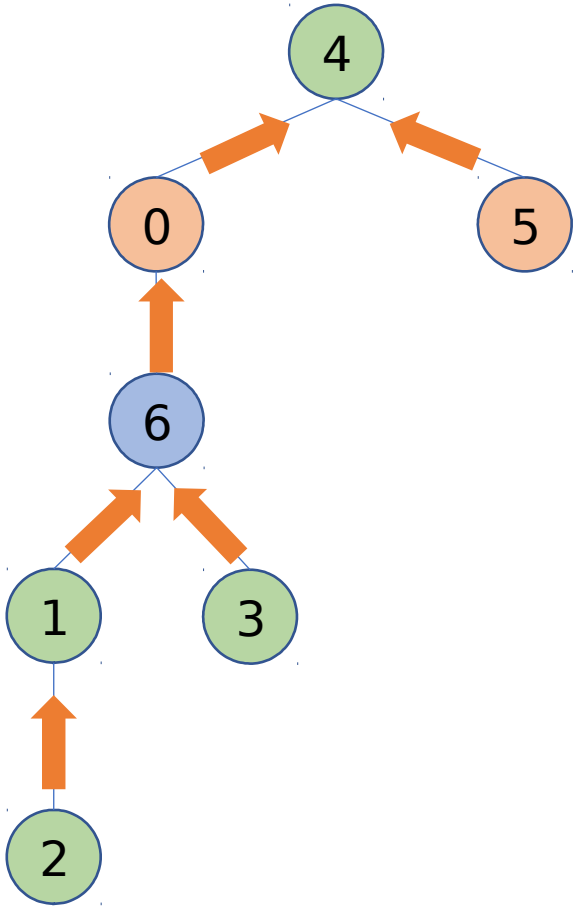
```
[[(4, 0), (4, 5)], [(0, 6)], [(6, 1), (6, 3)], [(1, 2)]]
```

order2

```
[[(0, 4), (5, 4)], [(6, 0)], [(1, 6), (3, 6)], [(2, 1)]]
```

```
[[(2, 1)],
 [(1, 6), (3, 6)],
 [(6, 0)],
 [(0, 4), (5, 4)],
 [(4, 0), (4, 5)],
 [(0, 6)],
 [(6, 1), (6, 3)],
 [(1, 2)]]
```

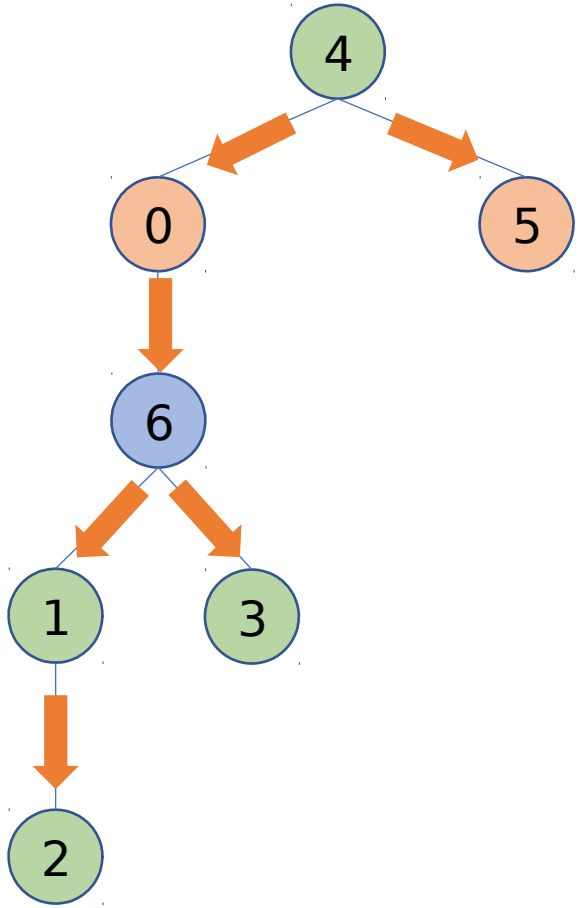
Generated for each molecule



0
1
2
3
4

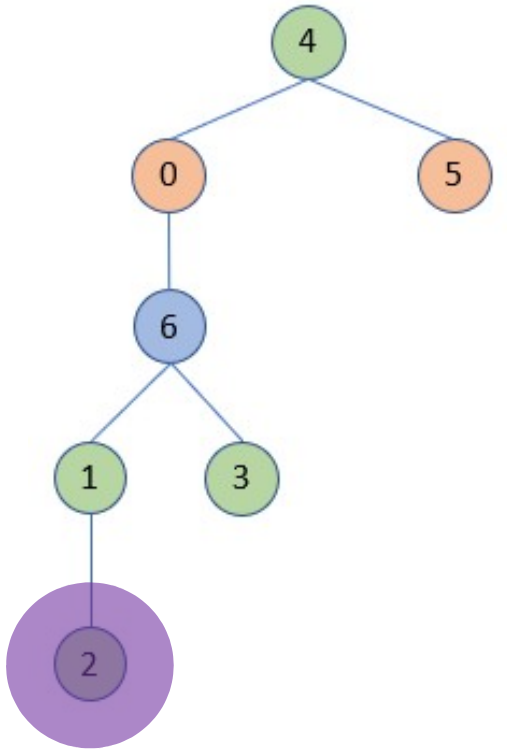
[(2, 1), (1, 6), (3, 6), (6, 0), (0, 4), (5, 4)]

Bottom up



[(4, 0), (4, 5), (0, 6), (6, 1), (6, 3), (1,

Top Down



Bottom-up shown only

[(2, 1), (1, 6), (3, 6), (6, 0), (0, 4), (5, 4), (4, 0), (4, 5)]

arbitrary leaf → **Get word id from vocabulary** → **Combine results for all compounds**

Batch Size=40

Size=420

```
tensor([[[ 1.4333e+00, -1.9953e-01,  1.0620e-01,  ...,  4.1402e-01,
          -5.0984e-01, -2.7352e-01],
         [-9.2368e-01, -2.8222e-01,  8.2388e-01,  ..., -2.2700e+00,
          -3.0900e-01,  2.0751e+00],
         [-7.9447e-01, -1.3179e+00, -1.1315e+00,  ...,  2.4284e+00,
           5.0793e-01,  1.1991e+00],
         ...,
         [-1.7938e-02,  8.7719e-01, -2.0225e-01,  ...,  1.4530e+00,
           1.0174e+00,  9.4328e-01],
         [-2.7240e-01, -1.1221e+00,  1.4381e+00,  ...,  9.0776e-01,
          -2.0167e+00, -3.5418e-01],
         [ 1.7372e+00,  6.8685e-01,  2.1169e-01,  ..., -4.4971e-01,
           2.3141e+00,  5.3667e-01]]])
```

torch.Size([40, 420])

embedding

750 words in dictionary

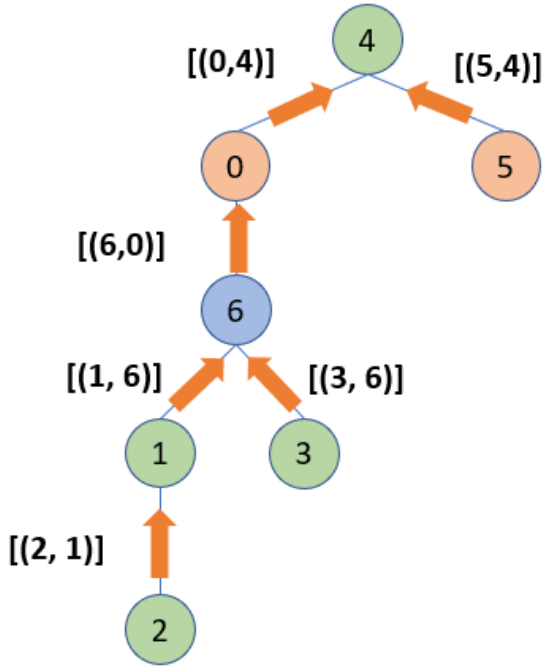
```
tensor([[  23,  568,  289,  490,  734,  463,  121,  305,  729,  110,
          529,  579,  388,  352,  147,  211,  169,   63,  331,  704,
          567,  321,  460,  578,   15,  385,  482,  414,   74,  617,
          123,  277,  306,  360,  312,  102,  427,  522,   60,  508]])
```

Batch Size=40

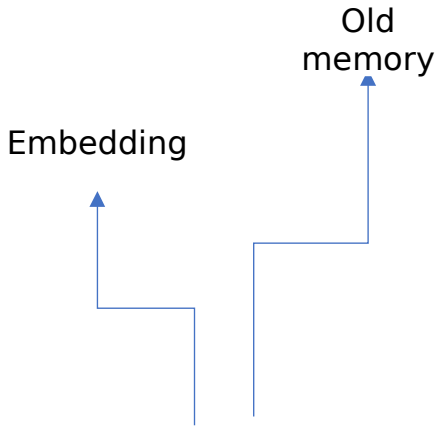
RU adapted for junction tree message pas

$$\mathbf{m}_{ij} = \mathrm{GRU}(\mathbf{x}_i, \{\mathbf{m}_{ki}\}_{k \in N(i) \setminus j})$$

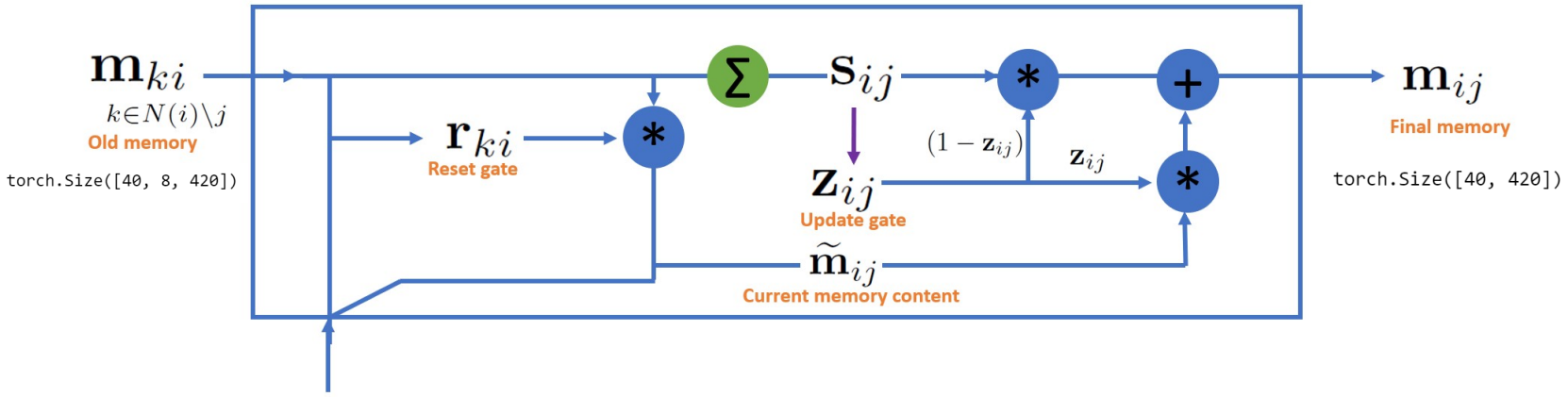
$$\begin{aligned}
\mathbf{s}_{ij} &= \sum_{k \in N(i) \setminus j} \mathbf{m}_{ki} \\
\mathbf{z}_{ij} &= \sigma(\mathbf{W}^z \mathbf{x}_i + \mathbf{U}^z \mathbf{s}_{ij} + \mathbf{b}^z) \\
\mathbf{r}_{ki} &= \sigma(\mathbf{W}^r \mathbf{x}_i + \mathbf{U}^r \mathbf{m}_{ki} + \mathbf{b}^r) \\
\widetilde{\mathbf{m}}_{ij} &= \tanh(\mathbf{W}\mathbf{x}_i + \mathbf{U} \sum_{k \in N(i) \setminus j} \mathbf{r}_{ki} \odot \mathbf{m}_{ki}) \\
\mathbf{m}_{ij} &= (1 - \mathbf{z}_{ij}) \odot \mathbf{s}_{ij} + \mathbf{z}_{ij} \odot \widetilde{\mathbf{m}}_{ij}
\end{aligned}$$



Bottom-up shown only

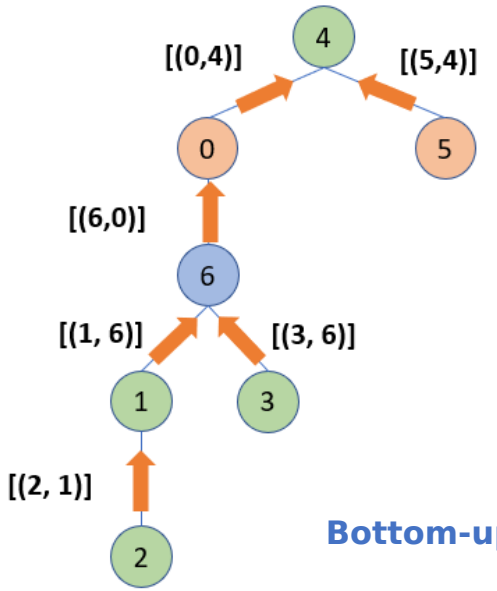



```
def GRU(x, h_nei, W_z, W_r, U_r, W_h):
    hidden_size = x.size()[-1]
    sum_h = h_nei.sum(dim=1)
    z_input = torch.cat([x,sum_h], dim=1)
    z = nn.Sigmoid()(W_z(z_input))

    r_1 = W_r(x).view(-1,1,hidden_size)
    r_2 = U_r(h_nei)
    r = nn.Sigmoid()(r_1 + r_2)

    gated_h = r * h_nei
    sum_gated_h = gated_h.sum(dim=1)
    h_input = torch.cat([x,sum_gated_h], dim=1)
    pre_h = nn.Tanh()(W_h(h_input))
    new_h = (1.0 - z) * sum_h + z * pre_h
    return new_h
```



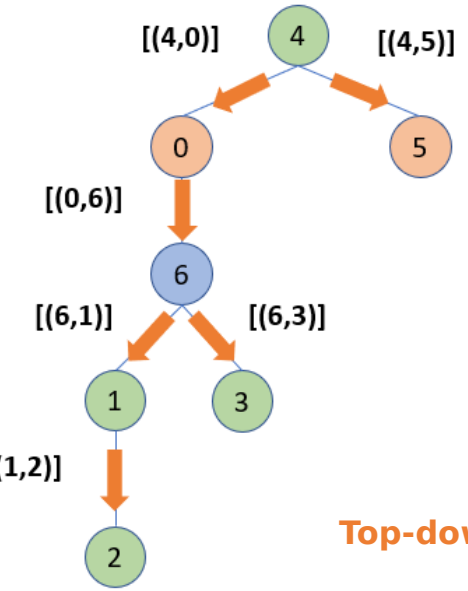
Batch=40
Neighbors=8
Hidden
size=450

[(2, 1), (1, 6), (3, 6), (6, 0), (0, 4), (5, 4)]



[2, 1] late
Bottom-up
mess
[1, 6] add neighbor message [2, 1]
[1, 6] - skip neighbor 6 calculate message [1, 6]
[3, 6] - skip neighbor 6 calculate message [3, 6]
[6, 0] - skip neighbor 0 calculate message [6, 0]
[6, 0] add neighbor message [1, 6]
[6, 0] add neighbor message [3, 6]
[0, 4] - skip neighbor 4 calculate message [0, 4]
[0, 4] add neighbor message [6, 0]
[5, 4] - skip neighbor 4 calculate message [5, 4]

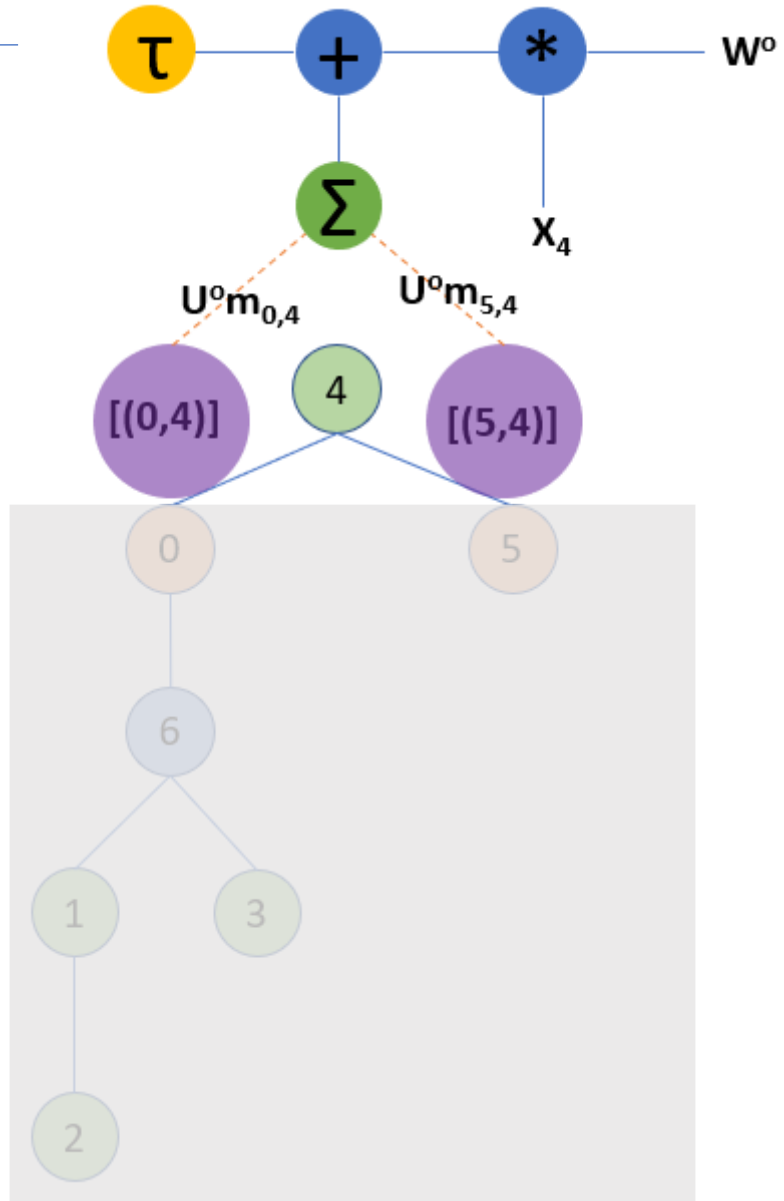
[(4, 0), (4, 5), (0, 6), (6, 1), (6, 3), (1, 2)]



[4, 0] ulate
Top-down
mes
[4, 0] add neighbor message [5, 4]
[4, 5] add neighbor message [0, 4]
[4, 5] - skip neighbor 5 calculate message [4, 5]
[0, 6] add neighbor message [4, 0]
[0, 6] - skip neighbor 6 calculate message [0, 6]
[6, 1] add neighbor message [0, 6]
[6, 1] - skip neighbor 1 calculate message [6, 1]
[6, 1] add neighbor message [3, 6]
[6, 3] add neighbor message [0, 6]
[6, 3] add neighbor message [1, 6]
[6, 3] - skip neighbor 3 calculate message [6, 3]

Node Aggregation

$$\mathbf{h}_{\mathcal{T}_G}$$



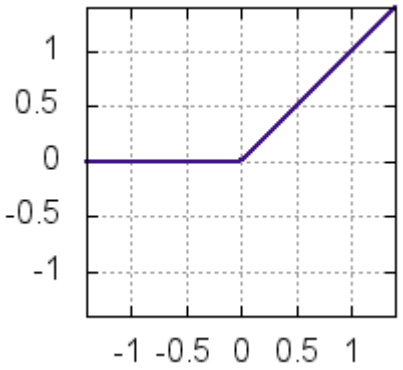
$$\mathbf{h}_i = \tau(\mathbf{W}^o \mathbf{x}_i + \sum_{k \in N(i)} \mathbf{U}^o \mathbf{m}_{ki})$$

torch.Size([40, 450])

torch.Size([40, 450])

word embedding

memory

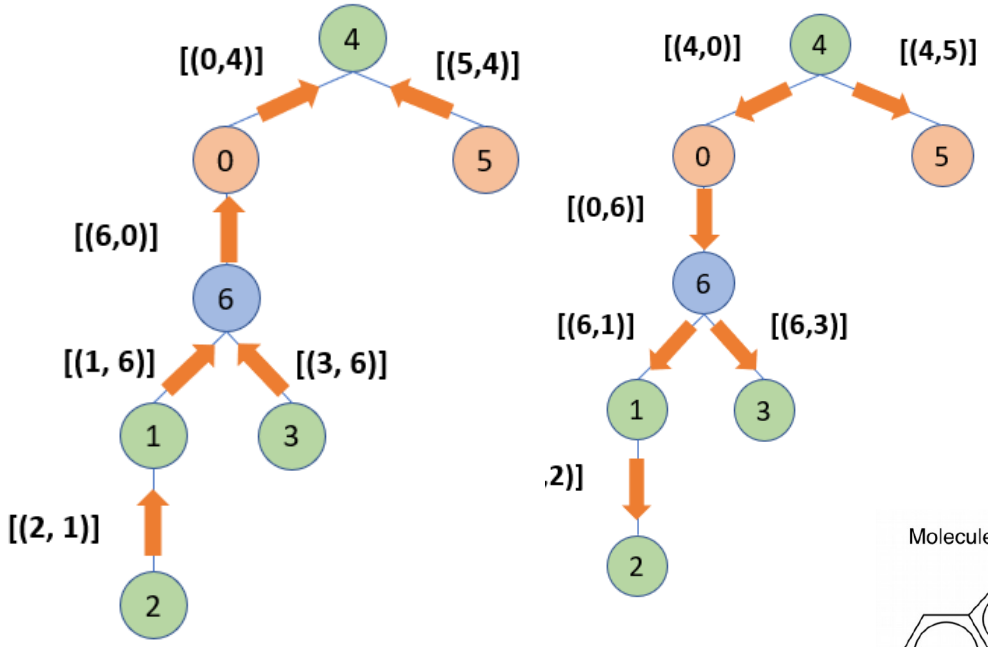


ReLU

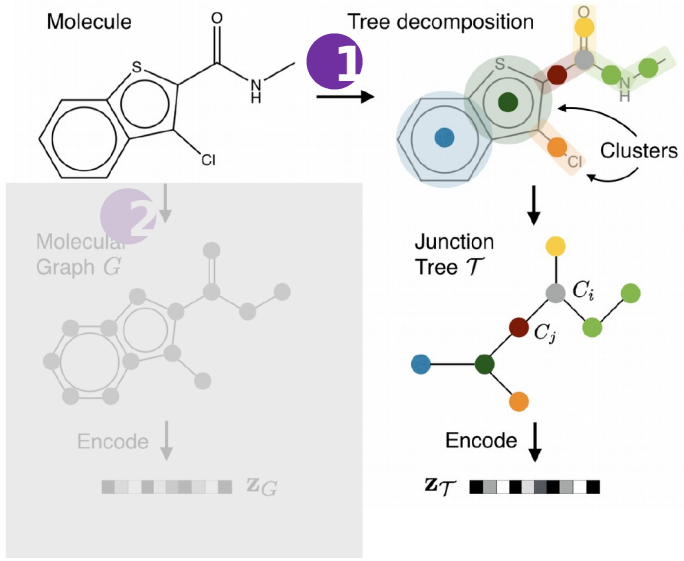
JT.nn Encoder output

$\mathbf{h}_{\mathcal{T}_G}$

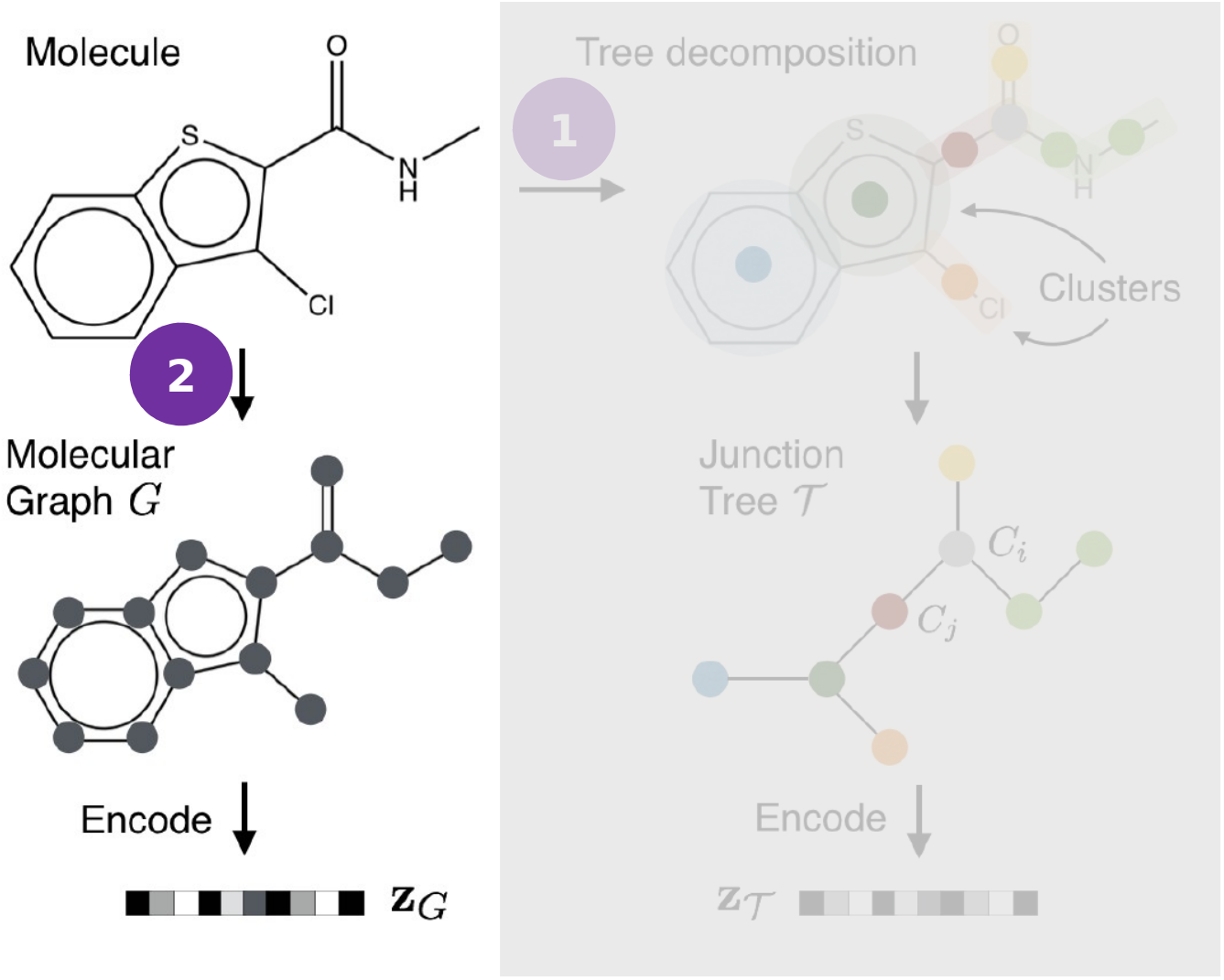
Tree vector



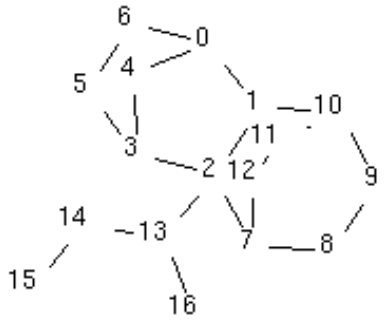
Tree message



Information is stored using node unique ids

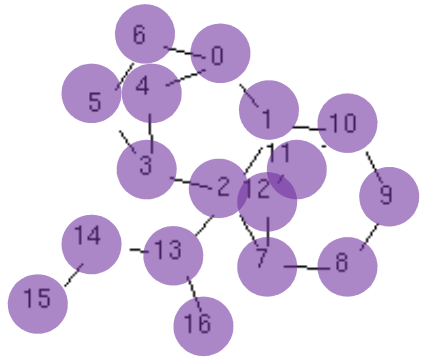

Molecule
O
S
N
H
Cl
2
Molecular
Graph G
Encode
$\mathbf{z}_G$
Tree decomposition
1
Clusters
Junction
Tree $\mathcal{T}$
C_i
C_j
Encode
$\mathbf{z}_\mathcal{T}$

Molecular Graph Generation



Molecular structure

1



Molecular Graph

2

Encode

Graph Message Passing Network

- fatoms
- fbonds

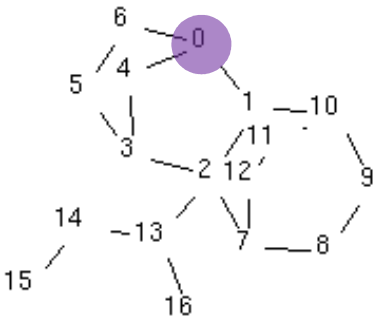
Features — **vertex**, **edge**

- agraph
- bgraph

Graphs

Scope: tracks atoms per molecule In the whole dataset

Graph Encoder – Atom Features



Index	Symbol	Number of bonds	Formal Charge	Chirality	Part of aromatic ring
0	C	3	0	0	FALSE
1	C	4	0	0	FALSE
2	C	3	0	0	FALSE
3	C	2	0	0	FALSE
4	C	2	0	0	FALSE
5	C	2	0	0	FALSE
6	C	3	0	0	FALSE
7	C	2	0	0	FALSE
8	C	2	0	0	FALSE
9	C	3	0	0	FALSE
10	C	2	0	0	FALSE
11	C	2	0	0	FALSE
12	C	3	0	0	FALSE
13	C	2	0	0	FALSE
14	C	1	0	0	FALSE
15	C	1	0	0	FALSE

Symbol :29, NB:6, FC:5, CH:4, A: 1

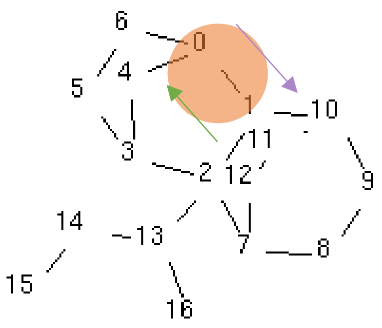
Mapping

```
tensor([ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,
         0.,  0.,  0.])

torch.Size([1,39])
```

Perform for all atoms

Atom Features (fatoms)

Graph Encoder – Bond Features



Bond Atoms	Type	Cis/Trans, E/Z
[0, 1]	SINGLE	0
[1, 2]	SINGLE	0
[2, 3]	SINGLE	0
[3, 4]	SINGLE	0
[3, 5]	SINGLE	0
[5, 6]	SINGLE	0
[2, 7]	SINGLE	0
[7, 8]	SINGLE	0
[8, 9]	SINGLE	0
[9, 10]	SINGLE	0
[10, 11]	SINGLE	0
[11, 12]	SINGLE	0
[2, 13]	SINGLE	0
[13, 14]	SINGLE	0
[14, 15]	SINGLE	0

Type:5, CT/EZ:6

Not all bonds shown

Mapping

torch.Size([2,11])

```
0 1
0 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
1 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
```

The same

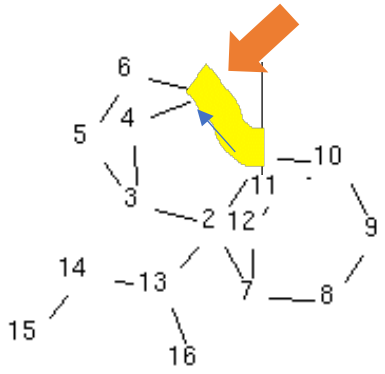
Perform for all bonds

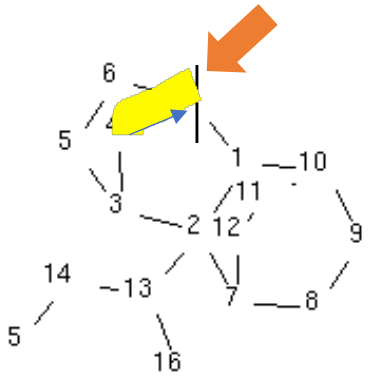
For 0-1 bon

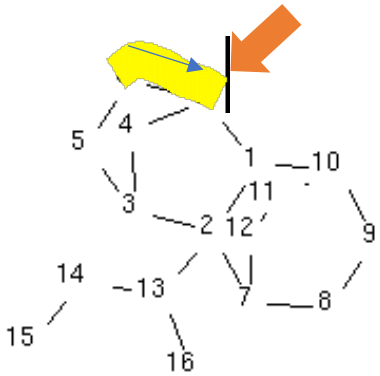
```
[tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.]),
 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])]
```

bond Feature

Graph Encoder – Atom graph







Atom

1 (0, 1)	21 (10, 11)
2 (1, 0)	22 (11, 10)
3 (1, 2)	23 (11, 12)
4 (2, 1)	24 (12, 11)
5 (2, 3)	25 (2, 13)
6 (3, 2)	26 (13, 2)
7 (3, 4)	27 (13, 14)
8 (4, 3)	28 (14, 13)
9 (3, 5)	29 (14, 15)
10 (5, 3)	30 (15, 14)
11 (5, 6)	31 (13, 16)
12 (6, 5)	32 (16, 13)
13 (2, 7)	33 (4, 0)
14 (7, 2)	34 (0, 4)
15 (7, 8)	35 (6, 0)
16 (8, 7)	36 (0, 6)
17 (8, 9)	37 (10, 1)
18 (9, 8)	38 (1, 10)
19 (9, 10)	39 (12, 7)
20 (10, 9)	40 (7, 12)

Related bonds

0 [2, 33, 35]
1 [1, 4, 37]
2 [3, 6, 14, 26]
3 [5, 8, 10]
4 [7, 34]
5 [9, 12]
6 [11, 36]
7 [13, 16, 39]
8 [15, 18]
9 [17, 20]
10 [19, 22, 38]
11 [21, 24]
12 [23, 40]
13 [25, 28, 32]
14 [27, 30]
15 [29]
16 [31]

```
tensor([[ 2,  33,  35,   0,   0,   0],
        [ 1,   4,  37,   0,   0,   0],
        [ 3,   6,  14,  26,   0,   0],
        [ 5,   8,  10,   0,   0,   0],
        [ 7,  34,   0,   0,   0,   0],
        [ 9,  12,   0,   0,   0,   0],
        [11,  36,   0,   0,   0,   0],
        [13,  16,  39,   0,   0,   0],
        [15,  18,   0,   0,   0,   0],
        [17,  20,   0,   0,   0,   0],
        [19,  22,  38,   0,   0,   0],
        [21,  24,   0,   0,   0,   0],
        [23,  40,   0,   0,   0,   0],
        [25,  28,  32,   0,   0,   0],
        [27,  30,   0,   0,   0,   0],
        [29,   0,   0,   0,   0,   0],
        [31,   0,   0,   0,   0,   0]])
```

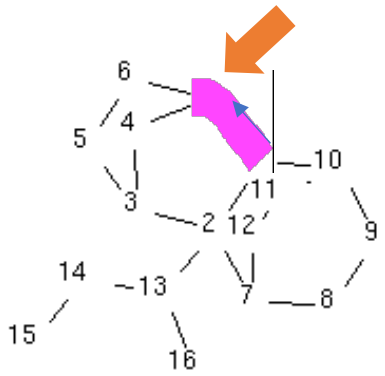
For atom 0

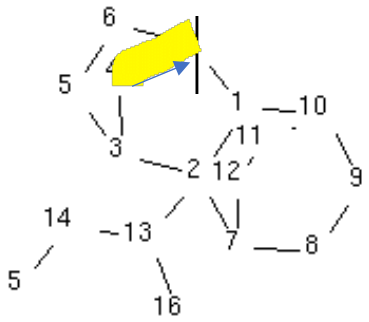
torch.Size([17,6]) for one molecule

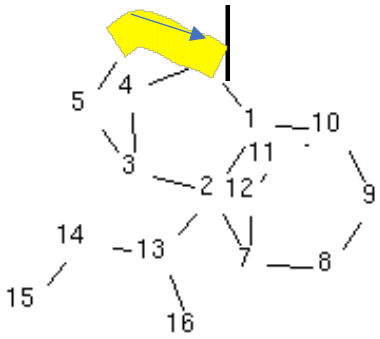
(agraph)

torch.Size([680,6]) for all molecules

Graph Encoder – Bond graph







1 (0, 1)	21 (10, 11)
2 (1, 0)	22 (11, 10)
3 (1, 2)	23 (11, 12)
4 (2, 1)	24 (12, 11)
5 (2, 3)	25 (2, 13)
6 (3, 2)	26 (13, 2)
7 (3, 4)	27 (13, 14)
8 (4, 3)	28 (14, 13)
9 (3, 5)	29 (14, 15)
10 (5, 3)	30 (15, 14)
11 (5, 6)	31 (13, 16)
12 (6, 5)	32 (16, 13)
13 (2, 7)	33 (4, 0)
14 (7, 2)	34 (0, 4)
15 (7, 8)	35 (6, 0)
16 (8, 7)	36 (0, 6)
17 (8, 9)	37 (10, 1)
18 (9, 8)	38 (1, 10)
19 (9, 10)	39 (12, 7)
20 (10, 9)	40 (7, 12)

(1, 0) **Is self**

0 [2, 33, 35]
1 [1, 4, 37]
2 [3, 6, 14, 26]
3 [5, 8, 10]
4 [7, 34]
5 [9, 12]
6 [11, 36]
7 [13, 16, 39]
8 [15, 18]
9 [17, 20]
10 [19, 22, 38]
11 [21, 24]
12 [23, 40]
13 [25, 28, 32]
14 [27, 30]
15 [29]
16 [31]

Max number of neighbors, n=6

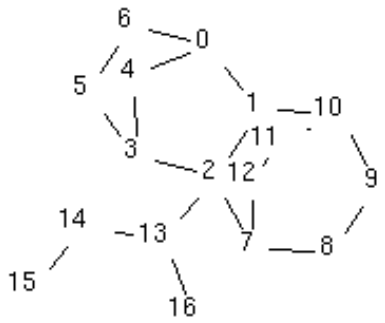
```
tensor([[ 0,  0,  0,  0,  0,  0],
        [ 0, 33, 35,  0,  0,  0],
        [ 0,  4, 37,  0,  0,  0],
        [ 1,  0, 37,  0,  0,  0],
        [ 0,  6, 14, 26,  0,  0],
        [ 3,  0, 14, 26,  0,  0],
        [ 0,  8, 10,  0,  0,  0],
        [ 5,  0, 10,  0,  0,  0],
        [ 0, 34,  0,  0,  0,  0],
        [ 5,  8,  0,  0,  0,  0],
        [ 0, 12,  0,  0,  0,  0],
        [ 9,  0,  0,  0,  0,  0],
        [ 0, 36,  0,  0,  0,  0],
        [ 3,  6,  0, 26,  0,  0],
        [ 0, 16, 39,  0,  0,  0],
        [13,  0, 39,  0,  0,  0],
        [ 0, 18,  0,  0,  0,  0],
        [15,  0,  0,  0,  0,  0],
        [ 0, 20,  0,  0,  0,  0],
        [17,  0,  0,  0,  0,  0],
        [ 0, 22, 38,  0,  0,  0],
        [19,  0, 38,  0,  0,  0],
        [ 0, 24,  0,  0,  0,  0],
        [21,  0,  0,  0,  0,  0],
        [ 0, 40,  0,  0,  0,  0],
        [ 3,  6, 14,  0,  0,  0],
        [ 0, 28, 32,  0,  0,  0],
        [25,  0, 32,  0,  0,  0],
        [ 0, 30,  0,  0,  0,  0],
        [27,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0],
        [25, 28,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0],
        [ 7,  0,  0,  0,  0,  0],
        [ 2,  0, 35,  0,  0,  0],
        [11,  0,  0,  0,  0,  0],
        [ 2, 33,  0,  0,  0,  0],
        [19, 22,  0,  0,  0,  0],
        [ 1,  4,  0,  0,  0,  0],
        [23,  0,  0,  0,  0,  0],
        [13, 16,  0,  0,  0,  0]])
```

For bond 0-1

torch.Size([40,6]) for one molecule

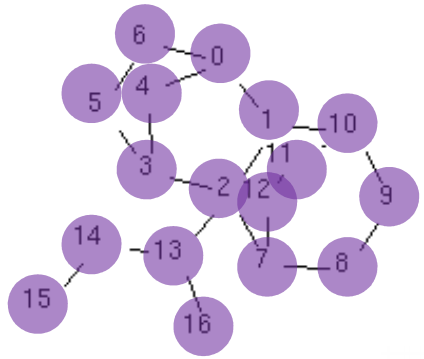
(bgraph)

torch.Size([1601,6]) for all molecules



Molec ule

1



Molecular Graph

fatoms

fbonds

agraph

bgraph

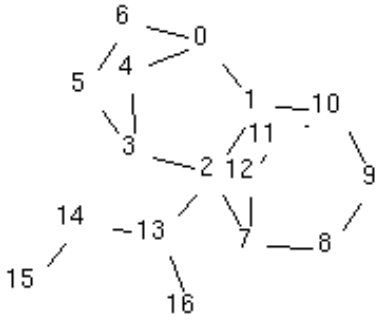
Scope: tracks atoms per molecu
In the whole dataset

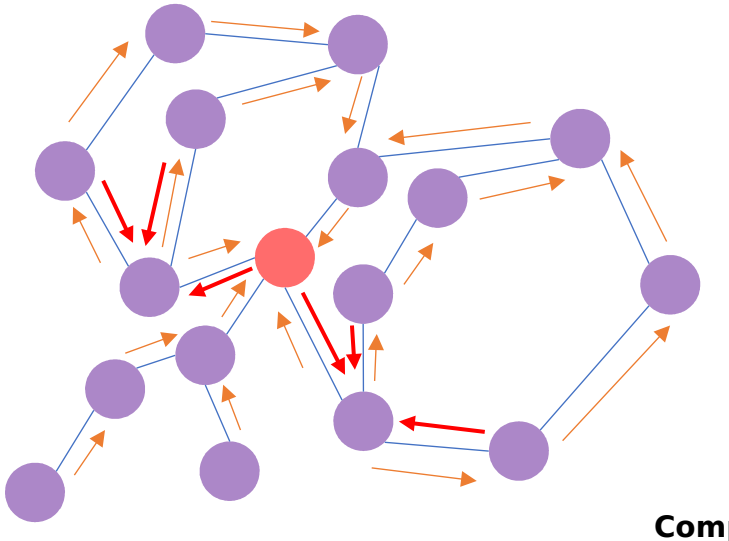
2

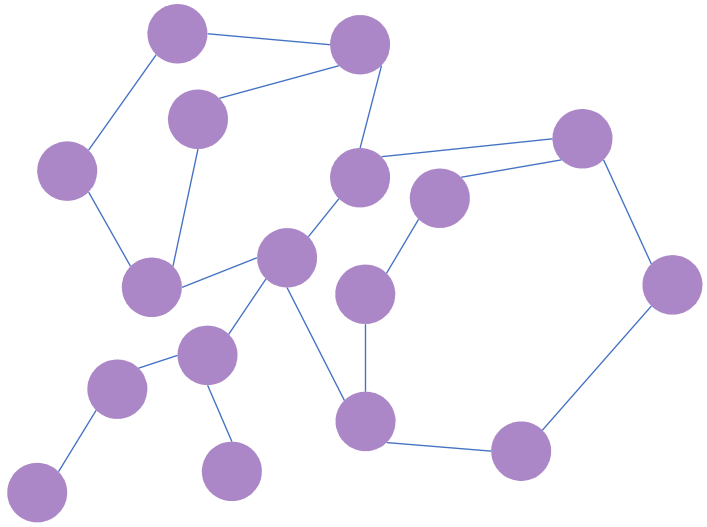
Enco de

Graph Message Passing Network

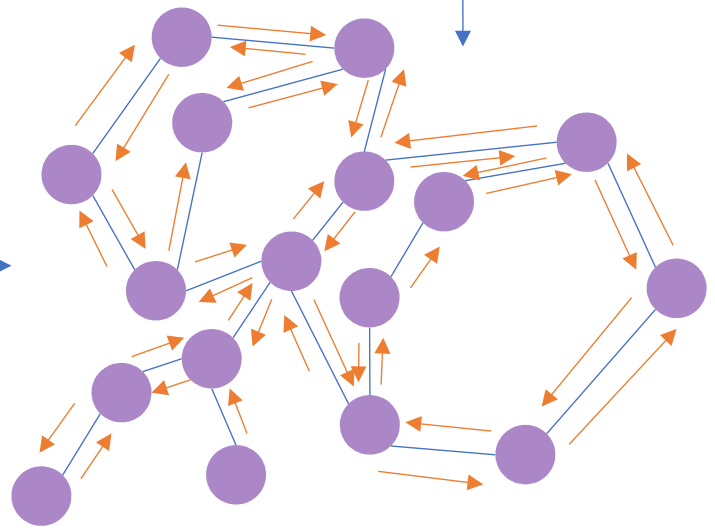
Loopy Belief Propagation







Initialize messages for all node



Compute messages locally (parallel schedule)

model

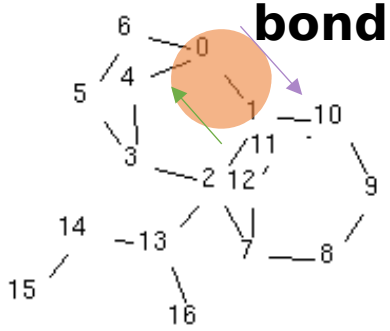
belief

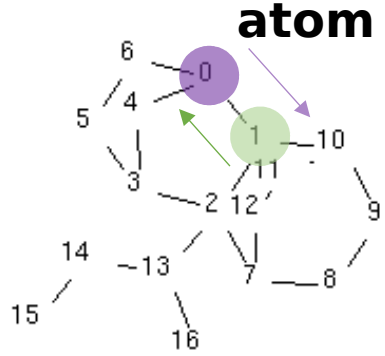
Compute all messages again

loop

Model Based Machine Learning - John Winn and Christopher Bishop (with Thomas Diethe)

Graph Message Passing Network





1 (0, 1)	21 (10, 11)
2 (1, 0)	22 (11, 10)
3 (1, 2)	23 (11, 12)
4 (2, 1)	24 (12, 11)
5 (2, 3)	25 (2, 13)
6 (3, 2)	26 (13, 2)
7 (3, 4)	27 (13, 14)
8 (4, 3)	28 (14, 13)
9 (3, 5)	29 (14, 15)
10 (5, 3)	30 (15, 14)
11 (5, 6)	31 (13, 16)
12 (6, 5)	32 (16, 13)
13 (2, 7)	33 (4, 0)
14 (7, 2)	34 (0, 4)
15 (7, 8)	35 (6, 0)
16 (8, 7)	36 (0, 6)
17 (8, 9)	37 (10, 1)
18 (9, 8)	38 (1, 10)
19 (9, 10)	39 (12, 7)
20 (10, 9)	40 (7, 12)

torch.Size([1,11])

```
0 1
0 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
1 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
```

bond Features

Atom Features (fatoms)

torch.Size([1,39])

```
tensor([ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,
         0.,  0.,  0.])
```

torch.Size([1,50])

```
0 tensor([ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,
         0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,
         0.,  0.])
```

LBP

```
0 1
0 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
1 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
```

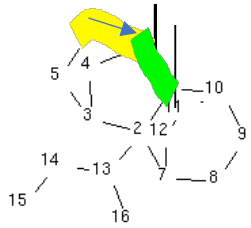
Perform for all atoms

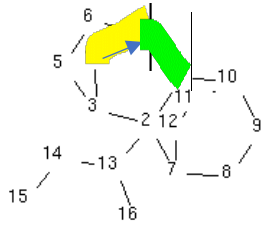
nn.linear → **ReLU** → **Messages**

torch.Size([1601, 50 torch.Size([1601, 450]) torch.Size([1601, 450])

20 bonds per molecule, 2 atoms per molecule, 40 tensors

LBP





$$\nu_{uv}^{(t)} = \tau(\mathbf{W}_1^g \mathbf{x}_u + \mathbf{W}_2^g \mathbf{x}_{uv} + \mathbf{W}_3^g \sum_{w \in N(u) \setminus v} \nu_{wu}^{(t-1)})$$

Message — Atom Features — Bond Features — Old messages from neighbors

Look up in messages

```
tensor([[  0,   0,   0,   0,   0,   0],
        [  0,  33,  35,   0,   0,   0],
        [  0,   4,  37,   0,   0,   0],
        [  1,   0,  37,   0,   0,   0],
        [  0,   6,  14,  26,   0,   0],
        [  3,   0,  14,  26,   0,   0],
        [  0,   8,  10,   0,   0,   0],
        [  5,   0,  10,   0,   0,   0],
        [  0,  34,   0,   0,   0,   0],
        [  5,   8,   0,   0,   0,   0],
        [  0,  12,   0,   0,   0,   0],
        [  9,   0,   0,   0,   0,   0],
        [  0,  36,   0,   0,   0,   0],
        [  3,   6,   0,  26,   0,   0],
        [  0,  16,  39,   0,   0,   0],
        [ 13,   0,  39,   0,   0,   0],
        [  0,  18,   0,   0,   0,   0],
        [ 15,   0,   0,   0,   0,   0],
        [  0,  20,   0,   0,   0,   0],
        [ 17,   0,   0,   0,   0,   0],
        [  0,  22,  38,   0,   0,   0],
        [ 19,   0,  38,   0,   0,   0],
        [  0,  24,   0,   0,   0,   0],
        [ 21,   0,   0,   0,   0,   0],
        [  0,  40,   0,   0,   0,   0],
        [  3,   6,  14,   0,   0,   0],
        [  0,  28,  32,   0,   0,   0],
        [ 25,   0,  32,   0,   0,   0],
        [  0,  30,   0,   0,   0,   0],
        [ 27,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0],
        [ 25,  28,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0],
        [  7,   0,   0,   0,   0,   0],
        [  2,   0,  35,   0,   0,   0],
        [ 11,   0,   0,   0,   0,   0],
        [  2,  33,   0,   0,   0,   0],
        [ 19,  22,   0,   0,   0,   0],
        [  1,   4,   0,   0,   0,   0],
        [ 23,   0,   0,   0,   0,   0],
        [ 13,  16,   0,   0,   0,   0]])
```

torch.Size([41,6])

For bond 0-1

For bond 1-0

1 (0, 1)	21 (10, 11)
2 (1, 0)	22 (11, 10)
3 (1, 2)	23 (11, 12)
4 (2, 1)	24 (12, 11)
5 (2, 3)	25 (2, 13)
6 (3, 2)	26 (13, 2)
7 (3, 4)	27 (13, 14)
8 (4, 3)	28 (14, 13)
9 (3, 5)	29 (14, 15)
10 (5, 3)	30 (15, 14)
11 (5, 6)	31 (13, 16)
12 (6, 5)	32 (16, 13)
13 (2, 7)	33 (4, 0)
14 (7, 2)	34 (0, 4)
15 (7, 8)	35 (6, 0)
16 (8, 7)	36 (0, 6)
17 (8, 9)	37 (10, 1)
18 (9, 8)	38 (1, 10)
19 (9, 10)	39 (12, 7)
20 (10, 9)	40 (7, 12)

bgraph for one molecule only

Messages

```
tensor([[[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0601,  0.0000,  0.1630,  ...,  0.0000,  0.0000,  0.1028],
         [ 0.0677,  0.1672,  0.0931,  ...,  0.0000,  0.1168,  0.1680],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        ...,

        [[ 0.0677,  0.1672,  0.0931,  ...,  0.0000,  0.1168,  0.1680],
         [ 0.0601,  0.0000,  0.1630,  ...,  0.0000,  0.0000,  0.1028],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0601,  0.0000,  0.1630,  ...,  0.0000,  0.0000,  0.1028],
         [ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]]])
```

torch.Size([41, 6, 450])

padding

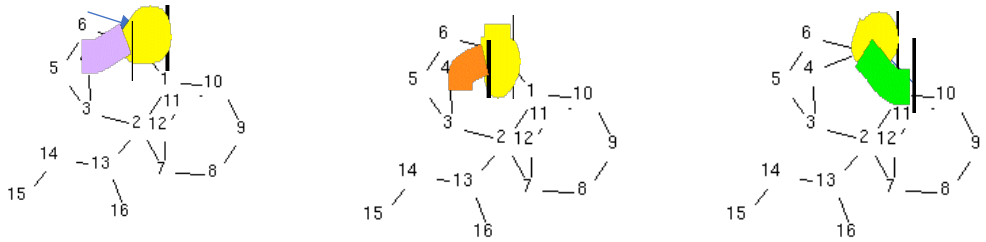
Loop

Sum dim=1 — **nn.Linear** — **Add to old message** — **ReLU**

stop after depth-1

New messages

Default is 3, provided as input for training



$$\mathbf{h}_u = \tau(\mathbf{U}_1^g \mathbf{x}_u + \sum_{v \in N(u)} \mathbf{U}_2^g \boldsymbol{\nu}_{vu}^{(T)})$$

vertex

Atom Features

final messages from neighbors

```
torch.Size([17, 6])
tensor([[ 2, 33, 35,  0,  0,  0],
        [ 1,  4, 37,  0,  0,  0],
        [ 3,  6, 14, 26,  0,  0],
        [ 5,  8, 10,  0,  0,  0],
        [ 7, 34,  0,  0,  0,  0],
        [ 9, 12,  0,  0,  0,  0],
        [11, 36,  0,  0,  0,  0],
        [13, 16, 39,  0,  0,  0],
        [15, 18,  0,  0,  0,  0],
        [17, 20,  0,  0,  0,  0],
        [19, 22, 38,  0,  0,  0],
        [21, 24,  0,  0,  0,  0],
        [23, 40,  0,  0,  0,  0],
        [25, 28, 32,  0,  0,  0],
        [27, 30,  0,  0,  0,  0],
        [29,  0,  0,  0,  0,  0],
        [31,  0,  0,  0,  0,  0]])
```

agraph for one molecule only

Look up in messages

For atom 0

```
tensor([[[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0601,  0.0000,  0.1630,  ...,  0.0000,  0.0000,  0.1028],
         [ 0.0677,  0.1672,  0.0931,  ...,  0.0000,  0.1168,  0.1680],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        ...,

        [[ 0.0677,  0.1672,  0.0931,  ...,  0.0000,  0.1168,  0.1680],
         [ 0.0601,  0.0000,  0.1630,  ...,  0.0000,  0.0000,  0.1028],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.0601,  0.0000,  0.1630,  ...,  0.0000,  0.0000,  0.1028],
         [ 0.1111,  0.2055,  0.0000,  ...,  0.0000,  0.0369,  0.0335],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]]])
```

Sum dim=1 → nn.linear → Molecular Vector

concat fatoms, dim=1

1 (0, 1)	21 (10, 11)
2 (1, 0)	22 (11, 10)
3 (1, 2)	23 (11, 12)
4 (2, 1)	24 (12, 11)
5 (2, 3)	25 (2, 13)
6 (3, 2)	26 (13, 2)
7 (3, 4)	27 (13, 14)
8 (4, 3)	28 (14, 13)
9 (3, 5)	29 (14, 15)
10 (5, 3)	30 (15, 14)
11 (5, 6)	31 (13, 16)
12 (6, 5)	32 (16, 13)
13 (2, 7)	
14 (7, 2)	34 (0, 4)
15 (7, 8)	
16 (8, 7)	36 (0, 6)
17 (8, 9)	37 (10, 1)
18 (9, 8)	38 (1, 10)
19 (9, 10)	39 (12, 7)
20 (10, 9)	40 (7, 12)

Final Messages torch.Size([41, 6, 450])

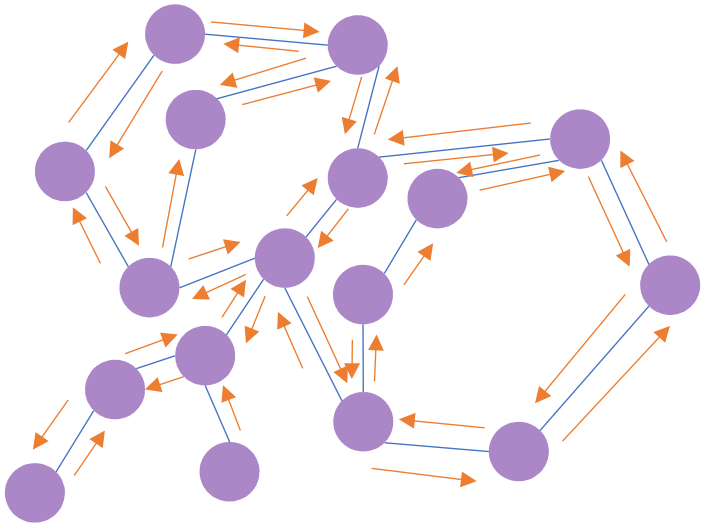
$$\mathbf{h}_G = \sum_i \mathbf{h}_i / |V|$$

Total number of vertices/atoms

Messages for each vertex/atom

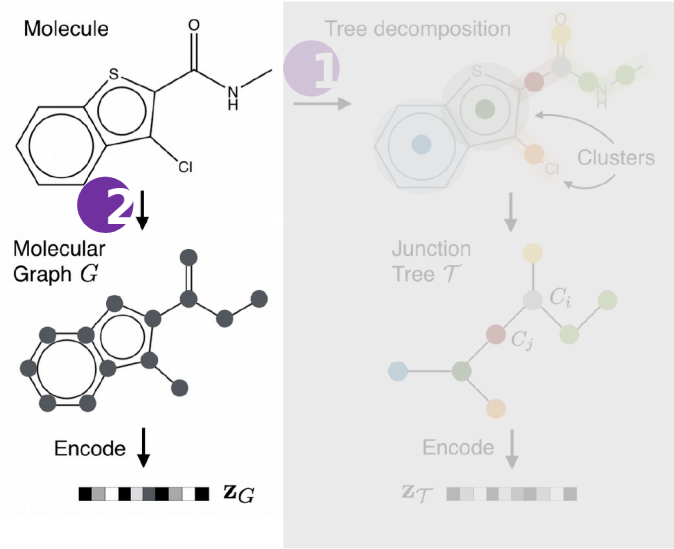
For each molecule

MPN.nn Encoder output

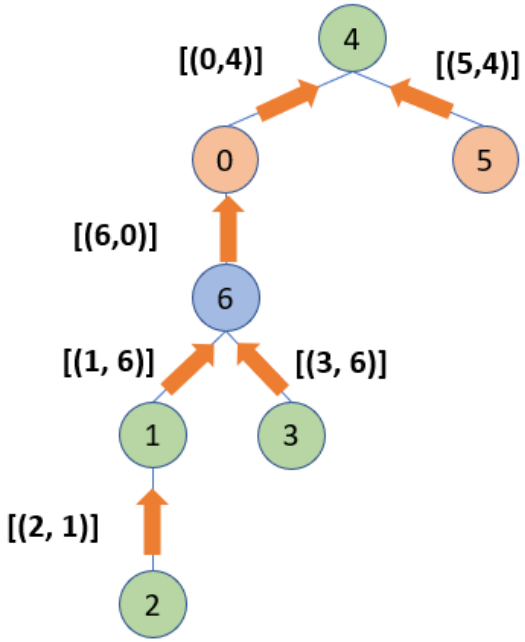


$\mathbf{h}_G$

Molecular vector



JT.nn Encoder output



$\mathbf{h}_{\mathcal{T}_G}$

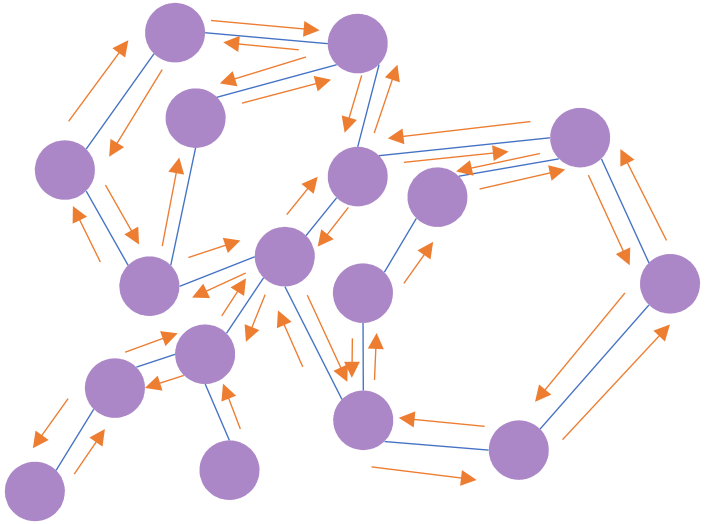
Tree vector

Tree message

$\mathbf{z}_{\mathcal{T}}$

New Tree vector

MPN.nn



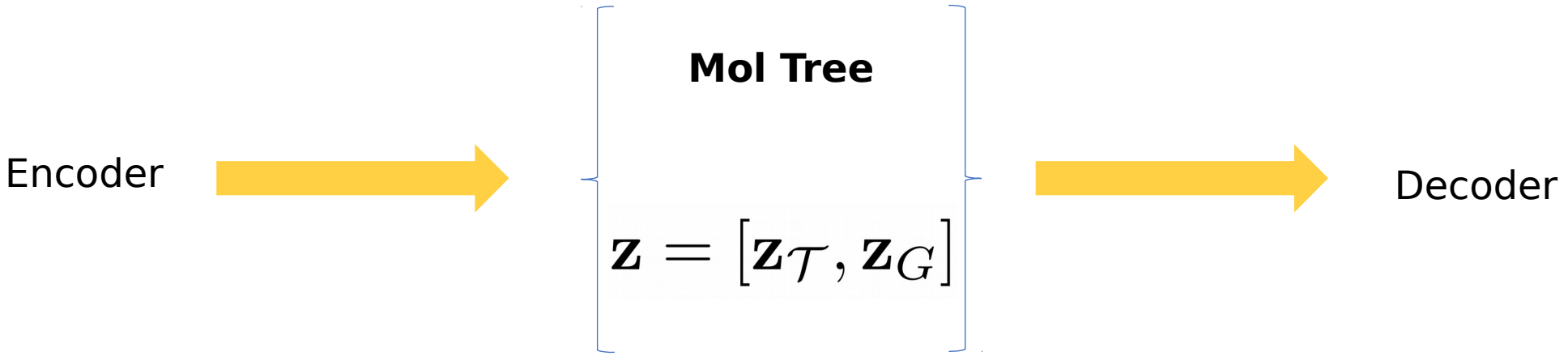
$\mathbf{h}_G$

Molecular vector

$\mathbf{z}_G$

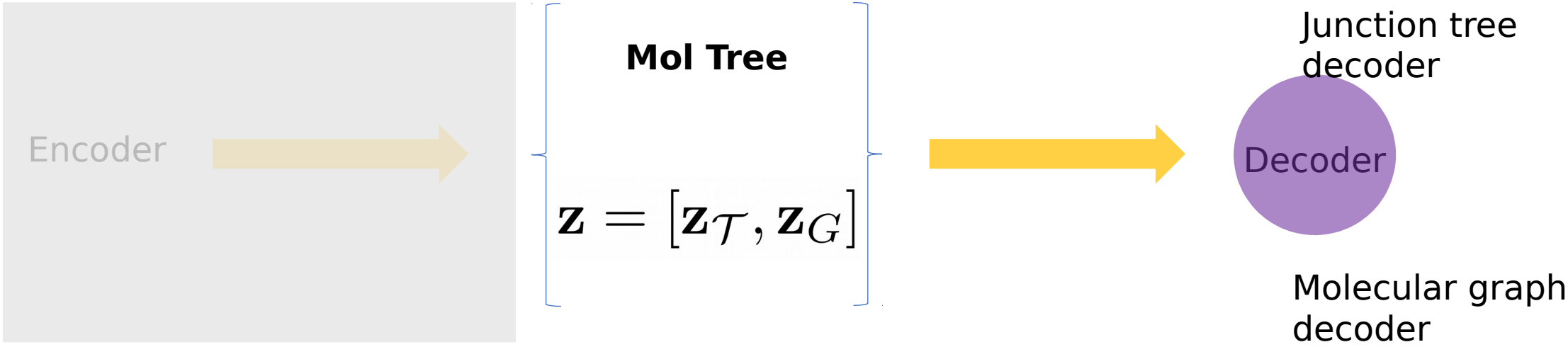
New Molecular vector

$$\mathbf{z} = [\mathbf{z}_{\mathcal{T}}, \mathbf{z}_G]$$



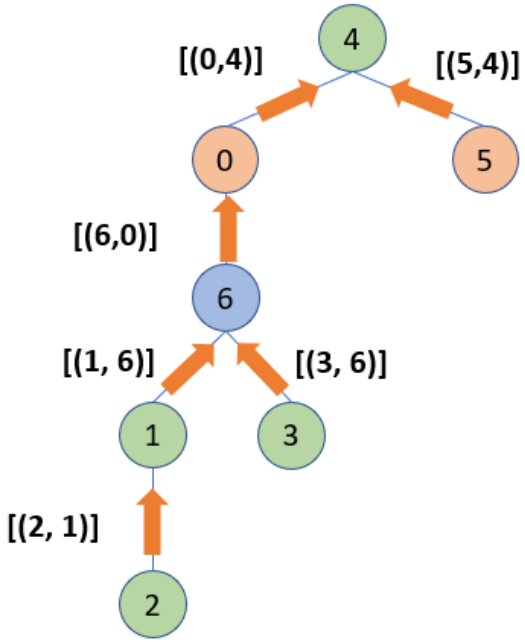

```
loss = word_loss + topo_loss + assm_loss + 2 * stereo_loss + beta * kl_loss + prop_loss
```

Beta is set to zero during training




```
loss = word_loss + topo_loss + assm_loss + 2 * stereo_loss + beta * kl_loss + prop_loss
```

JT.nn Encoder output



$\mathbf{h}_{\mathcal{T}_G}$

Tree vector **Tree message**

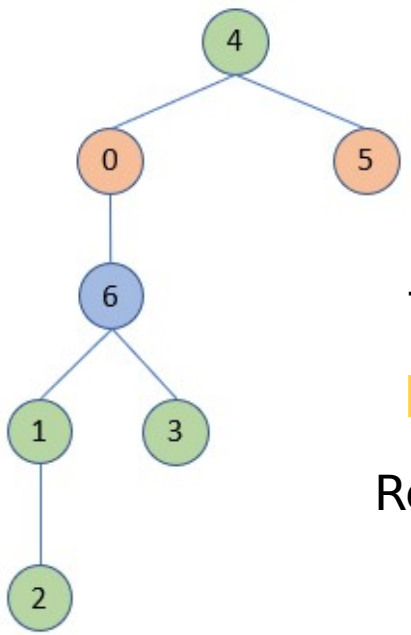
$\mathbf{z}_{\mathcal{T}}$

New Tree vector

Junction tree decoder

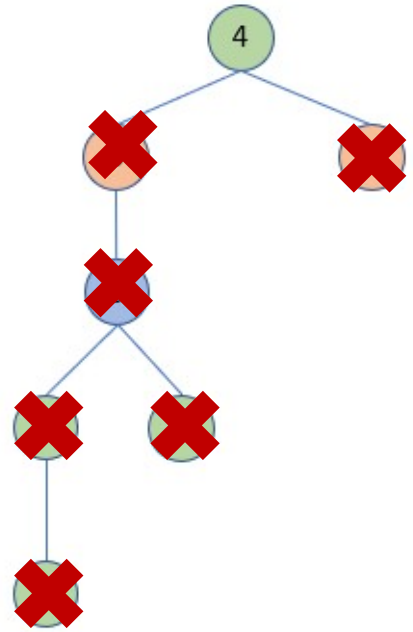
Molecular graph decoder

Tree Decoder



Tree Traversal

Remove neighbors



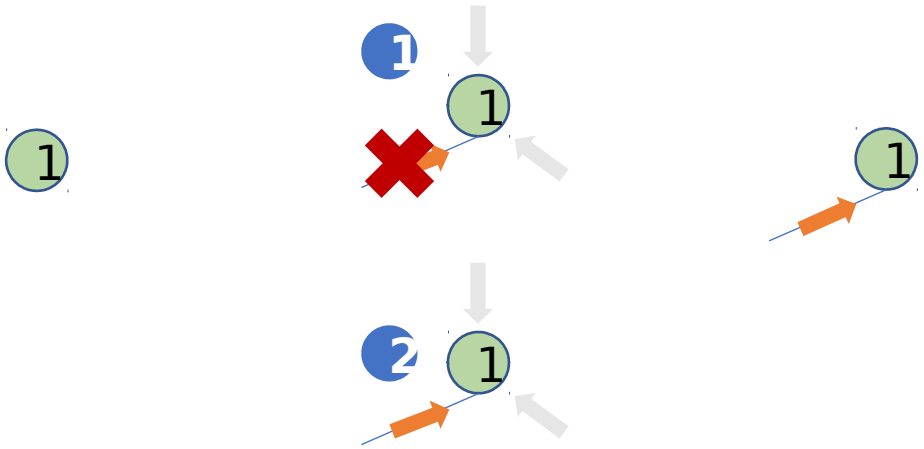
```
[(4, 0, 1), (0, 6, 1), (6, 1, 1), (1, 2, 1),
(2, 1, 0), (1, 6, 0), (6, 3, 1), (3, 6, 0),
(6, 0, 0), (0, 4, 0), (4, 5, 1), (5, 4, 0)]
```

For teacher forcing??

Process is performed for all molecules

Tree Decoder

Mol Tree



root

Calculate message
Get WID

+ embedding

GRU

New message
Add node 1 as
neighbor of 0,
If direction is 1

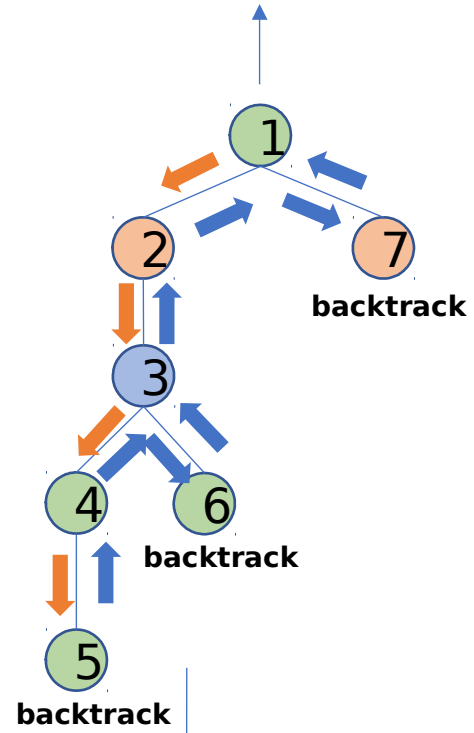
Node
aggregate

Predict current node
Add label
Predict next node
Predict stop

.....

.....

Overall message

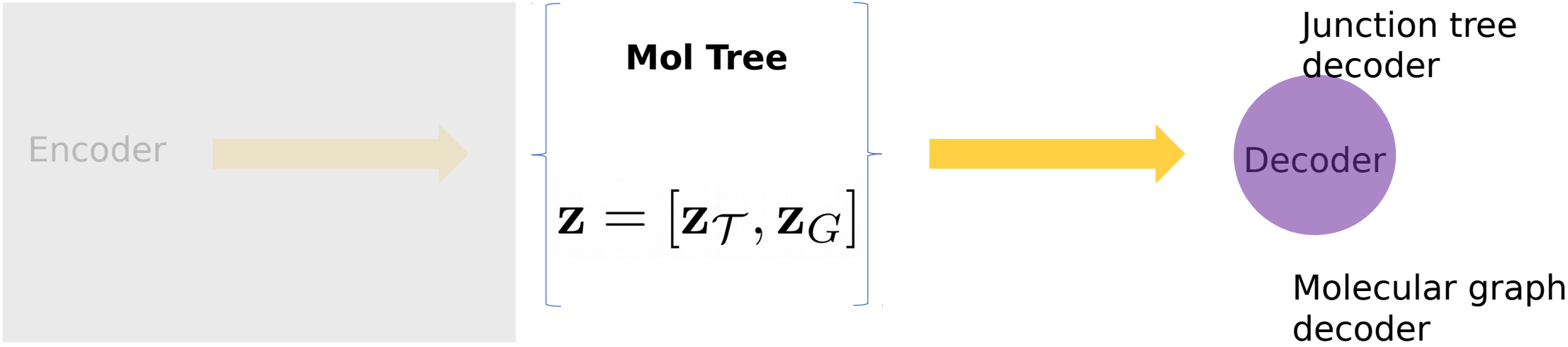


Compare results

$\mathbf{z}_{\mathcal{T}}$

Tree vector

Stop/topological
Loss
Prediction/word
Loss




```
loss = word_loss + topo_loss + assm_loss + 2 * stereo_loss + beta * kl_loss + prop_loss
```

Mol tree **Assemble**

Tree message

New Molecular Vector

$\mathbf{z}_G$

For each node get all the possible combinations with neighbors

candidates, mol tree nodes, original node

Enumerate subgraphs

Score subgraphs

The graph decoder parameters are learned to **maximize the log-likelihood of predicting correct subgraphs Gi of the ground true graph G at each tree node**. where Gi is the set of possible candidate subgraphs at tree node i.

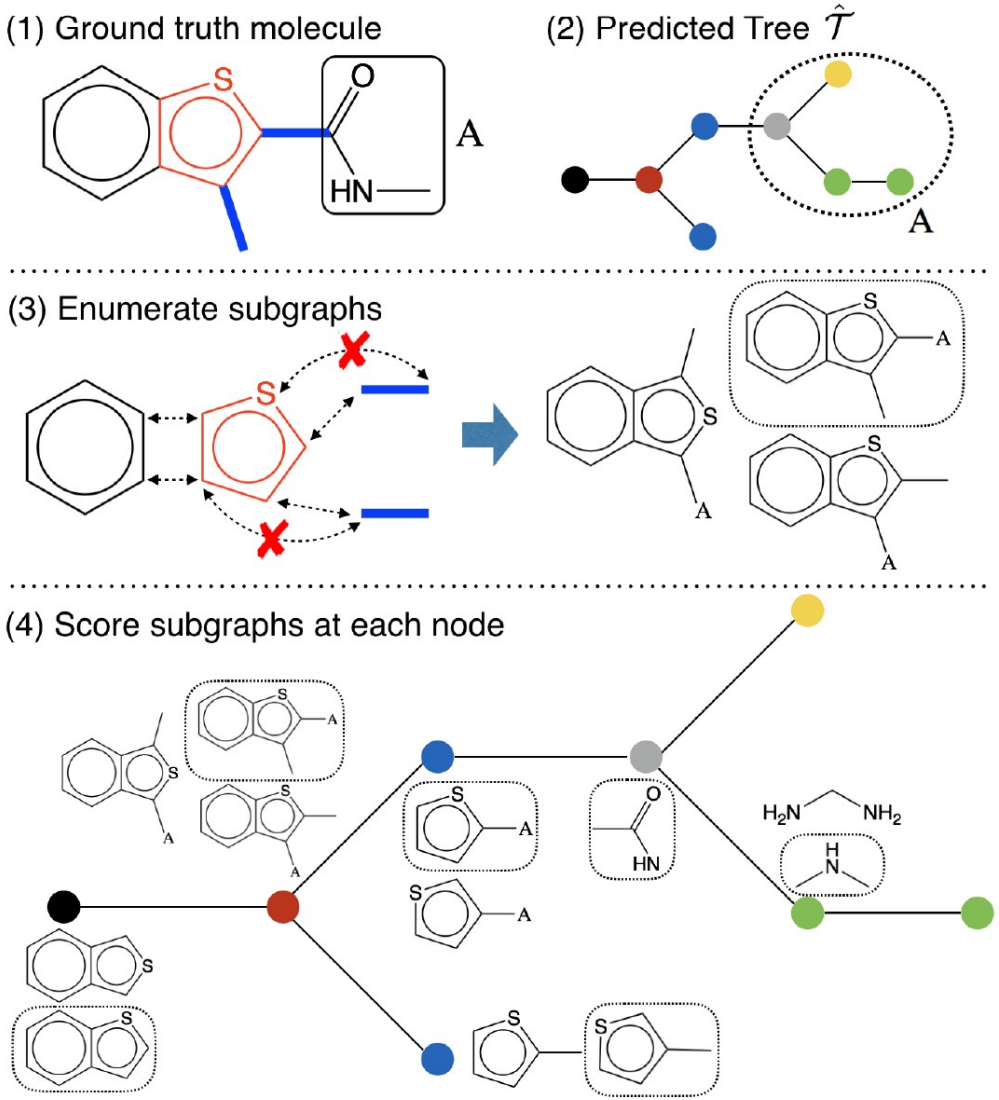
$$\mathcal{L}_g(G) = \sum_i \left[f^a(G_i) - \log \sum_{G_i' \in \mathcal{G}_i} \exp(f^a(G_i')) \right]$$

Subgraph of ground truth

Subgraphs from model

set of possible candidate subgraphs

ground truth



arXiv:1802.04364v2

$$\boldsymbol{\mu}_{uv}^{(t)} = \tau(\mathbf{W}_1^a \mathbf{x}_u + \mathbf{W}_2^a \mathbf{x}_{uv} + \mathbf{W}_3^a \widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)})$$

Atoms of subgraph u-v bond (edge)

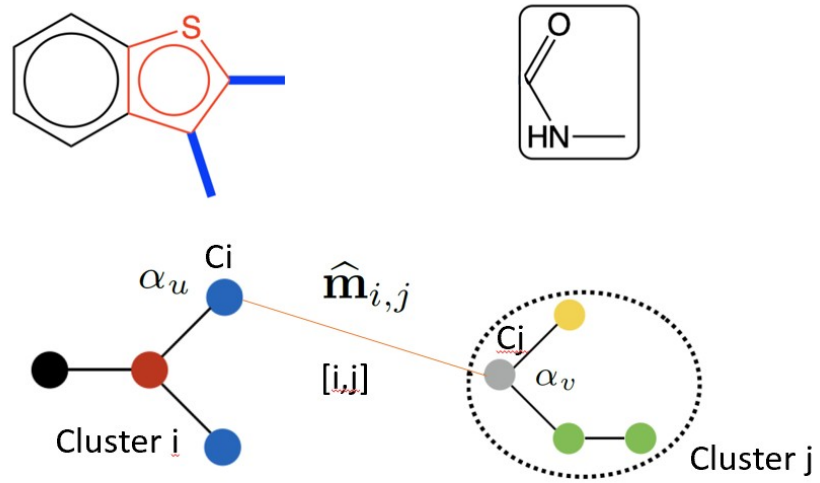
Atom Features

Bond Features

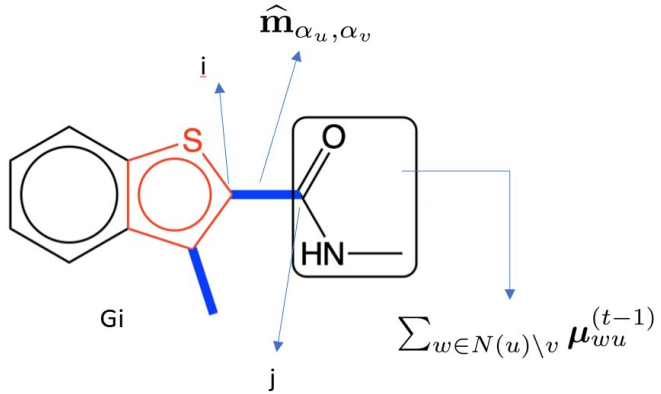
Old message from neighbors

$$\widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)} = \begin{cases} \sum_{w \in N(u) \setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u = \alpha_v \\ \widehat{\mathbf{m}}_{\alpha_u, \alpha_v} + \sum_{w \in N(u) \setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u \neq \alpha_v \end{cases}$$

Tree message



$\alpha_v = i$ if $v \in C_i$ and $\alpha_v = j$ if $v \in C_j \setminus C_i$



arXiv:1802.04364v2

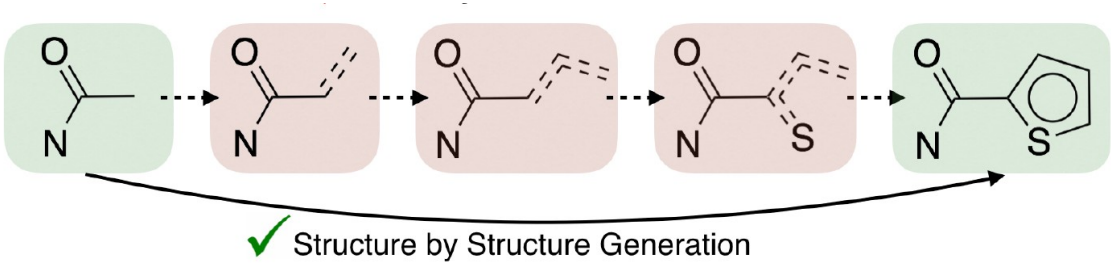
Conclusion:

New approach: interpret each molecule as having been built from subgraphs chosen out of a vocabulary of valid components.

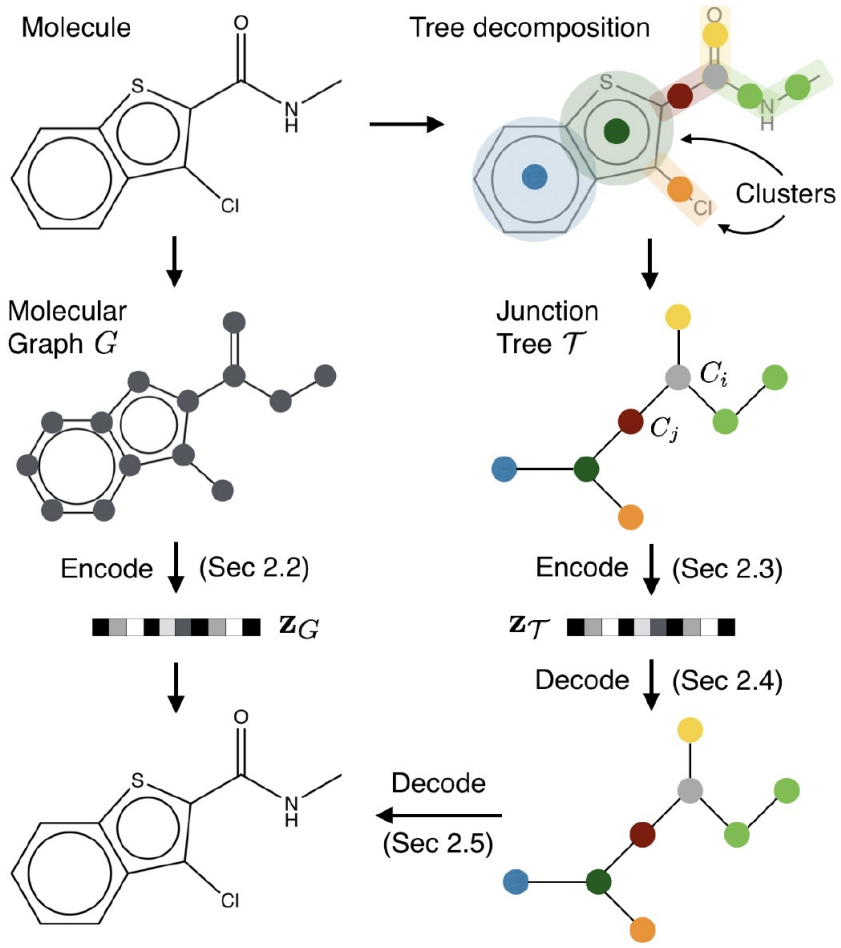
These components are used as building blocks both when encoding a molecule into a vector representation as well as when decoding latent vectors back into valid molecular graphs.

Key advantage: the decoder can realize a valid molecule piece by piece by utilizing the collection of valid components and how they interact, rather than trying to build the molecule atom by atom through chemically invalid intermediaries.

Key contribution: direct realization of molecular graphs, a task previously approached by generating linear SMILES



Method	Reconstruction	Validity
CVAE	44.6%	0.7%
GVAE	53.7%	7.2%
SD-VAE[2]	76.2%	43.5%
GraphVAE	-	13.5%
JT-VAE	**76.7%**	**100.0%**


Molecule
Tree decomposition
Clusters
Molecular Graph G
Junction Tree $\mathcal{T}$
C_i
C_j
Encode (Sec 2.2)
Encode (Sec 2.3)
$\mathbf{z}_G$
$\mathbf{z}_\mathcal{T}$
Decode (Sec 2.4)
Decode (Sec 2.5)

Experiments

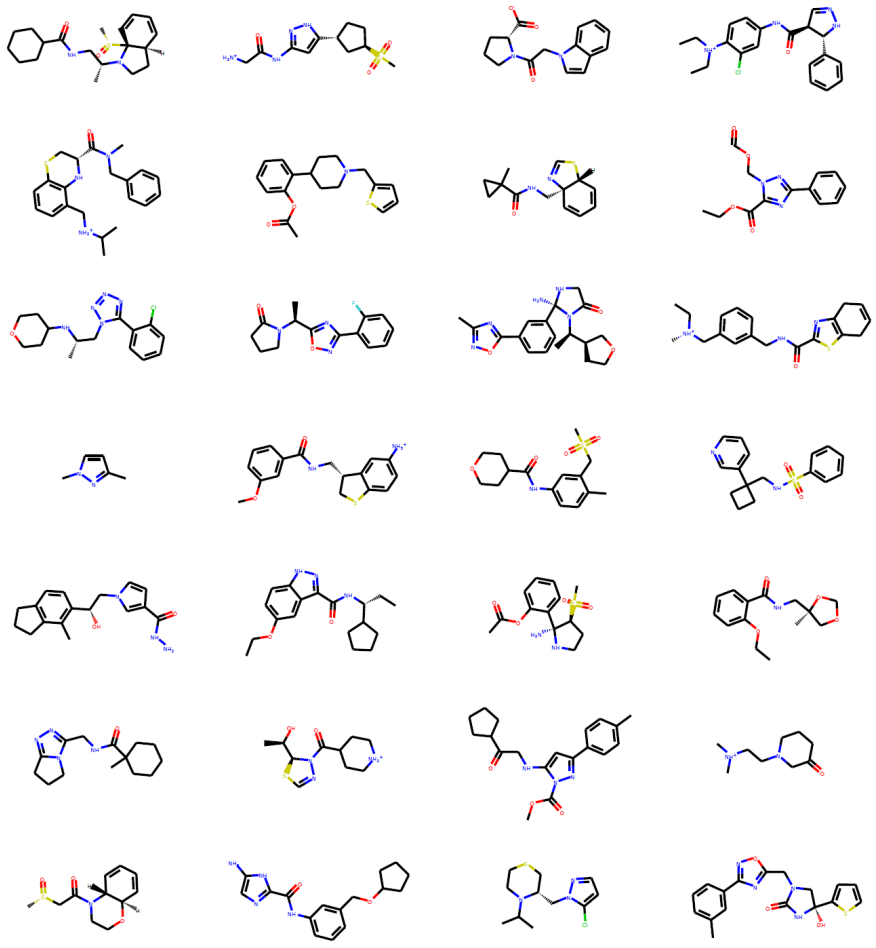
olecule Reconstruction and Validity

The first task is to reconstruct and sample molecules from latent space. Since both encoding and decoding process are stochastic, estimate reconstruction accuracy was done by **Monte Carlo method** used in (Kusner et al., 2017):

Each molecule is encoded 10 times and each encoding is decoded 10 times. The portion of the 100 decoded molecules that are identical to the input molecule are reported. For a fair comparison, two molecules were defined as identical if they have the same SMILES string. At testing time, all generated graphs were converted to SMILES using RDKit.

To compute validity, we sample 1000 latent vectors from the prior distribution N(0; I), and decode each of these vectors

Method	Reconstruction	Validity
CVAE	44.6%	0.7%
GVAE	53.7%	7.2%
SD-VAE[2]	76.2%	43.5%
GraphVAE	-	13.5%
JT-VAE	**76.7%**	**100.0%**



Bayesian Optimization

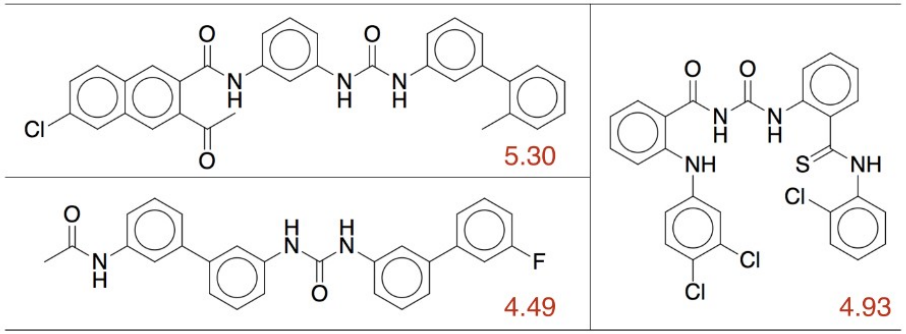
The second task is to produce novel molecules with desired properties. Following (Kusner et al., 2017), target chemical property y(.) is octanol-water partition coefficients (logP) penalized by the synthetic accessibility (SA) score and number of long cycles.

To perform Bayesian optimization (BO), first a VAE was trained and each molecule was associate ed with a latent vector, given by the mean of the variational encoding distribution.

After the VAE is learned, a **sparse Gaussian process (SGP)** was trained to predict y(m) given its latent representation. Then five iterations of batched BO were performed using the expected improvement heuristic.

JT-VAE finds over 50 molecules with

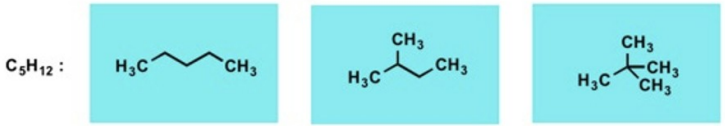
Method	1st	2nd	3rd
CVAE	1.98	1.42	1.19
GVAE	2.94	2.89	2.80
SD-VAE	4.04	3.50	2.96
JT-VAE	**5.30**	**4.93**	**4.49**



Constrained Optimization not discussed

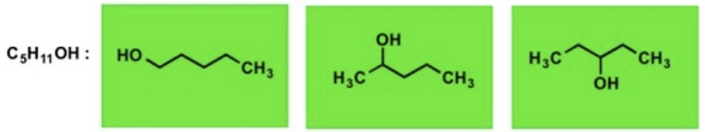
Supporting Slides

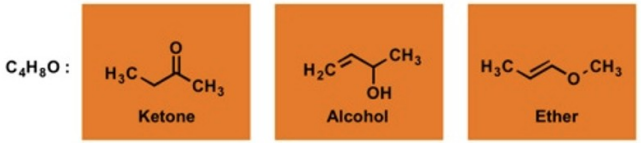
Why is Carbon Important?



Chain Isomerism

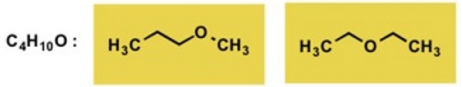
Position Isomerism

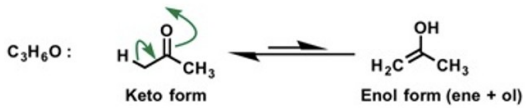




Functional Isomerism

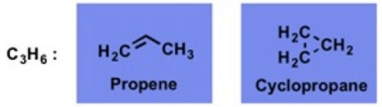
Metamerism

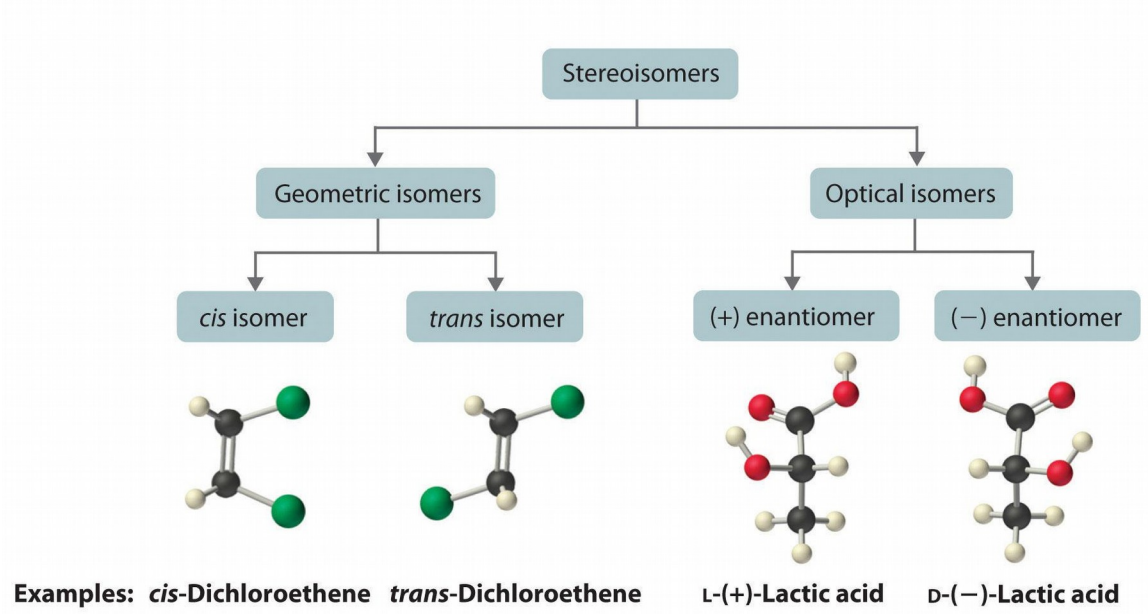


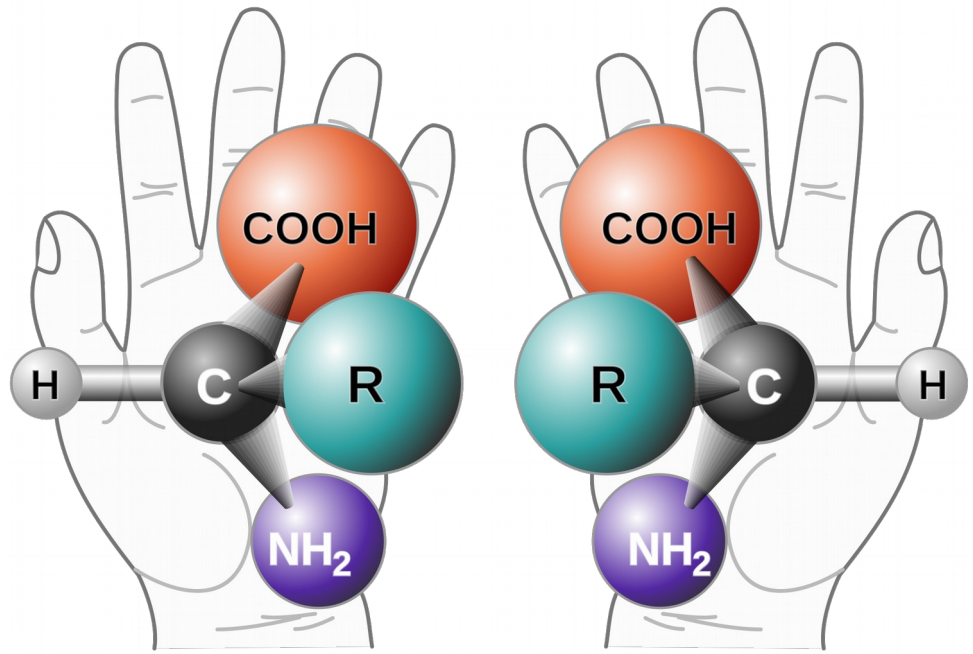


Tautomerism

Ring-Chain isomerism






COOH
H
C
R
NH2
COOH
R
C
H
NH2

atent Representation calculations

```
tree_mean = self.T_mean(tree_vec)
tree_log_var = -torch.abs(self.T_var(tree_vec)) #Following Mueller et al.
mol_mean = self.G_mean(mol_vec)
mol_log_var = -torch.abs(self.G_var(mol_vec)) #Following Mueller et al.

z_mean = torch.cat([tree_mean,mol_mean], dim=1)
z_log_var = torch.cat([tree_log_var,mol_log_var], dim=1)
kl_loss = -0.5 * torch.sum(1.0 + z_log_var - z_mean * z_mean - torch.exp(z_log_var)) / batch_size

epsilon = create_var(torch.randn(batch_size, self.latent_size / 2), False)
tree_vec = tree_mean + torch.exp(tree_log_var / 2) * epsilon
epsilon = create_var(torch.randn(batch_size, self.latent_size / 2), False)
mol_vec = mol_mean + torch.exp(mol_log_var / 2) * epsilon
```

```
self.embedding = nn.Embedding(vocab.size(), hidden_size)
self.jtnn = JTNNEncoder(vocab, hidden_size, self.embedding)
self.jtmpn = JTMPN(hidden_size, depth)
self.mpn = MPN(hidden_size, depth)
self.decoder = JTNNDecoder(vocab, hidden_size, latent_size / 2, self.embedding)

self.T_mean = nn.Linear(hidden_size, latent_size / 2)
self.T_var = nn.Linear(hidden_size, latent_size / 2)
self.G_mean = nn.Linear(hidden_size, latent_size / 2)
self.G_var = nn.Linear(hidden_size, latent_size / 2)

self.propNN = nn.Sequential(
        nn.Linear(self.latent_size, self.hidden_size),
        nn.Tanh(),
        nn.Linear(self.hidden_size, 1)
)
self.prop_loss = nn.MSELoss()
self.assm_loss = nn.CrossEntropyLoss(size_average=False)
self.stereo_loss = nn.CrossEntropyLoss(size_average=False)
```

Pre-train

```
mkdir pre_model/
CUDA_VISIBLE_DEVICES=0 python pretrain.py --train ../data/train.txt --vocab ../data/vocab.txt \
--hidden 450 --depth 3 --latent 56 --batch 40 \
--save_dir pre_model/
```

Train

```
mkdir vae_model/
CUDA_VISIBLE_DEVICES=0 python vaetrain.py --train ../data/train.txt --vocab ../data/vocab.txt \
--hidden 450 --depth 3 --latent 56 --batch 40 --lr 0.0007 --beta 0.005 \
--model pre_model/model.2 --save_dir vae_model/
```

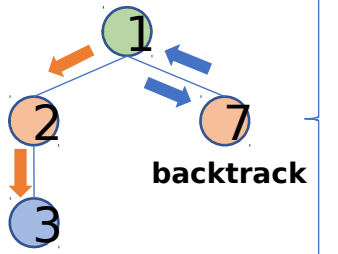
Sampling

Random Tensors

Molecular Vector

Tree Vector

Sampling Decoder



All chemically compatible options from vocab

. . .

Tree Vector

Predict WID for root

Get chemical properties of the root

Create message for root

Convert WID to embedding

```
#Predict stop
cur_h = cur_h_nei.sum(dim=1)
stop_hidden = torch.cat([cur_x,cur_h,mol_vec], dim=1)
stop_hidden = nn.ReLU()(self.U(stop_hidden))
stop_score = nn.Sigmoid()(self.U_s(stop_hidden) * 20).squeeze()
```

Predict Stop

Yes

No

Backtrack GRU

New Message

$$p_t = \sigma(\mathbf{u}^d \cdot \tau(\mathbf{W}_1^d \mathbf{x}_{i_t} + \mathbf{W}_2^d \mathbf{z}_{\mathcal{T}} + \mathbf{W}_3^d \sum_{(k,i_t) \in \tilde{\mathcal{E}}_t} \mathbf{h}_{k,i_t}))$$

Softmax

ReLU

Mol Vec

New Message

Passed to multinomial and gives all chemically compatible candidates

Check chemical compatibility and add as new node

Root and all nodes

Graph Decoder

Root and all nodes → **JT.nn Encoder** → **Tree message**

JT.nn Encoder has dual purpose

Not Deterministic

Assemble molecule

↓

JT.MPN.nn

↓

Predict JT

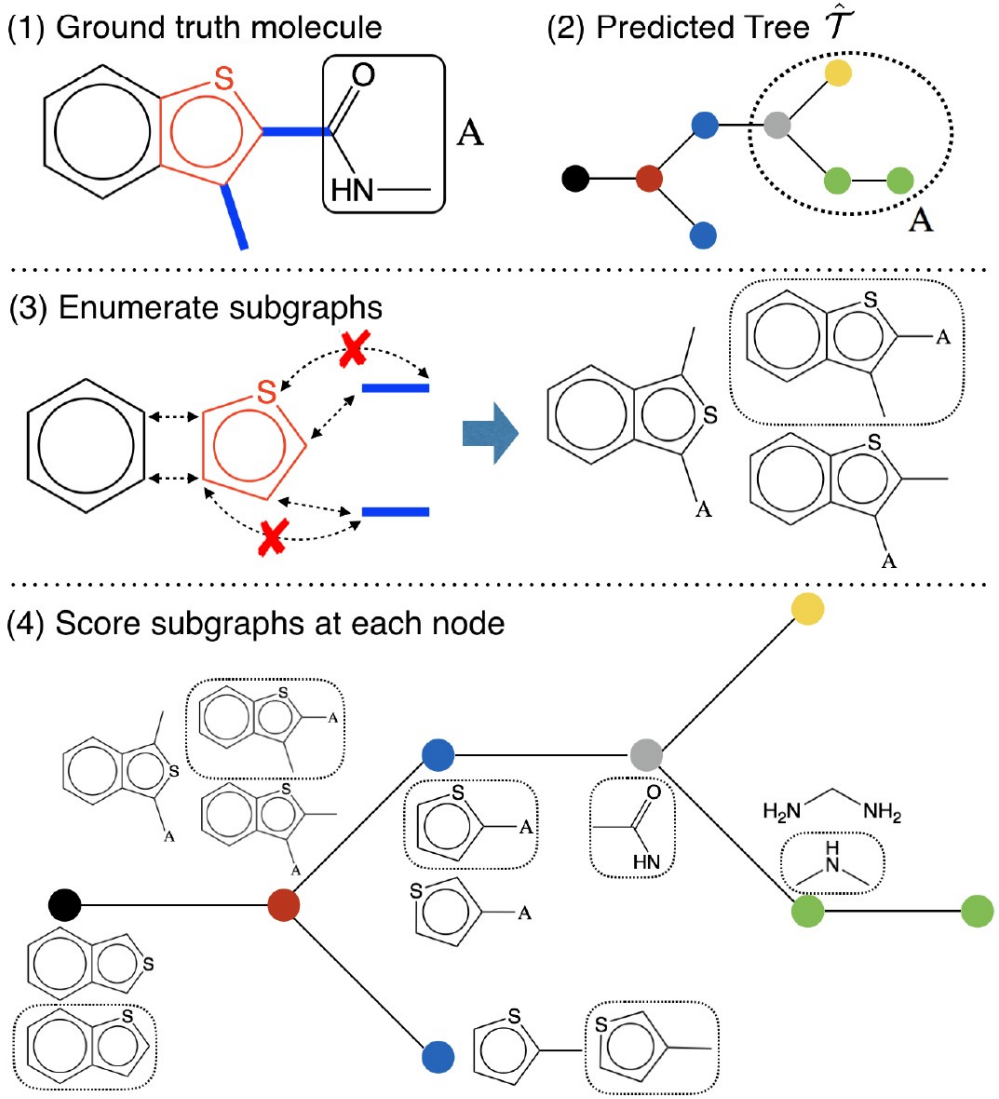
↓

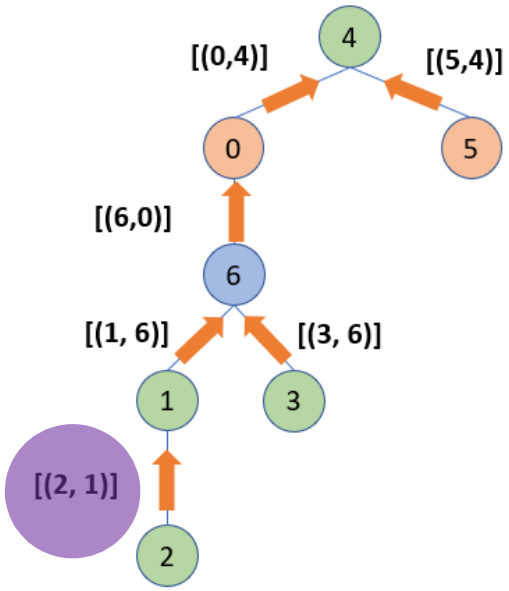
Score candidates → **Return molecule with best stereo score**

$$\hat{G} = \arg \max_{G' \in \mathcal{G}(\hat{\mathcal{T}})} f^a(G')$$

↓

Scoring Function





Bottom-up shown only

Calculate message

8 vectors, one for each neighbor with values available, the rest are padded with zeros

Size=420

```
tensor([[ 0.1217,  0.4944,  0.1524,  ...,  0.8717,  0.0364,  0.6448],
        [ 0.8932,  0.4646,  0.3229,  ...,  0.0195,  0.4559,  0.1710],
        [ 0.4896,  0.5342,  0.7170,  ...,  0.4839,  0.4389,  0.3686],
        ...,
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]])
```

Batch Size=40

Size=420

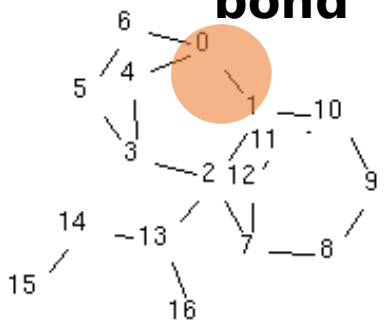
Max neighbors=8

```
tensor([[[ 0.0182,  0.2862,  0.5036,  ...,  0.9948,  0.4333,  0.1643],
         [ 0.5298,  0.6082,  0.3990,  ...,  0.7159,  0.4312,  0.8723],
         [ 0.7814,  0.4285,  0.5161,  ...,  0.2744,  0.0597,  0.8197],
         ...,
         [ 0.0197,  0.1079,  0.5523,  ...,  0.0964,  0.1815,  0.3792],
         [ 0.4990,  0.0072,  0.3741,  ...,  0.9504,  0.3556,  0.5845],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.6733,  0.7312,  0.0722,  ...,  0.4355,  0.9517,  0.4943],
         [ 0.9745,  0.1447,  0.4472,  ...,  0.0343,  0.4236,  0.8479],
         [ 0.4690,  0.3002,  0.2390,  ...,  0.8961,  0.4806,  0.0154],
         ...,
         [ 0.4629,  0.2799,  0.6524,  ...,  0.3698,  0.5789,  0.1225],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        [[ 0.1217,  0.4944,  0.1524,  ...,  0.8717,  0.0364,  0.6448],
         [ 0.8932,  0.4646,  0.3229,  ...,  0.0195,  0.4559,  0.1710],
         [ 0.4896,  0.5342,  0.7170,  ...,  0.4839,  0.4389,  0.3686],
         ...,
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]],

        ...,

torch.Size([40, 8, 420])
```

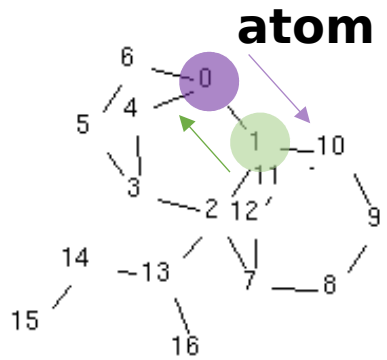
Combine results for all compounds

Graph Encoder – Bond Features

bond



atom



Atom Features (fatoms)

torch.Size([1,11])

```
0 1
0 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
1 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
```

bond Features

torch.Size([1,39])

```
tensor([ 1.,                                                  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,
         0.,  0.,  0.])
```

concatenate

torch.Size([1,50])

```
0 tensor([ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
           0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
           0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,
           0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,
           0.,  0.])
```

```
0 1
0 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
1 tensor([ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.])
```

Perform for all atoms

(fbonds)

torch.Size([1601, 50])

20 bonds per molecule, 2 atoms per molecule, 40 tensors per molecule

Overall Bond Feature (fbonds)

It remains to be specified how each neighborhood realization is scored. Let G_i be the subgraph resulting from a particular merging of cluster C_i in the tree with its neighbors C_j, $j \in N_{\widehat{\mathcal{T}}}(i)$. We score G_i as a candidate subgraph by first deriving a vector representation $\mathbf{h}_{G_i}$ and then using $f_i^a(G_i) = \mathbf{h}_{G_i} \cdot \mathbf{z}_G$ as the subgraph score. To this end, let u, v specify atoms in the candidate subgraph G_i and let $\alpha_v = i$ if $v \in C_i$ and $\alpha_v = j$ if $v \in C_j \setminus C_i$. The indices α_v are used to mark the position of the atoms in the junction tree, and to retrieve messages $\widehat{\mathbf{m}}_{i,j}$ summarizing the subtree under i along the edge (i, j) obtained by running the tree encoding algorithm. The neural messages pertaining to the atoms and bonds in subgraph G_i are obtained and aggregated into $\mathbf{h}_{G_i}$, similarly to the encoding step, but with different (learned) parameters:

$$\boldsymbol{\mu}_{uv}^{(t)} = \tau(\mathbf{W}_1^a \mathbf{x}_u + \mathbf{W}_2^a \mathbf{x}_{uv} + \mathbf{W}_3^a \widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)}) \tag{15}$$

$$\widetilde{\boldsymbol{\mu}}_{uv}^{(t-1)} = \begin{cases} \sum_{w \in N(u) \setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u = \alpha_v \\ \widehat{\mathbf{m}}_{\alpha_u, \alpha_v} + \sum_{w \in N(u) \setminus v} \boldsymbol{\mu}_{wu}^{(t-1)} & \alpha_u \neq \alpha_v \end{cases}$$

Loss calculations

```
loss = word_loss + topo_loss + assm_loss + 2 * stereo_loss + beta * kl_loss + prop_loss
```

Assembly loss

Mol tree

Tree message

New Molecular Vector

Stereochemical loss

Mol tree

New Molecular Vector

prop loss

New Tree Vector

New Molecular Vector

Just in case

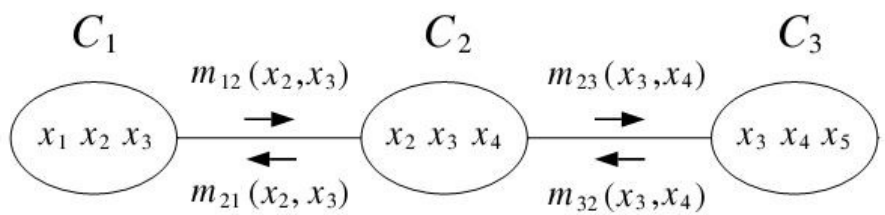
Junction Tree Algorithm

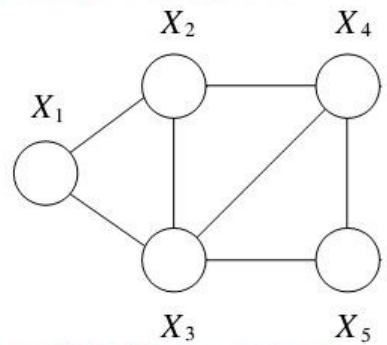
3. Construct the junction tree

one node for every maximal *clique*

form maximal spanning tree of cliques

clique tree is a junction tree if for every pair of cliques V and W, then all cliques on the (unique) path between V and W contain V∩W

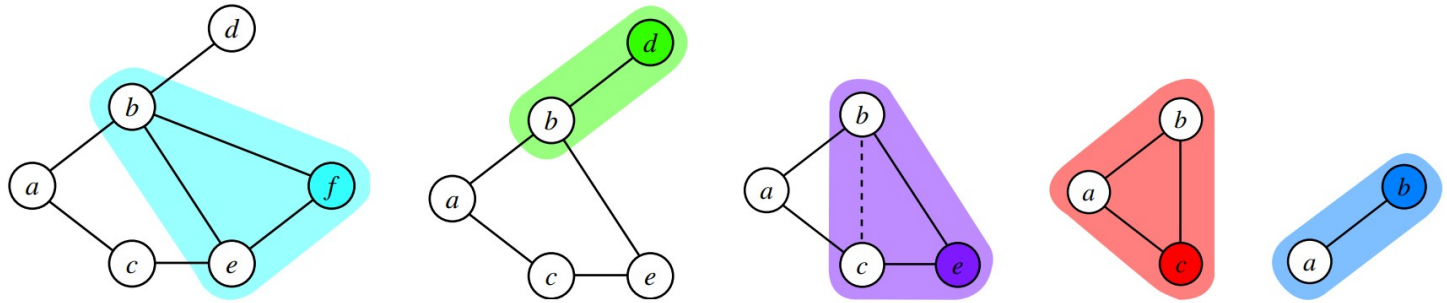




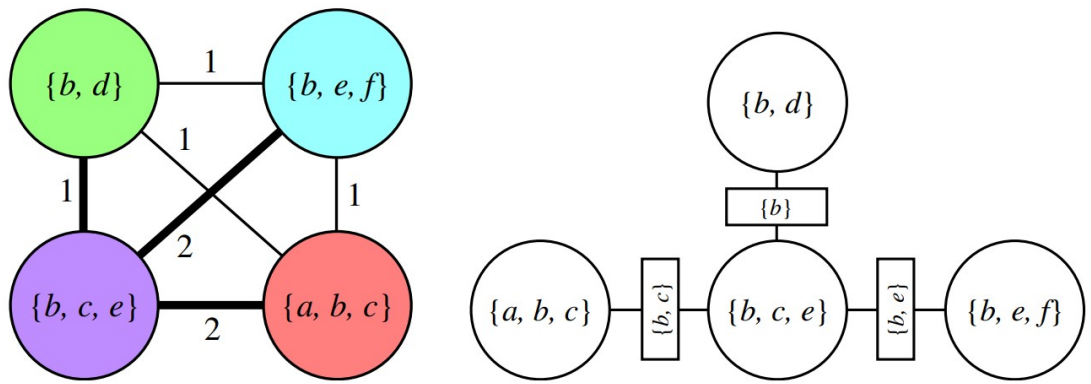
If this property holds, then local propagation of information will lead to global consistency.

An example of building junction trees

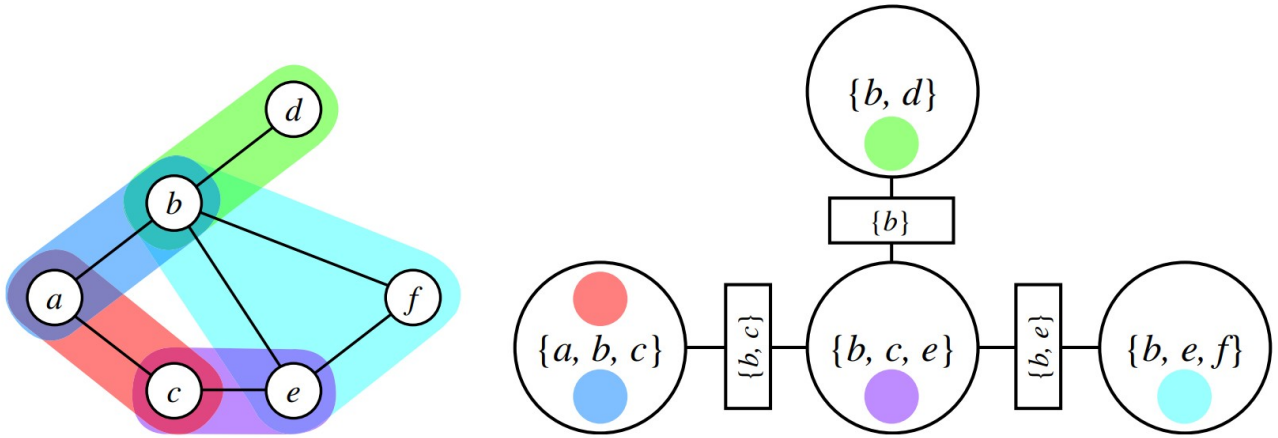
1. Compute the elimination cliques (the order here is f, d, e, c, b, a).



2. Form the complete cluster graph over the maximal elimination cliques and find a maximum-weight spanning tree.



Junction trees

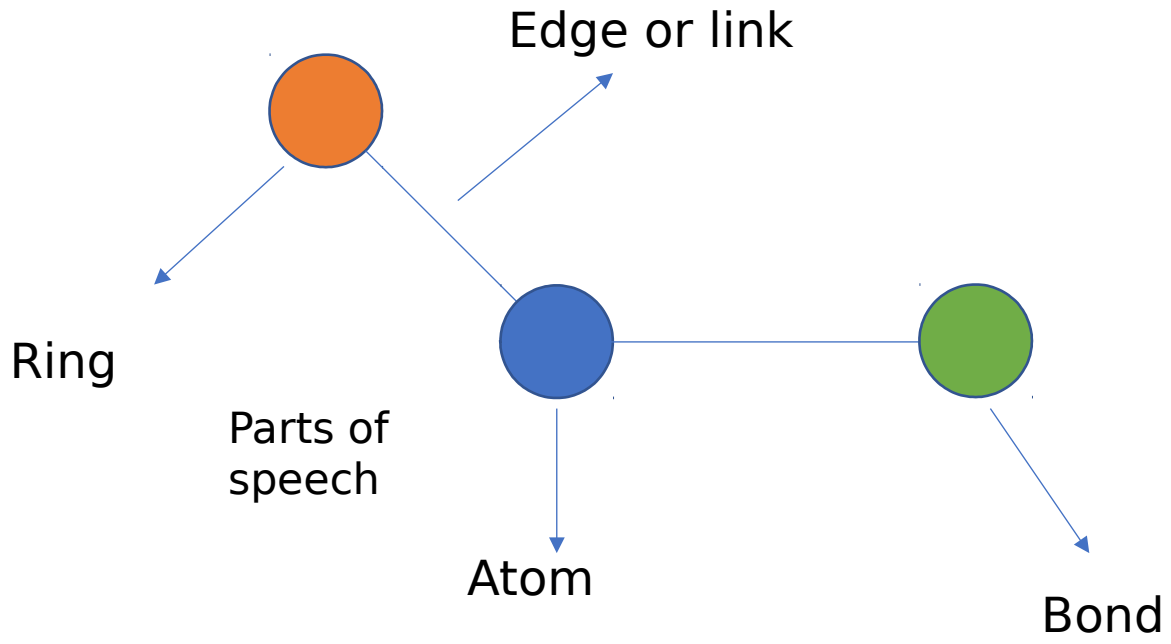


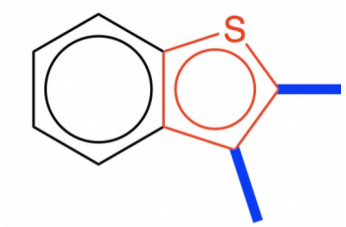
G T

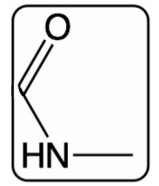
A cluster graph T is a **junction tree** for G if it has these three properties:

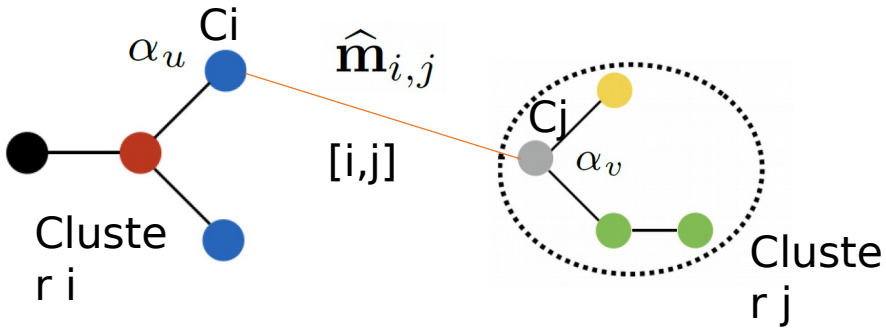
1. **singly connected**: there is exactly one path between each pair of clusters.
2. **covering**: for each clique A of G there is some cluster C such that $A \subseteq C$.
3. **running intersection**: for each pair of clusters B and C that contain i, each cluster on the unique path between B and C also contains i.

Other

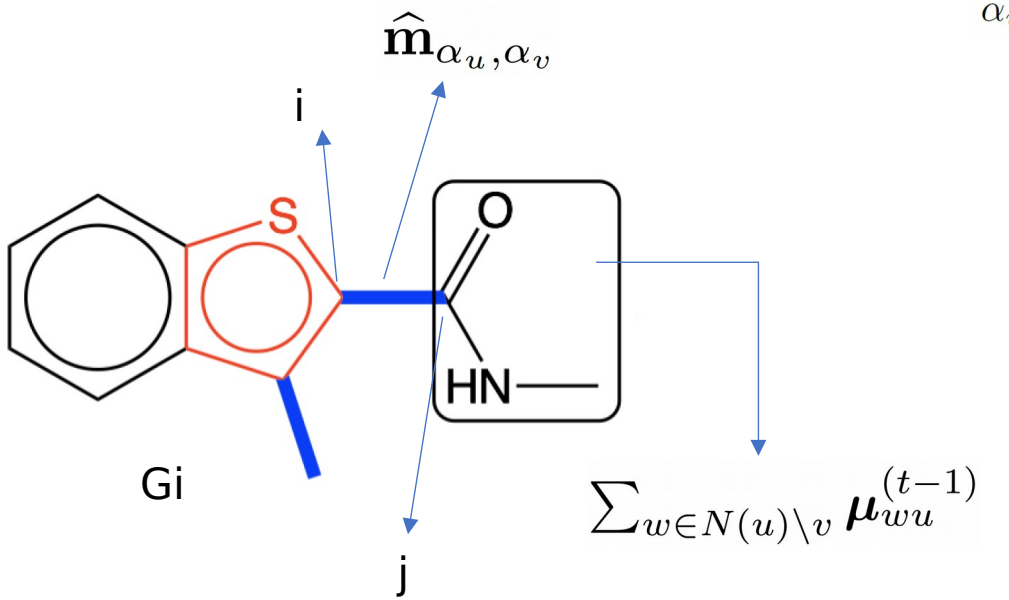

Edge or link
Ring
Parts of
speech
Atom
Bond



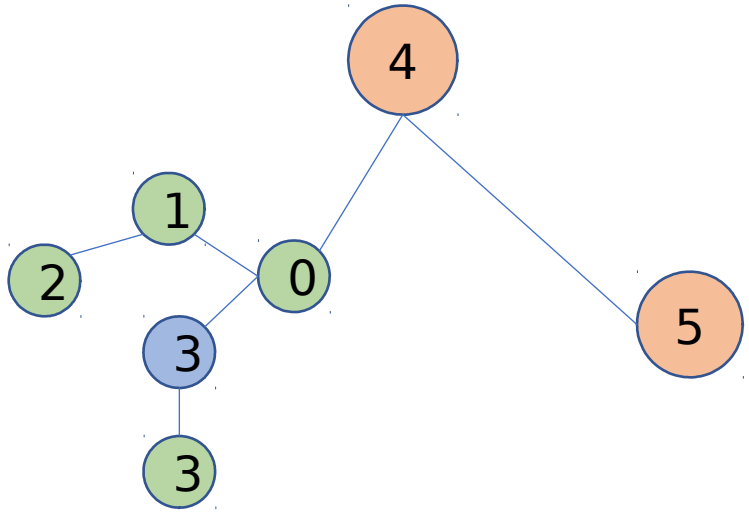




$\alpha_v = i$ if $v \in C_i$ and $\alpha_v = j$ if $v \in C_j \setminus C_i$

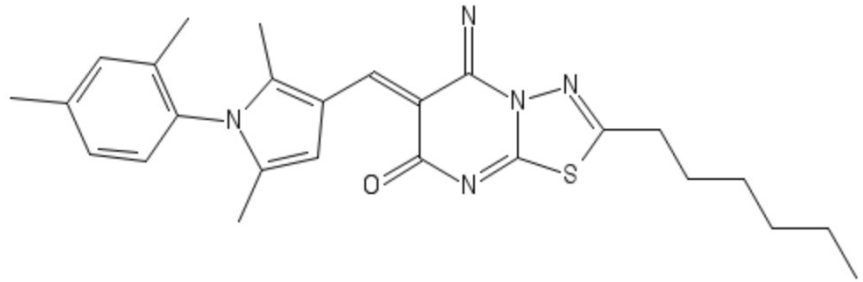


arXiv:1802.04364v2



MILES: **S**implified **M**olecular-**i**nput **l**ine-**e**ntry **s**ystem

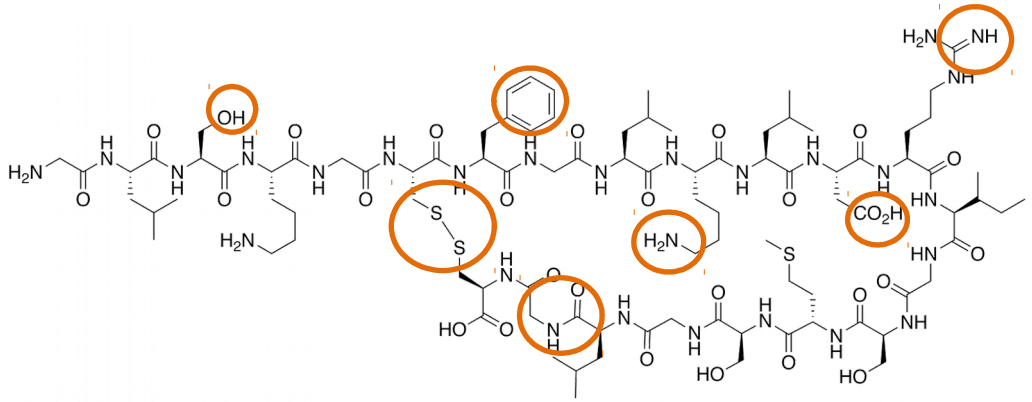
```
CCCCCCC1=NN2C(=N)/C(=C\c3cc(C)n(-c4ccc(C)cc4C)c3C)C(=O)N=C2S1
COCC[C@@H](C)C(=O)N(C)Cc1ccc(O)cc1
C=CCn1c(S[C@H](C)c2nc3sc(C)c(C)c3c(=O)[nH]2)nnc1C1CC1
C[NH+](C/C=C/c1ccco1)CCC(F)(F)F
COc1ccc(N2C(=O)C(=O)N(CN3CCC(c4nc5ccccc5s4)CC3)C2=O)cc1
Cc1ccc([C@@H](C)[NH2+][C@H](C)C(=O)Nc2ccccc2F)cc1
O=c1cc(C[NH2+]Cc2cccc(Cl)c2)nc(N2CCCC2)[nH]1
O=C(Cn1nc(C(=O)[O-])c2ccccc2c1=O)Nc1ccc2c(c1)C(=O)c1ccccc1C2=O
O=C(Nc1cccc(S(=O)(=O)N2CCCCC2)c1)c1cc(F)c(F)cc1Cl
CC(C)Cc1nnc(NC(=O)C(=O)NCCC2CCCCC2)s1
C[C@H](NC(=O)[C@@H](O)c1ccccc1)c1nnc2ccccn12
O=S(=O)(Nc1cc(F)ccc1F)c1ccc(Cl)cc1F
```



The aim is to automate the design of molecules based on specific chemical properties

Why is Carbon Important?

"What we normally think of as 'life' is based on chains of carbon atoms, with a few other atoms. Carbon has the richest chemistry." Stephen Hawking



A highly important and very unusual property of carbon is its ability to form long chains. There is almost no limit to the size and shape of molecules that can be made with carbon atoms.

Carbon atoms can share not only a single electron with another atom to form a single bond, but it can also share two or three electrons, forming a double or triple bond.

The same collection of atoms and bonds, but in a different arrangement within the molecule, makes a molecule with a different shape and hence different properties.