Resonance-aware substructure searching with RDKit

Paolo Tosco

Outline

> Background

> Description of the algorithm

> Results

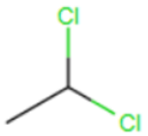
> Overview of the API

cresset

Background

Background

```
In [1]: import sys
        import rdkit
        from rdkit import Chem
        from rdkit.Chem import rdmolops
        from rdkit.Chem.Draw import IPythonConsole

In [2]: mol = Chem.MolFromSmiles('CC(Cl)Cl')

In [3]: mol

Out[3]:
```




```
In [4]: query = Chem.MolFromSmarts('C(Cl)Cl')
        mol.GetSubstructMatches(query, uniquify = False)

Out[4]: ((1, 2, 3), (1, 3, 2))
```

Due to the graph isomorphism, if we set `uniquify` to `False` we get two matches for the `C(Cl)Cl` substructure

Applications (1)

In [1]:
```
import sys
import rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem.Draw import IPythonConsole
IPythonConsole.ipython_3d = True
```

In [2]:
```
mol = Chem.MolFromSmiles('CC(Cl)Cl')
```

In [3]:
```
mol = AllChem.AddHs(mol, addCoords = True)
```

In [4]:
```
AllChem.EmbedMolecule(mol)
AllChem.MMFFOptimizeMolecule(mol)
```

Out[4]:
```
0
```

In [5]:
```
mol2 = Chem.Mol(mol)
```

In [6]:
```
mol2.GetConformer().SetAtomPosition(2, mol.GetConformer().GetAtomPosition(3))
mol2.GetConformer().SetAtomPosition(3, mol.GetConformer().GetAtomPosition(2))
```

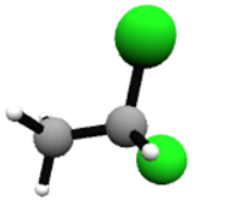
In [7]:
```
w = Chem.SDWriter('dichloroethane.sdf')
w.write(mol)
w.write(mol2)
w.close()
```

Let’s prepare two isomorphic dichloroethanes exchanging 3D coordinates between the chlorine atoms

Applications (2)

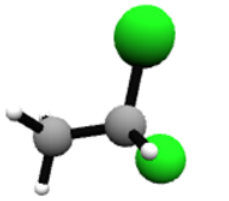
```
In [8]: mol
```

Out[8]:



```
In [9]: mol2
```

Out[9]:



A symmetry-aware calculation between these two molecules should yield RMSD = 0.0, whereas the RMSD obtained through a standard calculation would be >0.0

Symmetry-aware RMSD calculation (1)

In [1]:
```
import sys
import copy
import math
import rdkit
from rdkit import Chem
from rdkit.Chem import rdmolfiles, rdMolAlign
from rdkit.Chem.Draw import IPythonConsole
```

In [2]:
```
def saveConformer(mol, confId):
    conf = mol.GetConformer(confId)
    confCopy = Chem.Conformer(conf.GetNumAtoms())
    for i in range(conf.GetNumAtoms()):
        confCopy.SetAtomPosition(i, conf.GetAtomPosition(i))
    return confCopy
```

In [3]:
```
def loadConformer(mol, confId, confCopy):
    conf = mol.GetConformer(confId)
    for i in range(conf.GetNumAtoms()):
        conf.SetAtomPosition(i, confCopy.GetAtomPosition(i))
```

A few days ago on the RDKit mailing list there was a request for such a tool to compare docking poses

Symmetry-aware RMSD calculation (2)

In [4]:
```
def symmUniqueFit(prbMol, refMol, prbConfId = -1, refConfId = -1):
    matches = refMol.GetSubstructMatches(prbMol, uniquify = False)
    if (not matches):
        raise ValueError('mols don\'t match')
    amaps = [list(enumerate(match)) for match in matches]
    first = True
    rmsdMin = 0.0
    amapBest = []
    for amap in amaps:
        conf = saveConformer(prbMol, prbConfId)
        rmsd = Chem.rdMolAlign.AlignMol(prbMol, refMol,
            prbConfId, refConfId, atomMap = amap)
        loadConformer(prbMol, prbConfId, conf)
        if (first or (rmsd < rmsdMin)):
            first = False
            rmsdMin = rmsd
            amapBest = amap
    return amapBest
```

The core of this script is the `symmUniqueFit()` function, which uses `GetSubstructMatches()` to find graph isomorphisms, and returns the `atomMap` which yields the lowest RMSD

Symmetry-aware RMSD calculation (3)

In [5]:
```
def getRmsdImmobile(prbMol, refMol,
    prbConfId = -1, refConfId = -1, atomMap = None):
    refConf = refMol.GetConformer(refConfId)
    prbConf = prbMol.GetConformer(prbConfId)
    if (not atomMap):
        atomMap = []
        for i in range(0, refMol.GetNumAtoms()):
            if (refMol.GetAtomWithIdx(i).GetAtomicNum() == 1):
                continue
            atomMap.append((i, i))
    sqDist = 0.0
    for pair in atomMap:
        sqDist += (prbConf.GetAtomPosition(pair[0]) \
            - refConf.GetAtomPosition(pair[1])).LengthSq()
    sqDist /= float(len(atomMap))
    return math.sqrt(sqDist)
```

In [6]:
```
suppl = Chem.SDMolSupplier('dichloroethane.sdf')
refMol = suppl[0]
prbMol = suppl[1]
amap = None
amap = symmUniqueFit(refMol, prbMol)
if (len(amap) != refMol.GetNumAtoms()):
    raise ValueError('atomMap does not contain all atoms')
rmsd = getRmsdImmobile(prbMol, refMol, atomMap = amap)
sys.stdout.write('{0:.2f}\n'.format(rmsd))
```

```
0.00
```

As expected, RMSD = 0.00 Å

cresset

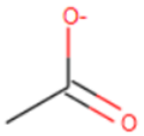
What about acetate, then? (1)

In [1]:
```
import sys
import rdkit
from rdkit import Chem
from rdkit.Chem import rdmolops
from rdkit.Chem.Draw import IPythonConsole
```

In [2]:
```
mol = Chem.MolFromSmiles('CC(=O)[O-]')
```

In [3]:
```
mol
```

Out[3]:



In [4]:
```
query = Chem.MolFromSmarts('C(=O)[O-]')
mol.GetSubstructMatches(query, uniquify = False)
```

Out[4]:
```
((1, 2, 3),)
```

Acetate is not isomorphic anymore, because of the localized formal charge and double bond

What about acetate, then? (2)

In [1]:
```
import sys
import rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem.Draw import IPythonConsole
IPythonConsole.ipython_3d = True
```

In [2]:
```
mol = Chem.MolFromSmiles('CC(=O)[O-]')
```

In [3]:
```
mol = AllChem.AddHs(mol, addCoords = True)
```

In [4]:
```
AllChem.EmbedMolecule(mol)
AllChem.MMFFOptimizeMolecule(mol)
```

Out[4]:
```
0
```

In [5]:
```
mol2 = Chem.Mol(mol)
```

In [6]:
```
mol2.GetConformer().SetAtomPosition(2, mol.GetConformer().GetAtomPosition(3))
mol2.GetConformer().SetAtomPosition(3, mol.GetConformer().GetAtomPosition(2))
```

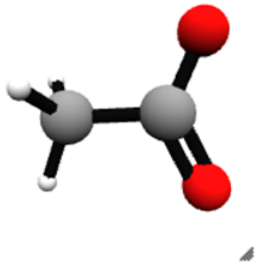
In [7]:
```
w = Chem.SDWriter('acetate.sdf')
w.write(mol)
w.write(mol2)
w.close()
```

Let’s do the coordinate flip between the two oxygen atoms

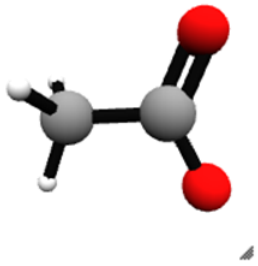
cresset

Something unpleasant is going to happen...

```
In [8]: mol
Out[8]:
```



```
In [9]: mol2
Out[9]:
```



When we try to compute a symmetry-aware RMSD between these two molecules...

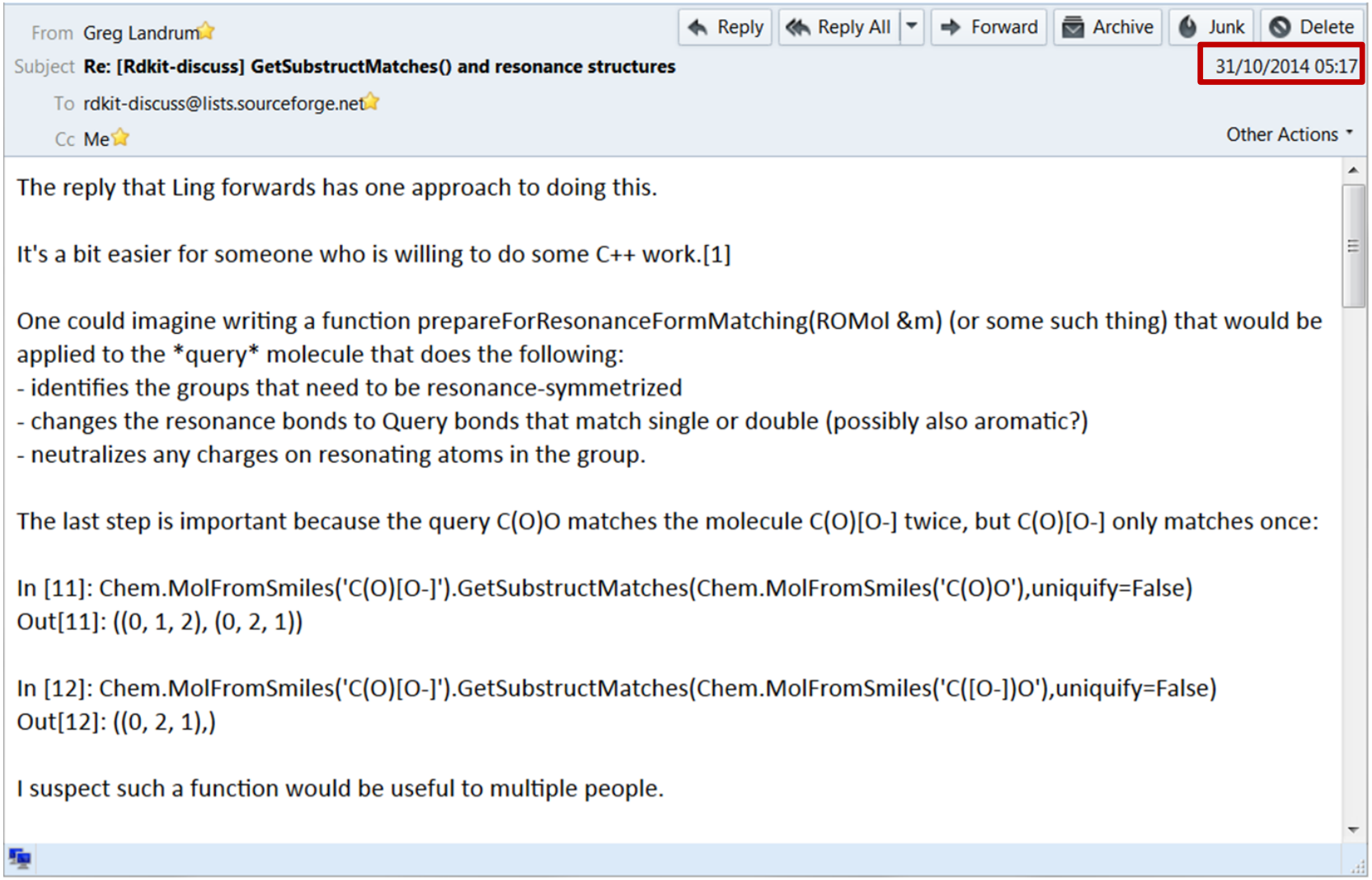
...and here it is

```
In [8]: suppl = Chem.SDMolSupplier('acetate.sdf')
        refMol = suppl[0]
        prbMol = suppl[1]
        amap = None
        amap = symmUniqueFit(refMol, prbMol)
        if (len(amap) != refMol.GetNumAtoms()):
            raise ValueError('atomMap does not contain all atoms')
        rmsd = getRmsdImmobile(prbMol, refMol, atomMap = amap)
        sys.stdout.write('{0:.4f}\n'.format(rmsd))

        1.6178
```

...since the carboxylate group is not perceived as isomorphic, the RMSD is much higher than expected

This issue involves all conjugated groups where a formal charge can be delocalized on multiple atoms

Solutions?



From Greg Landrum
Subject **Re: [Rdkit-discuss] GetSubstructMatches() and resonance structures**
To rdkit-discuss@lists.sourceforge.net
Cc Me

Reply | Reply All | Forward | Archive | Junk | Delete | Other Actions

31/10/2014 05:17

The reply that Ling forwards has one approach to doing this.

It's a bit easier for someone who is willing to do some C++ work.[1]

One could imagine writing a function prepareForResonanceFormMatching(ROMol &m) (or some such thing) that would be applied to the *query* molecule that does the following:
- identifies the groups that need to be resonance-symmetrized
- changes the resonance bonds to Query bonds that match single or double (possibly also aromatic?)
- neutralizes any charges on resonating atoms in the group.

The last step is important because the query C(O)O matches the molecule C(O)[O-] twice, but C(O)[O-] only matches once:

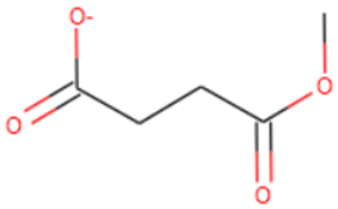
In [11]: Chem.MolFromSmiles('C(O)[O-]').GetSubstructMatches(Chem.MolFromSmiles('C(O)O'),uniquify=False)
Out[11]: ((0, 1, 2), (0, 2, 1))

In [12]: Chem.MolFromSmiles('C(O)[O-]').GetSubstructMatches(Chem.MolFromSmiles('C([O-])O'),uniquify=False)
Out[12]: ((0, 2, 1),)

I suspect such a function would be useful to multiple people.

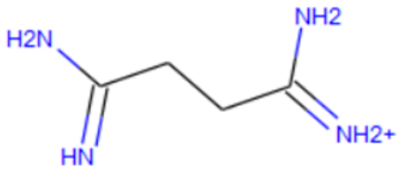
However...

```
In [1]: import sys
        import rdkit
        from rdkit import Chem
        from rdkit.Chem.Draw import IPythonConsole

In [2]: mol = Chem.MolFromSmiles('COC(=O)CCC(=O)[O-]')

In [3]: mol

Out[3]:
```



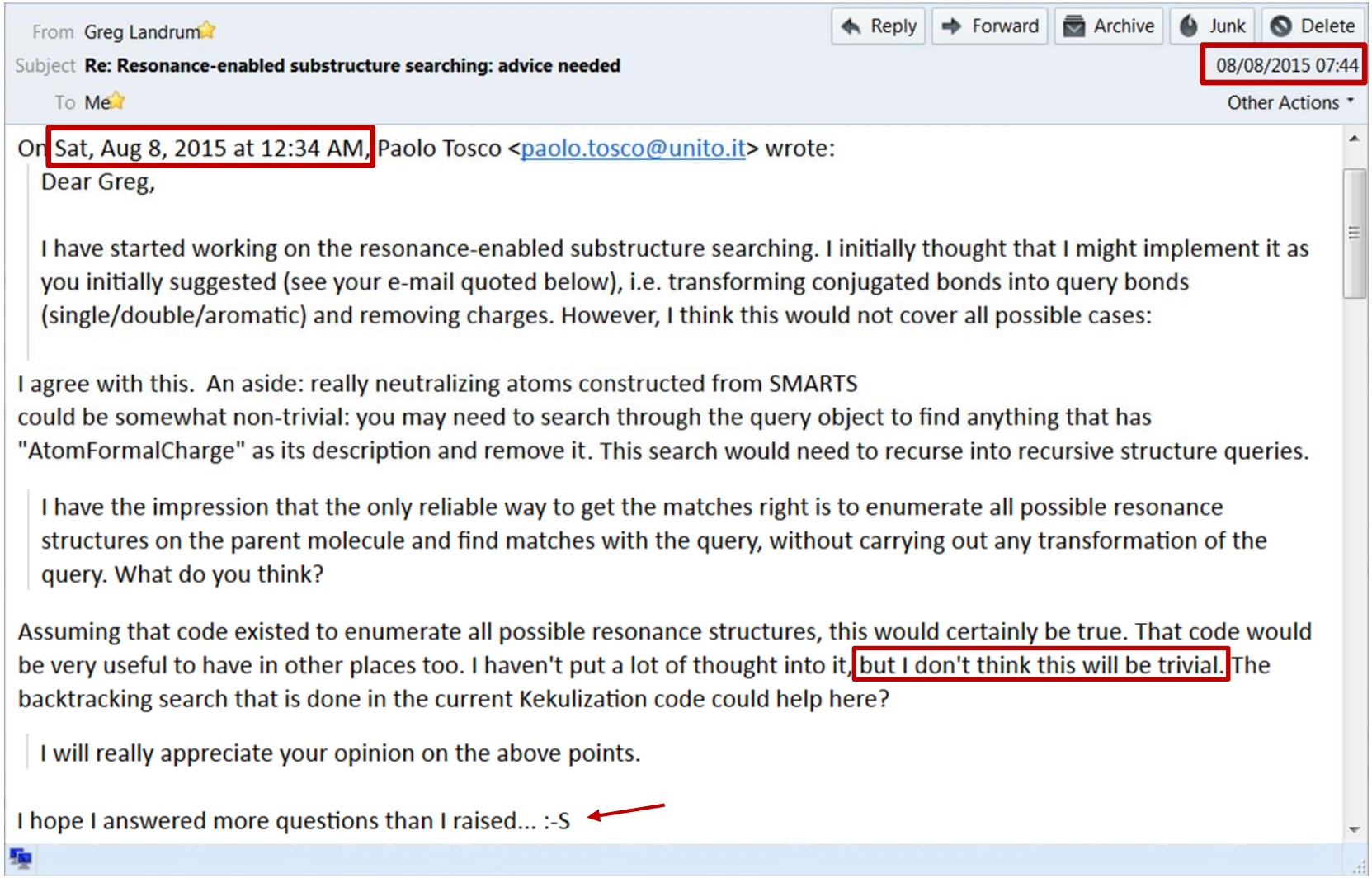
Here a `C(=O)[O-]` query would match `(1, 2, 3), (3, 2, 1)` and `(6, 7, 8), (8, 7, 6),` while it should match only `(6, 7, 8), (8, 7, 6)`

```
In [4]: mol = Chem.MolFromSmiles('NC(=N)CCC(=[NH2+])N')

In [5]: mol

Out[5]:
```



Here a `C(=[NH2+])N` query would match `(1, 0, 2)` and `(5, 6, 7), (5, 7, 6),` while it should match only `(5, 6, 7), (5, 7, 6)`

Well, I'm sure Greg will have a solution to this...



From Greg Landrum
Subject **Re: Resonance-enabled substructure searching: advice needed**
To Me

Reply | Forward | Archive | Junk | Delete

08/08/2015 07:44

Other Actions

On Sat, Aug 8, 2015 at 12:34 AM, Paolo Tosco <paolo.tosco@unito.it> wrote:

> Dear Greg,
>
> I have started working on the resonance-enabled substructure searching. I initially thought that I might implement it as you initially suggested (see your e-mail quoted below), i.e. transforming conjugated bonds into query bonds (single/double/aromatic) and removing charges. However, I think this would not cover all possible cases:

I agree with this. An aside: really neutralizing atoms constructed from SMARTS could be somewhat non-trivial: you may need to search through the query object to find anything that has "AtomFormalCharge" as its description and remove it. This search would need to recurse into recursive structure queries.

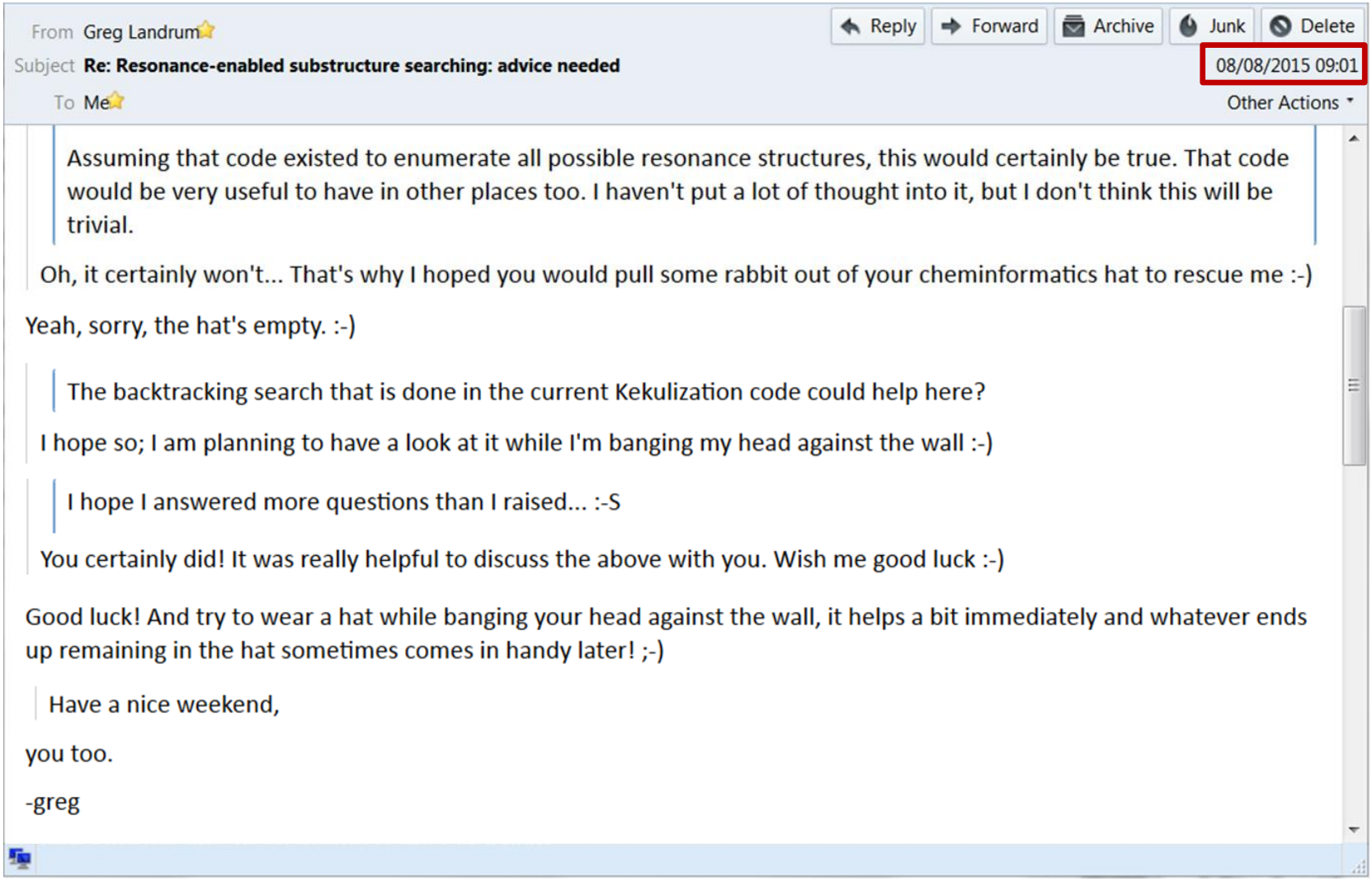
> I have the impression that the only reliable way to get the matches right is to enumerate all possible resonance structures on the parent molecule and find matches with the query, without carrying out any transformation of the query. What do you think?

Assuming that code existed to enumerate all possible resonance structures, this would certainly be true. That code would be very useful to have in other places too. I haven't put a lot of thought into it, but I don't think this will be trivial. The backtracking search that is done in the current Kekulization code could help here?

> I will really appreciate your opinion on the above points.

I hope I answered more questions than I raised... :-S

...won’t he?



From Greg Landrum
Subject **Re: Resonance-enabled substructure searching: advice needed**
To Me

Reply | Forward | Archive | Junk | Delete

08/08/2015 09:01

Other Actions

> Assuming that code existed to enumerate all possible resonance structures, this would certainly be true. That code would be very useful to have in other places too. I haven't put a lot of thought into it, but I don't think this will be trivial.

Oh, it certainly won't... That's why I hoped you would pull some rabbit out of your cheminformatics hat to rescue me :-)

Yeah, sorry, the hat's empty. :-)

> The backtracking search that is done in the current Kekulization code could help here?

I hope so; I am planning to have a look at it while I'm banging my head against the wall :-)

> I hope I answered more questions than I raised... :-S

You certainly did! It was really helpful to discuss the above with you. Wish me good luck :-)

Good luck! And try to wear a hat while banging your head against the wall, it helps a bit immediately and whatever ends up remaining in the hat sometimes comes in handy later! ;-)

> Have a nice weekend,

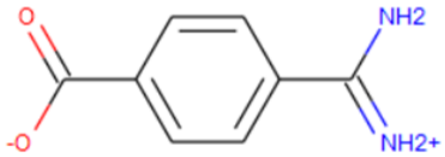
you too.

-greg

Description of the algorithm

Description of the algorithm

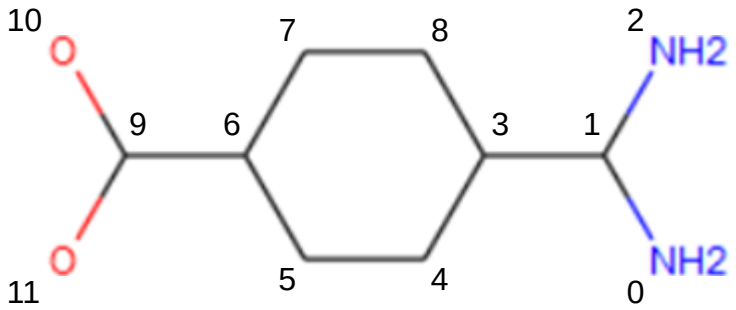
```
In [1]: import sys
        import rdkit
        from rdkit import Chem
        from rdkit.Chem.Draw import IPythonConsole

In [2]: mol = Chem.MolFromSmiles('[NH2+]=C(N)c1ccc(cc1)C(=O)[O-]')

In [3]: mol

Out[3]:
```



To enumerate all stable resonance structures for this molecule, all disconnected conjugated groups in the molecule are identified and enumerated separately

Bond order assignment

- Start from the first conjugate group setting all bonds to single and removing all formal charges
- Push the first heavy atom (0) to the begin stack, then loop until the begin stack and the state stack are empty



Pull an atom from the begin stack (0)

Pick the first non-definitive neighbour of atom 0 (1)

Which bonds can atom 0 accept? (single, double)
Mark atom 0 as definitive

Which bonds can atom 1 accept? (single, double)
Atom 1 is not definitive – push 1 to begin stack

Logical AND

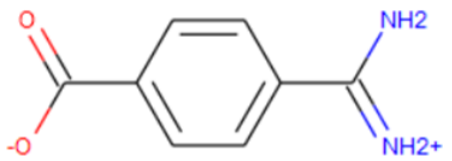
The bond between atoms 0 and 1 can be

Single:
keep looping

Double or triple: push current structure to state stack

Atom/bond fingerprint storage

- Once all bonds have been added, fingerprints are computed for each bond set and converted to a hash
- Depending on whether the `KEKULE_ALL` flag is set, fingerprints are computed using the atom total valences or the bond orders, respectively



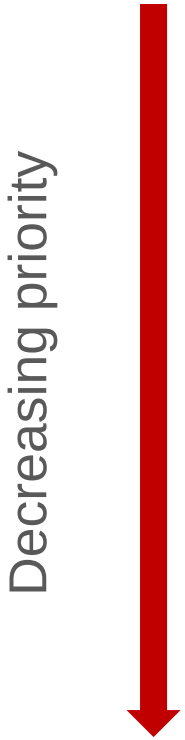
- The hash is used as the key to store the resonance structure in a `boost::unordered_map`
- If `KEKULE_ALL` was not chosen, equivalent Kekulé structures are discarded at an early stage

Assignment of non-bonded electrons

- If not enough non-bonded electrons are available to satisfy all octets, or if the `ALLOW_INCOMPLETE_OCTETS` flag was set, the available electrons are distributed among atoms which need non-bonded electrons, enumerating all possible combinations
- If all octets can be satisfied, non-bonded electrons are assigned univocally to all atoms which need them
- A final check on formal charges is carried out to rule out unlikely resonance structures; the extent to which charge separation is allowed depends on whether the `UNCONSTRAINED_CATIONS` and `UNCONSTRAINED_ANIONS` flags are set

Resonance structure sorting

- Resonance structures are sorted based on the following criteria; the last two are arbitrary and are provided only to make the sort stable



- Number of unsatisfied octets
- Number of formal charges
- Number of formal charges weighted by atom electronegativity
- Distance between formal charges with the same sign
- Distance between formal charges with opposite signs
- Index of the first atom bearing a formal charge
- Index of the first multiple bond

cresset

Re-generate complete molecules

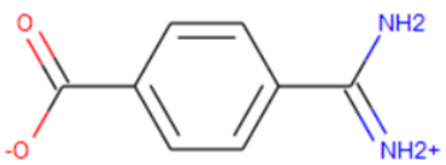
- Once resonance structures have been enumerated and sorted in isolation for each conjugated group, they are combined in all possible permutations to yield resonance structures for the complete molecule

- Resonance structures for the complete molecule can be browsed *via* a `ResonanceMolSupplier` object

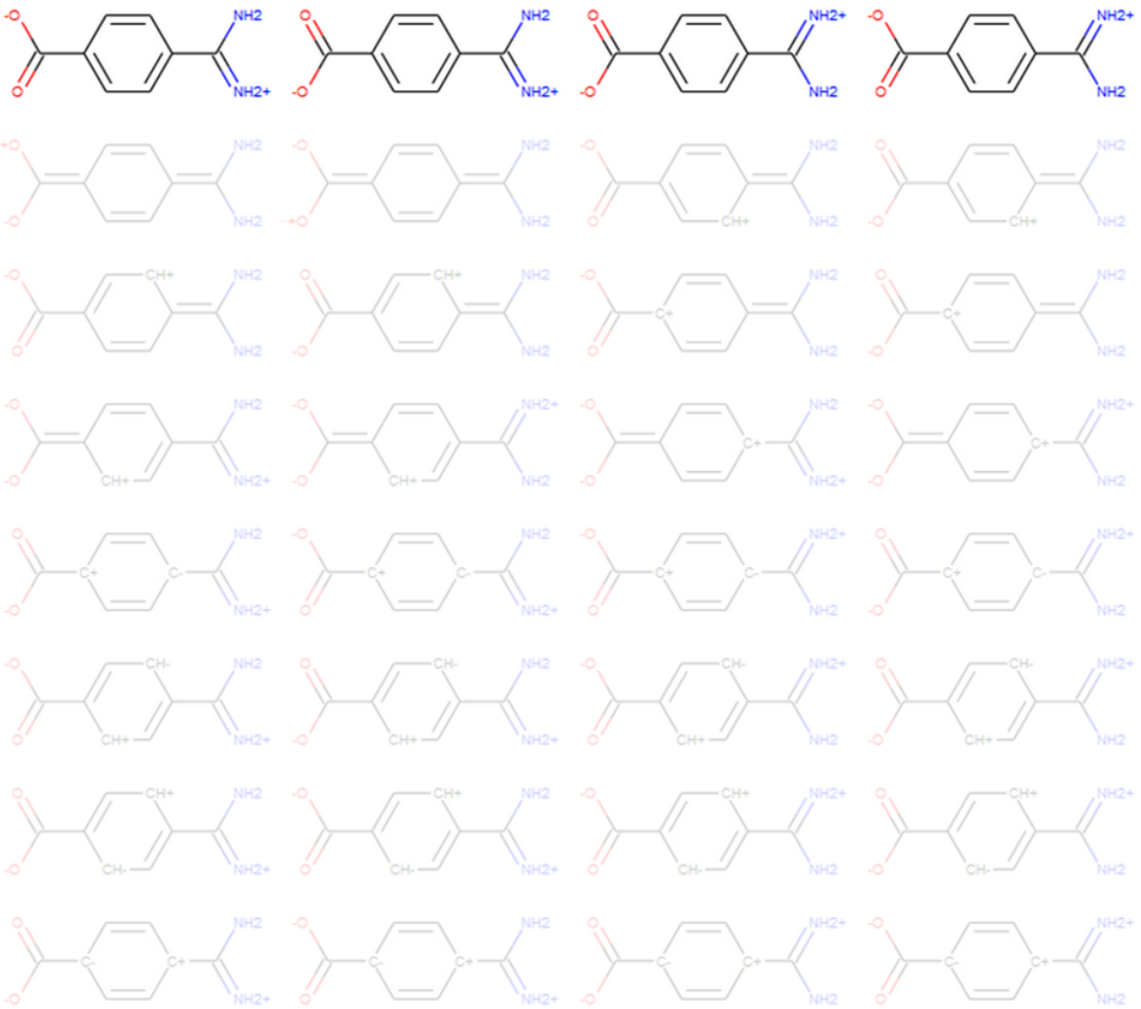
- The `ResonanceMolSupplier` object works just like the familiar `SDMolSupplier` object (lazy evaluation: the resonance structures are not constructed until the user asks for them)

Results

Results (click here for more results)

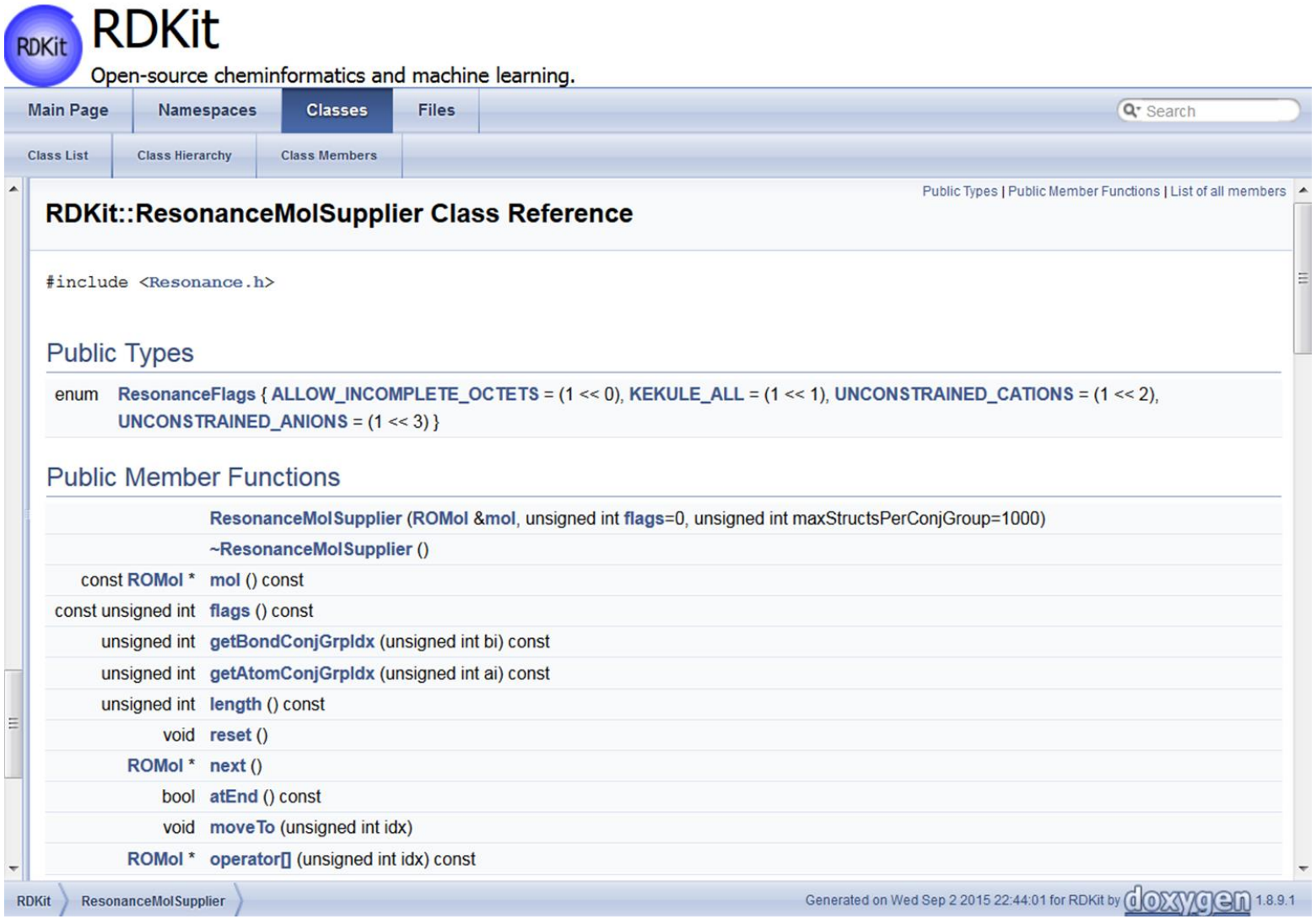


- The above molecule gives rise to 32 resonance structures
- Only the first 4 feature complete octets
- By default, the others are not generated, unless the `ALLOW_INCOMPLETE_OCTETS`, `UNCONSTRAINED_CATIONS` and `UNCONSTRAINED_ANIONS` flags are set

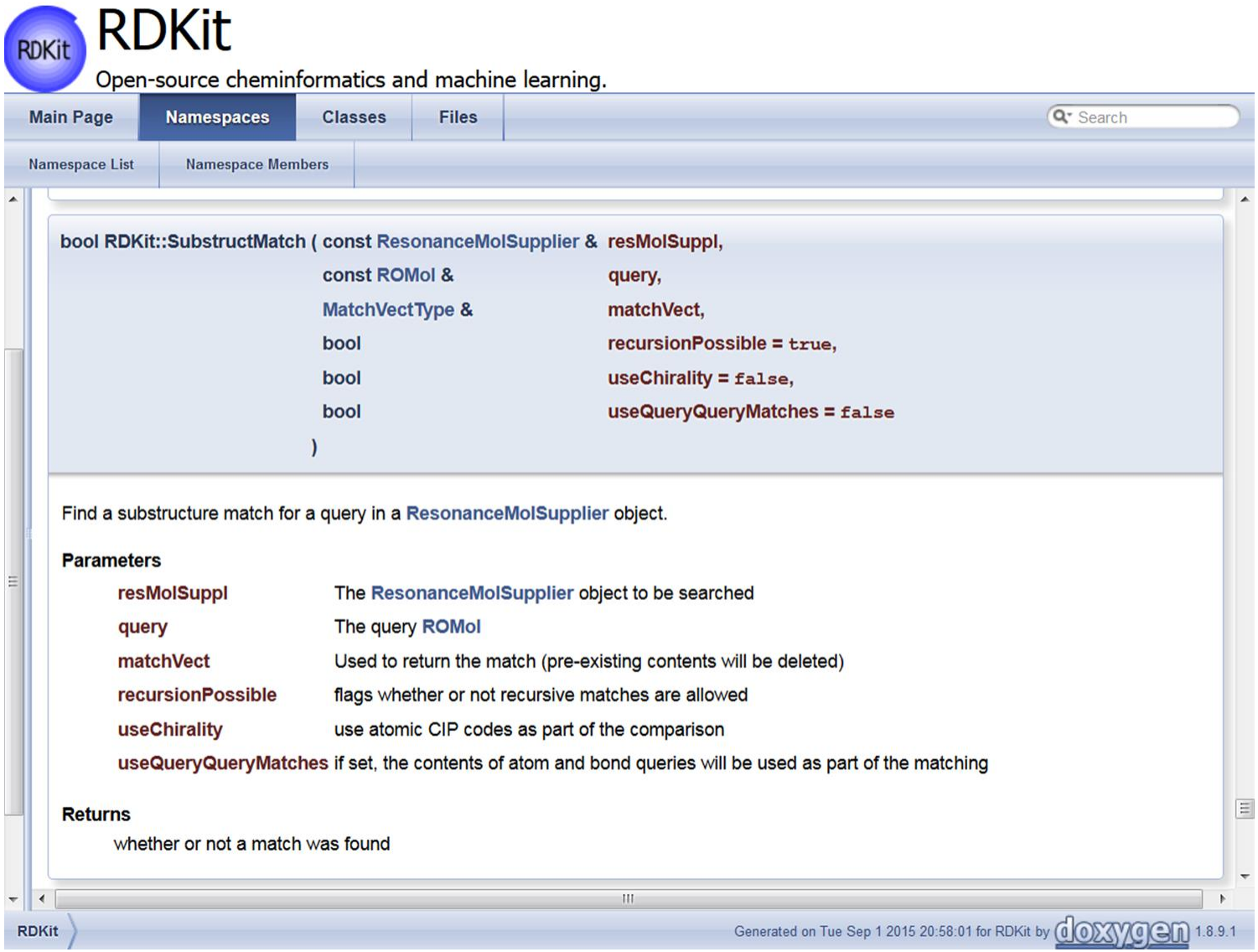


Description of the API

C++ API (1)



C++ API (2)



RDKit

Open-source cheminformatics and machine learning.

Main Page | Namespaces | Classes | Files | Search

Namespace List | Namespace Members

```
bool RDKit::SubstructMatch ( const ResonanceMolSupplier &  resMolSuppl,
                             const ROMol &                 query,
                             MatchVectType &               matchVect,
                             bool                          recursionPossible = true,
                             bool                          useChirality = false,
                             bool                          useQueryQueryMatches = false
                           )
```

Find a substructure match for a query in a ResonanceMolSupplier object.

Parameters

resMolSuppl	The ResonanceMolSupplier object to be searched
query	The query ROMol
matchVect	Used to return the match (pre-existing contents will be deleted)
recursionPossible	flags whether or not recursive matches are allowed
useChirality	use atomic CIP codes as part of the comparison
useQueryQueryMatches	if set, the contents of atom and bond queries will be used as part of the matching

Returns

whether or not a match was found

RDKit | Generated on Tue Sep 1 2015 20:58:01 for RDKit by doxygen 1.8.9.1

C++ API (3)

RDKit

Open-source cheminformatics and machine learning.

Main Page | Namespaces | Classes | Files | Search

Namespace List | Namespace Members

```
unsigned int RDKit::SubstructMatch ( const ResonanceMolSupplier &  resMolSuppl,
                                     const ROMol &                 query,
                                     std::vector< MatchVectType > & matchVect,
                                     bool                          uniquify = true,
                                     bool                          recursionPossible = true,
                                     bool                          useChirality = false,
                                     bool                          useQueryQueryMatches = false,
                                     unsigned int                  maxMatches = 1000
                                   )
```

Find all substructure matches for a query in a **ResonanceMolSupplier** object.

Parameters

resMolSuppl	The **ResonanceMolSupplier** object to be searched
query	The query **ROMol**
matchVect	Used to return the matches (pre-existing contents will be deleted)
uniquify	Toggles uniquification (by atom index) of the results
recursionPossible	flags whether or not recursive matches are allowed
useChirality	use atomic CIP codes as part of the comparison
useQueryQueryMatches	if set, the contents of atom and bond queries will be used as part of the matching
maxMatches	The maximum number of matches that will be returned. In high-symmetry cases with medium-sized molecules, it is very easy to end up with a combinatorial explosion in the number of possible matches. This argument prevents that from having unintended consequences

Returns

the number of matches found

RDKit

Generated on Tue Sep 1 2015 20:58:01 for RDKit by doxygen 1.8.9.1

Python API (1)

Trees Indices Help RDKit

Package rdkit :: Package Chem :: Module rdchem :: Class ResonanceMolSupplier

[hide private]
[frames] | no frames]

Class ResonanceMolSupplier

```
 object --+
          |
??.instance --+
              |
             ResonanceMolSupplier
```

A class which supplies resonance structures (as mols) from a mol.

Usage examples:

1) Lazy evaluation: the resonance structures are not constructed until we ask for them:

```
>>> suppl = ResonanceMolSupplier(mol)
>>> for resMol in suppl:
...     resMol.GetNumAtoms()
```

2) Lazy evaluation 2:

```
>>> suppl = ResonanceMolSupplier(mol)
>>> resMol1 = suppl.next()
>>> resMol2 = suppl.next()
>>> suppl.reset()
>>> resMol3 = suppl.next()
# resMol3 and resMol1 are the same:
>>> MolToSmiles(resMol3)==MolToSmiles(resMol1)
```

3) Random Access:

```
>>> suppl = ResonanceMolSupplier(mol)
>>> resMol1 = suppl[0]
>>> resMol2 = suppl[1]
```

NOTE: this will generate an IndexError if the supplier doesn't have that many molecules.

4) Random Access 2: looping over all resonance structures

```
>>> suppl = ResonanceMolSupplier(mol)
>>> nResMols = len(suppl)
>>> for i in range(nResMols):
...    suppl[i].GetNumAtoms()
```

cresset

Python API (2)

Instance Methods	[hide private]
	GetSubstructMatch(...) GetSubstructMatch((ResonanceMolSupplier)self, (Mol)query [, (bool)useChirality=False [, (bool)useQueryQueryMatches=False]]) -> object : Returns the indices of the molecule's atoms that match a substructure query, taking into account all resonance structures in ResonanceMolSupplier.
	GetSubstructMatches(...) GetSubstructMatches((ResonanceMolSupplier)self, (Mol)query [, (bool)uniquify=True [, (bool)useChirality=False [, (bool)useQueryQueryMatches=False [, (int)maxMatches=1000]]]]) -> object : Returns tuples of the indices of the molecule's atoms that match a substructure query, taking into account all resonance structures in ResonanceMolSupplier.
	__getitem__(...) __getitem__((ResonanceMolSupplier)arg1, (int)arg2) -> Mol :
	__init__(...) __init__((object)arg1, (Mol)mol [, (int)flags=0 [, (int)maxStructsPerConjGroup=1000]]) -> None :
	__iter__(...) __iter__((ResonanceMolSupplier)arg1) -> ResonanceMolSupplier :
	__len__(...) __len__((ResonanceMolSupplier)arg1) -> int :
	__reduce__(...) helper for pickle
	atEnd(...) atEnd((ResonanceMolSupplier)arg1) -> bool : Returns whether or not we have hit the end of the resonance structure supplier.
	next(...) next((ResonanceMolSupplier)arg1) -> Mol : Returns the next resonance structure in the supplier.
	reset(...) reset((ResonanceMolSupplier)arg1) -> None : Resets our position in the resonance structure supplier to the beginning.

cresset

Back to acetate: now we are talking!

```
In [1]: import sys
        import rdkit
        from rdkit import Chem
        from rdkit.Chem.Draw import IPythonConsole, MolsToGridImage

In [2]: mol = Chem.MolFromSmiles('CC(=O)[O-]')

In [3]: resSuppl = Chem.ResonanceMolSupplier(mol)

In [4]: resMols = [mol for mol in resSuppl]

In [5]: MolsToGridImage(resMols, subImgSize = (200, 90), molsPerRow = 4)
```

Out[5]:

```
In [6]: query = Chem.MolFromSmarts('C(=O)[O-]')
        resSuppl.GetSubstructMatches(query, uniquify = False)

Out[6]: ((1, 3, 2), (1, 2, 3))
```

Thanks to the `GetSubstructMatches()` method on `ResonanceMolSupplier`, the matches are now the union of the matches on each individual resonance structure

Resonance-aware RMSD calculation

This is the only change

In [4]:
```
def symmUniqueFit(prbMol, refMol, prbConfId = -1, refConfId = -1):
    refSuppl = Chem.ResonanceMolSupplier(refMol)
    matches = refSuppl.GetSubstructMatches(prbMol, uniquify = False)
    if (not matches):
        raise ValueError('mols don\'t match')
    amaps = [list(enumerate(match)) for match in matches]
    first = True
    rmsdMin = 0.0
    amapBest = []
    for amap in amaps:
        conf = saveConformer(prbMol, prbConfId)
        rmsd = Chem.rdMolAlign.AlignMol(prbMol, refMol,
            prbConfId, refConfId, atomMap = amap)
        loadConformer(prbMol, prbConfId, conf)
        if (first or (rmsd < rmsdMin)):
            first = False
            rmsdMin = rmsd
            amapBest = amap
    return amapBest
```

In [6]:
```
suppl = Chem.SDMolSupplier('acetate.sdf')
refMol = suppl[0]
prbMol = suppl[1]
amap = None
amap = symmUniqueFit(refMol, prbMol)
if (len(amap) != refMol.GetNumAtoms()):
    raise ValueError('atomMap does not contain all atoms')
rmsd = getRmsdImmobile(prbMol, refMol, atomMap = amap)
sys.stdout.write('{0:.2f}\n'.format(rmsd))
```

Yay!

```
0.00
```

Conclusions

- A `ResonanceMolSupplier` object has been implemented to enumerate resonance structures in molecules having conjugated groups
- The `ResonanceMolSupplier` object can be accessed from both C++ and Python APIs
- Resonance-aware substructure matching is now possible taking advantage of the new `ResonanceMolSupplier` object

Thank you for your attention

paolo@cresset-group.com