chem-bla-ics

Modern chemistry in the CDK: beyond the two-atom bond

Egon Willighagen

Published December 30, 2006

Citation
Willighagen, E. (2006). Modern chemistry in the CDK: beyond the two-atom bond. In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/h2ytc-6pt51

Keywords
Cheminf, Cdk

Abstract
Rich recently blogged about the limitations of the two-atom bond representation often used in chemoinformatics, triggered by the four ferrocene entries in PubChem . In reply to himself, Rich described FlexMol , an XML language that can describe bond systems that involve more than two atoms.

Rich recently blogged about the limitations of the two-atom bond representation often used in chemoinformatics, triggered by the four ferrocene entries in PubChem . In reply to himself, Rich described FlexMol , an XML language that can describe bond systems that involve more than two atoms.

Obviously, the problems originates from the lack of mathematical knowledge of chemists: the current chemoinformatics heavily depends on graph theory, where each atom is a vertex and each bond an edge. This has the advantage that we can borrow all algorithms that work with graph representations, such as Dijkstra's algorithm to find the shortest path between two vertices. Or, in chemical language, an algorithm to calculate how many bonds two atoms are apart in a molecule.

When discussing FlexMol, Rich mentions the work by Dietz (DOI:10.1021/ci00027a001), but I would like to mention the PhD thesis of S. Bauerschmidt to this (see DOI:10.1021/ci9704423) done in Gasteiger's group. Dropping this 'two-atom bond' representation in favor of something that better describes compounds like ferrocene, like the Dietz and Bauerschmidt approaches, has the unfortunate disadvantage of loosing compatibility with graph theory algorithms. Nevertheless, in order to take chemoinformatics to the next level, we have to address these issues. But hope is not lost, and people are working on rewriting our toolkit of chemoinformatics algorithms to match such new representations.

CDK

I will postpone analyzing the CDK for compatibility with such more modern representations (look out for a CDK News article), and now just describe how the CDK can be used for FlexMol/Dietz/Bauerschmidt representations. Consider the four examples Rich gives in his blog. Here are the CDK ways of doing the same.

For example, 1,3,5-cyclohexatriene:

```
public IMolecule makeCycloHexaTriene() {
  IMolecule cyclohexatriene = builder.newMolecule();

  IAtom atomC0 = builder.newAtom(Elements.CARBON);
    atomC0.setID("C0"); atomC0.setHydrogenCount(1);
  IAtom atomC1 = builder.newAtom(Elements.CARBON);
    atomC1.setID("C1"); atomC1.setHydrogenCount(1);
  IAtom atomC2 = builder.newAtom(Elements.CARBON);
    atomC2.setID("C2"); atomC2.setHydrogenCount(1);
  IAtom atomC3 = builder.newAtom(Elements.CARBON);
    atomC3.setID("C3"); atomC3.setHydrogenCount(1);
  IAtom atomC4 = builder.newAtom(Elements.CARBON);
    atomC4.setID("C4"); atomC4.setHydrogenCount(1);
  IAtom atomC5 = builder.newAtom(Elements.CARBON);
```

```
    atomC5.setID("C5"); atomC5.setHydrogenCount(1);

  IBond bondB0 = builder.newBond(atomC0, atomC1, 1.0);
    bondB0.setElectronCount(2);
  IBond bondB1 = builder.newBond(atomC1, atomC2, 2.0);
    bondB1.setElectronCount(4);
  IBond bondB2 = builder.newBond(atomC2, atomC3, 1.0);
    bondB2.setElectronCount(2);
  IBond bondB3 = builder.newBond(atomC3, atomC4, 2.0);
    bondB3.setElectronCount(4);
  IBond bondB4 = builder.newBond(atomC4, atomC5, 1.0);
    bondB4.setElectronCount(2);
  IBond bondB5 = builder.newBond(atomC0, atomC5, 2.0);
    bondB5.setElectronCount(4);

  cyclohexatriene.addAtom(atomC0); cyclohexatriene.addAtom(atomC1);
  cyclohexatriene.addAtom(atomC2); cyclohexatriene.addAtom(atomC3);
  cyclohexatriene.addAtom(atomC4); cyclohexatriene.addAtom(atomC5);

  cyclohexatriene.addBond(bondB0); cyclohexatriene.addBond(bondB1);
  cyclohexatriene.addBond(bondB2); cyclohexatriene.addBond(bondB3);
  cyclohexatriene.addBond(bondB4); cyclohexatriene.addBond(bondB5);

  return cyclohexatriene;
}
```

Summarizing, the key thing is to use the `IBond.setElectronCount()` method. The call is sort of redundant, as the CDK defaults to two electrons if not explicitly given. This compound is, of course, benzene which we can represent like this too:

```
public IMolecule makeBenzene() {
  IMolecule benzene = builder.newMolecule();

  IAtom atomC0 = builder.newAtom(Elements.CARBON);
    atomC0.setID("C0"); atomC0.setHydrogenCount(1);
  IAtom atomC1 = builder.newAtom(Elements.CARBON);
    atomC1.setID("C1"); atomC1.setHydrogenCount(1);
  IAtom atomC2 = builder.newAtom(Elements.CARBON);
    atomC2.setID("C2"); atomC2.setHydrogenCount(1);
  IAtom atomC3 = builder.newAtom(Elements.CARBON);
    atomC3.setID("C3"); atomC3.setHydrogenCount(1);
  IAtom atomC4 = builder.newAtom(Elements.CARBON);
    atomC4.setID("C4"); atomC4.setHydrogenCount(1);
  IAtom atomC5 = builder.newAtom(Elements.CARBON);
```

```
    atomC5.setID("C5"); atomC5.setHydrogenCount(1);

  IBond bondB0 = builder.newBond(atomC0, atomC1);
    bondB0.setElectronCount(2);
  IBond bondB1 = builder.newBond(atomC1, atomC2);
    bondB1.setElectronCount(2);
  IBond bondB2 = builder.newBond(atomC2, atomC3);
    bondB2.setElectronCount(2);
  IBond bondB3 = builder.newBond(atomC3, atomC4);
    bondB3.setElectronCount(2);
  IBond bondB4 = builder.newBond(atomC4, atomC5);
    bondB4.setElectronCount(2);
  IBond bondB5 = builder.newBond(atomC0, atomC5);
    bondB5.setElectronCount(2);

  IBond bondingSystem = builder.newBond();
    bondingSystem.setElectronCount(6);
    bondingSystem.setAtoms(
      new IAtom[] { atomC0, atomC1, atomC2,
                    atomC3, atomC4, atomC5}
    );

  benzene.addAtom(atomC0); benzene.addAtom(atomC1);
  benzene.addAtom(atomC2); benzene.addAtom(atomC3);
  benzene.addAtom(atomC4); benzene.addAtom(atomC5);

  benzene.addBond(bondB0); benzene.addBond(bondB1);
  benzene.addBond(bondB2); benzene.addBond(bondB3);
  benzene.addBond(bondB4); benzene.addBond(bondB5);
  benzene.addBond(bondingSystem);

  return benzene;
}
```

This version represents the delocalized aromatic pi-system as one IBond: one with 6 electrons, and 6 associated atoms.

The cyclopentadienyl anion is represented similarly:

```
public IMolecule makeCycloPentadienylAnion() {
  IMolecule cp = builder.newMolecule();

  IAtom atomC0 = builder.newAtom(Elements.CARBON);
 atomC0.setID("C0"); atomC0.setHydrogenCount(1);
  IAtom atomC1 = builder.newAtom(Elements.CARBON);
```

```
 atomC1.setID("C1"); atomC1.setHydrogenCount(1);
  IAtom atomC2 = builder.newAtom(Elements.CARBON);
 atomC2.setID("C2"); atomC2.setHydrogenCount(1);
  IAtom atomC3 = builder.newAtom(Elements.CARBON);
 atomC3.setID("C3"); atomC3.setHydrogenCount(1);
  IAtom atomC4 = builder.newAtom(Elements.CARBON);
 atomC4.setID("C4"); atomC4.setHydrogenCount(1);

  IBond bondB0 = builder.newBond(atomC0, atomC1);
    bondB0.setElectronCount(2);
  IBond bondB1 = builder.newBond(atomC1, atomC2);
    bondB1.setElectronCount(2);
  IBond bondB2 = builder.newBond(atomC2, atomC3);
    bondB2.setElectronCount(2);
  IBond bondB3 = builder.newBond(atomC3, atomC4);
    bondB3.setElectronCount(2);
  IBond bondB4 = builder.newBond(atomC4, atomC0);
    bondB4.setElectronCount(2);

  IBond bondingSystem = builder.newBond();
    bondingSystem.setElectronCount(6);
  bondingSystem.setAtoms(
    new IAtom[]{ atomC0, atomC1, atomC2, atomC3, atomC4}
  );

  cp.addAtom(atomC0); cp.addAtom(atomC1);
  cp.addAtom(atomC2); cp.addAtom(atomC3);
  cp.addAtom(atomC4);

  cp.addBond(bondB0); cp.addBond(bondB1);
  cp.addBond(bondB2); cp.addBond(bondB3);
  cp.addBond(bondB4); cp.addBond(bondingSystem);

  return cp;
}
```

And the final step in this series, is ferrocene:

```
public IMolecule makeFerrocene() {
  IMolecule ferrocene = builder.newMolecule();

  IAtom atomC0 = builder.newAtom(Elements.CARBON);
    atomC0.setID("C0"); atomC0.setHydrogenCount(1);
  IAtom atomC1 = builder.newAtom(Elements.CARBON);
```

```
    atomC1.setID("C1"); atomC1.setHydrogenCount(1);
  IAtom atomC2 = builder.newAtom(Elements.CARBON);
    atomC2.setID("C2"); atomC2.setHydrogenCount(1);
  IAtom atomC3 = builder.newAtom(Elements.CARBON);
    atomC3.setID("C3"); atomC3.setHydrogenCount(1);
  IAtom atomC4 = builder.newAtom(Elements.CARBON);
    atomC4.setID("C4"); atomC4.setHydrogenCount(1);
  IAtom atomC5 = builder.newAtom(Elements.CARBON);
    atomC5.setID("C5"); atomC5.setHydrogenCount(1);
  IAtom atomC6 = builder.newAtom(Elements.CARBON);
    atomC6.setID("C6"); atomC6.setHydrogenCount(1);
  IAtom atomC7 = builder.newAtom(Elements.CARBON);
    atomC7.setID("C7"); atomC7.setHydrogenCount(1);
  IAtom atomC8 = builder.newAtom(Elements.CARBON);
    atomC8.setID("C8"); atomC8.setHydrogenCount(1);
  IAtom atomC9 = builder.newAtom(Elements.CARBON);
    atomC9.setID("C9"); atomC9.setHydrogenCount(1);
  IAtom iron = builder.newAtom(Elements.IRON);
    iron.setID("Fe10"); iron.setHydrogenCount(0);

  IBond bondB0 = builder.newBond(atomC0, atomC1);
    bondB0.setElectronCount(2);
  IBond bondB1 = builder.newBond(atomC1, atomC2);
    bondB1.setElectronCount(2);
  IBond bondB2 = builder.newBond(atomC2, atomC3);
    bondB2.setElectronCount(2);
  IBond bondB3 = builder.newBond(atomC3, atomC4);
    bondB3.setElectronCount(2);
  IBond bondB4 = builder.newBond(atomC4, atomC0);
    bondB4.setElectronCount(2);
  IBond bondB5 = builder.newBond(atomC5, atomC6);
    bondB5.setElectronCount(2);
  IBond bondB6 = builder.newBond(atomC6, atomC7);
    bondB6.setElectronCount(2);
  IBond bondB7 = builder.newBond(atomC7, atomC8);
    bondB7.setElectronCount(2);
  IBond bondB8 = builder.newBond(atomC8, atomC9);
    bondB8.setElectronCount(2);
  IBond bondB9 = builder.newBond(atomC9, atomC5);
    bondB9.setElectronCount(2);

  IBond bondingSystem1 = builder.newBond();
    bondingSystem1.setElectronCount(6);
    bondingSystem1.setAtoms(
```

```
      new IAtom[] {
       atomC0, atomC1, atomC2, atomC3, atomC4, iron
      }
    );
  IBond bondingSystem2 = builder.newBond();
    bondingSystem2.setElectronCount(6);
    bondingSystem2.setAtoms(
      new IAtom[] {
        atomC5, atomC6, atomC7, atomC8, atomC9, iron
      }
    );
  IBond bondingSystem3 = builder.newBond();
    bondingSystem3.setElectronCount(6);
    bondingSystem3.setAtoms(
      new IAtom[]{
        atomC0, atomC1, atomC2, atomC3, atomC4,
        atomC5, atomC6, atomC7, atomC8, atomC9,
        iron
      }
    );

  ferrocene.addAtom(atomC0); ferrocene.addAtom(atomC1);
  ferrocene.addAtom(atomC2); ferrocene.addAtom(atomC3);
  ferrocene.addAtom(atomC4); ferrocene.addAtom(atomC5);
  ferrocene.addAtom(atomC6); ferrocene.addAtom(atomC7);
  ferrocene.addAtom(atomC8); ferrocene.addAtom(atomC9);
  ferrocene.addAtom(iron);

  ferrocene.addBond(bondB0); ferrocene.addBond(bondB1);
  ferrocene.addBond(bondB2); ferrocene.addBond(bondB3);
  ferrocene.addBond(bondB4);
  ferrocene.addBond(bondB5); ferrocene.addBond(bondB6);
  ferrocene.addBond(bondB7); ferrocene.addBond(bondB8);
  ferrocene.addBond(bondB9);
  ferrocene.addBond(bondingSystem1);
  ferrocene.addBond(bondingSystem2);
  ferrocene.addBond(bondingSystem3);

  return ferrocene;
}
```

Now, you will note that this approach does not exactly follow Rich's FlexMol examples: the skipped atom pair concepts in the FlexMol version of ferrocene. His example, more closely

follows what we are likely to draw, while the CDK code above more closely follows the molecular orbital concept. (I have to check to see how Dietz and Bauerschmidt did this.)

As said, the real trick is to have the chemoinformatics toolkit that can work with this representation, but I will save that for later. At least our algorithms to calculate the molecular mass should work ;)