The RDKit and PostgreSQL: an open-source database system for chemistry

Gregory Landrum, Andy Palmer
NIBR IT
Novartis Institutes for BioMedical Research, Basel and Cambridge

5th Meeting on U.S. Government Chemical Databases and Open Chemistry
August 2011
Acknowledgements

- **Novartis:**
  - Tom Digby (Legal)
  - John Davies (CPC/LFP)
  - Eddie Cao (NIBR IT)
  - Richard Lewis (GDC/CADD)
  - Nikolaus Stiefl (GDC/CADD)
  - Peter Gedeck (GDC/CADD)

- **Rational Discovery:**
  - Santosh Putta (currently at Nodality)
  - Julie Penzotti

- **RDKit open-source community**

- **postgresql cartridge:**
  - Michael Stonebraker
  - Oleg Bartunov
  - Teodor Sigaev
  - Pavel Velikhov

- **Entagen (SWIG wrappers):**
  - Chris Bouton
  - Erik Bakke
  - James Hardwick

- **knime.com**
  - Michael Berthold
  - Thorsten Meinl
  - Bernd Wiswedel
Overview

- **RDKit: what is it?**
- **RDKit + Knime**
- **RDKit + PostgreSQL**
- **Case study: matched pairs analysis**
Cheminformatics toolkits

Timeline of cheminformatics toolkits

Source: Andrew Dalke
RDKit: What is it?

- Python (2.x), Java, and C++ toolkit for cheminformatics
  - Core data structures and algorithms in C++, including heavy use of Boost
  - Python wrapper generated using Boost.Python
  - Java wrapper generated with SWIG

- Functionality:
  - 2D and 3D molecular operations
  - Descriptor generation for machine learning
  - Molecular database cartridge
  - Knime nodes
  - Supports Mac/Windows/Linux

- History:
  - 2000-2006: Developed and used at Rational Discovery for building predictive models for ADME, Tox, biological activity
  - June 2006: Open-source (BSD license) release of software, Rational Discovery shuts down
  - to present: Open-source development continues, use within Novartis, contributions from Novartis back to open-source version
What is this all about?

Exact same algorithms/implementations accessible from many different endpoints
RDKit: Where is it?

- Web page: http://www.rdkit.org
- Sourceforge: svn repository, bug tracker, mailing lists, downloads
  - http://sourceforge.net/projects/rdkit
- Google code: wiki, downloads
  - http://code.google.com/p/rdkit/
- Releases: quarterly (more or less)
- Licensing: new BSD
- Documentation:
  - HTML/PDF “Getting Started” documentation
  - in-code docs extracted by either doxygen (C++) or epydoc (python)
- Getting help:
  - Check the wiki and “Getting Started” document
  - The rdkit-discuss mailing list
RDKit: Documentation?

The new documentation:

Built using Python’s standard docs tool: Sphinx
RDKit: Documentation?

Sample section from introductory docs:

**Reading and Writing Molecules**

**Reading single molecules**
The majority of the basic molecular functionality is found in module `rdkit.Chem`:

```python
>>> from rdkit import Chem
```

Individual molecules can be constructed using a variety of approaches:

```python
>>> m = Chem.MolFromSmiles('Cc1ccccc1')
>>> m = Chem.MolFromMolFile('data/input.mol')
>>> stringWithMolData=file('data/input.mol','r').read()
>>> m = Chem.MolFromMolBlock(stringWithMolData)
```

All of these functions return a `Mol` object on success:

```python
>>> m
<rdkit.Chem.rdchem.Mol object at 0x...>
```

Note: docs that include python code snippets are *tested*. 
### RDKit: Documentation?

The wiki: http://code.google.com/p/rdkit/w/list

<table>
<thead>
<tr>
<th>PageName</th>
<th>Summary + Labels</th>
<th>Changed</th>
<th>ChangedBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working Builds</td>
<td>Summary of platforms where the RDKit has been built</td>
<td>23 hours ago</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>GettingStartedInCPP_1</td>
<td>A sample application in C++</td>
<td>Aug 14</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>LoadingChEMBL</td>
<td>Loading the full ChEMBL database into PostgreSQL</td>
<td>Aug 12</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>BuildingModelsUsingDescriptors1</td>
<td>Building simple predictive models using descriptors</td>
<td>Aug 10</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>TrainAThreeClassSolubilityModel</td>
<td>Train a DecisionTree based on the Huuskonen data set</td>
<td>Jul 29</td>
<td><a href="mailto:paul@tonair.de">paul@tonair.de</a></td>
</tr>
<tr>
<td>DatabaseCreation2</td>
<td>Creating and using a chemical database 2: Working with the eMolecules catalog</td>
<td>Jul 5</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>UsingTheCartridgeFromPython</td>
<td>Using the database cartridge from python</td>
<td>Jul 4</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>BuildingOnWindows</td>
<td>Building on Windows</td>
<td>Jul 1</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>BuildingWithCmake</td>
<td>New Linux (and Mac) build instructions</td>
<td>Jun 30</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>BuildingOnCentOS</td>
<td>RDKit build walk through for CentOS 5.5</td>
<td>Jun 13</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>Benchmarking</td>
<td>Some benchmarking results</td>
<td>Jun 10</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>InstallingOnWindows</td>
<td>Getting a binary installation working on windows</td>
<td>Jun 10</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>Gettinginvolved</td>
<td>Getting involved in the project.</td>
<td>May 15</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>BuildingTheCartridge</td>
<td>How to build the PostgreSQL cartridge</td>
<td>May 14</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>BuildingModelsUsingFingerprints1</td>
<td>Building simple predictive models using fingerprints</td>
<td>May 2</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>BenchmarkingMoleculeConstruction</td>
<td>Aspects of building a molecule</td>
<td>Apr 5</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>ExampleStructureQueries</td>
<td>Some example queries for doing structure searching</td>
<td>Apr 5</td>
<td>greg.landrum</td>
</tr>
<tr>
<td>ReadingAtomsInTheRDKit</td>
<td>Overview of the descriptors available in the RDKit</td>
<td>May 25</td>
<td>greg.landrum</td>
</tr>
</tbody>
</table>
RDKit: Who is using it?

- Hard to say with any certainty
- ~300 downloads of each new version
- Active contributors to the mailing list from:
  - Big pharma
  - Small pharma/biotech
  - Software/Services
  - Academia
- Starting to see contributions coming from the community (wiki pages, code patches, changes to the build system, etc.) as well as active use in other systems.
Things you should know

- Molecules should be “correct”: i.e. there should be a valid Lewis-dot structure. If not, they will be rejected:

  >>> Chem.MolFromSmiles('CC(F)(Cl)(Br)I')
  [08:58:09] Explicit valence for atom # 1 C, 5, is greater than permitted

- The software generally doesn’t try and read the user’s mind

  Nitro groups and N-oxides are repaired:
  
  >>> Chem.CanonSmiles('CN(=O)=O')
  'C[N+](=O)[O-]'

  >>> Chem.CanonSmiles('c1ccccc1=O')
  '[O-][n+]1ccccc1'

  but some odd constructs (this one from ChEMBL SMILES) are not:

  >>> Chem.MolFromSmiles('CN=N#N')
  [16:30:08] Explicit valence for atom # 2 N, 5, is greater than permitted
  ... snip ...

  >>> Chem.CanonSmiles('CN=[N+]=[N-]')
  'CN=[N+]=[N-]'
What can you do with it?

A laundry list

- Input/Output: SMILES/SMARTS, SDF, TDT, SLN\(^1\), Corina mol2\(^1\), InChI\(^2\)
- "Cheminformatics":
  - Substructure searching
  - Canonical SMILES
  - Chirality support (i.e. R/S or E/Z labeling)
  - Chemical transformations (e.g. remove matching substructures)
  - Chemical reactions
  - Molecular serialization (e.g. mol <-> text)
- 2D depiction, including constrained depiction
- 2D->3D conversion/conformational analysis via distance geometry
- UFF implementation for cleaning up structures
- Fingerprinting:
  Daylight-like, atom pairs, topological torsions, Morgan algorithm, “MACCS keys”, etc.
- Similarity/diversity picking
- 2D pharmacophores\(^1\)
- Gasteiger-Marsili charges
- Murcko analysis
- Hierarchical subgraph/fragment analysis
- RECAP and BRICS implementations

\(^1\) functional, but not great implementations
\(^2\) Solid output, experimental input
What can you do with it?

A laundry list, cntd

- Feature maps
- Shape-based similarity
- Molecule-molecule alignment
- Shape-based alignment (subshape alignment) \(^1\)
- Integration with PyMOL for 3D visualization
- Database integration

Molecular descriptor library:
- Topological (\(\kappa 3\), Balaban J, etc.)
- Electrotopological state (Estate)
- clogP, MR (Wildman and Crippen approach)
- “MOE like” VSA descriptors
- Feature-map vectors

Machine Learning:
- Clustering (hierarchical)
- Information theory (Shannon entropy, information gain, etc.)
- Decision trees, naïve Bayes\(^1\), kNN\(^1\)
- Bagging, random forests
- Infrastructure (data splitting, shuffling, enrichment plots, serializable models, etc.)

\(^1\) functional, but not great implementations
Overview

- RDKit: what is it?
- RDKit + Knime
- RDKit + PostgreSQL
- Case study: matched pairs analysis
Knime integration

- Out of the box Knime is strong on data processing and mining, weak on chemistry.
- Goal: develop a set of open-source RDKit-based nodes for Knime that provide basic cheminformatics functionality
- Distributed from knime community site
- Binaries available as an update site (no RDKit build/installation required)
- Work in progress: more nodes being added

Note: We’ll be developing more open-source nodes at the Knime open-source days in October: http://www.knime.org/kos-2011

1 Work done together with knime.com
Overview

- RDKit: what is it?
- RDKit + Knime
- RDKit + PostgreSQL
- Case study: matched pairs analysis
The database cartridge

- Integration of RDKit fingerprinting and substructure search functionality with PostgreSQL
- Bit-vector and counts-based fingerprints along with common similarity metrics (Tanimoto and Dice) integrated with PostgreSQL indexing system to allow fast searches (~1 million compounds/sec on a single CPU)
- Available similarity fingerprints:
  - Morgan (ECFP-like), available as bit vectors or counts
  - FeatMorgan (FCFP-like), available as bit vectors or counts
  - RDKit (Daylight-like), available as bit vectors
  - atom pairs, available as bit vectors or counts
  - topological torsions, available as bit vectors or counts
- SMILES- and SMARTS-based substructure querying integrated with indexing system
- Standard Lipinski-like descriptors (logp, tpsa, MW, etc.)
- Part of the RDKit open-source distribution since July 2010.
Cartridge performance #1

- Database: 100K diverse drug-like molecules from ZINC
  - Molecules load/index time: 109 sec / 343 sec
  - Fingerprints (Morgan2) calculation/index time: 23.1 sec / 9.3 sec
- "Fragments" queries: 500 diverse fragment-like molecules from ZINC
- "Leads" queries: 500 diverse lead-like molecules from ZINC
- Hardware: MacBook Pro (2.5GHz Core2 Duo)
- Do queries via a cross join (i.e. 500 queries x 100K database molecules = 50M possible comparisons/searches)
- Results:

<table>
<thead>
<tr>
<th>Query Set</th>
<th>SSS</th>
<th>Similarity (0.8)</th>
<th>Similarity (0.6)</th>
<th>Similarity (0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zinc Fragments</td>
<td>23.3</td>
<td>14.6</td>
<td>36.4</td>
<td>37.1</td>
</tr>
<tr>
<td>Zinc Leads</td>
<td>8.2</td>
<td>14.8</td>
<td>36.2</td>
<td>38.5</td>
</tr>
</tbody>
</table>
Cartridge performance #2

- Database: 5 million molecules from the emolecules catalog
  - Molecules load/index time: 4040 sec / 7832 sec
- Hardware: Dell Studio XPS (2.9GHz i7 with 8GB of RAM, Ubuntu Linux)

### Similarity queries:
- zinc lead-like set: queries return 0-50 hits in 5-8 seconds

<table>
<thead>
<tr>
<th>SSS Query</th>
<th>Count</th>
<th>Run time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1cccc2c1nncc2</td>
<td>996</td>
<td>2.4</td>
</tr>
<tr>
<td>c1ccnc2c1nccn2</td>
<td>1366</td>
<td>0.8</td>
</tr>
<tr>
<td>c1cncc2n1ccn2</td>
<td>523</td>
<td>2.3</td>
</tr>
<tr>
<td>Nc1ncnc(N)n1</td>
<td>14227</td>
<td>3.8</td>
</tr>
<tr>
<td>c1scnn1</td>
<td>88564</td>
<td>31.6</td>
</tr>
<tr>
<td>c1cccc2c1ncs2</td>
<td>95288</td>
<td>55.3</td>
</tr>
</tbody>
</table>

- Performance here is a function of the SSS fingerprint quality
- SMARTS-based queries can be considerably slower
- Getting first 50 results from any of these queries: <0.5 sec
The database cartridge

- How hard is it to add something new?
  1. Write C++ function implementing new functionality (probably not cartridge specific, could also be used from Python or Java)
  2. Add one boilerplate C++ function to cartridge code exposing that function
  3. Add one boilerplate C function to cartridge code
  4. Add one boilerplate SQL function to cartridge code
  5. Add tests

- The first step is the hardest part (unless the function already exists), the rest takes 5-10 minutes.

- Reference/HowTo for this: [http://www.mail-archive.com/rdkit-discuss@lists.sourceforge.net/msg01549.html](http://www.mail-archive.com/rdkit-discuss@lists.sourceforge.net/msg01549.html)
Using the database cartridge

**Similarity search with Morgan fingerprint:**
```sql
vendors=# select \
    id,tanimoto_sml(morganbv_fp('N=C1OC2=C(C=CC=C2)C=C1',2),mfp2)\nfrom fps where morganbv_fp('N=C1OC2=C(C=CC=C2)C=C1',2)%mfp2 ;
```
```
<table>
<thead>
<tr>
<th>id</th>
<th>tanimoto_sml</th>
</tr>
</thead>
<tbody>
<tr>
<td>9171448</td>
<td>0.538461538461538</td>
</tr>
<tr>
<td>765434</td>
<td>0.538461538461538</td>
</tr>
</tbody>
</table>
```
(2 rows)

**Substructure Search:**
```sql
vendors=# select count(*) from mols where m@>'N=C1OC2=C(C=CC=C2)C=C1';
count
-----
2854
```
(1 row)

**Properties:**
```sql
vendors=# select mol_tpsa('C1CCC1CC(=O)O':mol) tpsa, 
  mol_logp('C1CCC1CC(=O)O':mol) clogp;
  tpsa | clogp
-----+-------
 37.3 | 1.2612
```
(1 row)
Overview

- RDKit: what is it?
- RDKit + Knime
- RDKit + PostgreSQL
- Case study: matched pairs analysis
Case Study: bringing the pieces together

- Combine the RDKit + the PostgreSQL cartridge + Knime to do matched-pairs analysis
- Idea: find pairs of molecules that are structurally similar but that have quite different activities to identify interesting/useful transformations.
- Key concept is disparity: \(\Delta\text{Activity} / (1-\text{similarity})\)
- Easily done from Python using the RDKit, but it becomes time consuming as the number of molecules increases (\(N^2\) similarity calculations required).
Atom-Pair and Topological-Torsion Fingerprints
related descriptors from the “distant” past

- Atom-type:
  (Element, #heavy neighbors, #pi electrons)
- Atom Pair\(^1\):
  Atom-type – topological distance – Atom-type

- Topological Torsion\(^2\):
  Atom-type – Atom-type – Atom-type – Atom-type

- Both fingerprints can use counts (not 1/0 values)

Using count-based fingerprints for matched pairs

```python
>>> DataStructs.TanimotoSimilarity(fp1a, fp1b)
0.89189189189189189
>>> DataStructs.TanimotoSimilarity(fp2a, fp2b)
0.85185185185185186
>>> DataStructs.TanimotoSimilarity(fp3a, fp3b)
0.89743589743589747

>>> f1 = fp1a - fp1b
>>> f2 = fp2a - fp2b
>>> f3 = fp3a - fp3b
>>> f1 == f2
True
>>> f1 == f3
True
```

Fingerprint math: subtracting two fingerprints from another gives another fingerprint
Using count-based fingerprints for matched pairs

```python
>>> for bit, v in f1.GetNonzeroElements().items():
     .....:
     print Torsions.ExplainPathScore(bit), v
     .....:
     (('C', 1, 0), ('C', 3, 1), ('C', 2, 1), ('C', 2, 1)) 1
     (('C', 1, 0), ('C', 3, 1), ('C', 2, 1), ('C', 3, 1)) 1
     (('C', 3, 1), ('C', 2, 1), ('C', 3, 1), ('Cl', 1, 0)) -1
     (('C', 2, 1), ('C', 2, 1), ('C', 3, 1), ('Cl', 1, 0)) -1
```
A standard molecular db schema

Table "herg_data.mols"
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>compound_id</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>mol</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "mols_pkey" PRIMARY KEY, btree (id)
- "molidx" gist (m)

Triggers:
process_mol AFTER INSERT OR DELETE OR UPDATE ON herg_data.mols FOR EACH ROW EXECUTE PROCEDURE herg_data.process_update_mol()

Table "herg_data.countfps"
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>pairfp</td>
<td>sfp</td>
<td></td>
</tr>
<tr>
<td>torsionfp</td>
<td>sfp</td>
<td></td>
</tr>
<tr>
<td>mfp2</td>
<td>sfp</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "countfps_pkey" PRIMARY KEY, btree (id)
- "apfpcountidx" gist (pairfp sfp_low_ops)
- "mfpscountidx" gist (mfp2 sfp_low_ops)
- "torsionfpcountidx" gist (torsionfp sfp_low_ops)

Table "herg_data.molvals"
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>mfp2</td>
<td>bfp</td>
<td></td>
</tr>
<tr>
<td>ffp2</td>
<td>bfp</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "molvals_pkey" PRIMARY KEY, btree (id)
- "mfp2idx" gist (mfp2)
- "ffp2idx" gist (ffp2)

Table "herg_data.mols".
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>compound_id</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>mol</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "mols_pkey" PRIMARY KEY, btree (id)
- "molidx" gist (m)

Triggers:
process_mol AFTER INSERT OR DELETE OR UPDATE ON herg_data.mols FOR EACH ROW EXECUTE PROCEDURE herg_data.process_update_mol()

Table "herg_data.fpss"
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>mfp2</td>
<td>bfp</td>
<td></td>
</tr>
<tr>
<td>ffp2</td>
<td>bfp</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "fps_pkey" PRIMARY KEY, btree (id)
- "ffp2idx" gist (ffp2)
- "mfp2idx" gist (mfp2)

Table "herg_data.pairbvfps"
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>pairfp</td>
<td>bfp</td>
<td></td>
</tr>
<tr>
<td>torsionfp</td>
<td>bfp</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "pairbvfps_pkey" PRIMARY KEY, btree (id)
- "apfpbvidx" gist (pairfp)
- "torsionfpbvidx" gist (torsionfp)
select *, (pact1-pact2)/dist disparity, pact2-pact1 dact from
  ( select ms1.id id1, ms1.m smiles1, ms2.id id2, ms2.m smiles2, dist,
    -1*log(vs1.ic50_nm*1e-9) pact1,
    -1*log(vs2.ic50_nm*1e-9) pact2
  from ( select fp1.id id1, fp2.id id2,
    1.0-dice_sml(fp1.torsionfp, fp2.torsionfp) dist
  from cdk2.countfps as fp1
  cross join cdk2.countfps as fp2
  where fp1.torsionfp#fp2.torsionfp and fp1.id!=fp2.id
  ) cliff_pairs
  join cdk2.mols ms1 on (id1=ms1.id)
  join cdk2.mols ms2 on (id2=ms2.id)
  join cdk2.molvals vs1 on (id1=vs1.id)
  join cdk2.molvals vs2 on (id2=vs2.id)
  where dist>0
  ) tmp
where pact1>=pact2 and (pact1-pact2)>.1
order by disparity desc
Matched pairs in SQL(simple)

```sql
select *, (pact1-pact2)/dist disparity, pact2-pact1 dact from
  ( select ms1.id id1, ms1.m smiles1, ms2.id id2, ms2.m smiles2, dist,
    -1*log(vs1.ic50_nm*1e-9) pact1,
    -1*log(vs2.ic50_nm*1e-9) pact2
  from ( select fp1.id id1, fp2.id id2,
    1.0-dice_sml(fp1.torsionfp, fp2.torsionfp) dist
  from cdk2.countfps as fp1
    cross join cdk2.countfps as fp2
    where fp1.torsionfp # fp2.torsionfp and fp1.id != fp2.id
  ) cliff_pairs
  join cdk2.mols ms1 on (id1 = ms1.id)
  join cdk2.mols ms2 on (id2 = ms2.id)
  join cdk2.molvals vs1 on (id1 = vs1.id)
  join cdk2.molvals vs2 on (id2 = vs2.id)
  where dist > 0
) tmp
where pact1 >= pact2 and (pact1 - pact2) > .1
order by disparity desc
```
select *, (pact1 - pact2)/dist disparity, pact2 - pact1 dact from
  ( select ms1.id id1, ms1.m smiles1, ms2.id id2, ms2.m smiles2, dist,
      -1*log(vs1.ic50_nm*1e-9) pact1,
      -1*log(vs2.ic50_nm*1e-9) pact2
  from ( select fp1.id id1, fp2.id id2,
         1.0 - dice_sml(fp1.torsionfp, fp2.torsionfp) dist
  from cdk2.countfps as fp1
  cross join cdk2.countfps as fp2
  where fp1.torsionfp # fp2.torsionfp and fp1.id != fp2.id
  ) cliff_pairs
  join cdk2.mols ms1 on (id1 = ms1.id)
  join cdk2.mols ms2 on (id2 = ms2.id)
  join cdk2.molvals vs1 on (id1 = vs1.id)
  join cdk2.molvals vs2 on (id2 = vs2.id)
  where dist > 0
  ) tmp
where pact1 >= pact2 and (pact1 - pact2) > .1
order by disparity desc
Matched pairs in SQL(simple)

```
select *, (pact1-pact2)/dist disparity, pact2-pact1 dact from
  ( select ms1.id id1, ms1.m smiles1, ms2.id id2, ms2.m smiles2, dist,
    -1*log(vs1.ic50_nm*1e-9) pact1,
    -1*log(vs2.ic50_nm*1e-9) pact2
    from ( select fp1.id id1, fp2.id id2,
      1.0-dice_sml(fp1.torsionfp, fp2.torsionfp) dist
      from cdk2.countfps as fp1
      cross join cdk2.countfps as fp2
      where fp1.torsionfp#fp2.torsionfp and fp1.id!=fp2.id
      ) cliff_pairs
    join cdk2.mols ms1 on (id1=ms1.id)
    join cdk2.mols ms2 on (id2=ms2.id)
    join cdk2.molvals vs1 on (id1=vs1.id)
    join cdk2.molvals vs2 on (id2=vs2.id)
    where dist>0
  ) tmp
where pact1>=pact2 and (pact1-pact2)>.1
order by disparity desc
```

---

```
Calculate disparity
```

---

```
Matched pairs in SQL(simple)
```

---

```
Matched pairs in SQL(simple)
```

---

```
Matched pairs in SQL(simple)
```
select *, (pact1-pact2)/dist disparity, pact2-pact1 dact from
  ( select ms1.id id1, ms1.m smiles1, ms2.id id2, ms2.m smiles2, dist,
    -1*log(vs1.ic50_nm*1e-9) pact1,
    -1*log(vs2.ic50_nm*1e-9) pact2,
    t4v_hash
  from ( select fp1.id id1, fp2.id id2,
    1.0-\text{dice}\_\text{sml}(fp1.torsionfp, fp2.torsionfp) dist,
    \text{md5}(\text{subtract}(fp1.torsionfp, fp2.torsionfp)::\text{text}) t4v_hash
  from cdk2.countfps as fp1
  cross join cdk2.countfps as fp2
  where fp1.torsionfp#fp2.torsionfp and fp1.id!=fp2.id
  ) cliff_pairs
  join cdk2.mols ms1 on (id1=ms1.id)
  join cdk2.mols ms2 on (id2=ms2.id)
  join cdk2.molvals vs1 on (id1=vs1.id)
  join cdk2.molvals vs2 on (id2=vs2.id)
  where dist>0
  ) tmp
where pact1>=pact2 and (pact1-pact2)>0.1
order by disparity desc
Performance

- Dataset: 1181 molecules with measured CDK2 IC50s (source: binding db)
- Fingerprints: topological torsions (count-based)
- Counting results:
  - Similarity cutoff 0.90: 1400 pairs, 0.39 sec
  - Similarity cutoff 0.85: 3719 pairs, 0.53 sec
  - Similarity cutoff 0.75: 11541 pairs, 0.85 sec
- Retrieving results:
  - Similarity cutoff 0.90: 1400 pairs, 2.0 sec
  - Similarity cutoff 0.85: 3719 pairs, 4.9 sec
  - Similarity cutoff 0.75: 11541 pairs, 14.1 sec
- Hardware: Dell Studio XPS (i7 870, 64bit)
Knime implementation
Knime implementation
<table>
<thead>
<tr>
<th>Row ID</th>
<th>S 4v hash</th>
<th>D Mean(mol1)</th>
<th>First(mol1)</th>
<th>First(mol2)</th>
<th>Count(mol1)</th>
<th>D Max(pact1)</th>
<th>D Mean(dact)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row2649</td>
<td>d3049eaf3c90e2ea202380a780a5b</td>
<td>50.461</td>
<td><img src="image1" alt="Chemical Structure" /></td>
<td><img src="image2" alt="Chemical Structure" /></td>
<td>6</td>
<td>7.699</td>
<td>-0.463</td>
</tr>
<tr>
<td>Row937</td>
<td>4890255b136b311a2078798761e0</td>
<td>51.404</td>
<td><img src="image3" alt="Chemical Structure" /></td>
<td><img src="image4" alt="Chemical Structure" /></td>
<td>7</td>
<td>7.824</td>
<td>-0.491</td>
</tr>
<tr>
<td>Row1523</td>
<td>78e01712a4f11b2e7adf2b33b21d0</td>
<td>56.469</td>
<td><img src="image5" alt="Chemical Structure" /></td>
<td><img src="image6" alt="Chemical Structure" /></td>
<td>8</td>
<td>8.155</td>
<td>-1.053</td>
</tr>
</tbody>
</table>
### Knime Implementation

<table>
<thead>
<tr>
<th>t4v hash</th>
<th>disparity</th>
<th>dact</th>
<th>mol1</th>
<th>mol2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4890255b136b311a2078798761e00...</td>
<td>100.179</td>
<td>-1.301</td>
<td><img src="image1" alt="mol1" /></td>
<td><img src="image2" alt="mol2" /></td>
</tr>
<tr>
<td>4890255b136b311a2078798761e00...</td>
<td>85.261</td>
<td>-0.812</td>
<td><img src="image3" alt="mol1" /></td>
<td><img src="image4" alt="mol2" /></td>
</tr>
<tr>
<td>4890255b136b311a2078798761e00...</td>
<td>30.155</td>
<td>-0.339</td>
<td><img src="image5" alt="mol1" /></td>
<td><img src="image6" alt="mol2" /></td>
</tr>
</tbody>
</table>
Knime implementation

<table>
<thead>
<tr>
<th>S</th>
<th>t4v_hash</th>
<th>disparity</th>
<th>dact</th>
<th>mol1</th>
<th>mol2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>78e01712a4f1:b2e7adff33b21d067b</td>
<td>93.59</td>
<td>-1.586</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>78e01712a4f1:b2e7adff33b21d067b</td>
<td>69.981</td>
<td>-1.296</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>78e01712a4f1:b2e7adff33b21d067b</td>
<td>65.55</td>
<td>-1.214</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Wrapping up

What is it?
- Cheminformatics toolkit useable from C++, Python, Java
- Postgresql cartridge for substructure/similarity searching
- Open-source Knime nodes for cheminformatics

Web presence:
- Main site: [http://www.rdkit.org](http://www.rdkit.org)
- Knime nodes: [http://tech.knime.org/community/rdkit](http://tech.knime.org/community/rdkit)
We’re hiring!

- Cambridge (the US one):
  - Cheminformatics developer in the Chemical Information Systems group
  - Technical lead in the Chemical Information Systems group
- If you’re interested, please contact me
Thanks!

C++:
Core data structures and algorithms
Backups
Comparing similarity measures

- Pick 10K random pairs of vendor compounds that have at least some topological similarity to each other (Avalon similarity $\geq 0.5$)
- Compare similarities calculated with Pipeline Pilot and the RDKit
Comparing similarity measures

- RDKit Morgan2 vs PP ECFP4

Larger differences are mostly aromaticity related

- RDKit Morgan3 vs PP ECFP6 is similar
Comparing similarity measures

- RDKit FeatMorgan2 vs PP FCFP4

Differences are due to feature definitions
84% of similarities differ by <0.05
95% differ by <0.1
Contributing to open source: why bother?

- Scientific argument for releasing source:
  
  ACS ethical guidelines: "A primary research report should contain sufficient detail and reference to public sources of information to permit the author's peers to repeat the work."
  
  (http://pubs.acs.org/userimages/ContentEditor/1218054468605/ethics.pdf)


N. Barnes, "Publish your computer code: it is good enough" *Nature* **467**:753 (2010)
Contributing to open source: why bother?

- Philosophy

- Improved code quality: users = testers/peer reviewers

- Gather new ideas/contributions from others

- Altruism: give something back to "the community"

- Selfishness: guarantee your own access to your work
Contributing to open source: why bother?

- We aren't the only big company doing this:
  - IBM
  - Apple
  - Google
  - Nokia
  - Microsoft(!)
  - many, many others

- We aren't even the only pharma company:
  - Sunesis
  - Lilly
  - Boehringer Ingelheim
  - Astra Zeneca
  - Sanofi Aventis
  - others
Practical Considerations

▪ Precompetitive

▪ Treat code publication process the same as publication of a scientific paper

▪ Pick a license carefully; use a standard one

▪ Management understanding and support

▪ Support from legal/patent department