RDKit, PostgreSQL, and Knime: Open-source cheminformatics in big pharma

Gregory Landrum, Richard Lewis, Andrew Palmer, Nikolaus Stiefl.
NIBR IT and Global Discovery Chemistry
Novartis Institutes for BioMedical Research, Basel and Cambridge
MIOSS 2011
Hinxton, 4 May 2011
Overview

- RDKit: what is it?
- RDKit + PostgreSQL
- RDKit + Knime
- Case study: matched pairs analysis
- Contributing to open source from big pharma
Acknowledgements

Novartis:
- Tom Digby (Legal)
- John Davies (CPC/LFP)
- Eddie Cao (NIBR IT)
- 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 + PostgreSQL
- RDKit + Knime
- Case study: matched pairs analysis
- Contributing to open source from big pharma
RDKit: What is it?

- Python (2.x), Java, and C++ toolkit for cheminformatics
  - Core data structures and algorithms in C++
  - Heavy use of Boost libraries
  - 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
  - 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
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:
  - “Getting Started in Python” document
  - 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: Who is using it?

- Hard to say with any certainty
- ~300 downloads of each new version
- Active contributors to the mailing list from:
  - Small pharma/biotech
  - Software/Services
  - Academia
What can you do with it?
A laundry list

- Input/Output: SMILES/SMARTS, SDF, TDT, SLN¹, Corina mol2¹
- “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 (including fuzzy similarity)
- 2D pharmacophores¹
- Gasteiger-Marsili charges
- Hierarchical subgraph/fragment analysis
- RECAP and BRICS implementations

¹ functional, but not great implementations
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, \textit{naïve Bayes}\(^1\), kNN\(^1\)
- Bagging, random forests
- Infrastructure (data splitting, shuffling, enrichment plots, serializable models, etc.)

\(^1\) functional, but not great implementations
Things you should know

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

  ```python
  >>> 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:**

  ```python
  >>> Chem.CanonSmiles('CN(=O)=O')
  'C[N+](=O)[O-]'
  >>> Chem.CanonSmiles('c1ccccn1=O')
  '[O-][n+]1ccccc1'
  ```

  **but some odd constructs (this one from CHEMBL) are not:**

  ```python
  >>> 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-]'```
Overview

- RDKit: what is it?
- RDKit + PostgreSQL
- RDKit + Knime
- Case study: matched pairs analysis
- Contributing to open source from big pharma
The database cartridge

- Integration of RDKit fingerprinting and substructure search functionality with PostgreSQL
- 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)
  - FeatMorgan (FCFP-like)
  - RDKit (Daylight-like)
  - atom pairs
  - topological torsions
- Bit vector and count-based fingerprints are supported (searches using count-based fingerprints are slower).
- SMILES- and SMARTS-based substructure querying
- Part of the RDKit open-source distribution since July 2010
The database cartridge

- Using the cartridge:

**Similarity search with Morgan fingerprint:**

```sql
vendors=# select \
    id, tanimoto_sml(morganbv_fp('N=C1OC2=C(C=CC=C2)C=C1', 2), mfp2) \
from 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';
```

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2854</td>
</tr>
</tbody>
</table>

(1 row)
Cartridge performance

- 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>
About the substructure fingerprints

- Benchmarking: determine screening accuracy (= number of SSS hits found / number of fingerprint matches) for three different types of queries run against 100K diverse drug-like molecules from ZINC:
  - 823 pieces constructed by doing a BRICS fragmentation of a set of molecules from the pubchem screening set. Size range from 1->64 atoms
  - 500 diverse lead-like molecules from ZINC
  - 500 diverse fragment-like molecules from ZINC

- Results:

<table>
<thead>
<tr>
<th>Query Set</th>
<th>Num matches</th>
<th>Avalon¹</th>
<th>RDKit-branched</th>
<th>RDKit-linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pubchem Pieces</td>
<td>2515777</td>
<td>0.35</td>
<td>0.35</td>
<td>0.32</td>
</tr>
<tr>
<td>Zinc Fragments</td>
<td>15643</td>
<td>0.30</td>
<td>0.20</td>
<td>0.13</td>
</tr>
<tr>
<td>Zinc Leads</td>
<td>1302</td>
<td>0.49</td>
<td>0.11</td>
<td>0.03</td>
</tr>
</tbody>
</table>

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
Overview

- RDKit: what is it?
- RDKit + PostgreSQL
- RDKit + Knime
- Case study: matched pairs analysis
- Contributing to open source from big pharma
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 frequently

---

1 Work done together with knime.com
Knime integration
Handling molecules
Substructure search

RDKit Mol column: rdkmol

SMARTS query: $(N-![#6]);!(N-C(=[O,N,S]))]

Do exact match

OK  Apply  Cancel
Reaction handling
Fingerprinting node

**FP type**

The type of fingerprint to generate. Choices are:

- **Morgan**: Circular fingerprint based on the Morgan algorithm and connectivity invariants (ECFP-like)
- **FeatMorgan**: Circular fingerprint based on the Morgan algorithm and feature invariants (FCFP-like)
- **AtomPair**: Atom-pair fingerprint
- **Torsion**: Topological-torsion fingerprint
- **RDKit**: Daylight-like topological fingerprint
- **Layered**: An experimental substructure-matching fingerprint
Coordinate generation

![Image of RDKit Generate Coords dialog box]

**Options**
- **RDKit Mol column**: rdkitmol
- **New column name**: rdkitmol
- **Remove source column**
- **Dimension**
  - 2D coordinates
  - 3D coordinates
- **Template Smarts**: c1ccccc2c1nc[n,c]c2

**Buttons**
- OK
- Apply
- Cancel

Node 3

[Novartis logo]
Murcko Scaffolds

Options

RDKit Mol column: Molecule (RDKit Mol)

New column name: Murcko_scaffold

Remove source column

OK  Apply  Cancel
Fragmentation

Options

RDKit Mol column: Molecule (RDKit Mol)

Min Path Length: 4

Max Path Length: 7

OK    Apply    Cancel

Row ID | Frag... | Fragment | S Fragm... | i Fragm... | Count

frag_01  1   cnc(C)c   4   20

frag_11  2   CnccN   4   5

Row ID | Name  | S Enzym... | Molecule (RDKit Mol) | (m) Fragment indices

Row1    Hidine deriv.... n/a [1,2,3,...]

Row2    Hidine deriv.... n/a [1,560,3,...]

Row3    Hidine deriv.... n/a [1,2,3,...]
R Group decomposition

RDKit Mol column: Molecule (RDKit Mol)

Core SMARTS: O=C2c1cnnc1c3cccccc23

R1, R2, R3, R4, R12, No 2D
More complex example: CDK2/CDK4 selectivity
More complex example… cont.
More complex example… cont.
More complex example... cont.
More complex example... cont.

### SSS Results

<table>
<thead>
<tr>
<th>Row ID</th>
<th>id</th>
<th>Molecule (RDKIT Mol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row0</td>
<td>162583</td>
<td>![Molecule Image]</td>
</tr>
<tr>
<td>Row1</td>
<td>178192</td>
<td>![Molecule Image]</td>
</tr>
<tr>
<td>Row2</td>
<td>247618</td>
<td>![Molecule Image]</td>
</tr>
</tbody>
</table>

### Similarity Results

<table>
<thead>
<tr>
<th>Row ID</th>
<th>id</th>
<th>First(similarity)</th>
<th>Molecule (RDKIT Mol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row0</td>
<td>167309</td>
<td>0.5</td>
<td>![Molecule Image]</td>
</tr>
<tr>
<td>Row1</td>
<td>178689</td>
<td>0.5</td>
<td>![Molecule Image]</td>
</tr>
<tr>
<td>Row2</td>
<td>196649</td>
<td>0.674</td>
<td>![Molecule Image]</td>
</tr>
</tbody>
</table>
Overview

- RDKit: what is it?
- RDKit + PostgreSQL
- RDKit + Knime
- Case study: matched pairs analysis
- Contributing to open source from big pharma
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: $\Delta$Activity / (1-similarity)
- Easily done from Python using the RDKit, but it becomes time consuming as the number of molecules increases ($N^2$ similarity calculations required).
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.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>pIC50</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>ACTIVITY_CLASS</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>CompoundName</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>MDLPublicKeys</td>
<td>text</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "molvals_pkey" PRIMARY KEY, btree (id)

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.fps"
<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)
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
```
Matched pairs in SQL(simple)

```sql
select *, (pact1-pact2)/dist as disparity, pact2-pact1 as dact
from (select ms1.id id1, ms1.m smiles1, ms2.id id2, ms2.m smiles2, dist,
        -1*log(vs1.ic50_nm*1e-9) as pact1,
        -1*log(vs2.ic50_nm*1e-9) as pact2
     from (select fp1.id id1, fp2.id id2,
            1.0-dice_sml(fp1.torsionfp, fp2.torsionfp) as 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)>0.1
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
```
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_nom*1e-9) pact1,
    -1*log(vs2.ic50_nom*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 (complete)

```
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-dice_sml(fp1.torsionfp,fp2.torsionfp) dist,
    md5(subtract(fp1.torsionfp,fp2.torsionfp)::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)>.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

*0: cdk2 pairs 2

Row1: cdk2, c50_nm, 9 countfps, torsionfp, 0.85

*0:22: MatchedPairs (Count-Based)

Table Row To Variable
Node 20

Database Connector
Node 3

Database Column Filter
Node 2

Database Connection Reader
Node 4

Molecule Type Cast
Node 5

Molecule Type Cast
Node 11

Molecule to RDKIT
To RDKIT
Knime implementation

<table>
<thead>
<tr>
<th>Row ID</th>
<th>s4v hash</th>
<th>D</th>
<th>Mean(...)</th>
<th>First(mol1)</th>
<th>First(mol2)</th>
<th>I Count(...)</th>
<th>D</th>
<th>Max(pact)</th>
<th>D</th>
<th>Mean(dact)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row2649</td>
<td>d3049eaf3c90e2ea202380a780a5b...</td>
<td>50.461</td>
<td></td>
<td><img src="image1" alt="Molecule Image" /></td>
<td><img src="image2" alt="Molecule Image" /></td>
<td>6</td>
<td>7.699</td>
<td>-0.463</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Row937</td>
<td>4890255b136b311a2078798761e0...</td>
<td>51.404</td>
<td></td>
<td><img src="image3" alt="Molecule Image" /></td>
<td><img src="image4" alt="Molecule Image" /></td>
<td>7</td>
<td>7.824</td>
<td>-0.491</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Row1523</td>
<td>78e01712a4f11b2e7ad72b33b21d0...</td>
<td>56.469</td>
<td></td>
<td><img src="image5" alt="Molecule Image" /></td>
<td><img src="image6" alt="Molecule Image" /></td>
<td>8</td>
<td>8.155</td>
<td>-1.053</td>
<td>0</td>
<td></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.png" alt="mol1" /></td>
<td><img src="image2.png" alt="mol2" /></td>
</tr>
<tr>
<td>4890255b136b311a2078798761e00...</td>
<td>85.261</td>
<td>-0.812</td>
<td><img src="image3.png" alt="mol1" /></td>
<td><img src="image4.png" alt="mol2" /></td>
</tr>
<tr>
<td>4890255b136b311a2078798761e00...</td>
<td>30.155</td>
<td>-0.339</td>
<td><img src="image5.png" alt="mol1" /></td>
<td><img src="image6.png" alt="mol2" /></td>
</tr>
</tbody>
</table>
# Knime implementation

<table>
<thead>
<tr>
<th>S14v hash</th>
<th>Disparity</th>
<th>dact</th>
<th>mol1</th>
<th>mol2</th>
</tr>
</thead>
<tbody>
<tr>
<td>78e01712a4f11b2e7adf2b33b21d067b</td>
<td>93.59</td>
<td>-1.586</td>
<td><img src="image" alt="molecule1" /></td>
<td><img src="image" alt="molecule2" /></td>
</tr>
<tr>
<td>78e01712a4f11b2e7adf2b33b21d067b</td>
<td>69.981</td>
<td>-1.296</td>
<td><img src="image" alt="molecule1" /></td>
<td><img src="image" alt="molecule2" /></td>
</tr>
<tr>
<td>78e01712a4f11b2e7adf2b33b21d067b</td>
<td>65.55</td>
<td>-1.214</td>
<td><img src="image" alt="molecule1" /></td>
<td><img src="image" alt="molecule2" /></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)
Overview

- RDKit: what is it?
- RDKit + PostgreSQL
- RDKit + Knime
- Case study: matched pairs analysis
- Contributing to open source from big pharma
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
Thanks!

C++:
Core data structures and algorithms

RDKIT

Python
Boost.Python
PostgreSQL

Java
SWIG
Knime

???