RESEARCH

The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching

Egon L Willighagen[1*], John W Mayfield[2], Jonathan Alvarsson[3], Arvid Berg[3], Lars Carlsson[4], Nina Jeliazkova[5], Stefan Kuhn[6], Tomáš Pluskal[7], Miquel Rojas-Chertó[8], Ola Spjuth[3], Gilleain Torrance[9], Chris T Evelo[1], Rajarshi Guha[10] and Christoph Steinbeck[11]

Correspondence:
gon.willighagen@maastrichtuniversity.n
Dept of Bioinformatics -
iGCaT, NUTRIM, Maastricht
niversity, NL-6200 MD,
aastricht, The Netherlands
ull list of author information is
vailable at the end of the article

Abstract

Background: The Chemistry Development Kit (CDK) is a widely used open source cheminformatics toolkit, providing data structures to represent chemical concepts along with methods to manipulate such structures and perform computations on them. The library implements a wide variety of cheminformatics algorithms ranging from chemical structure canonicalization to molecular descriptor calculations and pharmacophore perception. It is used in drug discovery, metabolomics, and toxicology. Over the last ten years, the code base has grown significantly resulting in many complex interdependencies among components and poor performance of many algorithms.

Results: We report improvements to the CDK v2.0 since the v1.2 release series, specifically addressing the increased functional complexity and poor performance. We first summarize the addition of new, and improvements to existing functionality that has led to significant improvement in performance and support for important chemistry features including atom type perception, substructure searching, molecular fingerprints, rendering of molecules, and handling of molecular formulas. Second, we outline how the CDK has evolved with respect to quality control and the approaches we have adopted to ensure stability, including a code review mechanism.

Conclusions: This paper highlights our continued efforts to provide a community driven, open source cheminformatics library, and shows that such collaborative projects can thrive over extended periods of time, resulting in a high-quality and performant library. By taking advantage of community support and contributions, we show that an open source cheminformatics project can act as a peer reviewed publishing platform for scientific computing software.

Keywords: Java; cheminformatics; bioinformatics; metabolomics; depiction

Background

The open source cheminformatics community has made significant steps forward recently [1] as evidenced by the growing number of tools and underlying toolkits, along with the usage of these software components in a variety of applications. The Chemistry Development Kit (CDK) is one of the tools developed under the aegis of the Blue Obelisk and previously documented versions have been widely adopted [2, 3]. Use of the CDK ranges from inclusion of CDK functionality in wrapper platforms such as Cinfony [4], incorporation within the R environment (rcdk [5]), and as plugins for Taverna [6], KNIME [7], Cytoscape (ChemViz2 [8]), and for Microsoft Excel (LICSS [9]). In contrast to scenarios that have made CDK functionality available in larger systems, a number of projects have employed the CDK as a general cheminformatics toolkit. Examples include jCompoundMapper [10], ScaffoldHunter [11, 12], OMG [13], PaDEL [14], ChemDes [15], ReactPRED [16], SMSD [17, 18, 19], WhichCyp [20], MetaPrint2D [21], MetFrag [22], and the IUPHAR/BPS Guide to Pharmacology [23] and QSAR DataBank [24] databases. A number of such tools were initially developed using older versions of the CDK and are updated to new releases as they are made available. Examples include Bioclipse [25, 26] and AMBIT [27, 28, 29]. The CDK has also played a role in a number of chemical studies, such as finding the maximally bridging rings in chemical structures [30], prediction of organic reactions [31], and bioactivities of compounds [32].

While the CDK has purported to be a general purpose cheminformatics toolkit, older versions were designed by a community with specific applications in mind, primary among them being structure elucidation. In addition, an implicit goal of previous versions was to have the CDK serve as an educational resource to enable students of cheminformatics to understand the underlying algorithms. This resulted in certain functionalities, such as fingerprinting [33, 34], receiving more attention than others, such as stereochemistry. The outcome was significant variance in performance and features throughout the toolkit.

The growth of open source software over the last 10 years is evidence of the ability of communities of developers to develop systems and processes that lead to high quality software systems for long term use. The CDK is no different. The adoption of automatic build systems and quality control methodologies such as unit testing, automated source code validation, and peer review by fellow developers have greatly improved the stability of the library. While it has slowed development somewhat, it has allowed for cleaning up interdependencies between modules of functionality, and importantly, has improved the scalability of the development model. This has resulted in significant new functionality in core application programming interfaces (APIs) while maintaining the quality of code depending on those core APIs.

Examples of new features supported by the improved development model include InChI functionality [35], greatly improved ring detection algorithms [36], improvements to the core atom type perception module that now covers a much more comprehensive set of elements, charge states and radical species than previous versions, a more comprehensive fingerprinting API, new depiction functionality, and many speed and stability improvements.

Implementation and Results

This section describes the specifics of new APIs and improvements to pre-existing methods that are available in the latest CDK. We then discuss how we have improved and formalized the development model for the project using unit testing, code review and guidelines for handling version control. Finally we report on the availability of binary distributions of the library, allowing users to include specific modules (and their dependencies) of the CDK in their own projects (as opposed to developers who work on the CDK library itself).

New APIs and improved implementations

We here outline various new and improved APIs in the CDK library since the two previous publications in 2003 and 2006 [2, 3].

Atom Typing

Atom type perception is core cheminformatics functionality: the atom types describe chemical features of atoms, such as the number of neighbors, possible formal charges, (approximate) hybridization, electron distribution over orbitals and so on. However, previous versions of the CDK implemented atom type perception as part of different algorithms, resulting in duplicated and sometimes divergent typing schemes. As a result it was cumbersome to add new atom types and implement support for new charged and radical species in a consistent manner.

This CDK version has a new, centralized atom typing framework, removing the perception of atom types from various algorithms. This allows for a consistent and extensive typing scheme, that can be also be tested independently of other code. The new code defines the atom types using a list that specifies for each type the element symbol, hybridization, formal charge, number of lone pairs, and an enumeration of the bond orders (see Figure 1). This list of properties captures the information needed for the various algorithms in the CDK. For example, hybridization information can be used in certain aromaticity models (see later), and the lone pair information is needed for resonance structure calculation needed, for example, for Gasteiger π-charges.

A reference implementation, `CDKAtomTypeMatcher`, has been written that perceives these atom types, and validates the perception automatically against the properties defined by the ontology. This class handles a variety in types of missing information, as commonly resulting from various (file) formats; for example, it can handle undefined hydrogen counts and undefined double bond positions if hybridization information is provided instead. That makes the perception code flexible but also more complex. Alternative algorithms for atom typing have not been explored. This reference implementation can be used on a single atom:

```
for (IAtom atom : molecule.atoms()) {
  type = matcher.findMatchingAtomType(molecule, atom);
}
```

And on a full molecule, in which case the list of types is ordered in the same order as the atoms in the molecule object:

```
types = matcher.findMatchingAtomTypes(molecule);
```

Stereochemistry

Previous versions of the API represented stereochemistry in different ways, this hindered interconversion between and within file formats CDK v2.0 standardizes upon a new core representation and procedures have been updated or added to enable duplicate checking, pattern matching, and interconversion.

The preferred representation of stereochemistry is now for it to be stored at the molecule level as a `StereoElement`. In abstract terms a stereo element describes local geometry using a type, focus, carriers, and configuration (Figure 2). Currently the most common types of stereochemistry are supported: Tetrahedral, Cis-trans isomerism around a double bond, and Extended Tetrahedral. Rarer types of stereochemistry, such as: Square Planar, Trigonal Bipyramidal, Octahedral, could easily be incorporated into the chosen description given sufficient demand from the community.

Along with the new stereochemistry representation, algorithms were required in several areas. Generally a user does not need to invoke these procedures explicitly as they are called as needed within existing APIs:

- perception from 2D coordinates,
- perception from 3D coordinates,
- wedge assignment,
- graph (sub)isomorphism matching,
- SMARTS matching, and
- canonicalization.

The perception from coordinates and wedge assignment algorithms are fundamental for conversion between formats that store stereochemistry implicitly based on coordinates (e.g. molfile, CML) and explicitly (e.g. SMILES, CML, InChI). Perception from 2D coordinates can optionally identify perspective projects, specifically: Fischer, Haworth, and Chair projections. With the perception of perspective projections enabled, database entries currently consider distinct can be merged (Figure 3). The perception is based on an algorithm briefly described by [37].

Graph matching of stereochemistry with the described representation is straight forward. Given the atom-atom mapping from a query structure to a target molecule, the focus and carriers of the query stereochemistry are mapped to the target. Using the permutation parity of this mapping the configurations compared. SMARTS matching requires some special handling for complex cases [38]. For canonicalization, a partial canonical ordering is used to assign an absolute label which can then be integrated into the ordering. More detailed descriptions of these algorithms are described in detail in Chapter 6 of [39].

Atomic and Molecular Signatures

An implementation has been provided of the Signature structure descriptor for molecules [40]. These act as a linear notation - like the SMILES format - for the whole molecule as well as for connected substructures rooted at a single atom. The descriptor can also be canonicalized to provide isomorphism-independent representations [41]. Signatures of depth two can be calculated for atoms with:

```
for (IAtom atom : molecule.atoms()) {
  String signature = new AtomSignature(atom, 2, molecule).toCanonicalString();
}
```

But they can also can be calculated for full molecules:

```
String signature = new MoleculeSignature(molecule).toCanonicalString();
```

Finally, a signature fingerprint can be calculated for molecules, to allow similarity calculations. This can then be used in QSAR modeling [32, 42, 43, 44, 45, 46, 47].

Rendering API

A new rendering API has been introduced to make the rendering code independent from Java widget toolkits. The previous code was tightly linked to the Swing toolkit, but other tools use different widget toolkits. For example, Bioclipse is based on Eclipse which uses the Standard Widget Toolkit (SWT) [25].

A second new design goal was introduced to balance between size restrictions of some use cases, such as Java applets, and the rendering functionality. In particular, some functionality, even after modularization, needed considerable parts of the CDK library, making creation of a small-sized applet unfeasible. Therefore, the rendering API was modularized to allow splitting up rendering functionality into modules, with varying CDK dependencies.

Rendering is split up into several generation steps, previous versions split up bond from atom rendering, heteroatom symbols were simply drawn over lines representing bonds using a white rectangle to mask. A new `StandardGenerator` has been introduced that does bond and atom rendering at the same time, it incorporates many ideas described by Alex Clark [48, 49]. The depictions generated are of much higher quality and suitable for publication.

Moreover, a simplified high-level API has been introduced that addresses most of the common rendering needs, with the DepictionGenerator class. To depict a molecule loaded into a variable ‘benzene’ the following code can be used:

```
IAtomContainer benzene = ...;
new DepictionGenerator()
  .withSize(300, 300)
  .depict(benzene)
  .writeTo("benzene.png");
```

Many of the rendering options are available as parameters in the core API and as methods on the DepictionGenerator class. This includes substructure coloring, exemplified with an example reaction shown in Figure 4. When missing, 2D coordinates are generated on the fly with the new structure diagram layout functionality.

Structure Diagram Layout

The structure diagram layout has been improved and the new code solves a number of long standing issues. In particular, collision avoidance has been greatly improved. Figure 5 shows a difference in output between the old code base, with and without overlap resolving, and with the new refinement based implementation[50]. Generation of 2D coordinates is done as shown below:

```
IAtomContainer mol = ...;
StructureDiagramGenerator sdg = new StructureDiagramGenerator();
sdg.generateCoordinates(mol);
```

While the API itself has not been significantly changed, the internals have been revamped. In addition to improved overlap resolution noted above, the engine appropriately handles large ring systems, maintains input stereochemistry, and makes use of a large template library. Templates are useful for laying out substructure While previous CDK versions partially supported double bond stereochemistry the new engine is more efficient in using this information when generate 2D layouts. Furthermore, the engine assigns wedge bond information based on tetrahedral stereochemistry. These features are exemplified by the following code and the resulting layout depiction in Figure 6:

```
String smi = "C[C@@H](\N=C(/S)\Nc1ccccn1)C2CCCCC2 CHEMBL305150";
IAtomContainer mol = parser.parseSmiles(smi);
sdg.generateCoordinates(mol);
```

Molecular Formula

A chemical formula is the simplest chemical representation of a compound. It defines the number of isotopes or elements that compose a compound without describing how atoms are bonded. With the rise of metabolomics it has become increasingly relevant to have full support for these in cheminformatics libraries [22, 51, 52, 53, 54].

The CDK interfaces can handle several concepts related to chemical formulas: the formula itself, sets of formulas, chemical formula ranges, adducts, isotope containers and patterns, and rules to filter formula sets. These new tools can be used for a number of tasks, including calculating the isotopic pattern from a given chemical formula, determining the possible elemental compositions for a given mass (mass decomposition), and calculating the exact mass from a given chemical formula.

The CDK contains two algorithms for the decomposition of mass ranges into possible elemental formulas. For most inputs, a Round Robin algorithm, originally developed for the SIRIUS metabolite identification tool [55], is used. The algorithm discretizes the real-value mass decomposition problem into an integer-value knapsack problem [56]. It first computes a dynamic programming table and then backtracks it to generate matching formulas [57, 58]. Data for the Round Robin algorithm is stored in an extended residue table [59], resulting in a low memory footprint of several kilobytes. For certain problem instances, such as very large mass values (above 400 000 Da) or mass range span larger than 1 Da, the Round Robin algorithm is not suitable and CDK falls back to an optimized full enumeration search method, originally developed as part of the MZmine 2 framework for mass spectrometry data processing [52, 53].

The following code calculates all possible chemical formulas for a given accurate mass, within allowed counts for each element:

```
Isotopes ifac = Isotopes.getInstance();
MolecularFormulaRange range =
  new MolecularFormulaRange();
range.addIsotope( ifac.getMajorIsotope("C"), 8, 20);
range.addIsotope( ifac.getMajorIsotope("H"), 0, 20);
range.addIsotope( ifac.getMajorIsotope("O"), 0, 1);
range.addIsotope( ifac.getMajorIsotope("N"), 0, 1);

MolecularFormulaGenerator tool =
  new MolecularFormulaGenerator(
    SilentChemObjectBuilder.getInstance(),
    133.0, 133.1, range
  );
IMolecularFormulaSet mfSet = tool.getAllFormulas();
for (mf in mfSet) {
  println MolecularFormulaManipulator.getString(mf) + " " +
    MolecularFormulaManipulator.getTotalExactMass(mf)
}
```

This gives the following output:

```
C11H   133.007825032
C9H11N 133.089149352
C9H9O  133.065339908
C8H7NO 133.052763844
```

To evaluate the performance of the CDK molecular formula generator, we compared its runtimes to those of the classic, full enumeration-based HR2 formula generator [60] and those of a recently developed Parallel Formula Generator (PFG) [61] (Table 1). As inputs, we used two sets of 10 000 small ($<$ 500 Da) and 20 large ($>$ 1 500 and $<$ 3 500 Da) molecular mass values downloaded from the Global Natural Products Social Molecular Networking database [62]. The mass tolerance was set to 0.001 or 0.01 Da. The CDK v2.0's Round-Robin formula generator outperformed the other methods in all cases, despite running in a single thread (PFG utilizes multiple threads). The performance gain of the Round Robin algorithm was particularly apparent when narrow mass ranges were queried (e.g. ±0.001 Da), thus showing its suitability for applications in high-resolution mass spectrometry.

SMILES parser and generator

The SMILES parsing has been replaced by code from the external Beam project [63]. This BSD-licensed SMILES parser is a complete implementation of the SMILES and OpenSMILES (http://opensmiles.org/) specifications by one of the authors (including stereochemistry), and is independent of the CDK library. The SmilesParser API uses this library underneath, and the Beam API is hidden by this class. Basic usage is as follows:

```
  IChemObjectBuilder bldr
     = SilentChemObjectBuilder.getInstance()
```

```
SmilesParser       smipar = new SmilesParser(bldr);
IAtomContainer     mol    = smipar.parseSmiles('[nH]1cccc1');
```

The most significant functional change here is that the SMILES parser automatically locates the positions of double bonds in de-localised aromatic systems (Kekulisation). If this invariant cannot be met the SMILES is rejected as invalid. It is possible to override this check but this is strongly discouraged as rejected molecules do not have a fixed formula or tautomer.

The SMILES generation API has also been simplified and made more flexible able to produce several different flavours. The `SmiFlavor` flags are used to control the type of SMILES generated. Historically the terms: generic, isomeric, unique, absolute have been used in other toolkits and are also supported.

```
SmilesGenerator smigen = new SmilesGenerator(SmiFlavor.Canonical |
                                              SmiFlavor.StereoTetrahedral)
```

Support for ChemAxon Extended SMILES (CXSMILES)[64] layers has been added to CDK v2.0. CXSMILES provides a powerful means of including auxiliary information in a SMILES string such as 2D/3D coordinates, atom values, generic labels, repeat units, and positional variation. CXSMILES is achieving by placing additional information between pipe characters ('`|`') in the SMILES title field. Information is annotated based on the order of the atoms in the SMILES string. An example CXSMILES for a generic structure is shown below.

```
c1(:*:c2c(:*:c1*)C(N(N2)*)=O)* |$;Y;;;X;;R10;;;;Z;;R11$| US 2007/0129374 (I)
<- SMILES -------------------> <- CXSMILES ------------> <- Title --------->
```

Substructure and SMARTS matching

Substructure matching is fundamental cheminformatics operation and plays a key role in many other functions such as fingerprint and descriptor generation, and atom typing. Since CDK v1.2, functionality has been added to handle the SMARTS query language. The SMARTS language is supported well including features such as stereochemistry, component grouping, and atom maps (to match reaction transformations). A new *Pattern* API has been added to CDK v2.0, which simplifies finding, filtering, and transforming search results. The API is immutable allowing a pattern to be initialized once and then matched against several molecules or reactions across multiple threads. During initialization the pattern is inspected as to determine what invariants will be needed (e.g. ring size) and only required invariants are calculated. The internal matching algorithms provide a lazy iterator, such that the next match is only computed when it is needed. The API handles reactions in addition to molecules, and both can be specified as either queries or targets.

```
// initialize SMARTS pattern API
Pattern pattern =
  SmartsPattern.findSubstructure("O=[C,N]aa[N,O;!H0]");

IAtomContainer mol = ...;
IReaction      rxn = ...;

// check if the query matches, molecules and reactions
boolean mMatch = pattern.matches(mol);
boolean rMatch = pattern.matches(rxn);

// lazily iterate over all unique atom matches as query atom
// index bijection to atoms in ’mol’, ’rxn’ can also be used
for (int[] m : pattern.matchAll(mol)
                      .uniqueAtoms()) {
   ...
}
```

CDK v2.0 includes large improvements to algorithm efficiency. A simple benchmark experiment demonstrates the relative improvements between recent CDK versions and other open source toolkits (Table 5). The efficiency improvements are a combination of optimising data structures and key molecule processing algorithms (e.g. kekulisation and aromaticity) needed before a SMARTS match can be run.

Ring finding

Ring finding is another key functionality in a cheminformatics library, and the CDK knows a long history of ring finding [65, 36]. Specifically, non-redundant ring sets have seen particular interest, such as the smallest set of smallest rings, for which the

CDK implements two classical algorithms [66, 65]. Recent work has implemented a new, faster algorithm, allowing searching for various types of (non-redundant) ring sets [36]. These are available via the new Cycles API:

```
  allCycles = Cycles.all(mol)
  relevantCycles = Cycles.relevant(mol)
  essentialCycles = Cycles.essential(mol)
  sssrCycles = Cycles.sssr(mol)
```

Aromaticity

Aromaticity has seen many definitions in the past and for cheminformatics it frequently is algorithmically defined. The outcome of an aromaticity calculation depends on a number of atom type features and heuristics, which are often ambiguously defined in the published literature. Based on the information used, several different algorithmic definitions of aromaticity can be defined. Older CDK versions had various aromaticity models implemented but the code was scattered throughout the library, resulting in an inconsistent API to compute aromaticity and a significant maintenance burden. The API was unified in the current version, resulting in three models, of which two are based on the CDK atom typer. The difference between these two models is how contributions from exocyclic double bonds are handled.

The current CDK version further generalizes the idea that aromaticity is a model, and provides an API that allows the user to select one of several aromaticity models, leading to greater interoperability with other toolkits. The new `Aromaticity` class allows to build a custom model by selecting and combining options. For example, to reproduce the functionality of the previous `CDKAromaticity` class:

```
Aromaticity aromaticity = new Aromaticity(
  ElectronDonation.cdk(), Cycles.cdkAromaticSet()
);
```

Here, the CDK model for counting donated electrons is used, along with the rings systems that were identified by the older algorithm in previous versions that was limited in the number of fused rings systems that were considered. However, an alternative aromaticity calculator that considers all possible ring systems can now be easily created with:

```
Aromaticity aromaticity = new Aromaticity(
  ElectronDonation.cdk(), Cycles.all()
);
```

For SMARTS matching and SMILES generation a model based on Daylight [67] can be used and offers significant speed improvements to the one based on CDK Atom Types. This model has recently been documented as part of the OpenSMILES specification (http://opensmiles.org/):

```
Aromaticity aromaticity = new Aromaticity(
  ElectronDonation.daylight(), Cycles.all()
);
```

CTfile Format Improvements

The molfile format is still very popular and despite it being a proprietary format it has become a de facto standard. The format forms the core of the larger CTfile family which was originally developed by MDL Information Systems [68]. The current format specification is published by BIOVIA and available on request [69].

The CTAB block (connection table) of a molfile comes in two versions, V2000 and V3000. The V3000 provides several enhancements including but not limited to: removing atom and bond count limits, enhanced stereochemistry, and link nodes. For backwards compatibility V2000 is often preferred resulting in limited usage of V3000.

CDK v2.0 adds support for V3000 and has optimized and extended support for V2000. Currently these are considered separate formats requiring a user to know what version is being read beforehand. Future APIs will aim to simplify this and provide a unified reader. An overview of currently supported CTfile formats is given in Table 2.

CTfile Sgroups capture and organise high level information about sets of atoms and bonds [70]. There are four types of Sgroup: Display Short-cuts, Polymers, Mixtures, and Data. The most familiar Sgroups from an end user perspective are structure repeat units (e.g. bracketing) and abbreviations (Figure 8). CDK v2.0 adds supports for representation, reading, writing, and depiction of Sgroups.

New Object Builders

Originally, the CDK was developed as a shared library between JChemPaint [71] and Jmol [72, 73]. JChemPaint used a MVC approach with an event-passing mechanism to update the view when the model was changed. This can cause an cascade of change events being passed around. This was not always a desirable feature, especially for non-UI code. To address this, interfaces were introduced allowing multiple implementations of the core interfaces. With much code of the CDK library no longer based on the original data model, a builder is needed to create objects of that data model, such as an implementation of the IAtom. The new `IChemObjectBuilders` allow implementations to be created, allowing implementations of the interfaces to be instantiated without the need of explicitly referencing those implementations. This way, any algorithm implementation in the CDK can use any of the data model interface implementations.

The CDK v1.0 and v1.2 implementations of the `IChemObjectBuilder` had, however, one method for each data object constructor, resulting in a very large interface. Moreover, this interface API had to be updated each time a new class was introduced, and when existing methods changed and constructors were updated. To simplify the API, the new `IChemObjectBuilder` collapses all methods into a single method, which takes as a first parameter the class of the interface that is to be constructed. All further parameters are passed as parameters to the class constructor.

For example, to construct a new atom from its element symbol, one would write previously:

```
IChemObjectBuilder builder = ...;
IAtom atom = builder.newAtom("C");
```

With the new builder, the code looks like:

```
IChemObjectBuilder builder = ...;
IAtom atom = builder.newInstance(IAtom.class, "C");
```

The CDK library is now mostly written that it no longer depends on a specific implementation of the `IChemObjectBuilder`, allowing the user of the CDK to select a builder suitable to their software. Therefore, if software depends on event passing, then the `DefaultChemObjectBuilder` can be used, in most cases this isn't needed and the `SilentChemObjectBuilder` is preferred resulting in a typical speed up of 10%-20%:

```
IChemObjectBuilder builder = SilentChemObjectBuilder.getInstance();
```

The third builder is the `DataDebugChemObjectBuilder` which generates debug information for all changes to the content of the data classes. This can be useful for debugging and other forms of code inspection.

Molecular Fingerprints

Molecular fingerprints have also seen significant development in this CDK version. Previously, fingerprints were represented using the `BitSet` class from the Java library. While using this class allowed the use of pre-existing methods to manipulate bit strings, it keeps a vector of bits in memory. The solution was excellent for hashed, relatively small fingerprints, *e.g.*, 1024 bits, *i.e.* with a 2^{10} indexing space (128 B). However, implementing a fingerprint designed to avoid collisions with a 2^{32} bit indexing space using this approach would be memory-inefficient (512 MiB). To allow for multiple fingerprint representations, a bit fingerprint interface was introduced: `IBitFingerprint`.

```
IFingerprinter  ecfp = new CircularFingerprinter();
IBitFingerprint fp   = ecfp.getBitFingerprint(mol);
```

Also, although fingerprints traditionally are bit vectors a count fingerprint was also introduced making fingerprints based on integer vectors supported in CDK as well. The counts in the fingerprint then represent how often this substructure is found in the molecule it represents.

```
IFingerprinter    ecfp = new CircularFingerprinter(CLASS_ECFP4);
ICountFingerprint fp   = ecfp.getCountFingerprint(mol);
```

The fingerprints currently provided by the CDK are listed in Table 3.

Improved Coding Standards

As the CDK library grew over the years, so did the complexity of the maintenance. The main branch frequently failed to compile and bug fixes became more onerous due to unexpected side effects. Often fixing a bug in one part of the code, broke some other code which made the incorrect assumptions about the fixed code. With the increased size of the CDK developer community, such issues were inevitable in the absence of any formal coding and testing standards.

To address these issues, we have adopted a number of coding standards. While not a comprehensive implementation of software engineering best practices, they attempt to find a balance between increasing code maintainability and being flexible enough to allow efficient code development. We appreciate the subjective nature of this statement, and some adopted guidelines have been heavily discussed and debated in the CDK community.

Arguably, perhaps the biggest factor in improved code quality is a peer review process where any functionality changing patch is required to be reviewed by one independent, senior CDK developer for the development branch, and by two reviewers for stable branches. This patch development system is supported by a number of automated validations steps as outlined below. The next sections describe some approaches the project have adopted that allows us to maintain the CDK library as it is today.

Modularization

One of the central approaches we have adopted, is to make the CDK more modular. The CDK assigns every class to a module, and defines dependencies between modules. For example, core modules are not allowed to depend on modules with data classes implementing the CDK interfaces; instead, they may only depend on the interfaces themselves. This ensures that dependencies are minimized. Furthermore, it also allows cherrypicking CDK functionality, reducing the number of third-party library dependencies that are needed. An overview of key modules with description, important changes, and dependencies on third-party libraries is given in Table 4 and the dependencies between the CDK modules are depicted in Figure 7.

Documentation

The quality of the JavaDocs was originally tested with DocCheck, and later replaced by a custom written tool called OpenJavaDocCheck. With the move to Maven (explained later), which does not have integration for this tool, we adopted CheckStyle (http://checkstyle.sourceforge.net/). This tool reports on missing documentation and on documentation which is not properly annotated in the Java source files.

Testing

Years of development of the CDK library has resulted in a large suite of tests of various kinds. This include unit tests, which test core APIs, and functional testing, which test higher level functionality of the CDK. The latter include tests if algorithm implementations calculate the expected values, but also contain integrated tests, which involve more than one algorithm, such as SMILES parsing. The suite consists of more than 23 thousand tests.

Code Quality

The project continues to use PMD (http://pmd.sf.net/) for code quality checking, but deviates from the default rules. For example, we are more liberal with variable name length. Moreover, a number of additional PMD tests have been developed specifically for the CDK, that, for example, test if a class uses the core interfaces instead of implementations of those interfaces. That is, that the code uses `IAtom` instead of `Atom`. However, these tests do generate a few false positives, as the tests check the class name only, and not the Java package the class is in.

Git, branching, and patches

Older versions of the CDK employed Subversion for version control. A few years back, the project switched to the Git version control system. A key advantage of this shift is the ability to have distributed repositories, easier branching and provision for patches. GitHub (https://github.com/cdk/cdk) has replaced SourceForge as the main source code hosting service where we can use novel approaches for commenting on code (peer review), pull requests, etc. These new features simplify our code review process.

Binary distributions

Maven packages

The build system has been converted from Ant to Maven. The shift was motivated by the easier dependency handling, cleaner separation of testing code from the main library and automated packaging. The move to modules necessitated splitting the original monolithic source code tree in to per-module source folders. While this makes the on-disk layout of the source code more complex, this is usually hidden by modern IDEs.

As a result for many modules, the test code is now more closely linked to the code being tested: both reside in the same folder, though we adhere to the Maven custom to have `src/main/java` and a `src/test/java` folders. For a few modules, however, this solution introduces circular dependencies, in which case a separate Maven module is created for the tests.

The Maven packages for the CDK are available from Maven Central, which makes it easy for other projects to use. The full library can be included in other software by depending on the cdk artifact (http://search.maven.org/#search|ga|1|org.openscience) but dependencies can also be defined on individual CDK modules.

OSGi bundles

OSGi bundles are available for the CDK too, which are used by e.g. Bioclipse [25, 26] and KNIME [7]. However, because CDK Java packages are occassionally split between CDK modules, the CDK currently needs to be bundled as a single OSGi jar. The bundle is available from http://pele.farmbio.uu.se/bioclipse/cdk/cdk-1.5.13/. This Java package and bundle incompatibilities are currently being explored and constitutes an area where improvements can be done on modularization.

Conclusions

Since the second CDK publication, in 2006, the library has been improved in many aspects including architecture, new functionality, improved code testing, management, peer review, and deployment. These changes have led a more functionally rich cheminformatics library, with significant performance improvements. Updates on the common SMILES and molfile formats and the improved structure diagram generation are very visible and benefit many of the tools using the CDK. Furthermore, the stability of the development model has significantly improved, providing greater stability of the library over time. With more than 90 contributors, a long list of tools based on the CDK, and hundreds of article citations, the CDK is alive and kicking.

Availability and requirements

- **Project Name**: The Chemistry Development Kit
- **Project home page**: https://github.com/cdk/cdk
- **Operating system(s)**: Windows, GNU/Linux, OS/X
- **Programming language**: Java
- **Other (optional) requirements**: JNI-InChI, Vecmath, Beam, Guava, JGraphT, Signatures, CMLXOM, XOM, JavaCC
- **License**: LGPL v2.1 or later
- **Any restrictions to use by non-academics**: None additional

Competing interests

JWM and NJ work for companies that sell solutions based on the CDK. ELW sells a book describing the CDK functionality.

Authors contributions

All authors wrote and contributed source code or documentation to the CDK library. Some authors have peer-reviewed source code for the library. ELW, JWM, RG, and CS are project leaders. All authors have contributed to the content of this paper and approved the final version.

Acknowledgments

The authors acknowledge the great number of people who have contributed smaller and larger contributions to the CDK library. A full list of contributors is found in the Maven parent POM [74]. OS acknowledges support from the Swedish strategic research programs eSSENCE and Swedish e-Science Research Center (SeRC). TP is a Simons Foundation Fellow of the Helen Hay Whitney Foundation. We also thank K. Dührkop for his contributions during the writing of this paper.

Author details

[1] Dept of Bioinformatics - BiGCaT, NUTRIM, Maastricht University, NL-6200 MD, Maastricht, The Netherlands. [2] NextMove Software Ltd, CB4 0EY, Cambridge, UK. [3] Department of Pharmaceutical Biosciences, Uppsala University, 751 24, Uppsala, Sweden. [4] AstraZeneca, Innovative Medicines & Early Development, Quantitative Biology, Mölndal, SE. [5] Ideaconsult Ltd, A. Kanchev 4, 1000, Sofia, Bulgaria. [6] Department of Informatics, University of Leicester, Leicester, UK. [7] Whitehead Institute for Biomedical Research, 9 Cambridge Center, MA 02142, Cambridge, USA. [8] Química Clínica Aplicada, 43870, Amposta, Spain. [9] 4 Hanway Place, W1T 1HD, London, UK. [10] National Center for Advancing Translational Science, 9800 Medical Center Drive, MD 20878, Rockville, USA. [11] Metabolomics and Molecular Informatics, EMBL-EBI, Hinxton, UK.

References

1. O'Boyle, N., Guha, R., Willighagen, E., Adams, S., Alvarsson, J., Bradley, J.C., Filippov, I., Hanson, R., Hanwell, M., Hutchison, G., James, C., Jeliazkova, N., Lang, A., Langner, K., Lonie, D., Lowe, D., Pansanel, J., Pavlov, D., Spjuth, O., Steinbeck, C., Tenderholt, A., Theisen, K., Murray-Rust, P.: Open Data, Open Source and Open Standards in chemistry: The Blue Obelisk five years on. Journal of Cheminformatics **3**(1), 37 (2011)

2. Steinbeck, C., Han, Y., Kuhn, S., Horlacher, O., Luttmann, E., Willighagen, E.: The Chemistry Development Kit (CDK): an open-source Java library for Chemo- and Bioinformatics. J Chem Inf Comput Sci **43**(2), 493–500 (2003)
3. Steinbeck, C., Hoppe, C., Kuhn, S., Floris, M., Guha, R., Willighagen, E.L.: Recent developments of the Chemistry Development Kit (CDK) - an open-source java library for chemo- and bioinformatics. Current Pharmaceutical Design **12**(17), 2111–2120 (2006)
4. O'Boyle, N.M., Hutchison, G.R.: Cinfony - combining open source cheminformatics toolkits behind a common interface. Chemistry Central Journal **2** (2008)
5. Guha, R.: Chemical informatics functionality in R. Journal of Statistical Software **18**(5), 1–16 (2007)
6. Truszkowski, A., Jayaseelan, K.V., Neumann, S., Willighagen, E.L., Zielesny, A., Steinbeck, C.: New developments on the cheminformatics open workflow environment CDK-Taverna. Journal of Cheminformatics **3**(1), 1–10 (2011)
7. Beisken, S., Meinl, T., Wiswedel, B., de Figueiredo, L., Berthold, M., Steinbeck, C.: KNIME-CDK: Workflow-driven cheminformatics. BMC Bioinformatics **14**(1), 257 (2013)
8. ChemViz2: Cheminformatics App for Cytoscape (2016). http://www.rbvi.ucsf.edu/cytoscape/chemViz2/
9. Lawson, K.R., Lawson, J.: LICSS - a chemical spreadsheet in microsoft excel. Journal of Cheminformatics **4**(1), 3 (2012)
10. Hinselmann, G., Rosenbaum, L., Jahn, A., Fechner, N., Zell, A.: jCompoundMapper: An open source java library and command-line tool for chemical fingerprints. Journal of Cheminformatics **3**(1), 3 (2011)
11. Wetzel, S., Klein, K., Renner, S., Rauh, D., Oprea, T.I., Mutzel, P., Waldmann, H.: Interactive exploration of chemical space with Scaffold Hunter. Nature Chemical Biology **5**(8), 581–583 (2009)
12. Klein, K., Koch, O., Kriege, N., Mutzel, P., Schäfer, T.: Visual analysis of biological activity data with scaffold hunter. Molecular Informatics **32**(11-12), 964–975 (2013)
13. Peironcely, J.E., Rojas-Chertó, M., Fichera, D., Reijmers, T., Coulier, L., Faulon, J.-L., Hankemeier, T.: OMG: open molecule generator. Journal of Cheminformatics **4**(1), 1–13 (2012)
14. Yap, C.W.: PaDEL-descriptor: An open source software to calculate molecular descriptors and fingerprints. Journal of Computational Chemistry **32**(7), 1466–1474 (2011)
15. Dong, J., Cao, D.-S., Miao, H.-Y., Liu, S., Deng, B.-C., Yun, Y.-H., Wang, N.-N., Lu, A.-P., Zeng, W.-B., Chen, A.F.: ChemDes: an integrated web-based platform for molecular descriptor and fingerprint computation. Journal of Cheminformatics **7**(1) (2015)
16. Sivakumar, T.V., Giri, V., Park, J.H., Kim, T.Y., Bhaduri, A.: ReactPRED: a tool to predict and analyze biochemical reactions. Bioinformatics, 491 (2016)
17. Rahman, S.A., Bashton, M., Holliday, G.L., Schrader, R., Thornton, J.M.: Small molecule subgraph detector (SMSD) toolkit. Journal of Cheminformatics **1**(1), 12 (2009)
18. Rahman, S.A., Cuesta, S.M., Furnham, N., Holliday, G.L., Thornton, J.M.: EC-BLAST: a tool to automatically search and compare enzyme reactions. Nature Methods **11**(2), 171–174 (2014)
19. Rahman, S.A., Torrance, G., Baldacci, L., Cuesta, S.M., Fenninger, F., Gopal, N., Choudhary, S., May, J.W., Holliday, G.L., Steinbeck, C., Thornton, J.M.: Reaction Decoder Tool (RDT): extracting features from chemical reactions. Bioinformatics **32**(13), 2065–2066 (2016)
20. Rostkowski, M., Spjuth, O., Rydberg, P.: WhichCyp: prediction of cytochromes p450 inhibition. Bioinformatics **29**(16), 2051–2052 (2013)
21. Carlsson, L., Spjuth, O., Adams, S., Glen, R.C., Boyer, S.: Use of historic metabolic biotransformation data as a means of anticipating metabolic sites using MetaPrint2D and bioclipse. BMC Bioinformatics **11**(1), 362 (2010)
22. Wolf, S., Schmidt, S., Müller-Hannemann, M., Neumann, S.: In silico fragmentation for computer assisted identification of metabolite mass spectra. BMC Bioinformatics **11**(1), 148 (2010)
23. Southan, C., Sharman, J.L., Benson, H.E., Faccenda, E., Pawson, A.J., Alexander, S.P.H., Buneman, O.P., Davenport, A.P., McGrath, J.C., Peters, J.A., Spedding, M., Catterall, W.A., Fabbro, D., Davies, J.A., NC-IUPHAR: The IUPHAR/BPS guide to PHARMACOLOGY in 2016: towards curated quantitative interactions between 1300 protein targets and 6000 ligands. Nucleic Acids Research **44**(D1), 1054–1068 (2016)
24. Ruusmann, V., Sild, S., Maran, U.: QSAR DataBank repository: open and linked qualitative and quantitative structure–activity relationship models. Journal of Cheminformatics **7**(1) (2015)

25. Spjuth, O., Helmus, T., Willighagen, E.L., Kuhn, S., Eklund, M., Wagener, J., Murray-Rust, P., Steinbeck, C., Wikberg, J.E.: Bioclipse: an open source workbench for chemo-and bioinformatics. BMC Bioinformatics **8**(1), 59 (2007)
26. Spjuth, O., Alvarsson, J., Berg, A., Eklund, M., Kuhn, S., Mäsak, C., Torrance, G., Wagener, J., Willighagen, E.L., Steinbeck, C., *et al.*: Bioclipse 2: A scriptable integration platform for the life sciences. BMC Bioinformatics **10**(1), 397 (2009)
27. Jeliazkova, N., Jeliazkov, V.: AMBIT RESTful web services: an implementation of the OpenTox application programming interface. Journal of Cheminformatics **3**(1), 1–18 (2011)
28. Jeliazkova, N., Kochev, N.: AMBIT-SMARTS: Efficient Searching of Chemical Structures and Fragments. Molecular Informatics **30**(8), 707–720 (2011)
29. Kochev, N.T., Paskaleva, V.H., Jeliazkova, N.: Ambit-tautomer: An open source tool for tautomer generation. Molecular Informatics **32**(5-6), 481–504 (2013)
30. Marth, C.J., Gallego, G.M., Lee, J.C., Lebold, T.P., Kulyk, S., Kou, K.G.M., Qin, J., Lilien, R., Sarpong, R.: Network-analysis-guided synthesis of weisaconitine d and liljestrandinine. Nature **528**(7583), 493–498 (2015)
31. Segler, M.H.S., Waller, M.P.: Modelling Chemical Reasoning to Predict Reactions (2016). 1608.07117. http://arxiv.org/abs/1608.07117
32. Alvarsson, J., Lampa, S., Schaal, W., Andersson, C., Wikberg, J.E.S., Spjuth, O.: Large-scale ligand-based predictive modelling using support vector machines. Journal of Cheminformatics **8**(1) (2016)
33. Clark, A., Sarker, M., Ekins, S.: New target prediction and visualization tools incorporating open source molecular fingerprints for tb mobile 2.0. Journal of Cheminformatics **6**(1), 38 (2014)
34. Cannon, E., Mitchell, J.B.O.: Classifying the world Anti-Doping agency's 2005 prohibited list using the Chemistry Development Kit fingerprint. In: Berthold, Glen, R., Fischer, I. (eds.) Computational Life Sciences II. Lecture Notes in Computer Science, vol. 4216, pp. 173–182. Springer, Heidelberg, Germany (2006)
35. Spjuth, O., Berg, A., Adams, S., Willighagen, E.L.: Applications of the InChI in cheminformatics with the CDK and bioclipse. Journal of Cheminformatics **5**(1), 14 (2013)
36. May, J.W., Steinbeck, C.: Efficient ring perception for the Chemistry Development Kit. Journal of Cheminformatics **6**(1), 3 (2014)
37. Karapetyan, K., Batchelor, C., Sharpe, D., Tkachenko, V., Williams, A.: The Chemical Validation and Standardization Platform (CVSP): large-scale automated validation of chemical structure datasets. Journal of Statistical Software **7**(30) (2015)
38. May, J.W.: Mischievous SMARTS Queries (2014). http://efficientbits.blogspot.co.uk/2014_03_01_archive.html
39. May, J.W.: Cheminformatics for genome-scale metabolic reconstructions. University of Cambridge (2015). https://www.repository.cam.ac.uk/handle/1810/246652
40. Faulon, J.-L., Visco, J. Donald P., Pophale, R.S.: The signature molecular descriptor. 1. using extended valence sequences in QSAR and QSPR studies. Journal of Chemical Information and Computer Sciences **43**(3), 707–720 (2003)
41. Faulon, J.-L., Collins, M.J., Carr, R.D.: The signature molecular descriptor. 4. canonizing molecules using extended valence sequences. Journal of Chemical Information and Computer Sciences **44**(2), 427–436 (2004)
42. Alvarsson, J., Eklund, M., Engkvist, O., Spjuth, O., Carlsson, L., Wikberg, J.E.S., Noeske, T.: Ligand-based target prediction with signature fingerprints. Journal of Chemical Information and Modeling **54**(10), 2647–2653 (2014)
43. Spjuth, O., Eklund, M., Ahlberg Helgee, E., Boyer, S., Carlsson, L.: Integrated decision support for assessing chemical liabilities. Journal of Chemical Information and Modeling **51**(8), 1840–7 (2011)
44. Moghadam, B.T., Alvarsson, J., Holm, M., Eklund, M., Carlsson, L., Spjuth, O.: Scaling predictive modeling in drug development with cloud computing. Journal of Chemical Information and Modeling **55**(1), 19–25 (2015)
45. Alvarsson, J., Eklund, M., Andersson, C., Carlsson, L., Spjuth, O., Wikberg, J.E.S.: Benchmarking study of parameter variation when using signature fingerprints together with support vector machines. Journal of Chemical Information and Modeling **54**(11), 3211–7 (2014)
46. Spjuth, O., Carlsson, L., Alvarsson, J., Georgiev, V., Willighagen, E., Eklund, M.: Open source drug discovery with Bioclipse. Current Topics in Medicinal Chemistry **12**(18), 1980–6 (2012)

47. Norinder, U., Ek, M.E.: QSAR investigation of NaV1.7 active compounds using the SVM/signature approach and the Bioclipse modeling platform. Bioorg Med Chem Lett **23**(1), 261–3 (2013)
48. Clark, A.M.: Basic primitives for molecular diagram sketching. Journal of Cheminformatics **2**(1), 8 (2010)
49. Clark, A.M.: Rendering molecular sketches for publication quality output. Molecular Informatics **32**(3), 291–301 (2013)
50. Helson, H.E.: Structure Diagram Generation, pp. 313–398. John Wiley & Sons, Inc., Oxford, UK (2007)
51. Rojas-Chertó, M., Kasper, P.T., Willighagen, E.L., Vreeken, R.J., Hankemeier, T., Reijmers, T.H.: Elemental composition determination based on MSn. Bioinformatics **27**(17), 2376–2383 (2011)
52. Pluskal, T., Uehara, T., Yanagida, M.: Highly accurate chemical formula prediction tool utilizing high-resolution mass spectra, MS/MS fragmentation, heuristic rules, and isotope pattern matching. Analytical Chemistry **84**(10), 4396–4403 (2012)
53. Pluskal, T., Castillo, S., Villar-Briones, A., Orešič, M.: MZmine 2: modular framework for processing, visualizing, and analyzing mass spectrometry-based molecular profile data. BMC Bioinformatics **11**(1), 1–11 (2010)
54. Dührkop, K., Shen, H., Meusel, M., Rousu, J., Böcker, S.: Searching molecular structure databases with tandem mass spectra using CSI:FingerID. Proceedings of the National Academy of Sciences **112**(41), 12580–12585 (2015)
55. Böcker, S., Letzel, M.C., Lipták, Z., Pervukhin, A.: SIRIUS: decomposing isotope patterns for metabolite identification. Bioinformatics **25**(2), 218–224 (2009)
56. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc., New York, NY, USA (1990)
57. Dührkop, K., Ludwig, M., Meusel, M., Böcker, S.: Faster mass decomposition. In: Proc. of Workshop on Algorithms in Bioinformatics (WABI 2013), pp. 45–58 (2013). Springer. http://arxiv.org/abs/1307.7805
58. Böcker, S., Lipták, Z., Martin, M., Pervukhin, A., Sudek, H.: DECOMP—from interpreting mass spectrometry peaks to solving the money changing problem. Bioinformatics **24**(4), 591–593 (2008)
59. Böcker, S., Lipták, Z.: Efficient mass decomposition. In: Proceedings of the 2005 ACM Symposium on Applied Computing, pp. 151–157 (2005). ACM
60. Kind, T., Fiehn, O.: Seven Golden Rules for heuristic filtering of molecular formulas obtained by accurate mass spectrometry. BMC Bioinformatics **8**(1), 1–20 (2007)
61. Zhang, M., Zhang, Z., Chen, C., Lu, H., Liang, Y.: Parallel formula generator based on branch-and-bound algorithm for elucidating high resolution mass spectra. Chemometrics and Intelligent Laboratory Systems **153**, 106–109 (2016)
62. Wang, M., Carver, J.J., Phelan, V.V., Sanchez, L.M., Garg, N., Peng, Y., Nguyen, D.D., Watrous, J., Kapono, C.A., Luzzatto-Knaan, T., Porto, C., Bouslimani, A., Melnik, A.V., Meehan, M.J., Liu, W.-T., Crusemann, M., Boudreau, P.D., Esquenazi, E., Sandoval-Calderon, M., Kersten, R.D., Pace, L.A., Quinn, R.A., Duncan, K.R., Hsu, C.-C., Floros, D.J., Gavilan, R.G., Kleigrewe, K., Northen, T., Dutton, R.J., Parrot, D., Carlson, E.E., Aigle, B., Michelsen, C.F., Jelsbak, L., Sohlenkamp, C., Pevzner, P., Edlund, A., McLean, J., Piel, J., Murphy, B.T., Gerwick, L., Liaw, C.-C., Yang, Y.-L., Humpf, H.-U., Maansson, M., Keyzers, R.A., Sims, A.C., Johnson, A.R., Sidebottom, A.M., Sedio, B.E., Klitgaard, A., Larson, C.B., Boya P, C.A., Torres-Mendoza, D., Gonzalez, D.J., Silva, D.B., Marques, L.M., Demarque, D.P., Pociute, E., O'Neill, E.C., Briand, E., Helfrich, E.J.N., Granatosky, E.A., Glukhov, E., Ryffel, F., Houson, H., Mohimani, H., Kharbush, J.J., Zeng, Y., Vorholt, J.A., Kurita, K.L., Charusanti, P., McPhail, K.L., Nielsen, K.F., Vuong, L., Elfeki, M., Traxler, M.F., Engene, N., Koyama, N., Vining, O.B., Baric, R., Silva, R.R., Mascuch, S.J., Tomasi, S., Jenkins, S., Macherla, V., Hoffman, T., Agarwal, V., Williams, P.G., Dai, J., Neupane, R., Gurr, J., Rodriguez, A.M.C., Lamsa, A., Zhang, C., Dorrestein, K., Duggan, B.M., Almaliti, J., Allard, P.-M., Phapale, P., Nothias, L.-F., Alexandrov, T., Litaudon, M., Wolfender, J.-L., Kyle, J.E., Metz, T.O., Peryea, T., Nguyen, D.-T., VanLeer, D., Shinn, P., Jadhav, A., Muller, R., Waters, K.M., Shi, W., Liu, X., Zhang, L., Knight, R., Jensen, P.R., Palsson, B.O., Pogliano, K., Linington, R.G., Gutierrez, M., Lopes, N.P., Gerwick, W.H., Moore, B.S., Dorrestein, P.C., Bandeira, N.: Sharing and community curation of mass spectrometry data with global natural products social molecular networking. Nature Biotechnology **34**(8), 828–837 (2016)
63. May, J.W.: Beam. GitHub (2013). https://github.com/johnmay/beam

64. ChemAxon Extended SMILES. http://onlinelibrarystatic.wiley.com/marvin/help/formats/cxsmiles-doc.html
65. Berger, F., Flamm, C., Gleiss, P.M., Leydold, J., Stadler, P.F.: Counterexamples in chemical ring perception. Journal of Chemical Information and Computer Sciences **44**(2) (2004)
66. Figueras, J.: Ring perception using Breadth-First search. Journal of Chemical Information and Computer Sciences **36**(5), 986–991 (1996)
67. Daylight Chemical Information Systems Inc. http://www.daylight.com
68. Dalby, A., Nourse, J.G., Hounshell, W.D., Gushurst, A.K.I., Grier, D.L., Leland, B.A., Laufer, J.: Description of several chemical structure file formats used by computer programs developed at molecular design limited. Journal of Chemical Information and Computer Sciences **32**(3), 244–255 (1992)
69. CTfile Formats. http://accelrys.com/products/collaborative-science/biovia-draw/ctfile-no-fee.html
70. Gushurst, A.J., Nourse, J.G., Hounshell, W.D., Leland, B.A., Raich, D.G.: The substance module: the representation, storage, and searching of complex structures. Journal of Chemical Information and Computer Sciences **31**(4), 447–454 (1991)
71. Krause, S., Willighagen, E., Steinbeck, C.: JChemPaint - using the collaborative forces of the internet to develop a free editor for 2D chemical structures. Molecules **5**(1), 93–98 (2000)
72. Willighagen, E., Howard, M.: Fast and scriptable molecular graphics in web browsers without Java3D. Nature Precedings (713) (2007)
73. Hanson, R.M.: Jmol - a paradigm shift in crystallographic visualization. Journal of Applied Crystallography **43**, 1250–1260 (2010)
74. AUTHORS (2015). https://github.com/cdk/cdk/blob/master/pom.xml
75. Rogers, D., Hahn, M.: Extended-connectivity fingerprints. Journal of Chemical Information and Modeling **50**(5), 742–754 (2010)
76. Hall, L.H., Kier, L.B.: Electrotopological State Indices for Atom Types: A Novel Combination of Electronic, Topological, and Valence State Information. Journal of Chemical Information and Modeling **35**, 1039–1045 (1995)
77. Klekota, J., Roth, F.P.: Chemical substructures that enrich for biological activity. Bioinformatics **24**(21), 2518–25 (2008)
78. Vidal, D., Thormann, M., Pons, M.: Lingo, an efficient holographic text based method to calculate biophysical properties and intermolecular similarities. Journal of Chemical Information and Modeling **45**(2), 386–393 (2005)
79. PubChem Substructure Fingerprint v1.3. ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.txt
80. Murray-Rust, P., Rzepa, H.S.: CML: Evolution and design. Journal of Cheminformatics **3**(1), 44 (2011)
81. Ihlenfeldt, W.D., Gasteiger, J.: Hash codes for the identification and classification of molecular structure elements. Journal of Computational Chemistry **15**(8), 793–813 (1994)
82. Hicklin, J., Moler, C., Webb, P., Boisvert, R.F., Miller, B., Pozo, R., Remington, K.: JAMA: A Java Matrix Package (2012). http://math.nist.gov/javanumerics/jama/
83. O'Boyle, N.M., Banck, M., James, C.A., Morley, C., Vandermeersch, T., Hutchison, G.R.: Open Babel: An open chemical toolbox. Journal of Cheminformatics **3**(1), 1–14 (2011)
84. RDKit: Open-source cheminformatics. http://www.rdkit.org

Figures

Figure 1 Atom type information specified for a sp3-hybridized carbon.

```
<at:AtomType rdf:ID="C.sp3">
  <at:categorizedAs rdf:resource="&cdkat;C.tetrahedral"/>
  <at:hasElement rdf:resource="&elem;C"/>
  <at:hybridization rdf:resource="&at;sp3"/>
  <at:formalCharge>0</at:formalCharge>
  <at:lonePairCount>0</at:lonePairCount>
  <at:formalBondType rdf:resource="&bo;single"/>
  <at:formalBondType rdf:resource="&bo;single"/>
  <at:formalBondType rdf:resource="&bo;single"/>
  <at:formalBondType rdf:resource="&bo;single"/>
</at:AtomType>
```

Figure 2 Relative storage of stereochemistry, the type and focus of stereochemsitry are fixed for a given stereocenter description but the carriers and configuration are relative. The multiple rows for each stereochemistry type are different internal representation that would be considered equivalent. In the tetrahedral types, hydrogens may be suppressed in a molecular graph so the **focus** is reused in the **carriers** list as a placeholder.

Figure 3 The raw input files of CHEMBL23970 and CHEMBL444314 are displayed (ChEMBL 21). Without perceiving the stereochemistry indicated by Haworth projection in CHEMBL23970, the database entries are incorrectly considered distinct. Down stream aggregation databases mirror this separation (PubChem CID 5280, CID 65119).

Figure 4 Integrated example showing the rendering and SMILES parsing functionality. Example from U.S. Patent US 2014 231770 A1 para 287.

Figure 5 The improved structure diagram generation has improved code to solve overlap. The original SDG code used general heuristics (left) and the OverlapResolver would fine tune the layout to ensure atoms would not be placed at the same location (middle). The new SDG algorithm is able to make more rigorous changes, making the final output must more pleasing (right).

Figure 6 Structure diagram generation for structures with double bond and tetrahedral stereochemistry.

Figure 7 Dependencies between CDK modules. Visualization of the dependencies between CDK modules. For example, the cdk-core depends on the cdk-interfaces module. A few higher level modules have been left out: cdk-builder3dtools, cdk-legacy, and cdk-depict.

Figure 8 Examples of Sgroups now captured by the CDK and encoded in molfiles and CXSMILES. a) Ethyl esterification fully expanded reaction. b) Using Sgroup abbreviations allows display short cuts and more compact depiction. c) An example of a structure repeat unit in DNA 5'-phosphate (CHEBI:4294)

Tables

Table 1 - Evaluation of molecular formula generators.

The resulting formula counts and runtimes of the HR2, PFG, and CDK chemical formula generators on two different inputs with two different mass tolerance settings. For the set of small masses, 10,000 mass values in the range of 0–500 Da were randomly selected from the Global Natural Products Social Molecular Networking database [62]. For the set of large masses, 20 mass values in the range of 1,500–3,500 Da were randomly selected from the same database. Formulas were generated using chemical elements C,H,N,O,P,S without bounds (the allowed atom count was set to 0–10,000 for each element). All heuristic filtering rules were disabled for the purpose of the evaluation. The slight differences in the number of generated formulas were caused by different isotope masses embedded in each software and/or by rounding errors during calculation. The runtimes are average values from three independent runs performed on three different 16-core Intel Xeon 2.9 GHz CPU workstations equipped with 189 GB RAM, running Ubuntu Linux version 12.04.5 LTS and OpenJDK Java runtime version 1.7.0_101.

Input	Mass tolerance (± Da)	# of generated formulas			Runtime (s)		
		HR2	PFG	CDK	HR2	PFG	CDK
10,000 small masses	0.001	616,846	616,846	616,843	669	168	**41**
10,000 small masses	0.01	6,163,303	6,163,302	6,163,326	689	501	**212**
20 large masses	0.001	4,912,939	4,912,939	4,912,904	26,370	1,292	**177**
20 large masses	0.01	49,128,811	49,128,810	49,128,815	26,587	3,406	**1,580**

Table 2 - CTfile Format Support.

Format	**V2000**	**V3000**
MOLfile	read and write	read and write
RXNfile	read and write	read
SDfile MOLfile	read and write	read
RGfile	read and write	
RDfile		

Table 3 - The molecular fingerprints in CDK.

Listed are the currently available molecular fingerprint in CDK with information about whether they come as a bit and/or count version, what CDK version they were introduced in, their default size, and relevant references, where applicable.

	Bit version	Count version	CDK version	Default Size
CircularFingerprinter [75, 33]	✓	✓	v2.0	1 024 / 2^{32}[*]
EStateFingerprinter [76]	✓		v1.2.0	79
ExtendedFingerprinter	✓		v1.0	1 024
Fingerprinter	✓		v1.0	1 024
GraphOnlyFingerprinter	✓		v1.0	1 024
HybridizationFingerprinter	✓		v1.4.0	1 024
KlekotaRothFingerprinter [77]	✓		v1.4.6	4 860
LingoFingerprinter [78]	✓		v2.0	NA[†]
MACCSFingerprinter	✓		v1.2.0	166
PubchemFingerprinter [79]	✓		v1.4.0	881
ShortestPathFingerprinter	✓		v2.0	1 024
SignatureFingerprinter [42]	✓	✓	v2.0	2^{32}
SubstructureFingerprinter	✓		v1.0	307

[*] For the CircularFingerprinter the bit version is folded to 1024 whereas the count version is unfolded.

[†] The LingoFingerprinter does not have a default size.

Table 4 - A selection of key CDK modules with major changes.

An overview of a selection of often used CDK modules with description, dependencies on third-party libraries, and the major changes since version 1.2. Dependencies between modules are depicted in Figure 7.

Module	**Description**	**Major Changes**	**Dependencies**
interfaces	Interfaces for the data models.		Vecmath 1.5.2
core	Core functionality.		Google Guava 17.0
standard	Common functionality.		
render	Graphical rendering.	Redesigned to make it more modular and support multiple widget toolkits, like AWT and SWT.	
isomorphism	Isomorphism and substructure searching.		
atomtype	Various non-core atom type schemes.	Unified approach where atom typing is separated from other algorithms.	
ioformats	Definitions of (chemical) input/output formats.		
io	Readers and writers for input/output formats.	The molfile reader has been rewritten and supports in atom types defind in the specification.	XPP3 1.1.4c
iordf	Stores data models as in the Resource Description Framework serialization formats.	New.	Jena 2.7.4
inchi	IUPAC International Chemical Identifier support.		JNI-InChI 0.8 [35]
libiocml	Writer for the Chemical Markup Language format.		XOM 1.2.5, CMLXOM 3.1 [80]
sdg	Structure diagram generation.	Much improved overlap resolution.	
smiles	Reading and writing in the SMILES format.	SMILES support performance and coverage is greatly improved.	Beam 0.9.1 [63]
smarts	Substructure searching with the SMARTS format.		Beam 0.9.1 [63]
hash	Molecular hash codes [81].		
formula	Chemical formula support.	New.	
fingerprint	Calculate fingerprints.	Many new fingerprint types (see text).	Apache Commons Math 3.1.1
qsar and qsarmolecular	Molecular descriptors.		XOM 1.2.5, JAMA 1.0.3 [82]
signatures	Calculation of molecular and atomic signatures.		Signatures 1.1

Table 5 - SMARTS matching performance.

Total time taken to match the SMARTS `O=[C,N]aa[N,O;!H0]` against ∼1.5 million SMILES from ChEMBL 21. The reported time is for single core execution, measure with the Unix time utility (Cent OS 7, Intel Core i7-4790 CPU @ 3.60GHz). The times for recent versions of Open Babel [83] and RDKit [84] are included for reference but this is not intended to be a systematic comparison. The difference in number found is due to variations in valence and aromaticity models.

Toolkit	Time (s)	Matched
CDK v1.4.9	3,843	69,782
CDK v2.0	67	62,518
Open Babel 2.3.90	835	61,906
RDKit 2016.3.4	378	62,402