ChemCpp User-Guide

Jean-Luc Perret , Pierre Mahé

October 2006

Introduction

ChemCpp is a C++ project developed with KDEvelop 3.2.2. It consists of a C++ library to handle molecule datasets, together with a set of binary programs for the manipulation of molecule structures files, under the MDL MOL and SDF file formats[1], and the computation of kernel matrices for datasets of molecules. In particular ChemCpp provides tools to compute different kernel for graphs corresponding to the 2D structures of molecules :

- the marginalized graph kernel [1] and its extensions [2],
- Tanimoto graph kernels [3],
- fast approximations of the marginalized [1] and geometric [4] kernels,
- tree-pattern graph kernels [5]
- pharmacophore kernels based on the 3D molecular structures [6].

We tried to make the installation and use of ChemCpp as easy as possible. Nevertheless, feel free to contact the authors for any question/suggestion/problem/bug.

1 Project architecture and installation

1.1 Basic architecture

Uncompressing the ChemCpp.**.tar.gz project creates a `chemcpp` directory in the current location, with the following sub-directories :

- the `src` directory : contains the C++ classes defining the ChemCpp library, see section 2.
- the `tools` directory : contains sub-directories corresponding to binary programs, or 'tools', based on the ChemCpp library, see section 3.

[1] See `http://www.mdli.com/downloads/public/ctfile/ctfile.jsp` for more details about these file formats, and `http://en.wikipedia.org/wiki/Chemical_file_format` for a general discussion, including conversion issues.

- the `bin` directory : contains symbolic links to the executables of the programs of the 'tools' directories.
- the `data` directory : contains several files potentially required for ChemCpp tools. A `README` file is included in the `data` directory for more details.
- the `doc` directory : gathers documentation about the ChemCpp project (in particular a user guide).
- the `html` directory : contains HTML documentation of the C++ classes of the `src` directory. Documentation was generated by `Doxygen`, according to the `Doxygen-configFile` found in the `chemcpp` directory. Please refer to the `html/index.html` HTML page for more details.
- the `examples` directory : contains pieces of codes illustrating the different classes of the project. Note that these examples are included in the `Doxygen` generated documentation.

Moreover, the `chemcpp.kdevelop` is the chemcpp project file that can be opened with KDEvelop.

1.2 Installation

In order to install ChemCpp simply:

- edit the file `src/constant.h` and define in line 19 the `CHEMCPPPATH` variable to be the `chemcpp` directory created while uncompressing the `.tar.gz` project file
- in the `chemcpp` directory, type : `./configure` to generate the project Makefiles, and build the project with `make`
- add the `chemcpp/bin` directory to your `PATH` environment variable, and the `chemcpp/src` directory to your `LD_LIBRARY_PATH` environment variable

and you're done: all the tools executables can be run in command line (type `aToolName -h` to get a description of the tool `aToolName`).

Note that during the building process :

- the C++ classes of the `chemcpp/src` directory are compiled, and the shared library `libchemcpp.so` is created in the `chemcpp/src` directory.
- the main programs of the subdirectories of the `chemcpp/tools` directory are compiled based on the `chemcpp/src/libchemcpp.so` shared library, and executables are created in every `chemcpp/tools` subdirectory.

The `chemcpp/bin` directory contains symbolic links to the different "tools" executable programs. It must therefore be included in the `PATH` environment variable so that the tools can be called in command lines. Moreover, the `chemcpp/src` directory must be included in the `LD_LIBRARY_PATH` so that the shared library `libchemcpp.so` can be found at runtime by the different tools executables.

2 C++ classes and ChemCpp library : the "src" directory

Below is a brief description of the different C++ classes of the `chemcpp/src` directory that constitute the `libchemcpp.so` shared library. Please refer to the HTML documentation to be found in the `chemcpp/html` directory for more details (just open the file `chemcpp/html/index.html` in your favorite browser). Note however that even the HTML documentation is not fully complete, but will be soon. Here is the list of the C++ classes :

- `Descriptor`: this class implements a general storage type for a scientific descriptors with label, value, unit and comment members.
- `DataContainer`: this class implements a generic collection of `Descriptor` instances. The main methods concern the addition or modification of `Descriptor` instances into the container. This class is the parent class of the classes `Elements`, `Node` (itself the parent of class `Atom`), and `Molecule` (itself parent of class `KcfMolecule`). This therefore means that any descriptor can be added at runtime for these derived classes.
- `Node`: molecules are seen as graphs in ChemCpp, and `Node` implements the notion of graph vertex. The main field of a `Node` instance consist of a (String) label, and a pair of start/end probabilities related to the random walk model involved in the marginalized kernel [1] . It is the parent class of `Atom`.
- `Atom`: implements the notion of atom. `Atom` is a daughter of `Node`, itself a daughter class of `DataContainer`. Any user descriptor can therefore be included into the atom description. However, several descriptors are included as native descriptors, e.g., Atomic Number, type, (x,y,z) coordinates, Morgan label, partial charge. Moreover, an `Atom` instance is associated a vector of `Bond` pointers (the bonds for which it constitutes the 'source' atom) arranged in a Hash indexed by the 'target' atoms. Main methods concern the manipulation of the class members, e.g., corresponding `Bond` instances, Morgan index, partial charges.
- `Bond`: implements the notion of chemical bond between atoms. Main attributes of a `Bond` instance are a pair of pointers on `Atom` instances : a 'source' and 'target' atom. Bonds considered are directed, which means that a covalent bond in a molecule gives rise to a pair of bonds having opposite 'source' and 'target' atom pointers. Another important attribute is the bond label that describes the bond type (see constants.h). Note that since `Bond` is a daughter class of `DataContainer`, other Descriptors may be added at runtime.
- `Ring`: implements the notion of ring. A ring is defined as a collection of `Bonds`.
- `Molecule`: implements the notion of molecule. It is basically defined as a vector of `Atom` instances (nb : the covalent bonds of the molecule are owned by these `Atom` instances). Main methods include Construction methods (e.g., to read a molecule from a file, introduce the bonds in the `Atom` instances, define a product-graph molecule representation [2], or a 3D-graph molecule representation [6]), Manipulation methods (e.g., to hide bonds in the molecule, introduce Morgan indices, or transform the molecule into a representation preventing 'tottering paths' in the marginalized graph kernel [2]), Kernel related methods (marginalized [1] or 3D-pharmacophore related [6]), and Output functions (e.g., to write the molecule instance in an output file).

- `KcfMolecule`: this class is derived from the `Molecule` class, and provides several functions for molecules represented in KEGG Chemical Function (KCF) format[2].

- `MoleculeSet`: represents the notion of a database of molecules. It is derived from a vector of pointers to `Molecule` instances. Among its attributes are a pair of Double matrices corresponding to the 'raw' kernel matrix (Moleculeset::gram) and to the 'normalized' kernel matrix (MoleculeSet::gramNormal). Main methods can be grouped into distinct categories : Input methods (e.g., to read molecules or molecular descriptors from files), Manipulation methods (to perform diverse operations on the molecules of the database), Kernel related methods (to build the kernel matrices associated to the molecule set for a particular kernel function), and Output methods (e.g., to write the set of molecules after their manipulation, or to write Gram matrices after kernel computation).

- `Elements`: this class instantiates the chemical elements (each of class `Atom`) and load their physico-chemical properties.

- `MoleculeUtils`: this class gathers several utilities functions, and in particular functions to read and write files, and functions related to the different kernels between molecules (e.g., basic kernels to compare nodes or edges of the graphs associated to molecules).

- `JlpIOutils`: this class provides several utilities for general Input/Output operations.

- `Cerror`: implements a class of error that can be raised by the classes of the ChemCpp project. An instance of `Cerror` is defined by a type, identifying the error occurring, and a comment, the message output when the error is raised. Several error types are defined.

- `StringUtils`: gathers a collection of string processing utilities.

3 ChemCpp executables : the "tools" directory

We now turn on to the description of the different utility programs implemented in the `chemcpp/tools` subdirectories. These tools can be split into three categories : tools for the manipulation of MOL and SDF files, tools for the computation of kernel matrices, and tools processing kernel matrices. These tools are command lines tools. For a description of their usage, type their name with the option `-h` as command line.

Below is a list of the different tools together with a brief description of their function and usage. For now, we don't go into the details of the data manipulation tools. However, an interested user can probably figure out their function from their name and description. They are provided with no guarantee, and we strongly encourage the user to check their effect on his data. The next section will be dedicated to a detailed description of the tools producing kernel matrices.

[2] In a KCF description of a molecule, additional information is included in the atoms types depending on the neighborhood properties of the atoms. See `http://web.kuicr.kyoto-u.ac.jp/simcomp/doc/keggatom.html`. More details to come.

3.1 Manipulation tools

Some options are generic to the different data manipulation tools. Among them are :

- the `-m option`, which specifies the input data, that can be a single file (in most `sd*` tools), or an input directory (in some `mol*` tools).
- the `-o option`, which specifies the output of the program, which can consist of a single file, or a directory. Default values are proposed for this option. If this option specifies a directory, the default output is the current location. If it specifies a file, the default output is a file located in the current directory, or in the directory of the input file, depending on the nature of the tool.
- the `-r option`, which removes the Hydrogen atoms from the molecules.
- the `-z option`, which specifies that the molecules are under the KEGG Molecular Function (KCF) format. See `http://web.kuicr.kyoto-u.ac.jp/simcomp/doc/keggatom.html` for more details.

In addition, each tool may require other arguments. Option `-h` outputs the correct usage of the file (required/optional parameters and default values).

3.1.1 File conversion

- **mol2sd** : merges a collection of mol files into a sd file.
- **sd2mol** : splits a given sd file into a collection of individual mol files.
- **mol2mol** : tries to converts an input mol file into a new mol file. Note that values for: bond topology, mass diff, charge diff, stereo parity, hydrogen counts, stereocare box, valence, H0 designator, reaction component type, reaction component number, atom-atom mapping number, inversion/retention flag, exact change flag are all *lost in the conversion!*
- **mutag2mol** : splits the Mutag dataset in the Prolog format into a collection of mol files[3].
- **mutag2sd** : converts the Mutag dataset in the Prolog format into a sd file. This tool offers the possibility to split the Mutag dataset into active/inactive compounds based on the activity file provided with its Prolog description.
- **sd2dots** : creates a dot file for each molecule of an input sd file. Can be used for visualization with graphviz tools (http://www.graphviz.org/)

[3]Note: the Mutag dataset is available in the `chemcpp/data` directory. It comes as the .tar.gz file with Prolog description of the molecular structures.

3.1.2 Descriptors manipulation

- **sd2descriptors** : reads a sd file and outputs a ';' separated table with molecules names and descriptors for all molecules of the sd file (missing values are indicated by 'NA').
- **sdadddescriptor** : adds a descriptor to the molecules of an input sd file. An additional argument must specify a file gathering the values of the descriptor to add for all the molecules of the input sd file (`-d option`).
- **sdaddgistclassify :** adds descriptors corresponding to the output of the GIST classification program[4] 'gist-classify' to the molecules of an input sd file. An additional argument must specify the output file of 'gist-classify'(`-d option`).
- **sdbinclassifyfromdescriptor** : initializes an 'activity' descriptor for the molecules of an input sd file by thresholding the values of a specified descriptor. Additional arguments must specify the name of the descriptor upon which the activity binarization is based, and the corresponding threshold value (`-d` and `-a options`).
- **sdqualify** : adds a constant descriptor to all molecules in a sd file. Additional arguments must specify the name, type and value of the constant descriptor type to be added (`-n`, `-i`, `-f` and `-s options`).

3.1.3 Compound alteration

- **molremovesalts**: removes all but the largest connected graph in an input mol file and outputs two mol files : <output name>.salts.mol and <output name>.noSalts.mol
- **sdremovesalts**: removes all but the largest connected graph in all entries of an input sd file. outputs two sd files : <output name>.salts.sd and <output name>.noSalts.sd

3.1.4 Molecule set manipulation

- **sd2gistactivity** : creates the activity file of an input sd file in the GIST file format.
- **sd2subset** : extracts a subset of a input sd file. The molecules of the subset are specified by either a list of molecule names (`-l option`), a single molecule name (`-n option`), or a number of molecules (`-u option`, the subset is then made of the first molecules of the sd file). Outputs a sd file. The `sdremovesubset` does the converse operation.
- **sdexclude** : removes from an input sd file the molecules identical to a set of specified molecules. Note that the criterion of 'molecular identity' is not based on the molecule names but involves several filters (number of atoms, molecular weight, value of the graph kernel). An additional argument must specify a sd file of molecules that are to be removed from the input sd file (`-l option`).

[4]Note that the design of the different tools was motivated by the use of the GIST software to perform SVM classification. More details in section 4 related to kernels computation.

- **sdfiltermw** : removes from the molecules of an input sd file those having a molecular weight outside of a given range. Additional arguments must specify the range tolerated for the molecular weight (`-i` and `-a options`). If the range is not specified, a 'molecular weight' descriptor is added to the molecules of the sd file.

- **sdfilternumatoms** : removes molecules of an input sd file having a number of atoms outside a given range. Additional arguments must specify the range tolerated for the number of atoms (`-i` and `-a options`). If the range is not specified, a 'number of atoms' descriptor is added to the molecules of the sd file.

- **sdremovesubset** : removes a subset of molecules from an input sd file. The molecules of the subset are specified by either a list of molecule names (`-l option`), or a single molecule name (`-n option`). Outputs a sd file. The `sd2subset` does the converse operation.

- **sdremoveduplicates** : removes duplicate molecules of an input sd file and outputs a sd file.

- **sdsort** : sorts the entries of an input sd file according to a given descriptor. An additional argument must specify the sorting descriptor (`-d option`).

- **sddescribe** : displays a description of an input sd file.

- **sdsplitfromdescriptor** : splits an input sd file into a collection of sd files, according to the values of a given integer descriptor. An additional argument must specify the splitting descriptor (`-d option`).

- **sdsplitfromgistclassify** : splits an input sd file according to the output of the GIST classification program 'gist-classify'. produces a couple of sd files gathering positive and negative compounds. An additional argument must specify the output file of 'gist-classify' (`-d option`).

- **sd2atomGram**: computes kernel values between the different types of atoms found in a sd file. Usage:

```
OPTIONS
  -m <input sd file> train set
  [-s <input sd file> test set]
  [-r remove H atoms]
  -t <kernel type>: 1 = RBF ; 2 = triangle
  [-p <value>] kernel parameter : bandwidth or cut-off percent
       (default: 1, cannot be <= 0)
  -c <property/ies> property/ies string: prop1-prop2-...-propN
       NB: possible choices = Am;Vwr;Ar;Cr;Mp;Bp;Ea1;Ie1;
                              Enpauling;Enallred;Enabs
         --> see chemcpp/data/elements.csv for their meaning
  [-o <output file>]
    (if ommited : output file = atomGram + kernelType + parameter value)
```

This tool computes a kernel matrix between atom types found in a dataset of molecules, specified by the `-m` (and possibly `-s`) option(s)[5], based on atomic physicochemical properties. The output of this tool is a file made of 109 lines (1 line per possible atom types ; each line gathering 109 kernel values separated by ';') that can be used in the `sd2gram` and `sd2gram3Dpharma` tools using `option -k`, in order to replace the default "binary matching" between atoms based on their types by a more flexible matching taking into account the similarity of atoms based on physicochemical property/ies.

This tool starts by building a matrix X of size $(n \times p)$, each line gathering the values of the p properties specified in the command line using `option -c`, for the n different types of atom found in the dataset. The values of the properties are read from the file `chemcpp/data/elements.csv`, possible properties being: the atomic mass (Am), the van der Waals radius (Vwr), the atomic radius (Ar), the covalent radius (Cr), the melting point (Mp), the boiling point (Bp), the first electron affinity (Ea1), the first ionization energy (Ie1), the electronegativity Pauling (Enpauling), the electronegativity Allred (Enallred) and the electronegativity Absolute (Enabs). Several properties can be specified by giving `option -c` a string made of the different properties labels, separated by the '-' delimiter, e.g., `Vwr-Ar-Enpauling` to consider simultaneously the Van der Waals radius, atomic radius and electronegativity Pauling.

The matrix X is then standardized such that each column has a zero mean and a unity variance:

$$X[:, j] \leftarrow \frac{X[:, j] - \text{mean}(X[:, j])}{\text{std}(X[:, j])},$$

and a matrix D of size $n \times n$ gathering (Euclidean) distances between pairs of atom types is computed:

$$D[i, j] = || X[i, :] - X[j, :] || = \sqrt{\sum_{k=1}^{p} (X[i, k] - X[j, k])^2}.$$

Finally, kernel values are computed from this distance matrix. Possible kernel functions are:

- a RBF kernel (`option -t 1`): $K(i, j) = \exp\left(\frac{-D[i,j]^2}{2\sigma^2}\right)$, where the parameter σ is specified by `option -p`.
- a triangular kernel (`option -t 2`): $K(i, j) = \frac{C - D[i,j]}{C}$ if $D[i, j] \leq C$, 0 otherwise. The cut-off value C is computed from the parameter given by `option -p`, which specifies the amount of sparsity in the resulting atom-types kernel matrix. The value of the parameter given by `option -p` is comprised between 0 and 1, and specifies the amount of kernel values to be retained. For instance, using `option -p 0.25` specifies to set 75% of the kernel values between different atom types to zero, and the value of C is computed accordingly. Therefore, the smaller the value specified by `option -p`, the smaller is C, and therefore, the sparser is the kernel matrix. At the extreme where `option -p 0`, we get a binary diagonal matrix. Conversely, with `option -p 1`, we get a full matrix. Note that the sparsity of the matrix specified by `option -k` in the `sd2gram` and `sd2gram3Dpharma` has an effect on the computational cost of the kernels (the sparser the matrix, the smaller is the cost of computing the kernel, see [6] for a discussion).

[5]Note that `option -m` is required, and `option -s` can be used in a "test set" mode, as explained in Section 4.

The entries of the output file (specified by `option -o`) corresponding to pairs of atoms found in the dataset are filled with the computed kernel values, and the remaining entries are set to 0.

3.2 Kernel matrices computation tools

The tools presented in this section are designed to compute kernel matrices for input datasets. Below is a brief description of the different tools, section 4 is dedicated to their precise characterization.

- **sd2gram, mol2gram, molsd2gram, mutag2gram, moblast**: compute the marginalized kernel [1], and proposed extensions [2] for different input files.
- **sd2gramSpectrum**: computes several walk-based graph kernel functions based on finite length walks and a fast implementation for input sd file(s).
- **sd2gramSubtree**: computes several graph kernels based on the count of common subtrees [5] for input sd file(s).
- **sd2gram3Dpharma**: computes the (exact) *pharmacophore kernel* [6] for input sd file(s).
- **sd2gram3Dspectrum**: computes the fast approximations of the *pharmacophore kernel* [6] for input sd file(s).

3.3 Tools processing a similarity matrix

- **gramraw2self** : reads a raw gram matrix and writes diagonal elements to a file with the names of the molecules.
- **gram2diversity** : reads a gram matrix and returns the mean distance of the points from their barycenter in the feature space, which constitutes a particular *diversity* criterion of the dataset according to the kernel function used to compute the kernel matrix.
  - A kernel function k corresponds to a dot-product between the data mapped to a vector space $\mathcal{F}$ (the so-called *feature-space*): $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$. A distance in the feature-space is therefore implicitly defined by the kernel function : $d_{\mathcal{F}}(x_1, x_2) = ||\phi(x_1) - \phi(x_2)||^2 = k(x_1, x_1) + k(x_2, x_2) - 2k(x_1, x_2)$. The diversity of the dataset can be measured by the average distance between the data points and their barycenter in the feature space. For a set $S = \{x_1, ..., x_n\}$, this writes as $D(S) = \frac{1}{n} \sum_i d_{\mathcal{F}}(x_i, M)$, where M is the center of mass of S, i.e. $M = \frac{1}{n} \sum_i \phi(x_i)$. The diversity measure computed by `gram2diversity` is therefore given by $D(S) = \frac{1}{n} \sum_i k(x_i, x_i) - \frac{1}{n^2} \sum_{i,j} k(x_i, x_j)$.

4 Computation of kernel matrices

The tools presented in this section are designed to compute kernel matrices for input datasets. They are said to work in 'training mode' if they take a single dataset as input and output a square kernel matrix, gathering all pairwise kernel values for all molecules of the dataset. Conversely, they are said to work in 'testing mode' if they take a pair of datasets as input and output a kernel matrix gathering all pairwise

kernel values between the molecules of the first dataset and the molecules of the second one.
More precisely :

- all `sd2gram*` tools take sd files as input: 1 sd file in 'training mode', or 2 sd files in 'testing mode'.
- the `mol2gram` tool takes one or two sets of mol files (located in specified directories) as input to compute a kernel matrix in 'training' or 'testing' modes.
- the `molsd2gram` outputs a kernel matrix gathering all pairwise kernel values between the molecules of a sd file and that of a set of mol files, and is therefore working in 'testing mode' only.
- the `mutag2gram` outputs a square kernel matrix gathering all pairwise kernel values between the molecules of the Mutag dataset in the Prolog format, and is therefore working in 'training mode' only.

These tools input options are arranged in four categories : *Input*, *Preprocessing*, *Kernel* and *Output* options. The *Kernel* options depend on the tool, but the other options are common to all the tools. More precisely :

- the **Input options** specify the input data :

```
INPUT OPTIONS
  -m <file name>
  [-s <file name to compare with>]
    (if ommited: gram matrix of the molecules in the -m directory)
  [-z <format>] file format of input files. (default: mol/sd)
                allowed values: mol/sd, genericmol/genericsd, kcf
                note: -k cannot be used when the input file is
                        genericmol/genericsd or kcf
```

  The `-m option` specifies the first (required) source of input data. It consists of a sd file for most of the tools (all `sd2gram*` tools), but can be a directory of mol files (`mol2gram` and `molsd2gram` tools), or the Prolog file representing the Mutag dataset (`mutag2gram`). The `-s option` specifies the second source of input data in 'testing mode' configuration. It is optional for most of the tools (all `sd2gram*` tools and the `mol2gram` tool), but required for the `mutagsd2gram` tool, and prohibited for the `mutag2gram` tool. The `-z option` specifies the format of the input data (can be sd/mol, KCF[6], or 'generic')

- the **Preprocessing options** enable to perform various operations on the molecules before the kernel computation :

```
PREPROCESSING
  [-r remove H atoms]
  [-i <anOrder>] introduce Morgan index labels of order i
```

[6] See `http://web.kuicr.kyoto-u.ac.jp/simcomp/doc/keggatom.html` for more details.

The `-r option` specifies to remove the Hydrogen atoms from the molecules. The `-i option` has the effect of introducing Morgan indices computed at the given `anOrder` into the labels of the atoms (see [2] for more details).

- finally, the **Output options** are related to the output of the methods :

```
OUTPUT OPTIONS:
  [-y silent mode]
  [-o <output directory>]
    (if ommited: current location is used)
  [-b <base name>]
    (if ommited : base name = name of the tool ex: 'sd2gram')
```

The `-y option` is a flag indicating to turn to 'silent mode' (i.e., to reduce the verbose). The `-o option` specifies the output directory to write the various output files. By default, the output files are written in the current directory. The `-b option` specifies the 'base name' of the various output files, a common prefix to all output files. A default base name value is proposed for each tool (which is 'sd2gram' in the above example corresponding to the `sd2gram` tool). Use option `-h` for more details.

Chemcpp tools output the following files, where `***` stands for the 'base name' of the output files, specified by the `-b option`:

- a `***.gramRaw` file containing the computed kernel matrix
- a `***.gramNormal` file containing a normalized version of the kernel matrix, using the following normalization scheme : $\tilde{K}[i][j] = \frac{K[i][j]}{\sqrt{K[i][i]K[j][j]}}$
- a `***.train_self` file containing the self kernel values of the dataset of molecules specified by the `-m` option, and possibly a `***.test_self` file containing the self kernel values of the dataset of molecules specified by the `-s` option in 'testing mode' configuration.

Note that these tools were designed to build kernel matrices to be used as input to the GIST classification program[7]. Please refer to this software website for more details about the file format. Note finally that this file format is compatible with the more recent python PyML[8] machine learning package. We now turn on a detailed characterization of the different tools for computing kernel matrices. We detail the kernel functions computed and the corresponding *Kernel* options field.

4.1 sd2gram, mol2gram, molsd2gram, mutag2gram

These four tools compute the marginalized kernel [1] and its proposed extensions [2] for various input files:

[7] Available at `http://microarray.cpmc.columbia.edu/gist/`.
[8] Available at `http://pyml.sourceforge.net/`.

- `sd2gram` : takes one or two sd files in input ('training/testing' modes respectively).
- `mol2gram` : takes one or two sets of mol files in input ('training/testing' modes respectively).
- `molsd2gram` : takes a sd file and a set of mol files in input ('testing' mode).
- `mutag2gram` : takes the Prolog Mutag dataset in input ('training' mode).

They are based on the same following kernel options :

```
KERNEL OPTIONS
  [-q <stop prob>] (0 > q < 1, default = 0.1)
  [-c <convergence condition>] (default = 1000)
  [-k <atom Kernel matrix>] (default: identity matrix)
    WARNING this option cannot be specified when -i is specified
  [-g filter the 'tottering paths']
```

We first briefly recall the marginalized kernel formulation. The molecules read from the input dataset(s) are here seen as labeled graphs corresponding to their 2D structures (atoms + covalent bonds). More formally, a labeled graph G is defined by :

- a set of vertices $\mathcal{V}_G = (v_1, ..., v_{|G|})$, where $|G|$ is the number of vertices of G (in our case, the number of atoms of the molecule)
- a set of edges $\mathcal{E}_G \in \mathcal{V}_G \times \mathcal{V}_G$, that connect pairs of vertices (in our case it corresponds to the set of covalent bonds of the molecule)
- a labeling function $\mathcal{L}$ that assigns a label to any vertex or edge of the graph. In our case, the vertices of the graphs are labeled by the type of their corresponding atoms (that is, `C, H, O`...), and the edges labels correspond to the degree of the covalent bonds (single, double, aromatic...)

Moreover we introduce the following definitions:

- $h = (v_0, .., v_n)$ is a walk of length n in the graph G if it is defined as a set of $n + 1$ connected vertices : $v_i \in \mathcal{V}_G$, $0 \leq i \leq n$, $(v_i, v_{i+1}) \in \mathcal{E}(G)$, $0 \leq i \leq n - 1$.
- $\mathcal{H}(G)$ is the whole set of walks of the graph G
- we extend the vertex/edge labeling function $\mathcal{L}$ to the walks of the graphs : the label of a walk is defined as the concatenation of the labels of the vertices and edges it is made of.

For the pair of graphs G and G', the marginalized kernel defines as:

$$K(G, G') = \sum_{h \in \mathcal{H}(G)} \sum_{h' \in \mathcal{H}(G')} p_G(h) p_{G'}(h') K_{Walk}(\mathcal{L}(h), \mathcal{L}(h')) \quad (1)$$

where p_G (resp. $p_{G'}$) is a probability distribution over $\mathcal{H}(G)$ (resp. $\mathcal{H}(G')$), and K_{Walk} is a kernel between walk labels.

Specifying the kernel therefore resumes to choosing the probability distributions and the kernel between walks K_{Walk}. By default, the `sd2gram` tool implementation follows the parametrization proposed in [1]:

- the walk probabilities are defined by a first order random walk model which is completely defined by a single parameter: the random walk ending parameter (called p_q in [1]). This parameter is specified by `option -q` (default value is 0.1). Please refer to [1] for the definition of the random walk model.
- the kernel between walks K_{Walk} is a Dirac kernel : pairs of walks are given a perfect similarity if they have the same label, and null otherwise.

and the kernel is computed by the iterative method proposed in [1]. The corresponding convergence condition is given in `option -c`: the algorithm stops when the kernel value does not change by more than $1/c$, where c is the value specified by the `-c` option.

Computing the kernel proposed as it proposed in [1] (i.e., using a Dirac walk kernel and the particular random walk mode described in the paper) therefore simply requires the options `-q` and `-c`. However, the `sd2gram` tools offers several other options :

- `option -k` offers the possibility to specify a matrix of kernel values between different atom types. This has the effect of replacing the Dirac kernel between atom types implicitly involved in the Dirac walk kernel K_{Walk} (see eqs 7.4 and 7.6 in [1]) by a more flexible kernel. As a result, the similarity between walks is not binary anymore. The directory `chemcpp/data` gathers possible input matrices, and the `R` script `chemcpp/data/makeGramAtoms.R` makes it possible to compute other based on atomic physicochemical properties (please refer to the `chemcpp/data/README` file for more details about the required file format). Alternatively, the `sd2atomGram` tool makes it possible to compute such kernel matrices for the types of atoms actually found in a sd file, based on simple kernel functions (RBF and triangular) and general physicochemical properties of atoms.
- `option -g` filters the "tottering paths" from the kernel. Please refer to [2] for more details about tottering paths and the way to filter them from the kernel. This option removes a set of features thought to be noisy from the kernel, and was proposed as a way to increase the expressive power of the kernel. Note however that this comes at the expense of an increase in the cost of computing the kernel. Moreover, recall that `option -i` implements the second extension based on Morgan indices that was proposed in [2]. Note however that this tool does not implement the kernel computation based on product-graphs and does not fully benefit from the gains in computation times that could be obtained using Morgan indices.

4.2 sd2gramSpectrum

This tool implements several walk-based kernel for graphs corresponding to the 2D structure (atoms and covalent bonds) of the molecule, based on the following options:

```
KERNEL OPTIONS
  -l <n> kernel involving paths of length only n
  -u <n> kernel involving paths of length until n
  [-t <kernel type>]
    (if ommited, t = 0)
     -t = 0 : Spectrum kernel
     -t = 1 : Tanimoto Kernel
     -t = 2 : Min/Max Tanimoto Kernel
     -t = 3 : Marginalized kernel approximation
     -t = 4 : Lambda^k kernel
  [-p <kernel parameter>]
     -if t = 3 --> p = ending probability pq (0 < p < 1)
       (if ommited : p = 0.1)
     -if t = 4 --> Lambda parameter
       (if ommited : p = 1 <--> choosing t = 0)
```

These kernels are based on sets of *molecular fragments* : sequences of atoms connected by covalent bonds. A labeled walk of a graph therefore corresponds to a particular molecular fragment. In opposition to the marginalized kernel formulation (based on an infinite sum of walks), the set of molecular fragments considered in the kernels is finite, and for the pair of graphs G and G', these kernels formulate as :

$$K(G, G') = \sum_{p \in \mathcal{P}} \phi_G(p) \phi_{G'}(p) \tag{2}$$

where $\mathcal{P}$ is the set of molecular fragments taken into account in the kernels, and $\phi_G(p)$ is the weight given to the molecular fragment p in the graph G. Note that this formulation corresponds to a dot product in a feature space indexed by $\mathcal{P}$, and leads to positive definite kernels.

The set of molecular fragments $\mathcal{P}$ defines the feature space corresponding to the kernels and is determined by the options `-l` and `-u` :

- `option -l <n>`, "only n", defines $\mathcal{P}$ as the set of possible molecular fragments of length n (i.e., fragments defined by $n + 1$ types of atoms + n types of covalent bonds).
- `option -u <n>`, "until n", defines $\mathcal{P}$ as the set of possible molecular fragments of length ranging from 0 to n.

(Note that the different types of atoms and bonds that define the set of possible molecular fragments are determined by the types of atoms and bonds that can be found in the input dataset(s))

The kernel function itself is defined by the molecular fragment weighting function ϕ, which is determined by the option `-t`, that takes values in $\{0, 1, 2, 3, 4\}$ (note that we keep the notations introduced

in the previous subsection to represent graphs : $\mathcal{H}(G)$ represents the walks of the graph G, and $\mathcal{L}$ is the graph labeling function extended to $\mathcal{H}(G)$):

- option `-t = 0` corresponds to the so-called *spectrum kernel*. The function ϕ counts the number of times the molecular fragment are found in the graphs :

$$\phi_G(p) = \#\{p \in G\} = \sum_{h \in \mathcal{H}(G)} \mathbf{1}(\mathcal{L}(h) = p)$$

- option `-t = 1` corresponds to the *Tanimoto kernel* introduced in [3]. ϕ is a binary functional indicating whether or not the molecular fragments are found in the graphs :

$$\phi_G(p) = \mathbf{1}(p \in G)$$

  In this case, the output `.gramRaw` file is the kernel matrix corresponding to the kernel of equation (2): the dot product between binary vectors ϕ_G and $\phi_{G'}$ defined above. The "true" Tanimoto kernel matrix is given by the output `.gramNormal` file, which can be seen as a particular normalized version of the `.gramRaw` kernel matrix :

$$\tilde{K}[i][j] = \frac{K[i][j]}{K[i][i] + K[j][j] - K[i][j]}$$

- option `-t = 2` corresponds to the *"Min/Max" Tanimoto kernel* introduced in [3]. This kernel is a variation of the Tanimoto kernel presented above. Please refer to the reference paper for more details. Note that the "true" Min/Max Tanimoto kernel is given by the output `.gramNormal` file.

- option `-t = 3` corresponds to an approximation of the marginalized kernel [1] when it is parametrized by a Dirac kernel to compare labeled walks. Walks of the graphs are considered as realization of the random walk process proposed in [1], where the stop-probability parameter p_q is specified by the `option -p`. The functional ϕ weights a molecular fragment by the sum of the probabilities of the walks of the graph that have the corresponding label :

$$\phi_G(p) = \sum_{h \in \mathcal{H}(G)} p_G(h) \mathbf{1}(\mathcal{L}(h) = p)$$

  Note that in the "until n" case (`option -u`), this kernel is equivalent to the marginalized kernel, where the summation over the walks is limited to the walks having a length up to n. At the limit, this kernel is therefore identical to the marginalized kernel (nb : when it is parametrized by a Dirac walk kernel). Note however that because of the random walks process, walks probabilities exponentially decrease with their length. Long walks are therefore barely taken into account in the marginalized kernel formulation and it makes sense to limit the length of the walks.

- option `-t 4` corresponds to the so-called *lambda-k* kernel:

$$\phi_G(p) = \sum_{h \in \mathcal{H}(G)} \lambda^{|p|} \mathbf{1}(\mathcal{L}(h) = p),$$

where the parameter λ is specified by the option `-p`, and $|p|$ is the number of atoms of the molecular fragment p. This kernel can be seen as a generalization of the spectrum kernel, where paths are not only counted but also weighted. Indeed, in the case where λ equals one, this kernel corresponds exactly to the spectrum kernel, but a value of λ greater (resp. smaller) than one stresses the influence of long (resp. short) walks. Alternatively, it can be seen as a variation of the marginalized kernel approximation discussed previously, where the probabilistic model is replaced by a simple walks weighting scheme. Finally, in the "until n" case (`option -u`), this kernel constitutes an approximation to the geometric kernel introduced in [4], where the infinite summation over walks length is replaced by a finite one, just as in the previous approximation of the marginalized kernel.

Note importantly that the set of walks of the graphs considered in these kernels (i.e., $\mathcal{H}(G)$ for the graph G in the above equations) is restricted to the *non-tottering* ones. See [2] for a precise definition of non-tottering walks. The alternative `sd2gramSpectrum-TOTTERING.cpp` program located in the `chemcpp/tools/sdgramSpectrum` enables to consider tottering walks as well: just rename `sd2gramSpectrum-TOTTERING.cpp` into `sd2gramSpectrum.cpp` and run `make` in order to switch to this "tottering" formulation.

These kernels benefit from a fast implementation that circumvent the limitations of walk kernels based on infinite sum of walks such as the marginalized [1] or geometric/exponential[4]. This implementation is derived from that of spectrum kernels for string[7].

4.3 sd2gramSubtree

This tools computes several graph kernels based on the detection of common subtrees: the so-called *tree-pattern graph kernels*, originally introduced in [8], and revisited in [5]. The `sd2gramSubtree` tool computes the different kernels proposed in [5], based on the following parameters:

```
KERNEL OPTIONS
  [-t <kernel-type>]
    -t = 0 : size-based subtree penalization
    -t = 1 : branching-based subtree penalization
    (if ommited, t=0)
  [-l <depthMax>] tree-patterns of depth = depthMax
    (if ommited, depthMax = 3)
  [-p <lambda>] weighted contribution of tree-patterns depending on their sizes
    (if ommited: lambda = 1)
    (NB: increasing lambda favors patterns of increasing complexity)
  [-g filter the 'tottering tree-patterns']
  [-u consider 'until N tree-patterns']
    WARNING can only be used in conjunction with option -t = 1
```

This class of kernel constitutes an extension of the kernels based on the count of common walks implemented by the `sd2gramSpectrum` tool, where common subtrees are detected instead of common

walks. For more details about the kernel definitions, please refer to [5], the usage of this tool should be clear after wards. A pseudo-code of this tool is given in Pierre Mahé's PhD thesis[9].

Note that by default this tool computes a kernel based on tree-patterns of a given length, specified by `option -l <depthMax>`. However, since the implementation is based on dynamic programming, all kernels computed based on patterns of depth = 1 to depth = depthMax (where depthMax is specified by `option -l`) can be obtained at no extra cost. The alternative `sd2gramSubtree-ALL-ORDERS.cpp` program located in the `chemcpp/tools/sdgramSubtree` enables to do so (just rename `sd2gramSubtree-ALL-ORDERS.cpp` in `sd2gramSubtree.cpp` and run `make`).

4.4 sd2gram3Dpharma

This tool implements the (exact version of) *pharmacophore kernel* for 3D structures of molecules proposed in [6]. Molecules are here represented by the 3D coordinates of their atoms, which are read from the input dataset(s). The kernel is based on the following set of options :

```
KERNEL OPTIONS
  [-k <atom Kernel matrix>] (default: identity matrix)
      WARNING this option cannot be specified when -i is specified
  [-e <edge kernel type>]
    (if ommited, e = 1)
    - e=1 : RBF kernel
            (option 'p' = bandwidth sigma)
    - e=2 : triangular kernel
            (option 'p' = threshold C)
  [-p <edge kernel parameter>]
    (if ommited = value of 1.0 is used)
  [-g <file name>  chargesFileName no1]
       WARNING this option cannot be used in conjunction with option -k
  [-j <file name> chargesFileName no2]
      - required if (-s + -g) options is specified
  [-d <threshold on partial charges>]
      - if ommited : threshold = 0.0
```

This tool was designed to compute the kernel matrices involved in the experimental section of the paper [6]. Options can be arranged in three categories:

- options to specify the kernel function comparing distances between atoms:
  - `option -e` specifies the kernel function:
    - `option -e 1`: RBF kernel: $K(d, d') = \exp \frac{-(d-d')^2}{2\sigma^2}$
    - `option -e 2`: Triangular kernel: $K(d, d') = \frac{C-|d-d'|}{C}$ if $|d - d'| \leq C$; 0 otherwise

[9] Available at `http://cbio.ensmp.fr`, or upon request to `jean-philippe.vert@ensmp.fr`

- `option -p` specifies the parameter associated to these kernels : the bandwidth (σ) of the RBF kernel, or the cut off (C) of the triangular kernel. Default values are given, refer to the usage with `option -h`.

- options to specify the kernel function to compare the atoms. By default, the kernel is based on a binary matching between atom types, possibly enriched by Morgan indices (with `option -i`), or the sign of their partial charges (with `options -g/j/d`, see below). However, it is possible to rely on physicochemical properties instead of atom types to compare atoms, as it is suggested in [6]. This can be done by specifying a kernel matrix between atom types using `option -k`. Such a matrix comes as a set of 109 lines (1 atom per line), where each line gathers 109 kernel values (comparing pairs of atom types) separated by ";". The `sd2atomGram` enables to automatically compute simple atom-types kernel matrix (using a RBF or triangular kernel) based on standard physicochemical properties. However, it is worth mentioning that preliminary experiments (see [6]) tend to indicate that characterizing atoms by general physicochemical properties does not bear additional information to simply considering their types. This actually makes sense because these properties are directly defined from the atom types and thus do not depend on the molecule itself, in opposition to partial charges for instance.

- options to specify the partial charges of the atoms:

  - `option -g` specifies the file gathering the atomic partial charges of the molecules of the dataset given by `option -m` (see the "Input options" section). This option can't be used in conjunction with `option -k`. Each line of this file corresponds to a molecule of the input dataset, and gathers the values of the partial charges of the atoms separated by a ';'.
  - `option -j` specifies the file gathering the atomic partial charges of the molecules of the dataset given by `option -s` (see the "Input options" section). It is required when `option -g` and `option -s` are simultaneously specified.
  - `option -d` specifies a threshold above which partial charges are considered as positive/negative. By default this threshold is zero, and every positive (resp. negative) partial charge is seen as a positive (resp. negative) charge. However, it might be interesting to consider a threshold of 0.2 for example, in which case only partial charges greater than 0.2 (resp. smaller than -0.2) would be seen as positive (resp. negative).

In the experiments presented in [6], a RBF kernel is used to compare inter-atomic distances, which is the more natural choice. As mentioned in the paper, the main interest of choosing a triangular kernel lies in the fact that it might improve the algorithm from the computational point of view, if sparse matrix algorithms were available to compute the kernel, which is not the case for the moment. Moreover, experiments suggest that partial charges increase the description of the atoms from the pharmacophore point of view. When partial charges are available, they should therefore be specified using options `-g` (and `-j` in a 'testing mode' configuration). Note finally that option `-d` may be relevant to introduce constraint on a minimum magnitude of the partial charges, but this did not lead to significant improvements in our experiments.

From these preliminary experiments, a safe default choice for this tool would be to use `option -e 1` + `option -p 0.1`, with the additional specification of partial charges, if available, with options `-g` and `-j`.

4.5 sd2gram3Dspectrum

This tool implements the six discrete approximations of the pharmacophore kernel presented in [6]. It is based on the following options :

```
KERNEL OPTIONS
  [-t <kernel type>]
    (if ommited, t = 0)
     -t = 0 : 3-points spectrum kernel
     -t = 1 : 3-points binary kernel
     -t = 2 : 3-points Tanimoto kernel
     -t = 3 : 2-points spectrum kernel
     -t = 4 : 2-points binary kernel
     -t = 5 : 2-points Tanimoto kernel
  [-p <bins>] number of bins used to discretize the inter-atomic lengths
     (if ommited p = 20)
  [-f <distMin>] minimum distance for inter-atomic distance range
      (if ommited distMin = 0)
  [-u <distMax>] maximum distance for inter-atomic distance range
      (if ommited distMax = 20 angstroms)
  [-c <fileName>] atomic partial charges (--> label = AtomType + {+/0/-})
  [-d <fileName>] 2nd atomic partial charges files (if -s option specified)
  [-v <threshold>] threshold value to consider +/- atoms charges
   (if ommited : v = 0.0)
```

The usage is quite simple (see the reference paper, section 6 for more details) :

- `option -t` specifies one of the 6 proposed approximations
- `option -p` specifies the number of bins used in the discretization of the space of pharmacophores
- `option -f` and `option -u` specify the range of inter-atomic distances considered in the discretization. By default, the 0-20 angstroms range is used. Note that only pairs of atoms entering this distance range are taken into account in the pharmacophore characterization of molecules
- `option -c` specifies the file gathering the atomic partial charges of the molecules of the dataset given by `option -m` (see the "Input options" section). Each line of this file corresponds to a molecule of the input dataset, and gathers the values of the partial charges of the atoms separated by a ';'
- `option -d` specifies the file gathering the atomic partial charges of the molecules of the dataset given by `option -s` if it is specified (see the "Input options" section). This option is required if options `-s` and `-c` are specified simultaneously

- `option -v` specifies a threshold above which partial charges are considered as positive/negative. By default this threshold is zero, and every positive (resp. negative) partial charge is seen as a positive (resp. negative) charge. However, it might be interesting to consider a threshold of 0.2 for example, in which case only partial charges greater than 0.2 (resp. smaller than -0.2) would be seen as positive (resp. negative)

Note that in practice, we observed that an adequate value for the number of bins was between 20 and 30.

A Modification of the library and creating new tools

To introduce new functionalities in the ChemCpp library, you simply need to modify the source classes of the `chemcpp/src` directory and rebuild the library. Note that while the `make` command in the `chemcpp` builds the whole project (i.e., library + tools), the `make` in the `chemcpp/src` directory simply builds the library.

The process of creating a new 'tool' can be summarized as follows :

- open the file `chemcpp/chemcpp.kdevelop` with KDEvelop[10]
- in the `view` menu of KDEvelop, select `"Tool Views/show Automake manager"`. A tab named `Automake manager` will be opened in the right hand side of the KDEvelop window, see figure 1, top left.
- right-click on the `tools` directory in the upper part of the `Automake manager` tab, choose the `Add subproject` option, and give name to the new tool, say `myNewTool`. This will create a subdirectory `myNewTool` in the `tools` directory.
- right-click on the newly created `myNewTool` directory, and choose the `Add target` option. Choose `Program` and `Bin` to indicate that the target is an executable, and repeat `myNewTool` for the name of the target, see figure 1, top right.
- right-click on the `myNewTool` directory a second time, choose the `Options` option. In the `Compiler` tab, indicate `-lpthread -lchemcpp -L../../src/` in the `CXX flags` field (this indicates that the `chemcpp/src/libchemcpp.so` library must be included in the compilation), and in the `include` tab, choose to include the `src` directory, see figure 1, bottom left.
- right-click on the previously created `myNewTool` target in the bottom part of the `Automake manager` tab, and choose `Create new file` option. Choose `C++ source`, and name the file `myNewTool.cpp`, see figure 1, bottom right.
- choose `Run Automake and friends` in the `Build` menu of KDEvelop to create the appropriated makefiles.

[10] The following process can be carried out manually, but KDEvelop makes the task much easier.

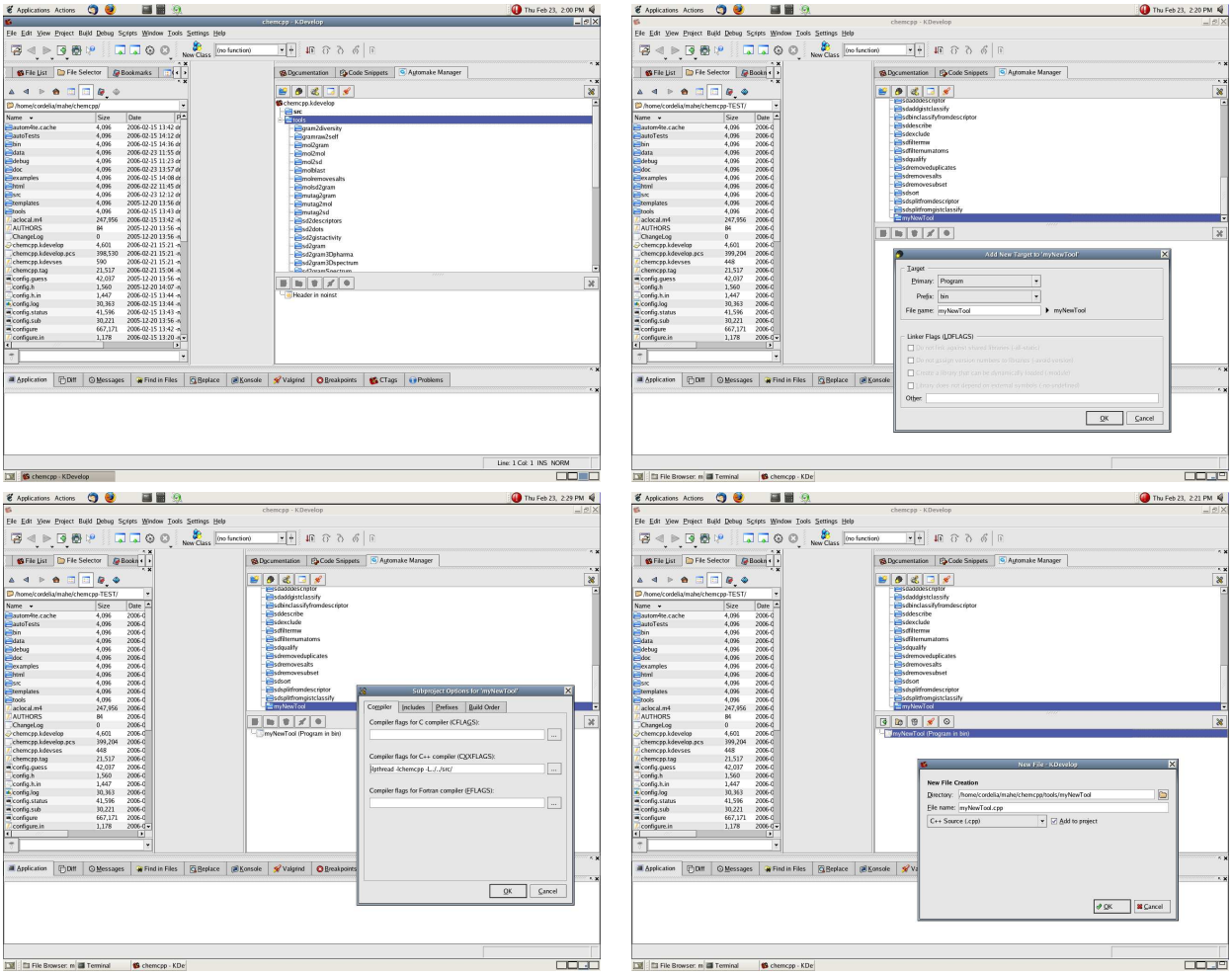

Table 1: KDEvelop screenshots of the process of creating a new tool described in section A

- eventually, create a symbolic link in the `chemcpp/bin` directory. In the `chemcpp/bin` directory, type : `ln -s myNewTool ../tools/myNewTool/myNewTool`

The last step is to fill in the `chemcpp/tools/myNewTool/myNewTool.cpp` source file.

References

[1] H. Kashima, K. Tsuda, and A. Inokuchi. Kernels for graphs. In B. Schölkopf, K. Tsuda, and J.P. Vert, editors, *Kernel Methods in Computational Biology*, pages 155–170. MIT Press, 2004.

[2] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *J Chem Inf Model*, 45(4):939–51, 2005.

[3] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Netw.*, 18(8):1093–1110, Sep 2005.

[4] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: hardness results and efficient alternatives. In B. Schölkopf and M. Warmuth, editors, *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory and the Seventh Annual Workshop on Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 129–143, Heidelberg, 2003.

[5] P. Mahé and J.-P. Vert. Graph kernels based on tree patterns for molecules. Technical Report Technical report HAL:ccsd-00095488, Ecoles des Mines de Paris, September 2006.

[6] P. Mahé, L. Ralaivola, V. Stoven, and J.-P. Vert. The pharmacophore kernel for virtual screening with support vector machines. Technical Report Technical Report HAL:ccsd-00020066, Ecole des Mines de Paris, march 2006.

[7] C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: a string kernel for SVM protein classification. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Kevin Lauerdale, and Teri E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing 2002*, pages 564–575. World Scientific, 2002.

[8] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In T. Washio and L. De Raedt, editors, *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.