Guide & Documentation

M. Robinson

2012-2014

`NanoCap` provides both libraries and a standalone application for the construction of capped nanotubes of arbitrarily chirality and fullerenes of any radius. Structures are generated by constructing a set of optimal dual graph topologies which are subsequently optimised using a carbon interatomic potential. Combining this approach with a GUI featuring 3D rendering capabilities allows for the rapid inspection of physically sensible structures which can be used as input for molecular simulation.

The `NanoCap` source and builds for different platforms can be found at:

`http://sourceforge.net/projects/nanocap/`

The `NanoCap` documentation is outlined in the following sections.

Contents

1 Installation

There are three approaches to using `NanoCap`:

1. As a standalone application.
2. From source without rendering/GUI capabilities
3. From source with rendering/GUI capabilities

The installation procedures involved in each of the options above vary with increasing complexity yet this is balanced with an increase in versatility. For example, `NanoCap` compiled from source with with rendering and GUI capabilities can be used in parallelised code to produce and visualise multiple structures.

1.1 Requirements

1. As a standalone application.

   `NanoCap` is built into a DMG for OSX and a .EXE for Windows. The required libraries listed below are bundled but not modified in line with the associated licenses.

   OSX: `NanoCap` works *straight out of the box* simply extract the application from the DMG and drag it into the **Applications** folder.

   Windows: `NanoCap` requires the Microsoft Visual C++ 2008 Redistributable Package which can be obtained from:

   `http://www.microsoft.com/en-au/download/details.aspx?id=29`

2. From source without rendering/GUI capabilities

   - NumPy - Version 1.6.2
   - Scipy - Version 0.11.0
   - sqlite3 Version 2.6.0 (bundled with Python)
   - C compiler (e.g. GCC)
   - Fortran compiler (e.g. GFortran)

3. From source with rendering/GUI capabilities

   - Qt - Version 4.8.5
   - PySide - Version 1.1.1 (depends on Qt)
   - VTK - Version 5.8 (+ Python Wrappers)

Installation of the dependencies above is platform dependent and there are multiple methods of achieving the required python working environment. The simplest options are using package managers or binary distributions, i.e:

OSX:

- Homebrew `http://brew.sh`
- MacPorts `http://www.macports.org`
- Fink `http://www.finkproject.org/`

Windows:

- PythonXY `http://code.google.com/p/pythonxy`
- Enthought Python `http://www.enthought.com/products/epd`

Linux:

- apt-get
- YUM (RPM Package Manager)

1.2 Installing from source

After obtaining and installing the previously outlined requirements, a tar ball of `NanoCap` can be downloaded from:

`http://sourceforge.net/projects/nanocap/files/src/`

As an alternative, the latest release can be checked out from the `mercurial` repository:

```
bash-3.2$ hg clone http://hg.code.sf.net/p/nanocap/code-0 nanocap-code-0
```

After download and unpacking (if required), installation proceeds in the typically python fashion:

```
bash-3.2$ python setup.py install
```

The installation runs a configure script (generated by `autoconf`) that detects available C and Fortran compilers and builds the associated shared libraries. To test for a successful installation, simply attempt to import `NanoCap` in a python terminal.

```
bash-3.2$ python
Python 2.7.3 (default, Jul 19 2012, 13:57:53)
[GCC 4.2.1 Compatible Apple Clang 3.1 (tags/Apple/clang-318.0.61)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import nanocap
>>> print nanocap.__version__
1.0b9
```

The successful import and printing of the version number indicates the libraries installed correctly. To test the installation further try running the example scripts shown in Section 10.

If the GUI and rendering libraries have been successfully installed then `NanoCap`can be ran either as an application or as libraries. To run the GUI enabled `NanoCap`simply type:

```
bash-3.2$ nanocap
```

into the command prompt.

2 Generating a Single Structure

There are two methods of generating structures using `NanoCap`, producing structures individually or a batch of structures found using a structure search. Creation of a single structure is useful when it is required quickly and there is little need for the structure to be the lowest in energy.

A single structure is constructed via:

File–>New Structure–>Single Structure

which displays the current list of available structures in `NanoCap`: Adding a structures produces

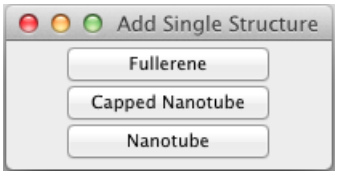

Figure 1: `NanoCap` structure list for adding a single structure

a blank *template* containing no points or atoms. Each of the structures listed above are discussed in the following sections.

2.1 Fullerenes

The options to define the input parameters for the construction of the fullerene are displayed in the **Calculations–>Input** options:

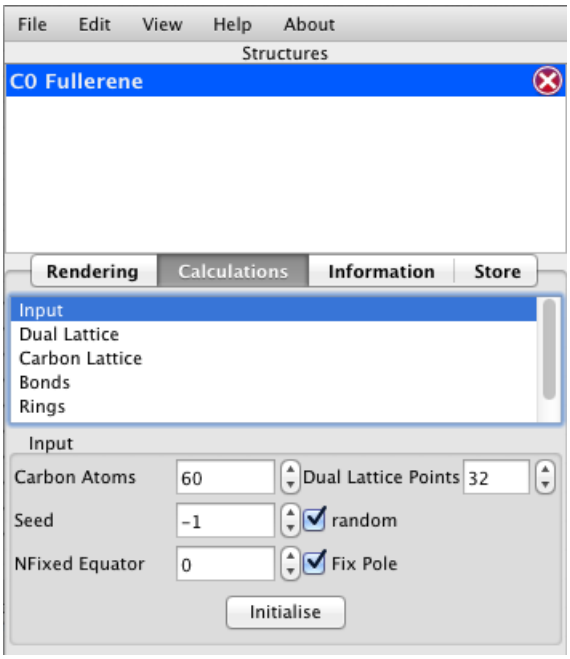

Figure 2: `NanoCap` input options for a fullerene

Here you can set the options for:

- the number of dual lattice points or carbon atoms
- the seed for the initial random arrangement of dual lattice points

- the number of dual lattice points to hold fixed at either the poles or the equator

Upon on clicking *Initialise* the dual lattice points belonging to the fullerene will be constructed via the following:

$$\begin{aligned} x_i &= \sqrt{1-z_0}\cos(t_0) \\ y_i &= \sqrt{1-z_0}\sin(t_0) \\ z_i &= z_0 \end{aligned} \tag{1}$$

where z_0 and t_0 are two random numbers in the range $[-1,1]$ and $[0,2\pi]$. To visualise these points check the options outlined in Section 7. The process of optimisation of these points is described in Section 5

2.2 Nanotubes

To construct the carbon lattice the user has the option for either a finte-length nanotube or one that is periodic along the axial direction.

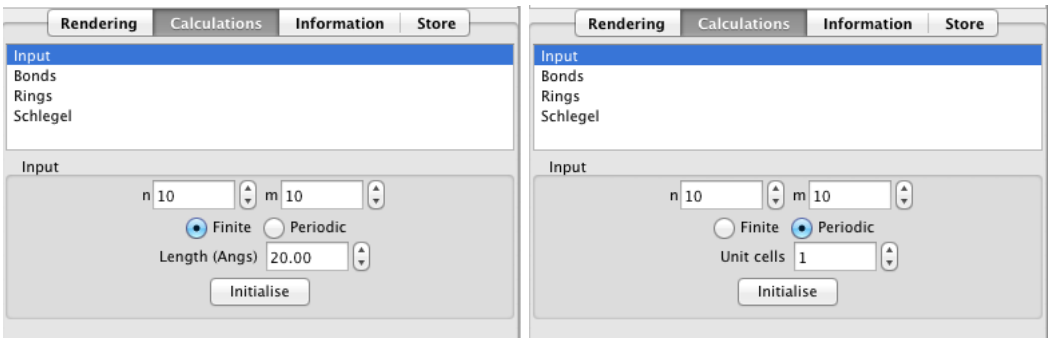

Figure 3: Nanotube construction options

2.2.1 Finite tubes

Finite tubes are constructed as close to a user defined length as possible. This is done by constructing *strips* of basis points along the chiral vector (n, m):

$$\begin{aligned} \mathbf{P}_i &= n_i\mathbf{a_1} + m_i\mathbf{a_2} \\ \mathbf{a_1} &= \frac{\sqrt{3}a_c}{2}(\sqrt{3}, 1) \\ \mathbf{a_2} &= \frac{\sqrt{3}a_c}{2}(\sqrt{3}, -1) \end{aligned}$$

where a_c is the carbon bond length of 1.421 Å. The incremented values (n_i, m_i) range from (0,0) to (n, m) and depends on:

$$\begin{aligned} n_i\text{++} &\quad \text{if} \quad m_i/(2n_i + m_i) > m/(2n + m) \\ m_i\text{++} &\quad \text{if} \quad m_i/(2n_i + m_i) \leq m/(2n + m) \end{aligned}$$

After each new row of points, the origin is translated in the z direction by $\sqrt{3}a_c$. The current distance along the nanotube axis is then compared against the user defined length to determine if another strip should be added. The user defined length is inputted in the **Calculations**–>**Input** options.

At each basis point **P** at position (p_x, p_z), the positions of the carbon atoms (**A** and **B**) and dual lattice points (**D**) are given by:

$$\begin{aligned} \mathbf{A} &= (p_x, p_z) \\ \mathbf{B} &= (p_x + a_c, p_z) \\ \mathbf{D} &= (p_x + 2a_c, p_z) \end{aligned}$$

2.2.2 Periodic tubes

Periodic tubes are constructed using a user defined number of unit cells in the z direction. The **periodic length** L of a nanotube of chirality (n, m) with u unit cells is given by:

$$L = u * |\mathbf{T}|$$
$$\mathbf{T} = t_1\mathbf{a_1} + t_2\mathbf{a_2}$$

where the coefficients t_1 and t_2 have no common divisors except for unity and are given by:

$$t_1 = (2m + n)/d_R$$
$$t_2 = -(2n + m)/d_R$$
$$d_R = \texttt{gcd}(2n + m, 2m + n)$$

During construction the carbon atoms and dual lattice points are constructed as for the finite tubes with the replacement of the user-defined length with the periodic length. After construction, any points surpassing the periodic length are removed.

The number of unit cells can also be found in the **Calculations**–>**Input** options.

The **periodic length** can be found in the **Information** options tab. This will be required by simulation software if the nanotube is to be used in a periodic simulation.

2.3 Capped Nanotubes

The options to define the input parameters for the construction of the capped nanotube are displayed in the **Calculations**–>**Input** as shown in Fig 4:

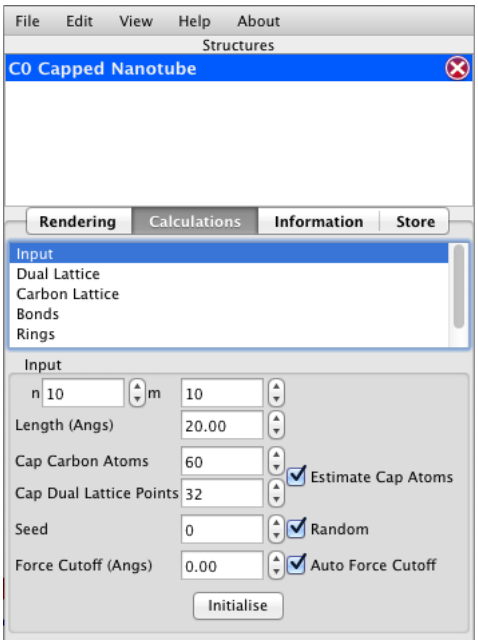

Figure 4: `NanoCap` input options for a capped nanotube

The following input options can be set for the capped nanotube construction.

- the chirality (n, m)
- the length of the nanotube (Å)

- the number of cap carbon atoms and dual lattice points (or enabled the estimation based upon the nanotube density)
- the seed for the initial random cap point placement
- the force cutoff relating to the dual lattice force field (Section 4.1)

3 Generating Multiple Structures

3.1 Structure Search

As an alternative to constructing a single structure, NanoCapincludes a tool for finding low energy structures by performing a *structure search*. This is useful when finding the lowest energy topology is important or when an ensemble of structures are required.

The structure search options are accessed through the **File–>New Structure–>Structure Search** menu. The structure search window includes a panel of parameters that define the structure type and search criteria. Here the user can define the required structural properties as well as the force field and optimiser to use during the search. The search results are dynamically displayed in a table where the user can browse the details of each structure. During the search the user can view an individual structure by pressing the **view** button in the associated row in the table. This will load the structure into the main NanoCap window. An example of the structure search window during a search in shown in Fig. 5

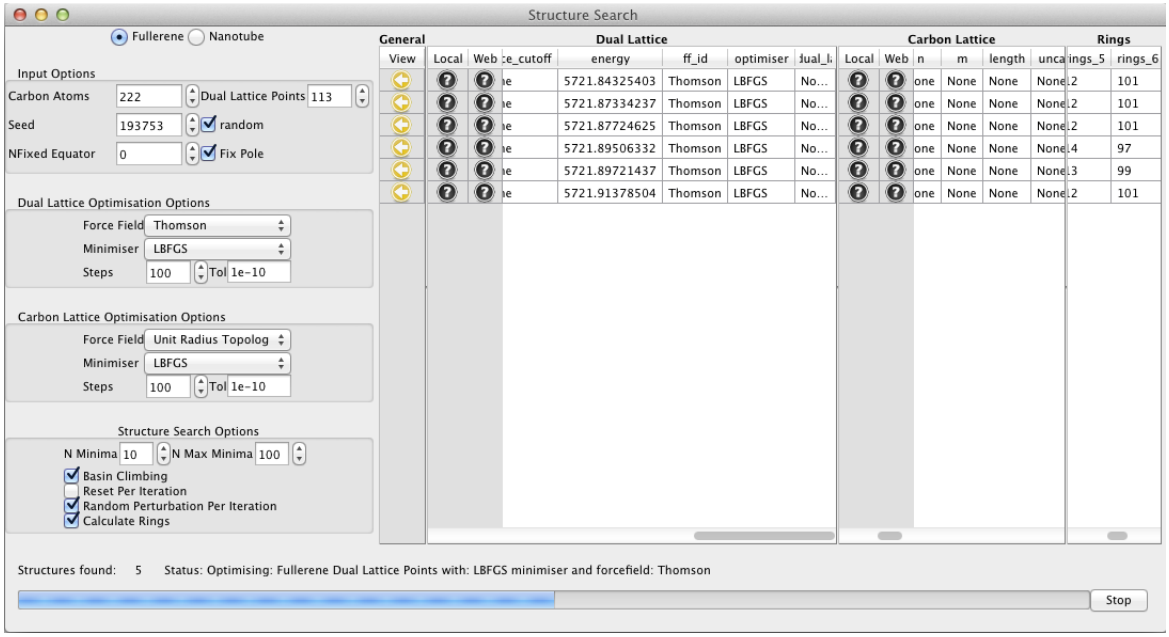

Figure 5: NanoCap structure search window during a search

After a structure search, the current set of results can be compared against the local and online (in future releases) databases. The columns in the structure search table denoted **Local** and **Web** indicate the presence of the structure in the associated databases. If a structure is not found it's icon will change (to a + symbol) and it can be immediately added to the corresponding database with a single click. Results from a typical structure search after checking against the local NanoCap database are shown in Fig. 6

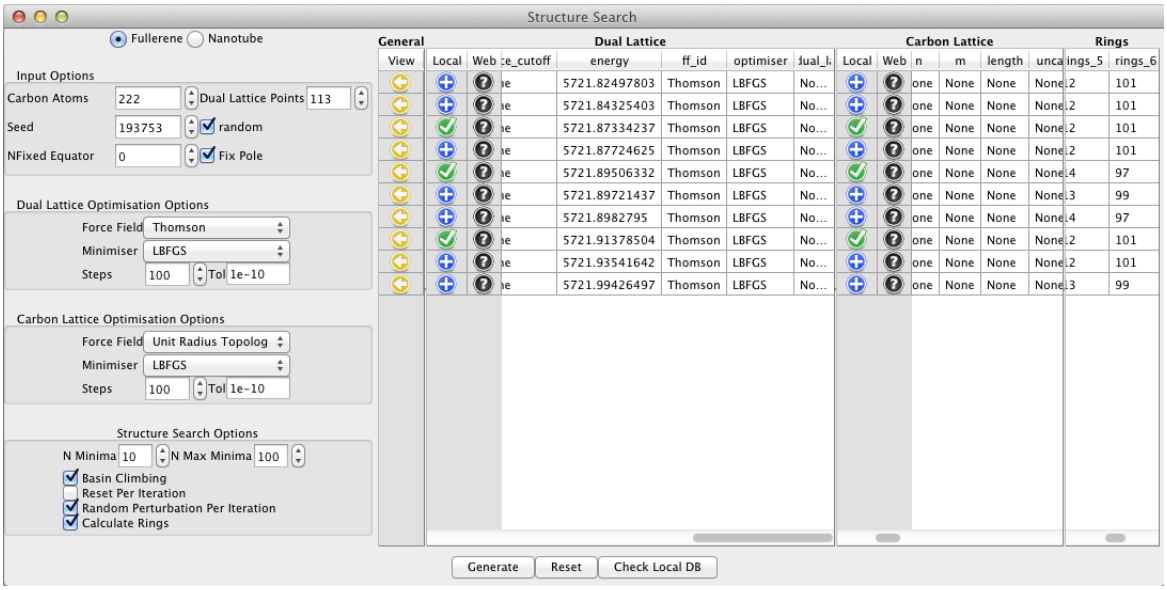

Figure 6: `NanoCap` structure search window after a search and results have been checked against the local database

4 Force Fields

`NanoCap` implements force fields for the optimisation of the both the dual lattices and carbon lattices of each structure. When a structure is saved information relating to the force field is also stored. This allows the same topologies to be optimised by various force fields. Outlined in the next sections are the force fields currently available in `NanoCap` .

4.1 Dual Lattice Force Fields

Currently, only one force field is implemented to optimise a structure's dual lattice - labelled: *The Thomson Problem*. The total energy of a system of N_D dual lattice points is given by the sum of pair interaction energies:

$$\phi = \sum_{i=1}^{N_D} \sum_{j=i+1}^{N_D} \frac{1}{|\mathbf{r_i} - \mathbf{r_j}|}$$

where $\mathbf{r}$ denotes the position vector of each point. When the dual lattice belongs to a fullerene, the full system is included in the loop of pair interactions. For a capped nanotube however, there is restriction to the points included in the force field calculation. A cutoff length is introduced along the nanotube beyond which points are exclude from the force evaluation. This is required to ensure a uniform arrangement of points in the capped region and reduce the concentration of points in the apex of the cap. This cutoff length is automatically determined basen upon the density of points in the nanotube but can be set manually in the options described in Section 2.3

4.2 Carbon Lattice Force Fields

Currently there are 3 force fields implemented in `NanoCap`. These are selected in the **Calculations–>Carbon Lattice** options as shown in Fig. 7.

Each force field is described below:

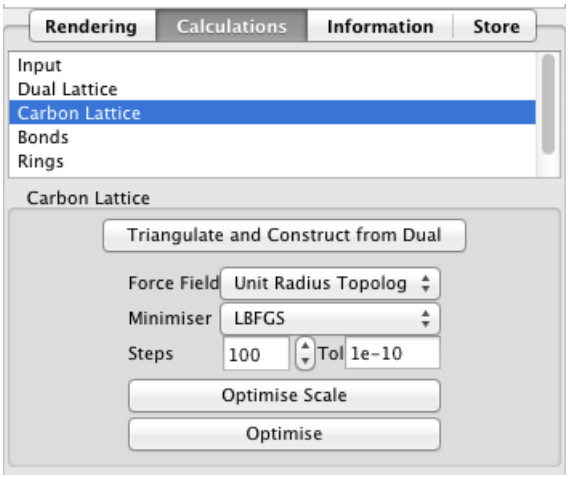

Figure 7: `NanoCap` input options for a fullerene

1. **Unit Radius Topology**

   By default, when a carbon lattice is constructed a carbon force field is not used and the structure is simply a result of the triangulated dual lattice. This generates a topology of unit radius (cylindrical if a capped nanotube) and as such is labelled the

2. **Scaled Topology**

   The simplest force field that produces a carbon structure of physical dimensions is the **scaled topology** force field. This forcefield uses the ideal C-C bond length of 1.421 Å to optimise the carbon lattice. The fictitious *energy* is given by the sum of squares deviation from this ideal bond length. This force field only does return derivatives and as such can only be used with the **MC** or **SIMPLEX** optimisers (see Section 5).

3. **EDIP**

   The most sophistic force field implemented in `NanoCap` is the Environmental Dependence Interatomic Potential (EDIP). Using EDIP produces the most physically sound structures and its use is recommended. For a theoretical description of EDIP please refer to the following paper:

   Generalizing the environment-dependent interaction potential for carbon

   N. A. Marks *Phys. Rev. B 63, 035401 2000*

   url: `http://journals.aps.org/prb/abstract/10.1103/PhysRevB.63.035401`

5 Optimisation

The underlying design of `NanoCap` and the generalisation of point sets and forcefields, allows for a great deal of flexibility in optimisation routines. The same optimisation routine that is used to minimised the total dual lattice energy can be used to optimise the carbon lattice using a force field such as EDIP. Currently three methods of minimising the total energy are implemented and can be selected in either the **Calculations–>Dual Lattice** or **Calculations–>Carbon Lattice** options as shown in Fig. 8

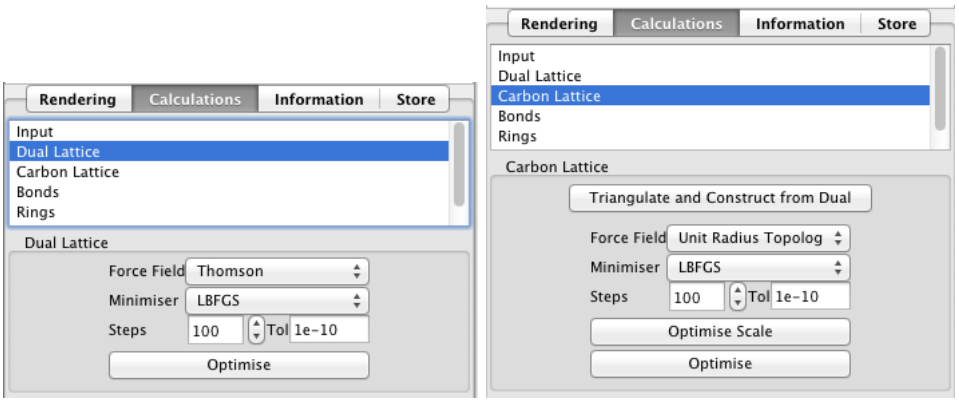

Figure 8: `NanoCap` Optimisation options are shown for both the dual lattice and carbon lattice option windows.

These options include the number of minimisation steps and the tolerance used to determine convergence. A brief description of each of the optimisers is given below:

1. **L-BFGS**

   The tastes, most robust optimiser is the Limited-memory Broyden–Fletcher–Goldfarb–Shanno method (LBFGS). This routine requires forces which are used to iteratively update an estimate of the Hessian matrix, which in turn is used to define new directions along which to perform line searches. `NanoCap` uses the the LBFGS method as implemented in the `scipy.optimise` libraries (`http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html`).

   A Limited Memory Algorithm for Bound Constrained Optimization R. H. Byrd, P. Lu and J. Nocedal. *(1995), SIAM Journal on Scientific and Statistical Computing, 16, 5, pp. 1190-1208.*

   L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization C. Zhu, R. H. Byrd and J. Nocedal. *(1997), ACM Transactions on Mathematical Software, 23, 4, pp. 550 - 560.*

   L-BFGS-B: Remark on Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization J.L. Morales and J. Nocedal. *(2011), ACM Transactions on Mathematical Software, 38, 1.*

2. **SD**

   The steepest descent (SD) method minimises energy by simply following the direction of force (or the negative gradient of the potential field). The step size per iteration is determined by the current magnitude of force. The SD method is useful when visualising the optimisation process.

3. **SIMPLEX**

   The simplex method is an optimisation routine which does not require the calculation of forces. As such is the only optimiser that works with the **Scaled Topology** force field. The implementation in `NanoCap` again comes from the `scipy.optimise` libraries which implement algorithms presented in the following publications:

A Simplex Method for Function Minimization Nelder, J.A. and Mead, R. *The Computer Journal, 7, pp. 308-313*

Direct Search Methods: Once Scorned, Now Respectable Wright, M.H. *Numerical Analysis 1995, Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis pp. 191-208*

6 Storing, Loading and Exporting

Each individual structure loaded or created can be checked against the local and online NanoCap databases. This is carried out in the **Storage** window on the main toolbar as shown in Fig. 9.

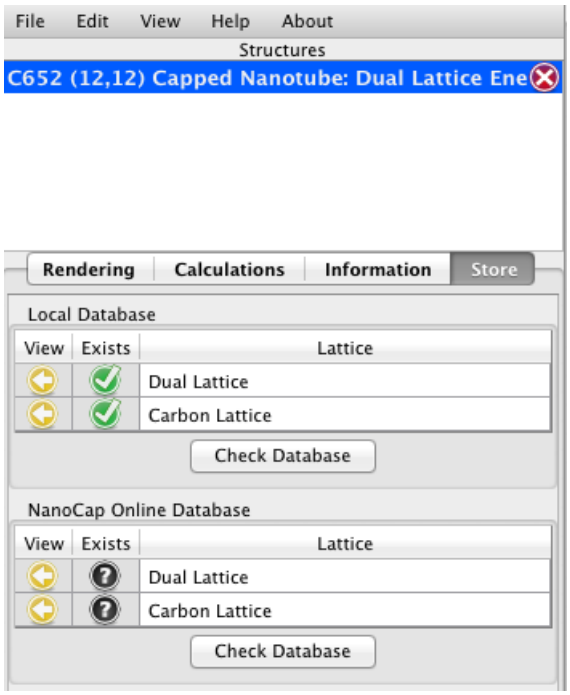

Figure 9: Checking a structure against the local and online NanoCap databases

6.1 The Local NanoCap Database

One of the main features of NanoCap is the local database of structures which allows the user to efficiently store previously generated structures. By default the database is stored in the NanoCap user directory (.nanocap). The database can be accessed in several ways; using the NanoCap GUI, using the NanoCap scripts or using an external SQLite client. Using the GUI,the **Database Viewer** window is located in the **View–>Local Database** menu.

The database viewer provides an interactive interface to the NanoCap database allowing real time searching and loading of stored structures (Fig. 10). Additional search parameters can be added by pressing the **Select Properties** button which displays all possible database fields. When searching for a structure, logical expressions can be used for each file. For example, if '`> 100`' is entered into the `Natoms` fields then only structures with more than 100 atoms will be returned. The column labelled '`>View`' provides buttons to load a structure into the main NanoCap window. The loaded structures can then be modified, for example re-optimised with a different force field.

To access the database via Python scripts, the NanoCap libraries can be used. For examples of these scripts see Section 10.

As the NanoCap database is written using SQLite, a suitable database client can be used to view the database. An example using the `sqlite3` client is shown below:

```
bash-3.2$ sqlite3 ~/.nanocap/nanocap.db
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .headers on;
sqlite> .mode column
```

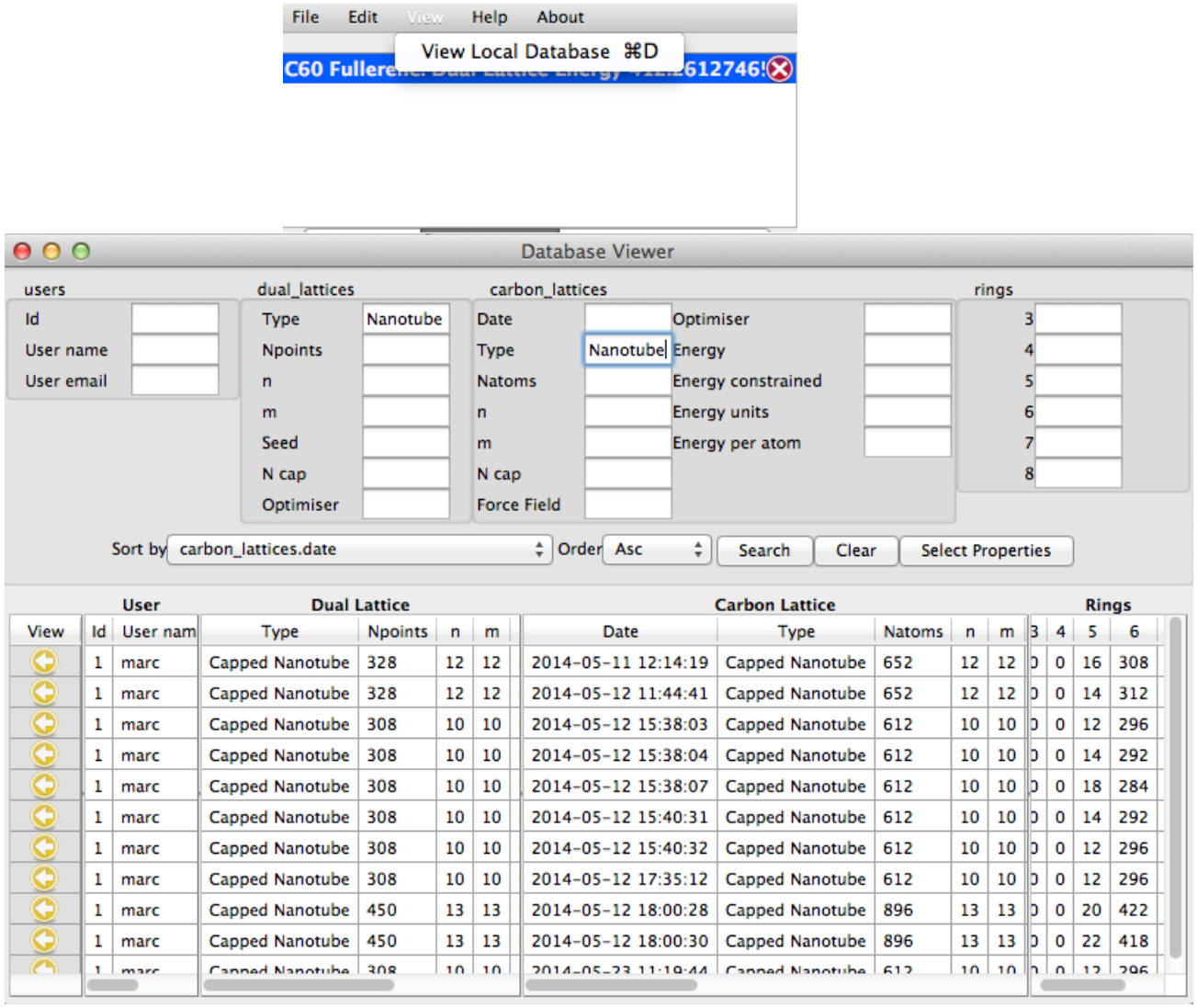

Figure 10: `NanoCap` database viewer

```
sqlite> select id,type,Natoms,energy,ff_id from carbon_lattices
   ...> where type="Fullerene" and ff_id="EDIP";
id          type        natoms      energy            ff_id
----------  ----------  ----------  ----------------  ----------
11          Fullerene   220         -1550.4155355469  EDIP
50          Fullerene   200         -1403.7245939477  EDIP
51          Fullerene   200         -1403.5958295534  EDIP
52          Fullerene   200         -1404.0657391286  EDIP
53          Fullerene   200         -1402.8514086776  EDIP
54          Fullerene   200         -1403.3230258698  EDIP
55          Fullerene   200         -1404.3263445713  EDIP
56          Fullerene   200         -1403.4800906205  EDIP
57          Fullerene   200         -1404.3965982273  EDIP
58          Fullerene   200         -1404.1298467551  EDIP
59          Fullerene   200         -1403.4657122674  EDIP
64          Fullerene   200         -1403.7245939160  EDIP
```

6.2 The Online NanoCap Database

The online `NanoCap` database is currently under construction.

6.3 Exporting

Each structure in the current structure list can be exported to file. The options for exporting a structure can be found in the **File–>Export Structure** menu. The full list of export options are shown in Fig. 11.

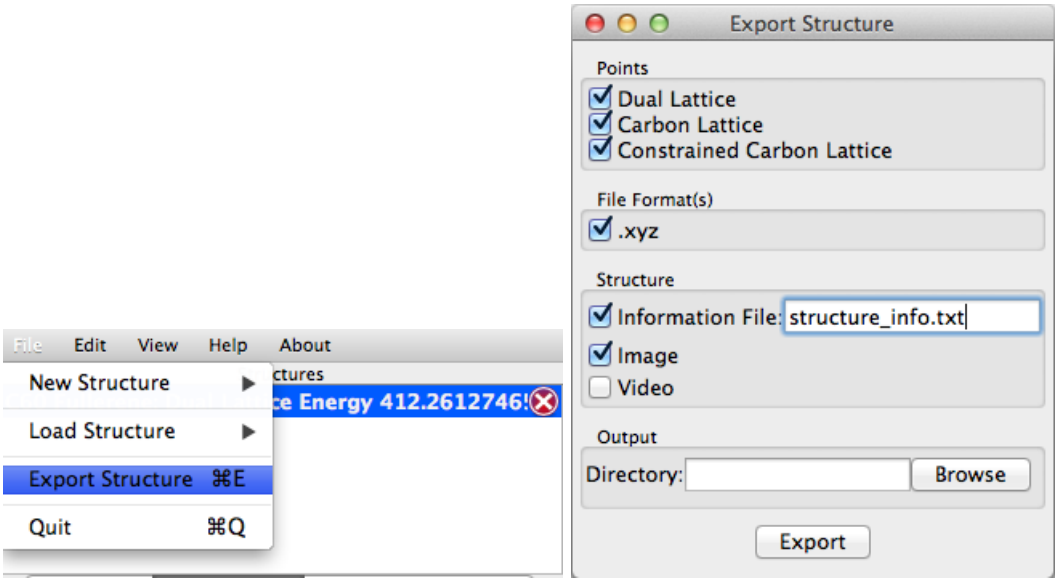

Figure 11: `NanoCap` export structure options

The export options include the ability to select which lattices are saved and in what file format. Currently only a simple *xyz* file format is implemented which contains a minimum amount of data (the number of points and positions). In addition to the topology of the structure, structural information can also be saved. This includes energies, dimensions, connectivity (ring statistics) amongst other things.Using the rendering capabilities of `NanoCap`, images of the structure can also be automatically saved. This includes a series of images that capture a full rotation of the structure. These images can be encoded into a movie to simplify future inspection of the stored structure. Finally, the export options require a parent directly to save the aforementioned data. Within this directory a new directory will be created whose name uniquely identifies the current structure.

7 Rendering

NanoCap provides a 3D rendering and interaction interface using the VTK libraries and the associated widgets in Qt. This allows real time inspection of the structures that are being constructed or previously found structures loaded from the database. Current capabilities include the rendering of both dual and carbon lattices, the carbon-carbon bonds, the ring network and the dimensions of the current structure. These options are dynamics, with any changes to the appearance of the current structure changing in real time. An example of the NanoCaprendering options and render window are shown in Fig. 12.

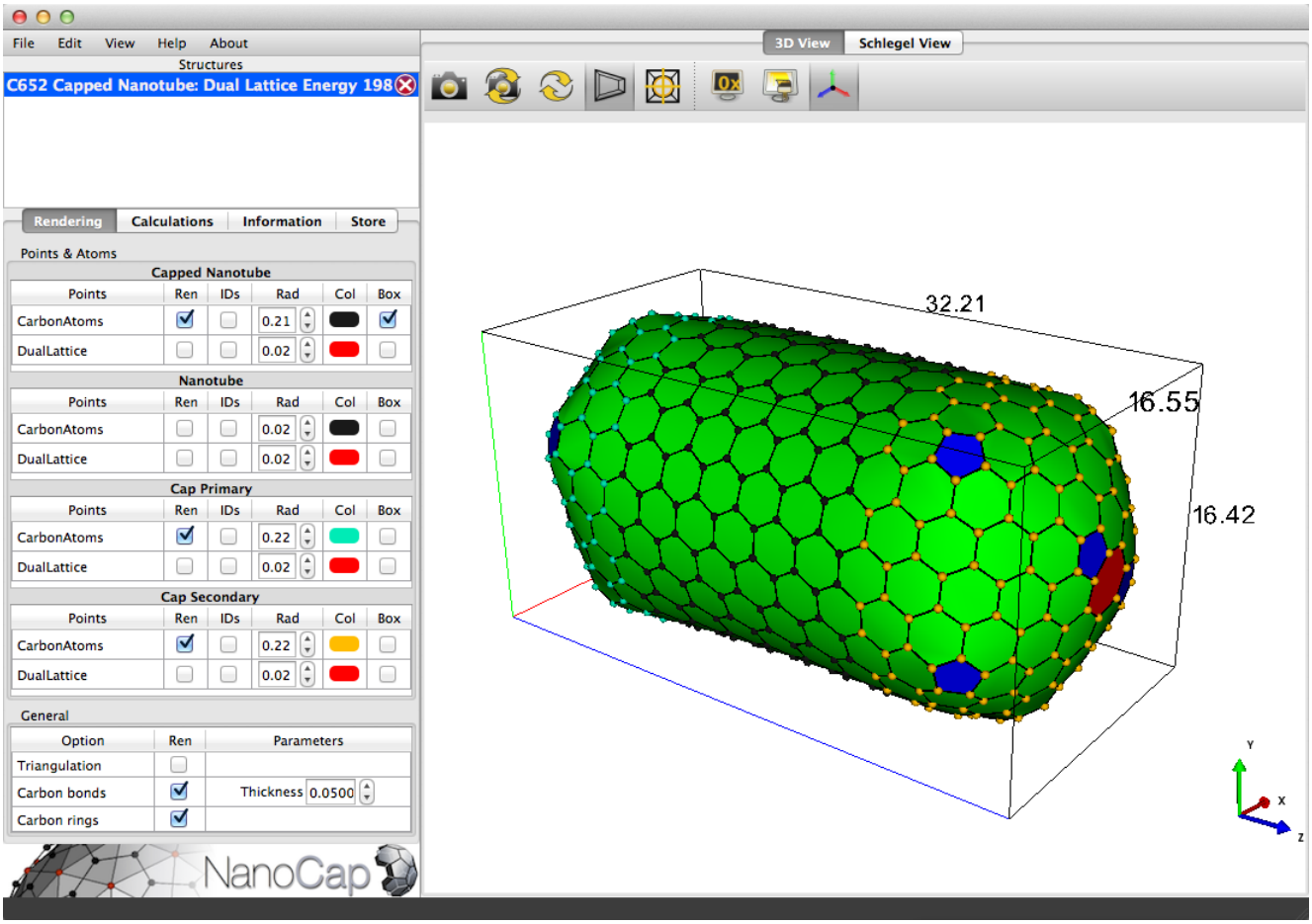

Figure 12: The NanoCap rendering options and the corresponding render window.

7.1 Schlegel View

In addition to the 3D view of the current structure a 2D projection (Schlegel) view can also be rendered. The two parameters involved in the calculation of this projection are accessed through **Calculations**–>**Schlegel**. The parameter Gamma(γ) determines the magnitude of the projection via:

$$\mathbf{r'} = (x_i, y_i)$$
$$x' = x + \gamma_s \cdot \frac{x}{|\mathbf{r'}|}$$
$$y' = y + \gamma_s \cdot \frac{y}{|\mathbf{r'}|}$$

The Cutoff value determines the points that are used for the projection. An example projection is shown in Fig. 13.

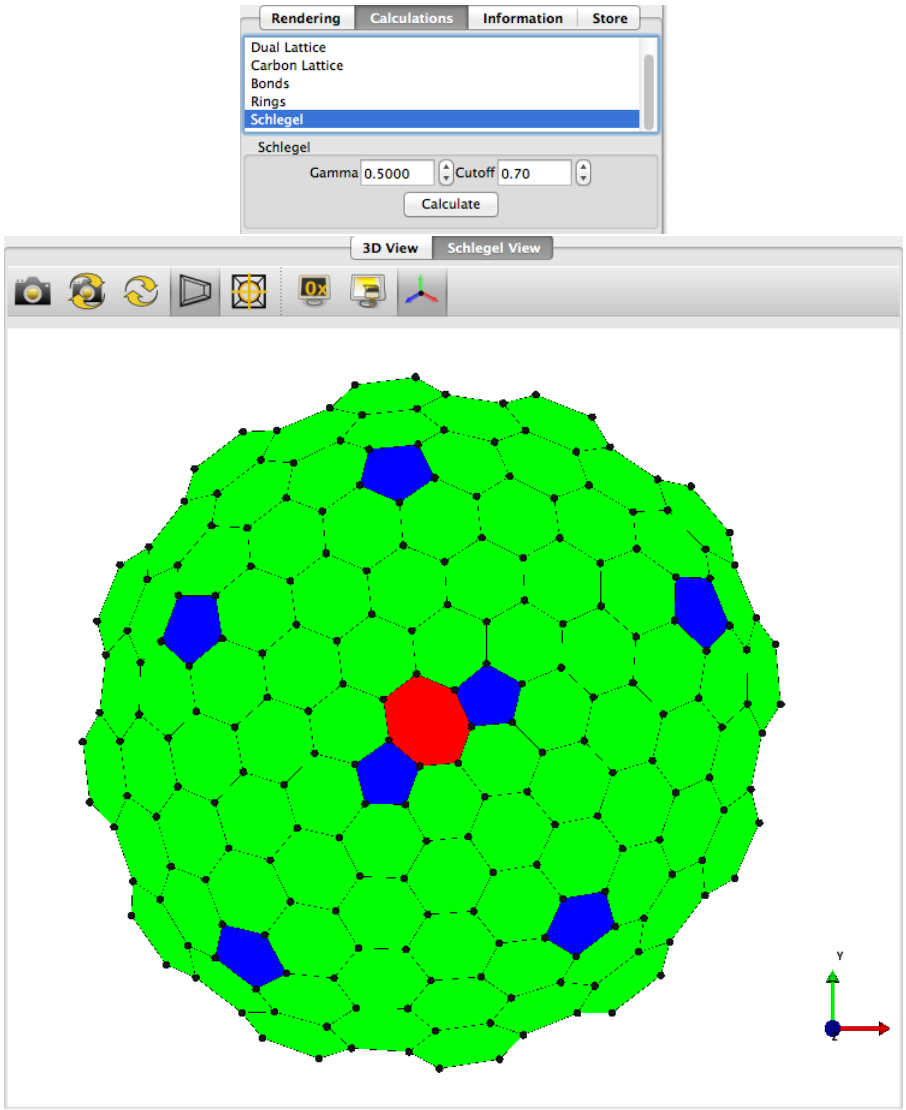

Figure 13: Schlegel options and render window.

8 Scientific Publications

There are two papers associated with the theory and implementation of `NanoCap`. If `NanoCap` is used in your work, please cite the following:

1. **Generalized method for constructing the atomic coordinates of nanotube caps**

   M. Robinson, I. Suarez-Martinez, and N. A. Marks *Phys. Rev. B 87, 155430 2013*

   Abstract:

   A practical numerical method for the rapid construction of nanotube caps is proposed. Founded upon the notion of lattice duality, the algorithm considers the face dual representation of a given nanotube which is used to solve an energy minimization problem analogous to The Thomson Problem. Not only does this produce caps for nanotubes of arbitrary chirality, but the caps generated will be physically sensible and in most cases the lowest energy structure. To demonstrate the applicability of the technique, caps of the (5,5) and the (10,0) nanotubes are investigated by means of density-functional tight binding (DFTB). The calculation of cap energies highlights the ability of the algorithm to produce lowest energy caps. Due to the preferential construction of spherical caps, the technique is particularly well suited for the construction of capped multiwall nanotubes (MWNTs). To validate this proposal and the overall robustness of the algorithm, a MWNT is constructed containing the chiralities (9,2)@(15,6)@(16,16). The algorithm presented paves the way for future computational investigations into the physics and chemistry of capped nanotubes.

   url: `http://journals.aps.org/prb/abstract/10.1103/PhysRevB.87.155430`

2. **NanoCap: A Framework for Generating Capped Carbon Nanotubes and Fullerenes**

   M. Robinson and N. A. Marks *Com. Phys. Comm 2014*

   Abstract:

   NanoCap provides both libraries and a standalone application for the construction of capped nanotubes of arbitrarily chirality and fullerenes of any radius. Structures are generated by constructing a set of optimal dual graph topologies which are subsequently optimised using a carbon interatomic potential. Combining this approach with a GUI featuring 3D rendering capabilities allows for the rapid inspection of physically sensible structures which can be used as input for molecular simulation.

   url: *tba*

9 Code

9.1 Non-GUI Class Structure

The basic class structure of the non-GUI elements of `NanoCap` are is shown in Fig. 14. The class diagrams are constructed with the help of `pyreverse` and `graphviz`.

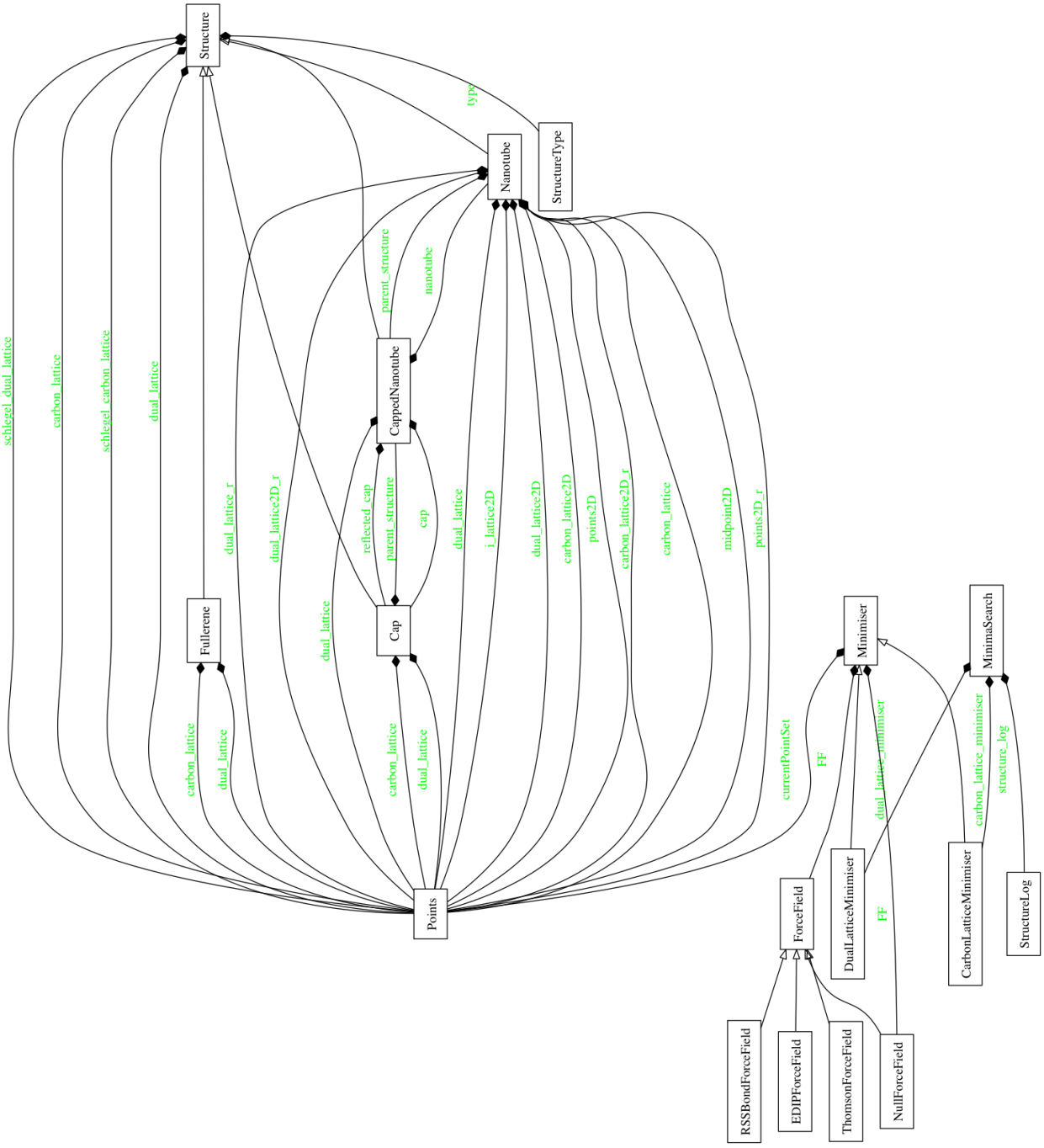

Figure 14: Class diagram of the non GUI classes in `NanoCap`.

9.2 GUI Class Structure

The main GUI elements are shown in the class diagram in Fig. 15.

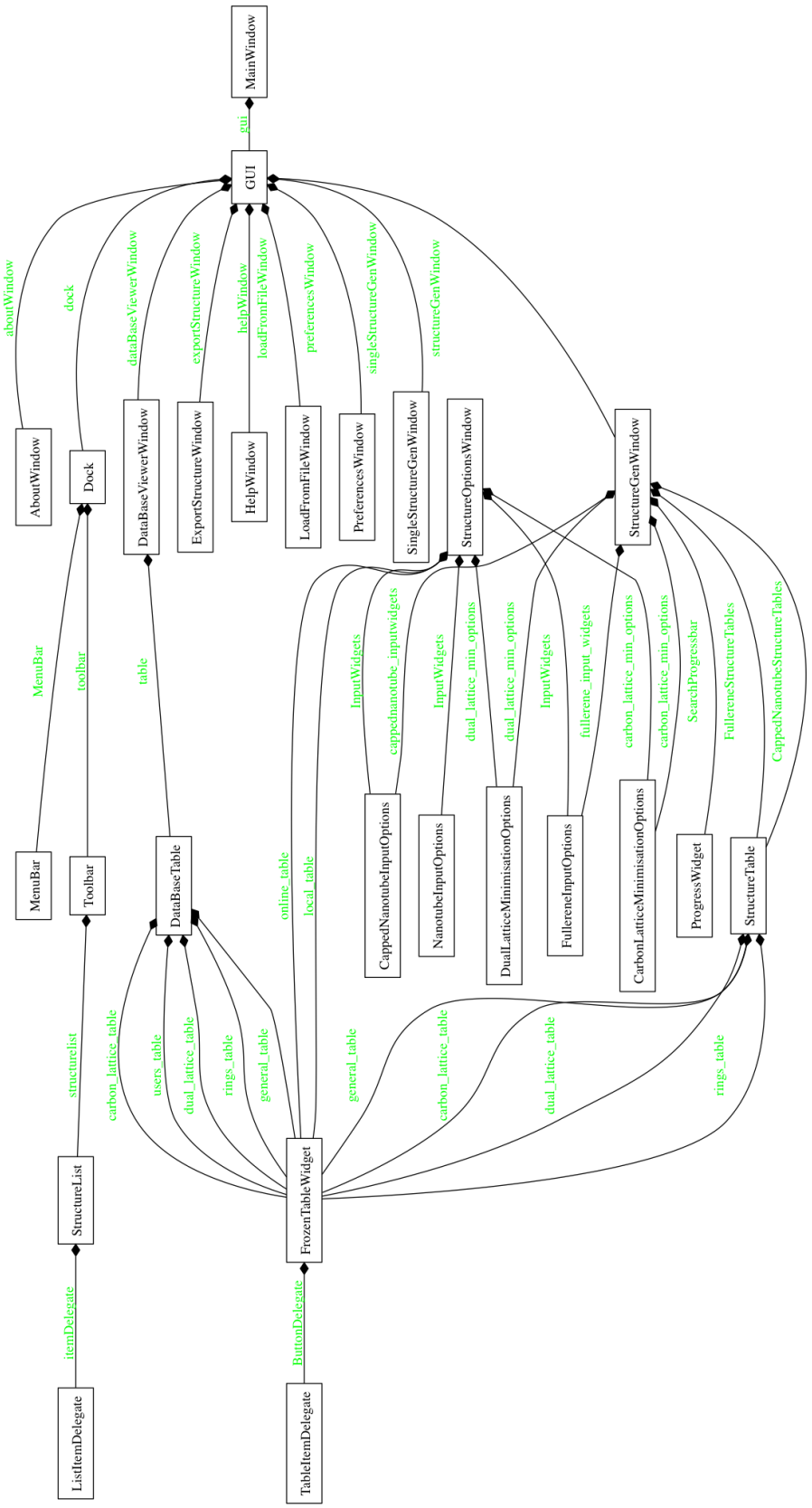

Figure 15: Class diagram of the GUI classes in `NanoCap`.

10 Examples

The following section lists example scripts that utilise the `NanoCap` libraries.

These examples can be found in the `example_scripts` directory of the `NanoCap` source.

10.1 Nanotube Construction

`nanocap_nanotube.py` an example script to construct an uncapped nanotube. The code is shown below.

```
'''
-=-=-=-=-=-=-=-=-=-=-=-=NanoCap=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Copyright Marc Robinson  2014
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

A script to construct an uncapped
nanotube.

Input:
    n,m = Chirality (n,m)
    l = length
    u = number of unit cells
    p = periodic
Output:
    xyz file containing carbon
    lattice

if periodic, length is ignored and
unit cells is used
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
'''
from nanocap.structures import nanotube
from nanocap.core import output

n=6
m=4
l=5.0
u=1
p=True

my_nanotube = nanotube.Nanotube()
my_nanotube.construct(n,m,length=l,
                      units=u,periodic=p)

output.write_xyz("nanotube_carbon_lattice",
                 my_nanotube.carbon_lattice)
```

10.2 Fullerenes

10.2.1 Single Fullerene Construction

`nanocap_single_fullerene.py` is an example script to construct and save a fullerene. The code is shown below.

```python
'''
-=-=-=-=-=-=-=-=-=-=-=-=-=NanoCap=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Copyright Marc Robinson  2014
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

A script to construct a fullerene

Input:
    N_carbon = Number of carbon atoms
    dual_lattice_force_field = force field
                               for dual lattice
    carbon_force_field = force field
                        for carbon lattice
    dual_lattice_mintol= energy tolerance for
                         dual lattice optimisation
    dual_lattice_minsteps= steps for dual lattice
                            optimisation
    carbon_lattice_mintol=as above for carbon lattice
    carbon_lattice_minsteps=as above for carbon lattice
    optimiser=optimsation algorithm
    seed = seed for initial cap generation

Output:
    xyz files containing dual lattice
    and carbon lattice

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
'''

import sys,os,random,numpy
from nanocap.core.minimisation import DualLatticeMinimiser, \
                                      CarbonLatticeMinimiser
from nanocap.structures.fullerene import Fullerene
from nanocap.core.output import write_xyz

N_carbon = 200

dual_force_field = "Thomson"
carbon_force_field = "EDIP"
dual_lattice_mintol=1e-10
dual_lattice_minsteps=100
carbon_lattice_mintol=1e-10
carbon_lattice_minsteps=100
optimiser="LBFGS"
seed = 12345

my_fullerene = Fullerene()
my_fullerene.construct_dual_lattice(N_carbon=N_carbon,seed=seed)
my_fullerene.set_fix_pole(False)
my_fullerene.set_nfixed_to_equator(0)


Dminimiser = DualLatticeMinimiser(FFID=dual_force_field,
                                  structure = my_fullerene)
Dminimiser.minimise(my_fullerene.dual_lattice,
                    min_type=optimiser,
                    ftol=dual_lattice_mintol,
                    min_steps=dual_lattice_minsteps)
```

```
outfilename = "C{}_dual_lattice.xyz".format(N_carbon)
write_xyz(outfilename,my_fullerene.dual_lattice)

my_fullerene.construct_carbon_lattice()

Cminimiser = CarbonLatticeMinimiser(FFID=carbon_force_field,
                                    structure = my_fullerene)

Cminimiser.minimise_scale(my_fullerene.carbon_lattice)
Cminimiser.minimise(my_fullerene.carbon_lattice,
                    min_type=optimiser,
                    ftol=carbon_lattice_mintol,
                    min_steps=carbon_lattice_minsteps)

outfilename = "C{}_carbon_atoms.xyz".format(N_carbon)
write_xyz(outfilename,my_fullerene.carbon_lattice)
outfilename = "C{}_carbon_atoms_constrained.xyz".format(N_carbon)
write_xyz(outfilename,my_fullerene.carbon_lattice,constrained=True)
```

10.2.2 Constructing Multiple Fullerenes

`nanocap_bulk_fullerenes.py` is an example script to perform a structure search to construct and save multiple fullerenes. The code is shown below.

```
'''
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=NanoCap=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Copyright Marc Robinson  2014
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

A script to construct a series of
fullerenes

Input:
    N_carbon = number of carbon atoms
                in the fullerene
    dual_lattice_force_field = force field
                               for dual lattice
    carbon_force_field = force field
                         for carbon lattice
    dual_lattice_mintol= energy tolerance for
                         dual lattice optimisation
    dual_lattice_minsteps= steps for dual lattice
                           optimisation
    carbon_lattice_mintol=as above for carbon lattice
    carbon_lattice_minsteps=as above for carbon lattice
    optimiser=optimsation algorithm
    seed = seed for initial cap generation
    N_nanotubes = required number of structures
    N_max_structures = maximum number of possible
                       structures to search through
    basin_climb = True/False - climb out of
                  minima
    calc_rings = True/False - calculate rings for
                  each structure

Output:
```

```
    -A structure log in myStructures.out

    -xyz files containing the carbon lattices

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
'''

import sys,os,random,numpy
from nanocap.core.minimisation import DualLatticeMinimiser, \
                                       CarbonLatticeMinimiser
from nanocap.core.minimasearch import MinimaSearch
from nanocap.structures.fullerene import Fullerene
from nanocap.core.output import write_points

N_carbon = 200
dual_lattice_minimiser = "Thomson"
carbon_lattice_minimiser = "EDIP"
seed = 12345

N_fullerenes = 5
N_max_structures = 20
basin_climb = True
calc_rings = True

dual_lattice_minimiser = "Thomson"
carbon_lattice_minimiser = "EDIP"
dual_lattice_mintol=1e-10
dual_lattice_minsteps=100
carbon_lattice_mintol=1e-10
carbon_lattice_minsteps=100
optimiser="LBFGS"
seed = 12345

my_fullerene = Fullerene()
my_fullerene.construct_dual_lattice(N_carbon=N_carbon,seed=seed)
my_fullerene.construct_carbon_lattice()
my_fullerene.set_fix_pole(False)
my_fullerene.set_nfixed_to_equator(0)

Dminimiser = DualLatticeMinimiser(FFID=dual_lattice_minimiser,
                                  structure = my_fullerene,
                                  min_type= "LBFGS",
                                  ftol = 1e-10,
                                  min_steps = 10)

Cminimiser = CarbonLatticeMinimiser(FFID=carbon_lattice_minimiser,
                                  structure = my_fullerene,
                                  min_type= "LBFGS",
                                  ftol = 1e-10,
                                  min_steps = 10)

Searcher = MinimaSearch(Dminimiser,
                        carbon_lattice_minimiser= Cminimiser,
                        basin_climb=basin_climb,
                        calc_rings=calc_rings)

Searcher.start_search(my_fullerene.dual_lattice,
                      N_fullerenes,
```

```
                        N_max_structures)

Searcher.continue_search(my_fullerene.dual_lattice,
                         N_fullerenes,
                         N_max_structures)

Searcher.structure_log.write_log(os.getcwd(),"myStructures.out")

for i,structure in enumerate(Searcher.structure_log.structures):
    outfilename = "C{}_carbon_atoms_{}"
    outfilename = outfilename.format(structure.carbon_lattice.npoints,i)
    write_points(outfilename,
                 structure.carbon_lattice,
                 "xyz")
```

10.3 Capped Nanotube Construction

10.3.1 Single Capped Nanotube Construction

`nanocap_single_capped_nanotube.py` is an example script to construct and save a capped nanotube. The code is shown below.

```
'''
-=-=-=-=-=-=-=-=-=-=-=-=-=NanoCap=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Copyright Marc Robinson  2014
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

A script to construct a capped
nanotube.

Input:
    n,m = Chirality (n,m)
    l = length
    cap_estimate = estimate cap from
                   tube density
    dual_lattice_force_field = force field
                               for dual lattice
    carbon_force_field = force field
                         for carbon lattice
    dual_lattice_mintol= energy tolerance for
                         dual lattice optimisation
    dual_lattice_minsteps= steps for dual lattice
                           optimisation
    carbon_lattice_mintol=as above for carbon lattice
    carbon_lattice_minsteps=as above for carbon lattice
    optimiser=optimsation algorithm
    seed = seed for initial cap generation

Output:
    xyz files containing dual lattice
    and carbon lattice

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
'''
import sys,os,random,numpy
from nanocap.core import minimisation
```

```
from nanocap.structures import cappednanotube
from nanocap.core import output

n,m = 7,3
l = 10.0
cap_estimate = True

dual_force_field = "Thomson"
carbon_force_field = "EDIP"
dual_lattice_mintol=1e-10
dual_lattice_minsteps=100
carbon_lattice_mintol=1e-10
carbon_lattice_minsteps=100
optimiser="LBFGS"
seed = 12345

my_nanotube = cappednanotube.CappedNanotube()

my_nanotube.setup_nanotube(n,m,l=l)

if(cap_estimate):
    NCapDual = my_nanotube.get_cap_dual_lattice_estimate(n,m)

my_nanotube.construct_dual_lattice(N_cap_dual=NCapDual,seed=seed)

my_nanotube.set_Z_cutoff(N_cap_dual=NCapDual)

cap = my_nanotube.cap
outfilename = "n_{}_m_{}_l_{}_cap_{}_dual_lattice_init"
outfilename = outfilename.format(n,m,l,cap.dual_lattice.npoints)
output.write_xyz(outfilename,my_nanotube.dual_lattice)


Dminimiser = minimisation.DualLatticeMinimiser(FFID=dual_force_field,
                                               structure = my_nanotube)
Dminimiser.minimise(my_nanotube.dual_lattice,
                    min_type=optimiser,
                    ftol=dual_lattice_mintol,
                    min_steps=dual_lattice_minsteps)

my_nanotube.update_caps()
outfilename = "n_{}_m_{}_l_{}_cap_{}_dual_lattice"
outfilename = outfilename.format(n,m,l,cap.dual_lattice.npoints)
output.write_xyz(outfilename,my_nanotube.dual_lattice)

my_nanotube.construct_carbon_lattice()

Cminimiser = minimisation.CarbonLatticeMinimiser(FFID=carbon_force_field,
                                                 structure = my_nanotube)

Cminimiser.minimise_scale(my_nanotube.carbon_lattice)
Cminimiser.minimise(my_nanotube.carbon_lattice,
                    min_type=optimiser,
                    ftol=carbon_lattice_mintol,
                    min_steps=carbon_lattice_minsteps)

outfilename = "n_{}_m_{}_l_{}_cap_{}_carbon_atoms"
outfilename = outfilename.format(n,m,l,cap.dual_lattice.npoints)
```

```
output.write_xyz(outfilename,my_nanotube.carbon_lattice)
outfilename = "n_{}_m_{}_l_{}_cap_{}_carbon_atoms_constrained"
outfilename = outfilename.format(n,m,l,cap.dual_lattice.npoints)
output.write_xyz(outfilename,my_nanotube.carbon_lattice,constrained=True)
```

10.3.2 Constructing Multiple Capped Nanotubes

nanocap_bulk_capped_nanotubes.py is an example script to perform a structure search to construct and save multiple capped nanotube. The code is shown below.

```
'''
-=-=-=-=-=-=-=-=-=-=-=-=-=NanoCap=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Copyright Marc Robinson  2014
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

A script to construct a series of capped
nanotubes of the same chirality

Input:
    n,m = Chirality (n,m)
    l = length
    cap_estimate = estimate cap from
                   tube density
    dual_lattice_force_field = force field
                               for dual lattice
    carbon_force_field = force field
                         for carbon lattice
    dual_lattice_mintol= energy tolerance for
                         dual lattice optimisation
    dual_lattice_minsteps= steps for dual lattice
                           optimisation
    carbon_lattice_mintol=as above for carbon lattice
    carbon_lattice_minsteps=as above for carbon lattice
    optimiser=optimsation algorithm
    seed = seed for initial cap generation
    N_nanotubes = required number of structures
    N_max_structures = maximum number of possible
                       structures to search through
    basin_climb = True/False - climb out of
                  minima
    calc_rings = True/False - calculate rings for
                  each structure

Output:
    -A structure log in myStructures.out

    -xyz files containing the carbon lattices

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
'''

import sys,os,random,numpy
sys.path.append(os.path.abspath((__file__+"/../../")))
from nanocap.core.minimisation import DualLatticeMinimiser, \
                                      CarbonLatticeMinimiser
from nanocap.core.minimasearch import MinimaSearch
from nanocap.structures.cappednanotube import CappedNanotube
```

```
from nanocap.core.output import write_points

n,m = 10,10
l = 20.0
cap_estimate = True

N_nanotubes = 5
N_max_structures = 20
basin_climb = True
calc_rings = True

dual_lattice_minimiser = "Thomson"
carbon_lattice_minimiser = "EDIP"
dual_lattice_mintol=1e-10
dual_lattice_minsteps=100
carbon_lattice_mintol=1e-10
carbon_lattice_minsteps=100
optimiser="LBFGS"
seed = 12345

my_nanotube = CappedNanotube()
my_nanotube.setup_nanotube(n,m,l=l)

if(cap_estimate):
    N_cap_dual = my_nanotube.get_cap_dual_lattice_estimate(n,m)

my_nanotube.construct_dual_lattice(N_cap_dual=N_cap_dual,seed=seed)
my_nanotube.set_Z_cutoff(N_cap_dual=N_cap_dual)

Dminimiser = DualLatticeMinimiser(FFID=dual_lattice_minimiser,
                                  structure = my_nanotube,
                                  min_type= optimiser,
                                  ftol = dual_lattice_mintol,
                                  min_steps = dual_lattice_minsteps)

Cminimiser = CarbonLatticeMinimiser(FFID=carbon_lattice_minimiser,
                                  structure = my_nanotube,
                                  min_type= optimiser,
                                  ftol = carbon_lattice_mintol,
                                  min_steps = carbon_lattice_minsteps)

Searcher = MinimaSearch(Dminimiser,
                        carbon_lattice_minimiser= Cminimiser,
                        basin_climb=basin_climb,
                        calc_rings=calc_rings)

Searcher.start_search(my_nanotube.dual_lattice,
                      N_nanotubes,
                      N_max_structures)

Searcher.structure_log.write_log(os.getcwd(),"myStructures.out")

for i,structure in enumerate(Searcher.structure_log.structures):
    carbon_lattice = structure.carbon_lattice
    filename = "C{}_carbon_atoms_{}".format(carbon_lattice.npoints,i)
    write_points(filename,carbon_lattice,format="xyz")
```

10.4 Database Operations

As `NanoCap` uses an sqlite database, the database can be browsed from the command line:

10.4.1 Saving structures to the local database

`nanocap_add_structure_to_db.py` is an example script to save a fullerene structure to disk. The code is shown below.

```
'''
-=-=-=-=-=-=-=-=-=-=-=-=-=-=NanoCap=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Copyright Marc Robinson  2014
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

A script to construct and add a fullerene to the local database

Input:
    N_carbon = Number of carbon atoms
    dual_lattice_force_field = force field
                               for dual lattice
    carbon_force_field = force field
                         for carbon lattice
    dual_lattice_mintol= energy tolerance for
                         dual lattice optimisation
    dual_lattice_minsteps= steps for dual lattice
                           optimisation
    carbon_lattice_mintol=as above for carbon lattice
    carbon_lattice_minsteps=as above for carbon lattice
    optimiser=optimsation algorithm
    seed = seed for initial cap generation

Output:
    structure is added to local database

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
'''

import sys,os,random,numpy
from nanocap.core.minimisation import DualLatticeMinimiser, \
                                      CarbonLatticeMinimiser
from nanocap.structures.fullerene import Fullerene
from nanocap.db.database import Database
from nanocap.core.output import write_xyz

N_carbon = 200

dual_force_field = "Thomson"
carbon_force_field = "EDIP"
dual_lattice_mintol=1e-10
dual_lattice_minsteps=100
carbon_lattice_mintol=1e-10
carbon_lattice_minsteps=100
optimiser="LBFGS"
seed = 12345

my_fullerene = Fullerene()
my_fullerene.construct_dual_lattice(N_carbon=N_carbon,seed=seed)
```

```
my_fullerene.set_fix_pole(False)
my_fullerene.set_nfixed_to_equator(0)


Dminimiser = DualLatticeMinimiser(FFID=dual_force_field,
                                  structure = my_fullerene)
Dminimiser.minimise(my_fullerene.dual_lattice,
                    min_type=optimiser,
                    ftol=dual_lattice_mintol,
                    min_steps=dual_lattice_minsteps)

outfilename = "C{}_dual_lattice.xyz".format(N_carbon)
write_xyz(outfilename,my_fullerene.dual_lattice)

my_fullerene.construct_carbon_lattice()

Cminimiser = CarbonLatticeMinimiser(FFID=carbon_force_field,
                                    structure = my_fullerene)

Cminimiser.minimise_scale(my_fullerene.carbon_lattice)
Cminimiser.minimise(my_fullerene.carbon_lattice,
                    min_type=optimiser,
                    ftol=carbon_lattice_mintol,
                    min_steps=carbon_lattice_minsteps)


my_db = Database()
my_db.init()
my_db.add_structure(my_fullerene,
                    add_dual_lattice=True,
                    add_carbon_lattice=True)
```

10.4.2 Loading structures from the local database

`nanocap_load_from_db.py` is an example script to load a set of fullerene dual lattices and save to disk. The code is shown below.

```
'''
-=-=-=-=-=-=-=-=-=-=-=-=-=NanoCap=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Created: May 23, 2014
Copyright Marc Robinson  2014
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

Example script showing how to load from a database.
Simple example showing how to load fullerene dual lattice

Input:
    type - the type of structure to return the dual
           for = "Fullerene","Capped Nanotube","Nanotube"

Output:
    folders for each structure containing xyz files and
    info files.

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
'''
import sys,os
```

```
sys.path.append(os.path.abspath(os.path.dirname(__file__)+"/../"))
from nanocap.db.database import Database
from nanocap.core.output import write_xyz

type = "Fullerene"

my_db = Database()
my_db.init()

#let's query for all fullerene dual lattices

tables = ["dual_lattices"]
selects = ["id",]
checks  = { "dual_lattices" : ['type',] }
data = {"dual_lattices" : {"type" : type}}

sql,data = my_db.construct_query(data,tables,selects,checks)
results  = my_db.query(sql,data)
#out now contains dual lattice IDs of fullerenes
print results

for result in results:
    id = result[0]
    structure = my_db.construct_structure(id)
    Nd = structure.dual_lattice.npoints
    folder = "Fullerene_dual_lattice_id_{}_N_{}".format(id,Nd)
    structure.export(  folder=".",
                       save_info=True,
                       save_image=False,
                       save_video=False,
                       save_carbon_lattice=False,
                       save_con_carbon_lattice=False,
                       info_file='structure_info.txt',
                       save_dual_lattice=True,
                       formats=['xyz',])
```