BitQT

Release 0.0.1

Roy Gonzalez-Aleman

Feb 15, 2021

CONTENTS

1 Why BitQT ? **3**
- 1.1 How Does BitQT Works ? . . . 3
- 1.2 Performance Benchmark . . . 6
- 1.3 Useful Alternatives . . . 7

2 Installation **9**
- 2.1 **MDTraj** . . . 9
- 2.2 **BitQT** . . . 9
- 2.3 **VMD** and **VMD clustering plugin** (optional) . . . 10

3 Technical Reference **11**
- 3.1 Basic help . . . 11
- 3.2 Arguments in Details . . . 12
- 3.3 Syntax Selection . . . 12

4 Case Study Tutorials **15**
- 4.1 Clustering a MD . . . 15
- 4.2 Visualizing Clusters in VMD . . . 15

5 Citation **19**

6 Version 0.0.1 **21**

7 Frequently Asked Questions **23**

BitQT is a Python command-line interface (CLI) conceived to speed up the Heyer's Quality Threshold (QT)[1] clustering of long Molecular Dynamics. The package implements a heuristic approach to this exact variant of QT.

The construction of a binary-encoded RMSD matrix, instead of the classical (half/single/double)-precision float matrix, led to considerable RAM savings compared to the few existing QT implementations. This binary matrix also allows implementing the significant steps as bitwise operations, which are faster than the corresponding set operations when dealing with considerable amounts of data.

[1] Heyer, L. J.; Kruglyak, S.; Yooseph, S. Exploring Expression Data Identification and Analysis of Coexpressed Genes. Genome Res. 1999, 9 (11), 1106–1115.

CHAPTER

ONE

WHY BITQT ?

There are plenty of clustering algorithms for analyzing MD out there, so why QT over those others? Well, not all kind of algorithms suit all problemic situations with the same performance. In those particular cases where strongly geometrically correlated conformations are needed to be returned as clusters, QT stands out as an ideal option because that guarantees that no pair of frames having a similarity value greater than a user-specified cutoff will coalesce into the same cluster.

This sounds great, but in practice, exact (and even approximate) implementations of QT are computationally expensive. That's where BitQT enters the scne. This heuristic makes a parallel with the Maximum Clique Problem (MCP) and treats QT problem as a search of cliques in a mathemathical graph. As it is possible to conduct this search using vector of bits, BitQT is a fast and memory efficient alternative to the few other current implementations.

1.1 How Does BitQT Works ?

Here we expose briefly the basis of BitQT assumptions and the key aspects of the algorithm. You can refer to the academic publication for more details.

1.1.1 Original QT algorithm

The application of the original QT to an MD trajectory can be described as follows: After the selection of a similarity threshold **k**, one arbitrary frame is selected and marked as a candidate cluster **C1**. The remaining frames are iteratively added to **C1** if and only if two conditions hold;

- *Condition 1:* the similarity distance between the entering frame and every frame inside **C1** is the minimum possible, and
- *Condition 2:*- the similarity distance between the entering frame and every frame already inside **C1** does not exceed the threshold **k**.

This process continues for all frames **n** in the trajectory until **Cn** candidate clusters have been formed. The one with more frames is set as a cluster, its elements removed from further consideration, and the entire process repeated until no more clusters can be discovered.

1.1.2 Parallel with the Maximum Clique Problem

The most importat aspect of the original algorithm is its guarantee that all pairwise similarities inside a cluster will remain under the threshold **k**. This aspect is assured by *Condition 2*. *Condition 1* merely limits the size of retrieved clusters but has no impact in maintaining the similarity threshold.

From Graph Theory, we know that a *clique* is a subgraph in which vertices are all pairwise adjacent. If a clique is not contained in any other clique, it is said to be *maximal*, while the term *maximum clique* denotes the maximal clique with a maximum number of nodes. The maximum clique problem (MCP) deals with the challenge of finding the maximum clique inside a given graph.

To make a parallel between QT and the MCP, we represent each frame of an MD trajectory as a node of an undirected graph in which edges depict RMSD similarity between nodes. Only edges with an RMSD less or equal than the threshold **k** are allowed. In that context, QT can be declared as an iterative search of cliques. QT cliques, however, are not necessarily maximum due to *Condition 1* of the algorithm, which ensures that they should have a minimum weight instead of a maximum cardinality.

Conveniently, a redefinition of the QT algorithm can be made to search for maximum-sized clusters instead of minimum-weighted without compromising the pairwise similarity assured by the *Condition 2*. Relaxation of *Condition 1* in this way, automatically converts QT in an MCP problem, accessible by the graph theory tools.

This approach has a profound impact on how molecular similarity can be encoded and in the efficiency of algorithms that can be used to solve the problem, as discussed in the next sections.

1.1.3 BitQT Algorithm

1. RMSD-encoded Binary Matrix

If we conceive the QT algorithm as an MCP problem, after relaxation of *Condition 1* our search will be focused on finding cliques of maximum cardinality, and no useful information is extracted from the weight of the edges other than its absence or existence. This information can therefore be encoded as a binary matrix **M** where **M_ij=1** if nodes **i** and **j** are similar (*RMSD_ij* <= *k*) or 0 otherwise.

Besides the RAM saving, expressing similarity as a binary matrix offers the possibility to perform the search of cliques using binary operators (AND and XOR), contributing to the speedup of the heuristic we propose in the following sections.

2. Nodes coloring

Each vertex of the input graph (Graph 1, Figure 1) is ranked (column R, Matrix 1, Figure 1) in descending order of their corresponding degrees (column D, Matrix 1, Figure 1). Following the rank order, each vertex takes a color label that it shares with all other vertices that are neither colored nor neighbors (column C, Matrix 1, Figure 1).

3. Clique search from the maximum degree node

After all vertices are colored, the search of a clique starts considering only neighbors of the maximum degree node of the graph (Graph 1A, Figure 1), which is called the *seed* of the clique (node 1 in Matrix 1A, Graph 1A, Figure 1). Neighbors of the seed are strictly ordered for further processing following three criteria (DCg ordering); descending order of their degrees, ascending order of their color class, and ascending order of the degeneracy of the color class (columns D, C, and g, respectively, Matrix 1A, Figure 1). Following this ordering, the first node is selected to start a clique, and subsequent nodes will be added to that clique if they have a still-not-explored color and if they are adjacent to previously explored nodes (clique propagation).

BitQT performs this search using bitwise operations. The bit-vector Bi corresponding to the maximum degree node is set as the clique bit-vector (B1 in Heuristic search of Graph 1A, Figure 1). Following the DCg ordering, an AND operation is performed between the clique bit-vector and the next node bit-vector if it has a new color (B6 in Heuristic search of Graph 1A, Figure 1). Indices corresponding to bits that become zero by this operation are discarded from further consideration (B2, B3, B4, and B5) as they are not adjacent to processed nodes (B1 and B6). The resulting

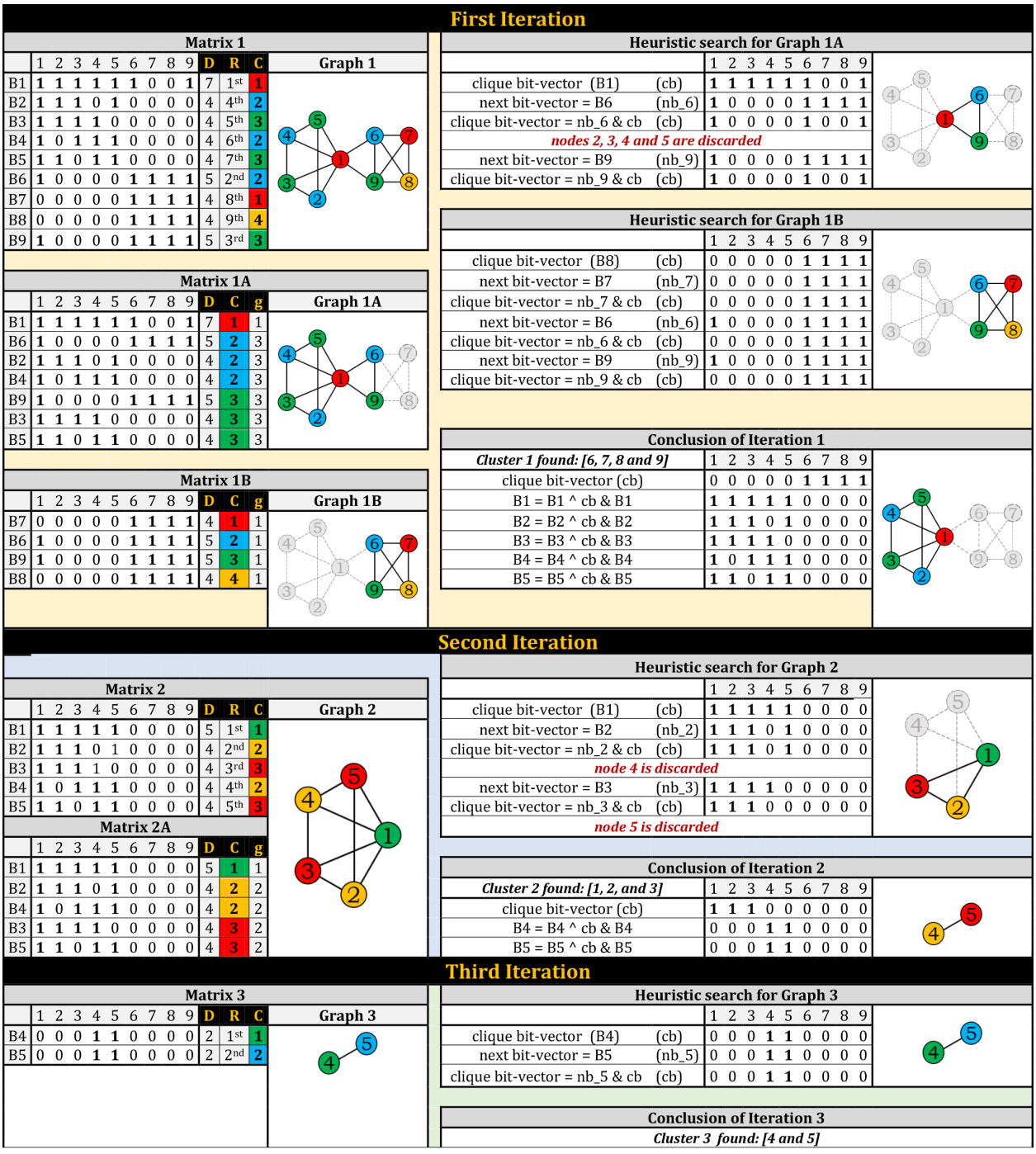

Fig. 1: Figure 1: Workflow diagram of BitQT algorithm

bit-vector becomes the new clique bit-vector used for the AND operation with the next candidate following the DCg ordering (B9). The bit-vector resulting from the iterative AND operations contains the members of the first clique.

4. Clique search from promising nodes

Once the clique retrieved by using the maximum degree node as the seed is found in the previous step, the same exploration strategy is conducted for every emph{promising node} in the original graph (Graph 1). A promising node (B8 in Graph 1, Figure 1) is defined as a node with a color not present in the first clique and whose degree is higher than the number of nodes in the first clique. Using such nodes as seeds for propagation might lead to the formation of a bigger clique (Heuristic search of Graph 1B, Scheme 1).

5. Conclusion and updating

When the maximum degree node and all promising nodes have been used as seeds, the maximum clique found is picked as a cluster, and their nodes removed from the input graph (the corresponding Bi vectors removed from the binary matrix). An updating of the remaining bit-vector is necessary to set as zero all entries corresponding to nodes that formed the cluster, which will not be available for subsequent iterations. This updating is bitwise encoded as a consecutive AND/XOR operation between remaining bit-vectors and the clique bit-vector (Conclusion of iteration 1, Figure 1). The same steps are repeated from Step 3 until no more cliques can be found.

1.2 Performance Benchmark

The two QT implementations used for comparisons correspond to the QT code (QTPy), and the qtcluster command distributed in version 6.0.1 of the ORAC package.

MD trajectories of different sizes and compositions were selected: **6K**- a 6001 frames REMD simulation of the Tau peptide, **30K**- a 30605 frames MD of villin headpiece based on PDB 2RJY, **50K**- a 50500 frames MD of serotype 18C of Streptococcus Pneumoniae, **100K**- a 100500 frames MD of Cyclophilin A based on PDB 2N0T, and **250K**- a 250000 frames MD of four chains of the Tau peptide that corresponds to the MD simulation of an extended Tau peptide (PHF8) during 1 microsecond.

		BitQT		**qtcluster**		**QTPy**	
Trajectory	**No. of atoms**	Run time	RAM peak	Run time	RAM peak	Run time	RAM peak
	(selection)	*h:mm:ss*	*GB*	*h:mm:ss*	*GB*	*h:mm:ss*	*GB*
6K	217 (all)	0:00:08	0.101	0:08:21	0.529	0:04:36	0.181
30K	64 (CA)	0:02:15	0.470	0:18:55	0.270	3:41:11	2.710
50K	78 (no H)	0:12:34	0.435	1:14:08	1.526	181:51:57	7.101
100K	660 (backbone)	1:15:37	4.355	***0:00:49***	***>81.014***	***>200:00:00***	***18.626***
250K	160 (backbone)	6:36:04	8.128	130:18:06	17.476	***0:00:03***	***>117.000***

[1] *Bold entries denote either a time crash (job taking more than 200 h) or a memory crash (job taking more than 64GB)*

Fig. 2: Figure 2: Performance benchmark of BitQT vs QTPy vs qtcluster.

All calculations were performed on an AMD Ryzen5 Hexa-core Workstation with a processor speed of 3.6 GHz and 64GB RAM under a 64-bit Xubuntu 18.04 operating system. Run times and RAM peaks were recorded with the emph{/usr/bin/time} Linux command.

For more details, please refer to the supporting information of the academic publication.

1.3 Useful Alternatives

As we have described, BitQT is an heuristic approach that can be used as a replacement for the very time-consuming exact variants of Quality Threshold clustering of Molecular Dynamics. However, there exist other cheaper, popular, useful alternatives for geometrical clustering that might equally fit your needs. Here you go ...

QTPy is an exact implementation of the original Quality Thresold for Molecular Dynamics. Technically, this ones is not cheaper, but you might want to consider it for benchmark purposes. Implemented using an RMSD square matrix of floats (half-precision).

BitClust is an exact implementation of a very popular clustering algorithm. You may have heard of it as daura, qt-like, qt, neighbor-based or gromos. It has been implemented in VMD, GROMACS, WORDOM, PyPROCT and others. BitClust is implemented using an RMSD-encoded square matrix of bits.

RCDPeaks is, to the best of our knwoledge, the first exact implementation of Density Peaks clustering that does not need a square matrix of floats. Instead, it uses a dual-heap approach so it is very lightweight and faster than other alternatives.

MDSCAN is an alternative to HDBSCAN that uses RMSD as metric but does not need a square matrix to work as it was implemented using an efficient dual-heap approach. HDBSCAN is perhaps on of the most robust clustering algorithms out there. It has been succesfully apllied to Molecular Dynamics. However, most implementations do not include RMSD as similarity metric. The workaround for those alternatives is to accept a precomputed square float matrix that is too costly when dealing with long trajectories.

CHAPTER

TWO

INSTALLATION

There are some easy-to-install dependencies you must have before running BitQT. MDTraj (mandatory) will perform the heavy RMSD calculations, while VMD (optional) will help with visualization tasks. The rest of the dependencies will be automatically managed by BitQT.

2.1 MDTraj

It is recommended that you install `mdtraj` using conda.

```
$ conda install -c conda-forge mdtraj
```

You can install `mdtraj` with `pip`, if you prefer.

```
$ pip install mdtraj
```

2.2 BitQT

Via **pip**:

After successfully installing `mdtraj` you can easily install BitQT and the rest of its critical dependencies using pip.

```
$ pip install bitqt
```

Via **GitHub**:

$ git clone https://github.com/LQCT/bitqt $ cd bitqt $ python setup.py install

Then, you should be able to see BitQT help by typing in a console:

```
$ bitqt -h
```

2.3 VMD and VMD clustering plugin (optional)

BitQT clusters can be visualized by loading a *.log* file in VMD via a clustering plugin. This is described in section *Case Study Tutorials*.

Official site for VMD download and installation can be found here.

Instructions on how to install the clustering plugin of VMD are available here.

CHAPTER

THREE

TECHNICAL REFERENCE

3.1 Basic help

BitQT help is displayed in the console when typing **bitqt -h**

```
$ bitclust -h

usage: bitqt -traj trajectory [options]

BitQT: A Graph-based Approach to the Quality Threshold Clustering of Molecular
Dynamics

optional arguments:
  -h, --help           show this help message and exit

Trajectory options:
  -traj trajectory     Path to trajectory file [required]
  -top topology        Path to the topology file
  -first first_frame   First frame to analyze (counting from 0) [default: 0]
  -last last_frame     Last frame to analyze (counting from 0) [default: last
                       frame]
  -stride stride       Stride of frames to analyze [default: 1]
  -sel selection       Atom selection (MDTraj syntax) [default: all]

Clustering options:
  -cutoff k            RMSD cutoff [default: 2]
  -min_clust_size m    Minimum size of returned clusters [default: 2]
  -nclust n            Number of clusters to retrieve [default: 2]

Output options:
  -odir bitQT_outputs  Output directory to store analysis [default:
                       bitQT_outputs]

As simple as that ;)
```

3.2 Arguments in Details

`-traj (str)`: This is the only argument that is **always** required. Valid extensions for trajectories are `.dcd`, `.dtr`, `.hdf5`, `.xyz`, `.binpos`, `.netcdf`, `.prmtop`, `.lh5`, `.pdb`, `.trr`, `.xtc`, `.xml`, `.arc`, `.lammpstrj` and `.hoomdxml`.

`-top (str)`: If trajectory format (automatically inferred from file extension) includes topological information this argument is not required. Otherwise, user must pass a path to a topology file. Valid topology extensions are `.pdb`, `.pdb.gz`, `.h5`, `.lh5`, `.prmtop`, `.parm7`, `.prm7`, `.psf`, `.mol2`, `.hoomdxml`, `.gro`, `.arc` and `.hdf5`.

`-first (int, default=0)`: The first frame to analyze (starting the count from 0)

`-last (int, default=last)`: Last frame to analyze (starting the count from 0). The last frame is internally detected.

`-stride (int, default=1)`: Stride of frames to analyze. You might want to use this argument for reducing the trajectory size when performing exploratory analysis.

`-sel (str, default='all')`: Atom selection. BitQT inherits `MDtraj` very flexible syntax selection. Common cases are listed at the *Syntax Selection* section.

`-cutoff (int, default=2)`: RMSD cutoff for similarity measures given in Angstroms (1 A = 0.1 nm).

`-min_clust_size (int, default=2)`: Minimum number of frames inside returned clusters. 0 is not a meaningful value, and 1 implies an unclustered frame (no other frame is similar). Greater values of this parameter may speed up the algorithm with loss of uniformity in retrieved clusters.

`-nclust (int, default=all)`: The maximum number of calculated clusters. Change the default for a better performance whenever you only need to inspect the first clusters.

`-ref (int, default=0)`: Reference frame to align trajectory.

`-odir (str, default="./bitQT_outputs")`: Output directory to store analysis. BitQT checks for outdir existence to avoid overwriting it.

3.3 Syntax Selection

BitQT inherits atom selection syntax from **MDTraj**, which is similar to that in VMD. We reproduce below some of the **MDTraj** examples. Note that in BitQT, all keywords (or their synonyms) string are passed directly to `-sel` argument. For more details on possible syntax, please refer to MDTraj original documentation.

BitQT recognizes the following keywords.

Keyword	Synonyms	Type	Description
`all`	`everything`	`bool`	Matches everything
`none`	`nothing`	`bool`	Matches nothing
`backbone`	`is_backbone`	`bool`	Whether atom is in the backbone of a protein residue
`sidechain`	`is_sidechain`	`bool`	Whether atom is in the sidechain of a protein residue
`protein`	`is_protein`	`bool`	Whether atom is part of a protein residue
`water`	`is_water`, `waters`	`bool`	Whether atom is part of a water residue
`name`		`str`	Atom name
`index`		`int`	Atom index (0-based)
`type`	`element`, `symbol`	`str`	1 or 2-letter chemical symbols from the periodic table
`mass`		`float`	Element atomic mass (daltons)
`residue`	`resSeq`	`int`	Residue Sequence record (generally 1-based, but depends on topology)
`resid`	`resi`	`int`	Residue index (0-based)
`resname`	`resn`	`str`	Residue name
`rescode`	`code`, `resc` `	`str`	1-letter residue code
`chainid`		`int`	Chain index (0-based)

3.3.1 Operators

Standard boolean operations (`and`, `or`, and `not`) as well as their C-style aliases (`&&`, `||`, `!`) are supported. The expected logical operators (<, <=, ==, !=, >=, >) are also available, as along with their FORTRAN-style synonyms (`lt`, `le`, `eq`, `ne`, `ge`, `gt`).

3.3.2 Range queries

Range queries are also supported. The range condition is an expression of the form `<expression> <low> to <high>`, which resolves to `<low> <= <expression> <= <high>`. For example

```
# The following queries are equivalent
-sel "resid 10 to 30"
-sel "(10 <= resid) and (resid <= 30)"
```

CHAPTER

FOUR

CASE STUDY TUTORIALS

Here you can find a simple example on how to run a clustering job using BitQT as well as how to visualize the clusters using a VMD plugin.

4.1 Clustering a MD

Note: We included an example folder where you can find the topology and trajectory files we have used in this section.

As we already mentioned, the only required argument for BitQT is the trajectory file. We will use the binary dcd file *aligned_original_tau_6K.dcd*. As dcd format does not contain any topological information, it is necessarty to pass the **-top** argument with an appropiate topology file. In this case, we will be using the PDB formatted file *aligned_tau.pdb*.

Then you can run

```
$ bitqt -top examples/aligned_tau.pdb -traj examples/aligned_original_tau_6K.dcd -sel␣
↪all -cutoff 4 -odir 6K_4
```

After succesful termination, BitQT will produce some output files to the specified folder 6K_4:

- A *cluster_statistics.txt* file containing clusterID, cluster_size, and its percentage from the total frames analyzed
- A *frames_statistics.txt* file containing every frameID and its clusterID.
- A *file.log* to visualize all the clusters via VMD plugin as discussed in the next section.

4.2 Visualizing Clusters in VMD

BitQT produces a *file.log* that contains cluster frames in the NMRcluster format. This can be visualized in VMD using the clustering plugin (see *Installation*).

Figure 1 shows the main window of the plugin and the steps you should follow:

1. Selection section: Here you can define the selection of atoms that you would like to visualize.
2. Import section: After loading in VMD the topology and trajectory files that you used to run BitQT, go to the Import button of the plugin, select *NMRcluster* option and navigate to the *file.log* produced by BitQT.
3. Results section: Here you can select which clusters to visualize. Note that through the standard VMD commands, you can change the representations and customize the visualization as you want.

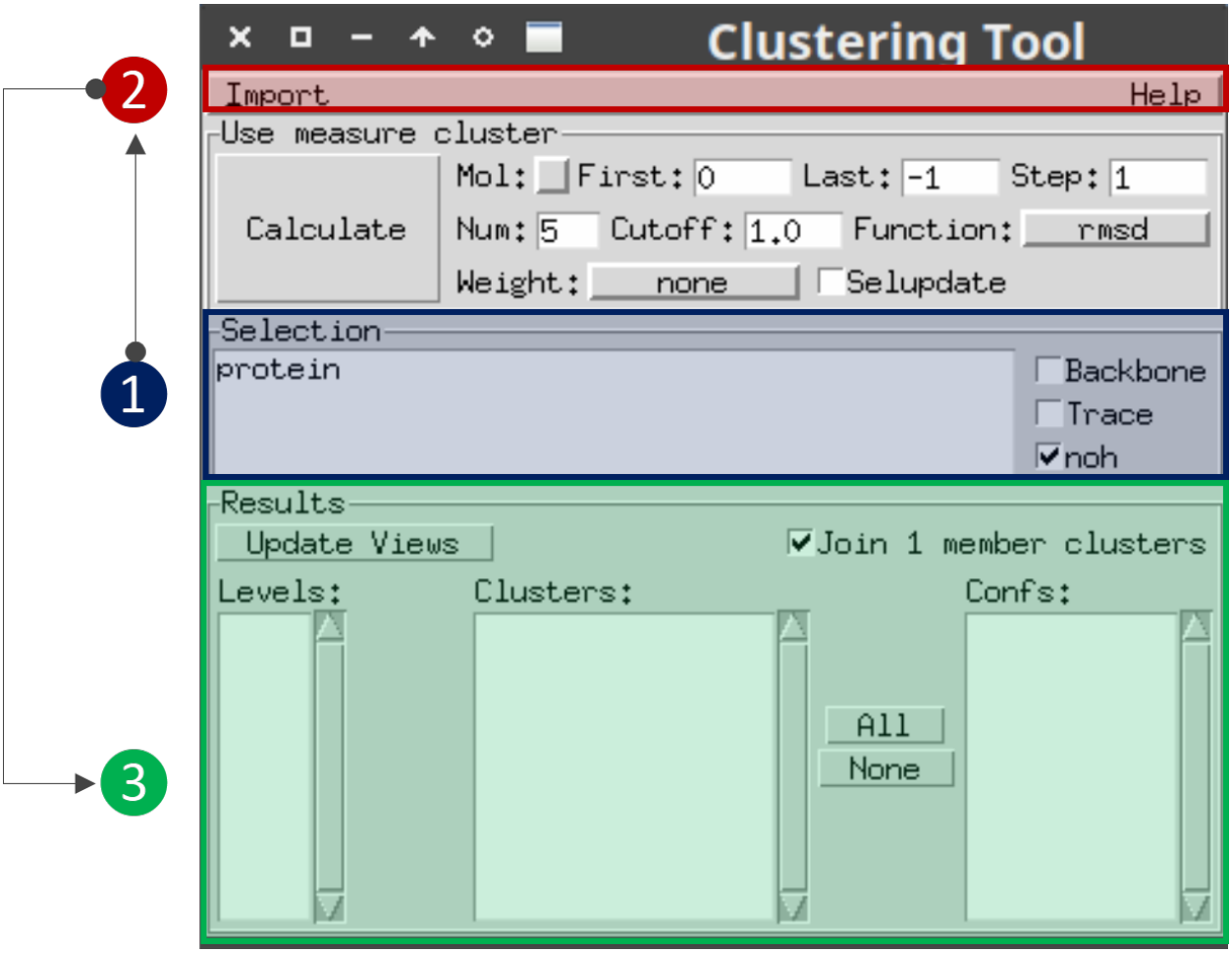

Fig. 1: Figure 1: Main window of the VMD clustering plugin

Do not change any of the parameters from the *Use measure cluster* section. As it indicates, these are for triggering the internal *measure cluster* command of VMD that does not implement QT.

Figure 2 shows a loaded example. Note that only the backbone of clusters 1 and 4 have been selected.

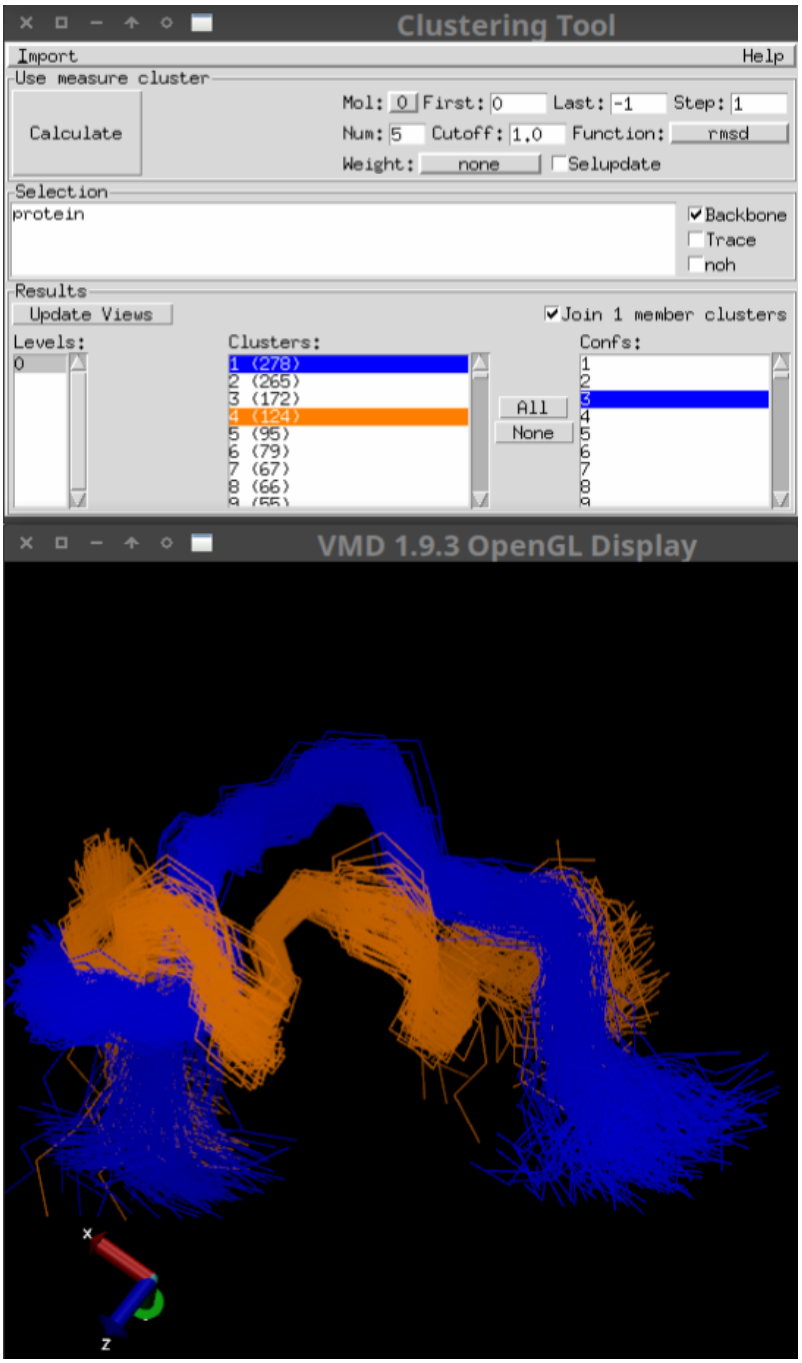

Fig. 2: Figure 2: Visualization example

CHAPTER

FIVE

CITATION

Note: This section will be completed upon academic publication.

CHAPTER

SIX

VERSION 0.0.1

First version. It corresponds to what is described in the academic publication of BitQT.

CHAPTER

SEVEN

FREQUENTLY ASKED QUESTIONS

Note: This section will be completed upon academic publication.