BitQT: a graph-based approach to the quality threshold clustering of molecular dynamics

Roy González-Alemán, Daniel Platero-Rochart, David Hernández-Castillo, Erix Hernández-Rodríguez, Julio Caballero, Fabrice Leclerc, Luis Montero-Cabrera

Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Original Paper

OXFORD

Structural Bioinformatics

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

Roy González-Alemán [1, 2,*], Daniel Platero-Rochart [1], David Hernández-Castillo [3], Erix W. Hernández-Rodríguez [5], Julio Caballero [4], Fabrice Leclerc [2,*] and Luis Montero-Cabrera [1]

[1]Laboratorio de Química Computacional y Teórica (LQCT), Facultad de Química, Universidad de La Habana, La Habana, 10400, Cuba
[2]Institute for Integrative Biology of the Cell (I2BC), CEA, CNRS, Université Paris Saclay, Gif-sur-Yvette, F-91198, France
[3]Institute of Theoretical Chemistry, University of Vienna, Währinger Str. 17, 1090 Vienna, Austria
[4]Departamento de Bioinformática, Facultad de Ingeniería, Centro de Bioinformática, Simulación y Modelado (CBSM), Universidad de Talca, Talca, Chile, and
[5]Laboratorio de Bioinformática y Química Computacional, Escuela de Química y Farmacia, Facultad de Medicina, Universidad Católica del Maule, 3460000 Talca, Chile.

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Classical Molecular Dynamics is a standard computational approach to model time-dependent processes at the atomic level. The inherent sparsity of increasingly huge generated trajectories demands clustering algorithms to reduce other post-simulation analysis complexity. The quality threshold (QT) variant is an appealing one from the vast number of available clustering methods. It guarantees that all members of a particular cluster will maintain a collective similarity established by a user-defined threshold. Unfortunately, its high computational cost for processing big data limits its application in the molecular simulation field.

Results: In the present work, we propose a methodological parallel between QT clustering and another well-known algorithm in the field of Graph Theory, the Maximum Clique Problem. Molecular trajectories are represented as graphs whose nodes designate conformations, while unweighted edges indicate mutual similarity between nodes. The use of a binary-encoded RMSD matrix coupled to the exploitation of bitwise operations to extract clusters significantly contributes to reaching a very affordable algorithm compared to the few implementations of QT for Molecular Dynamics available in the literature. Our alternative provides results in good agreement with the exact one while strictly preserving the collective similarity of clusters.The source code and documentation of BitQT are free and publicly available on GitHub (https://github.com/LQCT/BitQT.git) and ReadTheDocs (https://bitqt.readthedocs.io/en/latest/) respectively.

Contact: roy_gonzalez@fq.uh.cu, fabrice.leclerc@i2bc.paris-saclay.fr

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Molecular dynamics (MD) is a powerful tool to gain insight into the conformational behavior of nanoscopic systems. Nowadays, methodologies like coarse-grained MD, accelerated MD, and replica-exchange MD are common ways to reach a representative sampling of dynamically meaningful states. As the computational power grows, the size of trajectories generated by these techniques represents a massive amount of information that is potentially difficult to analyze. Geometrical clustering is a classical way to simplify those trajectories by grouping

similar conformations into sets known as clusters. In such a way, conformations inside a cluster are more similar between them than those from other clusters.

Many clustering algorithms exist for analyzing MD (Peng *et al.*, 2018), having benefits and shortcomings that make them suitable for particular applications and inappropriate for others (Röttger, 2016). Due to the inherent subjectivity associated with classification (the same set of elements can be grouped according to many different criteria), some authors consider clustering as an art (von Luxburg *et al.*, 2012). However, in those particular cases where strongly geometrically correlated conformations are needed to be returned as clusters, the Quality Threshold (QT) algorithm (Heyer *et al.*, 1999) stands out as an ideal option.

QT appeared in the context of clustering gene expression patterns. Since then, it has been applied to many areas other than microbiology (Tang *et al.*, 2010; Yaakob *et al.*, 2010; Olson *et al.*, 2011; Dutta and Overbye, 2011; Yaakob and Jain, 2012), including the MD field (Procacci *et al.*, 1997; Danalis *et al.*, 2012). Two remarkable features of this algorithm are the guarantee that no pair of frames having a similarity value greater than a user-specified cutoff will coalesce into the same cluster and that the number of clusters to retrieve must not be known *a priori*. However, QT has an expensive computational cost (Danalis *et al.*, 2012) that currently limits its applicability.

Several popular software have inaccurately qualified their clustering implementations as QT or QT-like variants in the past (González-Alemán *et al.*, 2020b). These pseudo-QT alternatives correspond to another simple and largely disseminated algorithm (Daura *et al.*, 1999) that has been recently optimized for the efficient treatment of long molecular trajectories (González-Alemán *et al.*, 2020a).

After careful inspection of current literature, we have found only two valid and ready-to-use attempts to implement QT to analyze MD trajectories. The first one corresponds to the *qtcluster* command of the ORAC suite (Procacci *et al.*, 1997) while the second one is an implementation previously published by authors of this study (González-Alemán *et al.*, 2020b), referred to as *QTPy* from now on.

While *QTPy* can be stated as an exact version of the QT proposed by Heyer in 1999, it should be emphasized that *qtcluster* only partially complies with the original algorithm. Perhaps the most essential feature that makes qtcluster a fast QT implementation lies in the fact that it is not an exact QT attempt, only preserving one condition from the exact algorithm; the one assuring the collective similarity of retrieved clusters. It is also worth noting that *qtcluster* uses the maximum difference between corresponding pairs of atoms as the similarity measure while *QTPy* employs the more customary optimal RMSD. Both of them are marked by a run time and RAM consumption that impedes the processing of relatively long trajectories.

Here we propose a heuristic variation of QT that can output equivalent results to the exact algorithm at a much less computational cost. It has been devised using a parallel with the Maximum Clique Problem (MCP). A clique is a fully connected sub-graph, i.e. all pairs of nodes in it are connected by an edge, so the MCP is concerned with searching for the biggest clique in a graph. In our workflow, molecular trajectories are represented as graphs in which each frame is depicted as a node. The similarity between frames is encoded as binary (unweighted) edges, and clusters are found following a heuristic search of big cliques.

The construction of a binary-encoded similarity matrix, instead of the classical half/single-precision float matrix, leads to considerable RAM savings regarding the existing QT implementations. This binary matrix also allows implementing the fundamental clustering steps as bitwise operations faster than the corresponding set operations when dealing with considerable amounts of data. Our proposal, *BitQT*, is free and publicly available at GitHub (https://github.com/LQCT/BitQT.git).

2 Computational Details

BitQT heuristic has been coded in Python 3 programming language and makes heavy use of two third-party libraries: version 1.9.4 of MDTraj (McGibbon *et al.*, 2015) for the fast RMSD calculations and version 1.6.1 of bitarray for all the binary-related operations (https://github.com/ilanschnell/bitarray). The two QT implementations used in this work for comparisons against *BitQT* correspond to the *QTPy* code, previously published by authors of this study and available at GitHub (https://github.com/rglez/QT) and the *qtcluster* command distributed in version 6.0.1 of the ORAC package (Procacci *et al.*, 1997).

We selected MD trajectories of different sizes and compositions to benchmark the performance of these algorithms. They are referred generically by their size as follows: 6K- a 6001 frames REMD simulation of the Tau peptide (Shea and Levine, 2016), 30K- a 30605 frames MD of villin headpiece based on PDB 2RJY (Melvin *et al.*, 2016), 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 1μs (Álvarez-Ginarte *et al.*, unpublished work). Not referenced trajectories were obtained by the authors of this work. The details of the MD are available in the Supporting Information (S2: Details of the Molecular Dynamics Simulations). All trajectory and topology files used in this work can be found online at the following addresses: 6K, 50K, 100K, and 250K at https://doi.org/10.6084/m9.figshare.c.5403930.v1, and 30K at https://doi.org/10.6084/m9.figshare.3983526.v1.

QTPy and *BitQT* used the same quality threshold value k for each trajectory; 4 Å for trajectories 6K and 30K, 3 Å for trajectory 250K and 2 Å for trajectories 50K and 100K. These values were set after a trial/error procedure aided by visual inspection of the generated clusters uniformity. However, as *qtcluster* does not use the RMSD metric (Steipe, 2002), we adjusted the k values for each trajectory ran with this software. We multiplied the corresponding k by 2.4, in analogy with a previously published report of *qtcluster*'s authors (see S.I of Guardiani *et al.* (2012)).

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

3 Approaching QT Clustering from a Maximum Clique Problem Perspective

If we define the *diameter of a cluster* C as the maximum distance between any pair of its elements (equation 1), the exact QT algorithm applied to an MD trajectory can be described as follows: After the user sets a similarity threshold k, one arbitrary frame is selected and marked as a candidate cluster C_1. The remaining frames are iteratively added to C_1 if and only if two conditions hold: *Condition 1*- the entering frame increases the diameter of C_1 by the minimum amount, and *Condition 2*- the diameter of C_1 does not exceed the threshold k. A second candidate cluster is formed by starting with another frame and repeating the procedure. Note that all frames are made available to the second candidate cluster (frames from the first candidate cluster are not discarded from consideration). This process continues for all frames n in the trajectory until C_n 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.

$$diam(C) = max(d_{ij}) \mid \forall\ (i,j)\ \in C \tag{1}$$

The crucial aspect of the above-described algorithm lies in its ability to guarantee that all pairwise similarities inside a cluster will remain under the threshold k. This aspect is assured entirely by *Condition 2*, whose relevance has been previously discussed (González-Alemán *et al.*, 2020b). It is worth noting that *Condition 1* merely limits the size of retrieved clusters but has no impact in maintaining their collective similarity.

Concepts and tools from graph theory have been widely used to represent numerous situations in which several objects are mutually related. Before showing how QT can be approached from a graph-theoretical perspective, we will briefly define some basic underlying concepts.

A *graph* $G = (V, E)$ is a pair of a set of *vertices* (*nodes*) V and a set of edges E. Each *edge* is a two-element subset of V and denotes the adjacency between the nodes it connects. Two connected nodes are called *neighbors*, and the number of neighbors of a given node constitutes its *degree*. Connectivity of simple graphs can be represented using its *adjacency matrix*, a square symmetric matrix M in which $M_{ij} = 1$ if nodes i and j are connected and $M_{ij} = 0$ otherwise. If there is no directionality in the definition of the edges and there is no data associated to them, it is said that the graph is *undirected* and *unweighted*.

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 (maximum *cardinality*). The maximum clique problem (MCP) solves the challenge of finding the maximum clique inside a given graph.

A central idea of MCP algorithms is the notion of *vertex coloring*. A *proper vertex coloring* refers to assigning a particular color (or any other unique label) to each vertex of a graph so that adjacent vertices do not share the same color. The *vertex coloring problem* consists of finding a proper coloring that uses the fewest number of colors, known as the graph's chromatic number (χ). It is common to employ coloring techniques because χ is an upper bound to the maximum clique's size of a graph. This property is exploited to discard impossible solutions and guide the search of cliques (San Segundo and Tapia, 2014). As exact coloring itself is an NP-hard problem, heuristics are usually applied.

To make a parallel between QT and MCP, it is possible to 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 to the threshold *k* are allowed, so there would be no weights associated with them. In that context, QT can be declared as an iterative search of cliques. However, QT cliques 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. *Condition 1* requires the diameter of the clusters to be minimum. Still, it is *Condition 2* that assures the respect of a quality threshold in the pairwise similarity of retrieved clusters.

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 second condition. In most clustering applications, maximizing the size of the clusters is a desirable feature. Relaxation of *Condition 1* in this way automatically converts QT in an MCP problem, accessible by the graph theory tools. This approach profoundly impacts how molecular similarity can be encoded and the efficiency of algorithms used to solve the problem, as discussed in the following sections.

3.1 Binary encoding of RMSD pairwise similarity

As the ultimate goal of our clustering proposal is to partition all MD trajectory frames, all the pairwise similarities should be analyzed. This information can be saved in RAM as a matrix to accelerate the algorithm's run time. As $RMSD_{ij} = RMSD_{ji}$, the similarity matrix is symmetric. Although the valuable information is contained in one of the triangles, many current MD clustering software preserve the whole matrix to avoid the performance penalty of working with "triangular" data structures.

The amount of RAM needed for the storage of the matrix expressed in GB. can be calculated using the equation 2, where N is the total number of frames in the trajectory and m is the size of the numeric type used to express the RMSD values (in bytes). Being the RMSD a float number ranging from 0.0 to infinite, the common choice is to use float numeric types to represent inter-frame similarity. Some clustering alternatives like TTClust (Tubiana *et al.*, 2018) use the costly choice of double-precision float (m=8). Other options like GROMACS (Abraham *et al.*, 2015), and WORDOM (Seeber *et al.*, 2007) packages use single-precision floats (m=4), saving half of RAM just by adjusting the precision used to express RMSD. It is worth noting that the minimum size of standard available floats is a half-precision value (m=2), which is enough for most MD clustering applications and the one used in *QTPy*.

$$V_{RAM} = \frac{m * N^2}{2^{30}} \tag{2}$$

Here we followed a different approach to diminish the value of m. If we conceive the QT algorithm as an MCP problem, after considering the relaxation of *Condition 1* our search will be focused on finding cliques of maximum cardinality, and no helpful 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 ($RMSD_{ij} \leq k$) or 0 otherwise. Note that M contains the same information that the adjacency matrix of the graph except for the diagonal, which in this case will always be one instead of zero ($RMSD_{ii} \equiv 0.0$). For the sake of simplicity, we will refer to M as the adjacency matrix of the trajectory graph.

By using the binary adjacency matrix, we reduce the RAM consumption of this object in 16, 32, or 64 times ($m = \frac{1}{8}$) compared to other software that deals with half, single or double-precision float values to represent the pairwise RMSD distance. 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 clique search algorithm we propose in the following section.

3.2 QT as a heuristic search of big cliques

Since MCP is an NP-hard problem, no efficient exact polynomial-time algorithms are expected to be found. Nevertheless, exact proposals exist to treat the MCP relatively fast for real problems of limited size. San Segundo and co-workers' efforts are of particular relevance for us as they also use the binary adjacency matrix of graphs and bitwise operations to develop their algorithms (San Segundo *et al.*, 2010; San Segundo *et al.*, 2013; San Segundo and Artieda, 2015; San Segundo *et al.*, 2016, 2017b,a). However, they are mainly focused on exact solutions rather than approachable heuristics.

A heuristic method tries to find a satisfactory solution to a complex problem using logical assumptions. While heuristics for MCP reduce the time of finding cliques, there is no guarantee that found cliques would be maximum. Nevertheless, heuristics are widely used in applications where a marginal error is not of great importance. In our case, we want to keep the common similarity of clusters, but their size is not of a big concern. After all, the original QT does not provide either maximum cliques. We are interested in a cheaper way to keep pairwise similarity, and for that purpose, a heuristic approach may suffice. Next we describe the workflow of the *BitQT* clustering algorithm, which is built upon a not previously published heuristic for searching big cliques (see the pseudocode in the

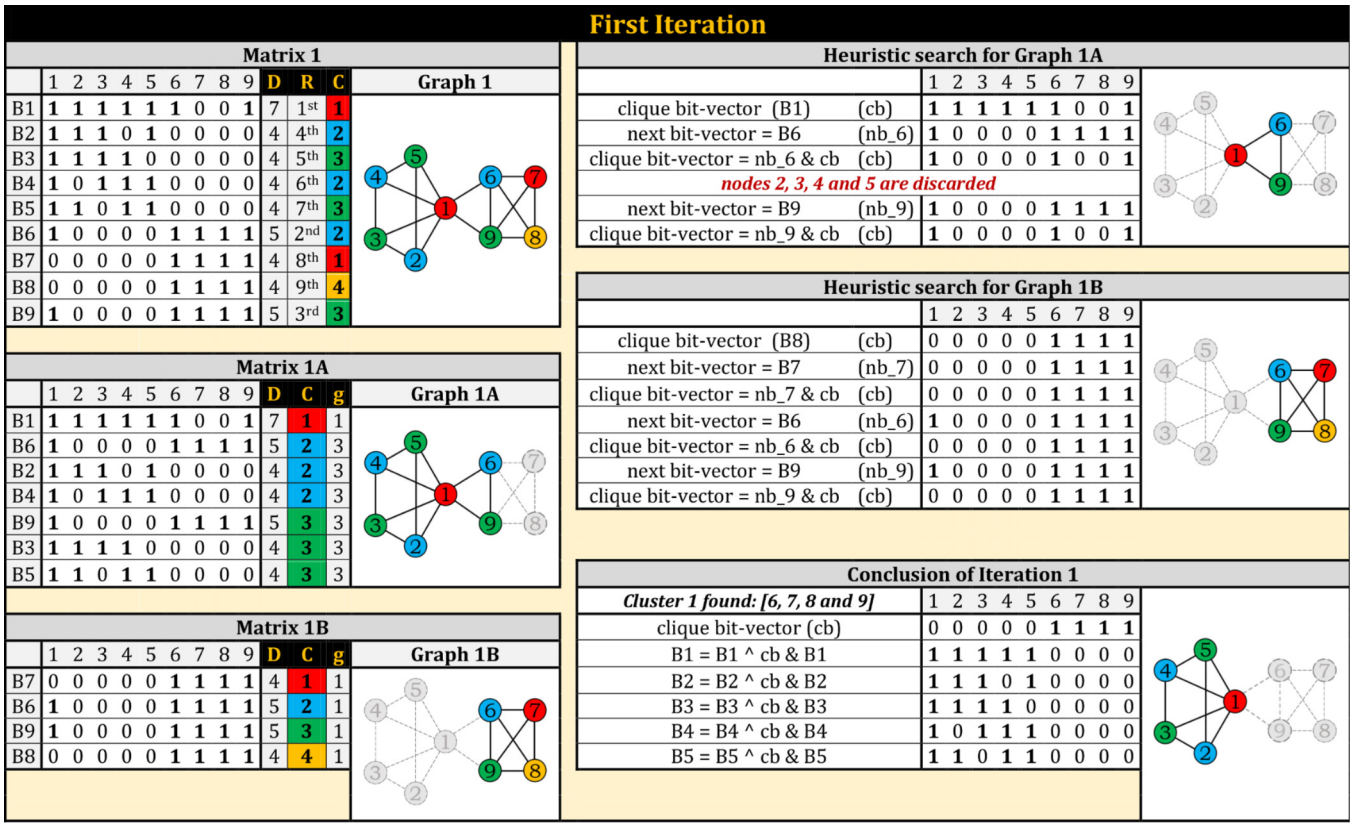

Fig. 1. First iteration of the binary heuristic for searching cliques implemented in BitQT.

Supplementary Information "S1: Pseudocode of BitQT algorithm"). A formal review of the many MCP heuristics available is out of this paper's scope and can be found elsewhere (Wu and Hao, 2015).

We start with the calculation of the binary similarity matrix that will be stored in RAM. The float vector containing the one-versus-all RMSD similarity of each frame is transformed into a bit-vector Bi (B1 to B9 in Matrix 1, Figure 1) in which $B_{ij} = 1$ if $RMSD_{ij} \leq k$, zero otherwise. Each vertex's degree is calculated as the total number of switched-on positions in the Bi vector (D column in Matrix 1, Figure 1).Note that Bi vectors always have 1 at the i^{th} position ($RMSD_{ii} == 0 \leq k$), so D column actually contain $degree + 1$ of each vertex in the trajectory graph..Then, the subsequent steps are followed.

1- Vertex 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).

2- 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). Note that for our purposes, degeneracy is perceived as the number of nodes of the color class in the context of the neighbors of a seed node, not in the entire graph (in which case using it for order would be meaningless).

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 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.

3- 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 *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).

4- 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 members 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

Table 1. Run time and RAM consumption of analyzed QT implementations on different trajectories.[1]

		BitQT		qtcluster		QTPy	
Traj. Name	**# 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)*

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 2 until no more cliques can be found.

During the execution of *BitQT*, some scenarios leading to ties may arise, for instance, selecting the node of the highest degree as seed (in "2-Clique search from the maximum degree node" and "3-Clique search from promising nodes"), or selecting the maximum clique (in "4-Conclusion and updating"). *BitQT* solves these cases by choosing the element with the lowest index among the available options as the "winner" of the tie. These ties can also appear in the original QT algorithm (see Section 3 when selecting the candidate cluster with most neighbors as a cluster). *QTPy* also picks as "winner" of the tie the element with the lowest index from the available options. Choosing one or another "winner" does impact the outcome of algorithms in terms of cluster composition. However, the choice of a "winner" in a tied scenario will never invalidate the discussed guarantees of *BitQT* or *QTPy*.

4 *BitQT* benchmark

4.1 Performance

In this section, we compare the run time and memory usage of *BitQT*, *QTPy* and *qtcluster*, which are the only QT implementations for MD we have found in the literature. These parameters are shown in Table 1 for the clustering of the six different MD trajectories that we described in Section 2 (6K, 30K, 50K, 100K, and 250K). Given that these software are programmed following distinct algorithms and also using different programming languages (Fortran 90 for *qtcluster* and Python 3 for *BitQT* and *QTPy*), we are only able to provide general insights into the disparate performances observed in Table 1.

From the three options, *QTPy* is the only one that always creates a square float matrix for saving the RMSD distances, so its RAM peak is expected to be the highest. The only exception is 6K, where the pairwise matrix uses only about 69 MB of RAM, so other data structures (or merely the molecular trajectory) will be responsible for the peak. RAM usage of *BitQT* also grows quadratically with the number of frames in the trajectory. However, as it uses bits instead of half-precision floats, there is a 16X memory saving in this object's construction compared to *QTPy*.

The memory usage of *qtcluster* may be confusing at first sight, as it can process a 250K trajectory but produced a memory crash when dealing with a simulation of 100K frames. This behavior is a direct consequence of the similarity metric, *the maximum difference between corresponding pairs of atoms*. As expressed in equation 3, under this metric, the similarity of two frames S_m and S_n is assessed by the absolute maximum value of the difference between their inter-atomic distances.

$$d_{S_m,S_n} = max_{i,j}|d_{ij}(S_m) - d_{ij}(S_n)| \quad (3)$$

This means that it is necessary to hold the square matrix of the selected inter-atomic distances for each conformation in RAM. In practice, *qtcluster* allocates the values of only one triangle of that matrix for every conformation.

The RAM used by the *qtcluster* similarity matrix (in GB) is expressed by equation 4, in which N is the total number of frames in the trajectory, m is the size of the numeric type used to express the similarity values (in bytes), and $natoms$ is the number of selected atoms. It is clear why *qtcluster* crashed at 100K but could process 250K; the 100K trajectory contained 660 atoms and 250K only 160. Substituting in equation 4 and taking $m = 4$ we obtain 81 GB for 100K and about 12 GB for 250K. Inconveniently, *qtcluster* can analyze big trajectories only when the number of selected atoms is relatively small.

$$V_{RAM_{qtcluster}} = \frac{m * N * \frac{natoms*(natoms-1)}{2}}{2^{30}} \quad (4)$$

In a nutshell, while the three algorithms have quadratic memory complexity, the costs of *BitQT* and *QTPy* are governed by the trajectory size. In contrast, *qtcluster* is dominated by the size of the atomic selection.

Run time reported in Table 1 exhibits a general trend; *QTPy* is the slowest choice, followed by *qtcluster*, which is greatly outperformed by *BitQT*. It is worth noting that *QTPy* is the only one that implements the exact version of QT (Heyer *et al.*, 1999). As we have commented before, the exact QT has a very high computational cost evinced in the *QTPy* run times. The RMSD computation step can be safely discarded as the main contributor to the slow time performance of *QTPy* because it employs the same library that *BitQT* for this purpose (MDTraj). Given its slowness, *QTPy* applications are limited to the processing of small trajectories or as a reference for the development of future QT algorithms applied to the MD field.

qtcluster was designed as a high-speed alternative for the QT partitioning of MD. The similarity metric employed by this script (equation 3) is cheaper than the more customary RMSD and avoids any alignment. Somewhat similar to *BitQT*, *qtcluster* only preserves the original condition assuring the collective similarity of retrieved clusters. For big trajectories, however, *qtcluster* is not a fast option.

Comparatively, *BitQT* has the best run time performance allowing it to handle relatively long MD trajectories. The accelerated computing of optimal RMSD distances through the MDTraj engine joined to the developed binary-based heuristic for searching cliques are the cornerstones of its cheap cost.

4.2 Preservation of the Quality Threshold

As we discussed earlier in section 3, there are two fundamental restrictions in the QT original algorithm: *Condition 1*, which requires the diameter of clusters to be of minimum size, and *Condition 2*, which ensures the respect of a quality threshold in the values of intra-cluster similarity. *BitQT* conveniently relaxed the former, but it carefully does preserve the latter one. The previous claim implies that all clusters returned by *BitQT* must have a diameter less equal than the quality threshold k.

Figure 2 shows the distribution of all clusters' diameter for every analyzed trajectory. As it is appreciated, pairwise distances between frames of the same cluster never surpass the predefined quality threshold k (4 Å for 6K and 30K, 3 Å for 250K, and 2 Å for 50K and 100K).

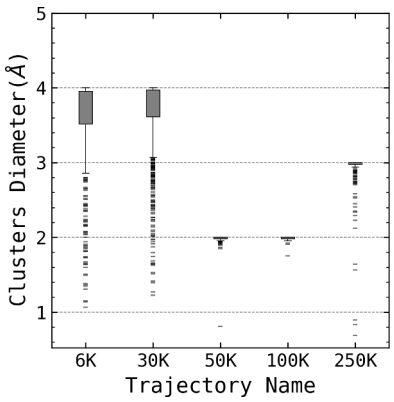

Fig. 2. Distributions of cluster diameters returned by BitQT for each analyzed trajectory.

Figure 2 also demonstrates that *BitQT* clusters are cliques in the MD trajectory graph. As we discussed in Section 3, an edge between two nodes i and j of the trajectory graph is set if and only if $d_{ij} \leq k$. If all pairwise distances between frames in every cluster are under k, then the corresponding nodes of the trajectory graph are pairwise connected, implying that clusters are indeed cliques.

4.3 Equivalence between *BitQT* and *QTPy*

If we consider an MD trajectory T as a set of N elements (frames) $T = \{t_1, t_2, ..., t_N\}$, the outcome of applying a given clustering algorithm on T is a partition P of the N objects into C clusters, $P = \{p_1, p_2, ..., p_C\}$, such that the union of all the subsets in P is equal to T and the intersection of any two subsets in P is empty. *QTPy* and *BitQT* produced such partitions ($Q = \{q_1, q_2, ..., q_C\}$ and $B = \{b_1, b_2, ..., b_C\}$ respectively) for the 6K, 30K, and 50K trajectories.

Considering $\binom{N}{2} = N(N-1)/2$ as the total number of element pairs (t_i, t_j) in T, there exist four classifications of pairs when comparing Q and B outcomes; a-) elements in a pair are placed in the same group in Q and in the same group in B (true positives), b-) elements in a pair are placed in the same group in Q and in different groups in B (false negatives), c-) elements in a pair are placed in the same group in B and in different groups in Q (false positives), and d-) elements in a pair are placed in different groups in Q and B (true negatives). It is possible to assess the equivalence between Q and B based on the number of pairs of elements lying in any of these four categories.

The Rand Index (Rand, 1971) (Equation 5) expresses the fraction of pairs of elements on which two clusterings coincide (from 0 for unrelated to 1 in a perfect match). However, RI approaches its upper limit as the number of clusters increases because d tends to grow even for poorly related partitions, giving a high score. An Adjusted RI (Hubert and Arabie, 1985; Steinley, 2004)corrected against "agreements-by-chance" (ARI) has been extensively used (Equation 6) to measure the correspondence between partitions created by clustering algorithms. ARI values extend from -1 (poorly related partitions) to 1 (highly similar partitions).

$$RI = \frac{a+d}{a+b+c+d} \tag{5}$$

$$ARI = \frac{\binom{N}{2}(a+d) - [(a+b)(a+c)+(c+d)(b+d)]}{\binom{N}{2}^2 - [(a+b)(a+c)+(c+d)(b+d)]} \tag{6}$$

An ARI analysis between partitions obtained with *QTPy* (Q) and *BitQT* (B) for trajectories 6K, 30K, and 50K is shown at Figure 3. Note that instead of reporting just the global ARI between Q and B, we explicitly compared the ARI between both partitions at the top-X clusters (Q_X and B_X), taking X from 1 (the first cluster) to C (the total number of clusters). Consequently, the global ARI between Q and B corresponds to the last point of each curve. The remaining points indicate the correspondence between the first X clusters of Q and B.

For trajectories 6K and 30K, the global ARI is 0.87, indicating a good agreement between clusters produced by *QTPy* and *BitQT*. An even higher index is reported for the first X clusters with sizes bigger than 1% of the trajectory size ($ARI_{1\%}$). These most populated clusters are often considered the most relevant of the trajectory as they groups the representative conformational states explored in an MD simulation. $ARI_{1\%}$ (represented by a bold point in Figure 3A-C) is 0.96, 0.88 and 0.XX for trajectories 6K, 30K, and 50K, respectively. This is indicative of a very good agreement between the most popular clusters obtained by*QTPy* and *BitQT*.

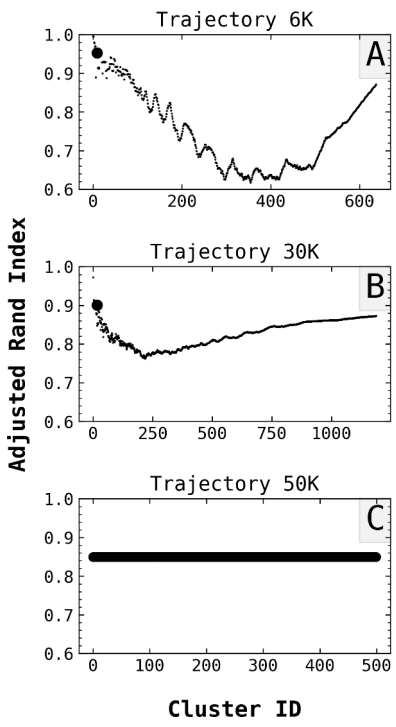

Fig. 3. To do

Observed ARI fluctuations at different top-X are expected because both algorithms pick their seeds to form clusters differently. It is possible that at a given value of X, clusters formed by *QTPy* were still not recovered by *BitQT* or vice versa. However, fluctuations are more pronounced for the less populated clusters.

5 Conclusions

The QT algorithm is an appealing option for partitioning MD trajectories as it assures a collective similarity of frames in recovered clusters. However, its inherent complexity currently limits its application. In the present work, we have relaxed a condition in the original formulation of QT

Instead of looking for minimum-sized clusters where all pairwise similarity values were under a threshold, we reformulated the problem to maximize the size of those clusters. This trivial change allowed us to

approach QT from an MCP perspective. The use of a similarity binary matrix (rather than a float-encoded one) greatly diminished the RAM resources. It made it possible to implement most clustering steps as fast bitwise operations.

Rather than an exact implementation of the MCP, we developed our modified version of QT called *BitQT* using an MCP heuristic whose out-coming clusters are in good agreement with those obtained by the original version *QTPy*. *BitQT* strictly guaranteed the preservation of the user-defined quality threshold in all reported clusters.

Acknowledgements

D.H.C. thanks Joan-Emma Shea and Zach Levine for providing the 6K trajectory used in this work.

Funding

This work was supported by the Eiffel Scholarship Program of Excellence of Campus France [P744468L to R.G.A]; the Project Hubert Curien-Carlos J. Finlay [41814TM to R.G.A, F.L, and L.M.C]; and the Fondo Nacional de Desarrollo Científico y Tecnológico [CONICYT FONDECYT/INACH/POSTDOCTORADO/No. 3170107 to E.W.H.R].

References

Abraham, M. J. *et al.* (2015). Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers.

Danalis, A. *et al.* (2012). Efficient quality threshold clustering for parallel architectures. *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012*, pages 1068–1079.

Daura, X. *et al.* (1999). Peptide Folding: When Simulation Meets Experiment. *Angewandte Chemie International Edition*, **38**(1/2), 236–240.

Dutta, S. and Overbye, T. (2011). A clustering based wind farm collector system cable layout design. In *2011 IEEE Power and Energy Conference at Illinois*, pages 1–6. IEEE.

González-Alemán, R. *et al.* (2020a). BitClust: Fast Geometrical Clustering of Long Molecular Dynamics Simulations. *Journal of Chemical Information and Modeling*, **60**(2), 444–448.

González-Alemán, R. *et al.* (2020b). Quality Threshold Clustering of Molecular Dynamics: A Word of Caution. *Journal of Chemical Information and Modeling*, **60**(2), 467–472.

Guardiani, C. *et al.* (2012). Conformational Landscape of N-Glycosylated Peptides Detecting Autoantibodies in Multiple Sclerosis, Revealed by Hamiltonian Replica Exchange. *J. Phys. Chem. B*, **116**(18), 5458–5467.

Heyer, L. J. *et al.* (1999). Exploring expression data identification and analysis of coexpressed genes. *Genome Research*, **9**(11), 1106–1115.

Hubert, L. and Arabie, P. (1985). Comparing partitions. *J. Classif.*, **2**(1), 193–218.

McGibbon, R. T. *et al.* (2015). MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal*, **109**(8), 1528–1532.

Melvin, R. L. *et al.* (2016). Uncovering Large-Scale Conformational Change in Molecular Dynamics without Prior Knowledge. *Journal of Chemical Theory and Computation*, **12**(12), 6130–6146.

Olson, M. T. *et al.* (2011). Production of reliable MALDI spectra with quality threshold clustering of replicates. *Journal of the American Society for Mass Spectrometry*, **22**(6), 969–975.

Peng, J.-h. *et al.* (2018). Clustering algorithms to analyze molecular dynamics simulation trajectories for complex chemical and biological systems. *Chinese J. Chem. Phys.*, **31**(4), 404–420.

Procacci, P. *et al.* (1997). ORAC: A molecular dynamics program to simulate complex molecular systems with realistic electrostatic interactions. *Journal of Computational Chemistry*, **18**(15), 1848–1862.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.*, **66**(336), 846–850.

Röttger, R. (2016). Clustering of Biological Datasets in the Era of Big Data. *Journal of integrative bioinformatics*, **13**(1), 300.

San Segundo, P. and Artieda, J. (2015). A novel clique formulation for the visual feature matching problem. *Applied Intelligence*, **43**(2), 325–342.

San Segundo, P. and Tapia, C. (2014). Relaxed approximate coloring in exact maximum clique search. *Computers and Operations Research*, **44**, 185–192.

San Segundo, P. *et al.* (2013). An improved bit parallel exact maximum clique algorithm. *Optimization Letters*, **7**(3), 467–479.

San Segundo, P. *et al.* (2016). A new exact maximum clique algorithm for large and massive sparse graphs. *Computers and Operations Research*, **66**, 81–94.

San Segundo, P. *et al.* (2017a). A parallel maximum clique algorithm for large and massive sparse graphs. *Optimization Letters*, **11**(2), 343–358.

San Segundo, P. *et al.* (2017b). An enhanced bitstring encoding for exact maximum clique search in sparse graphs. *Optimization Methods and Software*, **32**(2), 312–335.

San Segundo, P. *et al.* (2010). Fast exact feature based data correspondence search with an efficient bit-parallel MCP solver. *Applied Intelligence*, **32**(3), 311–329.

Seeber, M. *et al.* (2007). Wordom: A program for efficient analysis of molecular dynamics simulations. *Bioinformatics*, **23**(19), 2625–2627.

Shea, J.-E. and Levine, Z. A. (2016). Studying the early stages of protein aggregation using replica exchange molecular dynamics simulations. In *Protein Amyloid Aggregation*, pages 225–250. Springer.

Steinley, D. (2004). Properties of the Hubert-Arabie adjusted Rand index. *Psychol. Methods*, **9**(3), 386–396.

Steipe, B. (2002). A revised proof of the metric properties of optimally superimposed vector sets. *Acta Crystallographica Section A*, **58**, 506.

Tang, Z. *et al.* (2010). A new method for alignment of lc-maldi-tof data. In *2010 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 346–351. IEEE.

Tubiana, T. *et al.* (2018). TTClust: A Versatile Molecular Simulation Trajectory Clustering Program with Graphical Summaries. *Journal of Chemical Information and Modeling*, **58**(11), 2178–2182.

von Luxburg, U. *et al.* (2012). Clustering: Science or Art? *JMLR: Workshop and Conference Proceedings*, **27**, 6579.

Wu, Q. and Hao, J.-K. (2015). A review on algorithms for maximum clique problems. *European Journal of Operational Research*, **242**(3), 693–709.

Yaakob, S. N. and Jain, L. (2012). An insect classification analysis based on shape features using quality threshold ARTMAP and moment invariant. *Applied Intelligence*, **37**(1), 12–30.

Yaakob, S. N. *et al.* (2010). A novel Euclidean quality threshold ARTMAP network and its application to pattern classification. *Neural Computing and Applications*, **19**(2), 227–236.