Comparing geometric and kinetic cluster algorithms for molecular simulation data

Bettina Keller,[1,a)] Xavier Daura,[2] and Wilfred F. van Gunsteren[1,b)]
[1]*Laboratory of Physical Chemistry, Swiss Federal Institute of Technology, ETH, CH-8093 Zürich, Switzerland*
[2]*Catalan Institution for Research and Advanced Studies (ICREA) and Institute of Biotechnology and Biomedicine (IBB), Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain*

(Received 31 January 2009; accepted 8 January 2010; published online 19 February 2010)

The identification of metastable states of a molecule plays an important role in the interpretation of molecular simulation data because the free-energy surface, the relative populations in this landscape, and ultimately also the dynamics of the molecule under study can be described in terms of these states. We compare the results of three different geometric cluster algorithms (neighbor algorithm, K-medoids algorithm, and common-nearest-neighbor algorithm) among each other and to the results of a kinetic cluster algorithm. First, we demonstrate the characteristics of each of the geometric cluster algorithms using five two-dimensional data sets. Second, we analyze the molecular dynamics data of a β-heptapeptide in methanol—a molecule that exhibits a distinct folded state, a structurally diverse unfolded state, and a fast folding/unfolding equilibrium—using both geometric and kinetic cluster algorithms. We find that geometric clustering strongly depends on the algorithm used and that the density based common-nearest-neighbor algorithm is the most robust of the three geometric cluster algorithms with respect to variations in the input parameters and the distance metric. When comparing the geometric cluster results to the metastable states of the β-heptapeptide as identified by kinetic clustering, we find that in most cases the folded state is identified correctly but the overlap of geometric clusters with further metastable states is often at best approximate. © *2010 American Institute of Physics*. [doi:10.1063/1.3301140]

I. INTRODUCTION

Molecular simulation techniques, such as molecular dynamics (MD) or Monte Carlo (MC),[1] are powerful tools for the elucidation of the microscopic structure and the dynamics of biomolecules and for the elucidation of the functionality associated with the former two. As raw output, these simulations typically produce a large Boltzmann-weighted ensemble of molecular structures or statistical mechanical configurations which serve as the basic data set for further analysis. If one is solely interested in some average over the ensemble, it can be calculated without further detailed analysis. If one would (additionally) like to organize and conceptualize this enormous structural data set in order to understand (i) which parts of the configurational space are accessible to the molecule at a given temperature, (ii) how the molecule moves within this space, and (iii) how these parts and movements eventually connect to particular macroscopic and microscopic properties, clustering of the large amount of configurational data in one way or the other is a necessity. In this context, the term conformation is used to describe a (small) part of the configurational space or a subset in the configurational ensemble that comprises of structurally related configurations. In small molecules the conformations are usually defined such that they represent a minimum in the free-energy surface, i.e., a metastable state, and that different conformations are separated by significant barriers from each other. It is, however, important to point out that there is no exact definition of how to map a given configurational or structural ensemble onto different conformations.

For small molecules, such as *n*-butane, $CH_3(CH_2)_2CH_3$, conformational assignment can be done based on a somewhat intuitive insight into the molecule's conformational space. According to this insight, the largest barriers in *n*-butane will occur along the $C_1-C_2-C_3-C_4$-dihedral angle ϕ and the molecule's conformational isomers are *gauche* $(-)$ with $\phi \approx 60°$, *trans* with $\phi \approx 180°$, and *gauche* $(+)$ with $\phi \approx 300°$, of which the *trans*-conformation is the lowest in energy because in this conformation the two CH_3 units are farthest apart from each other. The strength of this description becomes immediately obvious if one acknowledges the fact that *n*-butane has $N=14$ atoms and therefore $3N-6=36$ internal configurational degrees of freedom. This high-dimensional space is projected onto only three states which suffice to accurately describe the molecule's free-energy landscape, the relative populations in this landscape, and the dynamics of the molecule.

For larger molecules, such an intuitive partitioning of the structural ensemble becomes impossible due to the enormous size and complexity of the configurational space. Consequently, also the term conformation becomes fuzzier. Frequently, one resorts to grouping structures according to conformational similarity using geometric cluster algorithms and

a)Electronic mail: bettina@igc.phys.chem.ethz.ch.
b)Author to whom correspondence should be addressed. Electronic mail: wfvgn@igc.phys.chem.ethz.ch.

refers to the resulting clusters as conformations.[2,3] In this sense, the term conformation merely denotes a subensemble of structures that are more similar to each other than to the remaining structures of the ensemble according to a given similarity measure. In order to interpret these results, it is then often (tacitly) assumed that these clusters also represent metastable states.[4,5] The underlying assumption here is that large conformational changes are likely to be hindered by barriers in the free-energy surface, whereas structures that populate a common minimum in the free-energy surface should be conformationally similar. Although these assumptions seem reasonable at a first glance, there is no guarantee that they are correct for all types of molecules. As a matter of fact, there are known cases in which large movements of, e.g., side chains or other flexible parts of a molecule are hardly hindered by any barrier.[6,7] If such a behavior is known or suspected for a molecule, the problem is commonly evaded by adapting the similarity measure, thereby introducing a certain arbitrariness into the conformational analysis. For example, when structurally clustering small peptides with an eye to the (un)folding equilibrium, one typically includes only the backbone atoms or C_α-atoms of the central amino residues and neglects the movement of the side chains and the terminal residues.

In order to extract information such as relative free energies, rates of conformational changes, or folding pathways from MD simulations, it is essential to know the basins of the free-energy surface as precisely as possible. A basin can be either described as a region in phase space which is surrounded by free-energy barriers or—in terms of lifetimes—as a region in phase space in which a molecule is likely to stay for a long time, i.e.,

$$t_{ii}(\Delta\tau) \gg t_{ij}(\Delta\tau), \tag{1}$$

where t_{ii} denotes the probability to stay in region i for a time period $\Delta\tau$ and t_{ij} denotes the transition probability of going from region i to region j within time $\Delta\tau$. Note that Eq. (1) represents a clear definition of the term *metastable state*, which is independent of the size of the molecule. The link between free-energy barrier heights and transition probabilities can be understood with help of the Arrhenius equation, which states that the rate k_{ij} of going from state i to j decreases exponentially as the barrier between the two states, the activation energy E_{ij}, increases,

$$k_{ij} = A \exp\left(-\frac{E_{ij}}{RT}\right), \tag{2}$$

where A is a prefactor that depends on the system, R is the gas constant, and T is the temperature. In other words, if a state i is surrounded by large barriers, all rates k_{ij} are small and the probability of staying in i for a long period of time is high.

Although the term metastable state has a precise definition, actually assessing the metastable states of a large molecule proves to be challenging and rather costly.[8–10] In order to identify metastable states, the phase space—usually reduced to the molecule's conformational subspace—is discretized into the so-called microstates, where the term *microstates* simply denotes a *very small part of the phase space*.[9] These states are then grouped together into metastable states according to kinetic proximity, i.e., the transition probability of one microstate to another microstate of the same group should be much higher than the transition probability to a microstate outside the group—a technique to which we will refer as *kinetic clustering*.[8–12] The reason why this procedure is costly is that the transition probabilities between all pairs of microstates have to be sampled to convergence, whereas in geometric clustering, only the conformational space has to be sampled with the appropriate weights. Nevertheless, kinetic clustering is—in principle—capable of directly identifying the metastable states of a molecule.

In this contribution, we first wish to assess how sensitive the geometric cluster results are with respect to the choice of the algorithm and its parameters and with respect to the choice of the similarity measure. Second, we wish to test how accurately geometric cluster algorithms reproduce the metastable states. To this end, we cluster 15 000 structures of a β-heptapeptide obtained from a MD simulation using three different geometric cluster algorithms and compare the results. Then, we identify the metastable states of the molecule by kinetic clustering and using 20 trajectories of a length of 500 ns, each of which started from a different initial configuration. We sort the data set that was used for the geometric clustering into these metastable states and compare the result to the clusters obtained using the geometric cluster algorithms.

II. THEORY

A. Geometric cluster algorithms

In geometric cluster algorithms, data points are literally understood as points in a (potentially high-dimensional) space for which some distance metric is defined.[13] The goal of the algorithm is then to partition a data set S into smaller sets $\{s_1, s_2, s_3, \ldots\}$ such that the distances between the data points of a given set are smaller than their distances to data points in any other set,

$$S = \{s_1, s_2, s_3, \ldots, s_c\}. \tag{3}$$

These sets are called "clusters." We only consider nonfuzzy cluster algorithms, that is, algorithms that sort data points into *disjoint* clusters,

$$s_i \cap s_j = 0 \quad \forall\, i,j \quad \text{with} \quad i \neq j. \tag{4}$$

Given a distance measure d_{ij} between two data points i and j, a distance matrix D for all pairwise distances in S can be constructed. In principle, this distance matrix then has to be permuted in such a way that it takes an approximately block-diagonal form, where the blocks represent the clusters and the matrix elements in the blocks should be smaller than all other matrix elements,

$$\mathbf{D}' = \mathbf{PDP}^{-1}. \tag{5}$$

Due to the sheer number of possible permutations—they grow with $n!$, where n is the number of data points in S—it is usually impossible to find the optimal permutation in a brute force manner. Instead, geometric cluster algorithms

rely on a number of iterative schemes and different convergence criteria.[14,15]

We tested and compared three different types of geometric cluster algorithms, the first one being the neighbor algorithm, as described in Ref. 4. This is a very simple and fast algorithm in which the neighbors of a data point i are defined to be those data points that lie within a (predefined) distance from i. The data point with the most neighbors is considered to be the medoid of the first cluster and all its neighbors are members of the cluster. After the first cluster has been found, all its members are removed from the data pool and the algorithm is iterated until all data points have been assigned to a cluster. The algorithm has one parameter: the distance cutoff c, which represents the radii of the clusters. The neighbor approach has the advantage that one does not need to keep the entire distance matrix in the working memory of the computer, it rather suffices to store a list of neighbors for each data point. This allows the processing of large data sets and makes the algorithm very fast.

The second algorithm we tested is the K-medoids algorithm, which belongs to the class of partitioning cluster algorithms. These algorithms assign in an initialization step all data points to a predefined number of clusters and then iteratively optimize the assignment until some convergence criterion is reached. Partitioning algorithms have the advantage that wrong cluster assignments made in the course of the algorithm have the chance to be corrected during a later iteration. However, they need the total number of clusters k as an input parameter—a number that is usually not known *a priori*. It can be estimated by preclustering the data using other algorithms or by applying the partitioning algorithms several times with different total numbers of clusters as input and then deciding, using some cluster validity measure, which clustering represents the structure of the data set best. Also, the initial cluster assignment, which is usually done in a random fashion, heavily influences the final partitioning of the data set, and it is therefore customary to apply the algorithm several times to the same data set with the same input parameters but different assignments. On the one hand, this approach can reveal several equally good partitionings, on the other hand, it is not always possible to decide unambiguously which solution is the best.

The third algorithm is a close variant of the Jarvis–Patrick algorithm,[16] which we—in order to have a more descriptive name—will call the common-nearest-neighbor algorithm. In contrast to most geometric cluster algorithms, this algorithm is *not* based on the idea that members of a cluster are closer to each other than to all other data points in the data set, and it therefore also abandons the notion that clustering is equivalent to reorganizing the distance matrix into a block-diagonal form. Instead, it bases its cluster definition on a measure for the local data-point density around a point i which mimics the way we (as human beings) intuitively recognize clusters in data sets such as, e.g., the scatterplots in Fig. 1. For this intuitive perception, neither the distance of a given data point from the cluster center nor the specific shape of the cluster—parameters that are utilized in many geometric cluster algorithms—plays a crucial role. Nor is the condition that intracluster distances should be smaller

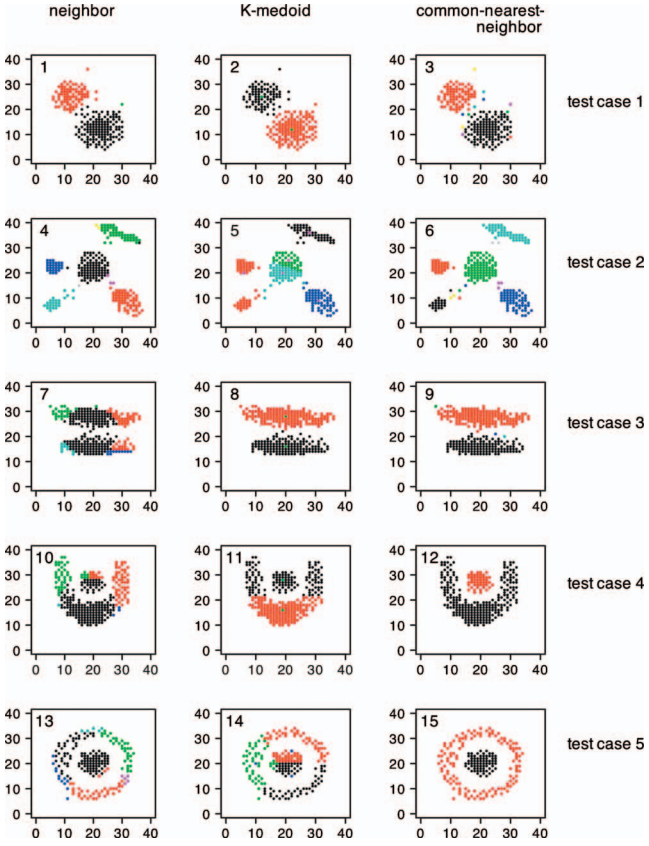

FIG. 1. Results of three geometrical cluster algorithms-neighbor algorithm (column 1), K-medoids algorithm (column 2), common-nearest-neighbor algorithm (column 3) for five 2D test cases (rows 1–5). Data points of the same color belong to one cluster. For the K-medoids algorithm, the cluster centers are indicated as differently colored dots at the center of each cluster.

than intercluster distances always fulfilled (see Fig. 1, test cases 3–5). Rather, we perceive the clusters as continuous areas of high data point density and the cluster boundaries are designated by a steep drop in data point density. In the common-nearest-neighbor algorithm, a hitherto unassigned data point is added to a cluster if it is connected to one of its members by an area of sufficiently high data-point density. A cluster is complete and is removed from the data pool if no further data point can be added. Since the data-point density between two data points i and j is hard to calculate if i and j are points in a high-dimensional space, it is instead estimated as the number of their common nearest neighbors. The nearest neighbors of i are those data points that lie within the *nearest-neighbor-distance cutoff* (nndc). The number of common-nearest neighbors of i and j is the number of data points that are both nearest neighbors of i and of j. (This number is, of course, 0 if i and j are further apart than 2nndc.) Suppose i is a member of cluster c_1 and j is still unassigned, then j will become a member of c_1 if it has at least *nearest-neighbor-number cutoff* (nnnc) neighbors with i or any other member of c_1. Once it is assigned to c_1, it can also attract unassigned data points. In contrast to the original formulation of the Jarvis–Patrick algorithm, in which the n data points closest to i are considered to be neighbors of i independent of their distance to i, in the common-nearest-neighbor algorithm *only* data points that lie within nndc are considered to be neighbors. Neither is the total number of

nearest neighbors restricted nor do all data points necessarily have nearest neighbors. In this way, it ensured that the algorithm stops enlarging the cluster when it reaches an area with a drop in the local data-point density. Other than in the neighbor algorithm where distance cutoff represents the cluster radius, nndc designates a very local area around a data point and should be much smaller than the cluster radius. However, just as in the neighbor algorithm, it is not necessary to store the entire distance matrix in the working memory; instead, it suffices to keep the neighbor list, which makes it easy to handle large data sets.

In this study, we use the atom-positional root-mean-square difference (RMSD) as distance measure between two i and j,

$$\mathrm{RMSD}_{ij} = \sqrt{\frac{1}{N}\sum_{k=1}^{k=N} (\mathbf{x}_{i,k} - \mathbf{x}_{j,k})^2}. \tag{6}$$

Here, N is the number of atoms, and $\mathbf{x}_{i,k}$ and $\mathbf{x}_{j,k}$ are the respective positions of atom k in structures i and j, possibly after translational superposition of the centers of mass and rotational fit. Although RMSD is the preferred distance measure in most cluster studies of molecular simulation data, other measures such as the dihedral angle RMSD are possible.

B. Kinetic cluster algorithms

In contrast to geometric cluster algorithms, which partition a given data set without any direct reference to the underlying conformational space, kinetic cluster algorithms directly partition the conformational space into regions such that each region represents a metastable state [Eq. (1)]. Metastable states are equivalent to minima in the free-energy surface, and the Arrhenius equation [Eq. (2) links the free-energy-barrier heights surrounding a minimum to the probability of staying in this minimum, i.e., its metastability.

The first and crucial step in kinetic cluster algorithms is the discretization of the conformational space C into disjoint microstates,

$$C = \{\mu_1, \mu_2, \mu_3, \ldots\}, \tag{7}$$

where the microstates typically form a complete cover of C. The definition of the microstates is not trivial. On the one hand, they have to be large enough that their total number is still computationally manageable; on the other hand, they have to be so small that they represent the conformational space with sufficient resolution. A variety of approaches to this problem have been developed.

The aim of kinetic cluster algorithms is to group these microstates according to the kinetic proximity where the transition probability $t_{ij}(\Delta\tau)$, which is the probability of finding the system in μ_j at time $t+\Delta\tau$ given that it was in μ_i at time t,

$$t_{ij}(\Delta\tau) = P(\mu_j(t+\Delta\tau)|\mu_i(t)), \tag{8}$$

is used as measure of the kinetic proximity. The higher the value of t_{ij}, the closer two mircostates μ_i and μ_j are kinetically. Note that the transition probability must not depend on the state of the system at $t-\Delta\tau, t-2\Delta\tau, \ldots$, that is to say, the system must have a Markovian behavior on the time scale $\Delta\tau$. Analogous to geometric cluster algorithms, one arranges these t_{ij} in a matrix of kinetic proximities, the transition matrix $\mathbf{T}$. Since the elements of the matrix represent probabilities and the system can either stay in the current state or transfer to another state, $\mathbf{T}$ is a row-stochastic matrix,

$$\sum_j t_{ij} = 1 \quad \forall\, i. \tag{9}$$

One now looks for a permutation $\mathbf{P}$, which transforms $\mathbf{T}$ into an almost block-diagonal form such that the matrix elements within one block are significantly larger than the off-block elements,

$$\mathbf{T}' = \mathbf{PTP}^{-1}. \tag{10}$$

Each block then corresponds to a metastable state and the number of the metastable states can be determined by analyzing the eigenvalue spectrum of the transition matrix $\mathbf{T}$.[12] However, just as with the geometric cluster algorithms, a brute-force search for $\mathbf{P}$ is prohibitive because the number of possible permutations grows with $n!$, where n is the number of microstates. In contrast to the distance matrices that are used in geometric clustering, one can here exploit the fact that transition matrices are row-stochastic and can be coarse grained to a transition matrix $\mathbf{T}_{cg}$ over the metastable states. The search-problem for $\mathbf{P}$ then becomes an optimization problem in which the trace of the coarse-grained matrix $\mathbf{T}_{cg}$ has to be maximized and which could be tackled, e.g., using a Monte Carlo simulated annealing (MCSA) scheme.

In practice, one finds that in order for the MCSA to converge to a reasonable result, one needs to have a good starting guess. We use an idea proposed by Deuflhard *et al.*[12] which exploits the properties of almost block-diagonal row-stochastic matrices to generate such a starting guess. See Ref. 17 for an illustration of the transformation of a sample transition matrix $\mathbf{T}$ to the corresponding coarse-grained transition matrix $\mathbf{T}_{cg}$.

1. Definitions of the microstates

Numerous methods for discretizing the conformational space C into microstates $\mu_1, \mu_2, \mu_3, \ldots$ have been proposed in literature, of which we summarize the eminent ones. Uniformly discretizing each (slow) degree of freedom into small bins[12,18–20] suggests itself and is also the most rigorous of all methods. However, due to combinatorial explosion, this approach yields unmanageably large numbers of microstates for systems with more than a few degrees of freedom. One way to evade this problem is by reducing the dimensionality of the space using, e.g., essential dynamics[11] or by discretizing each degree of freedom independently along the minima of its marginal probability distribution[21] (performing a kinetic clustering on the isolated degrees of freedom).[9] The latter two approaches are, however, only valid if the degrees of freedom are mutually independent. An intermediate approach is to perform a kinetic cluster analysis on one degree of freedom at a time and successively subdivide the data set along the resulting clusters.[22]

Alternatively, one can detach oneself from the description of the conformational space in terms of internal degrees

of freedom and instead regard the conformations of the molecule as a whole, e.g., by performing a geometric cluster analysis which is then iteratively refined using kinetic clustering[8] or by projecting the configuration of a protein onto a binary code in which each digit represents one amino acid and denotes whether this amino acid is in a helical state or not (encoded as 1 and 0, respectively).[23] In another version of these secondary structure strings, multiple secondary states are encoded as letters and the configuration of the protein is represented by a letter string.[10]

2. Generation of the transition matrix

Once one has decided on a definition of the microstates, each structure $\mathbf{q}(t)$ from a trajectory of molecular structures is projected onto the appropriate microstate $\mu=f(\mathbf{q}(t))$, thereby transforming this trajectory into a trajectory of microstates

$$(\mathbf{q}(0),\mathbf{q}(t_1),\mathbf{q}(t_2),\ldots)\rightarrow(\mu(0),\mu(t_1),\mu(t_2),\ldots). \tag{11}$$

It is now possible to count the transitions $\mu_i\rightarrow\mu_j$ that occur within a lag time of $t=\Delta\tau$.

By moving a frame of $t=\Delta\tau$ over a MD trajectory, one obtains the number of transitions f_{ij} between microstate μ_i and microstate μ_j which can be arranged into a so-called frequency matrix $\mathbf{F}$. Normalizing the rows of $\mathbf{F}$, one obtains the transition matrix $\mathbf{T}$ whose elements t_{ij} represent the probability of moving from microstate μ_i to microstate μ_j within time $t=\Delta\tau$.

3. Markovian behavior

The description of the kinetics of a molecules in terms of a transition matrix is *only* valid if the kinetics are Markovian on the chosen time scale $\Delta\tau$. Given that the system is in microstate μ_i at time t, the probability that it will be in μ_j at $t+\Delta\tau$ must not depend on the previous states of the system but only on a time-invariant transition probability t_{ij},

$$t_{ij}=P(\mu(t+\Delta\tau)=\mu_j|\mu(t)=\mu_i). \tag{12}$$

Analyzing the eigenvalues $\lambda_i(\tau)$ of transition matrices with varying lag times τ of a given system is an elegant way to test whether the dynamics of the systems is Markovian.[24] The quantity

$$\tau_{c,i}=-\frac{\tau}{\ln\lambda_i(\tau)} \tag{13}$$

is a characteristic time scale for the decay of the eigenvalue $\lambda_i(\tau)$ with increasing lag time τ. If the transition matrices describe the dynamics of a Markovian system, this time scale should be a constant and plotting $\tau_{c,i}$ versus τ should result in a horizontal line.

III. METHODS

A. Simulation

Twenty production runs of the β-heptapeptide H-β-HVal-β-HAla-β-HLeu-(S,S)-β-HAla(αMe)-β-HVal-β-HAla-β-HLeu-OH (see Fig. 2) in methanol were generated. The starting structures for each of the replicas were

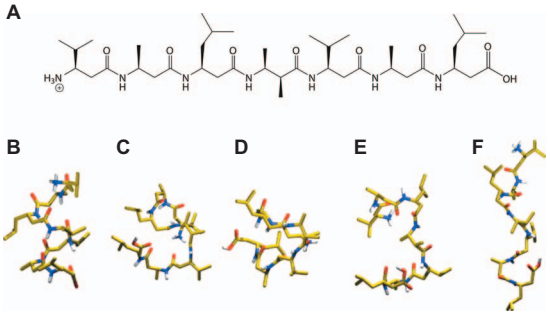

FIG. 2. Panel (a): chemical structure of the β-heptapeptide H_2-β-HVal-β-HAla-β-HLeu-(S,S)-β-HAla(αMe)-β-HVal-β-HAla-β-HLeu-OH^+. Panel (b): structure of the folded state of this β-heptapeptide. Panels (c)–(e): arbitrarily chosen unfolded structures of this β-heptapeptide.

taken randomly from a previous simulation[25] of 400 ns. Each of the replicas was simulated for 500 ns, adding up to a total of 10 μs of the simulation data. The simulations were carried out with the GROMOS96 software[26] and the GROMOS 43A1 force field[26] as previously.[25] All bond lengths were constrained using the SHAKE algorithm,[27] allowing for a time step of 2 fs. Solute configurations were saved every 0.1 ps. The system was simulated in a rectangular box using periodic boundary conditions. The volume was kept constant, and the solvent and solute molecules were independently weakly coupled to temperature baths of 310 K (Ref. 28) with a coupling time of 0.1 ps. The number of solvent molecules was 962. We used 0.8 nm/1.4 nm as twin-range cutoff and 1.4 nm as reaction field cutoff with $\epsilon_{rf}=1.0$. The atom pair list for short-range interactions and the intermediate-range forces were updated every 5 steps.

B. Geometric cluster algorithms

We extracted structures at intervals of 0.1 ns from the simulations (5000 structures per replica). 0.1 ns is the time step for which the system starts to behave Markovian (cf. Sec. III C 2) and the structures that are separated by this time interval can be regarded as uncorrelated. For these structures—after a translational superposition and rotational fit—distance matrices were calculated using three different distance measures:

(1) atom-positional RMSD of all atoms (aa),
(2) atom-positional RMSD of all backbone atoms (bb_{1-7}), and
(3) atom-positional RSMD of the backbone atoms of residues 2–6 (bb_{2-6}).

Generally, for molecules comparable in size to our β-heptapeptide, a few thousand structures are accepted to be sufficient for a structural cluster analysis. To test whether our RMSD matrices were indeed converged, we constructed RMSD matrices with different numbers of structures: 5000, 10 000, 15 000, 20 000, and 25 000.

We used three different geometric cluster algorithms to cluster these data sets: the neighbor algorithm, the K-medoids algorithm, and the common-nearest-neighbor algorithm. We present here the pseudocode of the common-nearest neighbor algorithm because this algorithm differs

TABLE I. Microstate boundaries of the flexible backbone torsional angles of residues 2–6 in the β-heptapeptide. Values are in degrees. The *cis*-conformation corresponds to 0°.

Residue No.	Residue name	$C(O)_{i-1}-N_i-C_{\beta,i}-C_{\alpha,i}$	$N_i-C_{\beta,i}-C_{\alpha,i}-C(O)_i$	$C_{\beta,i}-C_{\alpha,i}-C(O)_i-N_{i+1}$
2	β-HAla	0; 115	0; 120; 240	0; 190
3	β-HLeu	0; 135	0; 120; 240	90; 180; 240; 330
4	(S,S)-βHAla(αMe)	0; 125	0; 110; 240	0; 145
5	β-HVal	0; 120	0; 105; 240	0; 115; 180; 240
6	β-HAla	0; 130	0; 115; 240	0; 115; 185; 245

from the more commonly used K-medoids and neighbor algorithms in that it bases its cluster criterion on a local density estimate. The pseudocode of the two other algorithms can be found in the Appendix.

1. Common-nearest-neighbor-cluster algorithm

The common-nearest-neighbor-cluster algorithm has two input parameters—the nearest-neighbor-distance cutoff nnnc and the nearst-neighbor-number cutoff nncc—and is composed of the following steps:

(1) Loop over all data points that have not been assigned to a cluster yet.
- Find the data point with the highest number of nearest neighbors within the *nndc*.

(2) This data point is the medoid of the current cluster.
(3) Loop over all data points that have not been assigned to a cluster yet and keep looping until no further data point can be added to the current cluster.
- For each data point, check if its number of common-nearest neighbors with any of the points that have been assigned to the current cluster so far is equal to or greater than the nnnc.
- If this is true, add this data point to the current cluster.

(4) Add the current cluster to the list of clusters and remove its members from the pool of unassigned data points.
(5) Repeat steps (1)–(4) until all data points have been assigned to a cluster.

C. Kinetic cluster algorithms

1. Definition of the microstates and generation of the transition matrix

Two approaches are possible for the definition of microstates: (i) sorting the structure of the trajectory into very small clusters using a geometric cluster algorithm or (ii) discretizing the conformational space (or subspace of the conformational space) directly. With regard to a comparison with the results of geometric cluster algorithms, the former approach has the advantage that one could use the same metric for geometric and kinetic clustering, but the disadvantage that a microstate definition which relies on a geometric cluster algorithm is likely to bias the comparison. Also note that the concept of metastable states exists, independent of the representation of the molecule and as long as the chosen metric does not obscure the barriers in the system, kinetic clustering should reliably yield the metastable states. We therefore opted for the latter approach.

We discretized the three possible backbone torsional angles of residue i, $C(O)_{i-1}-N_i-C_{\beta,i}-C_{\alpha,i}$, the $N_i-C_{\beta,i}-C_{\alpha,i}-C(O)_i$, and the $C_{\beta,i}-C_{\alpha,i}-C(O)_i-N_{i+1}$ dihedral angles of residues 2–6 following a procedure described in Ref. 9. In this approach one checks whether the torsional angles are mutually independent and, if so, performs a kinetic cluster analysis on each torsional angle separately thereby discretizing this degree of freedom into a small number of bins. The possible microstates of the molecule are then a combination of these bins. The $C_{\alpha,i}-C(O)_i-N_{i+1}-C_{\beta,i+1}$ dihedral angle does not need to be discretized because it is the dihedral angle of the peptide plane which is restrained to a planar conformation. This yielded two dihedral angle microstates for each $C(O)_{i-1}-N_i-C_{\beta,i}-C_{\alpha,i}$ dihedral angle, three microstates for each $N_i-C_{\beta,i}-C_{\alpha,i}-C(O)_i$ dihedral angle and four (residues 3, 5, and 6) [two (residues 2 and 4)] for the $C_{\beta,i}-C_{\alpha,i}-C(O)_i-N_{i+1}$ dihedral angles. The exact boundaries of these dihedral angle microstates are given in Table I. A microstate of the overall peptide conformation is constructed as a combination of dihedral angle microstates. With the given discretization this amounts to a total of 1 990 656 possible microstates, most of which are, however, never visited during the simulation. In order to decide which of all these possible microstates should be taken into account for the construction of the transition matrix, we counted how often each of the possible microstates was visited during the 10 μs of simulation and discarded those microstates that were visited by less then 0.01% of all trajectory structures. This yielded a total of 87 microstates for which we constructed transition matrices with lag times ranging from $\tau=10$ ps to $\tau=500$ ps. Transitions from and to the discarded microstates were not included in the construction of the transition matrices and detailed balance was enforced by reading out the trajectories forward and backward, i.e., each transition from a microstate i to a microstate j was also counted as a transition from j to i. From a test set of 15 000 structures, 11 942 fall within these 87 microstates and 3058 structures occupy one of the discarded microstates and were classified as unstructured data (cf. line 2 in Table 8).

2. Identification of the metastable states

We checked whether the dynamics of the β-heptapeptide can be described as a Markov process using Eq. (13). For lag times τ greater than 100 ps, the characteristic time scales for

TABLE II. Combinations of nndc and nnnc for which test cases 1–5 were clustered correctly by the nearest-common-neighbor algorithm and the resulting number of large clusters (>5 members), small clusters (1–5 members), and the total number of clusters.

	Test case 1				Test case 2				Test case 3				Test case 4				Test case 5			
nndc	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5
nnnc	2	6	11	20	2	3	10	13	2	5	11	⋯	2	4	5	14	⋯	2	4	13
Large	2	2	2	2	5	5	5	5	2	2	2	⋯	2	2	2	2	⋯	2	2	2
Small	5	7	8	13	7	0	12	6	3	3	5	⋯	0	0	0	3	⋯	0	1	6
Total	7	9	10	15	12	5	17	11	5	5	7	⋯	2	2	2	5	⋯	2	3	8

the largest eigenvalues become approximately constant and we chose the transition matrix with $\tau=300$ ps for the identification of the metastable states. A plot of the eigenvalues λ_i of $\mathbf{T}(\tau=300$ ps) yielded a gap between the fifth and sixth eigenvalues. We can therefore expect to find five metastable states.[12] After having generated a starting guess for the definition of these five metastable states using an idea by Deuflhard *et al.*,[12] we optimized the state definition by maximizing the trace of the coarse-grained transition matrix [see Eq. (10) and Eqs. (A1)–(A3) in Ref. 17] using the MCSA scheme.[8] We started the MCSA at a temperature of $T_{\mathrm{MCSA}}=0.006$ and decreased it to 0.0001 in 60 steps, making 1000 trial moves at each temperature. A trial move consisted of randomly picking a microstate within a randomly chosen metastable state and assigning it to another metastable state which was also chosen randomly. If the trace of the resulting coarse-grained matrix was greater than or equal to the trace of the current coarse-grained matrix, we always accepted it. If it was smaller, we accepted it with a probability of

$$p=\exp(\Delta \mathrm{Tr}/T) \tag{14}$$

where ΔTr is the difference in the traces of two matrices and T is the current temperature.

Merging two metastable states often leads to a local maximum of the trace in which the algorithm gets trapped. For this reason, we prohibited empty metastable states, i.e., if a metastable state at some point in the optimization consisted of only one microstate, then this microstate could not be chosen for a trial move. Note, however, that if a system with n metastable states is described by a transition matrix with less than n states (merged metastable states), the trace of this matrix lies below the optimal trace. Therefore, the merging of metastable states during the optimization is an indication that the starting guess or temperature or the temperature steps of the MCSA scheme were not chosen appropriately. We repeated the algorithm 80 times and used the definition of the metastable states that corresponded to the coarse-grained matrix with the largest trace.

IV. RESULTS

A. Test cases

Two-dimensional (2D) data sets, such as the five test cases in Fig. 1, are particularly useful when characterizing geometric cluster algorithms because in contrast to high-dimensional data sets the results can be directly represented in terms of 2D scatterplots, thereby revealing the features and peculiarities of the cluster algorithm. Note that although the cluster algorithms might show a more complex behavior for higher-dimensional data sets, flaws which were detected for the 2D test cases will definitely also affect the results when the algorithms are applied to high-dimensional data sets such as molecular simulation data.

We clustered each of the test cases with all three geometric cluster algorithms (neighbor algorithm, K-medoids algorithm, and common-nearest-neighbor algorithm) and also systematically varied the input parameters. By visual inspection of the resulting 2D scatterplots, we decided whether the test sets had been clustered correctly, i.e., according to human intuition. In the following, we will refer to those groups of data points that are recognized as clusters by the human intuition as "data point heap," whereas the word "cluster" will denote the results of the respective cluster algorithms.

All three geometric cluster algorithms succeed in clustering test cases 1 and 2. For test case 3, the K-medoids and the common-nearest-neighbor algorithms converge to a correct solution, whereas for test cases with concave clusters (test cases 4 and 5), only the common-nearest neighbor algorithm partitions the data set correctly. Note, however, that the results of the K-medoids algorithm depend not only on the input parameter k (number of clusters) but also on the initialization (first assignment of data points to clusters), that is, different runs with the same value of k can and will lead to different partitions of the data set. Such a situation arose for test case 2 with $k=5$ and test case 3 with $k=2$. (All other values of k led to wrong partitions of the two data sets.) The common-nearest-neighbor algorithm is the only algorithm that partitions all five test cases correctly. It is also robust with respect to a variation in its two input parameters: nndc and nnnc. Table II illustrates this: independent of the test case, we can vary nndc largely and find at least one value of the nnnc for which the data set is clustered correctly.

Figure 1 also highlights the peculiarities and deficiencies of each of the algorithms. In the neighbor algorithm, the data point which has the most neighbors within a certain cutoff radius is considered the center of the next cluster. Once this cluster center is set, no correction is possible and all neighboring data points are assigned to this cluster. If two data-point heaps are not well separated or if their distance is smaller than the respective elongation (as in test case 3), the cluster center can very well be assigned to a data point that lies in between the two data point heaps and the resulting cluster will comprise data points from both. The characteristic structure of this "cutting effect" is a large circular cluster, such as the black cluster in panel (7) of Fig. 1, the borders of which do not correspond to the limits of the data-point heaps.

Panel (5) in the same figure shows a result of the K-medoids algorithm for the second test case. This algorithm starts by randomly choosing k cluster centers from the data set and then iteratively optimizes cluster memberships and cluster centers until convergence. Convergence is usually reached after a few iterations. The algorithm, however, does not always converge to the intuitive result. Often we see results in which a large data-point heap is split into two or more parts, such as the central data-point heap in panel (5) of Fig. 1, or small data-point heaps are merged into one cluster, such as the two data-point heaps on the left side of the same panel. This typically happens when during the initialization two cluster centers are assigned to data points which are in the same (large) data-point heap. If the data-point density more or less steadily decreases from the center of the data-point heap to its rims, such as in the first test case we often see that data points at the rim of the heap are split off as singletons [dots with various colors in panel (3) of Fig. 1]. This effect is not severe for a 2D data set as only a few data points are split off, but for higher-dimensional data sets, the ratio between the number of data points on the rim of a data-point heap and those at the center increases, which makes this effect pronounced.

B. Geometric clustering results for a β-heptapeptide

1. Structural RMSD value distributions

In geometric cluster analysis, the RMSD matrix has to meet two conditions in order to be a faithful representation of the conformational ensemble of the molecule under study: (i) the structures that were used for the construction of the matrix have to be uncorrelated and (ii) there have to be enough structures to cover the entire conformational space with correct weights. We satisfy the first condition by only using structures that are separated by 0.1 ns in the trajectory of the β-heptapeptide (Fig. 2) and we test the second condition by constructing two RMSD matrices with the same parameters (i.e., atom set and number of structures) from independent simulations and then comparing the distribution of RMSD values within these matrices. Figure 3 shows the distribution of RMSD matrix elements resulting from using 5000 (row 1), 10 000 (row 2), 15 000 (row 3), 20 000 (row 4), and 25 000 (row 5) molecular structures calculated using the three different atom sets: *aa* (column 1), bb_{1-7} (column 2), and bb_{2-6} (column 3). Each of the graphs shows two RMSD distributions for which the structures were drawn from independent simulations (5000 structures: simulations 1 and 10; 10 000 structures: simulations 1, 2, 11, and 12; 15 000 structures: simulations 1–3 and 11–13; 20 000 structures: simulations 1–4 and 11–14; and 25 000 structures: simulations 1–5 and 11–15). For 5000 structures, the general features of the RMSD distributions, i.e., number and position of the peaks, are similar for each of the atom sets. However, the relative weights of the peaks differ greatly for RMSD matrices that were constructed from different simulations, indicating that the various parts of conformational space have not yet been sampled with equilibrium weights. The two distributions become more similar as we add more structures, but even for 25 000 structures complete agreement is not achieved. Since the similarity of the two distributions does not significantly improve beyond 15 000 structures, we decided to use RMSD matrices with 15 000 structures for geometric clustering in this study. The fact that the two RMSD distributions have approximately the same shape is not a sufficient proof that the conformational space has been sampled completely. Rather, one should regard this as a necessary condition or an indication of complete sampling. In practice, however, the available simulation data often limits the number of structures that can be used for the construction of RMSD matrices to a few thousand even for considerably larger systems,[15,29] which almost certainly imposes a large uncertainty in the calculated cluster sizes.

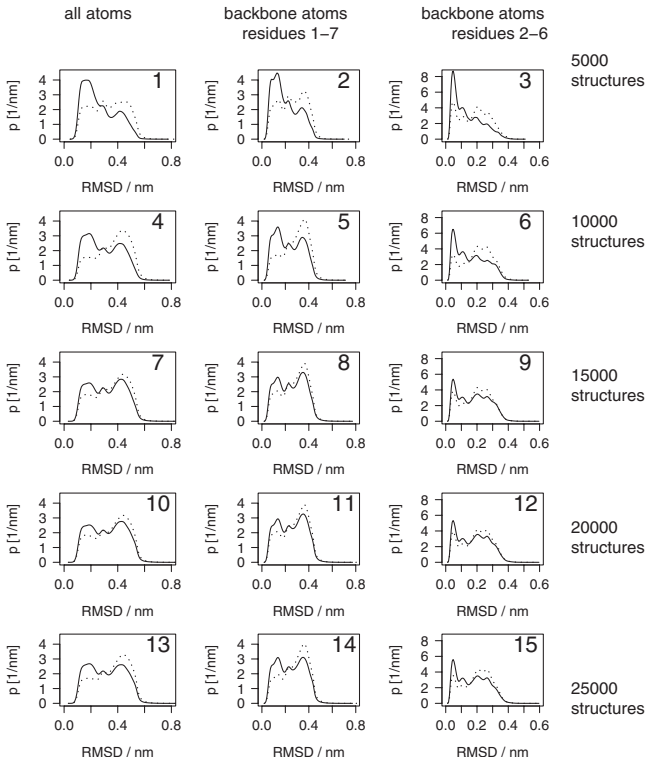

FIG. 3. Distribution of RMSD values in RMSD matrices of 5000 data points (row 1), 10 000 data points (row 2), 15 000 data points (row 3), 20 000 data points (row 4), 25 000 data points (row 5) calculated using different atom sets: *aa* (column 1), bb_{1-7} (column 2), bb_{2-6} (column 3), and data from 11 different 500 ns simulations of the β-heptapeptide; *solid line:* data points drawn from simulations 1 (row 1), 1 and 2, (row 2), 1–3 (row 3), 1–4 (row 4), and 1–5 (row 5); *dotted line:* data points drawn from simulations 10 (row 1), 11 and 12, (row 2), 11–13 (row 3), 11–14 (row 4), and 11–15 (row 5).

The RMSD distributions of the three different atom sets in Fig. 3 reflect two known effects: (i) the RMSD values increase if more atoms are added to the sets and (ii) if the very mobile side chains and terminal residues are included in the atom set, their large displacements dominate the RMSD values leading to a less structured RMSD distribution.

2. Choice of cluster parameters

The question of how to choose appropriate cluster parameters has to be answered differently for each of the algorithms. For the neighbor-cluster algorithm, the distance cutoff c represents the radius of a "typical" cluster within a given data set and can be estimated in two ways.

- The time series of the RMSD to the folded (NMR or x ray) structure shows, for a folding-unfolding equilibrium such as exhibited by β-heptapeptides, a gap between folded and unfolded structures. RMSD values that lie in this gap are a good estimate for radius of the folded state and can be used as a distance cutoff.
- The first minimum in the RMSD distribution (see Fig. 3) is an equally good estimate for this radius (and hence for the distance cutoff) and can be determined more precisely.

The common-nearest-neighbor algorithm has two parameters: a distance cutoff, nndc, and a number cutoff, nnnc. Unlike in the neighbor algorithm, the distance cutoff does not represent a cluster radius, but the very small volume around a given data point, in which its nearest neighbors can be found. It, therefore, should be smaller than the location of the first peak in the RMSD distribution (Fig. 3) and, second, large enough that it lies in an area where the RMSD distribution differs significantly from zero. Once the value of nndc is set, the optimal value of nnnc can be chosen as follows. Figure 4 shows the effect of varying nnnc on the number and on the size of large clusters (>100 members). We used the bb_{2-6}-RMSD matrix as dissimilarity matrix and set nndc to 0.038 nm. The value of nnnc was varied from 2 to 20. The first panel shows the number of large clusters as a function of nnnc. The number of large clusters grows stepwise until it reaches a plateau at 5, after which it decreases to 4 (and later on to smaller values and even 0; data not shown). The stepwise increase in the number of large clusters is related to jumps in the size of the five largest clusters. nndc and nnnc roughly define a data-point density (nnnc data points/hypersphere with radius nndc) that acts as limiting density separating highly populated areas in the data set from each other. That is, if two highly populated areas are connected by a higher data-point density, they are merged into one cluster, if not, each of them forms its own cluster. In other words, the higher the limiting density, the higher the resulting number of clusters. Consequently, for a low limiting density as defined by nnnc=2, the algorithm returns only two large clusters. As we increase nnnc, originally large clusters are split into smaller clusters. This happens the first time at nnnc=4 when the second cluster drops in size by about 700 members and a third large cluster with 610 members arises. The number of large clusters then stays constant until nnnc=7 when this third cluster is split into a cluster with 394 members and a new large cluster with 150 members. The maximum number of large clusters is reached for nnnc=9. Here, the size of the third cluster decreases by about 200 members and at the same time the last large cluster with about 150 members appears and the fourth cluster increases in size by about 50 members. After the maximum number of large clusters is reached, the clusters continuously decrease in size until the smallest of them eventually has less then 100 members and is no longer counted as a large cluster (nnnc=14). This is most likely due to the splitting effect we described in panel (3) of Fig. 1 as the density that is needed to merge a given data point with an existing cluster increases, more an more data points are split off as singletons from the original cluster. Between nnnc=2 and nnnc=14, each of the cluster results is a valid division of the data set. However, the resolution increases with increasing nnnc and we recommend using a parameter combination for which the maximum number of clusters is obtained, e.g., nndc=0.038 nm and nnnc=10.

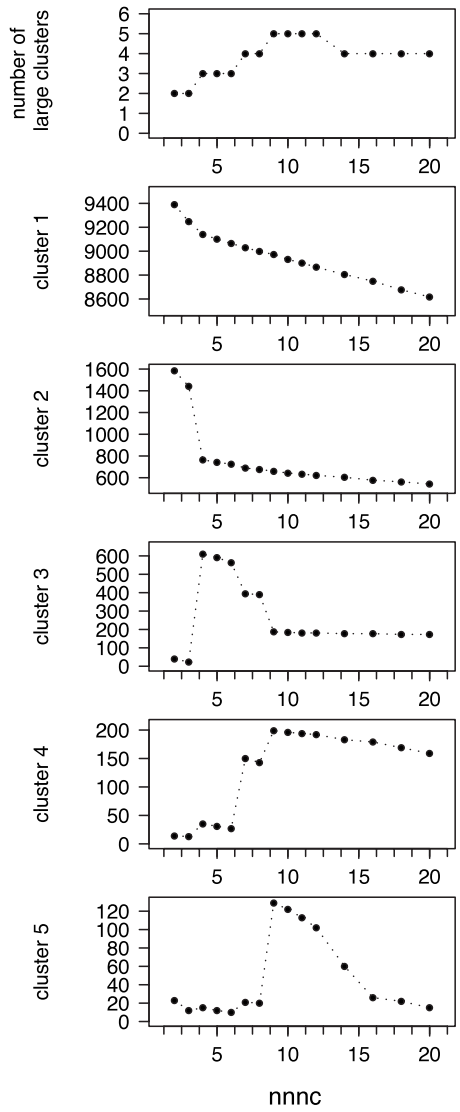

FIG. 4. Partitioning of the bb_{2-6}-RMSD matrix using the common-nearest-neighbor algorithm, change in the number of large clusters (>100 members), and the cluster sizes of clusters 1–5 with the value of nnnc, where nndc was set to 0.038 nm.

Two points are still noteworthy. First, since the algorithm itself is not expensive—the majority of the computational time is spent on reading in the RMSD matrix—this analysis can be performed on a routine basis. Second, the analysis itself already yields a lot of information about the data set. Note that the largest cluster slowly decreases in size by splitting off small clusters and singletons, while clusters 3–5 are obtained by splitting the second cluster into large ones. The number of data points that do not fall into any of the large clusters, about one-third of the entire data set, stays approximately constant. Obviously, we are dealing with a data set, in which about 60% of all data points fall into a quite homogenous cluster (cluster 1), about 30% are scat-

TABLE III. Sizes of the large clusters (>100 members) returned by the neighbor algorithm in percent of the complete data set (15 000 structures): *first row*, atom set; *second row* ,*n*=cluster number, number of the partition used in the text, and in other tables, *c*=distance cutoff in nm; *body*, cluster sizes; *last row*, s.c.=number of data points in small clusters, i.e., clusters with ≤100 members.

		All atom (*aa*)			Backbone residues 1–7 (bb_{1-7})			Backbone residues 2–6 (bb_{2-6})		
		I	II	III	I	II	III	I	II	III
n	*c*	0.22	0.24	0.26	0.14	0.16	0.18	0.08	0.10	0.12
1		59.53	61.99	65.63	54.35	58.30	61.2	51.77	59.85	63.54
2		7.72	8.33	7.25	6.75	7.63	9.127	6.46	7.03	8.99
3		4.15	3.06	3.53	3.73	3.89	3.247	5.02	4.25	4.78
4		2.06	2.37	2.32	3.11	2.78	3.06	3.15	3.11	1.98
5		1.18	1.41	1.68	1.90	1.31	1.29	2.69	1.43	1.84
6		0.91	1.09	1.47	1.15	1.10	1.26	1.46	1.29	1.19
7		0.91	1.07	1.14	1.04	1.05	1.12	0.87	0.93	1.19
8		0.85	0.93	0.93	0.71	0.87	1.00	0.83	0.89	1.09
9			0.86	0.83		0.86	0.87	0.75	0.81	0.99
10			0.85	0.75		0.73	0.86		0.78	0.95
11			0.77	0.73		0.70	0.74		0.70	0.85
12			0.73	0.73			0.69		0.68	0.71
13							0.69			0.68
s.c.		22.70	16.56	13.01	27.25	20.7	14.78	27.00	17.57	11.20

tered around the conformational space without forming clusters (unstructured data), and the remaining 10% are spread among 4 large clusters.

The K-medoids-cluster algorithm takes the number of clusters as an input parameter. Unfortunately, usually nothing is known about the optimal number of clusters unless one already has results from other cluster algorithms. Since both the neighbor-cluster and the common-nearest-neighbor-cluster algorithms indicated that there are only few dominant clusters, we varied the input parameter from $k=2$ to $k=9$ in steps of 1 (data not shown) and present results for $k=5$ in the remainder of this publication.

3. Cluster sizes

We clustered RMSD matrices that were constructed for three different atom sets (*aa*, bb_{1-7}, and bb_{2-6}) using three different geometric cluster algorithms. For each combination of RMSD matrix and cluster algorithm, we conducted the analysis three times with slightly varied input parameters.

Tables III–V show the results in terms of cluster sizes of large clusters (in percentage of the complete data set), where we define a large cluster as a cluster with more than 100 members. The numbers in these three tables give us a first characterization of the data set and it is most interesting to note that this characterization does not vary strongly when the cluster parameters or even the underlying atom set is varied, but rather depends strongly on the applied cluster algorithm. The neighbor algorithm (Table III) partitions our data set into one very large cluster comprising of ≈52%–66% of all data points, followed by a second cluster with ≈6%–9% of all data points, and then clusters of continuously decreasing size. For all three RMSD matrices, it finds between 8 and 13 large clusters and ≈11%–27% of the data points fall into small clusters, i.e., clusters with 100 members or less. Because the clusters returned by this algorithm typically continuously decrease in size, the distinction between large and small clusters is arbitrary.

The K-medoids algorithm (Table IV) divides the data set into approximately uniformly sized clusters and all of the cluster sizes are well above 100, making a distinction between small and large clusters unnecessary. We find significant differences between the various atom sets. Moreover, the results also depend on the initialization, e.g., for the *aa*-atom set, runs 1 and 2 yield comparable cluster sizes, which, however, differ from those of run 3.

The common-nearest-neighbor algorithm yields the most constant results, with respect to variation in the cluster parameters and the underlying atom set and cluster sizes (see

TABLE IV. Sizes of the large clusters (>100 members) returned by the K-medoids algorithm in percent of the complete data set (15 000 structures): *first row*, atom set; *second row*, *n*=cluster number, number of the partition used in the text and in other tables, number of the initialization; and *body*, cluster sizes.

	All atom (*aa*)			Backbone residues 1–7 (bb_{1-7})			Backbone residues 2–6 (bb_{2-6})		
	I	II	III	I	II	III	I	II	III
n	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3
1	27.56	27.42	26.06	27.70	46.45	34.81	50.71	49.39	29.59
2	26.93	27.24	20.11	26.69	21.01	31.61	18.15	16.99	22.37
3	22.63	22.73	18.47	16.93	12.67	15.83	11.03	15.15	18.41
4	12.91	11.64	17.96	15.32	11.16	9.71	10.69	11.96	17.60
5	9.98	10.97	17.40	13.36	8.71	8.05	9.41	6.51	12.03

TABLE V. Sizes of the large clusters (>100 members) returned by the common-nearest-neighbor algorithm in percent of the complete data set (15 000 structures): *first row*, atom set; *second row*, n=cluster number, number of the partition used in the text and in other tables, nndc=nearest-neighbor-distance cutoff in nm, nnnc=nearest-neighbor-number cutoff; *body*, cluster sizes; and *last row*, s.c.=number of data points in small clusters, i.e., clusters with ≤100 members.

		All atom (aa)			Backbone residues 1–7 (bb_{1-7})			Backbone residues 2–6 (bb_{2-6})		
		I	II	III	I	II	III	I	II	III
	nndc	0.09	0.10	0.11	0.06	0.07	0.08	0.036	0.038	0.04
n	nnnc	2	3	6	6	12	20	4	10	10
1		52.92	54.96	56.17	55.25	57.37	59.09	60.43	59.54	60.27
2		3.51	3.97	4.32	4.07	4.52	4.87	4.83	4.29	4.61
3		2.30	2.47	2.61	2.39	2.63	2.88	2.74	1.31	3.61
4		1.25	1.41	1.45	1.39	1.46	1.55	1.17	1.23	
5									0.81	
s.c.		39.91	37.10	35.22	36.75	33.96	31.61	30.66	32.83	31.37

Table V). Similar to the neighbor algorithm, it finds one large cluster covering ≈53%–60% of the data points, all other clusters are smaller by at least an order of magnitude with the second cluster covering about 4%–5% of the data points. The algorithm finds between 3 and 5 large clusters and sorts ≈31%–40% of the data points into small clusters. Contrary to the neighbor algorithm, the 100-member-limit marks a gap in the cluster sizes inherent to this algorithm: large clusters are well above this limit and the vast majority of the small clusters are singletons with only few small clusters comprising of 5–30 members.

4. Variation in the cluster parameters and the underlying atom set

When comparing two alternative partitions $A=\{a_1,a_2,\ldots\}$ and $B=\{b_1,b_2,\ldots\}$ of a given data set, similar cluster sizes are only a hint that these two partitions might be similar, but do not constitute a sufficient proof. To assess how similar the two partitions are, one needs to know how large the overlap between any pair of clusters a_i,b_j is, i.e., how many of the data points in a_i are also found in b_j. In Tables VI and VII we report this type of overlap numbers for various partitions of our data set. We consider only the five largest clusters of each partition and again sort them by size and arrange the overlap numbers in a matrix (three matrices per panel). If two partitions are very similar, one would expect numbers in the order of the corresponding cluster sizes on the diagonal elements and mostly zeros or small entries in the off diagonal elements. Conversely, if two partitions are very different, one expects large entries on both the diagonal and off-diagonal elements.

Table VI assesses the influence of a slight variation in the parameters on the partition of a given data set. Here, we only report for each algorithm [panels (1)–(3)] the overlap matrices for different partitions of the bb_{2-6}-RMSD matrix, but the overlap matrices of the other atom sets yield a similar picture. For each of the three distance cutoffs: (I) $c=0.08$ nm, (II) $c=0.10$ nm, and (III) $c=0.12$ nm, the neighbor algorithm identifies one dominant cluster, all of which have a large overlap with each other. More precisely, the size of this first cluster is directly linked to the distance cutoff c: the relatively small cluster of I is a subset of cluster 1 in II as well as in III. Likewise, cluster 1 in II is a subset cluster 1 in III. Apart from that, clear matches between clusters of different partitions are rare. More often one finds that a cluster in one partition is a subset of a larger cluster in another partition, e.g., cluster 3 in I is a subset of both cluster 2 in II and cluster 2 in III.

In the K-medoids algorithm we did not vary the cluster parameter k but rather the initialization and present the results for $k=5$. Runs 1 (I) and 2 (II) yield similar cluster sizes and also the overlap between the respective clusters 1 and 2 is large. Despite the fact that both clusters 1 and 2 in I also have overlap with cluster 5 from II, the overlap is so large that we can safely say that clusters 1 and 2 in I are the same as clusters 1 and 2 in II. The overlap pattern for clusters 3—5 is more complicated and no clear match is possible. Run 3 (III) yields cluster sizes that were quite different from those in I and II, yet we can still match some of its clusters to the partitions I and II: clusters 1 and 2 in III together constitute cluster 1 in I or II, also cluster 4 in III is largely identical to cluster 2 in I or II. Clusters 3 and 5 cannot be matched. One should, however, also note that these overlaps are not as clear as in the comparison of I and II.

The common-nearest-neighbor algorithm is the most robust of the three geometric cluster algorithms with respect to a variation in the input parameters. Clusters 1, 2, and 4 in I (nndc=0.036 nm, nnnc=4) are largely identical to clusters 1, 2, and 5 in II (nndc=0.038 nm, nnnc=10). In the comparison of I and III (nndc=0.040 nm, nnnc=10), again clusters 1 and 2 in I match clusters 1 and 2 in III, whereas clusters 3 and 4 in I together constitute cluster 3 in III. The overlap of II and III yields as similar picture: again the respective clusters 1 and 2 match and cluster 3 in III is split into clusters 3–5 in II.

In Table VII we test the sensitivity of the neighbor algorithm and the common-nearest-neighbor algorithm with respect to the variation in the underlying atom set. In the neighbor algorithm, the largest clusters for all atom sets show the largest overlap among each other meaning that clusters 1 in aa, bb_{1-7}, and bb_{2-6}, respectively, are largely identical. Also the second clusters for all three atom sets still show about 80% overlap among each other. Clusters 3–5 in aa cannot be clearly matched to any of the clusters in bb_{1-7}

TABLE VI. Variation in the cluster parameters (the initializations for different partitions of the bb_{2-6}-RMSD matrix): *first row and column of each subtable*, cluster number; *second row and column in each subtables*, cluster size; *body of each subtable*, overlap matrix of the first five large clusters; and *last row and column of each subtable*, s.c.=number of data points in small clusters, i.e., clusters with ≤100 members and their overlap with other clusters. *Neighbor algorithm:* (I) c=0.08 nm, (II) c=0.10 nm, and (III) c=0.12 nm. *K-medoids algorithm:* (I) run 1, (II) run 2, and (III) run 3. *Common-nearest-neighbor algorithm:* (I) nndc=0.036 nm, nnnc=4; (II) nndc=0.038 nm, nnnc=10; and (III) nndc=0.04 nm, nnnc=10.

Neighbor algorithm

	II	1	2	3	4	5	s.c.		III	1	2	3	4	5	s.c.		III	1	2	3	4	5	s.c.
I	Size	8978	1054	638	467	215	3648	I	Size	9531	1348	717	297	276	2831	II	Size	9531	1348	717	297	276	2831
1	7765	7765	0	0	0	0	0	1	7765	7765	0	0	0	0	0	1	8978	8976	0	0	0	0	2
2	969	678	0	0	281	0	10	2	969	942	0	0	0	0	27	2	1054	5	1018	0	0	0	31
3	753	0	751	0	0	0	2	3	753	0	753	0	0	0	0	3	638	2	61	575	0	0	0
4	473	0	0	471	0	0	2	4	473	0	12	461	0	0	0	4	467	364	0	4	0	0	99
5	403	344	0	0	0	0	59	5	403	375	0	0	0	0	28	5	215	0	130	3	0	17	65
s.c.	4637	191	303	167	186	215	3579	s.c.	4637	449	583	256	297	276	2776	s.c.	3648	191	139	135	297	17	2634

K-medoids algorithm, k=5

	II	1	2	3	4	5	s.c.		III	1	2	3	4	5	s.c.		III	1	2	3	4	5	s.c.
I	Size	7408	2548	2273	1794	977	⋯	I	Size	4439	3355	2761	2640	1805	⋯	II	Size	4439	3355	2761	2640	1805	⋯
1	7607	7396	0	0	0	211	⋯	1	7607	4341	3253	2	4	7	⋯	1	7408	4143	3259	2	4	0	⋯
2	2723	0	2504	2	0	217	⋯	2	2723	40	51	23	2600	9	⋯	2	2548	5	51	30	2462	0	⋯
3	1655	12	2	180	1275	186	⋯	3	1655	49	51	1057	1	497	⋯	3	2273	0	0	1785	9	479	⋯
4	1603	0	1	742	516	344	⋯	4	1603	9	0	301	1	1292	⋯	4	1794	36	41	884	0	833	⋯
5	1412	0	41	1349	3	19	⋯	5	1412	0	0	1378	34	0	⋯	5	977	255	4	60	165	493	⋯
s.c.	⋯	⋯	⋯	⋯	⋯	⋯	⋯	s.c.	⋯	⋯	⋯	⋯	⋯	⋯	⋯	s.c.	⋯	⋯	⋯	⋯	⋯	⋯	⋯

Common-nearest-neighbor algorithm

	II	1	2	3	4	5	s.c.		III	1	2	3	4	5	s.c.		III	1	2	3	4	5	s.c.
I	Size	8931	643	196	184	122	4924	I	Size	9040	691	542	⋯	⋯	4727	II	Size	9040	691	542	⋯	⋯	4727
1	9065	8929	0	0	0	0	136	1	9065	9011	0	0	⋯	⋯	54	1	8931	8931	0	0	⋯	⋯	0
2	725	0	641	0	0	0	84	2	725	0	679	0	⋯	⋯	46	2	643	0	643	0	⋯	⋯	0
3	411	0	0	196	184	0	31	3	411	0	0	391	⋯	⋯	20	3	196	0	0	196	⋯	⋯	0
4	175	0	0	0	0	121	54	4	175	0	0	148	⋯	⋯	27	4	184	0	0	184	⋯	⋯	0
5	⋯	⋯	⋯	⋯	⋯	⋯	⋯	5	⋯	⋯	⋯	⋯	⋯	⋯	⋯	5	122	0	0	122	⋯	⋯	0
s.c.	4624	2	2	0	0	1	4619	s.c.	4624	29	12	3	⋯	⋯	4580	s.c.	4924	109	48	40	⋯	⋯	4727

or bb_{2-6}. The difference between bb_{1-7} and bb_{2-6} is smaller: here, also the third clusters are largely identical. Despite the fact that the kernels of the largest clusters were not affected, part of their members was assigned to other clusters when the atom set changed. This is reflected by the large off-diagonal elements in the overlap matrices. For example, in the first row of the overlap matrix between *aa* and bb_{2-6}, one sees that 119 and 264 members of the largest cluster in *aa* have been assigned to clusters 2 and 4 in bb_{2-6}, respectively. This shows that the borders of cluster 1 in one atom set might cut through clusters 2, 3, or 4 in another atom set.

The picture is different for the common-nearest-neighbor algorithm. Here, the cluster definition seems to be hardly influenced by the choice of the atom set. The overlap between two clusters in different partitions is either in the order of the cluster size or zero. The difference between cluster size and overlap is covered by small clusters (last row and column in each matrix). One can speculate on the reason why this algorithm is so robust with respect to the variation in the atom set: in principle, it tries to cut along the minima of the distribution, thereby identifying its maxima, which, in turn, correspond to the minima of the free-energy surface. Due to sampling, one typically does not analyze the distribution in the complete conformational space but only its projection onto the conformational subspace of the chosen atom set. Minima that are present in the complete distribution might be blurred or even absent in its projection; however, the converse—minima that are present in the projection but are less pronounced in the complete distribution—is not possible. Therefore, if the atom set bb_{2-6} suffices to faithfully represent the essential barriers in the free-energy landscape, adding atoms that move over large distances but are essentially unhindered, such as side chains or the terminal residues, will not change the cluster results. Using cluster algorithms that define clusters based on some distance to a cluster center, such as the neighbor or the K-medoids algorithms, adding highly mobile atoms may obscure the cluster boundaries. However, note that these algorithms rely on the assumption that all clusters have an approximately spherical shape and are separated by distances larger than their diameter and here a projection can help fulfill these assumptions.

C. Kinetic clustering results for the β-heptapeptide

We have discretized the conformational space of the β-heptapeptide (cf. Fig. 2) into microstates and—using ki-

TABLE VII. Variation in the atom set: *first row and column of each subtable*, cluster number; *second row and column in each subtables*, cluster size; *body of each subtable*, overlap matrix of the first five large clusters; and *last row and column of each subtable*, s.c.=number of data points in small clusters, i.e., clusters with ≤100 members and their overlap with other clusters. *Neighbor algorithm: aa*: c=0.24 nm, bb_{1-7}: c=0.16 nm, bb_{2-6}: c=0.10 nm. *Common-nearest-neighbor algorithm*: *aa*: nndc=0.10 nm, nnnc=3, bb_{1-7}: nndc=0.07 nm, nnnc=12, bb_{2-6}: nndc=0.038 nm, nnnc=10.

Neighbor algorithm

	bb_{1-7}	1	2	3	4	5	s.c.		bb_{2-6}	1	2	3	4	5	s.c.		bb_{2-6}	1	2	3	4	5	s.c.
aa	Size	8757	1144	583	417	197	3902	*aa*	Size	8978	1054	638	467	215	3648	bb_{1-7}	Size	8978	1054	638	467	215	3648
1	9298	8724	92	0	256	31	195	1	9298	8767	119	9	264	0	139	1	8757	8572	22	0	72	0	91
2	1249	0	922	227	0	1	99	2	1249	0	836	302	0	48	63	2	1144	0	981	45	1	33	84
3	459	0	0	324	0	0	135	3	459	0	0	284	0	22	153	3	583	1	0	514	0	2	66
4	355	1	0	0	102	129	123	4	355	113	0	0	76	0	166	4	417	134	0	7	244	0	32
5	211	0	125	0	0	0	86	5	211	0	81	0	0	9	121	5	197	121	0	0	8	0	68
s.c.	3428	32	5	32	59	36	3264	s.c.	3428	98	18	43	127	136	3006	s.c.	3902	150	51	72	142	180	3307

Common-nearest-neighbor algorithm

	bb_{1-7}	1	2	3	4	5	s.c.		bb_{2-6}	1	2	3	4	5	s.c.		bb_{2-6}	1	2	3	4	5	s.c.
aa	Size	8606	678	395	219	⋯	5102	*aa*	Size	8931	643	196	184	122	4924	bb_{1-7}	Size	8931	643	196	184	122	4924
1	8244	8192	0	0	0	⋯	52	1	8244	8165	0	0	0	0	79	1	8606	8512	0	0	0	0	94
2	595	0	577	0	0	⋯	18	2	595	0	545	0	0	0	50	2	678	0	595	0	0	0	83
3	371	0	0	355	0	⋯	16	3	371	0	0	0	176	113	82	3	395	0	0	0	178	114	103
4	212	0	0	0	203	⋯	9	4	212	0	0	180	0	0	32	4	219	0	0	185	0	0	34
5	⋯	⋯	⋯	⋯	⋯	⋯	⋯	5	⋯	⋯	⋯	⋯	⋯	⋯	⋯	5	⋯	⋯	⋯	⋯	⋯	⋯	⋯
s.c.	5578	414	101	40	16	⋯	5007	s.c.	5578	766	98	16	8	9	4681	s.c.	5102	419	48	11	6	8	4610

netic clustering—have sorted these microstates into five metastable states. The remaining microstates represent a part of the conformational space, which is not characterized by clear minima and barriers but along which very diffusive dynamics occur. We classified structures that correspond to these microstates as "unstructured data." In order to be able to compare the kinetic cluster results to the ones from the geometric clustering, we assigned each structure from our test set of 15 000 structures to its microstate and then sorted them into the corresponding metastable state or to the group of unstructured data. The second line in Table VIII shows the partition of the data set into metastable states and unstructured data. Almost 60% of the data points are assigned to metastable state 5 which represents the folded state. Three out of the four remaining metastable states contain on the order of 1000 members. Finally, metastable state 2 is with 114 members very small. It is interesting to note that about 20% of the data is classified as unstructured, which is in the same order of magnitude as the portion of unstructured data identified by the neighbor and the common-nearest-neighbor algorithm.

1. Comparison of the geometric to the kinetic cluster results

Table VIII shows the overlap of the results obtained by geometric cluster algorithms with the metastable states obtained by kinetic clustering for the bb_{2-6}-RMSD matrix.

For the neighbor algorithm, there is a large overlap between metastable state 5 (folded state) and cluster 1: about 96% of the data points in metastable state 5 are assigned to cluster 1. Also despite the fact that 4.5% of the data points in metastable state 5 are assigned to cluster 4, and cluster 1 also has some overlap with metastable state 1, we can safely claim that cluster 1 represents the folded state. Furthermore, there is an approximate correspondence between metastable state 4 and cluster 2. 85% of all data points in metastable state 4 are assigned to cluster 2. Cluster 2, however, also has considerable overlap with metastable state 3, meaning that the neighbor algorithm does not accurately resolve the barrier between metastable states 3 and 4. Metastable state 3 has overlap with clusters 2, 3, and 5 and one may argue that this metastable state is essentially split into clusters 3 and 5 with

TABLE VIII. Comparison of algorithms using the atom set backbone, residues 2–6: (I) kinetic clustering; (II) neighbor algorithm, c=0.10 nm; (III) K-medoids, k=5, run 2; and (IV) common-nearest-neighbor algorithm, nndc=0.038 nm, nnnc=10.

	I	1	2	3	4	5	s.c.		I	1	2	3	4	5	s.c.		I	1	2	3	4	5	s.c.
II	Size	1098	114	1164	847	8719	3058	III	Size	1098	114	1164	847	8719	3058	IV	Size	1098	114	1164	847	8719	3058
1	8978	337	0	0	11	8404	226	1	7408	371	46	0	39	6719	233	1	8931	313	0	2	3	8469	144
2	1054	2	0	234	722	0	96	2	977	260	25	22	2	1	667	2	643	0	0	2	619	0	22
3	638	0	0	554	7	3	74	3	2273	175	6	893	12	4	1183	3	184	0	0	176	1	0	7
4	467	92	0	8	3	214	150	4	2548	155	32	25	7	1994	335	4	196	0	0	192	0	0	4
5	215	6	0	139	0	0	70	5	1794	137	5	224	787	1	640	5	122	0	0	122	0	0	0
s.c.	3648	661	114	229	104	98	2442	s.c.	⋯	⋯	⋯	⋯	⋯	⋯	⋯	s.c.	4924	785	114	670	224	250	2881

some contribution from cluster 2. Metastable state 2 is not recognized by the neighbor algorithm but all its data points are characterized as unstructured data. Similarly, metastable state 1 has some overlap with clusters 1 and 4, but the majority of its data points is characterized as unstructured data. Note that it is quite possible that metastable states are split into several clusters by a sensitive geometric cluster algorithm because a metastable state can consist of several minima which are separated by low energy barriers and which, therefore, are not resolved by the kinetic cluster algorithm. If, on the other hand, a geometric cluster covers two or more metastable states, the geometric cluster algorithm did not succeed in recognizing the large energy barrier separating these states and the clusters do not properly reflect the metastable states.

The overlap pattern for the K-medoids algorithm is a lot more crowded and no obvious match between its clusters and the metastable states can be found. The folded state, metastable state 5, has the largest overlap with cluster 1 but also significant overlap with cluster 4. On the other hand, cluster 1 has significant overlap with metastable state 1 and the unstructured data and some overlap with metastable states 2 and 4. Nevertheless, one can claim that cluster 1 and metastable state 5 approximately correspond to each other. The data points in Table IV are—for a large part—a subgroup of the data points in cluster 5. However, since cluster 5 has also large overlap with metastable states 1 and 3 and the unstructured data points, we cannot claim correspondence between metastable state 4 and cluster 5. For metastable states 1–3, we do not find any correspondence with the five clusters.

Of the three geometric cluster algorithms, the common-nearest-neighbor algorithm has the clearest overlap pattern with the metastable states. Its biggest cluster (cluster 1) corresponds to the folded state (metastable state 5): no other cluster has any overlap with this state and cluster 1 only has significant overlap with metastable state 1. Likewise cluster 2 and metastable state 4 are identical. Metastable state 3 has overlap with clusters 3–5, all of which have no overlap with any of the other metastable states. On could argue that metastable state 3 is split into three clusters. However, note that actually about half of the data points that are found in metastable state 3 are characterized as unstructured data by the nearest-neighbor algorithm. As with the neighbor algorithm, metastable state 2 is not recognized by the common-nearest-neighbor algorithm, instead all data points that belong to this state are characterized as unstructured data. This is possible if the data point density is very low in this state, which can either happen if the minimum is rather high in energy so that the overall probability of visiting it is low or if the minimum is very broad (entropic state) so that the (Boltzmann-weighted) fraction of data points that belong to this state are spread over a wide area of the conformational space. Metastable state 1 has some overlap with cluster 1 but is essentially not recognized. The same arguments as for metastable state 2 apply here. Note that the nearest-neighbor algorithm characterizes many data points that belong to metastable states as unstructured data, i.e., the overlap between the unstructured data of the nearest neighbor algorithm and the metastable states 1–5 is very large. This is most likely the same effect as we saw in the test cases: data points that lie at the rims of the metastable states where the data point density slowly decreases are split off as singletons by the nearest-neighbor algorithm.

V. CONCLUSION

In this contribution, we addressed the question: "To which extent do the results of geometric cluster algorithms when applied to molecular simulation data reflect the metastable states of a molecule?" To this end, we first compared and characterized three different geometric algorithms by applying them to 2D test data sets. Then, we tested their robustness with respect to the variation in their input parameters, including the underlying distance measure by applying them to a data set of 15 000 structures of a β-heptapeptide and comparing the cluster-overlap of the various results. Finally, we identified the metastable states of this β-heptapeptide using a kinetic cluster method and compared the overlap of these states with the geometric cluster results.

The test cases confirmed that geometric cluster algorithms, which base their cluster definition on the distance to a cluster center, such as the neighbor-cluster algorithm and the K-medoids-cluster algorithm, are generally not capable of identifying elongated or convex clusters. The common-nearest-neighbor algorithm, which bases its cluster definition on an estimate of the data-point density, however, correctly clustered all five test cases.

Additionally, we could show that the pattern of cluster sizes in a geometric cluster analysis is more dependent on the type of algorithm used for the clustering than on variations in the data set under study. The common-nearest-neighbor algorithm, for instance, clearly splits the data set into a large number of very small clusters and singletons, which we classified as unstructured data, and small number of rather large clusters. The neighbor algorithm shows a similar pattern, although, here, the clusters continuously decrease in size and the distinction between unstructured and structured data is not quite as obvious. The K-medoids algorithm, on the other hand, partitions a data set into k approximately uniformly sized clusters—none of which represents the group of unstructured data recovered by the former two algorithms.

Of the three geometric cluster algorithms, the common-nearest-neighbor algorithm is the most robust with respect to variation in the input parameters and variation in the distance measure. Its cluster definition is hardly affected by these changes, only the resolution changes: a large cluster in one partition of the data set can be split up into smaller cluster in another partition. In the neighbor algorithm, only the first and largest cluster, which represents the folded state, is reliably recognized independent of the distance cutoff c and the underlying atom set. For the K-medoids algorithm, we only presented results for various initializations, but not for a variation in the input parameter k or the underlying atom set because already for this change the cluster results differed considerably. Although the overlap pattern between the clusters of different initializations were rather complicated, the two largest clusters with $\approx$7400 and $\approx$2500 members

were—to a substantial degree—preserved. In one of the initializations, the largest cluster was, however, split into two clusters, which is in accordance with a trait of this algorithm as revealed by the test case.

Finally, we partitioned the data set into metastable states using a kinetic cluster algorithm and compared the results to the (geometric) clusters. Of all three geometric cluster algorithms the common-nearest-neighbor algorithm shows the clearest overlap pattern with the kinetic cluster results. It clearly recognized the folded state (metastable state 5), metastable state 4, and—to a lesser degree—metastable state 3. However, it characterized many data points which were assigned to metastable states by the kinetic cluster algorithm as unstructured data which could be explained by the "splitting effect" we observed in the test cases. The neighbor algorithm is capable of identifying the folded state and—to a certain extent—metastable states 4 and 3 but fails to cleanly separate the latter two states from each other. Similar to the common-nearest-neighbor algorithm, it does not recognize metastable states 1 and 2. For the K-medoids algorithm, none of the clusters could be matched unambiguously to the metastable state. The largest cluster, however, has significant overlap with the metastable state 5 so that it approximately corresponds to the folded state. In summary, if geometric cluster algorithms are to be used to identify metastable states, the results ought to be interpreted with caution since the overlap between the set of structures in a given metastable state and the set of structures in the corresponding geometric cluster is often only approximate. The data suggest avoiding the use of the K-medoids algorithm. The neighbor algorithm performs better by consistently identifying correctly the most populated cluster. Both algorithms do not provide a useful representation of the underlying dynamics. They can be used, though, as a tool for discretization of the conformational space, for example, for the construction of microstates or for a determination of overlap between two ensembles. The common-nearest-neighbor algorithm performs significantly better than the other two. Yet it does not yield a perfect representation of the underlying dynamics in the conformational space.

ACKNOWLEDGMENTS

Financial support by the National Centre of Competence in Research (NCCR) (Structural Biology) of the Swiss National Science Foundation (SNSF) is gratefully acknowledged. X.D. acknowledges funding from the Spanish MICINN/FEDER (Grant No. BIO2007-62954).

APPENDIX: PSEUDOCODE FOR THE GEOMETRIC CLUSTER ALGORITHMS

1. Neighbor algorithm

The neighbor algorithm has one input parameter—the distance cutoff c—and is composed of the following steps.

(1) Construct the list of nearest neighbors from the pool of data points that have not been assigned to a cluster yet.
(2) Loop over all data points in this pool.
- Find the data point with the highest number of neighbors within c.

(3) This data point is the medoid of the current cluster.
(4) Add all of its neighbors to the current cluster.
(5) Add the current cluster to the list of clusters and remove its members from the pool of unassigned data points
(6) Repeat steps (1)–(5) until all data points have been assigned to a cluster.

2. K-medoids algorithm

The K-medoids algorithm has one input parameter—the number of clusters k—and is composed of the following steps.

(1) Choose k data points randomly—they are the first medoids.
(2) Assign all other data points to the cluster whose medoid is closest.
(3) Recompute the medoids: in each cluster, set the data point which has the lowest distance to all other cluster members as the new medoid
(4) Recompute the cluster membership: Assign all other data points to the cluster whose medoid is closest.
(5) Repeat steps (3) and (4) until the cluster assignment does not change anymore or until a maximum number of iterations is reached.

[1] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford University Press, New York, 1989).
[2] O. M. Becker, Proteins **27**, 213 (1997).
[3] D. Chema and A. Goldblum, J. Chem. Inf. Comput. Sci. **43**, 208 (2003).
[4] X. Daura, W. F. van Gunsteren, and A. E. Mark, Proteins **34**, 269 (1999).
[5] A. Glättli, D. Seebach, and W. F. van Gunsteren, Helv. Chim. Acta **87**, 2487 (2004).
[6] X. Daura, B. Jaun, D. Seebach, W. F. van Gunsteren, and A. E. Mark, J. Mol. Biol. **280**, 925 (1998).
[7] D. Trzesniak, R. D. Lins, and W. F. van Gunsteren, Proteins: Struct., Funct., Bioinf. **65**, 136 (2006).
[8] J. D. Chodera, N. Singhal, V. S. Pande, K. A. Dill, and W. C. Swope, J. Chem. Phys. **126**, 155101 (2007).
[9] F. Noé, I. Horenko, C. Schütte, and J. C. Smith, J. Chem. Phys. **126**, 155102 (2007).
[10] S. Muff and A. Caflisch, Proteins: Struct., Funct., Bioinf. **70**, 1185 (2008).
[11] W. Huisinga, C. Best, R. Roitzsch, C. Schütte, and F. Cordes, J. Comput. Chem. **20**, 1760 (1999).
[12] P. Deuflhard, W. Huisinga, A. Fischer, and C. Schütte, Linear Algebr. Appl. **315**, 39 (2000).
[13] P. S. Shenkin and D. Q. McDonald, J. Comput. Chem. **15**, 899 (1994).
[14] G. M. Downs and J. M. Barnard, *Reviews in Computational Chemistry* (Wiley, New York, 2002), Vol. 18, pp. 1–40.
[15] J. Y. Shao, S. W. Tanner, N. Thompson, and T. E. Cheatham, J. Chem. Theory Comput. **3**, 2312 (2007).
[16] R. A. Jarvis and E. A. Patrick, IEEE Trans. Comput. **C-22**, 1025 (1973).
[17] See supplementary material at 10.1063/1.3301140 for an illustration of the transformation of a sample transition matrix $\mathbf{T}$ to the corresponding coarse-grained transition matrix $\mathbf{T}_{cg}$.
[18] W. C. Swope, J. W. Pitera, F. Suits, M. Pitman, M. Eleftheriou, B. G. Fitch, R. S. Germain, A. Rayshubski, T. J. C. Ward, Y. Zhestkov, and R. Zhou, J. Phys. Chem. B **108**, 6582 (2004).
[19] J. D. Chodera, W. C. Swope, J. W. Pitera, and K. A. Dill, Multiscale Model. Simul. **5**, 1214 (2006).
[20] C. H. Jensen, D. Nerukh, and R. C. Glen, J. Chem. Phys. **128**, 115107

(2008).
[21] S. P. Elmer, S. Park, and V. S. Pande, J. Chem. Phys. **123**, 114903 (2005).
[22] F. Cordes, C. Weber, and J. Schmidt-Ehrenberg, ZIB Report No. 02-40, 2002.
[23] N.-V. Buchete and G. Hummer, J. Phys. Chem. B **112**, 6057 (2008).
[24] W. C. Swope, J. W. Pitera, and F. Suits, J. Phys. Chem. B **108**, 6571 (2004).
[25] R. Boned, W. F. van Gunsteren, and X. Daura, Chem.-Eur. J. **14**, 5039 (2008).
[26] W. F. van Gunsteren, S. R. Billeter, A. A. Eising *et al.*, *Biomolecular Simulation: The GROMOS96 Manual and User Guide* (vdf Hochschulverlag AG an der ETH Zürich and BIOMOS b.v., Zürich, Groningen, 1996).
[27] J. P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, J. Comput. Phys. **23**, 327 (1977).
[28] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. Di Nola, and J. R. Haak, J. Chem. Phys. **81**, 3684 (1984).
[29] Y. Li, J. Chem. Inf. Model. **46**, 1742 (2006).