Voronota: A Fast and Reliable Tool for Computing the Vertices of the Voronoi Diagram of Atomic Balls

Kliment Olechnovič[a,b] and Česlovas Venclovas*[a]

The Voronoi diagram of balls, corresponding to atoms of van der Waals radii, is particularly well-suited for the analysis of three-dimensional structures of biological macromolecules. However, due to the shortage of practical algorithms and the corresponding software, simpler approaches are often used instead. Here, we present a simple and robust algorithm for computing the vertices of the Voronoi diagram of balls. The vertices of Voronoi cells correspond to the centers of the empty tangent spheres defined by quadruples of balls. The algorithm is implemented as an open-source software tool, Voronota. Large-scale tests show that Voronota is a fast and reliable tool for processing both experimentally determined and computationally modeled macromolecular structures. Voronota can be easily deployed and may be used for the development of various other structure analysis tools that utilize the Voronoi diagram of balls. Voronota is available at: http://www.ibt.lt/bioinformatics/voronota. © 2014 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.23538

Introduction

Biological macromolecules such as proteins and RNA typically function as complex three-dimensional (3D) shapes. These shapes are determined by the combined effect of interatomic interactions both within the macromolecule itself and with the environment (e.g., water or lipid bilayer). For comprehensive understanding of these interactions, it is essential to unambiguously identify all the neighbors of a given atom, to determine whether it is in contact with any of the neighboring atoms or with the environment, and how extensive these contacts are. The atomic neighborhood analysis can also be used for studying various geometric features of 3D structure including voids, pockets, and channels, for deriving molecular and solvent accessible surfaces and other geometric parameters. For these types of analyses, the Voronoi tessellation seems to be among the most suitable approaches.[1]

Voronoi diagram is named after Georgy Voronoi, who defined it back in 1908.[2] Given a set of points (centroids) in space, Voronoi diagram partitions the space into so-called Voronoi cells. The Voronoi cell may be considered as the volume "owned" by the centroid, because every point within the cell is closer to the centroid of the cell than to any other centroid. The Voronoi cell can be constructed as follows. Every line connecting a given centroid with other centroids is bisected by the plane perpendicular to that line. The smallest polyhedron formed around the centroid by such planes is termed the Voronoi cell (also known as the Voronoi region). Collectively, Voronoi cells corresponding to the set of points define the Voronoi tessellation, partitioning the space without any voids or overlaps. An important property of the Voronoi diagram is that every Voronoi cell has unambiguously defined neighbors without using any distance cutoffs.

However, the representation of protein or nucleic acids atoms as discrete points in many cases is an unacceptable oversimplification as it fails to reflect that different atoms have measurable volumes of different sizes. A more physically relevant representation of atoms is balls/spheres of van der Waals (VDW) radii and of molecules as unions of such balls. In such case, the Voronoi procedure for points (or balls of the same radii) has to be modified. Richards, who was the first to apply the Voronoi method to protein structures,[3] accounted for atomic diversity by introducing VDW radius-dependent weights for positioning the separating planes. Although this method became widely used, it has a serious drawback. Namely, the separating planes no longer intersect at common points resulting in some unallocated volume between the cells. One of the proposed solutions to this problem was to use radical plane as a separating plane between atomic balls.[4] This solution represents another weighted Voronoi scheme producing so-called Laguerre or power diagram. The advantage of the Laguerre diagram is that the cells all have flat faces making computations simpler. In addition, there is no unallocated volume in the resulting tessellation. The downside is that the weights assigned to two atoms are not directly proportional to the distance from each atom to the separating plane. This makes physical interpretation of the Laguerre tessellation problematic. Goede et al.[5] proposed a weighted Voronoi procedure resulting in a straightforward physical interpretation. In this procedure, the weights assigned to

[a] K. Olechnovič, Česlovas Venclovas
Institute of Biotechnology, Vilnius University, Graičiūno 8, Vilnius, LT-02241, Lithuania
E-mail: venclovas@ibt.lt
[b] K. Olechnovič
Faculty of Mathematics and Informatics, Vilnius University, Naugarduko 24, Vilnius, LT-03225, Lithuania
Contract grant sponsor: European Social Fund under the Global Grant measure

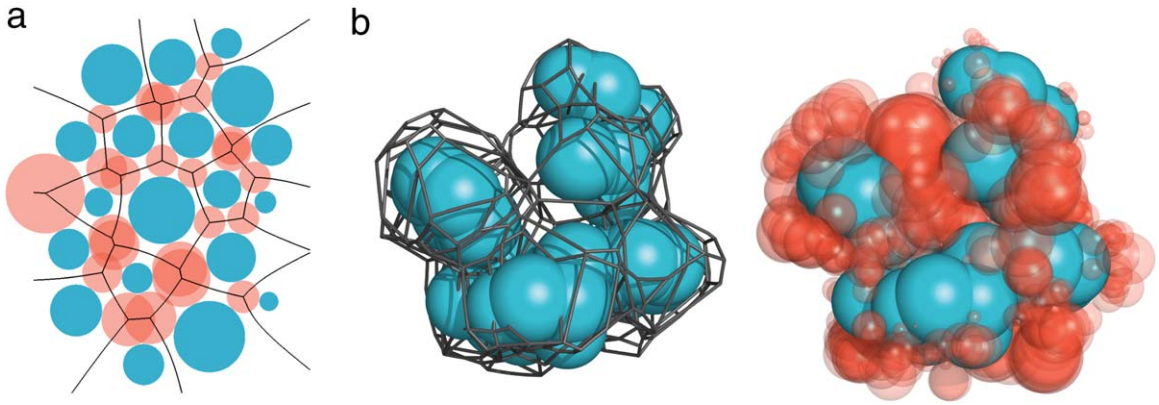

Figure 1. (a) Voronoi cells of two-dimensional (2D) balls (blue) and the empty tangent spheres (red) corresponding to the Voronoi vertices. (b) Edges of the Voronoi cells of 3D balls (left) and the empty tangent spheres corresponding to the Voronoi vertices (right).

atoms are linearly related to their respective distance to the dividing surface. The dividing surface is no longer a plane but a quadric surface (hyperboloid) producing Voronoi cells with faces that in general are not flat. This type of diagram is known as the Voronoi diagram of balls/spheres,[6] the additively weighted Voronoi diagram,[7] or the Apollonius diagram.[8]

Although the Voronoi diagram of balls is particularly well-suited for the analysis of 3D structures of biological macromolecules, so far this approach has not been utilized as widely as it might be expected. The main reason of its limited use appears to be the shortage of efficient algorithms and the associated software tools. Therefore, in most applications, in which the Voronoi diagram of balls would be the most appropriate approach, simpler methods such as the ordinary Voronoi diagram of points or the Laguerre (power) diagram are adopted instead.

To our knowledge, there are only few algorithms available for computing Voronoi diagram of balls with the focus on structures of biological macromolecules. One of the practical algorithms applied to protein structures was proposed by Kim et al.[6] The algorithm sequentially discovers the vertices of the Voronoi cells by tracing the edges of the cells. This algorithm was later improved by applying geometric filters for spatial search.[9,10] Medvedev et al.[11] published a similar algorithm, but it was reported[12] that the software implementing their algorithm is not suitable for typical proteins. Kim et al.[13,14] introduced an algorithm for constructing the quasitriangulation, which is a data structure dual to the Voronoi diagram of balls. Thus, the quasitriangulation is analogous to the Delaunay triangulation,[15] the dual of the Voronoi diagram of points. Previously, we used the Voronoi diagram of balls in Voroprot, an interactive tool for the analysis of complex geometric features of protein structure.[16] However, Voroprot was developed mainly as a visual analysis tool, not intended for batch processing or analysis of extremely large biomolecular structures.

In this article, we present a simple yet efficient algorithm and the corresponding open-source software for computing the vertices of the Voronoi diagram of 3D balls, a critical step in constructing the Voronoi diagram of 3D balls or its dual data structure, the quasitriangulation. The algorithm can be applied to 3D structures of various biological macromolecules including proteins, nucleic acids, protein–protein, and protein–nucleic acids complexes. The computed Voronoi vertices can be used in unequivocally defining atomic neighborhoods, describing internal cavities in molecular structures or constructing edges and faces of Voronoi cells of atoms. We have recently applied this approach for the large-scale computation of residue–residue contact areas in both experimental protein structures and theoretical protein models of different quality.[17] Here, we provide a detailed description of the algorithm. We then describe the software implementation and provide large-scale tests illustrating its speed and robustness. In addition, we compare the performance of our software with the performance of QTFier[18] and awVoronoi[19] that, to the best of our knowledge, are the only other publicly available tools that include similar functionality.

Methods

The Voronoi diagram of 3D balls and the corresponding Voronoi vertices

Let $B=\{b_1, b_2, \ldots, b_n\}$ be a set of balls, where $b_i=\langle c_i, r_i\rangle$ is a ball with a center $c_i \in \mathbb{R}^3$ and a radius $r_i \in \mathbb{R}_0^+$. A signed distance $d(p, b_i)$ from a point $p \in \mathbb{R}^3$ to a ball b_i is defined as follows:

$$d(p, b_i)=\|p-c_i\|-r_i \quad (1)$$

The Voronoi cell V_i for a ball b_i is a region containing all points closest to b_i:

$$V_i=\{p \in \mathbb{R}^3 | d(p, b_i) \leq d(p, b_j), \forall b_j \in B \backslash b_i\} \quad (2)$$

A set $\{V_1, V_2, \ldots, V_n\}$ is the Voronoi diagram for B. Figure 1 contains examples of the Voronoi cells of balls. Two balls are considered to be neighbors if their Voronoi cells intersect. The intersection of four Voronoi cells defines a point termed the Voronoi vertex. It is the center of an empty sphere tangent to the four neighboring balls (Fig. 2a). Notably, some Voronoi cells of balls may have no vertices—such situations are analyzed separately.

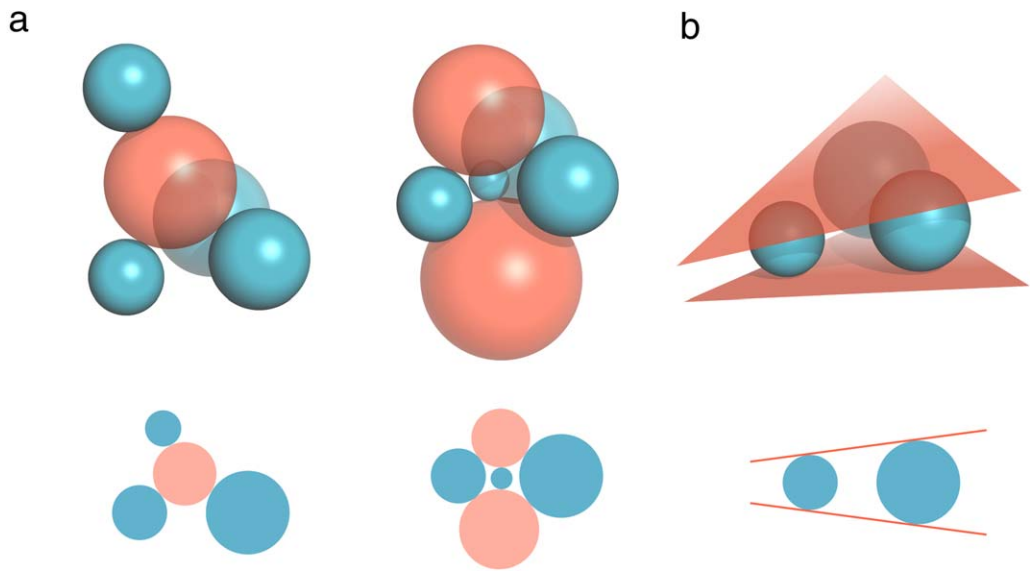

Figure 2. (a) Quadruples of 3D balls having either one (left) or two (right) tangent spheres. (b) A triple of 3D balls having two tangent planes. Diagrams below each 3D example show corresponding similar cases in 2D space. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Outline of the algorithm for finding the Voronoi vertices

Given an input set of balls B, our goal is to find the quadruples of balls that define all the vertices of the Voronoi diagram for B. In other words, we search for quadruples of balls that have at least one tangent sphere, which does not intersect with any ball from the input set. We term such quadruples valid. Each valid quadruple is a union of four valid triples. Starting with a single valid triple, we can discover valid quadruples by finding valid neighbors for previously detected valid triples. This principle, commonly known as "gift wrapping",[20] is used in algorithms for both the construction of the Delaunay triangulation of points[21,22] and for the construction of the quasitriangulation of balls.[6,11] We exploit the same principle, but use a different take on searching for valid triples and their neighbors.

In Procedure 1, we implement the "gift wrapping" strategy with an important modification: we take into account that a network of valid quadruples may be disconnected.[23] This is achieved by having two "while" cycles. The inner cycle (starting at line 8) finds as many quadruples as possible starting from a valid triple. The outer cycle (starting at line 6) runs while there still are valid triples containing balls that are not part of any of the already found quadruples.

In the next sections, we explain the algorithm in detail. To begin, we briefly describe the technique we use for computing tangent spheres. We then define the two complex subprocedures incorporated into Procedure 1: finding the first valid triple (lines 5 and 22) and finding all neighbors for a valid triple (line 11). Both subprocedures utilize the same technique for efficient searching in a large set of balls, which is also described later in the text.

Procedure 1 Find valid quadruples

input: B = (a set of balls)
output: Q = (a set of valid quadruples for B)
1: $Q \leftarrow$ (an empty set for found quadruples)
2: $T \leftarrow$ (an empty set for processed triples)
3: $M \leftarrow$ (an empty map to associate triples with sets of their neighbors)
4: *stack* $\leftarrow$ (an empty stack for triples)
5: $t_f \leftarrow$ (for B, find a first valid triple)
6: **while** $t_f \neq \varnothing$ **do**
7: *push* (*stack*, t_f)
8: **while** *stack* is not empty do
9: $t \leftarrow$ *pop*(*stack*)
10: $T \leftarrow T \cup t$
11: $X \leftarrow$ (for B, find a set of all neighbors of t, excluding $M[t]$)
12: **for all** $x \in X$ do
13: $q \leftarrow$ (a quadruple from t and x)
14: $Q \leftarrow Q \cup q$
15: $T_q \leftarrow$ (a set of all triples from q)
16: **for all** $t_q \in T_q$ do
17: $x_q \leftarrow$ (a neighbor of t_q in q)
18: $M[t_q] \leftarrow M[t_q] \cup x_q$
19: **if** $t_q \notin T$ then
20: *push*(*stack*, t_q)
21: $U \leftarrow$ (detect balls not included in any $q \in Q$)
22: $t_f \leftarrow$ (for B, find a first valid triple containing any $u \in U$)
23: **return** Q

Computing tangent spheres

To compute a tangent sphere for four balls $\{b_1, b_2, b_3, b_4\}$ (examples shown in Fig. 2a), we use the method proposed by Gavrilova and Rokne.[24] Let us assume without the loss of generality that b_4 has the smallest radius. We reduce the radii of all the four balls by the radius of b_4. We then move the balls such that b_4 coincides with the origin. This allows us to define an easily solvable system of equations for finding the coordinates and the radius of the tangent sphere. This system can have none, one, or two solutions. After solving the system, we restore the original positions and radii of $\{b_1, b_2, b_3, b_4\}$

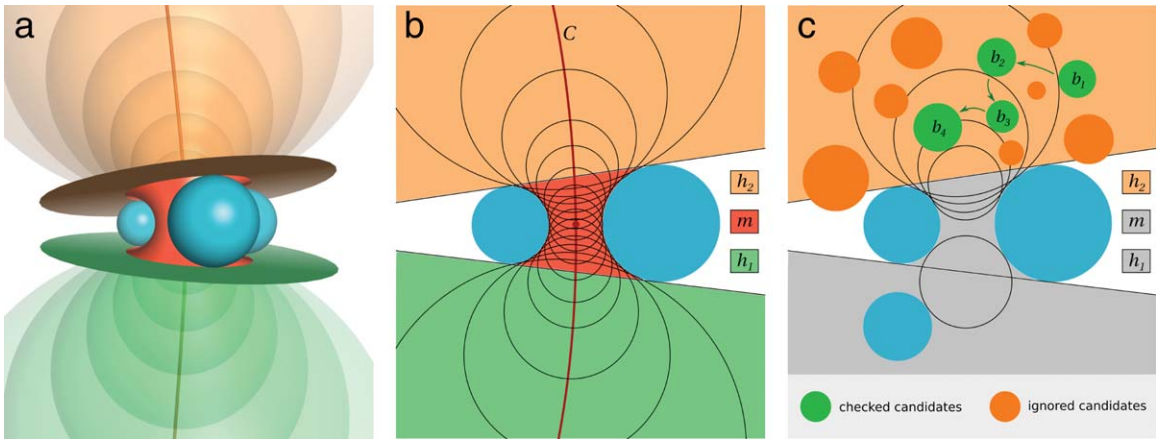

Figure 3. (a) Regions defined by two tangent planes in 3D. (b) Regions defined by two tangent planes in 2D. (c) Illustration of Procedure 2 for halfspace h_2: starting with b_1 the procedure runs until encountering b_4, which produces an empty h_2-related tangent sphere.

and transform each computed tangent sphere accordingly. Note that if the centers of tangent spheres are located inside the intersection of all four balls, tangent spheres will have negative radii.

There also may be tangent spheres of infinite radius. Their surfaces can be regarded as planes. To compute tangent planes for three 3D balls (an example shown in Fig. 2b), we use an approach similar to the one used when computing tangent spheres for four balls.

Finding the first valid triple

We make an assumption that balls close to each other are likely to form valid quadruples. Thus, we take the first ball $b_0 \in B$ and select a set of its nearest neighbors $B_0 \subset B$. We then start enumerating quadruples for B_0. If a quadruple has a tangent sphere that does not intersect any ball from B, then a quadruple is valid and one of its triples is returned. If no valid quadruples are found, B_0 is expanded and more quadruples are enumerated and tested.

In line 22 of Procedure 1, we need to find a first valid triple that should contain a ball that was previously not included in any of the already found valid quadruples. It may not always be possible, therefore, for such constrained search we limit the maximum size of B_0 to avoid enumeration of all quadruples for B.

Finding all neighbors for a valid triple

Constricted and loose triples. Consider a valid triple of balls $t=\{a,b,c\} \subset B$. Generally, t can have infinitely many possible tangent spheres, and a ball d can have a tangent sphere with t if and only if d intersects or touches the volume defined by the union of all the possible tangent spheres of t. If t has infinitely large tangent spheres, then these spheres can be regarded as tangent planes. If t has exactly two tangent planes, we call it a constricted triple. Otherwise, we call it a loose triple. We define separate algorithms of finding neighbors for constricted and loose triples.

Search space for a constricted triple. A constricted triple t has two tangent planes, therefore, the union of all the possible tangent spheres of t is a union of the following three regions:

- the halfspace h_1 defined by the first tangent plane;
- the halfspace h_2 defined by the second tangent plane, h_1 and h_2 may intersect;
- the region m located between the two tangent planes.

Figures 3a and 3b provides an illustration of such a subdivision. The centers of all the possible tangent spheres of t belong to a continuous curve.[6] Let us denote this curve as C. C intersects the plane defined by the centers of the three balls in t at a single point p, which corresponds to the center of the smallest possible tangent sphere of t. When moving away from p along the curve C, the radius of the corresponding tangent sphere always grows.

Finding neighbors in the halfspaces defined by a constricted triple. Let us assume that for a constricted triple t, there is a ball d_i such that d_i intersects halfspace $h_x \in \{h_1, h_2\}$. If t and d_i have a single tangent sphere s_i, let us call s_i a h_x-related tangent sphere for t and d_i (if t and d_i have two tangent spheres, then one of them closer to h_x is called h_x-related). Another tangent sphere s_j of t can be produced by moving the center of s_i along the curve C (with the radius of s_j changing so that s_j remains tangent to t). If the movement is directed toward h_x, then s_j intersects d_i, therefore, s_j is not empty. Otherwise the movement is directed away from h_x and s_j does not even touch d_i. In this case, if s_i is empty, then s_j does not touch any ball in h_x. Therefore, if s_i is empty, then it is the only empty h_x-related tangent sphere for t.

The properties of h_x-related tangent spheres allow us to define Procedure 2 for finding a valid neighbor of t in halfspace h_x. Along with the pseudocode, we provide a simplified description of the procedure:

1. The procedure starts with any ball that intersects h_x and produces a h_x-related tangent sphere with t;

2. The procedure selects any ball that intersects both h_x and the previously produced tangent sphere and produces another h_x-related tangent sphere;
3. If step 2 has produced a tangent sphere, then step 2 is repeated;
4. If the last produced tangent sphere is empty, then the last selected ball is a valid neighbor.

Procedure 2 is greedy, it does not check all the balls that intersect h_x. See Figure 3c for an illustration of the procedure run. Procedure 2 does not need to be called if a valid neighbor of t from h_x is already known from the previously found valid quadruple that contains t. Therefore, for most valid triples the procedure is performed only once. Also, the running time of the procedure can be reduced if in line 2 a ball is selected from a close neighborhood of t.

Procedure 2 Find a valid neighbor of a triple in a halfspace

input: B = (a set of balls), t = (a triple of balls, $t \subset B$), h_x = (a halfspace, $h_x \in \{h_1, h_2\}$)
output: d = (a valid neighbor of t) and s = (an empty h_x-related tangent sphere of t and d)
1: $\langle d, s\rangle \leftarrow \langle \varnothing, \varnothing\rangle$
2: $d_0 \leftarrow$ (select a ball $d_0 \in B$ such that there exists a h_x-related tangent sphere s_0 for t and d_0)
3: **while** $d_0 \neq \varnothing$ **do**
4: *intersection* $\leftarrow$ *false*
5: *replacement* $\leftarrow$ *false*
6: **repeat**
7: $d_1 \leftarrow$ (select another ball $d_1 \in B$ such that d_1 intersects s_0)
8: **if** $d_1 \neq \varnothing$ **then**
9: *intersection* $\leftarrow$ *true*
10: **if** there exists a h_x-related tangent sphere s_1 for t and d_1 then
11: *replacement* $\leftarrow$ *true*
12: **until** *replacement* = *true* or $d_1 = \varnothing$
13: **if** *replacement* = *true* then
14: $\langle d_0, s_0\rangle \leftarrow \langle d_1, s_1\rangle$
15: **else if** *intersection* = *true* then
16: $\langle d_0, s_0\rangle \leftarrow \langle \varnothing, \varnothing\rangle$
17: **else**
18: $\langle d, s\rangle \leftarrow \langle d_0, s_0\rangle$
19: $\langle d_0, s_0\rangle \leftarrow \langle \varnothing, \varnothing\rangle$
20: **return** $\langle d, s\rangle$

Finding neighbors in the middle region defined by a constricted triple

After valid neighbors of t from both h_1 and h_2 are determined, there may be remaining valid neighbors that do not intersect h_1 or h_2 but intersect middle region m and have empty tangent spheres with t. For a fast intersection checking, we need a simple approximation of m. Let us consider the surface of m. It is known to be a part of the Dupin cyclide defined by balls in t (Fig. 4a).[6,25] A Dupin cyclide is an envelope surface of spheres tangent (both externally and internally) to the three fixed spheres. The surface of m is part of the Dupin cyclide that corresponds only to externally tangent spheres, that is, tangent spheres that do not overlap the three fixed spheres. Each externally tangent sphere of t has three points touching balls in t. The circumcircles of such triples of touching points lie on the surface of m (Fig. 4b).[26] There are two circumcircles that also lie on the touching planes of t because they correspond to the largest possible tangent spheres. We use a bounding cylinder of these two circumcircles as an initial approximation of m (Fig. 4c). We can reduce the size of this bounding cylinder by considering circumcircles that correspond not to the largest possible tangent spheres of t, but to the largest empty tangent spheres of t, that is, empty h_1-related and h_2-related tangent spheres, if such exist. If a ball intersects the defined bounding cylinder and is located between the two tangent planes of t, then this ball is checked for having at least one empty tangent sphere with t.

Notably, circumcircles defined by tangent spheres of negative radii (see Fig. 4b3 for an example) may lie outside of the cylinder approximating region m. However, this does not present a problem, because if a valid neighbor of t corresponds to a tangent sphere of negative radius, then this neighbor overlaps the center of that tangent sphere and, therefore, intersects the cylinder approximating m.

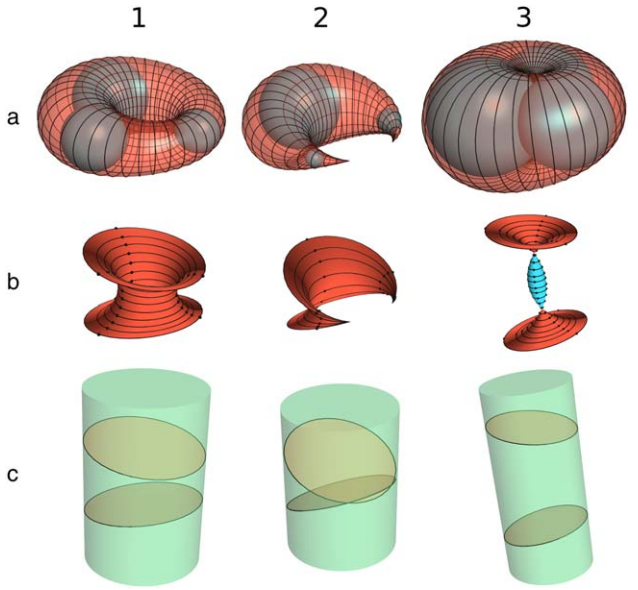

Figure 4. (a) For constricted triples of balls, it is possible to define Dupin cyclides of three types: ring-like (1), horn-like (2), and spindle-like (3). Horn-like cyclides are generally two-part, but we only show the part relative to the middle region m. (b) 3D surfaces of middle regions are parts of Dupin cyclides displayed above. Black circles indicate circumcircles of the points, at which externally tangent spheres touch the balls of a triple. The circles lie on the surface of a middle region. The topmost and bottommost circles also lie on touching planes and are used for approximating a middle region. In the last case (3), the middle part of the surface corresponds to tangent spheres that have negative radii because they lie inside the intersection of all the balls of a triple. (c) Bounding cylinders of the topmost and bottommost circles shown in (b). [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Finding all neighbors for a loose triple. Let us now consider a case, in which a valid triple t does not have exactly two tangent planes. In this case, the search for valid neighbors is performed in a brute-force manner: each ball $b \in B \backslash t$ is checked for having at least one empty tangent sphere with t. It should be possible to define a faster but a more complex procedure for handling loose triples. However, this would not significantly improve the overall performance of the algorithm, because our tests (described later in the text) show that in macromolecular structures loose triples occur very rarely.

Efficient searching in a large set of balls

Let us summarize geometric search operations that we need to implement: search for balls that intersect a half-space; search for balls that intersect a sphere; search for balls that intersect a cylinder. To implement them efficiently, we need a search data structure. We chose to use a bounding spheres hierarchy (BSH)[27] because it does not add any additional complexity when implementing geometric queries that we need: checking any bounding sphere for an intersection with some object is no different from checking an input ball for the same thing. Our approach to the construction of BSH for a set of input balls B can be summarized as follows:

1. The elements of B form the leaf nodes of the tree;
2. Nodes created in the previous step are grouped and enclosed within bounding spheres which form the higher level of nodes;
3. Step 2 is performed in a recursive fashion eventually resulting in a tree structure with a single bounding sphere at the top of the tree.

In step 2, we can use the following algorithm:

1. Select group centers from the input spheres using the greedy Procedure 3;
2. Assign each input sphere to the group with the nearest center;
3. Construct a bounding sphere for each group.

This algorithm is practical only for a relatively small number (less than 10^5) of input spheres, because Procedure 3 has quadratic time complexity. To overcome this problem, we provide input in smaller portions. The portions are determined by recursively subdividing the input set of spheres using *k-d* tree subdivision algorithm.[28]

Procedure 3 Select group centers in BSH construction

input: S = (a list of spheres), l_{min} = (minimal distance between group centers)
output: $S_{selected}$ = (a set of selected group centers)
1: $S \leftarrow$ (order S by the distance to $S[0]$)
2: $S_{selected} \leftarrow \varnothing$
3: $S_{locked} \leftarrow \varnothing$
4: **for all** $a \in S$ **do**
5: **if** $a \notin S_{locked}$ then
6: $S_{selected} \leftarrow S_{selected} \cup a$
7: **for all** $b \in S$ do
8: **if** $distance(a, b) < l_{min}$ then
9: $S_{locked} \leftarrow S_{locked} \cup b$
10: **return** $S_{selected}$

Two examples of bounding spheres hierarchies are shown in Figure 5. Searching in BSH is performed as in any other tree structure—children are not examined if their parent does not satisfy the predefined condition. In the case of BSH, a node is not examined if the bounding sphere of the parent node does not satisfy the predefined constraint. A search starts from the root node and can be performed in either depth-first or breadth-first manner. If we need to find out whether at least one ball from B satisfies some condition (e.g., if any ball intersects a tangent sphere), then the depth-first search method is more beneficial because it reaches the leaves level faster.

Handling special situations

The Voronoi diagram of balls may exhibit various special cases and anomalies.[14,29] As our algorithm searches for Voronoi vertices, we focus on handling two special situations that relate to the existence of Voronoi vertices and valid quadruples.

First, let us consider a ball that has the Voronoi cell without vertices and, therefore, is not part of any valid quadruple. As noted by Medvedev et al.,[11] such orphaned balls can be identified and handled separately after the search for the Voronoi vertices. We choose to simply report the orphaned balls. Our tests, described later in the text, show that occurrences of orphaned balls in macromolecular structures are extremely rare and that they represent physically nonrealistic stereochemistry.

Second, let us consider a situation where more than four balls share the same empty tangent sphere. If n is the number of these

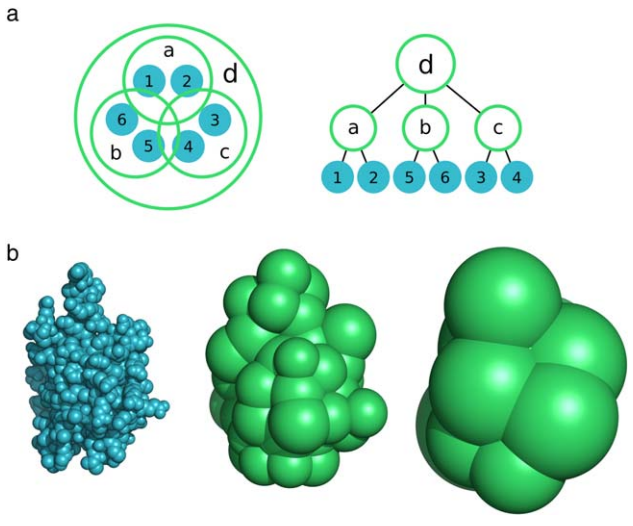

Figure 5. (a) 2D example of a BSH. (b) Illustration of a BSH applied to a protein structure: protein atoms (left), the first layer (middle), and the second layer (right) of bounding spheres. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

spheres, then it is possible to select up to $\binom{n}{4}$ quadruples defining the same Voronoi vertex. Our algorithm selects a smaller set of quadruples because it considers halfspaces defined by triples of balls. For example, if the algorithm is applied to a set of points where more than four points share the same circumsphere, it produces a valid triangulation where simplices meet edge-to-edge or vertex-to-vertex and do not overlap. However, we provide an optional procedure for finding all possible valid quadruples. After computing all the Voronoi vertices, we use a BSH to search for all the touching balls for each of the constructed tangent spheres. We then report surplus quadruples for each tangent sphere that has more than four touching balls.

Parallelization of the algorithm

We parallelize the algorithm by implementing the following strategy:

1. Subdivide the set of input balls B into k smaller sets $B_1, B_2, \ldots, B_k$.
2. In parallel: for each $B_i \in \{B_1, B_2, \ldots, B_k\}$ find a set Q_i of all valid quadruples that contain at least one ball from B_i.
3. Return the full set of quadruples $Q = Q_1 \cup Q_2 \cup \ldots \cup Q_k$.

In step 1, we recursively subdivide the input set of balls using k-d tree subdivision algorithm,[28] so that during each subdivision step the input is divided into parts that are as similar in size as possible. In step 2, we run the algorithm for B as defined in Procedure 1, but maintain the following constraint: every triple pushed into the stack should contain at least one element of B_i. This requires a simple modification in the procedure for finding a first valid triple (called in lines 5 and 22 of Procedure 1): a returned first valid triple should contain at least one element of B_i.

Convergence of the algorithm

Let us show that both sequential and parallel versions of our algorithm find all the Voronoi vertices. Missing a Voronoi vertex implies missing a valid quadruple (otherwise it would imply a wrongful rejection of an empty tangent sphere, which is not possible because the check for the sphere emptiness is performed explicitly). Let us assume that there is a valid quadruple q_m that was missed. Let us consider a ball $b_m \in q_m$. If b_m is not a part of some found valid quadruple, then the procedure for finding the first valid triple containing b_m would be called, which would find either q_m or some other valid quadruple containing b_m. Therefore, b_m must be a part of some found valid quadruple $q_f \neq q_m$. Let us now consider the Voronoi cell V_m of b_m. For any two vertices of V_m there is a path of Voronoi edges between them (none of the special cases[14] of Voronoi cells of balls have disjoint sets of vertices). Each Voronoi edge of V_m corresponds to a valid triple that contains b_m. Therefore, there is a path of valid triples containing b_m between any two valid quadruples containing b_m. The sequential version of the algorithm would follow such a path because it would search for neighbors of every valid triple (including the triples that are subsets of q_f). The parallel version would follow such a path because it would search for neighbors of every valid triple that contains b_m when processing the subset of the input balls that contains b_m. Thus, if q_f was found, then q_m would be found too, which contradicts the initial assumption that q_m was missed.

Results

Implementation

Our algorithm for computing the vertices of the Voronoi diagram of 3D balls is implemented as an open-source C++ program, named Voronota. It has no external dependencies, and only a C++ compiler is needed to build it. In addition, we developed parallel implementations of the algorithm using OpenMP and MPI technologies.

All the geometric calculations are implemented using double precision floating point numbers (C++ "double" data type). To reduce the effects of numerical errors, we apply several techniques. We use Kahan summation algorithm[30] in the code for solving equations when computing tangent spheres and tangent planes. For quadratic equations, we use a numerically safer solving algorithm.[31] After computing each tangent sphere or plane, we calculate to what extent the computed object is really tangent: the obtained tangency error estimate is used when performing intersection queries.

As an input, Voronota accepts a list of balls in the plain text format. The software provides a way to create such lists of balls from files in PDB and mmCIF formats. By default, all heteroatoms and all hydrogen atoms are ignored, but this behavior can be altered using command-line options. Voronota also offers the possibility to customize VDW radii of atoms. The output of Voronota is an easily parseable list of valid quadruples and the corresponding empty tangent spheres.

Testing on Protein Data Bank structures

The software was tested on Intel Core i7–2600 3.40GHz processor. First, we compared the performance of our software and of two other tools: QTFier[18] (version 1.0) and awVoronoi[19] (version 1.0.0). These tools output both the Voronoi vertices and the topological links between them, whereas Voronota outputs only Voronoi vertices. Therefore, we provide a running time comparison only for an informational purpose.

The test set consisted of all asymmetric units available from the protein data bank (PDB) database[32] as of May 15, 2013 (90,365 structures, each having at least four non-hydrogen "ATOM" records). Hydrogen atoms and heteroatoms were removed from the input structures. All the three tools were given the same set of coordinates and used the same set of VDW radii[33] (it was the only set of radii available in QTFier). For speed analysis, we measured CPU-time needed to process every input structure (Figs. 6a and 6b). Voronota processed the test set in about 34.8 h, QTFier and awVoronoi in 138.2 and 172.9 h, respectively. Importantly, Voronota did not fail on any of the input PDB structures, whereas QTFier failed on 259 and awVoronoi failed on 104.

Results obtained on the PDB test set enabled us to make some generalizations. For example, it turned out that the

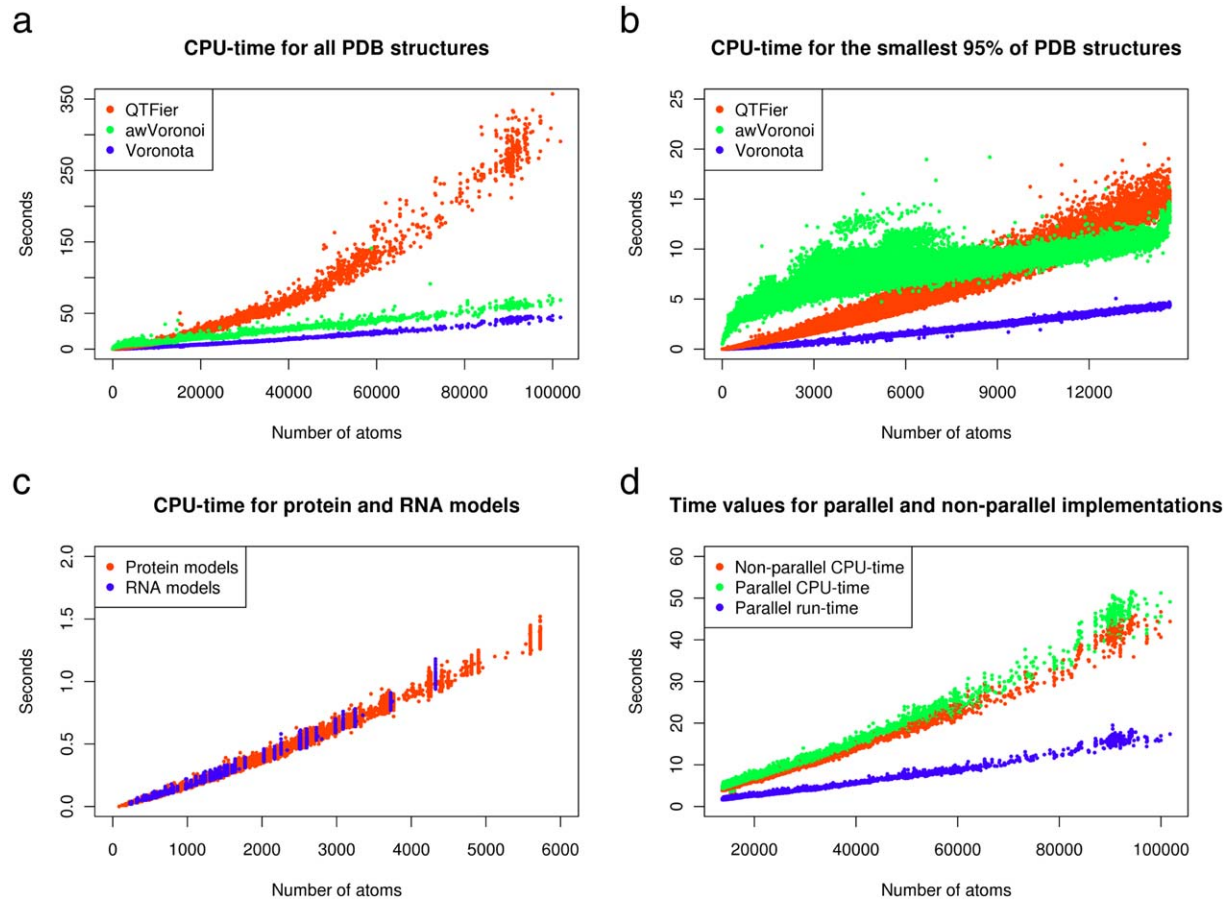

Figure 6. (a) CPU-time values for the macromolecular structures available from the PDB database. (b) CPU-time values for 95% of smallest structures from PDB. (c) Voronota CPU-time values for protein and RNA structural models. (d) Voronota run-time and CPU-time values for nonparallel and parallel implementations. Parallel implementation was executed on four computational units. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

number of valid quadruples linearly correlates with the number of atoms (Pearson's correlation coefficient being greater than 0.99). On average, the number of valid quadruples was about 6.6 times greater than the number of atoms and about two times smaller than the number of valid triples. Only less than 0.005% of all the valid triples were not constricted, and only 18 of about 4.5×10^8 atomic balls did not have any Voronoi vertices. On average, about 11 quadruples had to be examined to find the first valid triple.

We also asked if there is any quadruple q that meets both of the following two conditions: (1) q is found by QTFier or awVoronoi, but not by Voronota; (2) q is valid with respect to the 10^{-10} angstroms threshold used for checking if any of the tangent spheres defined by q is empty. There were some quadruples meeting the first condition, but none of them met the second condition. One of the reasons for slight differences in the output is that the three programs handle floating point arithmetic errors differently. The tools may also be using different approaches to handle degenerate situations.

To check if Voronota is capable of properly handling molecules with hydrogen atoms, we performed a similar test routine with all the NMR entries from the initial PDB set (9883 structures, only first structural model from each entry was used). This time hydrogen atoms were retained. Voronota successfully processed all the input structures. In comparison with the hydrogen-free testing results, there were more nonconstricted valid triples ($\sim$ 0.5%) and atomic balls that did not have any Voronoi vertices (117 atomic balls from 39 input structures). Voronota processed the input set in about 1.3 h, QTFier and awVoronoi in 3.3 and 15.4 h, respectively.

We analyzed all the situations where either a hydrogen or non-hydrogen atom did not have any corresponding Voronoi vertices. We found that these cases represent either unrealistically short covalent bonds or severe steric clashes of non-bonded atoms. Therefore, such situations may be considered to be indicators of dubious low-quality macromolecular structures.

Testing on protein and RNA structural models

Computational structural models are becoming widely used for various applications. However, models, especially of lower accuracy, may have a number of physically unfeasible features. Therefore, we decided to test whether Voronota is sufficiently robust to be used for computational models. To this end, we

used 28,806 protein models ranging widely in their quality submitted by modeling servers to CASP9[34] and CASP10[35] experiments. Voronota successfully processed all the structural models (QTFier failed on 31 and awVoronoi failed on 875 input structures). In addition, Voronota was successful in processing all 42,585 RNA models from the "randstr" decoys set.[36] For the protein and RNA models, the relation between structure size and CPU-time (Fig. 6c) was consistent with the results for the PDB structures (Figs. 6a and 6b).

Testing parallel implementations

We tested the performance of our OpenMP-based parallel implementation on the 5000 largest structures from PDB. The execution was performed on the same machine as before, four computational units were used for each input structure. Figure 6d shows the recorded run-time (real time) and CPU-time (total amount of time spent by all the used computational units) values for both nonparallel and parallel implementations.

The message passing interface (MPI)-based parallel implementation is likely most suitable for processing very large structures on a computing cluster. For example, we processed the HIV virus capsid structure (PDB ID 3J3Q, 2,440,800 atoms) on a cluster of Intel Xeon X5650 2.66 GHz processors. When nine CPU cores were used, run-time and CPU-time values were 511 and 4176 seconds, respectively. For 17 cores, the values were 293 and 4330 s, for 33 cores—189 and 5018 s.

Conclusions

In this article, we presented a simple and robust algorithm for computing the vertices of the Voronoi diagram of balls. The algorithm is particularly well-suited for processing 3D structures of biological macromolecules. It takes advantage of the observation that in the case of macromolecular structures the overwhelming majority of valid ball (atom) triples are constricted (have two tangent planes). When processing constricted triples, our algorithm efficiently combines the knowledge of the search space with the use of hierarchical spatial indexing. The algorithm uses a bounding spheres hierarchy (BSH) to iteratively search for neighbors so that the search space is reduced after each iteration. Importantly, we introduce a simple approximation for the middle region of the search space defined by the constricted triple. When processing rare loose triples (triples without two tangent planes) our algorithm does not attempt to reduce the search space, but still uses a BSH to speed up the search for neighbors. This strategy works well in terms of speed and simplicity of the algorithm implementation. Another important feature of the algorithm is the simplicity and generality of the procedure for finding the first valid triple, which enabled us to parallelize the algorithm in a straightforward manner.

We implemented the algorithm as an open-source console application, Voronota, which can be run on either single or multiple processors. Large-scale tests showed that Voronota is a fast and reliable tool for processing both experimentally determined and computationally modeled macromolecular structures. Voronota is easy to deploy and use. It can serve as a core component for developing other tools that exploit the Voronoi diagram of balls.

Acknowledgment

The authors thank Mindaugas Margelevičius, Visvaldas Kairys, Justas Dapkūnas, and Rimvydas Krasauskas for critically reading the manuscript.

Keywords: protein structure · RNA structure · Voronoi diagram of balls or spheres · Voronoi vertices · software

How to cite this article: K. Olechnovič, Č Venclovas. *J. Comput. Chem.* **2014**, *35*, 672–681. DOI: 10.1002/jcc.23538

[1] A. Poupon, *Curr. Opin. Struct. Biol.* **2004**, *14*, 233.
[2] G. Voronoi, *J. Reine Angew. Math.* **1908**, *134*, 198.
[3] F. M. Richards, *J. Mol. Biol.* **1974**, *82*, 1.
[4] B. J. Gellatly, J. L. Finney, *J. Mol. Biol.* **1982**, *161*, 305.
[5] A. Goede, R. Preissner, C. Frömmel, *J. Comput. Chem.* **1997**, *18*, 1113.
[6] D. S. Kim, Y. Cho, D. Kim, *Comput. Aided Des.* **2005**, *37*, 1412.
[7] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Wiley Series in Probability and Statistics; Wiley, Springer-Verlag New York, Inc., Secaucus, NJ, USA, **2000**.
[8] J. D. Boissonnat, M. Teillaud, Effective Computational Geometry for Curves and Surfaces (Mathematics and Visualization); Springer-Verlag: New York, **2006**.
[9] Y. Cho, D. Kim, H. Lee, J. Park, D. S. Kim, *Lect. Notes Comput. Sci.* **2006**, *3980*, 111.
[10] M. Manák, I. Kolingerová, In 2010 International Symposium on Voronoi Diagrams in Science and Engineering; IEEE Computer Society, Quebec, Canada, June 28-30, 2010. pp. 95–104.
[11] N. N. Medvedev, V. P. Voloshin, V. A. Luchnikov, M. L. Gavrilova, *J. Comput. Chem.* **2006**, *27*, 1676.
[12] N. Lindow, D. Baum, H. C. Hege, *IEEE Trans. Vis. Comput. Graph.* **2011**, *17*, 2025.
[13] D. S. Kim, D. Kim, Y. Cho, K. Sugihara, *Comput. Aided Des.* **2006**, *38*, 808.
[14] D. S. Kim, Y. Cho, J. K. Kim, J. Ryu, *Comput. Aided Des.* **2012**, *44*, 835.
[15] B. Delaunay, Izvestiya Akademii Nauk SSSR, *Otdelenie Matematicheskikh i Estestvennykh Nauk* **1934**, *7*, 793.
[16] K. Olechnovič, M. Margelevičius, Č. Venclovas, *Bioinformatics* **2011**, *27*, 723.
[17] K. Olechnovič, E. Kulberkytė, Č. Venclovas, *Proteins* **2013**, *81*, 149.
[18] QTFier software web-page, Available at http://voronoi.hanyang.ac.kr/software.htm. Accessed on January 30, 2014.
[19] awVoronoiURL software web-page, Available at http://sourceforge.net/projects/awvoronoi. Accessed on January 30, 2014.
[20] P. Su, R. L. S. Drysdale, *Comput. Geom.* **1997**, *7*, 361.
[21] T. Masaharu, T. Ogawa, N. Ogita, *J. Comput. Phys.* **1983**, *51*, 191.
[22] A. Maus, *BIT Numer. Math.* **1984**, *24*, 151.
[23] D. S. Kim, Y. Cho, K. Sugihara, *Comput. Aided Des.* **2010**, *42*, 874.
[24] M. L. Gavrilova, J. Rokne, *Comput. Aided Geom. D* **2003**, *20*, 231.
[25] W. Degen, In Handbook of Computer Aided Geometric Design; Elsevier, Amsterdam, **2002**; pp. 575–601.
[26] L. Druoton, L. Garnier, R. Langevin, H. Marcellier, R. Besnard, *Comm. Com. Inf. Sc.* **2011**, *167*, 406.
[27] J. Spillmann, M. Becker, M. Teschner, *J. Vis. Commun. Image Represent* **2007**, *18*, 101.
[28] J. L. Bentley, *Commun. ACM* **1975**, *18*, 509.
[29] D. S. Kim, Y. Cho, J. Ryu, J. K. Kim, D. Kim, *Comput. Aided Des.* **2013**, *45*, 35.
[30] W. Kahan, *Commun. ACM* **1965**, *8*, 40.
[31] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C++: The Art of Scientific Computing; Cambridge University Press, New York, NY, USA, **2002**.
[32] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, P. E. Bourne, *Nucleic Acids Res.* **2000**, *28*, 235.
[33] A. Bondi, *J. Phys. Chem.* **1964**, *68*, 441.

[34] J. Moult, K. Fidelis, A. Kryshtafovych, A. Tramontano, *Proteins* **2011**, *79*, 1.
[35] J. Moult, K. Fidelis, A. Kryshtafovych, T. Schwede, A. Tramontano, Proteins, **2014**, *82*, 1–6, ISSN 1097-0134, DOI: 10.1002/prot.24452.
[36] E. Capriotti, T. Norambuena, M. A. Marti-Renom, F. Melo, *Bioinformatics* **2011**, *27*, 1086.

Received: 25 October 2013
Revised: 11 December 2013
Accepted: 6 January 2014
Published online on 6 February 2014