Computer Physics Communications 181 (2010) 2116–2125

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

CAVE: A package for detection and quantitative analysis of internal cavities in a system of overlapping balls: Application to proteins ☆

Ján Buša [a,b,c], Shura Hayryan [a], Chin-Kun Hu [a,d,*], Jaroslav Skřivánek [a,e], Ming-Chya Wu [a,f,g]

[a] *Institute of Physics, Academia Sinica, Nankang, Taipei 11529, Taiwan*
[b] *Technical University in Košice, 040 01 Košice, Slovak Republic*
[c] *Joint Institute for Nuclear Research, Dubna, Russia*
[d] *Center for Nonlinear and Complex Systems and Department of Physics, Chung Yuan Christian University, Chungli 32023, Taiwan*
[e] *SORS Research, 04001 Košice, Slovak Republic*
[f] *Research Center for Adaptive Data Analysis, National Central University, Chungli 32001, Taiwan*
[g] *Department of Physics, National Central University, Chungli 32001, Taiwan*

ARTICLE INFO

Article history:
Received 22 March 2010
Received in revised form 20 August 2010
Accepted 24 August 2010
Available online 17 September 2010

Keywords:
Internal cavities
Excluded volume
Proteins
Triangulation
Fortran package
SMMP
ARVO
CAVE

ABSTRACT

We developed a software package (CAVE) in Fortran language to detect internal cavities in proteins which can be applied also to an arbitrary system of balls. The volume, the surface area and other quantitative characteristics of the cavities can be calculated. The code is based on the recently suggested enveloping triangulation algorithm [J. Buša et al., J. Comp. Chem. 30 (2009) 346] for computing volume and surface area of the cavity by analytical equations. Different standard sets of atomic radii can be used. The PDB compatible file containing the atomic coordinates must be stored on the disk in advance. Testing of the code on different proteins and artificial ball systems showed efficiency and accuracy of the algorithm. The program is fast. It can handle a system of several thousands of balls in the order of seconds on contemporary PC's. The code is open source and free.

Program summary

Program title: CAVE
Catalogue identifier: AEHC_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEHC_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 8670
No. of bytes in distributed program, including test data, etc.: 100 131
Distribution format: tar.gz
Programming language: Fortran
Computer: PC Pentium and Core
Operating system: Linux system and Windows XP system
Classification: 16.1
Nature of problem: Molecular structure analysis.
Solution method: Analytical method for cavities detection, and numerical algorithm for volume and surface area calculation based on the analytical formulas, after using the stereographic transformation.
Running time: Depends on the size of the molecule under consideration. The test example included in the distribution takes about 1 minute to run.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author (Chin-Kun Hu) at: Institute of Physics, Academia Sinica, Nankang, Taipei 11529, Taiwan.
E-mail addresses: Jan.Busa@tuke.sk (J. Buša), shura@phys.sinica.edu.tw (S. Hayryan), huck@phys.sinica.edu.tw (C.-K. Hu), Jaroslav.Skrivanek@tuke.sk (J. Skřivánek), mcwu@ncu.edu.tw (M.-C. Wu).

0010-4655/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2010.08.032

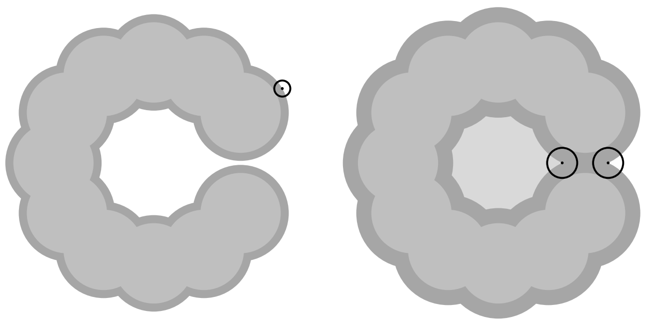

Fig. 1. The same molecule without (left) and with (right) a cavity depending on the radius of the probe atom.

1. Introduction

Native structures of many proteins have cavities [1–4]. Elucidating the role of the cavities in function and energetics of the proteins has been a challenge in experimental and computational studies of proteins in last two decades. Many experiments [5–9] have shown that changes in the size and the shape of the cavities influence considerably the stabilization energy of the protein structures. As described in [10], even the function of the protein can change when the size of the cavities changes.

Computational algorithms for detection and quantitative characterization of the cavities are usually based on the space filling geometry model of the protein by Lee and Richards [11] which interprets a protein as a union of mutually interpenetrating balls. Each ball represents an atom and its size is defined by the van der Waals radius. The algorithms belong to the family of shape description algorithms and are closely related to those for calculating the surface area and the volume of molecules [11–24]. The first computational investigation of the cavities has been reported by Lee and Richards [11]. Rashin, et al. [25] have developed a program for detection of the internal cavities and for prediction of the positions of buried water molecules. Hubbard, et al. [26,27] have analyzed internal cavities in 121 protein chains and found that the overall cavity volume increases with the protein size. In their calculations they used algorithms from [15,18,28]. Zhang and Hermans [29] used the molecular surface calculation algorithm [15] to calculate energies and free energies of a water molecule in cavities and discussed the hydrophobicity of protein cavities. The energetics of the empty cavities and internal mutations have been evaluated computationally in [30]. Liang, et al. [31] described an analytic method for computing the metric properties of macromolecules. Later this method has been applied to study quantitatively the inaccessible cavities in proteins [32]. Another analytical algorithm has been suggested in [33]. Procedures for identifying the buried water molecules in cavities were described in [34,35]. The surface triangulation method [36,37] has been applied by to study the structure and interactions of proteins [38]. Currently the cavity detection algorithms are included in some simulation packages like [39–41]. Collins and coworkers [42,43] used high-pressure crystallography and computer simulations to study cooperative water filling of a nonpolar protein cavity and the structural rigidity of large cavity-containing proteins. In [44], a problem of emptiness of apolar cavities has been reviewed.

Recently we proposed a new efficient analytical algorithm for detection and analysis of internal cavities [45]. The basic idea of the proposed method lies in the construction of a special *enveloping triangulation* such that the conclusion that a certain space point belongs or does not belong to the cavity, depends only on the relation between the point and the triangulation. In the present paper we describe the Fortran package `CAVE` which implements the algorithm. The package has been used to compute volumes and areas of cavities in many protein structures taken from the Protein Data Bank (PDB) [46]. The objective of this work has been to develop our own cavity detection software for the protein simulation package SMMP [47,48] which has been used widely to study structures and thermal properties of proteins [49,50]. The tests have shown that our algorithm is efficient, accurate, and reliable [45]. `CAVE` had been used to study the cavity of an antifreeze protein RD1 [51].

In order to better understand the program we suggest that users of CAVE also read our previous paper [45] and our papers about a related package ARVO [23,24], which can be used to calculate volume and surface area of a system of balls.

In above we have mentioned several packages for cavity calculations. However, to the best of our knowledge, CAVE should be the first open source code. Therefore, it has special value for students and teachers in the field of protein structure research.

This paper is organized as follows. In Section 2, we introduce basic definitions and terminology. A brief description of the algorithm is presented in Section 3 with the aid of two-dimensional (2D) systems, and the package structure of `CAVE` is introduced in Section 4. We comment on possible parallelization versions in Section 5. Steps of running `CAVE` are introduced in Section 6, with an exemplary run presented in Section 7 and its screen output in Appendix A.

2. Some definitions and terminology

The basic definitions and the related terminology are introduced in [45] but we review them briefly here in order to make easy the further reading of this paper.

(1) *Molecule, surface and center of the atom*. We consider a system of spherical balls $\mathcal{S}_i$ $(1 \leqslant i \leqslant N)$ that, in general, can overlap. $\mathcal{M} = \bigcup_i \mathcal{S}_i$ denotes the union of these balls. In case of proteins or other molecules the balls represent atoms, and correspondingly, $\mathcal{M}$ is the molecule. The spherical surface and the center of the atom $\mathcal{S}_i$ are denoted by $\delta\mathcal{S}_i$ and C_i, respectively.

(2) *Cavity*. A cavity $\mathcal{C}$ in the molecule $\mathcal{M}$ is a bounded, and *closed* domain in the three-dimensional (3D) space $\mathbb{R}_3$, whose full boundary consists of the parts of the molecular boundary, and whose interior, $\mathrm{Int}\,\mathcal{C}$, has no common points with $\mathcal{M}$. In case of nonzero size of probe radius we also consider a cavity as an internal domain, where the probe atom's center may be put, but which is unreachable for the probe atom from outside the molecule (see Fig. 1).

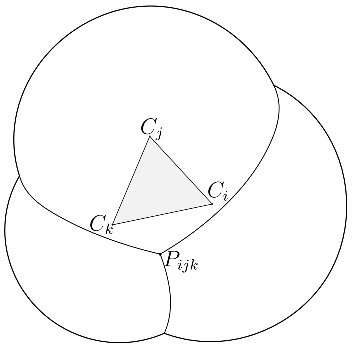

Fig. 2. A wall triangle $\triangle C_i C_j C_k$. $\mathcal{P}_{ijk}$ is the intersection point of three spheres, and it lies on the surface of the molecule.

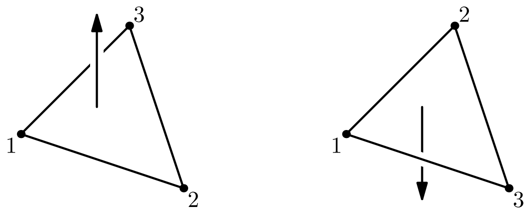

Fig. 3. Positively (arrow up) and negatively (arrow down) oriented triangles when viewed in the direction, perpendicular to the triangle's plane from above.

Important remark. When talking about the atoms we mean that their van der Waals radii are extended by the probe ball (usually the water molecule) radius [23,24]. This means that the number, sizes, and shapes of cavities depend strongly on the probe radius (see Fig. 1). Currently there is no common agreement about the van der Waals radii of protein atoms, and several sets are being used. This problem is discussed in [52] which contains an extensive list of related references.

(3) *Wall triangle*. Let C_1, C_2, C_3 be the center points of three atoms in $\mathcal{M}$. Then the triangle $\triangle C_1C_2C_3$ is called a *wall triangle* in $\mathcal{M}$ if and only if the intersection $\delta\mathcal{S}_1 \cap \delta\mathcal{S}_2 \cap \delta\mathcal{S}_3$ of the surfaces of the atoms is not empty and at least one of the intersection points does not belong to the interior of (lies on the molecular surface). For an example of a wall triangle, see Fig. 2.

(4) *Neighboring wall triangles, free vertex*. Two wall triangles are called neighbors to each other if they share a common edge. The planes of the neighbor triangles form a dihedral angle. Two vertices of the neighbor triangles coincide and the third one of each triangle is a *free* vertex.

(5) *Triangulation*. Triangulation is a special set of polyhedrons with wall triangular faces enclosing all cavities of the protein. The triangulation of the molecule $\mathcal{M}$ is denoted as $\triangle\mathcal{M}$.

(6) *Minimal covering triangulation*. Since a triangulation $\triangle\mathcal{M}$ is a set of wall triangles, it can be *nonempty* even if no cavity exists in $\mathcal{M}$. The term *minimal covering* regards to the polyhedron with smallest possible number of wall triangles which envelopes the given cavity. We can construct a *minimal covering triangulation* such that it is empty if there exists no cavity.
Note: It is obvious that the empty triangulation implies the lack of cavities.

(7) *Oriented triangle, orientation of the triangle*. Suppose the vertices of all triangles are enumerated. If we look perpendicularly to the triangle's plane and find that the order numbers of vertices increase in counter-clockwise direction, then we say that the triangle is *positively oriented*. Otherwise, the triangle is *negatively oriented*. Obviously, a positively oriented triangle will turn into a *negatively oriented* one when viewed from the opposite side of the triangle's plane. For an example of triangle's orientation, see Fig. 3.

(8) The *enveloping* triangulation $\triangle\mathcal{M}$ for cavities (briefly *envelope triangulation*) of a protein (or a system of balls) $\mathcal{M}$ is either an empty set or it is such a set of *wall triangles* in $\mathcal{M}$ with coincident inward orientation that
   (i) $\triangle\mathcal{M}$ consists of a family of closed polyhedrons $\mathcal{P}_i$ whose facets represent inwardly oriented wall triangles such that $\operatorname{Int}\mathcal{P}_i \cap \operatorname{Int}\mathcal{P}_j = \emptyset$, for all $i \neq j$,
   (ii) any cavity point lies inside one of these polyhedrons,
   (iii) if any point from $\mathbb{R}_3 - \mathcal{M}$ is not in any cavity of $\mathcal{M}$ then it lies outside the triangulation $\triangle\mathcal{M}$.

(9) *The surface area and the volume of the cavity*. There are two basic definitions of the cavity volume and surface area. When the probe is being rolled over the van der Waals surface of the atoms, exposed to the interior of the cavity, its center creates a solvent accessible (SA) surface. The volume of the corresponding body is called SA volume. At the same time, the front of the probe sphere which is in touch with the van der Waals surface of the atoms, creates a molecular surface (MS), and the volume of the corresponding body is called MS volume of the cavity. We can use CAVE to calculate the SA surface area and volume of a cavity based on analytic equations, and calculate numerically the MS volume of the cavity.

Proposition 1. *If $\triangle C_1C_2C_3$ is a wall triangle in $\mathcal{M}$ then the planes of intersections of all pairs of these spheres intersect with the plane of the centers $C_1(x_1, y_1, z_1)$, $C_2(x_2, y_2, z_2)$, $C_3(x_3, y_3, z_3)$ in the single point (x, y, z) which is, consequently, the unique solution to the following linear set of equations*

$$(x_2 - x_1)x + (y_2 - y_1)y + (z_2 - z_1)z = \frac{1}{2}\left(r_1^2 - r_2^2 - x_1^2 + x_2^2 - y_1^2 + y_2^2 - z_1^2 + z_2^2\right),$$

$$(x_3 - x_1)x + (y_3 - y_1)y + (z_3 - z_1)z = \frac{1}{2}\left(r_1^2 - r_3^2 - x_1^2 + x_3^2 - y_1^2 + y_3^2 - z_1^2 + z_3^2\right),$$

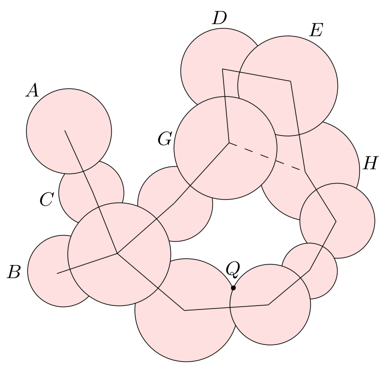

Fig. 4. Preliminary set of "wall" segments in a two-dimensional system.

$$\begin{vmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{vmatrix} = 0,$$

where r_i, $i = 1, 2, 3$ *are the radii of the spheres.*

In above, the first two equations represent the planes of intersections of spheres, and the third equation is the equation of the plane $C_1C_2C_3$.

3. Description of the 2D analogue of the enveloping triangulation algorithm

To show how the algorithm works, we will try to visualize some steps. It is extremely difficult to visualize on the plain paper the 3D pictures of the system of balls. In this section we introduce the 2D analogue of the algorithm. The overlapping spheres in 3D are mapped into 2D intersecting circles shown in Fig. 4. For more details we recommend the reader to consult our previous paper [45].

Detecting the cavity means to find out whether or not the given system contains any cavity. Recall that the existence of a cavity depends on the given radius of the probe sphere. The algorithm will not detect a cavity whose size is smaller than the probe sphere. *Localization of the cavity* means to determine all circles which surround the cavity (all spheres, in 3D case). *Quantitative analysis* in this case means to calculate the total length of the cavity border (the surface area in 3D system) and the area enclosed by the border (the volume in 3D).

Now we will try to construct some polygon in our system (Fig. 4) which must enclose the cavity *if it does exist*. It is obvious that the closed cavity can be formed only by intersecting circles. In the **first** step the algorithm finds all pairs of intersecting circles and connects their centers by segments of straight lines. In 3D case triplets of intersecting spheres are found and a triangle is constructed with the vertices at the centers of the three spheres, hence the term "triangulation" (see [45]).

Two types of segments are possible: (i) If at least one of the intersection points of any two circles does not belong to the interior of any other circle (e.g. the outer intersection point of circles D and E in Fig. 4) then we call the segment (e.g. the one connecting centers of D and E) a *wall* segment. (ii) Both intersecting points belong to the interior of some other circle (e.g. intersection points of circles G and E in Fig. 4 belong to circles D and H). Segments of this type are removed or are not drawn at all. Thus we obtain the *preliminary* list of wall segments (see Fig. 4). Intersection points which do not belong to the interior of any circle (e.g. point Q in Fig. 4) are being stored for later needs.

The **second** step is to optimize the preliminary list. It is evident that any segment which has only one neighbor cannot be a part of the cavity border (e.g. the segment connecting centers of circles A and C). These type of segments are called *lugs* and must be removed from the preliminary list. Removing one lug may create a new one. For example, after removing the segment connecting centers of A and C, another segment becomes a lug.

After the lugs are removed we start the **third** step — constructing the true envelope. We can choose an arbitrary wall segment from the optimized list (for example the southernmost) and assign an inward orientation (to the north) to it. Then we look for a neighbor of this segment at both sides (at all three edges of the triangle, in 3D). If there is only one neighbor then we add it to the new envelope. If there are more than one neighbors (circle H — segments HG and HE) then we choose the one which is maximally declined from the point of view of the inward orientation (segment HE). Eventually, we obtain an envelope shown in Fig. 5 (solid closed line).

In the **fourth** step we check whether the obtained envelope contains cavity. For this, the algorithm checks stored outer intersection points of all pairs of circles. If any of these points (for example point Q in Fig. 4 or 5) lies inside the envelope then a cavity does exist. We call these points Z-type points. This test is done by a special program module which can identify the position of any point in space with respect to the closed line (closed surface, in 3D).

The **last** step is to construct the minimum envelope consisting of circles which make the border of the cavity. If any Z-type point exists, we start a new triangulation construction with the line segment (the wall triangle in 3D) which corresponds to this point. This time, if there is more than one neighbor, we add the neighbor with *minimal* declination (e.g. segment HG for circle H — so, instead of the segments HE, ED, and DG, used before, we will use the dashed segment HG). Now we have obtained the *minimal* envelope.

Let us mention again that the algorithm for the 3D system is much more complicated and the interested reader will need to consult [45].

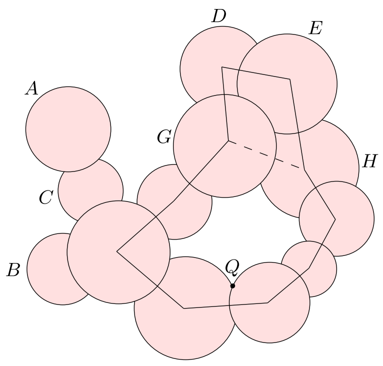

Fig. 5. The system after lugs cutting.

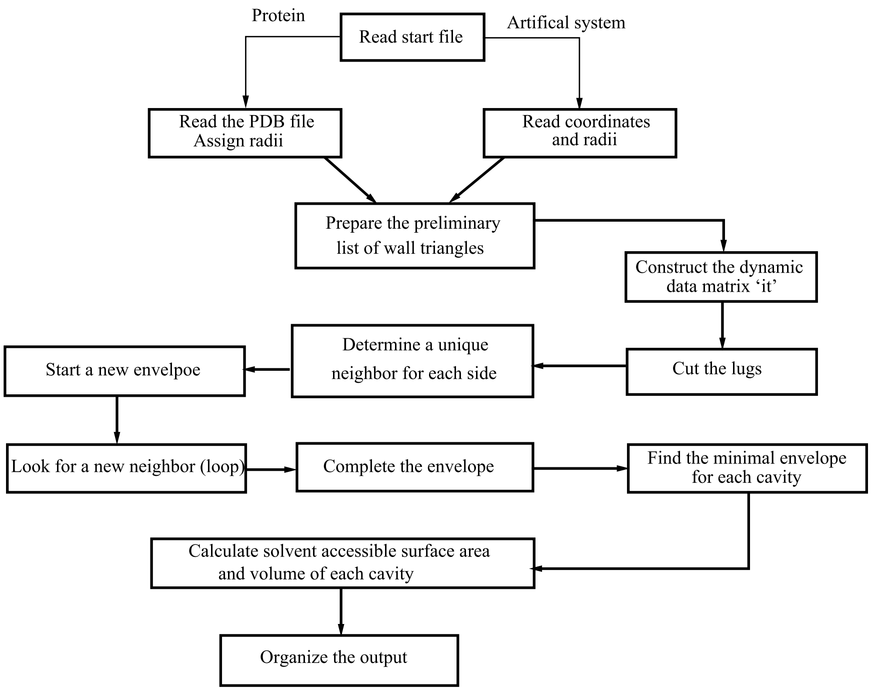
Protein
Read start file
Artifical system
Read the PDB file Assign radii
Read coordinates and radii
Prepare the preliminary list of wall triangles
Construct the dynamic data matrix 'it'
Start a new envelpoe
Determine a unique neighbor for each side
Cut the lugs
Look for a new neighbor (loop)
Complete the envelope
Find the minimal envelope for each cavity
Calculate solvent accessible surface area and volume of each cavity
Organize the output

Fig. 6. The general scheme of the package.

4. The package structure

CAVE was developed as a simple MAIN module which calls 37 subroutines and functions performing certain logical parts. In this section we describe the general structure of the package. The part of the program which calculates solvent accessible surface area and excluded volume has been described in [24]. The simplified scheme of the package is shown in Fig. 6.

The logical structure of the main module CAVE is simple. After reading the necessary input data, the algorithm first studies the neighborhood relations of the spheres/atoms. Here some useful lists are constructed which allow to work further only with the local subsets of atoms instead of checking neighborhood relations between distant atoms. This saves a considerable amount of time. Next, the relations of the triplets of atoms are studied, and the list of the *wall* triangles defined by the centers of the intersecting triplets of spheres as vertices is constructed. Then the triangles which do not have a neighbor at some side are recursively removed from this list. After this procedure, the enveloping triangulation is constructed. If any cavity is found then the spheres whose parts of the surface are exposed into the cavity are defined as boundary spheres, and are stored in the output file. Corresponding list of the wall triangles is stored too.

Afterwards the volume and surface area of the molecule with and without parts exposed to the cavities are calculated, as has been described in [24].

4.1. Input and output data

The user must fill the arrays of coordinates and radii as well as the radius of the probe sphere (see Subsections 4.2 and 4.3). Another parameter to be provided (variable ivdwr_set) is the integer number showing which set of atomic radii must be used. In total 4 different sets are included in the package. CAVE is an open code package, and the user is free to organize the input in his own way. However, since the proteins are our basic concern, here we provide a special subroutine pdbread which reads the amino acid sequence, the atom names and the coordinates from standard PDB file. Later the subroutine assigns the atomic radii according to the atom types.

Important note 1. Different authors often use different radii for the same atom even within the same set of radii. In order to make simple for the user to change the radius of any chosen atom we assign the radii in explicit manner through statements like this:

```
if(line(18:20).eq.'ALA'.and.line(14:15).eq.'N') r(ksf) = 1.7
      !! NH, trigonal NH
```

Here we put the radius of the atom N in alanine equal to 1.7 because it is a trigonal NH group. Recall that this statement corresponds to the format of the standard PDB file. The name of the PDB file, the value of the probe sphere, and the parameter for the set of radii are given in the main module explicitly. This makes an inconvenience that the program must be recompiled every time when these parameters are changed. However, (i) the compilation of CAVE takes only a couple of seconds; (ii) as mentioned above, the user can always organize the input in his own way. Let us mention that the extension .pdb in file name is not necessary. The program adds it automatically.

Important note 2. When reading from the PDB file the program automatically counts the number of the atoms (variable ksf). If the user wants to add some additional atoms to the system then he must increase the number of atoms accordingly, and to assign the coordinates and radii in the corresponding arrays. Alternatively, the data for these atoms can be added to the PDB file in an appropriate format.

The basic output is written in 5 ASCII files with extensions .log, .ats, .cav, .sph, and .tri. If the probe radius is equal to zero then 0 is added to the file name extension. For example, filename.cav0 instead of filename.cav.

name.log — contains the information about program run.
name.ats — contains the list of all spheres/atoms. The first three columns of each row are the x, y, and z coordinates of the atom's center, the 4-th column is the atomic radius enlarged by the radius of the probe sphere.
name.cav — contains the main information about cavities and their boundaries, and may be used by auxiliary programs which are working with the triangulation. Each row consists of 6 elements. The first column of the i-th row contains the order number of the first boundary triangle of the i-th cavity in the list of triangles name.tri. The second column is the number of boundary triangles for this cavity. The third column is the order number of the first boundary sphere in the list of spheres name.sph. The fourth column contains the number of cavity's boundary spheres (triangles' vertices). The fifth column contains the number of cavity's boundary spheres not belonging to the triangulation. The sixth column of the i-th row contains the number of cavities in the i-th segment. The last row of the file contains in the first and the third columns the auxiliary order numbers which allow to determine the number of triangles and spheres of the last cavity.
name.sph — the list of the boundary spheres. Each row consists of five columns: in the first column is the cavity's order number, in the second column is the sphere's order number with respect to the cavity, and in the third column is the sphere's order number in the main list of the atoms. Information about group is stored in columns four and five.
name.tri — the list of the boundary triangles of the cavities. Each row consists of six columns: in the first column is the order number, in the second column is the cavity's order number, in the third column is the triangle's order number with respect to the cavity, and in the columns 4–6 are the order numbers from the main list of atoms which correspond to the vertices of the triangle.

4.2. Important parameters

rwater — the radius of the probe sphere.
ks=20000 — maximum number of spheres (atoms).
kl=200 — maximum number of the neighbors of one sphere.
ki=2000000 — maximum number of neighborhood relations.
ka=2000000 — maximum total number of arcs and angles which arise from the intersections of local circles.
kan — maximum number of angles (cannot be more than twice the number of neighbors).
kt=60000 — maximum number of triangles.
maxtsn — maximum number of neighbor triangles at one side of the given triangle.
kn=3*maxtsn+7 — number of columns allocated for neighboring triangles information.
eps_deltas — accuracy level in the subroutine neighbors.
eps_nord_pole=1d-8 — the critical value for North Pole test. If the smallest distance from the North Pole to the surface of other atoms is smaller than eps_nord_pole, then the molecule is rotated by a random angle.
eps_deltat=1d-12 — the critical value for comparison of t_1 and t_2 coordinates of two circles in the (t, s) plane, when the intersection points are calculated in the circles_intersection subroutine.
eps_angle=1d-12 — the critical value for comparison of two angles in the function delete_equal. If two points on the circle are close to each other, they are declared to be the same, and only one point is left.

4.3. Important variables and data structures

spheres(ks,4) — contains the data on all atoms: spheres(i,1) $= x_i$, spheres(i,2) $= y_i$, spheres(i,3)$= z_i$, and spheres(i,4) $= r_i$ $(1 \leqslant i \leqslant k_s)$.
neighbors_number(ks) — neighbors_number(i) is the number of neighbors of the i-th sphere for.
index_start(ks) — index_start(i) is the order number of the index for the first neighbor of the i-th sphere in the array neighbors_indices.
neighbors_indices(ki) — contains the indices of neighbors for all atoms.
sphere_local(kl,4) — contains coordinates and radii of the i-th sphere and its neighbor spheres: sphere_local(j,1) $= x_j$, sphere_local(j,2) $= y_j$, sphere_local(j,3) $= z_j$, and sphere_local(j,4)$= r_j$ for $1 \leqslant j \leqslant k_l$.

`circles(kl,4)` – the data structure of the circles in (t,s) plane, see [24]. `circles(j,1)` $= t_j$ and `circles(j,2)` $= s_j$ are the coordinates of the center of j-th circle, `circles(j,3)` $= r_j$ is its radius, and `circles(i,4)` $= \pm 1$ shows the orientation of the circle, $1 \leqslant j \leqslant k_l$.
`arcs(ka,3)` – holds the circular arcs $C^i_{j,\lambda}$ composing the boundary $B(\Omega_i)$ of the (t,s) domain Ω_i [24]. Along these arcs the line integrals are calculated: `arcs(k,1)` $= ic_k$ is the index of the k-th arc circle, `arcs(k,2)` $= \alpha_k$ is the starting k-th arc's angle, and `arcs(k,3)` $= \delta_k$ is the oriented arc's angle, so the arc's end point corresponds to the angle $\beta_k = \alpha_k + \delta_k$.
`arcsnew(ka,3)` – is a help array with the same structure as the array `arcs`.
`angles(ka)` – is a help array to remember all angles corresponding to all intersection points of some circle with all other circles in (t,s) plane.
`av(2)` – `av(1)` is the value of surface integral of the volume calculation method and `av(2)` is the value of surface integral of the surface area calculation method [24].
`vertices(kt,3)` – the coordinates of the triple intersection points which do not lie inside of any sphere of $\mathcal{M}$.
`ivertices(kt,2)` – contains corresponding wall triangle for the intersection point from the list `vertices`.
`it(ka,kn)` – a help list of triangles in the given structure:

```
[i,j,k,n11,n12,...,n1maxtsn,...,
        n31,n32,...,n3maxtsn,n1,n2,n3,status].
```

`ntout(kt,6)` – the list of neighbors with maximal declination for each side. The first 3 columns correspond to the positive, and the last 3 columns correspond to the negative orientation.
`itel(kt,5)` – auxiliary list of triangles.
`ittt(kt,8)` – the dynamic list of triangles in the actual segment with some side unoccupied. The first 3 columns are the triangle's vertices spheres indices, columns 4–6 contain corresponding neighboring triangles order numbers for sides 1–3 of the triangle. The 7-th column contains the triangle's orientation, and the last column contains its order number in the list `it`.
`iit(kt,3)` – the final list of triangles which compose the triangulation. The number of these triangles is `ntt`.
`isph(ks)` – the final list of sphere indices for all cavities.
`icav(kt,6)` – the final information about cavities: `icav(i,1)` is the starting position in `iit` and `icav(i,2)` is triangles number for i-th cavity; `icav(i,3)` is the starting position in boundary and internal spheres list for the i-th cavity; `icav(i,4)` and `icav(i,5)` are the boundary and internal spheres numbers for the i-th cavity; `icav(i,6)` is the cavities number of the i-th segment.

4.4. Brief description of subroutines and functions

The following subroutines and functions have been used in ARVO package. They are described in [24], and we will omit their description in this paper:

`make_neighbors`, `neighbors`, `North_Pole_test`, `spheres_rotation`, `areavolume`, `local_spheres`, `make_ts_circles`, `circles_to_arcs`, `new_arcs`, `mysort`, `mydsort`, `circles_intersection`, `circle_in_circle`, `point_in_circle`, `delete_equal`, `fract`, `avintegral`.

Only the subroutine `make_neighbors` and the function `neighbors` are used in the triangulation method, all other subroutines and functions are used for the volume and the surface area evaluation.

The function `delete_cavities_internal_arcs` is used in the modified `areavolume` procedure of the evaluation of the surface area and the volume. All calculations are done twice. In the second run only arcs, which correspond to the outer boundary of molecule are used for calculation of the integrals (see [24]).

The following subroutines and functions correspond to the triangulation construction part of the program:

`pdbread(fname,x,y,z,r,ksf,atres,nowrite,ivdwr_set)` – reads the PDB file of a given protein, and prepares the information about the positions and sizes of all atoms.
`istart(str)` – returns position of the first non-blank character in `str`.
`istop(str)` – returns position of the last non-blank character in `str`.
`make_triangles` – the main function which calls all subroutines and functions below.
`intersect(ia,ib,ic,ik,in)` – checks the intersection of two vectors of indices `ia` and `ib`. It returns the number of common indices, and the lists `ic=[ind1,ind2,...]` – sorted common indices in vectors `ia` and `ib`. Positions of common indices in vector `ia` are written to the list `ik=[ind1a,...]`. `in=[ind1b,ind2b,...]` is the list of positions of common indices in vector `ib`. We suppose that there are no multiple indices in vectors `ia` and `ib`.
`internal_triangles(it,itel,itelp,is,spheres,nit,nt,ntp,kt,ks,kn)` – prepares the list `is` of triangles from the list `it` of the length `nit` which are inside the triangles' set `itel` of the length `nt`, and will be removed. Returns the number of internal triangles.
`internal_point(p,itel,spheres,kt,ks,ntel)` – returns value 1 if the point `p=(p1,p2,p3)` is inside the triangulation represented by triangles from `itel` with the vertices at the centres of `spheres(itel(i,j))`, $j = 1, 2, 3$. Returns 0, otherwise.
`myisort(ind,n_ind)` – sorts the list `ind` of the length `n_ind` in ascending order.
`idelete_equal(ind,indnew,kt,n_ind)` – deletes multiple indices in the sorted list `ind`.
`delete_triangle(i,it,kt,maxtsn,kn,iflag)` – makes all necessary changes in the matrix `it` for deleting the i-th triangle.
`find_other_bindings(ittt,ntout,it,kt,nt,kn,maxtsn)` – finds the neighbors from `ittt` (if such exist) to all unoccupied sides of the triangle at position `nt` in `ittt`. If some triangle from `ittt` is the neighbor to the given triangle, its `ittt` information is changed too – new neighbor `nt` is added at the corresponding unoccupied side position.

`make_orientation(i,is,io,k,it,kt,kn,maxtsn)` — calculates the orientation of the k-th triangle, if it is connected to the i_s-th side of the i-th triangle which has an orientation i_o.
`angles_calculation(i,is,io,lat,spheres,it,angle,kt,ks,kl,kn,na)` — calculates the angles between oriented triangles from the list `lat` (neighbors to the i_s-th side) and the i-th triangle whose orientation is i_o. Angles are written to the list `angle`.
`angles_inner(lat,spheres,angle,ks,kl,na)` — this subroutine calculates angles between inward oriented triangles from the second row up to the row `na+1` of the list `lat` and the first triangle of the list `lat`, with common side `lat(1,4)`. Angles are written to the list `angle`.
`cross(va,vb,vc)` — vector `vc` is the vector (cross) product of two vectors `va` and `vb` of the length 3.
`dot(va,vb)` — returns the scalar (dot) product of two vectors `va` and `vb` of the length 3.
`enorm(va)` — returns the Euclidean norm of the vector `va` of the length 3.
`det3(am)` — returns the determinant of 3×3 matrix `am`.
`gauss3(am,vb,vsol)` — vector `vsol` is the solution to the 3×3 system of linear equations with system matrix `am` and the right-hand side vector `vb`.

5. Notes on possible parallelization

`CAVE` is a fast program. The molecule of thousands of atoms is processed on modern computers within a few seconds. In this sense there is no direct need for parallelization when the program is used for single or occasional calculations. This is the usual case, and that is why `CAVE` was developed as a sequential program. If there is need to use `CAVE` for massive calculations of many molecules then the simplistic parallelization method can be applied when the same code is sent to the different nodes with different input files. However, there is a special case when one needs to calculate the cavities in numerous conformations during the Monte Carlo or molecular dynamic simulations. Let us mention that it is easy to incorporate `CAVE` into various simulation packages. For this case currently we don't have a clear algorithm for intrinsic parallelization of the code. One possible way is to send this code to a special node when others are busy with calculating the energy function. `CAVE` is an open code and the interested user can find many ways to parallelize certain parts of it. For example one can parallelize the process of searching for the neighbors of a given triangle.

6. Running the program

The code is written in standard Fortran language and it can run on any platform where the Fortran77 compiler is available. We have been using the program on Linux platform. To use the program the user must take some simple steps:

Uncompress the file `CAVE.tar`. A directory `CAVE` will be created which contains the following items:

- The file `CAVE.f` which is the program itself.
- The file `README` which provides some useful information.
- The PDB file which is used for testing the program.
- The directory `TEST` which contains output files for test run.

After this just compile the code and run. No makefile is necessary.

7. Example

The program has been tested to calculate the cavities in many proteins from PDB. The results are published in [45]. As an example, we bring here the results of calculation for the protein 2ptn from the PDB. All output files are included in the directory `TEST` which is the part of this package. After running the provided version, the output files must coincide with ones in `TEST` directory.

Appendix A shows the screen output. One can see that there are 17 cavities in this protein.

Acknowledgements

This work was supported by grants NSC 96-2112-M-008-021-MY3, 96-2911-M-001-003-MY3, 98-2627-B-008-004, AS-95-TP-A07, and National Center for Theoretical Sciences in Taiwan, and by VEGA grant 1/0819/08 of Slovak Grant Agency.

Appendix A. Screen output of the exemplary run

```
The probe radius    1.20000005
 The protein name 2ptn
  Total number of atoms:  1629  allocated:  20000
  Rashin set
 Total number of neighbors:  62054
 Neighbors total:    62054 allocated:  2000000
 Maximum:  66     allocated:  200
 Number of the preliminary triangles:  2236
 Number of vertices:  2278
 2197 active triangles before lugs cutting.

 After lugs cutting:  2197

    1. south:   1443  1444  1453
 340 internal points
 Clusters:  26   Relations:  11
    1. segment contains 17 cavities.

    11 triangles were deleted from the list IT.
   326 internal triangles were deleted from the list IT.

 340 internal triangles vertices!
    1. segment contains cavity!
 Triangles number after     1. segment:     0

 Number of segments:
   Total:    1          Deleted:    0         Cavity enveloping:    1
    1. cavity segment contains   340 triangles.
 Total number of cavities is  17.
 Total number of triangles is  340.
 Total number of spheres is  203.
 340
  Number of atoms:  1629
 There exists cavity!
 Inaccessible volume:       37436.2361
 Accessible surface area:   9315.41184
 Cavities volume:           32.4669432
 Surface area of cavities:     181.732348
 1. boundary volume:       563.196503
 1. boundary surface area:   389.741749
 1. cavity volume:           0.0590741273
 1. cavity surface area:     1.10512934
 2. boundary volume:       515.125257
 2. boundary surface area:   375.817426
 2. cavity volume:           0.00464992906
 2. cavity surface area:     0.194346908

  ...

 16. boundary volume:       1628.63036
 16. boundary surface area:   862.982504
 16. cavity volume:           6.12086952
 16. cavity surface area:     36.522405
 17. boundary volume:       614.397572
 17. boundary surface area:   423.464562
 17. cavity volume:           0.0684193718
 17. cavity surface area:     1.43204525
STOP End statement executed
```

References

[1] B.P. Schoenborn, J. Mol. Biol. 45 (1969) 297.
[2] R.F. Tilton, I.D. Kuntz, G.A. Petsko, Biochemistry 23 (1984) 2849.
[3] R.K. Wierenga, M.E.M. Noble, R.C. Davenport, J. Mol. Biol. 224 (1992) 1115.
[4] U. Sreenivasan, P.H. Axelsen, Biochemistry 31 (1992) 12785.
[5] J.T. Kellis Jr., K. Nyberg, D. Sali, A.R. Fersht, Nature 333 (1988) 784.
[6] J.T. Kellis Jr., K. Nyberg, A.R. Fersht, Biochemistry 28 (1989) 4914.
[7] V.G.H. Eijsink, B.W. Gijkstra, G. Vriend, J.R. van der Zee, O.R. Veltman, B. van der Vinne, B. van der Burg, S. Kempe, G. Venema, Protein Eng. Des. Sel. 5 (1992) 421.
[8] A.E. Eriksson, W.A. Baase, J.A. Wozniak, B.W. Mattheus, Nature 355 (1992) 371.
[9] A.E. Eriksson, W.A. Baase, B.W. Mattheus, J. Mol. Biol. 229 (1993) 747.
[10] D.G. Lambright, S. Balasubramanian, S.M. Decatur, Biochemistry 33 (1994) 5518.
[11] B. Lee, F.M. Richards, J. Mol. Biol. 55 (1971) 379.
[12] A. Shrake, J.A. Rupley, J. Mol. Biol. 79 (1973) 351.
[13] C. Chothia, Nature 248 (1974) 338.
[14] F.M. Richards, Ann. Rev. Biophys. Bioeng. 6 (1977) 151.
[15] M.L. Connolly, Science 221 (1983) 709.
[16] M.L. Connolly, J. Appl. Cryst. 16 (1983) 548.
[17] T.J. Richmond, J. Mol. Biol. 178 (1984) 63.
[18] M.L. Connolly, J. Am. Chem. Soc. 107 (1985) 1118.
[19] K.D. Gibson, H.A. Scheraga, Mol. Phys. 62 (1987) 1247.
[20] K.D. Gibson, H.A. Scheraga, J. Phys. Chem. 91 (1987) 4121.
[21] M. Petitjean, J. Comp. Chem. 15 (1994) 507.
[22] M. Totrov, R. Abagyan, J. Struct. Biol. 116 (1996) 138.
[23] Sh. Hayryan, C.-K. Hu, J. Skřivánek, E. Hayryan, I. Pokorný, J. Comp. Chem. 26 (2005) 334.

[24] J. Buša, J. Džurina, E. Hayryan, Sh. Hayryan, C.-K. Hu, J. Plavka, I. Pokorný, J. Skřivánek, M.-C. Wu, Comp. Phys. Comm. 165 (2005) 59.
[25] A.A. Rashin, M. Iofin, B. Honig, Biochemistry 25 (1986) 3619.
[26] S.J. Hubbard, K.-H. Gross, P. Argos, Protein Eng. Des. Sel. 7 (1994) 613.
[27] S.J. Hubbard, P. Argos, Protein Sci. 3 (1994) 2194.
[28] M.L. Connolly, J. Mol. Graph. 11 (1993) 139.
[29] L. Zhang, J. Hermans, Proteins 24 (1996) 433.
[30] A.A. Rashin, B.H. Rashin, A. Rashin, R. Abagyan, Protein Sci. 6 (1997) 2143.
[31] J. Liang, H. Edelsbrunner, P. Fu, P.V. Sudhakar, S. Subramaniam, Proteins: Struct. Funct. Genet. 33 (1998) 1.
[32] J. Liang, H. Edelsbrunner, P. Fu, P.V. Sudhakar, S. Subramaniam, Proteins: Struct. Funct. Genet. 33 (1998) 18.
[33] S. Chakravarty, A. Bhinge, R. Varadarajan, J. Biol. Chem. 277 (2002) 31345.
[34] M.A. Williams, J.M. Goodfellow, J.M. Thornton, Protein Sci. 3 (1994) 1224.
[35] D. Bakowies, W.F. van Gunsteren, Proteins 47 (2002) 534.
[36] M.L. Connolly, J. Appl. Cryst. 18 (1985) 499.
[37] N. Akkiraju, H. Edelsbrunner, Discrete Appl. Math. 71 (1996) 5.
[38] P.F.B. Goncalves, H. Stassen, J. Chem. Phys. 123 (2005) 214109.
[39] L. Willard, A. Ranjan, H. Zhang, H. Monzavi, R.F. Boyko, B.D. Sykes, D.S. Wishart, Nucl. Acids Res. 31 (2003) 3316.
[40] R.A. Laskowski, J. Mol. Graph. 13 (1995) 323.
[41] J. Liang, H. Edelsbrunner, C. Woodward, Protein Sci. 7 (1998) 1884.
[42] M.D. Collins, G. Hummer, M.L. Quillin, B.W. Matthews, S.M. Gruner, Proc. Natl. Acad. Sci. USA 102 (2005) 16668.
[43] M.D. Collins, M.L. Quillin, G. Hummer, B.W. Mattheus, S.M. Gruner, J. Mol. Biol. 367 (2007) 752.
[44] B.W. Matthews, L. Liu, Protein Sci. 18 (2009) 494.
[45] J. Buša, Sh. Hayryan, C.-K. Hu, J. Skřivánek, M.-C. Wu, J. Comp. Chem. 30 (2009) 346.
[46] http://www.rcsb.org/pdb/home/home.do.
[47] F. Eisenmenger, H.U.E. Hansmann, Sh. Hayryan, C.-K. Hu, Comp. Phys. Comm. 138 (2001) 192.
[48] F. Eisenmenger, H.U.E. Hansmann, Sh. Hayryan, C.-K. Hu, Comp. Phys. Comm. 174 (2006) 422.
[49] C.-Y. Lin, C.-K. Hu, H.U.E. Hansmann, Proteins 52 (2003) 436.
[50] R.G. Ghulghazaryan, Sh. Hayryan, C.-K. Hu, J. Comp. Chem. 28 (2007) 715.
[51] H.L. Chen, C.-c. Hsu, M.-H. Viet, M.S. Li, C.-K. Hu, C.-H. Liu, F. Luk, E. Chang, A. Wang, M.-F. Hsu, W. Fann, R. Chen, Proteins: Struct. Funct. Bioinform. (2010), doi:10.1002/prot.22823, in press.
[52] A.-J. Li, R. Nussinov, Proteins 32 (1998) 111.