Finding Cavities in a Molecule

Lukáš Jirkovský
Supervised by: Martin Maňák and Ivana Kolingerová

Department of Computer Science and Engineering
University of West Bohemia
Pilsen / Czech Republic

Abstract

In recent years, biochemistry is gaining more and more attention. The research involves analysis of large molecules, such as proteins. One of many properties we can study are molecular cavities, where cavity is understood as a free space inside a molecule. As we are usually interested only in a certain subset of cavities, the common approach is to use a spherical probe of a given radius to find the cavities. The probe can be imagined as a sphere which we try to slip through the molecule.

In this paper we discuss an algorithm to find inner cavities in a molecule for a given size of the probe. The algorithm has a preprocessing stage where an additively weighted Voronoi diagram of a molecule is computed. This diagram is then used to accomplish the task of finding cavities for varying probe sizes.

The algorithm presented proved to be very fast for a probe with a variable size. The implementation shows it is able to operate in real-time even on large structures, such as the Thermus Thermophilus 70S ribosome (PDB ID 3OH5, approximately 87 000 atoms).

Keywords: molecular cavity, molecule analysis, additively weighted Voronoi diagram

1 Introduction

A protein structure can be very complicated. This includes depressions on the molecular surface (often called pockets) and empty space inside the molecule. This empty space can form tunnels and cavities which are not connected to the surface (inner cavities). It can be represented as a union of spheres (Figure 1). When searching cavities, we usually introduce a spherical probe which we try to slip through the molecule. This is useful, because it allows us to specify the minimal radius of the cavity (the radius is usually measured in angstroms [Å], where $1\text{Å} = 10^{-10}$ m).

The protein structure affects the behavior of protein interactions. These interactions are a part of biological processes. This has led to the study of protein structures and using the knowledge of these structures to design new drugs. A lot of research has been done on so-called active sites, which are the places where the proteins can mutually interact. These sites are generally found on the surface of a protein. However, as protein molecules are not perfectly stationary, some of the inner cavities may become accessible from the outside at some point of time and become a part of an active site. Therefore, it is desirable to identify these cavities.

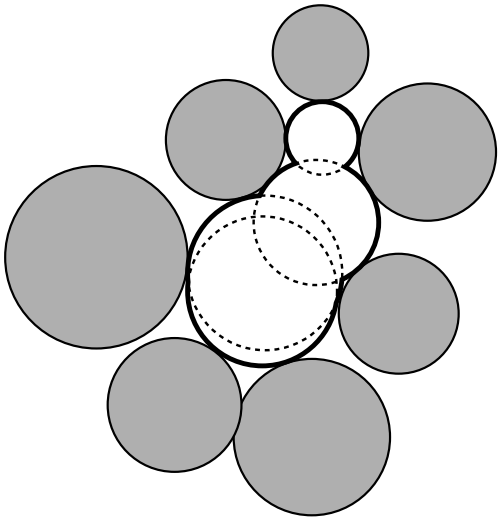


Figure 1: Cavity formed by gray atoms.

The inner cavities can also hold buried residui of other molecules, such as water [21, 23]. This is important as it can influence the stability of protein structure [23].

The existing cavity searching algorithms are proficient in searching the cavities, however, they are slow when it comes to a variable probe size, because they need to rerun all computations when the probe size changes. Our method improves the run time when different radii of probe are used by moving some of the computational complexity into the preprocessing stage. The preprocessing allows the algorithm to operate very quickly on large molecules (about 100 000 atoms).

Our algorithm has been implemented as a Java library, because in the future we would like to use it as a plugin in CAVER [2], a software tool for protein analysis and visualization.

In Section 2 the current methods used for finding inner cavities will be presented. In Section 3 the necessary the-

oretical background of the ordinary Voronoi diagram and the additively weighted Voronoi diagram with its dual representation will be given. In Section 4 the algorithm using the additively weighted Voronoi diagram for finding inner cavities will be described. In Section 5 we will present results of our implementation. Finally, Section 6 summarizes our findings.

2 State of the Art

The algorithm to find cavities in a molecule using alpha-shape [6] has been presented by Liang et al [15, 16]. First, the radii of all atoms are increased by the radius of probe to obtain the "solvent accessible" model [14, 15] – the model which is accessible by the given probe. Then a regular triangulation of atom centers is computed and the cavities are searched within the weighted alpha shape.

The weighted alpha shape is a subset of the regular triangulation. This subset is determined by the parameter α, which can be imagined as the radius of a probe is rolling over the molecular surface. This probe removes edges and thus collapses some of the tetrahedra.

The weighted alpha shape used in this algorithm is constructed by removing edges which are dual to Voronoi vertices (see Section 3) outside of the enlarged molecule atoms. This corresponds to using a probe of a zero radius. It has been shown [5] that voids in the alpha shape correspond to cavities in a molecule. The cavities are identified by searching the tetrahedra that were removed when the alpha-shape was constructed. Only the tetrahedra which were completely enclosed in the alpha shape are considered. The downside of this algorithm is that the alpha-shape has to be rebuilt whenever the probe size is changed.

Several algorithms using a regular grid exist [7, 8, 22]. The first step of these algorithms is construction of a regular grid, where each grid cell stores information whether it lies inside or outside of the molecular surface. The grid is then processed to identify cavities. The processing depends on the algorithm.

The algorithm described in [7] identifies cavities by checking the neighboring cells of an empty cell in the direction of each axis. The algorithm described in [8] identifies cavities by scanning the grid in the direction of each axis and diagonals. Another approach was presented by Tripathi and Kellogg [22] as the VICE algorithm (Vectorial Identification of Cavity Extents). This algorithm constructs a set of vectors from every empty grid cell. The visibility of the molecular surface for each vector is determined. The visibility describes how much the grid cell is enclosed within the molecule. The inner cavities are formed by points fully enclosed within the molecule.

Our algorithm uses the additively weighted Voronoi diagram in the preprocessing stage. This diagram has many other uses apart from searching cavities, such as finding pockets [9] or determining how spherical the molecule is [13]. The algorithm for the construction of the diagram has been described in [11, 12, 18]. To improve the speed of the edge-tracing algorithms for the diagram construction, spatial filtering is often used [17, 24, 3].

3 Geometric Background

To fully understand the idea of the algorithm, some of the properties of Voronoi diagrams need to be mentioned first. We will begin with the description of an ordinary Voronoi diagram. Then a basics of the additively weighted diagram and its dual representation called quasi-triangulation will be given.

3.1 Voronoi Diagram

A Voronoi diagram [19] is a decomposition of the space determined by a set of points. These points are often called *generators*. We can describe the Voronoi diagram as a tessellation of space, such that for every generator we define a *Voronoi region*, consisting of all points in space which have the smallest Euclidean distance to its generator. That means for each generator p_i there exists a Voronoi region R_i, such that:

$$R_i = \{x : \|p_i - x\| \leq \|p_j - x\|, \forall j \neq i\}$$

where $i, j \in \{1, 2, \ldots n\}$ and n is the number of generators. For an example of a two-dimensional Voronoi diagram see Figure 2.

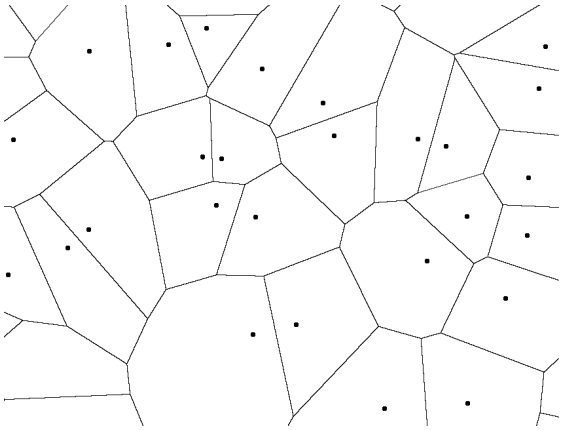


Figure 2: Two-dimensional Voronoi diagram.

Region boundaries are called *Voronoi edges*. It can be seen that in the two-dimensional Voronoi diagram each edge is shared among neighboring regions. In the three-dimensional Voronoi diagram the Voronoi edge is shared among three regions. Each point of an edge is equidistant to the edge generators, because the points lying on the edge have to meet the definition of Voronoi region for all regions containing the edge.

The point where multiple edges meet is called a *Voronoi vertex*. A Voronoi vertex can be also defined as the endpoint of the Voronoi edge. In two dimensions we can

also say that the Voronoi vertex is the point shared among three Voronoi regions. Similarly, in the three-dimensional Voronoi diagram the Voronoi vertex is a point shared by four regions.

The Voronoi diagram is usually seen as a graph of the Voronoi edges and the Voronoi vertices. The diagram is often stored in its dual representation, called the Delaunay triangulation. Edges in the triangulation are created by connecting the closest pair of generators, i.e., generators that share an edge. As a result we obtain a triangle in the Delaunay triangulation for each Voronoi vertex in two dimensions and a tetrahedron in three dimensions.

3.2 Additively Weighted Voronoi Diagram

It can be seen that all generators affect the resulting Voronoi diagram equally when the Euclidean metric is used. However, this is not always desired, consequently many variations exist. One of these variations is the additively weighted Voronoi diagram, also known as the Apollonius diagram or the Euclidean Voronoi diagram of spheres.

Each generator in the additively weighted Voronoi diagram has a weight. Unlike the ordinary Voronoi diagram, where generators are imagined as points, generators in the additively weighted Voronoi diagram are imagined as circles in 2D or spheres in 3D with the radius equal to their weight.

An *additively weighted distance*, which is defined as the Euclidean distance minus the weight of the generator [19, 11], is used instead of the Euclidean distance. Let $i, j \in \{1, 2, \ldots n\}$, where n is the number of generators. We define the Voronoi region R_i for a generator p_i with a weight w_i as:

$$R_i = \{x : \|p_i - x\| - w_i \leq \|p_j - x\| - w_j, \forall j \neq i\}$$

Due to the used distance, the Voronoi region can be interpreted as a set of points closest to the sphere p_i with the radius w_i. This has a few important implications. The edges are no longer necessarily line segments or half-lines. For an example of the additively weighted Voronoi diagram in two dimensions, see Figure 3.

Unfortunately, some anomalies can occur in the additively weighted Voronoi diagram. It is possible that two generators define more than one edge if there is a generator with a small weight among the generators with a big weight assigned (e.g., a small sphere among three large spheres). This small generator can split the edge into two parts which are connected by the edges generated by the small generator and the big generators. The generator with a small weight can also generate an elliptic edge (Figure 4). For more details about these anomalies see [12].

Similarly to the ordinary Voronoi diagram, a dual representation exists. This representation is called a quasi-triangulation [12]. This dual representation is not a valid triangulation (hence quasi-), because the anomalies break

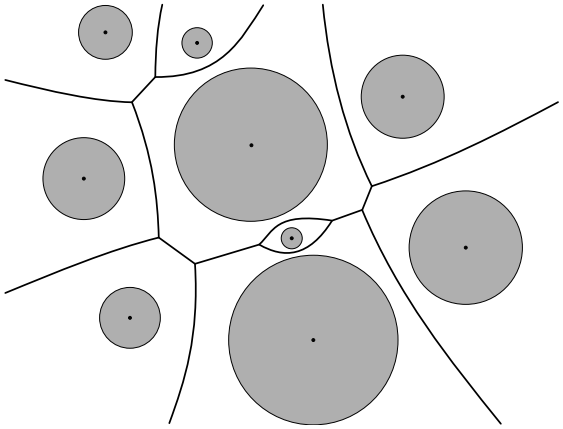


Figure 3: 2D additively weighted Voronoi diagram.

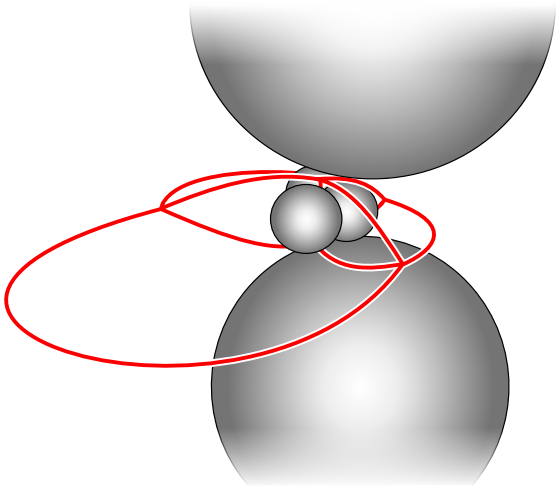


Figure 4: Eliptic edges in the 3D additively weighted Voronoi diagram.

the validity of the triangulation. For example, if an elliptic edge exists in the three-dimensional additively weighted Voronoi diagram, it may not be possible to construct a tetrahedron, because no Voronoi vertex lies on the edge. A triangle is stored in this case. Another example of an anomaly is that a split edge in three dimensions is represented by multiple tetrahedra with two or more common triangles.

4 Proposed method

Our method finds inner cavities in a molecule. Its input is a set of molecule atoms, which are represented as spheres. The output is a set of subgraphs of the graph representing the additively weighted Voronoi diagram. Each subgraph forms a connected cavity, as the probe can be slipped along the subgraph.

We can leverage the additively weighted Voronoi diagram for computing a diagram of molecules. The additively weighted Voronoi diagram, where the generators are

atoms of a molecule with the weight given by their radius, has an important feature, which can be used to quickly find cavities using a probe of a given size. It is the fact that an edge in such a diagram can be considered as an optimal path among the generators.

A very simple proof is that if we moved away from one generator, we would always get nearer to at least one other generator. We can use this knowledge to slip a spherical probe through the molecule. If the probe was not moved along the edge, it would be possible that the probe "hits" one of the generators, while there was a space between the probe and another generator. Moving the probe along the Voronoi edge ensures that the distance to the nearest generators, which are likely to collide with the probe, is equal.

The algorithm is built on the fact that the probe can pass among the generators if and only if it can pass on the edge through the narrowest place. As the algorithm does not need the position where this narrowest place is, only the radius of the probe which can pass among the generators is stored. We will call the radius of the probe on the narrowest place a *bottleneck*. We use the bottleneck to determine whether the probe can slip along an edge. Figure 5 illustrates the idea in two dimensions. This idea also applies in three dimensions.

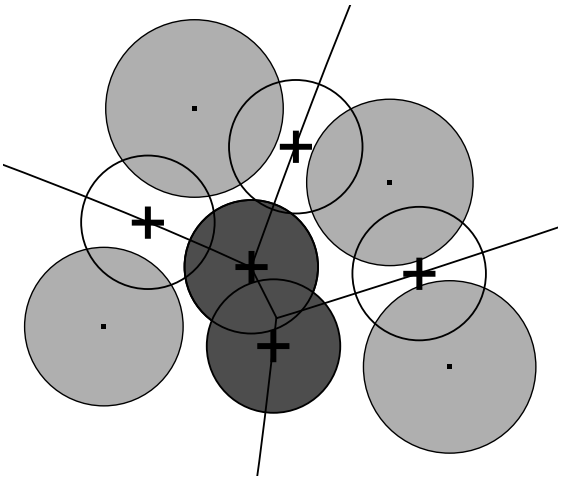


Figure 5: We try to slip a probe among the generators (light-gray circles). The position of the narrowest place is marked by a cross. If the probe can pass through, it is illustrated as a dark circle. If the probe cannot pass through the edge, it is shown as an empty circle.

Our algorithm consists of the following steps:

1. Creation of the additively weighted Voronoi diagram of molecule atoms.
2. Computation of the bottlenecks.
3. Sorting the Voronoi vertices using the distance from the vertex to its generators.
4. Traversal of a diagram using a graph traversal algorithm.

Steps 1–3 are done in a preprocessing step.

4.1 Preprocessing

The algorithm starts with the preprocessing stage. In this stage, the additively weighted Voronoi diagram of atoms is computed. Next, edge bottlenecks are computed, as described later. We also add a boolean flag "is outer" which we will set on for all Voronoi vertices which have at least one edge extending to the infinity. This flag will help us to discover an outer cavity. Finally, Voronoi vertices are sorted by the distance from a vertex to the surface of its generators.

The bottleneck can occur anywhere on the Voronoi edge. To compute the bottleneck we first compute a point that has the minimal distance from the generators defining this edge. Next, we must check whether this point lies on the edge, for which we want to obtain the bottleneck.

This check is done by defining vectors from the center of the generator with the smallest weight to the edge endpoints and a vector to the point with the minimal distance to the generators. The generator with the smallest weight needs to be used because of the elliptic edges. If the vector to the tested point lies within the angle between vectors to the edge endpoints, we store its distance to the surface of the edge generators as the bottleneck for the edge. Otherwise we use the endpoint for which the bottleneck is smaller. Figure 6 illustrates the possible positions of bottlenecks. For details see [11, 18].

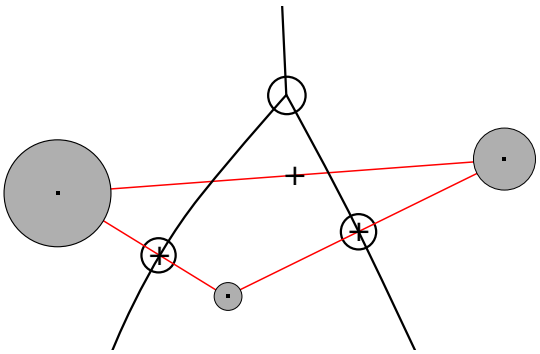


Figure 6: Bottlenecks. Crosses are positions of points with the minimum distance from the edge generators, black circles are actual positions of bottlenecks.

Now, it would be already possible to find the cavities by traversing the graph defined by the Voronoi vertices and edges and checking whether the bottleneck is larger than the probe size. However, this would be very inefficient, because it would require visiting all vertices every time the size of the probe changes.

The efficiency can be greatly improved by sorting the vertices. We store the distance from the vertex to its generators with other vertex information. We call this distance a *maximal bottleneck*. The maximal bottleneck allows us to quickly decide whether the given probe can fit among the generators. Its value is always greater or equal to the maximum of the bottlenecks. This ensures that we cannot

skip any vertex which has large enough bottlenecks during finding cavities.

The last step of preprocessing stage is sorting the vertices using the now obtained maximal bottleneck. The reason to include the distance among the generators and the vertex itself is that we need to take the space among the vertex generators into account. This is important, because the bottlenecks of all edges may be too small, but there is still enough space among the generators of the vertex (see Figure 7).

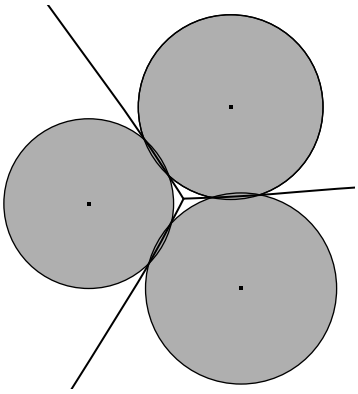


Figure 7: The empty space among generators in 2D. This may occur in three dimensions, too.

4.2 Finding Cavities

The input of this part of the algorithm is the additively weighted Voronoi diagram, represented as a graph of Voronoi vertices and edges and a sorted order of vertices from the preprocessing stage.

The search for cavities is performed as a graph traversal. We start from the vertex with the largest maximal bottleneck. First, we check whether the maximum of bottlenecks of the starting vertex is greater than the probe radius. If this condition is met, we recursively traverse the graph. During the traversal we mark the visited vertices, so we do not visit a vertex multiple times. The traversed part of the Voronoi diagram forms a cavity.

When it is not possible to continue with the traversal, we move to the vertex with the second largest bottleneck and start the traversal from there if possible. We repeat this procedure until a the first vertex is found, for which its maximal bottleneck is large enough for probe to slip through. We can safely stop the algorithm here, as all cavities were found. Since vertices are sorted using their maximal bottleneck, none of the remaining vertices has any bottleneck bigger than the probe radius.

Now, we have found all cavities in the molecule. However, they include the outer void, too. To remove it, we will utilize the flag "is outer" introduced earlier. After the graph has been traversed, we check the "is outer" flag for each visited vertex. If any of the visited vertices has the flag set to true, we disregard this part of the graph, because it is connected to the outer space, hence it cannot be an inner cavity. We can also modify the algorithm to identify first the outer cavity and then search cavities only within the remaining vertices. The advantage of such a modification is that it is possible to use a probe of a different size to remove the outer cavity, which would allow us to use the algorithm to find pockets on the surface of a molecule, too. This is similar to the creation of β-shape [10] prior to the search if the quasi-triangulation was used instead of the additively weighted Voronoi diagram.

5 Experiments and Results

We have developed a Java library implementing the algorithm and a simple visualization tool, the output of which can be seen in Figure 8. The visualization approximates the shape of a cavity by putting spheres into the Voronoi vertices forming the cavity. The implementation uses the awVoronoi library [1] for computing the quasi-triangulation. For that reason the algorithm described had to be converted to the dual representation.

The system used for experiments was a PC with Core i7 920 CPU (four cores at 2.7GHz with Hyper-Threading) and 12 GB RAM running Arch Linux 64bit. In all measurements, we evaluated our algorithm twelve times with the given parameters, removed the shortest and longest measured time and finally computed average of the ten remaining times.

5.1 Algorithm Run Time

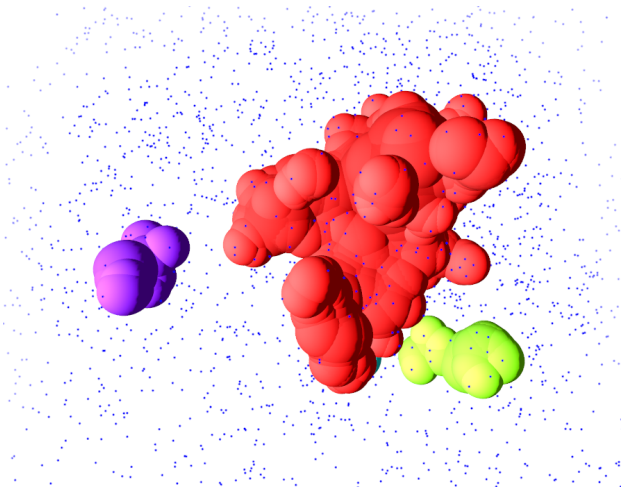
We have evaluated the run time of our algorithm with regard to a variable probe size and a variable molecule size.

Table 1 and Figure 9 shows the measured algorithm run time for the variable size of the probe. We used the Thermus Thermophilus 70S ribosome complexed with chloramphenicol (PDB ID 3OH5, approximately 87 000 atoms) in this experiment.

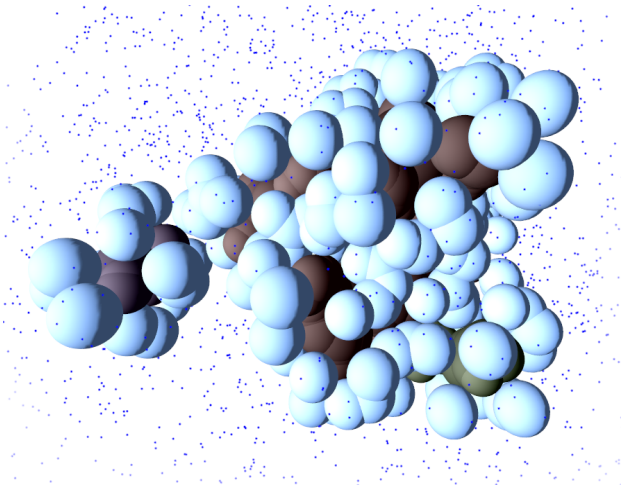
Probe Size [Å]	Time [ms]
0.2	174
0.4	146
0.6	114
0.8	91
1.0	75
1.2	64
1.4	55
1.6	47
1.8	41
2.0	35

Table 1: Dependency of the algorithm run time on the probe size.

Next, we have evaluated the run time of our algorithm on several molecules of different sizes. The measured



(a)



(b)

Figure 8: (a) Some of the cavities in the molecule 1AKD for the probe with the radius 1.4Å. (b) The cavities (dark) with the atoms forming them (light). Small dots represent centers of the molecule atoms.

times without preprocessing are presented in Table 2. In the first column, the PDB IDs of the tested molecules are presented. In the second column, the number of atoms is given. In the third column, the measured time is given. Figure 10 shows that the algorithm scales linearly for a variable molecule size.

Molecule	No. of atoms	Time [ms]
1CQW	2 754	3
3VMN	5 621	2
3AOB	23 385	14
3UXS	49 743	33
3OH5	87 539	55

Table 2: Algorithm run time for molecules of various sizes.

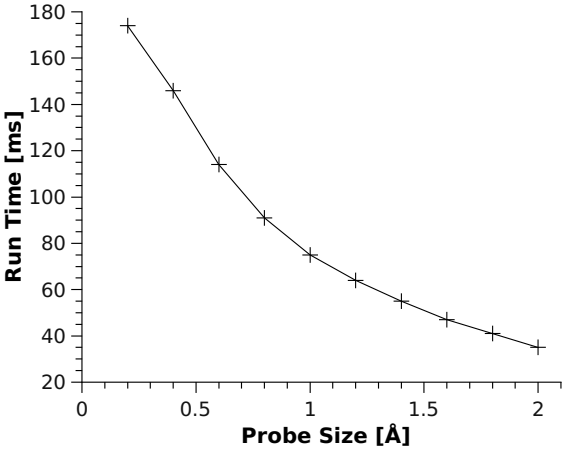


Figure 9: Dependency of the algorithm run time on the probe size.

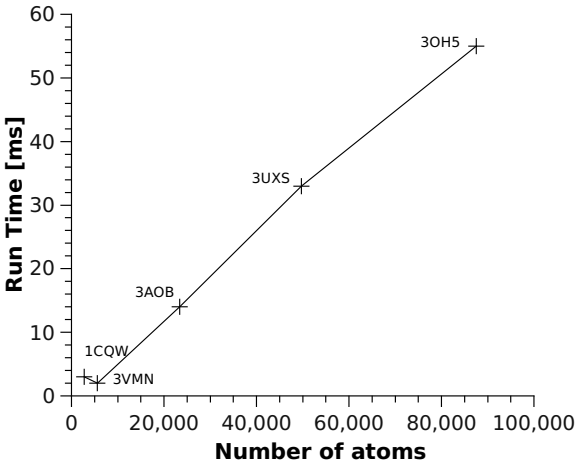


Figure 10: Dependency between the number of atoms and the run time for the probe with the radius of 1.4Å.

5.2 Comparison With Other Software

We have compared run time of our algorithm with Voroprot 0.7.6.4 [20], which implements similar algorithm using the additively weighted Voronoi diagram for finding cavities. We chose Voroprot also because we had problems running other implementations, such as CASTp [4] which is offered only as a web service.

Transaldolase from Corynebacterium glutamicum (PDB ID 3R5E, 2 957 atoms) was used for the comparison. The reason of the choice of such a small molecule was that we encountered the problems with processing larger molecules in Voroprot. The times for Voroprot are approximate, as the application has a graphical interface only. The measured times are in Table 3. It can be seen that our algorithm is several orders of magnitude faster than Voroprot. The reason is that Voroprot always searches all Voronoi vertices while our algorithm uses sorting to reduce the size of searched set.

Probe Size [Å]	Run Time [s]		Algorithm Speedup
	Our Algorithm	Voroprot	
0.2	7×10^{-3}	3.5	500
0.4	4×10^{-3}	2.5	625
0.6	3×10^{-3}	1.9	633
0.8	2×10^{-3}	1.4	700
1.0	2×10^{-3}	1.0	500

Table 3: Comparison with Voroprot.

5.3 Results, Summary and Future Work

We used the visualization tool to check the results visually. As far as we know, exact evaluation of results has not been developped yet. We have also compared the largest cavities found using our implementation with the cavities found using Voroprot.

Since Voroprot provides only a graphical interface, it was not possible to do exact run time measurements. Therefore, more exact time comparison should be carried out in the future.

Our algorithm currently does not handle elliptic edges. These edges may or may not be incident to any Voronoi vertex. It is also possible that only a part of an elliptic edge forms a cavity. Fortunately, elliptic edges are rare in proteins, because the difference among atom radii is small. This is left for future work.

6 Conclusion

We have presented the algorithm for finding inner cavities in a molecule. The algorithm computes the additively weighted Voronoi diagram of molecule atoms and sorts the Voronoi vertices using the distance to their generators in the preprocessing. The cavities are then found using a graph traversal, starting from the vertex with largest distance and ending when all cavities are found.

Our experiments shows that our algorithm is excellent for a variable probe size thanks to the preprocessing. The algorithm is able to process even large protein structures very quickly.

Acknowledgment

This work has been supported by the Grant Agency of the Czech Republic, project No. P202/10/1435 and project SGS-2010-028. The implementation uses the awVoronoi library [1], developed by Martin Maňák.

References

[1] http://awvoronoi.sourceforge.net/.

[2] P. Beneš, E. Chovancová, B. Kozlíková, A. Pavelka, O. Strnad, J. Brezovský, V. Šustr, M. Klvaňa, T. Szabó, A. Gora, M. Zamborský, L. Biedermannová, P. Medek, J. Damborský, and J. Sochor. Caver 2.1. http://www.caver.cz, 2009-2011.

[3] Y. Cho, D. Kim, H.-C. Lee, J. Y. Park, and D.-S. Kim. Reduction of the search space in the edge-tracing algorithm for the Voronoi diagram of 3D balls. In *ICCSA (1)*, pages 111–120, 2006.

[4] J. Dundas, Z. Ouyang, J. Tseng, A. Binkowski, Y. Turpaz, and J. Liang. CASTp: computed atlas of surface topography of proteins with structural and topographical mapping of functionally annotated residues. *Nucleic Acids Research*, 34(suppl 2):W116–W118.

[5] H. Edelsbrunner. The union of balls and its dual shape. *Discrete & Computational Geometry*, 13:415–440, Dec. 1995.

[6] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13:43–72, Jan. 1994.

[7] T. Exner, M. Keil, G. Moeckel, and J. Brickmann. Identifification of substrate channels and protein cavities. *Journal of Molecular Modeling*, 4(10):340–343, Oct. 1998.

[8] M. Hendlich, F. Rippmann, and G. Barnickel. Ligsite: automatic and efficient detection of potential small molecule-binding sites in proteins. *Journal of Molecular Graphics and Modelling*, 15(6):359–363, Dec. 1997.

[9] D. Kim, C.-H. Cho, Y. Cho, J. Ryu, J. Bhak, and D.-S. Kim. Pocket extraction on proteins via the voronoi diagram of spheres. *Journal of Molecular Graphics and Modelling*, 26(7):1104–1112, 2008.

[10] D.-S. Kim, Y. Cho, K. Sugihara, J. Ryu, and D. Kim. Three-dimensional beta-shapes and beta-complexes via quasi-triangulation. *Computer-Aided Design*, 42(10):911–929, 2010.

[11] D.-S. Kim, Y. Chob, and D. Kim. Euclidean voronoi diagram of 3d balls and its computation via tracing edges. *Computer-Aided Design*, 37:1412–1424, 2005.

[12] D.-S. Kim, D. Kim, Y. Cho, and K. Sugihara. Quasi-triangulation and interworld data structure in three dimensions. *Computer-Aided Design*, 38(7):808–819, 2006.

[13] D.-S. Kim, J.-K. Kim, C.-I. Won, C.-M. Kim, J. Y. Park, and J. Bhak. Sphericity of a protein via the β-complex. *Journal of Molecular Graphics and Modeling*, 28(7):636–649, 2010.

[14] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape. *PROTEINS: Structure, Function, and Genetics*, 33(1):1–17, Oct. 1998.

[15] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: II. inaccessible cavities in proteins. *PROTEINS: Structure, Function, and Genetics*, 33(1):18–29, Oct. 1998.

[16] J. Liang, C. Woodward, and H. Edelsbrunner. Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design. *The Protein Society*, 7(9):1884–1897, Sept. 1998.

[17] M. Maňák and I. Kolingerová. Fast discovery of Voronoi vertices in the construction of Voronoi diagram of 3D balls. *International Symposium on Voronoi Diagrams in Science and Engineering*, pages 95–104, 2010.

[18] N. N. Medvedev, V. P. Voloshin, V. A. Luchnikov, and M. L. Gavrilova. An algorithm for three-dimensional Voronoi S-network. *Journal of Computational Chemistry*, 27(14):1676–1692, 2006.

[19] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. John Wiley & Sons, Inc., 2nd edition, 2000.

[20] K. Olechnovič, M. Margelevičius, and Č. Venclovas. Voroprot: an interactive tool for the analysis and visualization of complex geometric features of protein structure. *Bioinformatics*, 27(5):723–724, 2010.

[21] S. Sonavane and P. Chakrabart. Cavities and atomic packing in protein structures and interfaces. *PLoS Computational Biology*, 4(9):e1000188, Sept. 2008.

[22] A. Tripathi and G. E. Kellogg. A novel and efficient tool for locating and characterizing protein cavities and binding sites. *Proteins: Structure, Function, and Bioinformatics*, 78(4):825–842, Mar. 2010.

[23] M. A. Williams, J. M. Goodfellow, and J. M. Thornton. Buried waters and internal cavities in monomeric proteins. *Protein Science*, 3(8):1224–1235, Aug. 1994.

[24] M. Zemek, M. Maňák, and I. Kolingerová. Advanced space filtering for the construction of 3D additively weighted Voronoi diagram. *ADVCOMP*, pages 37–43, 2011.