Computation of tunnels in protein molecules using Delaunay triangulation

Petr Medek
Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno,
Czech Republic
medek@fi.muni.cz

Petr Beneš
Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno,
Czech Republic
xbenes2@fi.muni.cz

Jiří Sochor
Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno,
Czech Republic
sochor@fi.muni.cz

ABSTRACT

This paper presents a new method of specific cavity analysis in protein molecules. Long-term biochemical research has the discovery that protein molecule behaviour depends on the existence of cavities (tunnels) leading from the inside of the molecule to its surface. Previous methods of tunnel computation were based on space rasterization. Our approach is based on computational geometry and uses Voronoi diagram and Delaunay triangulation. Our method computes tunnels with better quality in reasonable computational time. The proposed algorithm was implemented and tested on several real protein molecules and is expected to be used in various applications in protein modelling and analysis. This is an interesting example of applying computational geometry principles to practical problems.

Keywords
protein, tunnel, Voronoi diagram, Delaunay triangulation

1. INTRODUCTION

Long-term research into the biochemical characteristics of protein molecules has the discovery that protein reactivity is closely related to the presence of routes leading from the protein surface to a biochemically relevant cavity inside the protein, an active site. In the active site chemical reactions between the protein and some substrate molecule take place. One of the conditions the substrate molecule requires to get to the active site is the presence of an empty space connecting the surface of the protein molecule with the active site. This empty space is used by the substrate molecule to reach the active site without crossing any atom of the protein and is referred to as a tunnel. In Figure 1, a substrate molecule can use two different tunnels to get to the active site.

We emphasise that the geometrical existence of the tunnel alone is not sufficient to guarantee that a substrate molecule can access the active site. The ability of the protein to react with the substrate is based on many different physical and chemical factors. Still, a tunnel computed concerning just the geometrical point of view could provide information that will help chemists to focus on specific parts of the protein.

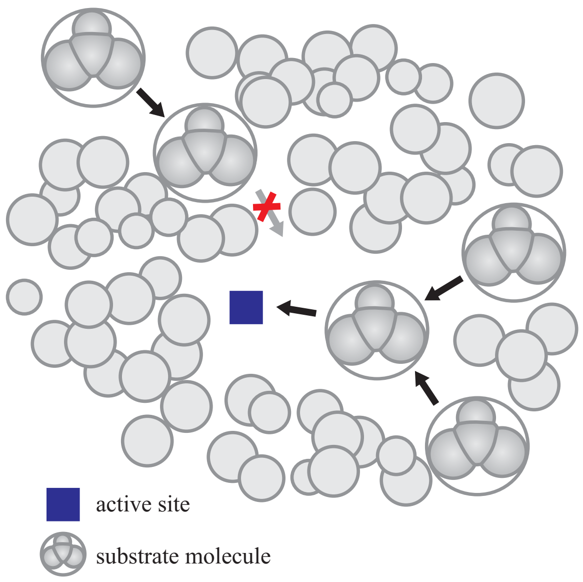

Figure 1. Tunnels in protein, the active site is accessible by two tunnels.

Considering only the geometry of the protein, the molecule will be simplified to a set of spheres where each sphere represents one atom. Each of these spheres is situated in a certain position in 3D space and has an appropriate Van der Waals radius. At this stage we will not consider any other chemical properties of the atoms or the molecule. The substrate molecule is simplified to one bounding sphere enclosing all substrate atoms. Due to these simplifications tunnels always have circular cross-sections and can be compared with one another with respect to the diameter of the minimal cross-section. This value limits the size of substrate molecules that are able to access the active site.

The computation time is crucial especially when analysing large sequences consisting of thousands of 'molecule snapshots', i.e. changes in positions of atoms over a time period. In such cases the tunnel computation has to be performed separately for each snapshot.

2. RELATED WORK

The existing method of tunnel computation is based on space discretization and is implemented in the program CAVER [Pet06]. The space containing the molecule is regularly sampled and the samples in the 3D raster (cubes) are evaluated by distances from the nearest atom. The raster is interpreted as a graph with weights at graph vertices. The search for a tunnel with the greatest minimal distance is based on the Dijkstra algorithm.

Another method of protein cavity analysis is based on α-shapes [Ede94] and is implemented in the program CASTp [Lia98]. CASTp deals with overall cavity analysis, including determination of atoms forming rims of cavities inside the molecule and other analyses such as volume measurements of cavities. Cavity analysis is done for all cavities present (whether accessible from the outside or not) and does not require any user input. As this algorithm was designed as a general solution to cavity analysis it does not deal with tunnels as export routes and cannot be used for tunnel computation. A fast and specialised algorithm is required.

A lot of research has been conducted in the area of Voronoi diagram (VD) and Delaunay triangulation (DT) modifications. In [Kim04] an algorithm for a construction of a VD of a set of spheres, referred as an additively weighted VD, is presented (sometimes also referred as Euclidean VD of spheres).

In [Gho03] the additively weighted DT is used as a solution to the problem of network routing. The algorithm is based on 2D additively weighted DT. Since the algorithm maintains the additively weighted DT of sites that dynamically change its position when users in mobile networks move, it is more general.

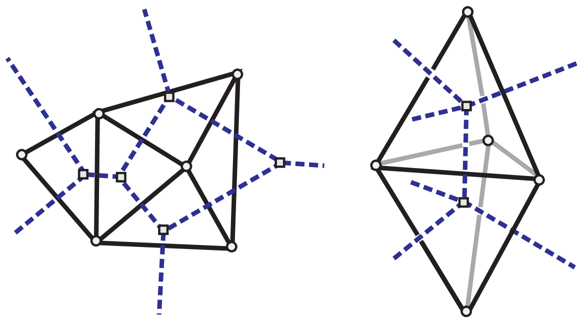
Figure 2. Duality between Voronoi diagram (dashed) and Delaunay Triangulation in 2D (left) and 3D space (right).

3. PROPOSED SOLUTION

As mentioned above, the protein molecule is simplified to a set of spheres *S*. Each sphere $s \in S$ is defined as $s\,(c_s, r_s)$, where c_s denotes its center point and r_s its radius. When we mention an atom below, we are referring to the sphere representing this atom.

The function $D\,(x,s)$ computes the Euclidean distance of a point *x* from the surface of a sphere *s*. If *x* is situated inside the sphere *s*, the result of the function is negative:

$$D\,(x,s) = \| x - c_s \| - r_s$$

For each atom with respective sphere $a \in S$, a *Voronoi cell* $V(a)$ is defined as a set of points satisfying the following condition:

$$\forall x \in R^d\!: \; x \in V(a) \Leftrightarrow D\,(x,a) \le D\,(x,b),\; \forall a,b \in S,\; a \neq b,\; d \in \{2,3\}$$

A *Voronoi diagram* $V(S)$ is a union of all Voronoi cells $V(u)$, $\forall u \in S$. Hereafter the border of a Voronoi cell will be referred to as a *Voronoi edge*. This definition is equal to the definition of an additively weighted Voronoi diagram [Kim04].

In our algorithm we employ duality between a VD and a DT, one of the most important features of these structures. For each VD a corresponding DT exists and vice versa (see Fig. 2).

The minimal distance from the point *x* to the nearest sphere (atom) is given by the function $r\,(x)$:

$$r\,(x) = \min \{D\,(x,s) \mid s \in S\}$$

Using these functions we can formalise intuitive notion of a *tunnel*. A tunnel *T* leading from a point *x* to a point *y* is defined by a centerline and a tunnel volume. The centerline is a continuous curve a_T starting in *x* and ending in *y*. The tunnel volume is formed by the union of spheres inserted at each point

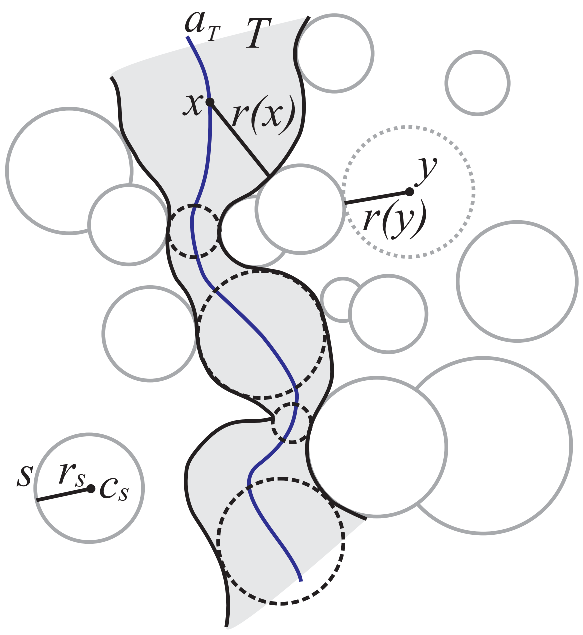

Figure 3. Tunnel T with the central line a_T

$x \in a_T$ with appropriate radius $r(x)$. Formally, T is defined as

$$T = \bigcup_{x \in a_T} s(x, r(x))$$

The function $n(T)$ returns the radius of the minimal sphere of a tunnel T:

$$n(T) = \min \{ r(x) \mid x \in a_T \}$$

The function $p(T)$ returns a set of points on the tunnel T centerline a_T satisfying the condition that their distance to the nearest atom is $n(T)$:

$$p(T) = \{ x \mid x \in a_T \wedge r(x) = n(T) \}$$

In most cases $p(T)$ returns just a single point, the centre of the „narrowest“ passage.

The order relation on the set S_T of tunnels with the centerline leading from x to y is defined by $n(T)$:

$$T \leq T' \Leftrightarrow n(T) \leq n(T'), \forall\, T, T' \in S_T$$

The tunnel $T \in S_T$ is *ideal* if for every other tunnel $T' \in S_T$ the condition $T \leq IT$ holds.

Now, we state an important condition that is essential for our method. This statement is valid for any additively weighted Voronoi diagram and obviously, when setting all atom radii to zero, it is also valid for a Voronoi diagram of a set of points.

Lemma 3.1: *Let S be a set of spheres in 3D, $|S| > 1$. Consider an ideal tunnel T with the center-line leading from a point x to a point y. If $x \notin p(T) \wedge y \notin p(T)$ then at least one point from $p(T)$ (the narrowest passage point) is situated on some Voronoi edge of Voronoi diagram $V(S)$.*

Proof: See Appendix.

Tunnel computation

For better understanding the following concept is explained in the plane. An extension to three dimensions is straightforward.

Given $V(S)$ for an input set of points S, the ideal tunnel can be computed according to the Lemma 3.1.

The “narrowest” point of a Voronoi edge shared by two atoms u and v is located at the intersection of the Voronoi edge with the edge connecting u and v. If such an intersection does not exist, the narrowest point is located in the Voronoi edge endpoint x with the smallest $r(x)$. Thus, the narrowest place of every tunnel must be located either in the Voronoi edge intersection or in the Voronoi edge endpoint. Note that knowledge of the shape of all Voronoi edges is not necessary for the computation of the ideal tunnel Voronoi edge intersections and endpoints mentioned above are sufficient.

For computation, it is more convenient to represent space partition with dual structure of VD, Delaunay triangulation. DT can be interpreted as a weighted graph G. Nodes of the graph are formed by triangles of DT and graph edges are formed by edges shared by the two neighbouring triangles. The edge weight is defined as the narrowest point of the corresponding Voronoi edge (see Fig. 4).

The graph G is used for computation of the ideal tunnel T leading from the active site A. A possible utilisation of this graph is described in Section 3.1.2. The algorithm extension for computation of more than one tunnel is proposed in Section 3.1.3. The algorithm described is summarised by the following pseudocode:

```
Input: set of atoms M
       active site A

Output: ideal tunnel

DT = delaunayTriangulation(M);
G = convertToGraph(DT);
T = computeTunnel(G,A);
output(T);
```

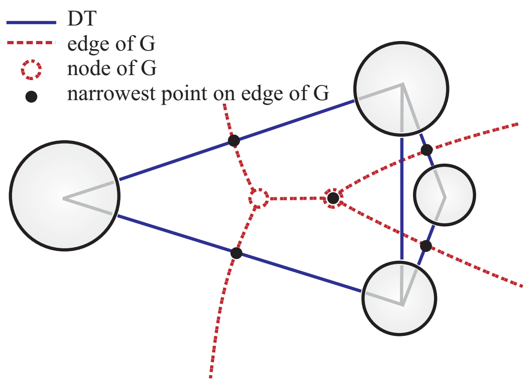

Figure 4. Example of graph G

3.1.1 *Delaunay triangulation computation*

The exact computation of an additively weighted VD and an corresponding DT would be very expensive. Since the algorithm for tunnels needs to be fast, we propose several simplifications. We proceed from the standard DT for a set of points instead and modify it for a set of spheres. The following simplifications of the exact solution are possible:

- **Conservative simplification** – By the definition, the VD of a set of atoms that have equal radius is the same as the VD of a set of points. The simplification could be performed by setting the radius of all atoms to the radius of the biggest atom in the molecule. Then the DT of a set of atom centers could be considered as the valid DT of the set of atoms. The process of the edge weight evaluation in the graph *G* is modified. The weight of every edge is reduced by an amount corresponding to the radius of the biggest atom in the molecule.
- **Approximate simplification** – Typical protein molecules usually consist of only a few types of atoms and their radii do not vary significantly (from 1.2 to 1.85 Ångström[1]). If we have the DT of a set of atom centers and presume that it is a valid DT for a set of atoms, possible error caused by this approximation is minimal. The weight of the edge in *G* is set to half the minimal distance of the two surfaces of the atoms forming the edge.

DT could be obtained by several algorithms, e.g. the lifting algorithm [Bar95] which uses the relation between the convex hull and the DT. The convex hull of a set of points in R^{d+1} corresponds to the DT in R^d. The time complexity of the convex hull computation in R^4 is $O(n^2)$, where *n* is the number of points in the input set. The evaluation of edge weights in the graph G is linear with respect to the number of nodes in *G*.

[1] 1 Å = 10^{-10} m

3.1.2 *Tunnel computation*

In order to compute the ideal tunnel, we process the graph *G* using a modified Dijkstra algorithm. The function *f*(*x*) evaluating each node in *G* provides maximization of the minimal weight on the way from the starting node to processed node. The Dijsktra algorithm guarantees that the tunnel found for the given starting point and the graph *G* is ideal. Time complexity of Dijkstra algorithm is $O(n^2)$, where *n* is the number of nodes in *G*, i.e. number of triangles in DT.

The output of our algorithm is one ideal tunnel for a given active site and therefore it is not necessary to evaluate every node in *G*. It is sufficient to perform the evaluation of nodes until we reach the exterior of the molecule. In our case, the exterior is determined by the convex hull of the molecule. In the graph G, a convex hull can be simply found as a set of nodes having at least one of its neighbours missing.

Although this simplification does not improve computational complexity in general, for real molecules the time of computation is much faster, as demonstrated in Section 4.

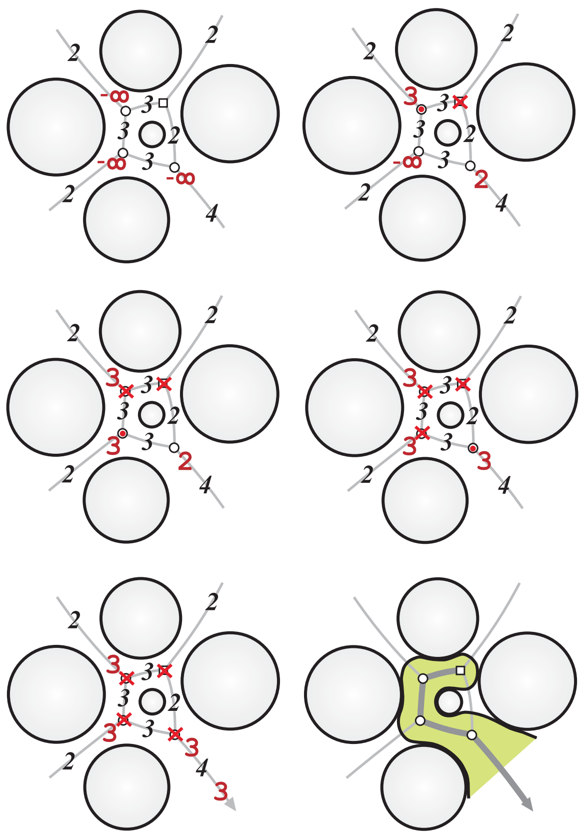

Figure 5. Example of Dijkstra algorithm progress.

The modified Dijkstra algorithm is described by the following pseudocode; $d[v]$ denotes the actual $n(T)$ on the centerline of the tunnel T leading from the starting node s to the node v, $previous[v]$ denotes the predecessor of v on the centerline of T and $w(u,v)$ denotes the weight of edge (u,v) in G.

```
computeTunnel

Input: undirected weighted graph G
       starting node s

Output: sequence of tunnel nodes

for each node n in G
   d[n] = -∞;
   previous[n] = null;

d[s] = ∞;
u = s;
while (!u.onBorder())
   u = getUnprocessedMaximum(G);
   for each edge (u,v) outgoing
             from u
      if (d[v] < max(d[v],w(u,v))
         d[v] =  max(d[v],w(u,v);
         previous[v] = u;

while (u != s)
   output(u);
   u = previous[u];
```

3.1.3 Modification for more tunnels

If two tunnels T_1, T_2 with the same $n(T_1) = n(T_2)$ satisfy the condition for the ideal tunnel, one of them is selected randomly by the Dijkstra algorithm. However, if $p(T_1) \neq p(T_2)$, it could be useful to compute both these tunnels. Also the computation of the next-best tunnels is required sometimes. Therefore if more than one tunnel is to be found, we propose the following solution. To exlude already found tunnels, the graph G is modified after each particular tunnel computation and the whole process is repeated. To find new tunnels we may use various graph modifications which are obvious from geometrical point of view. However, the chemical relevance of these modifications is not known yet. We propose the following modifications:

- Set to zero weight of all edges of G with the minimal weight along the computed tunnel.
- Set to zero weight of all edges of G along the computed tunnel from the surface to the edge with the minimal weight furthest from the surface.
- Set to zero weight of edges in the close neighbourhood of edges with the minimal weight.

We use constraints C to determine the number of computed tunnels. The process of tunnel computation is repeated until C is not satisfied. As an example, for compution of all tunnels with minimal width higher than 1.2 Å, C would be a condition "*n(T) > 1.2*".

```
Extension for more tunnels

Input: set of atoms M
       active site A
       constraints C

Output: computed tunnels

DT = delaunayTriangulation(M);
G = convertToGraph(DT);
do
{
   T = computeTunnel(G,A);
   G = modifyGraph(T,G);
   output(T);
} while (C)
```

Complexity

The time complexity of DT computation is quadratic with respect to the number of atoms. The number of nodes in G is linear to the number of atoms. The time complexity of the Dijkstra algorithm is also quadratic with respect to the number of nodes in G. Therefore the overall time complexity is $O(n^2)$, where n is the number of atoms

4. RESULTS

Implementation

The algorithm was implemented in Java. The implementation uses the standard DT of a set of points in 3D and extends it for a +set of spheres. Both conservative and approximate simplification were implemented. The output of our program is a set of spheres approximating the computed tunnel. A sphere is inserted into the center of each node and to the narrowest point on each edge through which the tunnel leads. Radii of these spheres are computed during the transformation phase of the DT to the graph G. For better accuracy of the approximate method we check possible collisions of these spheres with atoms in their close neighbourhood. If the collision test is positive we decrease the radius of the appropriate tunnel sphere so that the collision does not occur.

The output set of spheres is used for a simple tunnel visualization (see Fig. 6 and 7).

Minimal radius (Ångström)	CAVER using grid size			DT method	
	0.8Å	0.4Å	0.15Å	Conservative	Approximate
DhaA	1.05571	1.29359	1.4153	1.4207468	1.4772751
LinB	0.647456	0.674731	0.785306	0.7510569	0.8693304

Table 1. Accuracy comparison.

Time of computation (seconds)	CAVER using grid size			DT method	
	0.8Å	0.4Å	0.15Å	Conservative	Approximate
DhaA	3.782	108.64	37152.91	0.984	1.312
LinB	3.391	100.89	34913.98	1.031	1.406

Table 2. Time of computation.

Practical results

The output of both compared methods, CAVER and our method, is a set of spheres approximating the computed tunnel. The CAVER program was tested with sampling densities 0.8, 0.4 and 0.15 Ångström. The maximal error of the sampling method is dependent on sampling density.

We tested both algorithms on real protein molecules DhaA (consisted of 2358 atoms) and LinB (2479 atoms). The tests were performed on a computer with P4 3.0GHz CPU and 1GB RAM. Active sites were determined inside chemically significant cavities inside the protein molecule.

In Table 1, comparison between the two algorithms is done considering a real width of the narrowest tunnel radius. This value can be achieved in the following way. For each center c_s of sphere s in the sampled tunnel, the value $r(s)$ is determined. If $r_s > r(s)$ then r_s is changed to $r(s)$. Therefore the tunnel found does not intersect any other atom and the value $n(T)$ of the computed tunnel T could never be higher than $n(IT)$ of the actual ideal tunnel IT.

A solution obtained by using an approximate simplification cannot guarantee that the narrowest place is determined by one of the sampled tunnel spheres. It is possible that error can arise on the tunnel centerline between two neighbouring spheres in the sampled tunnel. Therefore, the tunnel centerline is sampled densely to minimise the probability of such errors arising.

It is not possible to set the sampling density in the program CAVER densely enough due to the high system requirements. Despite that we consider CAVER results to be accurate enough.

As shown in Table 1, when testing on two real molecules, the solution obtained by the approximate simplification is more accurate than the conservative simplification. In comparison with the program CAVER, the approximate simplification is significantly more accurate. On the DhaA molecule even the conservative simplification provides better results than CAVER.

Time (seconds)	Full Dijkstra	Reduced Dijkstra
DhaA	2.407	0.031
LinB	2.891	0.047

Table 3. Comparison of Dijkstra algorithm computation.

Furthermore, if we compare the computation time shown in Table 2, the ratio of accuracy to computation time obtained by both our methods is much better. The computation of tunnels on the LinB molecule took CAVER 34914 seconds to be more accurate than the conservative simplification (computed in 1.031 seconds) of our algorithm.

The comparison of computation time of the whole Dijkstra algorithm and the reduced Dijkstra algorithm is shown in Table 3. These results imply that stopping Dijkstra algorithm on the Convex hull brings a significant decrease in computation time of the Dijkstra algorithm.

5. CONCLUSION

In this paper we have described a novel method of tunnel computation in protein molecules. We demonstrated two possible simplifications of the proposed algorithm, which speed up the computation process without notable loss of accuracy. The conservative simplification gives worse but still reasonably precise results and is faster due to its simplicity. The approximate simplification computes wider tunnels at the cost of possible presence of an error in the result. In comparison with previous solution, both our methods have much better ratio of speed to accuracy.

There are several avenues possible for the further research in the future. We want to implement the

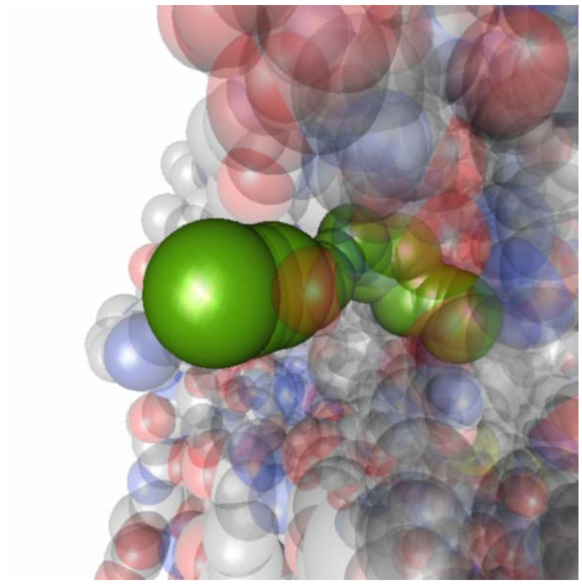
Figure 6. Ideal tunnel in DhaA molecule

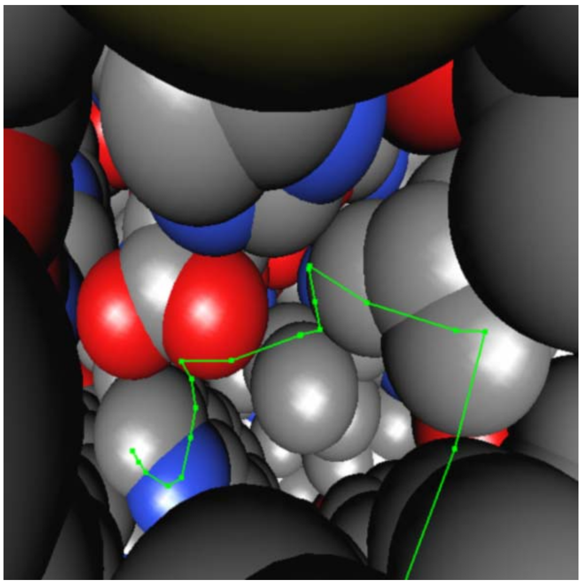
Figure 7. Tunnel centerline (denoted by the line)

exact solution using additively weighted DT in 3D to confirm our assumption that the accuracy improvement is not worth of the speed degradation. Utilization of more sophisticated methods of DT computation could significantly improve the speed of the algorithm, e.g. we could perform some space partitioning and compute only a part of the graph G on demand of Dijkstra algorithm.

Volume maximization of the tunnel instead of maximization of the narrowest cross-section could be biochemically significant. Fast analysis of large snapshot sequences sampling molecule in time is also demanded. Algorithm output could be also processed with other techniques, e.g. haptical devices could explore the tunnel surface for other biochemical properties.

6. ACKNOWLEDGEMENTS

This work was supported by The Ministry of Education of The Czech Republic, Contract No. LC06008 and by The Grant Agency of The Czech Republic, Contract No. 201/07/0927.

7. REFERENCES

[Bar95] Barber, C.B., Dobkin, D.P., and Huhdanpaa, H. The Quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software, 22(4), pp.469-483, 1996.

[Ede94] Edelsbrunner, H., and Mucke, E. P. Three-Dimensional Alpha Shapes. ACM Transactions on Graphics, 13(1), pp43-72, 1994.

[Gho03] Ghosh, R.K., Gupta, G., and Rao, S.V. A routing Algorithm for Multi-Hop Mobile Ad Hoc Network Using Additively Weighted Delaunay Triangulation. CIT, 2003.

[Kim04] Kim, D.S., Youngsong, C., Kim, D. Edge-tracing algorithm for Euclidean Voronoi diagram of 3D Spheres, Proc. of the 16th Canadian Conference on Computational Geometry, pp. 176-179, 2004.

[Lia98] Liang, J., Edelsbrunner, H., and Woodward, C. Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design. Protein Science, 7, pp.1884-1897, 1998.

[Pet06] Petřek, M., Otyepka, M., Banáš, P., Košinová P., Koča, J., and Damborský J. CAVER: a new tool to explore routes from protein clefts, pockets and cavities. BMC Bioinformatics, 2006.

[Pre85] Preparata, F.P., and Shamos, M.I. Computational Geometry: An introduction. Springer-Verlag, 1985.

APPENDIX

Lemma 3.1: *Let S be a set of spheres in 3D,* $|S| > 1$. *Consider an ideal tunnel T with the centerline leading from a point x to a point y. If* $x \notin p(T) \wedge y \notin p(T)$*, then at least one point from* $p(T)$ *(the narrowest passage point) is situated on some Voronoi edge of Voronoi diagram* $V(S)$.

Proof: Consider a tunel T with the centerline leading from the point x to the point y satisfying $x \notin p(T) \wedge y \notin p(T)$. Let the set $p(T)$ contain only one point p_T. If $p(T)$ contains more than one point, we perform the construction demonstrated in this proof for each element of $p(T)$.

Suppose that p_T is not situated on any Voronoi edge of $V(S)$ and T is ideal (for each tunnels T' leading from x to y, the condition $T \leq T'$ holds). If p_T is not situated on any Voronoi edge of $V(S)$, then p_T is situated inside one of the Voronoi cells $V(u)$. This implies $D(p_T,u) < D(p_T,v)$ for each sphere $v \in S$, $v \neq u$.

The point p_x denotes the intersection of the border of $V(u)$ with the part of a_T, that lies between x and p_T. If more than one intersection exists, we select the intersection closest to p_T. Similarly the point p_y denotes the closest intersection of the border of $V(u)$ with the part of a_T lying between p_T and y. If x lies inside $V(u)$, let p_x be the point x. If $y \in V(u)$, then $p_y = y$. The part of a_T between points p_x and p_y is denoted a'_T. The curve a'_T is continuous and all points of a'_T lie inside $V(u)$, $\forall v \in a'_T \Rightarrow v \in V(u)$.

For each point $w \in a'_T$ we create a half-line with the starting point in c_u passing through w. w' denote the intersection of this half-line with the border of $V(u)$. Note that for each w, $w \neq p_x \wedge w \neq p_y$, the condition $r(w) < r(w')$ holds. The set of all points w' forms a continuous curve leading from p_x to p_y. We denote this curve a''_T. Since we assume that T is ideal, for each point $w \in a'_T$, $w \neq p_T$ the condition $r(w) > r(p_T)$ holds, therefore for each point $w' \in a''_T$ the condition $r(w') > r(p_T)$ is satisfied as well.

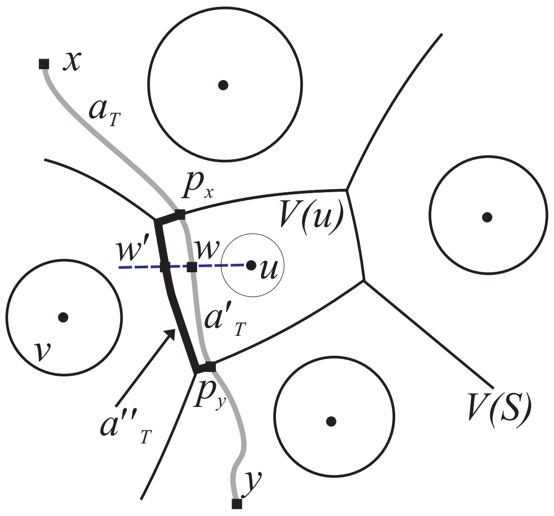

Figure 8. Proof illustration.

We modify the centerline of T by replacing a'_T with a''_T. The modified tunnel is denoted by T'. For each point $w \in a_{T'}$ the condition $r(w) > r(p_T)$ holds. However, this implies $n(T') > n(T)$, contradicting our assumption that T is ideal.