Tunnel Computation Methods Section

July 5, 2012

1 Methods

The tunnel computation is performed in several steps. First, the Voronoi diagram representation of the protein is computed. Next, the diagram is split into several smaller parts - the so called cavity diagrams - that can contain the tunnels. Afterwards, suitable starting and ending points are identified in each of the cavity diagrams (these points can also be user defined). Finally, the Dijkstra's Shortest Path algorithm is used to find the tunnels between the pairs of starting and ending points.

1.1 Voronoi Diagram Representation

The protein is represented by a Voronoi diagram. A Voronoi diagram divides a metric space according to the distances between discrete sets of specified objects. In the case considered here, the objects are the centers of protein atoms represented by van der Waals VDW) spheres with radii predefined by AMBER force field [**?**], and the Voronoi diagram consists of cells representing the set of points closest to the atom in the center of each cell. This simple approach partitions the solely according to atomic centers and does not take differing atomic radii into account. Additionally, the user is allowed to select a set of active residues that form the diagram.

The Voronoi diagram is a graph-theoretical dual of the Delaunay triangulation of the VDW sphere centers [**?**]. The vertices of the diagram correspond to the tetrahedrons of the triangulation and are located at the tetrahedron's circumcenter (of cource, strictly speaking there is no concept of location in graph theory, but we will allow ourselves to use it anyway). The edges of the diagram represent the adjacency of the tetrahedrons. Consequently, each vertex of the diagram has degree 4 (if we assume one "infinite vertex" v_∞ that represent the outer space of the protein). A vertex v is called boundary if one of it's neighbors is v_∞.

1.2 Tunnel Computation

Informally, a protein tunnel is a path between a point inside the protein to its boundary. To each point of the tunnel we assign a radius that indicates how wide the tunnel is. Formally, the tunnel is defined as a path $c = c_0 \dots c_n$ (with c_n boundary) in the Voronoi diagram (in its graph sense). The concept of the tunnel radius is captured by the function

d that assigns to each point (or line/edge) the distance to the surface of the closest VDW sphere. The cost w of a tunnel c is defined as $w(c) = \sum_{i=0}^{n-1} w^*(c_i, c_{i+1})$ where the cost of the edge $e = (c_i, c_{i+1})$ is defined as

$$w^*(e) = \frac{l(e)}{d(e)^2 + \epsilon}$$

with $l(e)$ the length of the edge and ϵ some small number to avoid division by zero. Additionally, the tunnel profile is defined as a function p that assigns to each point of the tunnel the set of residues that surround it. The tunnel length l is defined as $l(c) = \sum_{i=0}^{n-1} \|c_i - c_{i+1}\|$. It is also useful to consider the 3D natural spline [**?**] representation $C : [0, 1] \to \mathbb{R}^3$ of the tunnel that maps the normalized tunnel length to a set of 3D points. We refer to this representation as the centerline of the tunnel. The spline representation allows us sample the tunnel with varying precision depending on the context (for example 1.5 points per angstrom to display the 3D model of the tunnel consisting of spheres).

In essence, to find a tunnel the previous version of MOLE [**?**] required the user to specify a starting point and from this point, the Voronoi diagram got traversed in a depth first manner. Where paths with the lower cost were visited first. Each time a boundary vertex was reached, a tunnel got reported. There are numerous problems with this approach. For example, the the first tunnel got reported, the next one was very similar to the first one (and therefore provided no useful additional information) because the path "branched" near the end point of the first tunnel. Another issue (somewhat related to the previous one) was caused by small "ridges" that were formed by vertices near the boundary of the diagram. As a result, a large part of the tunnel was "going along" the surface of the protein and therefore did not provide a very relevant information about the structure of the tunnel. It was also very difficult to identify interesting starting points of the tunnels. The current version of the algorithm addresses these problems issues by first preprocessing the Voronoi diagram and splitting it into several smaller parts (called cavity diagrams). In these cavities, suitable start and end points of the tunnels are identified and then Dijkstra's algorithm is used to find the tunnels.

1.3 Preprocessing the Diagram

The preprocessing works in several steps:

- If the Voronoi diagram is simply computed from the VDW sphere centers, the boundary of the diagram corresponds to the convex hull of the protein. This is not desirable because then the tunnel exits might end up being too far from the actual protein surface (in a way this is similar to the shallow ridge issue from the previous section). To remedy this, the user is required to specify the probe radius parameter which is used to approximate the molecular surface. To achieve this, layers of vertices are removed repeatedly starting from the boundary layer. For each vertex, the corresponding tetrahedron is checked if the probe would pass

through it and if so, it is removed. This process is repeated as long as there is no longer any vertex to remove.

- The second step is to remove vertices that cannot be part of any tunnel. This is governed by a parameter called the interior threshold. A vertex v is removed if for all edges (v, u) it holds that $d(v, u) <$ int. thereshold.
- After the boundary and interior vertices are removed, connected components of the Voronoi diagram are computed. We call such a component the *cavity diagram*.
- The last preprocessing step is to remove the *shallow vertices*. These are the vertices that form the above mentioned ridges along the surface of the molecule. All vertices with depth (the distance from the surface) less than the given threshold (say 5) are removed if all their neighbors have the same or lower depth.

1.4 Detection of Start- and Endpoint

Once the diagram is split into several smaller cavity diagrams, these can be analyzed for suitable start and endpoints. These are represented by the vertices of the cavity. The general idea here is that the start point is the "deepest" point in the cavity and the endpoints are the "largest" (i.e. corresponding to the largest tetrahedron) boundary vertices.

1.4.1 Start Points

There are two ways to specify a start point for a tunnel - user specified list of residues or computation:

- User defined: The user specifies a list of residues or a 3D point. Next, the centroid s from all the corresponding atomic centers is computed. Then, for each cavity within a specified radius s is "snapped" to the closest vertex which is then marked as the start point.
- Calculated: The topology of the cavity is used to calculate the start point. The calculated starting point is the "deepest" vertex of a cavity. The depth of a vertex v is defined as the length of the path from v to the closest boundary vertex.

1.4.2 End Points

Similarly to the start points, endpoints are either user defined or calculated:

- User defined: By clicking on the surface of the molecule.
- Calculated: TODO.

1.5 Finding the Tunnels

Now that the set of start and end points is identified for each cavity, the Dijkstra's Shortest Path algorithm can be used to find the tunnels between all pairs of start- and endpoints using the weight function w^* defined above.

1.6 Outline of the Algorithm

1.7 Tunnel Representations and Properties

- lining
  TODO: Kratka sekce o vlastnostech.

1.8 The Complexity of the Algorithm

The worst case complexity of the algorithm is $O(M \log M)$ where M is the number of vertices of the Voronoi diagram. In the worst case, $M = N^2$ where $N =$ num. of atoms. However, as shown in [**?**], in most of the cases we have $M \approx N$. The complexity of calculating the Voronoi diagram is $O(N \log N)$. The complexity of all steps of the pre-processing phase is at most $O(M \log M)$. Finally, finding the paths using the Dijkstra's algorithm is $O(KM \log M)$ where $K =$ (num. of tunnels).

References

TODO