Tunnel Computation Methods Section

October 6, 2011

1 Methods

The tunnel computation is done in several steps. First, the voronoi graph representation of the molecule is computed. Next, the graph is analyzed for cavities. Within these cavities, suitable starting and ending points of tunnels are identified (either automatically or user defined). Finally, the Dijksra's Shortest Path algorithm is used to find the shortest paths between the identified start and end points. Additionally, we present a metric and an algorithm for comparing tunnels in different proteins.

1.1 Voronoi Diagram Representation

A Voronoi diagram divides a metric space according to the distances between discrete sets of specified objects. In the cases considered here, the objects are the centers of protein atoms represented by van der Waals 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 space solely according to atomic centers and does not take differing atomic radii into account.

The Voronoi diagram is a graph dual to the Delaunay triangulation of the VDW atom centers. The vertices of the diagram correspond to the tetrahedrons of the triangulation and the edges represent neighbor sides of the tetrahedrons. As a consequence, every vertex of the diagram has degree 4 (if we assume one "inifinite face" v_∞ at the boundary). We call a vertex *boundary* if it has a neighbor that corresponts to v_∞. Also, to each vertex v of the diagram we assign a 3D vector that is the center of the circumsphere of the corresponding tetrahedron. The volume $V(v)$ of vertex v is the volume of the corresponding tetrahedron minus the intersection of VDW spheres of the atoms that form the tetrahedron's vertices. However, in praxis we approximate this volume by replacing the sphere intersection volume with a small tetrahedron created by the atom's center and the intersections of the sphere with the tetrahedron's edges.

1.2 Tunnel Computation

In the old version, the used was required to specify the starting point of a tunnel and then the depth first search approach was used to search for the tunnels [**?**]. Each time

a boundary vertex was reached, the a tunnel get reported. To determine the cost of the path (tunnel), the edge weight function w was used:

$$w(e) = \frac{l(e)}{d_c(e)^2 + \epsilon},$$

where $l(e)$ is the length of the edge, $d_c(e)$ is the distance to the closest van der Waals sphere, and ϵ is a small number to avoid division by zero.

There were numerous problems with this approach. For example, when the first tunnel got reported, the next one was very similar to the first one, because the paths started to "branch" near the end points of the tunnel, and therefore added no additional useful information. Another issue ware small "ridges" that were formed by vertices near the boundary. As a result, large part of the reported tunnels were actually "going along" the surface of the molecule and again provided little relevant information. The current version of the algorithm adresses these issue by preprocessing the Voronoi diagram and splitting it into several smaller graphs before Dijkstra's algorithm is applied to find the tunnels.

1.3 Preprocessing the Diagram

The preprocessing works in several phases:

- If we simply compute the diagram from VDW centers, the boundary of the diagram corresponds to a convex hull of the molecule. This is not desirable, because then the tunnel exits might end up beeing too far from the actual protein surface. To remedy this, we let the user specify *probe radius*, which is used to approximate the molecular surface. To achieve this, layers of vertices are removed starting from the boundary layer. For each tetrahedron it is determined if the probe would pass thru it and if so, it is removed and it's neighbors are marked as boundary. This process is repeated so long as the is no longer any tetrahedron to remove.

- The second step is to remove the vertices that cannot be part of any tunnel. We let the user specify a prameter called *interior threshold*. Using this parameter, all tetrahedrons (vertices) thru which a probe with the specified radius would not pass are removed. As a result of this step, the diagram gets split into several smaller so called *cavity graphs*.

- The last preprocessing step is to remove the so called *shallow vertices*. These are vertices that form the above mentioned ridges along the surface of the molecule and can distort the result. In order to asses the shallow vertices, the depth calculation algorithm described bellow is used. Afterwards, all vertices with depth less than the given threshold (say 5) are removed if all their neighbors have the same or lower depth.

1.4 Tunnel Start and Endpoint Detection

Another problem with the old algorithm was the specification of the tunnel start and end points. We solve the problems in the following way.

1.4.1 Start Points

There are two ways to specify a starting point for a tunnel:

- Selecting several residues. In this case, the centeroid of these residues is computed and the center of the closest "cavity vertex" is chosen as the starting points. The residues can also be obtained from a database such as CAS ([**?**]). The parameter called *origin radius* is used to determine the maximum distance of the vertex and the user defined origin.
- The starting points are calculated from the topology of the cavity.

The calculated starting points are chosen as the "deepest" vertices of the cavity. The depth of vertex v is defined as the length of the path from v to the closest boundary vertex. We define the function h which assigns to each vertex of the cavity its depth. The function is computed by the following algorithm:

---

for $u \in V(G)$ **do**
  $h(u) \leftarrow 0$ if u is boundary, ∞ otherwise
end for
for $u \in V(G)$ where u is boundary **do**
  $q \leftarrow$ `CreateQueue`(u) // Create a FIFO queue
  while q is not empty **do**
    $u \leftarrow$ `Dequeue`(q) // Remove the top vertex
    for $(u, v) \in E(G)$ **do** // each edge containing the vertex u
      if $h(u) + 1 < h(v)$ **then**
        $h(v) \leftarrow h(u) + 1$
        `Enqueue`(q, v) // Add v to the queue
      end if
    end for
  end while
end for

The "real life" complexity of this algorithm is $O(N)$ where N is the number of vertices. This is because for each boundary vertex every vertex in the graph is visited at most once and the number of boundary vertices of a cavity is usually rather low compared to the total number of vertices.

The function h has one undesirable property. Namely that it can happen that $h(u) = h(v)$ for some edge (u, v). To remedy this situation, we collapse every edge with this

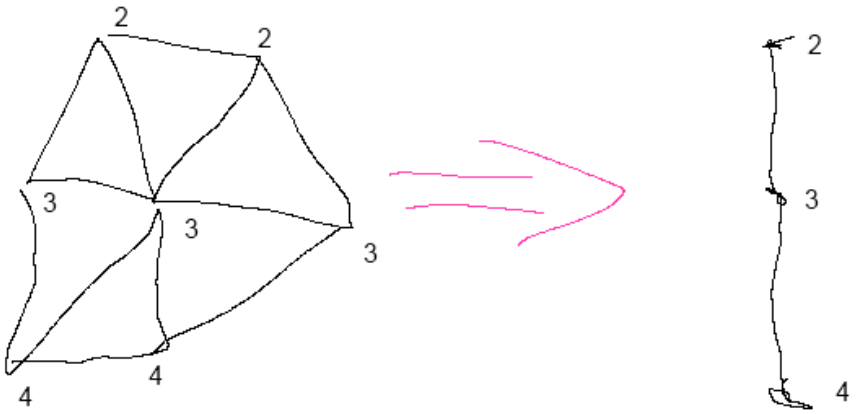

Figure 1: Collapsing cavity graph vertices.

property into either vertex u or v. The vertex whose corresponding face has larger volume is picked. This process is illustrated in figure 1.

Finally, as the set of starting points we pick the vertices v for which the condition $h(v) > h(u)$ holds for all neighbor vertices. Such vertices are illustrated in figure 2.

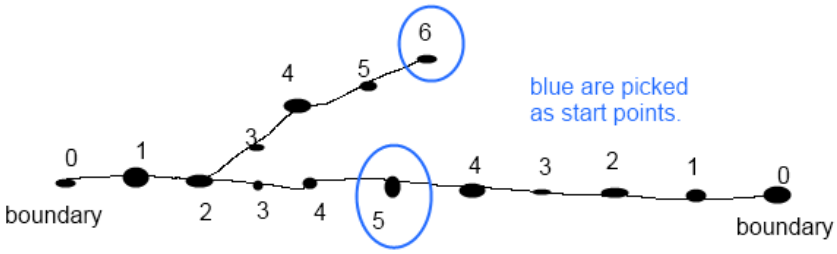

Figure 2: Starting points.

1.4.2 End Points

Similar to the start points, end points are either user defined (by clicking on the boundary of the molecular surface) or calculated.

The calculation is very straighforward. For each cavity graph a subgraph B is induced by the boundary vertices. All vertices that do no fit a probe with so called *opening threshold* are removed. Then, strongly connected components of B are computed. For each component, the vertex with the highest volume is picked as a tunnel end point.

1.5 Tunnel Computation

Now, having the start an end points, the tunnel calculation itself is very straightforward - Dijkstra's algorithm is used to compute the shortest paths between all pairs of start and end points. The function w used for edge weights is the same as in the previous version:

$$w(e) = \frac{l(e)}{d_c(e)^2 + \epsilon},$$

where $l(e)$ is the length of the edge, $d_c(e)$ is the distance to the closest van der Waals sphere, and ϵ is a small number to avoid division by zero.

1.5.1 Representing the Tunnels

The result of the Dijstra search is a set of tunnels each represented as a sequence of vertices. This is not a very desirable representation, therefore we fit a natural spline thru these points. Afterward, the spline is sampled and the several closest atoms, usually 10, are found for each point (actually, the closest point on the VDW sphere is found). These atoms are used to determine the radius of the tunnel at the given point and the list of surrounding residues. Information where the tunnel passes along the backbone is also provided. The sampling rate of the spline varies - it's 1.5 points per angstrom for visualization and 8 points per angstrom for tunnel profile (graph showing tunnel length and radius).

1.6 Tunnel Comparison

The tunnel comparison tool is used to mostly visually compare system of tunnel in different proteins. This works by letting the user select 3 residues which are used to align 2 or more proteins. After the proteins are aligned, a number representing the mutual distance of tunnels in the proteins is computed.

1.6.1 Superimposing the Molecules

The proteins are superimposed using 3 user selected residues. Centroids of these residues are used to align the molecules. To align 2 molecules, we use the quaternion based algorithm presented in [**?**]. If multiple molecules are being compared, they are all superimposed to the first one.

1.6.2 Tunnel distance

DO THIS AS 2 POINT SET DISTANCE

Once the molecules are superimposed, we are ready to compare the tunnels. The comparison only takes into account the centerlines of the tunnels which is represented

by a function $T : \mathbb{R} \to \mathbb{R}^3$ which maps the interval $< 0, 1 >$ onto points of the spline. We then define the so called *directed tunnel distance* as

$$\vec{d}(A, B) = \frac{\int_0^1 \|A(t) - c_B(A(t))\|^2 \mathrm{d}t}{\|A\|},$$

where $c_B(x)$ is the closest point on curve B to the point x.

Because in general $\vec{d}(A, B) \not= \vec{d}(B, A)$ we define the *tunnel distance* as

$$d(A, B) = \frac{\vec{d}(A, B) + \vec{d}(B, A)}{2}$$

To compute the function d we approximate both curves using N sample points (usually 4 points per angstrom) and the computation reduces to

$$\vec{d}_N(A^N, B^N) = \frac{\sum_{i=1}^N \|A_i^N - c_B^N(A_i^N)\|^2}{\|A^N\|}$$

and

$$d_N(A, B) = \frac{\vec{d}_N(A^N, B^N) + \vec{d}_N(B^N, A^N)}{2}$$

respectively.

To compute the function c_*^N, kD-tree [**?**] of the curve is used.

If more than 2 molecules are compared, the average distance between all pairs is returned.