PEPTOID DESIGN WITH ADAPTIVE GEOMETRIC SEARCH

1. Introduction

Peptoids are artificial proteins synthesised in the chemical labs, first developed around 20 years ago (ref). To endorse them with important chemical functions such as binding the ends of side chains with metal ions, their side chains need to be of certain types to be able to stablize in positions close to the binding configurations. When the peptoids' low energy states resemble these binding configurations, they are more likely to bind with metal ions since chemical structures tend towards low energy states. Various metal bindings are observed in natural proteins so we can measure these binding positions. Then we can synthesize peptoids whose low energy states resemble these observed binding positions in natural proteins that bind. However, for one peptoid design the choices of side chains on one particular backbone are exponential and impossible to be carried out in labs with the brute force of synthesising and testing them all. Therefore we need an efficient methodology to predict the low energy states of these peptoids to be synthesised. Predicting the low energy states of proteins is often called the hard problem of protein folding, which is the problem we are trying to solve in the case for peptoids.

Our approach in peptoid design is a two step process. In the first step, we consider the most influential energies in protein folding. Then an efficient geometric search is conducted to eliminate all the impossible designs. In the second step, designs that passed the quick initial screening will be further examined in the comprehensive protein folding software called Rosetta. This two step process efficiently saves all the time that the majority impossible designs would take to be evaluated by Rosetta and to be synthesised.

2. Adaptive geometric search

In peptoids, since energies of bond angles and bond lengths are larger than energies of torsion angles etc. by a magnitude of 100 or more(be more precise), in the first step we are going to consider just the torsion angles as variables in the energy function. Furthermore we are going to consider only the lowest energy bond lengths and bond angles because with all other minor energies ignored, this condition gives the lowest energy configuration of the peptoid.

Without loss of generality, consider the binding configuration as a polygon whose vertices are the binding sites of the configuration. For each binding site or each vertex of the polygon, all the possible positions of the potential binding node (often the end node) on all possible side chains form a manifold. The task is to find one potential binding node from each manifold such that the polygon whose vertices are these potential binding nodes is a "desirable configuration." Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be the target binding configuration. In this paper all polygons are denoted by putting curly brackets around their vertices. We define the error ϵ of a configuration or polygon $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ by its distance to the target configuration $\mathcal{P}$, that is,

$$\epsilon = \max_{1 \leq i,j \leq k} |P_iP_j - S_iS_j|.$$

Following the standard notation, we use capital letters to denote a point in space and two points written next to each other to denote the length of the line segment connecting them. Let ϵ_T be the maximum error we allow due to the negligence of smaller energies and the discretisation of manifolds in computing. If $\epsilon < \epsilon_T$, we call configuration or polygon $\mathcal{S}$ a "desirable configuration".

To search for the desirable configurations, the algorithm first builds an octree with initial length ℓ_0 for each manifold based on sample points with the stopping criterion the minimum cube length ℓ_s. Notice that the octrees are balanced and all leaf cubes are on the lowest level in the tree. Then the algorithm searches adaptively by refining the cubic regions that pass the necessary condition 1 until it reaches the leaf cubes. Then the sufficient condition 2 is tested on all pairs of the leaf cubes. Based on this condition we either accept or reject all pairs of points inside the leaf cubes. Moreover we compare two octrees at a time and then combine all the pairwise results through the matrix product trace.

Given a target polygon $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, a tolerance $\epsilon_T \geq 0$ and one edge (P_i, P_j). Let $\mathcal{C}_i, \mathcal{C}_j$ be two cubes with side length ℓ and the distance between their centers d. Then we have the following theorems.

Theorem 1. *If $d < P_iP_j - \epsilon_T - \sqrt{3}\,\ell$ or $d > P_iP_j + \epsilon_T + \sqrt{3}\,\ell$, then there are no polygons $\{\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_n\}$ within ϵ_T distance from polygon $\mathcal{P}$ where $\tilde{P}_i \in \mathcal{C}_i, \tilde{P}_j \in \mathcal{C}_j$.*

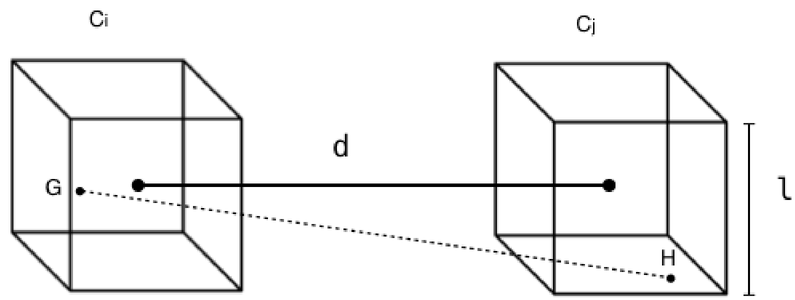

FIGURE 1.

Proof. As illustrated in Figure 1, if $d < P_iP_j - \epsilon_T - \sqrt{3}\,\ell$, by triangle inequality for any points $G \in \mathcal{C}_i, H \in \mathcal{C}_j$,

$$GH \le d + \sqrt{3}\,\ell < P_iP_j - \epsilon_T - \sqrt{3}\,\ell + \sqrt{3}\,\ell = P_iP_j - \epsilon_T.$$

If $d > P_iP_j + \epsilon_T + \sqrt{3}\,\ell$, by triangle inequality for any points $G \in \mathcal{C}_i, H \in \mathcal{C}_j$,

$$GH \ge d - \sqrt{3}\,\ell > P_iP_j + \epsilon_T + \sqrt{3}\,\ell + \sqrt{3}\,\ell - \sqrt{3}\,\ell = P_iP_j + \epsilon_T.$$

So either way GH can not be an edge of any polygon $\{\tilde{P}_1, \tilde{P}_2, \dots, \tilde{P}_n\}$ which is within ϵ_T distance from polygon $\mathcal{P}$. □

Thus Theorem 1 suggests a necessary condition to determine whether two cubic regions are possible to contain any desirable pairs of points. There are other necessary conditions based on positions of points in the cubes, but in comparison this condition is tighter and more efficient for computation. On the other hand, there are also sufficient conditions based on the distance d between the centers of the pairs of cubes.

Theorem 2. *If $P_iP_j - \epsilon_T + \sqrt{3}\,\ell \le d \le P_iP_j + \epsilon_T - \sqrt{3}\,\ell$, then all pairs of points from $\mathcal{C}_i, \mathcal{C}_j$ are within ϵ_T distance from P_iP_j.*

Proof. As illustrated in Figure 1 above, for any points $G \in \mathcal{C}_i, H \in \mathcal{C}_j$, we have $d - \sqrt{3}\,\ell \le GH \le d + \sqrt{3}\,\ell$. If $P_iP_j - \epsilon_T + \sqrt{3}\,\ell \le d \le P_iP_j + \epsilon_T - \sqrt{3}\,\ell$, then substituting the tighter bound of d on each side of the inequality we have $P_iP_j - \epsilon_T \le GH \le P_iP_j + \epsilon_T$. □

Notice that the condition of Theorem 2 is only possible when $P_iP_j - \epsilon_T + \sqrt{3}\,\ell \le P_iP_j + \epsilon_T - \sqrt{3}\,\ell$, or $\ell \le \epsilon_T/\sqrt{3}$. Since the leaf cubes of the octree must have length $\ell_T \le 2l_s$, we require $\ell_s \le \epsilon_T/(2\sqrt{3})$ so that we have $\ell_T \le \epsilon_T/\sqrt{3}$ and Theorem 2 can be applied.

Let $t_1, t_2, \ldots, t_n$ be the octrees generated by each manifold. Algorithm 1 gives the pseudo code of the adaptive geometric search algorithm.

Algorithm 1 Adaptive Geometric Search ($\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}, \mathcal{P}, \epsilon_T$)

$trees$ = $[t_1, t_2, \ldots, t_n]$
for i in range(n) **do**
    $pairs$ = []
    $l^* = P_i P_{(i+1)\, mod\, n}$
    t_1, t_2 = trees[i], trees[(i+1) mod n]
    $combos$ = [[] for x in range($t_1.depth + 1$)]
    $combos[0]$ = [($t_1.root, t_2.root$)]
    for i in range($t_1.depth$) **do**
        for (b_0, b_1) in $combos[i]$ **do**
            $combos[i+1] \mathrel{+}= Compare1(b_0, b_1, l^*, \epsilon_T)$
        end for
    end for
    for (b_0, b_1) in $combos[t_1.depth]$ **do**
        $pairs \mathrel{+}= Compare2(b_0, b_1, l^*, \epsilon_T)$
    end for
    $M_i = \{M_{i,j} = 1$ for $(p_i, p_j) \in pairs; M_{i,j} = 0$ otherwise$\}_{n \times n}$
end for
Trace($\prod_{i=1}^{n} M_i$)

function COMPARE1($b_0, b_1, l^*, \epsilon_T$)
    return $[(c_i, c_j)$ for (c_i, c_j) in $b_0.children \times b_1.children$ if $|(c_i.center, c_j.center) - l^*| \leq \epsilon_T + \sqrt{3}\, c_i.length]$
end function

function COMPARE2($b_0, b_1, l^*, \epsilon_T$)
    if $|(b_0.center, b_1.center) - l^*| \leq \epsilon_T - \sqrt{3}\, b_0.length$ **then**
        return $[(b_0, b_1)]$
    else
        return $[]$
    end if
end function

3. Analysis of the Algorithm

The algorithm has three parts, building the octrees, the adaptive search on two octrees and the matrix product trace computation. First let's consider the adaptive search part of the algorithm. Let N be the number of sample points in each manifold. The time complexity of building an octree with initial cube length ℓ_0 and minimum cube length ℓ_s is $\mathcal{O}(\log_2(\ell_0/\ell_s)N)$.

Next we compute the time complexity of the adaptive search.

Lemma 3. *For any cube $\mathcal{C}_1$ in an octree t_1, there are at most $\frac{4}{3}\pi \cdot (\frac{3\sqrt{3}}{2} + \frac{\ell^* + \epsilon_T}{\ell_s})^3$ many cubes $\mathcal{C}_2$ on the same level from another octree t_2 such that $(\mathcal{C}_1, \mathcal{C}_2)$ are possible pairs.*

Proof. Let ℓ be the length of $\mathcal{C}_1$. For any possible cube $\mathcal{C}_2$ on the same level from t_2, the distance between them d must satisfy that $d \leq \ell^* + \sqrt{3}\,\ell + \epsilon_T$. Thus all possible cubes $\mathcal{C}_2$ must be contained in the sphere $\mathcal{S}$ of radius $\ell^* + \sqrt{3}\,\ell + \epsilon_T + \frac{\sqrt{3}}{2}\,\ell$. Since there are no overlapping cubes on the same level in t_2, the maximum number of the possible cubes n_{max} satisfies

$$
\begin{aligned}
n_{max} \leq \frac{Vol(\mathcal{S})}{Vol(\mathcal{C}_2)} &= \frac{\frac{4}{3}\pi(\ell^* + \sqrt{3}\,\ell + \epsilon_T + \frac{\sqrt{3}}{2}\,\ell)^3}{\ell^3} \\
&= \frac{4}{3}\pi \cdot (\frac{3\sqrt{3}}{2} + \frac{\ell^* + \epsilon_T}{\ell})^3 \\
&\leq \frac{4}{3}\pi \cdot (\frac{3\sqrt{3}}{2} + \frac{\ell^* + \epsilon_T}{\ell_s})^3.
\end{aligned}
$$

□

For convenience we define $C_m = \frac{4}{3}\pi \cdot (\frac{3\sqrt{3}}{2} + \frac{\ell^* + \epsilon_T}{\ell_s})^3$ for the following theorems.

Theorem 4. *In terms of the minimum cube length ℓ_s, the time complexity of the adaptive search algorithm is $\mathcal{O}\left((\ell_0/\ell_s)^3\right)$.*

Proof. Let d_1 be the depth of the octree t_1. Recall that ℓ_0 denotes the length of the root cube of t_1. Since all cubes have minimum length ℓ_s, we have $\ell_0/2^{d_1} \geq \ell_s$, or $d_1 \leq \log_2(\ell_0/\ell_s)$. Then by Lemma 3, the total number of computations $\mathcal{N}$ satisfies

$$
\begin{aligned}
\mathcal{N} &\leq \mathcal{O}(8^2 + C_m \times 8^2 \times \sum_{k=1}^{d_1 - 1} 8^k) \\
&= \mathcal{O}(8^{d_1}) \leq \mathcal{O}\left(8^{\log_2(\ell_0/\ell_s)}\right) \\
&= \mathcal{O}((\ell_0/\ell_s)^3).
\end{aligned}
$$

□

Theorem 5. *If we set $\ell_s = \eta \frac{\epsilon_T}{4\sqrt{3}}$ for any $0 < \eta < 1$, then Algorithm 1 returns all pairs of points whose distances are within $[\ell^* - (1-\eta)\epsilon_T, \ell^* + (1-\eta)\epsilon_T]$, and some but not all pairs of points whose distances are within $[\ell^* - \epsilon_T, \ell^* - (1-\eta)\epsilon_T) \cup (\ell^* + (1-\eta)\epsilon_T, \ell^* + \epsilon_T]$ with time complexity $\mathcal{O}\left((\frac{\ell_0}{\eta\,\epsilon_T})^3\right)$.*

Proof. Let ℓ_T be the length of the leaf cubes. Then we have $\ell_T < 2l_s = \eta\frac{\epsilon_T}{2\sqrt{3}}$. Thus $\ell_T < \epsilon_T/\sqrt{3}$ and hence the sufficient condition 2 is possible. If the sufficient condition 2 is rejected, then the distance d between a pair of cubes $\mathcal{C}_1, \mathcal{C}_2$ satisfies $d > \ell^* + \epsilon_T - \sqrt{3}\,\ell_T$ or $d < \ell^* - \epsilon_T + \sqrt{3}\,\ell_T$. Let G, H be two points such that $G \in \mathcal{C}_1$, $H \in \mathcal{C}_2$. By the triangle inequality and the above inequalities, we have

$$GH \geq d - \sqrt{3}\,\ell_T > \ell^* + \epsilon_T - 2\sqrt{3}\,\ell_T > \ell^* + (1-\eta)\epsilon_T,$$

or

$$GH \leq d + \sqrt{3}\,\ell_T < \ell^* - \epsilon_T + 2\sqrt{3}\,\ell_T < \ell^* - (1-\eta)\epsilon_T.$$

Therefore, in rejecting all pairs of points in $\mathcal{C}_1 \times \mathcal{C}_2$ we may have rejected pairs of points whose distances are within $[\ell^* - \epsilon_T, \ell^* - (1-\eta)\epsilon_T) \cup (\ell^* + (1-\eta)\epsilon_T, \ell^* + \epsilon_T]$. From Theorem 4 the time complexity of the algorithm is $\mathcal{O}((\ell_0/\ell_s)^3)$. Substituting in $\ell_s = \eta\frac{\epsilon_T}{4\sqrt{3}}$ and we have the time complexity equal to $\mathcal{O}((\frac{\ell_0}{\eta\,\epsilon_T})^3)$. □

Now we consider the last part of the algorithm, computation of the matrix product and its trace. Let s_i be the number of ones in matrix M_i for $i = 1, 2, \ldots, n$. Since M_i's are sparse matrices, computing their product has time complexity at most $\mathcal{O}(\sum_{i=1}^n s_i N)$. The complexity of computing the trace of $\prod_{i=1}^n M_i$ is linear in N. Thus the second part has time complexity $\mathcal{O}(\sum_{i=1}^n s_i N + N)$. The trace of $\prod_{i=1}^n M_i$ gives us the total number of desirable polygons that's at most ϵ_T away from the target polygon. In particular if the trace yields a positive number, then there exists such desirable polygons in the current peptoid design.

Therefore the final time complexity of the algorithm is

$$\begin{aligned}
& \mathcal{O}(\log_2(\frac{\ell_0}{\ell_s})N + (\frac{\ell_0}{\ell_s})^3 + \sum_{i=1}^n s_i N + N) \\
= \quad & \mathcal{O}(\log_2(\frac{4\sqrt{3}\,\ell_0}{\eta\,\epsilon_T})N + (\frac{4\sqrt{3}\,\ell_0}{\eta\,\epsilon_T})^3 + \sum_{i=1}^n s_i N + N) \\
= \quad & \mathcal{O}\left(\left(1 + \sum_{i=1}^n s_i - \log_2(\eta\,\epsilon_T)\right) N + (\frac{1}{\eta\,\epsilon_T})^3\right).
\end{aligned}$$

4. Experiment

5. Applications in Computational Geometry

We solved the following problem. Given n sets $S_i, i = 1, 2, \cdots, n$ of points. The problem is to find all n-tuples of points, $(p_1, p_2, \ldots, p_n), p_i \in S_i$ for any i, that make a certain polygon shape within an error tolerance ϵ_T and an approximation margin $\eta\,\epsilon_T$. The problem is closely related to approximate spherical range search (e.g. Sunil Arya [2000]). A naive implementation of the approximate range search for our problem would be for every two manifolds $\mathcal{A}_1, \mathcal{A}_2$ conducting two approximate spherical range searches in the set of sample points A_2 for every point of A_1, resulting in total time complexity $\mathcal{O}(N \log N + (1/(\epsilon_T \eta))^3 N)$.

6. Applications in Computational Algebra

We can use adaptive geometric search to solve particular kinds of systems of equations. In many of these cases, computing Groebner bases is too slow to check if the system is consistent and looking for a solution.

If a system of equations includes at least one equation that can be written into the following form

$$\sum_{i=1}^{k} (f_i(x) - g_i(y))^2 = C,$$

then it can be viewed as a distance equation between points $(f_1(x), f_2(x), \ldots, f_k(x))$ and $(g_1(y), g_2(y), \ldots, g_k(y))$. Thus we separate the system of equations into the distance equations and the rest. We can always generate a set of manifolds based on the rest of the equations and use the adaptive geometric search algorithm to find all tuples of points, which are the solutions of the system, that satisfy the distance equations and lie on the manifolds. More specifically, after possibly needed changes of variables, let S_1 be the set of variables that appear in any of distance equations. Let S_2 be the set of variables that appear in all the other equations in the system. If $S_2 \subset S_1$, then we are ready to apply the adaptive geometric search algorithm. If however $S_2 \not\subset S_1$, then all variables in $S_2 \setminus S_1$ need to be scanned in a grid search before applying the adaptive geometric search algorithm, which may be inconvenient or not.

To generalize, we can view a system of equations as consisting of two parts. One part describes the manifolds, and the other part describes the geometric shape on the manifolds that we search for. If one can formulate a set of necessary conditions on geometric regions to contain solutions, then we can devise the adaptive geometric search algorithm according to these necessary conditions. In this way we can perhaps solve a broader category of systems of equations using adaptive geometric search algorithm.

References

D. M. M. Sunil Arya. Approximate range searching. *Computational Geometry: Theory and Applications*, 17:135–163, 2000.