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,

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

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 discretisation of manifolds in computing. If $\epsilon < \epsilon_T$, we call configuration or polygon $\mathcal{S}$ a "desirable configuration".

Given a target polygon $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, a tolerance $\epsilon_T \ge 0$ 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 results.

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

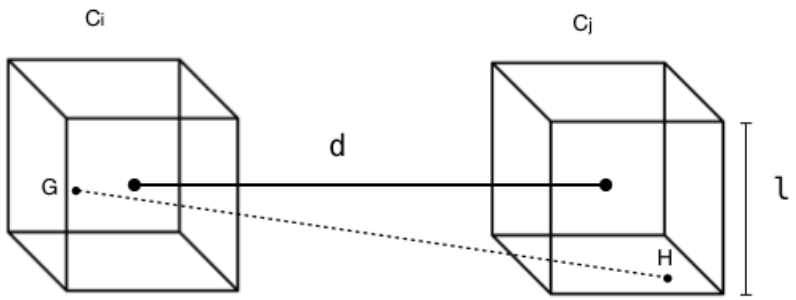

FIGURE 1.

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

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

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

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

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

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, sufficient conditions exist for all pairs of points from a pair of cubes being desirable. Again we use the sufficient condition based on the center positions of cubes.

Theorem 2. *If $P_iP_j - \epsilon_T + \sqrt{3}\,l \leq d \leq P_iP_j + \epsilon_T - \sqrt{3}\,l$, 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 $d - \sqrt{3}\,l \leq GH \leq d + \sqrt{3}\,l$. If $P_iP_j - \epsilon_T + \sqrt{3}\,l \leq d \leq P_iP_j + \epsilon_T - \sqrt{3}\,l$, then substituting the tighter bound of d on each side of the inequality we have $P_iP_j - \epsilon_T \leq GH \leq P_iP_j + \epsilon_T$. □

Notice that the condition of Theorem 2 is only possible when $P_iP_j - \epsilon_T + \sqrt{3}\,l \leq P_iP_j + \epsilon_T - \sqrt{3}\,l$, or $l \leq \epsilon_T/\sqrt{3}$. Thus we design an adaptive search algorithm which searches for possible cubic regions on each level according to Theorem 1, and checks for sufficiency when the cube length l satisfies $l \leq \epsilon_T/\sqrt{3}$. Pairs of cubes that satisfy the necessary condition but not the sufficient condition are refined into sub-cubes, until when they contain so few points such that a thorough search over every pair of points is not costly. This suggests limiting the maximum number of points in all the cubes, which in turn suggests using the data structure of an octree. However in octrees two nodes in comparison can be of different status, that is, comparison of an intermediate node with a leaf node on the same level. To efficiently refine an area in relation to a point, we need the following theorem.

Theorem 3. *Given a point G, a cube $\mathcal{C}$ with length l and the target distance P_iP_j. Let d be the distance from G to the center of $\mathcal{C}$. If $d > P_iP_j + \epsilon_T + \frac{\sqrt{3}}{2}\,l$ or $d < P_iP_j - \epsilon_T - \frac{\sqrt{3}}{2}\,l$, then there are no polygons $\{\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_n\}$ within ϵ_T distance from polygon $\mathcal{P}$ where $\tilde{P}_i \in \mathcal{C}, \tilde{P}_j = G$. If $P_iP_j - \epsilon_T + \frac{\sqrt{3}}{2}\,l \leq d \leq P_iP_j + \epsilon_T - \frac{\sqrt{3}}{2}\,l$, then GH for all points $H \in \mathcal{C}$ are within ϵ_T distance from P_iP_j.*

Proof. The proof is similar to the proofs of Theorem 1 and 2, except now G is a fixed point and thus $GH \in \left[d - \frac{\sqrt{3}}{2}\, l, d + \frac{\sqrt{3}}{2}\, l\right]$ for any point $H \in \mathcal{C}$. □

Again we check both the necessary and sufficient conditions when comparing a point to sub-cubes of a cubic region. If necessary but not sufficient conditions are satisfied by some cube, we refine the search there until on the last level all cubes either violate the necessary condition, or satisfy the sufficient condition, or are leaf nodes in the octree and all points are exhaustively searched.

Let $t_1, t_2, \ldots, t_n$ be the octrees generated by each manifold. Note that each node in the octree represents a corresponding cube. In particular, the root nodes represent the initial cubes. Recall that a manifold is a set of potential binding sites corresponding to a side chain. We offer two adaptive search algorithms below, a basic version Algorithm 1 and a full version Algorithm 2 that takes into consideration of sufficient conditions and point-cube adaptive search. Notice that the algorithm can be further optimized by checking and deleting cubes on each level over all n octrees.

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**
    $pairs$ = []
    $l = P_i P_{(i+1) \bmod n}$
    t_1, t_2 = trees[i], trees[(i+1) mod n]
    $depth = min(t_1.depth, t_2.depth)$
    $combos$ = [[] for x in range(depth+1)]
    $combos[0]$ = $[(t_1.root, t_2.root)]$
    for i in range($depth + 1$) **do**
        for (b_0, b_1) in $combos[i]$ **do**
            if b_0, b_1 are not leaves **then**
                $combos[i+1]$ += $Compare(b_0, b_1, l, \epsilon_T)$
            else
                $pairs$ += $SearchWithin(b_0, b_1, l, \epsilon_T)$
            end if
        end for
    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 COMPARE(b_0, b_1, l, ϵ_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 SEARCHWITHIN(b_0, b_1, l, ϵ_T)
    return [(p_i, p_j) for (p_i, p_j) in $b_0.containedPoints \times b_1.containedPoints$ if $|p_i p_j - l| \leq \epsilon_T$]
end function

Algorithm 2 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**
    $pairs$ = []
    $l = P_i P_{(i+1) \bmod n}$
    t_1, t_2 = trees[i], trees[(i+1) mod n]
    $depth = min(t_1.depth, t_2.depth)$
    $combos$ = [[] for x in range(depth+1)]
    $combos[0]$ = [$(t_1.root, t_2.root)$]
    for i in range($depth + 1$) **do**
        for (b_0, b_1) in $combos[i]$ **do**
            $l_b = \sqrt{3}\, b_0.length$
            $d =$ distance between b_0, b_1
            if $l - \epsilon_T + l_b \leq d \leq l + \epsilon_T - l_b$ **then**
                $pairs$ += [(p_i, p_j) for (p_i, p_j) in $b_0.containedPoints \times b_1.containedPoints$]
            else if neither b_0, b_1 are leaves **then**
                $combos[i+1]$ += $Compare(b_0, b_1, l, \epsilon_T)$
            else if b_0, b_1 are both leaves **then**
                $pairs$ += $SearchWithin(b_0, b_1, l, \epsilon_T)$
            else
                for p in leaf b_i, $i = 0$ or 1 **do**
                    $pairs$ += $AdaptSearchP(p, b_{1-i}, l, \epsilon_T)$
                end for
            end if
        end for
    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 COMPARE(b_0, b_1, l, ϵ_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 SEARCHWITHIN(b_0, b_1, l, ϵ_T)
    return [(p_i, p_j) for (p_i, p_j) in $b_0.containedPoints \times b_1.containedPoints$ if $|p_i p_j - l| \leq \epsilon_T$]
end function
function ADAPTSEARCHP(p, b, l, ϵ_T)
    $pairs$ = []
    $combos$ = [[] for x in range($b.depth + 1$)]
    $combos[0]$ = [b]
    for i in range($b.depth + 1$) **do**
        for c in $combos[i]$ **do**
            $l_c = \sqrt{3}/2 \times c.length$
            $d =$ distance between p and $c.center$
            if $l - \epsilon_T + l_c \leq d \leq l + \epsilon_T - l_c$ **then**
                $pairs$ += [(p, q) for $q \in c.containedPoints$]
            else if c is not a leaf **then**
                $combos[i+1]$ += $CompareP(p, c, l, \epsilon_T)$
            else
                $pairs$ += $SearchPWIthin(p, c, l, \epsilon_T)$
            end if
        end for
    end for
    return $pairs$
end function

function COMPAREP(p, c, l, ϵ_T)
    return [b_i for b_i in $c.children$ if $|(p, b_i.center) - l| \leq \epsilon_T + \sqrt{3}/2\, b_i.length$]
end function

function SEARCHPWITHIN(p, c, l, ϵ_T)
    return [(p, q) for q in $c.containedPoints$ if $|pq - l| \leq \epsilon_T$]
end function

3. Analysis of Algorithm

The algorithm has two parts, the adaptive search on two octrees and the matrix multiplication (including the computation of the trace) part. Let s_i be the number of ones in matrix M_i for $i = 1, 2, \ldots, n$. Let N be the number of sample points in each manifold. In the peptoid model, we have $N = n_s^{k-1}$ where n_s is the number of torsion angles we sample for each side chain node and k is the length of the side chains. Since M_i's are sparse matrices, computing their product has time complexity at most $\mathcal{O}(\max_{i=1}^{n} s_i N)$. The complexity of computing the trace of $\prod_{i=1}^{n} M_i$ is linear in N. Thus the second part has time complexity $\mathcal{O}(\max_{i=1}^{n} s_i N + N)$. 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. Now let's consider the adaptive search part of the algorithm.

3.1. **Any distribution.**

First we consider that the sample points can come from any distribution. In the worst case scenario, the algorithm finds all pairs of cubes from t_1, t_2 possible, and consequently compares exhaustively every pair of points from $\mathcal{A}_1, \mathcal{A}_2$. Note that although the sufficient condition Theorem 2 helps save computations that need to be done, it doesn't improve the time complexity of the algorithm since we still have to report all pairs of points inside the two cubes. The reporting step does less computation than comparing the distance between every pair of points by a constant factor.

Now let l^* be the target polygon edge length. The following examples show how the worst case scenario may happen. Let $\mathcal{A}_1, \mathcal{A}_2$ be two solid balls that are l^* distance apart with radii $\epsilon_T/2$. Then all pairs of points are going to be compared no matter how many points we sample from the two manifolds since they are all desirable. Another example is the following. Let $\mathcal{A}_1$ be a solid ball with radius ϵ_T. Let $\mathcal{A}_2$ be a sphere concentric with $\mathcal{A}_1$ of radius l^*. Then again all pairs of points are going to be compared no matter how many sample points are produced. Hence in order to avoid the worst case scenario we need to add some additional assumption.

3.2. **Minimum cube length.**

To analyze the time complexity of the algorithm, we modify the adaptive geometric search algorithm by adding a second stopping criteria to the octrees and restricting the minimum cube length on the octrees to some constant l_s, i.e. any node with cube length $l < 2l_s$ stops splitting and becomes a leaf node.

Lemma 4. *For any cube $\mathcal{C}_1$ in an octree t_1, there are at most 73 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. For any cube $\mathcal{C}_1$ in t_1, 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 $d \leq l^* + \sqrt{3}\, l + \epsilon_T$. Thus all possible cubes $\mathcal{C}_2$ must be contained in the sphere $\mathcal{S}$ of radius $l^* + \sqrt{3}\, l + \epsilon_T + \frac{\sqrt{3}}{2}\, l$. Since there are no overlapping cubes on the same level in t_2, the maximum number of the possible cubes n_{max} satisfies

$$\begin{aligned} n_{max} \leq \frac{Vol(\mathcal{S})}{Vol(\mathcal{C}_2)} &= \frac{\frac{4}{3}\pi(l^* + \sqrt{3}\, l + \epsilon_T + \frac{\sqrt{3}}{2}\, l)^3}{l^3} \\ &= \frac{4}{3}\pi(\frac{3\sqrt{3}}{2} + \frac{l^* + \epsilon_T}{l})^3 \\ &\leq \frac{4}{3}\pi \cdot (\frac{3\sqrt{3}}{2})^3 = \frac{27\sqrt{3}}{2}\pi. \end{aligned}$$

Since n_{max} must be an integer, $n_{max} \leq 73$. □

Theorem 5. *Given any manifolds $\mathcal{A}_1, \mathcal{A}_2$, and the corresponding octrees t_1, t_2, the adaptive geometric search algorithm doesn't search through all pairs of cubes from t_1, t_2 for arbitrarily large N_1, N_2.*

Proof. From Lemma 4, if the adaptive search algorithm searches through all pairs of cubes on each level, then the maximum width w_m of both octrees must satisfy $w_m \leq 73$. Let the length of the root cube of t_1 be l_0 and the depth of t_1 be d. Then since all cubes have minimum length l_s, we have $l_0/2^d \geq l_s$, or $d \leq \log_2(l_0/l_s)$. Hence the total number of cubes in the octree t_1, denoted by n_m, must satisfy $n_m \leq w_m d \leq 73 \log_2(l_0/l_s)$. Let n_l be the number of leaf nodes in t_1. Let n_T be the maximum number of points per cube in t_1. Then we conclude

$$N_1 \leq n_T\, n_l \leq n_T\, n_m \leq 73\, n_T \log_2 \left(\frac{l_0}{l_s}\right).$$

The same derivation holds for N_2. Therefore the worst case scenario that all pairs of cubes from t_1, t_2 being possible only occurs for bounded N_1, N_2. □

Theorem 6. *The time complexity of the adaptive search algorithm is $\mathcal{O}\left((l_0/l_s)^3\right)$.*

Proof. Let $\psi_k(t_i)$ denote the number of cubes on level k in the octree t_i for $i = 1, 2$. Let d_i be the depth of the octree t_i for $i = 1, 2$. Recall from Theorem 5 that $d_i \leq \log_2 \left(\frac{l_0}{l_s}\right)$ for $i = 1, 2$. In this proof we assume that when a leaf node from one octree and an intermediate node from the other octree make a possible pair, we keep refining the leaf node as many times as required. This procedure would result in a less efficient algorithm and

thus gives an upper bound of the time complexity of the current algorithm. Then by Lemma 4, the total number of computations $\mathcal{N}$ satisfies

$$\begin{aligned}\mathcal{N} &\leq \mathcal{O}(\psi_1(t_1)\psi_1(t_2) + 73 \times 64 \sum_{k=1}^{d_1-1} \psi_1(t_1)) \\ &= \mathcal{O}(\sum_{k=1}^{d_1-1} \psi_k(t_1)) \\ &\leq \mathcal{O}(\sum_{k=1}^{d_1-1} 8^k) \\ &= \mathcal{O}(8^{d_1}) \\ &\leq \mathcal{O}(8^{\log_2(l_0/l_s)}) \\ &= \mathcal{O}((l_0/l_s)^3).\end{aligned}$$

□

If we set l_s to be a constant factor of the "average distance" between sample points, which is $l_0/\sqrt[3]{N}$, then we can calculate the time complexity in terms of N.

Corollary 7. *If we set $l_s = Cl_0/\sqrt[3]{N}$ for some constant C, then the adaptive geometric search algorithm's time complexity is $\mathcal{O}(N)$.*

Proof. By Theorem 6 the time complexity of the algorithm is upper bounded by $\mathcal{O}((l_0/l_s)^3)$, which is equal to $\mathcal{O}(N)$ after substituting in $l_s = Cl_0/\sqrt[3]{N}$. □

3.3. **Rotational invariance.** Often we are building the same side chains that grow from different take-off sites and thus have different rotation angles with the backbone. In this case the manifolds are rotational copies of each other and thus their octrees have roughly the same structure. Under this condition and not assuming any regularity of the distribution of sample points, we have the following theorem.

Theorem 8. *If octrees t_1, t_2 have the same structure, then the ratio of the total number of the possible pairs of cubes over the total number of all pairs of cubes is bounded by C/N for some constant C. Furthermore, the total number of the possible pairs of cubes is bounded by $C'N$ for some constant C'.*

Proof. Let N be the total number of sample points in the octree t_i, d be the depth of the octree t_i and ψ_k be the total number of cubes on level k in the octree t_i, for $i = 1, 2$. Let n_T be the stopping threshold for both octrees,

i.e. the maximum points per cube. Since the total number of cubes in either octree, denoted by n_m, satisfies $n_m \geq N/n_T$, we can conclude that there exists a $k^* \in [0, d]$ such that $\psi_{k^*} \geq n_m/d \geq N/(n_T d)$. Let n_p be the number of all possible pairs of cubes between t_1, t_2, and n be the number of all pairs of cubes between t_1, t_2. Then by Lemma 4 the ratio n_p/n satisfies

$$\frac{n_p}{n} \leq \frac{73 \sum_{k=0}^{d} \psi_k}{\sum_{k=0}^{d} \psi_k \psi_k} \leq \frac{73\, N}{\psi_{k^*}^2} \leq \frac{73\, N}{(N/n_T d)^2} = \frac{73\, n_T^2 d^2}{N}.$$

Now we bound the total number of pairs of cubes n by

$$n = \sum_{k=0}^{d} \psi_k^2 \leq d \max_{k \in [0,d]} \psi_k^2 \leq dN^2.$$

Note that $\psi_k \leq N$ for any $k \in [0, d]$ since all cubes are nonempty and no points are contained in two cubes on the same level. Recall from Theorem 5 that $d \leq \log_2 \left(\frac{2l_0}{(\sqrt[3]{n_T} - 1)\delta} \right)$. Therefore we have

$$\begin{aligned} n_p &\leq \frac{73\, n_T^2 d^2}{N} \cdot n \\ &\leq \frac{73\, n_T^2 d^2}{N} \cdot dN^2 \\ &\leq 73\, n_T^2 \left(\log_2 \left(\frac{2l_0}{(\sqrt[3]{n_T} - 1)\delta} \right) \right)^3 N. \end{aligned}$$

□

4. Experiment

5. Applications in Computational Geometry

We basically 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. One distinction from other more efficient algorithms for approximate spherical range search (e.g. **?**) is that our algorithm searches out all and only the n-tuples that are within the error tolerance, whereas most of the other faster methods produce only some results within the tolerance. So ask a result of using these approximate methods, we can miss some working designs of peptoids. The adaptive geometric search method increased both the time and space complexity from the exhaustive search.

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.