A Cooperative Combinatorial Particle Swarm Optimization Algorithm for Side-chain Packing

Grecia Lapizco-Encinas, Carl Kingsford and James Reggia

***Abstract*—Particle Swarm Optimization (PSO) is a well-known, competitive technique for numerical optimization with real-parameter representation. This paper introduces CCPSO, a new Cooperative Particle Swarm Optimization algorithm for combinatorial problems. The cooperative strategy is achieved by splitting the candidate solution vector into components, where each component is optimized by a particle. Particles move throughout a continuous space, their movements based on the influences exerted by static particles that then get feedback based on the fitness of the candidate solution. Here, the application of this technique to side-chain packing (a proteomics optimization problem) is investigated. To verify the efficiency of the proposed CCPSO algorithm, we test our algorithm on three side-chain packing problems and compare our results with the provably optimal result. Computational results show that the proposed algorithm is very competitive, obtaining a conformation with an energy value within 1% of the provably optimal solution in many proteins.**

I. Introduction

PROTEINS are a fundamental class of molecules that are involved in nearly every process of life. Proteins are linear chains of amino acids that generally fold into compact three-dimensional structures. Each amino acid within the chain is called a *residue*. A protein's sequence of amino acids, called its *primary structure*, determines many of the characteristics of the protein, including strongly influencing its three-dimensional, *tertiary* structure. Discovering the tertiary structure of a protein can provide important clues about how the protein performs its function, and is an essential problem in molecular biology [1]. A particularly important subproblem of the general structure prediction problem is the side-chain packing problem, in which the side chains of the residues are positioned on a fixed and given backbone. It is this subproblem that we tackle here.

Protein structure prediction methods are faced with imprecise knowledge of many aspects of the physical forces that drive protein folding. Therefore, it has been argued that instead of providing the exact optimum solution to an imprecise energy function, computational methods should instead produce robust, fast, and near-optimal solutions [2]. This makes algorithms like PSO, which are known to efficiently produce near-optimal solutions, especially attractive. Another swarm intelligence method, Ant Colony Optimization, has already been applied to some protein structure prediction tasks, e.g. [3]-[5].

Grecia Lapizco-Encinas, Carl Kingsford, and James Reggia are with the Department of Computer Science, University of Maryland, College Park, MD (email: glapizco@cs.umd.edu, carlk@umiacs.umd.edu, reggia@cs.umd.edu). Supported in part by NSF ITR Award IIS-0325098. C.K. was partially supported by the NSF grant IIS-0812111.

Particle Swarm Optimization (PSO) was originally designed as a numerical optimization technique based on swarm intelligence [6][7], and it has shown its robustness and efficacy for solving function-value optimization problems in real-number spaces [7][8]. In the PSO framework, a set of particles (simple agents) search for good solutions to a given optimization problem. Each particle represents a candidate solution to the optimization problem and uses its own experience and the experience of neighbor particles to choose how to move in the search space.

The advantage of the PSO over many other optimization methods is its relative simplicity. However, the PSO algorithm is inherently a continuous algorithm and only a few attempts have been made to extend the PSO to combinatorial optimization problems [8]. The main issue when adapting PSO to any non-continuous problem is how to redefine the relationship between a particle and a candidate solution of the problem. There have been several proposed extensions of PSO to combinatorial spaces with various degrees of success [9]-[11], but all of these approaches sacrifice the basic nature of the PSO of particles moving in a continuous high-dimensional space by encoding the solution as a permutation and designing specialized velocity operators for this permutation space. To improve the performance of the PSO, particularly in high-dimensional problems, some cooperative strategies have been devised. These strategies vary from creating explicit groups of particles and allowing exchanges of particles between them [12]-[14] to partitioning the search space into lower dimensional subspaces, by splitting the solution vectors into smaller vectors [15]-[18].

A previous attempt to partition the search space while retaining the notion of particles moving in an abstract high-dimensional space is reported in [19]. In this algorithm, each particle contributes to the solution by moving independently throughout a shared solution space. There is no explicit division of solution space between particles. This algorithm is applied to diagnostic problem solving based on causal networks. Our work here presents a *cooperative combinatorial* PSO in which each component of the candidate solution is represented by a particle moving on its own continuous, high-dimensional space. The movement of the particles is based on the influences exerted by other static particles. These static particles, called *attractors*, represent choices for a value of a particular component of the problem instance. The candidate solution is then constructed cooperatively by decoding each particle's position into a choice for that solution component. Feedback to the attractors is given by computing a strength value which considers the fitness of the candidate solution with the current component replaced by the choice represented by the attractor. This cooperative scheme aims to partition the space, allowing fine tuning of each component by each particle, when the other components of the candidate solution do not change from one iteration to the next, and exploring a different part of the space and restarting this fine tuning when any component of the candidate solution changes. Our algorithm, CCPSO, is tested in three side-chain packing problems with very encouraging results, obtaining a protein conformation with an energy value within 1% of the optimal in most proteins.

II. Side-chain Packing

The problem of predicting side-chain conformations for each residue given a backbone structure is of central importance in homology modeling and the design of novel protein sequences. Homology modeling is based on the assumption that two proteins with similar sequences will have similar global structures. Therefore an initial backbone can be obtained by searching for a similar amino acid sequence in a database of known protein structures. Side-chain packing methods are then used to place the side chains of the target sequence onto the backbone template. In a protein design application, the goal is to find the sequence of amino acids for a given template backbone that will satisfy the desired structural features. Side-chain prediction algorithms are used to screen all possible amino acid sequences and find the amino acid sequence whose side chains best fit the desired backbone [20]. The same combinatorial problem underlies a natural formulation of both homology modeling and protein design.

Typically, the choice of possible conformations for a side chain is restricted to a library of discrete possibilities. This approximation is based on the observation that, in high-resolution experimental protein structure models, most side chains tend to cluster around a discrete set of preferred conformations, known as *rotamers* [21]. The inclusion of these discrete configurations implies an important problem reduction. Nevertheless, even this discrete version is known to be NP-hard [22] and hard to approximate [23]. Therefore, the conception of efficient search procedures arises as an important research problem. Here, it also serves as an example of a difficult combinatorial optimization problem on which the proposed CCPSO algorithm can be tested.

With the rotamer model, the total energy of a choice of rotamers can be described in terms of the pairwise interactions between the elements of the side-chain conformation:

$$E = E_{backbone} + \sum_{i} E(i_r) + \sum_{i<j} E(i_r, j_s) \tag{1}$$

This incorporates the contribution of three classes of energies: $E_{backbone}$ is the self-energy of a backbone template, $E(i_r)$ is the interaction energy between the backbone and the side chain of residue i in its rotamer conformation r, and $E(i_r, j_s)$ is the interaction energy between residue i in the rotamer conformation r and residue j in the rotamer conformation s. The problem of determining the side-chain conformation of minimum energy is reduced to choosing a rotamer selection for each residue so that Equation (1) is minimized. Search algorithms for side-chain packing fall into two broad categories: stochastic and deterministic. Stochastic algorithms, including simulated annealing [24] and genetic algorithms [25], semi-randomly sample sequence-structure space and move toward lower energy solutions, while deterministic algorithms, such as dead end elimination and its extensions [26]-[28], integer programming [29][30], and other graph search approaches [31][32] perform semi-exhaustive searches [20]. An advantage of stochastic methods, such as PSO, is that they can deal with problems of significant combinatorial complexity because they do not require an exhaustive search.

The side-chain packing problem, *SCP*, can be reformulated as a graph problem by using an interaction graph to represent the residues and their relationships. Each residue is represented by a subgraph that contains a node for each possible rotamer for this residue. Physical interactions between each possible rotamer of different residues are represented by weighted edges between the nodes, such that the edge between rotamer r of residue i and rotamer s of residue j has weight $E(i_r, j_s)$. By assigning a cost to each node equal to its interaction energy with the backbone $E(i_r)$, the global minimum conformation can be found by picking one node from each residue subgraph, such that it minimizes the cost of the entire induced subgraph. Figure 1 shows an example of a SCP with three amino acid residues A_1, A_2, and A_3.

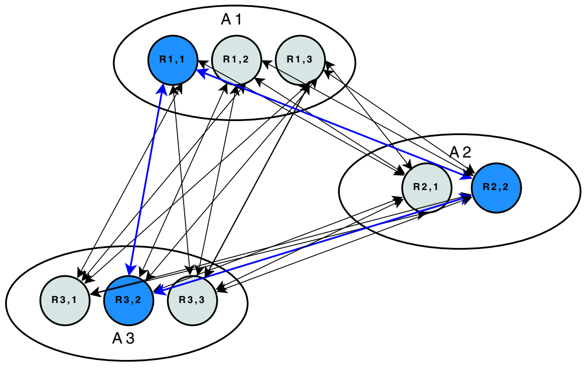

Fig. 1. SCP problem with three amino acid residues A_1, A_2, and A_3. Each amino acid residue is represented by a subgraph that contains all the rotamers that belong to that amino acid, $A_1 = \{R_{1,1}, R_{1,2}, R_{1,3}\}$, $A_2 = \{R_{2,1}, R_{2,2}\}$, $A_3 = \{R_{3,1}, R_{3,2}, R_{3,3}\}$. The solution to this SCP is $sol = \langle R_{1,1}, R_{2,2}, R_{3,2} \rangle$.

III. PSO and its variants

A. Traditional PSO

PSO [6][7] models a set of potential problem solutions as a swarm of particles moving in a virtual search space. Every particle in the swarm begins with a random position ($\vec{x}_i$) and a velocity ($\vec{v}_i$) in the n-dimensional search space, where x_{ij} represents the location of particle i in the j^{th} dimension of the search space. Candidate solutions are optimized by flying the particles through the virtual space. Each particle remembers at which position it achieved its highest performance ($\vec{b}_i$). Every particle is also a member of some neighborhood of particles, and it remembers which particle achieved the best overall position in that neighborhood ($\vec{n}_i$). At each iteration of the algorithm (as shown in Algorithm 1) the position of each particle i is updated. The *cognitive factor* c_1 and *social factor* c_2 are influential in striking a balance between the relative roles of the individual experience (governed by c_1) and of the social communication (governed by c_2). Uniform random selection of these two parameters avoids any a priori importance of either of the two sources of information. Inertia weight w is a parameter within the range [0,1] and is often decreased over time.

Algorithm 1 Traditional PSO

```
for each particle $i$ do
    initialize position $\vec{x}_i$ and velocity $\vec{v}_i$
end for
while stop criteria not met do
    for each particle $i$ do
        evaluate fitness($\vec{x}_i$)
        $\vec{b}_i \leftarrow$ best position found so far by the particle
        $\vec{n}_i \leftarrow$ best position found so far by its neighborhood
    end for
    for each particle $i$ do
        $\vec{v}_i \leftarrow w \times \vec{v}_i + U(0, c_1) \times (\vec{n}_i - \vec{x}_i) + U(0, c_2) \times (\vec{b}_i - \vec{x}_i)$
        $\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$
    end for
end while
```

Extending PSO to more complex combinatorial search spaces is of great interest but it requires a clear definition of the relationship between a particle position and a solution [8]. This representation method, as well as the notions of position and velocity of the original PSO, have no natural extensions in combinatorial space. We now review briefly some relevant attempts to deal with this representation.

B. Discrete and Combinatorial PSO

Since the original PSO algorithm can only optimize problems in which the elements of the solution are continuous real numbers, a modification of the PSO algorithm for problems with binary-valued solution elements was presented in [33]. This algorithm preserves the concept of social and cognitive learning but changes the updates of the particles' features. The velocity equation remains unchanged, except that now x_{ij}, b_{ij} and n_{ij} are integers in $\{0, 1\}$, and the particle's position equation is changed to:

$$x_{ij} = \begin{cases} 1 & \text{if rand()} \leq S(v_{ij}), \text{ where } S(v_{ij}) = \frac{1}{1+e^{-v_{ij}}} \\ 0 & \text{otherwise} \end{cases}$$

The particle's position x_{ij} now contains the solution component (i.e., the value 0 or 1), and the velocity v_{ij} is a probability value constrained to the interval $[0.0, 1.0]$ that indicates the probability of the corresponding solution element assuming the value 1. The notion of velocity of the standard PSO, representing the change the particle's position x_{ij}, is lost. Another way to apply traditional PSO to discrete problems is to simply cast them as continuous optimization problems and round the element of the position vectors to the nearest valid discrete value [34][35]. This approach makes the implicit assumption that discrete values are physically related, i.e, a $x_{ij} = 1$ would move to $x_{ij} = 2$ easier that it would to $x_{ij} = 5$. A more involved approach, MVPSO [36], defines the particles' positions to contain the probabilities of solution elements assuming the specific values. This strategy maintains a notion of position and velocity similar to the traditional PSO, but the accuracy of MVPSO decreases as the number of discrete values increases from binary to ternary [36].

The most common approach of PSO extensions for explicit combinatorial problems, such as the traveling salesman problem, is based on the principle of permutations [37][9][10]. In these algorithms, a particle position is a specific permutation, and the velocity becomes a swap operator that transforms one permutation into another. Various other heuristic rules have been proposed to apply PSO to combinatorial problem while attempting to keep the particles in continuous space. For example, the smallest position value rule [11] maps the positions of the particles to a permutation solution by placing the index of the lowest valued particle component as the first item, the next lowest as the second and so on.

C. Cooperative PSO

In cooperative PSO approaches, particles are explicitly or implicitly grouped into subswarms. Implicit grouping is achieved in the basis of particle behavior and overall state. Cooperation is mainly in terms of exchanging information about best positions found by different groups. In Concurrent PSO, CONPSO [13], two swarms search concurrently for a solution with frequent message passing of information. Breeding PSO developed in [12] has implicit cooperation based on the exchange of genetic material between parents of different subswarms. Multi-Phase PSO, MuPSO [14], divides the swarm into groups of particles that are allowed to exchange particles. Each group can be in an attraction or repulsion phase, where it moves toward or away from the best solutions found recently.

Cooperative Split PSO, CPSO-S, [17], divides the solution into components that are then optimized using a separate PSO for each component. The best particle of each PSO is used to define a *context-vector* which is needed to evaluate the fitness of the particles. The context vector is unique during the evaluation of particular PSO, which means that all particles are evaluated using only one template solution. A hybrid of CSPSO-S and traditional PSO was tested in several continuous optimization benchmark problems with promising results [17]. Similar work is presented in Multi-objective PSO [18] where the best-particle of each PSO is selected based on a partial order such that Pareto ranks are first considered, followed by niche counts.

IV. Cooperative Combinatorial PSO

Here we present a novel PSO algorithm, *CCPSO*, for combinatorial optimization problems that maintains some core concepts of traditional PSO, namely, particles move in a continuous high-dimensional space guided by social and cognitive learning terms. These social and cognitive terms are a result of keeping a memory of the best candidate solution seen so far and the best candidate solution seen by the neighbors. Applying PSO to combinatorial problems, where the relationship between the different elements of the solution is non-linear, allows for the possibility that some elements in the vector have moved closer to the optimal solution, while others actually moved away from the solution. If the cumulative effects of these changes improve over the previous stored solutions, the PSO will update its memory values, guiding the solution to a part of the space that could be farther away from the optimal solution. Previous approaches attempted to avoid this problem by partitioning the search space and optimizing each dimension separately, but this was found to introduce pseudo-minima [17]. The strategy implemented in our algorithm aims to solve the same problem and avoid pseudo-minima by keeping the concept of particles being guided by their own candidate solutions templates that represent the social and cognitive memories.

In combinatorial problems, candidate solutions are represented by a vector, $\mathit{solution} = \langle c_1, c_2, ..., c_n \rangle$ where each component c_i is selected from an ordered set of k possible values $M_i = \{m_{i1}, m_{i2}, ..., m_{ik}\}$ for position i. Note that k varies for each c_i. In CCPSO, each component c_i is represented by a particle p_i. Each possible value in M_i is represented by an attractor r_{ij}. Particle p_i moves in a k-dimensional space where all its attractors r_{ij} are located. Attractors try to pull p_i to its position, i.e, try to assign their value m_{ij} to c_i. The solution is represented by a collection of particles, a *subswarm*, and is decoded from the particle's positions. Each solution component is assigned the value m_{ij} represented by the attractor r_{ij} that is selected as winner for that particle. This winner is selected randomly using the particle's position, $\vec{x}_i$, as the distribution. Therefore, attractors which are more successful in pulling particles towards its position will have a higher chance of being selected as part of the decoded solution.

A. Subswarm

A subswarm represents a candidate solution for the optimization problem. It is composed of a collection of particles, where each particle is optimizing one component of the solution vector. Since a subswarm can be decoded into a candidate solution, its fitness value can be evaluated according to the optimization fitness function. Each subswarm s is defined by the tuple:

$$S_s = \langle \mathit{particles}_s, \mathit{sol}_s, \mathit{fitness}_s, b_s, n_s, \mathit{neighbors}_s \rangle$$

where $\mathit{particles}_s$ is the set of particles that belong to the subswarm, sol_s is the candidate solution decoded from $\mathit{particles}_s$, $\mathit{fitness}_s$ is the fitness value of the solution, $\mathit{neighbors}_s$ is a set of subswarms with which it interacts, b_s represents the cognitive memory of the subswarm, i.e, the candidate solution at which it achieved the best fitness value, and n_s keeps a social memory.

B. Attractor

An attractor represents a value m_{ij} for a particular component c_i in the candidate solution. Attractors are represented by static particles r_{ij} located throughout the k-dimensional space, where k is the number of possible values for this solution component. Extending notation to indicate the subswarm, each attractor j of particle i in subswarm s is defined by the tuple:

$$r_{sij} \;=\; \langle\; \vec{a}_{sij}, st^b_{sij}, st^n_{sij} \;\rangle \tag{2}$$

where $\vec{a}_{sij}$ = its static position, st^b_{sij} represents the strength of this attractor to pull particle p_{si} towards the best solution seen by subswarm s (i.e. b_s), similarly st^n_{sij} represents the strength of this attractor to pull particle p_{si} towards n_s.

The position $\vec{a}_{sij}$ is initialized to a unit vector in a direction of the jth coordinate axis orthogonal to all previous attractors. To update an the strength of an attractor, a new candidate solution $\vec{b}^{\prime\prime}_s$ is constructed by using $\vec{b}_s$ as a template candidate solution, and replacing the value of the component c_i with m_{ij}. The strength value st^b_{sij} is computed proportional to the fitness value of $\vec{b}^{\prime\prime}_s$. "The value st^n_{sij} is updated similarly:

$$st^b_{sij} \propto \mathit{fitness}(\vec{b}^{\prime}_s), \text{ where } \vec{b}^{\prime}_s = \vec{b}_s \text{ except } b^{\prime}_s[i] = j \tag{3}$$

$$st^n_{sij} \propto \mathit{fitness}(\vec{n}^{\prime}_s), \text{ where } \vec{n}^{\prime}_s = \vec{n}_s \text{ except } n^{\prime}_s[i] = j \tag{4}$$

The attractor's task is not to move around the space during the simulation, but instead to pull particles towards its position with a strength that reflects how good that particular value is for the solution. This strength value is used as an heuristic to guide the particles towards better areas of the search space. To explore how frequently this value needs to be updated to be effective we define an update probability value u_s. At each iteration a coin is flipped with u_s probability to determine if the strength values need to be updated.

C. Particles

A particle represents one component in the candidate solution. Particles move throughout a continuous space, their movements based on the influences exerted by attractors. Their positions are used by the subswarm to construct the candidate solution. Each particle i of subswarm s is defined by the tuple:

$$p_{si} = \langle \vec{x}_{si}, \vec{v}_{si} \rangle$$

where $\vec{x}_{si}$ = its current position, $\vec{v}_{si}$ = its current velocity vector. The particle's position $\vec{x}_{si}$ represents the probability of component i taking each possible value. $\vec{x}_{si}[j]$, in particular, represents the probability of selecting value m_{ij} for component c_i. The selection is made at random using $\vec{x}_{si}$ as the distribution. The initial position $\vec{x}_{si}$ is set to a random position in the k-dimensional space, and its initial velocity vector $\vec{v}_{si}$ is set to zero. The updates of the particles are accomplished according to Equations (5) - (6). As with the traditional PSO (refer to Algorithm 1), Equation (5) shows how a new velocity is computed based on three components: its previous velocity, a velocity component that drives the particle towards the location in the search space where it previously found the best solution (i.e, a cognitive-velocity $\vec{v}^{b}_{si}$), and a velocity component that drives the particle towards the location of the best neighborhood solution (i.e, a social-velocity $\vec{v}^{n}_{si}$). Then each particle's position is updated by simply adding the velocity vector (Equation (6)):

$$\vec{v}_{si} \leftarrow w \times \vec{v}_{si} + U(0,c_1) \times (\vec{v}^{b}_{si}) + U(0,c_2) \times (\vec{v}^{n}_{si}) \quad (5)$$

$$\vec{x}_{si} \leftarrow \vec{x}_{si} + \vec{v}_{si} \quad (6)$$

The value of the cognitive-velocity component, $\vec{v}^{b}_{si}$ is the result of all forces exerted by nearby attractors on particle p_i as shown below in Equation (7). Equation (7) shows that this attraction force is dependent upon particular attributes of each attractor: its strength (st^{b}_{sij}), which is proportional to the fitness value of the best memory of the subswarm, and its location (which is used to compute the direction of the velocity). The strength of these attractors are dependant on the best solution seen so far, i.e., its cognitive memory, therefore we name these *cognitive-attractors*. Cognitive-velocity guides the particle towards a position in space which is considered good from the perspective of the best memory of the swarm. The social component of the new velocity is computed in a similar manner in Equation (8):

$$\vec{v}^{b}_{si} \leftarrow \sum_{j} st^{b}_{sij} \times \vec{a}_{sij} \quad (7)$$

$$\vec{v}^{n}_{si} \leftarrow \sum_{j} st^{n}_{sij} \times \vec{a}_{sij} \quad (8)$$

Unlike the traditional PSO, these cognitive and social velocity components can be seen as a local search step, since the attractors strength values (st^{b}_{sij} and s^{n}_{sij}) are estimated by taking into account the fitness value of a candidate solution where that particular value is selected.

D. Algorithm

Algorithm 2 presents the CCPSO pseudocode that combines the components described above. The system is initialized at time $t = 0$ by creating all subswarms. For each subswarm, one particle is defined per dimension of the combinatorial problem. An attractor is created for each possible value of this variable. For each time step, each particle updates its position vector by simply adding a new velocity vector (Equation (6)). This new velocity vector is computed based on the influence exerted by the cognitive and social attractors. Equations (7) - (8) define how to get the resultant force of all attractors, and Equations (3) - (4) show how to the attractors recompute their strength value based on the memory values of the subswarm. The process iterates until all particles have arrived to a stationary position (i.e., the system is in equilibrium) or until a maximum number of time steps has passed.

Algorithm 2 Cooperative Combinatorial PSO

```
for each subswarm $s$ do
  for each particle $i$ do
    initialize position $\vec{x}_{si}$ and velocity $\vec{v}_{si}$
    for each attractor $j$ do
      initialize position $\vec{a}_{sij}$
    end for
  end for
end for
while stop criteria not met do
  for each subswarm $s$ do
    decode solution $sol_s$
    evaluate $fitness(sol_s)$
    $\vec{b}_s \leftarrow$ best solution found so far by the subswarm
    $\vec{n}_s \leftarrow$ best solution found so far by its neighborhood
    for each particle $i$ do
      for each attractor $j$ do
        if $flip(u_s)$
        $st^{b}_{sij} \propto fitness(\vec{b}'_s)$, where $\vec{b}'_s = \vec{b}_s$ with $b'_s[i] = j$
        $st^{n}_{sij} \propto fitness(\vec{n}'_s)$, where $\vec{n}'_s = \vec{n}_s$ with $n'_s[i] = j$
        end if
      end for
      $\vec{v}^{b}_{si} \leftarrow \sum_j st^{b}_{sij} \times \vec{a}_{sij}$
      $\vec{v}^{n}_{si} \leftarrow \sum_j st^{n}_{sij} \times \vec{a}_{sij}$
      $\vec{v}_{si} \leftarrow w \times \vec{v}_{si} + U(0,c_1) \times (\vec{v}^{b}_{si}) + U(0,c_2) \times (\vec{v}^{n}_{si})$
      $\vec{x}_{si} \leftarrow \vec{x}_{si} + \vec{v}_{si}$
    end for
  end for
end while
```

V. CCPSO for Side-chain Packing

To apply CCPSO to the SCP problem we need to define the solution representation and the decoding method. Each particle represents a residue and attractors represent the rotamers of each residue. For the SCP of Figure 1, we have three residues A_1, A_2 and A_3, where A_1=$\{R_{1,1}, R_{1,2}, R_{1,3}\}$, A_2=$\{R_{2,1}, R_{2,2}\}$, and A_3=$\{R_{3,1}, R_{3,2}, R_{3,3}\}$. Therefore, for subswarm s_1 we have *particles*$_1$=$\{p_1, p_2, p_3\}$. For residue A_1 we create p_1 that would move in a continuous k space, where k is the number of rotamer choices. Since $k = 3$ for A_1, particle p_1 moves in a 3-dimensional space. Attractors r_{111}, r_{112} and r_{113} would be in locations $\langle 1,0,0\rangle$, $\langle 0,1,0\rangle$, and $\langle 0,0,1\rangle$ respectively. This is depicted in Figure 2.

In a candidate solution for a SCP problem each component c_i is the rotamer selected for the side chain of residue i. The decoding method maps a subswarm s to a rotamer conformation sol_s. Each particle p_{si} of subswarm s contributes to the candidate solution sol_s with its choice of rotamer for the side chain of residue i. This rotamer is selected randomly by function *winner* which uses the particle's position $\vec{x}_i$ as distribution.

$$sol_s = \langle c_1, c_2, ..., c_n\rangle, \text{ where } c_i = winner(x_{si})$$

An example of a solution for the SCP defined in Figure 1 obtained from a subswarm s is depicted in Figure 3. For

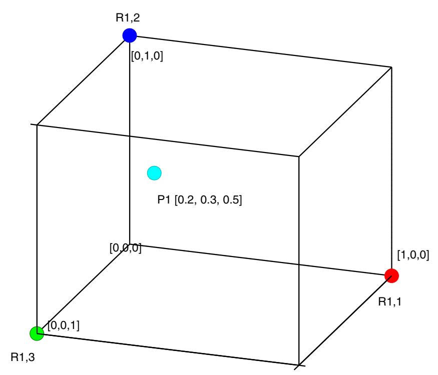

Fig. 2. Example of a particle and its attractors. Particle p_1 moves in a 3-dimensional space. Attractors are located in an orthogonal position to one another and try to pull p_1 towards its location.

With $particles \leftarrow \{p_1, p_2, p_3\}$

$$x_1 \leftarrow \begin{array}{|c|c|c|}\hline R_{1,1} & R_{1,2} & R_{1,3} \\ 0.2 & 0.3 & 0.5 \\ \hline \end{array} \quad winner(x_1) = R_{1,3}$$

$$x_2 \leftarrow \begin{array}{|c|c|}\hline R_{2,1} & R_{2,2} \\ 0.7 & 0.3 \\ \hline \end{array} \quad winner(x_2) = R_{2,1}$$

$$x_3 \leftarrow \begin{array}{|c|c|c|}\hline R_{3,1} & R_{3,2} & R_{3,3} \\ 0.3 & 0.6 & 0.1 \\ \hline \end{array} \quad winner(x_3) = R_{3,2}$$

we obtain the candidate solution

$$sol \leftarrow \langle R_{1,3} \quad R_{2,1} \quad R_{3,2} \rangle$$

Fig. 3. Solution representation for the SCP problem. A solution sol_1 is obtained by selecting a rotamer from each particle using x_i values as distribution.

clarity we omit the index s of the particles. Once a solution sol_s for each subswarm s is obtained, computing its fitness value is straightforward. Equation (9) shows how to compute the fitness value of a solution sol_s. Details about how to compute the energy value of rotamer conformation are given in the next section.

$$fitness(sol_s) = \frac{1}{energy(sol_s)} \tag{9}$$

VI. Experimental Setup

We test our algorithm on combinatorial problems resulting from three protein packing applications: predicting the conformation of a protein's side chains on its native backbone, predicting the structure of the protein using the backbone of a homologous sequence as a template, and the problem of designing novel sequences that fold into a known backbone.

In general, it is not straightforward to compare the results of different protein side-chain packing methods, as different accuracy measures, energy functions, rotamer libraries and proteins have been used. In order to make a fair comparison of search methods, we use the same rotamer library and energy function that a previous study used [30]. For our rotamer library we used Dunbrack's backbone-dependent rotamer library [38]. For each 10° range of ϕ, ψ backbone angles, this library has 320 rotamers, with the largest number of rotamers, 81, belonging to arginine and lysine.

The energy function of the rotamer conformation is computed using Equation (1), except that we dropped the $E_{backbone}$ term since the template backbone is fixed. To compute the self-energy terms $E(i_r)$ and the pairwise rotamer energy $E(i_r, j_s)$ we use the energy functions defined in [30], which are derived from the AMBER force field [39]. $E(i_r)$, the self-energy of rotamer i, is computed using both statistical potentials and van der Waals interactions terms. The statistical term takes into account the prior probabilities of rotamers in a training set so that the more common a rotamer, the lower the energy assigned to it. $E(i_r, j_s)$, the pairwise rotamer energy between rotamers r and s, is the sum of the van der Waals interactions between the side-chain atoms of r and s.

We compare our results with the provably optimal solution, obtained by a deterministic method as reported in [30]. The accuracy of our results is determined by computing the percentage error. Percentage error is defined in Equation (10), where OPT is the energy value of the optimal rotamer conformation and *solution-energy* is the energy value of the best solution found by the CCPSO.

$$error = 100 \times \frac{|OPT - solution\text{-}energy|}{|OPT|} \tag{10}$$

Since our method is stochastic, we execute each instance twenty times and report on the average percentage error, as well as the minimum percentage error (best solution found). The experiments were run on a Intel Xeon 3.2Ghz processor with 1GB of RAM.

VII. Results

The CCPSO parameters used in this paper are listed in Table I. These parameters were selected after preliminary experimentation. Tables II - IV present the results in the three datasets. Columns 1-4 describe the protein while columns 5-7 show the CCPSO results with respect to the optimal solution. The first column gives the protein PDB identifier, the second column indicates how many of its side chains have more than one possible rotamer, and the third gives the total number of rotamers. Fourth column points out the energy of the optimal configuration. Fifth column shows the minimum percentage error obtained by any run of the CCPSO, while the sixth column shows average percentage error over twenty runs. Finally the seventh column indicates the number of incorrect rotamer positions in the best solution obtained by the CCPSO.

TABLE I
CCPSO PARAMETERS

Parameter	Value
Number of subswarms	30
Maximum number of iterations	200
Cognitive factor c_1	1.5
Social factor c_2	1.5
Inertia weight w	1 to 0 (linearly decreasing)
Network-topology	ring
Update probability value u_s	0.4

TABLE II
RESULTS IN NATIVE DATASET

PDB name	number res	number rot	opt energy	Min error	Avg error	Incorrect rot
1c9o	53	1130	-697.75	0.00	0.90	0
1aac	85	1523	-765.47	0.00	0.00	0
1czp	83	1170	-434.19	0.00	2.41	0
1qu9	100	1817	-1336.14	0.00	0.00	0
5pti	46	1088	118.00	0.00	42.03	0
1b9o	112	2056	-1325.75	0.00	0.00	0
1ctj	61	1021	-920.44	0.00	0.00	0
1cex	146	2556	-1952.04	0.00	0.00	0
1mfm	118	2134	-1508.57	0.00	0.00	0
1eca	108	1885	-1526.37	0.00	0.47	0
1rcf	142	2396	-1246.81	0.12	1.26	2
1qtn	134	2516	-1661.19	0.00	0.03	0
7rsa	109	1958	-658.75	0.00	0.04	0
1c5e	71	1108	-930.64	0.00	0.02	0
1cz9	111	2332	-1209.86	0.00	0.58	0
5p21	144	2874	-1243.04	1.16	1.80	10
1aho	54	981	-374.49	0.00	0.11	0
1plc	82	1156	132.73	0.00	0.00	0
1cku	60	1093	198.50	0.00	0.00	0
1vfy	63	939	-712.38	0.00	1.38	0
1igd	50	926	-501.61	0.00	0.59	0
1qj4	221	4080	-3023.47	0.56	1.50	7
2pth	151	3077	-2165.46	0.05	0.32	2
1cc7	66	1396	-621.82	0.00	0.70	0
1d4t	89	1636	-1344.47	0.00	0.09	0
1qq4	143	2045	-1346.16	0.17	12.42	3
3lzt	105	2074	-1301.11	0.00	0.23	0

TABLE III
RESULTS IN HOMOLOGY DATASET

PDB name	number res	number rot	opt energy	Min error	Avg error	Inc. rot
1cku-1eyt	61	1095	216.05	0.00	0.00	0
1ctj-1f1f	64	1219	-835.75	0.00	0.00	0
1aac-1id2	86	1608	819.88	0.00	0.02	0
1czp-4fxc	81	961	-389.23	0.00	2.79	0
1cc7-1fe4	62	1222	281.09	0.00	4.42	0
1plc-1byo	79	1131	279.10	0.00	4.75	0
1czp-1doy	81	990	-380.20	0.00	1.92	0
1aho-1dq7	53	719	-173.58	0.00	0.00	0
1cku-3hip	65	1079	248.45	0.00	0.06	0
1mfm-1b4l	117	1978	2966.20	0.00	0.72	0
1c9o-1mjc	52	862	-654.63	0.00	0.00	0
1mfm-1xso	114	1826	-1102.35	0.00	0.24	0
1aac-2b3i	87	1242	3956.09	1.12	5.08	5
1cz9-1c6v	113	1979	172.89	0.00	15.57	0
1mfm-1cob	119	1980	-1270.05	0.00	0.00	0
1qj4-1e89	220	4154	-2886.80	0.37	1.38	6
1c9o-1csp	53	1076	-676.87	1.14	1.82	2
1ctj-1cyj	66	1291	-972.90	0.00	0.00	0
1igd-1mi0	49	723	-529.03	0.00	0.37	0
1qq4-1hpg	139	1514	4041.36	0.76	2.43	4
1c9o-1g6p	54	1409	-682.24	0.00	0.01	0

The native dataset contains 27 proteins that vary in size from 46 to 221 amino acid residues. Each amino acid residue is allowed to assume all the rotamers listed in the library. This produces search spaces with up to 10^{218} possibilities. From Table II it can be seen that among the 27 proteins, CCPSO is able to find the optimal conformation in 22 instances. The relative error in all four cases where the solution is not the rotamer conformation with the optimal energy value was fairly small, with an $error < 1.16\%$ in the worse case.

For the task of side-chain packing in homology modeling, 21 homologs to the proteins of the native dataset are selected. For each pair, a template/target protein is defined, where the template protein provides the backbone and the target protein is the protein for which the structure is to be predicted. CCPSO obtains the provably optimal solution in all but four cases. As in the native dataset, the relative error is very small, with $error < 1.12\%$. Details are shown in Table III.

TABLE IV
RESULTS IN DESIGN DATASET

PDB name	number res	number rot	opt energy	Min error	Avg error	Incorrect rot
1igd	11	552	-298.07	0.00	0.00	0
1aac	38	2153	-860.61	0.00	0.77	0
1qu9	43	2057	-1093.26	0.00	1.14	0
7rsa	46	1993	-764.04	2.22	6.28	5
1c5e	25	1369	-587.20	0.00	1.17	0
1b9o	48	1842	-494.88	1.93	5.84	2
1ctj	24	1262	-670.87	0.00	0.00	0
1cz9	53	2664	-1274.13	0.49	2.51	4
1plc	33	1691	157.34	0.00	7.60	0
1vfy	15	665	-365.03	0.00	0.00	0
1mfm	46	3215	-1095.26	0.25	0.69	3
1c9o	14	757	-380.96	0.00	0.32	0
1czp	30	1475	-703.26	0.63	1.97	2
1cex	78	3926	-1815.78	1.24	2.26	11
1rcf	65	3189	-1508.24	1.60	3.72	8
1qtn	49	2181	-1176.97	1.90	2.85	6
5p21	70	3624	-1616.96	3.64	5.35	16
1aho	18	668	-138.88	0.00	0.00	0
1cku	22	897	-574.65	0.00	0.10	0
1qj4	124	6655	-2569.98	2.77	4.07	29
2pth	76	4395	-1856.56	4.36	5.70	19
1cc7	18	866	-410.44	0.27	0.79	2
1d4t	32	1691	-914.55	1.41	2.40	8
1qq4	72	3500	-1190.81	0.04	1.49	2
3lzt	48	1940	-953.50	0.72	1.44	4

The design dataset contains 25 instances. Side-chain packing in a protein design task varies significantly from the two previous tasks studied. In this case, the amino acid as well as the residue needs to be chosen, which makes the search space significantly larger. As in [30], the amino acids are grouped into the following classes: AVILMF / HKR / DE / TQNS / WY / P / C / G. For each of the 25 proteins in the native test set, we allow each amino acid position in the backbone to be replaced for any other amino acid that belongs to the same class as the native residue. Therefore, the number of rotamers that need to be consider for each residue grows substantially. The sizes of the resulting problems are shown in column three of Table IV. Among the 25 design problems, the CCPSO is able to find the optimal configuration for 10 instances. Not surprisingly, the design problem is more difficult to solve for the CCPSO than fitting side chains on native and homologous backbones. Nevertheless, the CCPSO manages to obtain solutions that are close to the optimal, with the relative $error < 4.36$ even in the worst case.

To analyze the effect of the number of residues and rotamers on the performance of CCPSO we compare the running times of the native and design dataset. The native dataset proteins varies from 46 to 221 residues, with a maximum of 4080 rotamers. The running time for the full dataset was 128 minutes, with the longest time taken by 1qj4 which takes 74 minutes. Proteins in the design dataset have fewer residues but considerably more rotamers. The number of residues ranges from 11 to 124 with a maximum of 6655 rotamers. The running time for the full design dataset is less than 40 minutes, where over one third of the time was used by protein 1qj4. Running times vary from 8 seconds for 1aho to 15 minutes for 1qj4. CCPSO seems to be susceptible to residue size, therefore the native dataset takes more time. Unlike other traditional SCP methods, CCPSO does not seem to be as susceptible to the increase in rotamer size characteristic of design problems. This makes CCPSO particularly appealing for this protein prediction task.

VIII. Conclusions

We present a novel cooperative combinatorial PSO that preserves the concept of particles that move throughout a continuous high-dimensional space while keeping its own cognitive and social memories of candidate solutions. A new cooperative strategy partitions the search space by splitting the problem into components and using one particle to optimize each component. This component-by-component optimization allows fine tuning of each component by each particle when the candidate solution templates do not change from one iteration to the next and explores a different part of the space when the memories are modified.

We examine its application to a NP-hard problem, Side-chain Packing where we obtain results that are close to the provably combinatorial optimal solution. Results in the protein design task are particularly encouraging since CCPSO's performance seems not to be strongly affected by an increase in the number of rotamers per residue. Preliminary experiments showed that simple schemes such as casting the problem as a continuous optimization problem and rounding the result, as well as Discrete PSO, do not find as good solutions.

We expect our CCPSO approach may be a useful approach to solving other difficult combinatorial problems within the PSO framework and future work includes comparison in more traditional combinatorial tasks.

References

[1] B. Al-Lazikani, J. Jung, Z. Xiang, and B. Honig, "Protein structure prediction," *Current Opinion in Chemical Biol.*, 5(1), pp. 51–56, 2001.

[2] S. Das, A. Abraham, and A. Konar, "Swarm intelligence algorithms in bioinformatics," in *Comp. Intelligence in Bioinf.*, pp. 113–147, 2008.

[3] A. Shmygelska and H. Hoos, "An Improved Ant Colony Optimisation Algorithm for the 2D HP Protein Folding Problem," in *Advances in Artificial Intelligence*, p. 993, 2003.

[4] D. Chu, M. Till, and A. Zomaya, "Parallel Ant Colony Optimization for 3D Protein Structure Prediction using the HP Lattice Model," in *Proc. 19th IEEE Int. Parallel and Distributed Processing Symposium*, p. 193b, 2005.

[5] O. Korb, T. Stützle, and T. Exner, "An ant colony optimization approach to flexible protein ligand docking," *Swarm Intelligence*, 1(2), pp. 115–134, 2007.

[6] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conference on Neural Networks*, pp. 1942–1948, 1995.

[7] R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm Intelligence*, Morgan Kaufmann, 2001.

[8] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, 1(1), pp. 33–57, 2007.

[9] M. Clerc, "Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem," in *New Optimization Techniques in Engineering*, 141, pp. 219–239, Springer, 2004.

[10] K. P. Wang, L. Huang, C. G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proc. Int. Conference on Machine Learning and Cybernetics*, 3, pp. 1583–1585, 2003.

[11] M. F. Tasgetiren, M. Sevkli, Y.-C. Liang, and G. Gencyilmaz, "Particle swarm optimization algorithm for single machine total weighted tardiness problem," in *Proc. Congress on Evolutionary Computation, CEC '2004*, 2, pp. 1412–1419, 2004.

[12] M. Lovbjerg, T. Rasmussen, and T. Krink, "Hybrid Particle Swarm Optimizer with Breeding and Subpopulation," in *Proc. Genetic and Evolutionary Computation Conference, GECCO'2001*, 2001.

[13] S. Baskar and P. N. Suganthan, "A novel concurrent particle swarm optimization," in *Proc. Congress on Evolutionary Computation, CEC'2004*, 1, pp. 792–796, 2004.

[14] B. Al-Kazemi and C. K. Mohan, "Discrete multi-phase particle swarm optimization," in *Information Processing with Evolutionary Algorithms*, pp. 305–327, 2005.

[15] M. Potter and K. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature PPSN III*, pp. 249–257, 1994.

[16] N. Keerativuttitumrong, N. Chaiyaratana, and V. Varavithya, "Multi-objective co-operative co-evolutionary genetic algorithm," in *Proc. PPSN VII*, Springer-Verlag, pp. 288–297, 2002.

[17] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, 8(3), pp. 225–239, 2004.

[18] C. H. Tan, C. K. Goh, K. C. Tan, and A. Tay, "A cooperative coevolutionary algorithm for multiobjective particle swarm optimization," in *Proc. of IEEE Congress on Evolutionary Computation, CEC 2007*, pp. 3180–3186, 2007.

[19] G. C. Lapizco-Encinas and J. A. Reggia, "Diagnostic problem solving using swarm intelligence," in *Proc. IEEE Swarm Intelligence Symposium, SIS '05*, pp. 365–372, 2005.

[20] N. Pokala and T. M. Handel, "Review: Protein design–where we were, where we are, where we're going," *Journal of Structural Biology*, 134(2-3), pp. 269–281, 2001.

[21] J. Ponder and F. Richards, "Tertiary templates for proteins: use of packing criteria in the enumeration of allowed sequences for diff. structural classes,"*J. of Molecular Biology*, 193(4), pp. 775–791, 1987.

[22] N. A. Pierce and E. Winfree, "Protein design is NP-hard," *Protein Eng.*, 15(10), pp. 779–782, 2002.

[23] B. Chazelle, C. Kingsford, and M. Singh, "A semidefinite programming approach to side chain positioning with new rounding strategies," *INFORMS Journal on Computing*, 16(4), pp. 380–392, 2004.

[24] B. I. Dahiyat and S. L. Mayo, "De novo protein design: fully automated sequence selection," *Science*, 278(5335), pp. 82–87, 1997.

[25] J. R. Desjarlais and T. M. Handel, "De novo design of the hydrophobic cores of proteins," *Protein Science*, 4(10), pp. 2006–2018, 1995.

[26] J. Desmet, M. D. Maeyer, B. Hazes, and I. Lasters, "The dead-end elimination theorem and its use in protein side-chain positioning," *Nature*, 356(6369), pp. 539–542, 1992.

[27] W. Xie and N. V. Sahinidis, "Residue-rotamer-reduction algorithm for the protein side-chain conformation problem," *Bioinformatics*, 22(2), pp. 188–194, 2006.

[28] M. De Maeyer, J. Desmet, and I. Lasters, "The dead-end elimination theorem: math. aspects, implementation, optimizations, evaluation, and performance," *Methods in molecular biology*, 143, pp. 265–304, 2000.

[29] E. Althaus, O. Kohlbacher, H. P. Lenhof, and P. Müller, "A combinatorial approach to protein docking with flexible side chains," *Journal of Computational Biology*, 9(4), pp. 597–612, 2002.

[30] C. Kingsford, B. Chazelle, and M. Singh, "Solving and analyzing side-chain positioning problems using linear and integer programming," *Bioinformatics*, 21(7), pp. 1028–1039, 2005.

[31] A. A. Canutescu, A. A. Shelenkov, and R. L. Dunbrack, "A graph-theory algorithm for rapid protein side-chain prediction," *Protein Science*, 12(9), pp. 2001–2014, 2003.

[32] J. Xu, "Rapid protein side-chain packing via tree decomposition," *Research in Computational Molecular Biology*, pp. 423–439, 2005.

[33] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conference on Systems, Man, and Cybernetics*, 5, pp. 4104–4108, 1997.

[34] E. C. Laskari, K. E. Parsopoulos, and M. N. Vrahatis, "Particle swarm optimization for integer programming," in *Proc. IEEE Congress on Evolutionary Computation, CEC'02*, pp. 1582–1587, 2002.

[35] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems*, 26(8), pp. 363–371, 2002.

[36] J. Pugh and A. Martinoli, "Discrete Multi-Valued Particle Swarm Optimization," in *Proc. IEEE Swarm Intelligence Symposium, SIS '06*, pp. 103–110, 2006.

[37] B. Secrest and G. Lamont, "Communication in particle swarm optimization illustrated by the traveling salesman problem," in *Proc. Workshop on Particle Swarm Optimization*, 2001.

[38] R. L. Dunbrack and M. Karplus, "Backbone-dependent rotamer library for proteins application to side-chain prediction," *Journal of Molecular Biology*, 230(2), pp. 543–574, 1993.

[39] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, et al., "The amber biomolecular simulation programs," *J Comput Chem*, 26(16), pp. 1668–1688, 2005.