ABSTRACT

Title of dissertation: Cooperative Particle Swarm Optimization for Combinatorial Problems

Grecia Lapizco-Encinas, Doctor of Philosophy, 2009

Dissertation directed by: Professor James Reggia
Professor Carl Kingsford
Department of Computer Science

A particularly successful line of research for numerical optimization is the well-known computational paradigm *particle swarm optimization* (PSO). In the PSO framework, candidate solutions are represented as particles that have a position and a velocity in a multidimensional search space. The direct representation of a candidate solution as a point that flies through hyperspace (i.e., $\mathbb{R}^n$) seems to strongly predispose the PSO toward continuous optimization. However, while some attempts have been made towards developing PSO algorithms for combinatorial problems, these techniques usually encode candidate solutions as permutations instead of points in search space and rely on additional local search algorithms.

In this dissertation, I present extensions to PSO that by, incorporating a cooperative strategy, allow the PSO to solve combinatorial problems. The central hypothesis is that by allowing a set of particles, rather than one single particle, to represent a candidate solution, combinatorial problems can be solved by collectively constructing solutions. The cooperative strategy partitions the problem into components where each component is

optimized by an individual particle. Particles move in continuous space and communicate through a feedback mechanism. This feedback mechanism guides them in the assessment of their individual contribution to the overall solution.

Three new PSO-based algorithms are proposed. *Shared-space CCPSO* and *multi-space CCPSO* provide two new cooperative strategies to split the combinatorial problem, and both models are tested on proven NP-hard problems. *Multimodal CCPSO* extends these combinatorial PSO algorithms to efficiently sample the search space in problems with multiple global optima. Shared-space CCPSO was evaluated on an abductive problem-solving task: the construction of parsimonious set of independent hypothesis in diagnostic problems with direct causal links between disorders and manifestations. Multi-space CCPSO was used to solve a protein structure prediction subproblem, side-chain packing. Both models are evaluated against the provable optimal solutions and results show that both proposed PSO algorithms are able to find optimal or near-optimal solutions. The exploratory ability of multimodal CCPSO is assessed by evaluating both the quality and diversity of the solutions obtained in a protein sequence design problem, a highly multimodal problem. These results provide evidence that extended PSO algorithms are capable of dealing with combinatorial problems without having to hybridize the PSO with other local search techniques or sacrifice the concept of particles moving throughout a continuous search space.

Cooperative Particle Swarm Optimization for Combinatorial Problems

by

Grecia C. Lapizco-Encinas

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2009

Advisory Committee:
Professor James Reggia, Chair/Advisor
Professor Carl Kingsford, Co-Advisor
Professor Dana Nau
Professor Mihai Pop
Professor Lindley Darden

Dedication

I dedicate this dissertation to my family, especially...

To my mom, whose unwavering faith and words of encouragement kept me going. This dissertation would not exist without her constant support.

To my sister, a role-model for hard work, persistence and personal sacrifice in the pursuit of academic excellence.

To my husband, for the unexpected happiness that he brought into my life. I look forward to share our next journey together.

Acknowledgments

I would like to express my deepest gratitude towards my advisor, Jim Reggia, whose guidance and patience throughout the entire process were instrumental. Jim's dedication to his students is truly exceptional.

A special thanks goes to my co-advisor, Carl Kingsford, who is most responsible for helping me complete this dissertation as well as the research that lies behind it. I learned a lot from his dedication to research, invaluable comments and high standards. I will never forget how every paper "has to tell an interesting story".

I also like to acknowledge the members of my advisory committee Mihai Pop, Lindley Darden and Dana Nau for their valuable input and insightful comments of this dissertation.

I was lucky enough to work with two research groups, my gratitude goes to all my fellow graduate students in the *Biologically-Inspired Computing* and the *Networks Group*. The work presented here benefited vastly from their comments. Working at CBCB has been an inspiring experience, its not only a place for great research but it also has a highly collaborative and friendly research environment.

Last but not least, I would like to thank those people whose friendship and support made this experience a lot more positive. Thanks for the CSWomen potlucks, the tea-breaks and the movie and game nights. I'm specially thankful for all my lunch buddies, running partners, and the "Thursday's at Franklin's" crowd.

Thank you all!

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACO	Ant Colony Optimization
CCPSO	Cooperative Combinatorial Particle Swarm Optimization
DEE	Dead End Elimination
EC	Evolutionary Computation
GA	Genetic Algorithm
GMEC	Global Minimum Energy Conformation
IIA	Iterative Improvement Algorithm
PSO	Particle Swarm Optimization
SA	Simulated Annealing
SCP	Side-chain Packing
SI	Swarm Intelligence

Chapter 1

Introduction

Swarm intelligence (SI) is a biologically-inspired artificial intelligence technique based upon the study of collective behavior in decentralized, self-organized systems [16][43]. Inspiration has come, for example, from collectively moving animals (birds flocks, fish schools, herds of animals, etc.) and social insect swarms (ants, termites, etc.). This field encompasses a great variety of systems, but all approaches share some fundamental characteristics. The main idea in swarm intelligence systems consists of having simple agents with no centralized control structure dictating how the individual agents should behave, and local interactions between the agents often lead to the emergence of global behavior. These systems can be classified in three broad classes according to the structure of the space in which the agents interact: discrete cellular space where agents move in lattices of cells following some neighborhood rule [160][161], network-based space where agents move in discrete graph structures guided by a measure of "goodness" of each edge [39], and continuous space where agents' movements are based on direct interactions with one another in real-valued space [73][133]. This classification is shown in Figure 1.1.

The research described here is specifically concerned with swarm intelligence systems in continuous spaces, known as *self-organizing particle systems*. Self-organizing particle systems typically consist of numerous autonomous simple agents whose move-

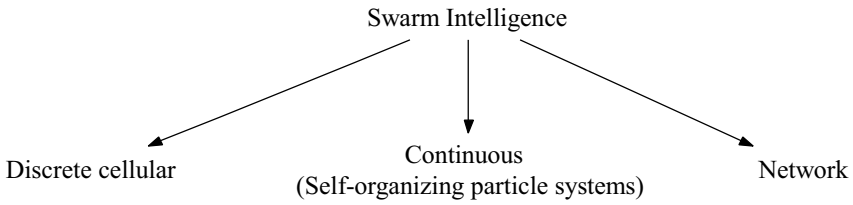


Figure 1.1: Classes of swarm intelligent systems. Each class is characterized by the space through which particles move.

ments through a continuous space are governed by various local "forces" exerted on them by other nearby agents or the environment. A majority of applications of this class of swarm intelligence involves modeling group behaviors in a 2D or 3D physical space. For example, these methods have been used in the simulation of group movements by animal populations in computer animation [133][150][151], and in swarm robotics, where the focus is on applying swarm intelligence techniques to control large groups of cooperating autonomous robots or vehicles [3][68][82][94][108][146]. Inspired by successes in these applications involving low dimensional physical space, there have been a much smaller number of efforts to generalize these methods to higher-dimensional abstract spaces.

Past studies illustrate the power of swarm intelligence, and show that it is a potentially valuable methodology for tackling complex problems [16][43]. In particular, self-organizing particle systems have proven to be interesting approaches in areas such as those mentioned above, but it remains to be established how general they are. Some attempts have been made to extend the paradigm toward problem solving tasks, for example using multi-robot teams for pushing objects [82], foraging [67], Robocup competitions [152], and more recently, in simulations of search-and-collect tasks [134][135].

An example of a particularly successful research direction in high-dimensional

swarm intelligence is *particle swarm optimization*, whose main focus is solving continuous optimization problems [43][73]. Examples of successful applications include learning weights for neural networks, tracking dynamic systems, and solving multiobjective optimization and constraint optimization problems [43]. Particle Swarm Optimization (PSO) was originally designed as a numerical optimization technique based on swarm intelligence [43][73], and it has shown its robustness and efficacy for solving function-value optimization problems in real-number spaces [43][122]. 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 direct representation of candidate solutions as points in a continuous virtual search space makes the PSO algorithm inherently geared toward continuous optimization problems. If we consider the heart of the field of competence of the traditional PSO, i.e., roughly continuous and mixed continuous-discrete (non-combinatorial problems), it is remarkably effective [24]. While the exploratory capacity of PSO remains interesting for combinatorial problems, in its basic form it cannot be used for more than the approximate detection of promising regions of the search space, which can be then passed to another algorithm to find an accurate solution [24]. Only a few previous attempts have been made to extend the PSO to fully solve combinatorial optimization problems [122], although some important exceptions do exist. One of the main reasons for this limitation is that when a combinatorial problem is difficult to solve, it is usually because the evaluation

function is very discontinuous. The PSO algorithm makes an implicit assumption that the closer two positions are, the closer their evaluations are too. For a problem such as the Traveling Salesman Problem, that is easily false: a simple transposition of two cities in the circuit, i.e., the shortest possible displacement in the search space, can change the evaluation (the cost of the path) from its minimum to its maximum [24]. Moreover, in combinatorial problems the relationship between the different elements of the solution is non-linear, allowing the possibility that some elements in the vector move closer to the optimal solution, while others actually move 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.

The main issue when adapting PSO to any non-continuous problem is how to define the relationship between a particle (and the operators used to transform it) and a candidate solution of the new optimization problem. There have been several proposed extensions of PSO to combinatorial spaces to try to deal with this representation issue with various degrees of success [23][149][157], but these approaches encode the solution as a permutation and design specialized velocity operators for this permutation representation, sacrificing the basic nature of the PSO of particles moving in a continuous high-dimensional space.

In this dissertation, I present *cooperative combinatorial PSO* (CCPSO), an extension to the particle swarm optimization algorithm aimed at tackling combinatorial problems beyond the mere approximate detection of promising regions in the search space.

CCPSO's goal is to obtain results that are close to the provably optimal solution. This is achieved by introducing a cooperative strategy that splits the problem into components, which are then optimized by individual particles. This extension redefines the relationship between particles and candidate solutions, while preserving the notion of particles moving in a continuous space. The central hypothesis is that by allowing a set of particles, rather than a single particle as in most particle swarms, to represent solutions to problems, one can generalize particle swarms to construct solutions to combinatorial problems. Cooperative strategies that represent solutions with multiples particles have been devised before to improve the performance of the PSO, particularly in high-dimensional problems; however, they have not been applied to combinatorial problems.

The goal of CCPSO is to provide a mechanism with which each particle optimizes its own choices, while being guided implicitly toward parts of the space that are more promising to the rest of the particles of the swarm. In CCPSO, unlike other PSO algorithms, 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 task of attractors is twofold. First, attractors pull moving particles towards areas of the space that are favorable for the specific component they optimize. Second, attractors mediate the communication between particles and guide individual particles to move toward areas which are promising to the whole swarm. The candidate solution is then constructed cooperatively by decoding each particle's position into a contribution for the solution. Two novel cooperative strategies are presented. The first strategy, *shared-space CCPSO*, consists of simple particles that move

in a *shared high-dimensional continuous space*, and based only on local interactions with their attractors, generate a solution as a result of their collective behavior. All particles share the same attractors, and feedback to the attractors is provided by a strength value which reflects the attractor's success in pulling particles towards its position, i.e., attractors are rewarded by their popularity among particles. The second strategy, *multi-space CCPSO*, consists of subwarms of particles, where each particle represents a component of a solution and moves in its *own high-dimensional continuous space*. The candidate solution is then constructed cooperatively by decoding each particle's position into a choice for that solution component. Each particle has its own set of attractors and feedback to the attractors is provided by a strength value that takes into account the fitness of the candidate solution with the current component replaced by the choice represented by the attractor. Multimodal CCPSO extends these combinatorial PSO algorithms to efficiently sample the search space in problems with multiple global optima. This new algorithm extends the ability of multi-space CCPSO to discover sets of high-quality and diverse solutions. Every time multimodal CCPSO detects a solution of high quality (i.e., a potential optimum) it "marks" the region in the search space by storing this solution in an external memory. The external memory is then used by diversity strategies to encourage particles to search different regions of the search space.

The proposed methods are studied and validated through three combinatorial optimization problems of increasing complexity. The first application is abductive diagnosis, in which the goal is to obtain the best or most plausible explanation for the manifestations (symptoms) known to be present in a given case. Such problems are often quite complex

due to the number of possible elementary hypotheses for each observation and the many different ways to combine these hypotheses into an explanation. Abductive diagnosis is an especially attractive task to be tackled because the plausibility of solutions depends on the interactions between explanation components in a highly non-linear fashion which makes it a complex problem.

The second problem is a particularly important subproblem of the general protein structure prediction problem: side-chain packing (SCP), in which the side chains of the protein residues are positioned on a fixed and given protein backbone. Specifically, it consists of selecting a side-chain conformation (from a discrete set of preferred conformations, known as rotamers [42]) for each side chain in the protein such that the protein energy conformation is minimized. The reasons for the selection of this particular combinatorial problem are twofold. First, like abductive diagnosis, SCP deals with a highly non linear and complex evaluation function. Second, the specific values of each solution component (i.e., the set of possible rotamers for each side-chain position) is distinct, and the dimensionality from one component to the next can vary significantly. This is the key difference with the type of combinatorial problems studied under the previous model. Additionally, 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 [28]. This makes algorithms like PSO, which are known to efficiently produce near-optimal solutions, especially attractive.

The third and final problem we consider is computational protein design by flexible side-chain packing [27][139]. In this bioinformatics task the goal is to find the sequence of amino acids for a given protein 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 [121]. This problem has a distinct goal from the previous problems explored. Here, the goal is not to obtain the optimal solution to an optimization problem, but to find an ensemble of low-energy solutions that maximizes the pairwise diversity of this ensemble. This provides several candidates sequences that can then reranked by more physically realistic (and computationally more expensive) energy functions. Several approaches for forcing the PSO to return a diverse ensemble of solutions are analyzed and compared.

One advantage of PSO over many other optimization methods is its relative simplicity. Another, perhaps more important, advantage over other global minimization strategies such as simulated annealing is that the interactions between the large number of members that make up the particle swarm make the technique more resilient to the problem of local minima [43]. The research presented here aims to further this advantage by extending these interactions from sharing solutions between the individuals of the swarm to collectively constructing solutions.

In summary, the research described here builds on the methods used in traditional PSO algorithms, extending this technology. A significant innovation from the existing PSO methods is the concept of collective construction of a solution as a strategy for tackling combinatorial problems; unlike traditional particle swarms algorithms, where

each particle represents a solution, here a solution is represented by the collection of particles. Thus, each particle does not have an associated concept of "goodness" with respect to some objective function as has usually been done in the past. This raises the question of how the individual particle, based solely on local information, influences the problem-solving of the collective agent team as a whole; and how a particle, with no global vision of the solution of the problem, can contribute to the solution and adapt its contribution according to its limited interactions with neighboring particles.

1.1 Organization

The rest of this dissertation is organized as follows. In Chapter 2, a brief introduction to combinatorial problems is presented, including a brief description of metaheuristics similar to PSO. This is followed by an overview of swarm intelligence, focusing on the main concepts employed in particle-systems. The last part of the chapter introduces the traditional PSO and surveys relevant combinatorial and cooperative extensions. Chapter 3 presents the first extension of PSO to combinatorial problems, detailing the proposed cooperative strategy that partitions the combinatorial problem into components that are optimized independently. It then describes how this algorithm is applied to diagnosis problem solving, along with experimental results of its performance and comparison to the provable optimal solution. Chapter 4 introduces multi-space CCPSO, an extension of PSO to combinatorial problems where each solution component has its own set of independent values, and its application to the problem of side-chain packing in protein structure prediction. Chapter 5 presents the study of diversity strategies for the multi-space

CCPSO algorithm and shows its performance in a combinatorial problem with multiple optima: computational protein design. Finally, Chapter 6 presents a summary of the findings and contributions of this dissertation research, and considers possible future research that may emanate from it.

Chapter 2

Background

This chapter presents a brief introduction to combinatorial problems and swarm intelligence methodology to better understand the framework in which the new combinatorial PSO fits in. The chapter starts with an introduction to combinatorial optimization problems, including a concise description of related metaheuristics. This is followed by an overview of swarm intelligence, focusing on self-organizing particle systems and pointing out some of its most illustrative applications. The last part of this chapter surveys relevant past extensions of PSO algorithms proposed in the literature.

2.1 Combinatorial Optimization and Metaheuristics

Combinatorial problems are intriguing because they are easy to state but often very difficult to solve [148]. Combinatorial problems involve finding values for discrete variables such that certain conditions are satisfied. They can be classified either as optimization or satisfaction problems. Satisfaction problems aim at finding a valid solution, regardless of any quality criterion. Optimization problems, on the other hand, have the goal to find an optimal arrangement, grouping, ordering, or selection of discrete objects that optimizes a quality criterion and fulfills the given constraints. Prominent examples of combinatorial problems are tasks such as finding the shortest or cheapest round trips in graphs, planning, scheduling, time-tabling, resource allocation, protein structure predic-

tion and internet data packet routing, among others. Formally, an instance of a combinatorial optimization problem is a pair (S, f), where S is the finite set of candidate solutions and $f : S \rightarrow \mathbb{R}^n$ is a function which assigns to every $s \in S$ a real value $f(s)$. A combinatorial optimization problem is either a maximization problem or a minimization problem with an associated set of instances. The goal is to find a globally optimal solution s^*. For minimization problems, this consists in finding a solution s^* with minimum cost, that is, a solution such that $f(s^*) \leq f(s) \; \forall \; s \; \in \; S$. Similarly, a maximization problem consists in finding a candidate solution s^* such that $f(s^*) \geq f(s) \; \forall \; s \; \in \; S$. Combinations of *solution components* form the potential solutions of a combinatorial problem. A scheduling problem, for instance, can be seen as an assignment problem in which the solution components are the events to be scheduled, and the values assigned to events correspond to the time at which they occur. Typically, a huge number of candidate solutions can be obtained this way. For most combinatorial optimization problems, the space of potential solutions for a given problem instance is at least exponential in the size of the instance [64].

2.1.1 Algorithms for Combinatorial Optimization

Optimization techniques for combinatorial problems can be classified into exact, approximate, and heuristic. Exact techniques guarantee finding the optimal solution in every single instance of the problem within an instance-dependent bounded time [13][14]. For finite size problems, a straightforward exact algorithm is to simply enumerate the full solution space. This approach is generally undesirable as well as intractable due to the exponential growth of most solution spaces. To increase efficiency, all modern exact

methods use pruning rules to discard parts of the search space in which the optimal solution cannot be found. These approaches are doing an implicit enumeration of the search space. Among the exact methods are branch-and-bound (B&B) [86][99], dynamic programming [9][60], and linear and integer programming based methods [58][105]. One important drawback of exact algorithms is that they suffer from a strong increase in computational time when the problem size grows, effectively rendering this type of algorithm infeasible for large-sized problems.

When an optimal solution cannot be efficiently obtained in practice, one possibility is to sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions using approximate methods. Approximate methods can be classified in approximate algorithms and heuristics algorithms. While approximate algorithms do not guarantee optimal solution, they guarantee a good solution within some factor of the optimum. Specifically, an ϵ-approximation algorithm is a polynomial-time algorithm which always produces a solution of value within ϵ times the value of an optimal solution [4]. Heuristics methods, on the other hand, usually find reasonable good solutions in a reasonable time but do not have an approximation guarantee on the obtained solutions.

A significant amount of research has been devoted to the design of general heuristic methods which are applicable to a wide range of different combinatorial optimization problems. For these general-purpose methods the term *metaheuristics* has been coined [84]. The goal of metaheuristics is to guide the search algorithm towards promising regions of the search space containing high-quality solutions [14]. Generally, metaheuristics contain mechanisms to avoid getting trapped on local optima. Many of the

methods use information gathered during the search process to promote finding high-quality solutions quickly. This information can be based on changes of the values of the objective function, on previously made decisions, or on prior performance. Additionally, many of the metaheuristic approaches rely on probabilistic decisions made during the search. Basically, the main difference with pure random search is that, in metaheuristic algorithms, randomness is not used blindly but in an intelligent way in order to find efficiently near-optimal solutions [14].

Metaheuristics can be classified into trajectory methods and population-based methods. Trajectory methods are algorithms that work on single solutions. They encompass local search-based metaheuristics, like simulated annealing [76][77][83], tabu search [54][52][53], iterated local search [1][64] and variable neighborhood search [59][100]. They all share the property of describing a single trajectory in the search space during the search process. Population-based metaheuristics, on the other hand, perform search processes that describe the evolution of a set of points in the search space. The most studied population-based methods in combinatorial optimization are evolutionary computation [57][80][98] and ant colony optimization [37][38][39].

The next subsections describe three relevant metaheuristics, local search, simulated annealing and evolutionary computation. Local search encompasses a variety of specific algorithms, from the most basic one, iterative improvement algorithm, to very complex strategies. Simulated annealing is one of the most popular metaheuristics. Evolutionary computation, like PSO, is a nature-inspired population-based algorithm.

Algorithm 1 Pseudocode of the Simple Iterative Improvement Algorithm

```
Initialize candidate solution s
while s is not local optimum do
   Randomly choose a neighbor s' of s
   if f(s') > f(s) then
      s ← s'
   end if
end while
return s
```

2.1.2 Local Search Algorithms

One of the most popular strategies for approximate algorithms are local search algorithms. Local search algorithms start from some given solution and try to find a better solution in an appropriately defined neighborhood of the current solution. In case a better solution is found it replaces the current solution and the local search is continued from there.

One of most basic local search metaheuristics is the iterative improvement algorithm (IIA) [66][92]. The IIA algorithm only moves towards solution points that improve the best-found solution quality. IIA stops whenever a local optimal point is reached. The high-level algorithm is sketched in Algorithm 1. The search starts with an initial solution, then moves iteratively into a nearby landscape region. Since the next candidate solution is always selected within the current neighborhood set, the search is always based on local information. This use of local information inevitably leads to one problem: the algorithm is easily trapped in local optima. Therefore, the performance of iterative improvement procedures on combinatorial optimization problems is usually quite unsatisfactory.

Several techniques have been developed to prevent algorithms from getting trapped in local minima, by adding mechanisms that allow them to escape from these minima.

One of these mechanisms is to increase the size of the neighborhood used in the local search algorithm. This increases the chance of finding an improved solution, but it also takes a longer time to evaluate the neighboring solutions, making this approach infeasible for larger neighborhoods. Another simple approach is to restart the algorithm from a new, randomly generated solution. This simple mechanism typically works in scenarios with a small number of local optima, but since a typical search space contains a large number of local optima, this approach becomes increasingly inefficient on large instances. Another alternative is to use constructive algorithms to generate good initial solutions to seed the local search algorithm. For many problems this has been shown to be a promising approach to provide better solutions than when starting the local search from randomly generated solutions.

2.1.3 Simulated Annealing

Simulated Annealing (SA) is an early metaheuristic algorithm that has been quite successful in combinatorial optimization problems (see [83] for details). SA is one of the first algorithms that had an explicit strategy to escape from local minima. Its name and inspiration come from annealing process in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects [76][77].

SA uses a temperature parameter as an explicit strategy to guide the search. The solution space is usually explored by taking random tries. An uphill move is always accepted since it is a better solution. Downhill moves are accepted conditionally with a

probability that depends on both the current temperature level as well as the change in solution quality. Usually, the change in solution quality is measured by the difference in the objective function value. If we denote the temperature value as T, and the change in solutions quality as δE, the probability of accepting a downhill move is generally computed following the Boltzmann distribution $p = e^{-\frac{\delta E}{k_B * T}}$, where k_B is the Boltzmann constant, that is used for adjusting the effect of temperature T in computing the downhill probability. Another important part of SA is the *cooling scheme*, which defines how T decreases during the search. The basic idea is to allow more downhill moves at the early stage of the search to keep the algorithm actively exploring the solution space (the higher T, the higher the probability to accept a downhill move) and as the search continues, downhill moves should become less likely so that the search would focus on a limited area that is identified as good solution region. Basically, this means that the algorithm is the result of the combination of two strategies: random walk and iterative improvement, and the cooling scheme defines how these strategies are combined. It has been claimed that given a cooling scheme that decreases the temperature slowly enough, SA is guaranteed to find the global optimal solution [83], although the scheme is usually too slow for practical use [61]. Therefore, problem-specific algorithm tuning is usually required in real applications. The SA pseudocode is presented in Algorithm 2.

2.1.4 Evolutionary Computation

Evolutionary Computation (EC) is, in general, the field encompassing computational techniques based on, or inspired by Darwinian evolution. Any attempt to describe

Algorithm 2 Pseudocode of the Simulated Annealing Algorithm

Initialize candidate solution $s \leftarrow s'$, set $s_{best} \leftarrow s$
Initialize cooling scheme, set temperature $T \leftarrow T_0$
while termination criterion not met **do**
  Randomly choose a neighbor s' of s
  if $f(s') > f(s)$ **then**
    $s \leftarrow s'$
  else
    $\delta E \leftarrow f(s) - f(s')$
    $s \leftarrow s'$ with probability $p = e^{-\frac{\delta E}{k_B * T}}$
  end if
  Update temperature T according to the cooling scheme
  if $f(s) > f(s_{best})$ **then**
    $s_{best} \leftarrow s$
  end if
end while
return s_{best}

the components of an EC technique that are common to all varieties will not do justice to all approaches. But, without being specific to a particular mechanism, any EC technique generally has the same abstract idea. First, it maintains a population of individuals (usually randomly generated initially) and every individual is assigned a fitness value. All individuals or candidate solutions are evaluated with respect to some performance criteria. Candidate solutions that are of higher quality/fitness are used for generating new solutions with higher probability than those that are of low quality, and some low quality solutions may be discarded. New candidate solutions are created by random variations and combinations of the existing candidate solutions, and those variations may propagate to future candidate solutions. This process leads to the evolution of populations of individuals that are better suited to their environment (specified by the objective function) than the individuals that they were created from, just as in natural adaptation. Although simplistic from a biologist's point of view, these algorithms are sufficiently complex to

provide robust and powerful adaptive search mechanisms [144]. A notable strength of evolutionary methods is that they frequently have an advantage over many traditional local search heuristic methods when search spaces are highly modal, discontinuous, or highly constrained [98].

Four well-known paradigms currently exist in evolutionary computation: genetic algorithms [57], evolutionary programming [46], evolution strategies [129][138], and genetic programming [80]. Most of these techniques are similar in spirit, but differ in the details of their implementation and the nature of the particular problem to which they have been applied [5][159]. In the case of combinatorial optimization problems, the best suited technique seems to be genetic algorithms.

Genetic algorithms (GA) are the among most popular type of evolutionary methods. Most GA represent the genotype of an individual as a fixed-length binary string, where usually the string length must be determined prior to optimization and reflects the dimensionality of the problem. The binary nature of the representation that is classically used in GA strikes a clear difference between genotype (the genetic makeup of an individual) and phenotype (the expression of those genes), as well as the encoding/decoding transformations needed to navigate between the two. The genetic operators *crossover* and *mutation* are executed over the genotype of the individual. With the binary representation, the cross-over operator consists of extracting bit strings from each parent in order to recreate two new bit strings. Cross-over occurs at a particular rate called the *cross-over probability*. The mutation operator is applied by randomly alternating bit values. Mutation occurs with a certain probability referred to as the *mutation rate*. A popular selection

Algorithm 3 Pseudocode of a Genetic Algorithm

```
Choose initial population randomly
Evaluate the fitness of each individual in the population
repeat
    Select best ranking individuals to reproduce
    Breed new generation through crossover and mutation (genetic operations) and give
    birth to offspring
    Evaluate the individual fitness of the offspring
    Select individuals for next generation
until termination condition is met
```

scheme in GA is a stochastic method that biases selection by preferring more fit individuals over less fit ones by a degree proportional to their respective fitness. The pseudocode for a GA algorithm is sketched in Algorithm 3.

2.2 Swarm Intelligence

In the following, I use the term *swarm* in a general manner to refer to any restrained collection of interacting agents or individuals [43]. The classical example of a swarm is bees swarming around their hive; nevertheless the metaphor can easily be extended to other systems with a similar architecture. An ant colony can be thought of as a swarm whose individual agents are ants. The same idea can be applied to many organisms ranging from colonies of simple bacteria, to flocks of birds, to herds of large animals. These animal societies have the interesting property that they seem to conduct themselves in a very organized way with seemingly purposeful behavior that enhances their collective survival. Surprisingly, however, the individual organisms often utilize very simple rules of interaction [16]. Such collective behavior has inspired algorithms and heuristics known as *swarm intelligence* which have practical applications in many different areas.

Unfortunately, the term "swarm intelligence" suffers from a vague definition, and there is no widely agreed upon taxonomy of the systems or algorithms belonging to this area. The term was first defined as a property of systems of unintelligent agents exhibiting collectively intelligent behavior [10]. This definition was used in the context of cellular robotics systems, where many simple agents occupy a one or two-dimensional environment to generate patterns and to self-organize through nearest-neighbor interactions [10]. The definition has been extended since then to include any attempt to design algorithms or problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies [16].

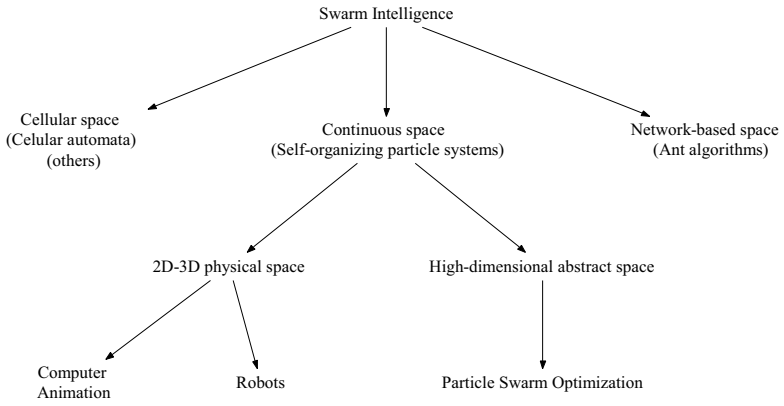


Figure 2.1: A classification scheme for swarm intelligence based on the structure of the underlying space.

A classification scheme for swarm intelligence according to the structure of the underlying space is shown in Figure 2.1. This scheme groups swarm intelligence systems into three broad classes: discrete cellular spaces, continuous spaces and network-based spaces. Cellular spaces, such as cellular automata, are regular lattices of cells, arranged in some geometric pattern and connected to other similar cells by some neighborhood rule. Each cell has a finite number of states which are determined by the states of its immediate

neighbors. This model, despite the simplicity of local rules, can show complex global behaviors like self-replication [50][155][160][161].

In network-based spaces, such as the ones used in *ant algorithms*, agents move over discrete graph structures guided by a measure of "goodness" of each edge [39]. Its most representative algorithm, *ant colony optimization* (ACO), is an agent-based metaheuristic motivated by the foraging strategies of real ants [39]. In ACO, each artificial ant is considered as a simple agent that communicates with other ants only indirectly by effecting changes to a common environment. The set of agents (a colony of ants) moves through states of the problem corresponding to partial solutions of the problem to solve. They move by applying a stochastic local decision policy. By moving, each ant incrementally constructs a solution to the problem (e.g., a route that solves a traveling salesperson problem). Ants evaluate the solution and modify the trail value on the components used in it (i.e., ants release pheromone) while building the solution (step by step) or once the solution is built. This pheromone information will direct the search of the future ants. Many difficult optimization problems have been solved by these algorithms such as the traveling salesman problem, the quadratic assignment problem, packet scheduling, vehicle routing, etc. [16][39].

Continuous spaces are generally handled by self-organizing particle systems, in which particles (agents) move guided by local "forces" exerted on them by other nearby agents or the environment [73][133]. Approaches where agents move in 2D or 3D physical space are used for modeling group behaviors. For example it is used in the simulation of group movements by animal populations in computer anima-

tions [133][150][151] and in swarm robotics, where the focus is on applying swarm intelligence techniques to the control of large groups of cooperating autonomous robots or vehicles [3][68][82][94][108][146]. Generalized methods in higher-dimensional abstract spaces focus on solving continuous optimization problems. As mentioned earlier, *particle swarm optimization* is an example of a particularly successful algorithm in this class.

Since self-organizing particle systems are within the area of swarm intelligence in which this research focuses, they will be discussed in more detail in the following sections.

2.2.1 Computer Animation and Swarm Robotics

Perhaps the simplest approach to swarm intelligence is the modeling of flocks, herds and schools that give rise to quite appealing spatial configurations especially suited for computer animation. Early work in this area includes particles systems developed by Reeves [130]. This model was extended by Reynolds in his seminal model of flocking of birds, where he studied in detail the problem of group trajectories without central control [133]. More recently, Terzopoulos developed a similar model for behavioral animation of fish based on the repertoire of behaviors that are dependent on their perception of the environment [150].

A particle system, as defined by Reeves, is a collection of a large number of simple individual particles, each having its own behavior, and originally used to model fire, smoke, and other "fuzzy" objects that have irregular and complex shapes [130]. Over a period of time, the system generates particles (setting their initial attributes such as posi-

tion, velocity, size, color, shape, lifetime), and these particles move in discrete steps in a 3D world (by adding their velocity vector and position vector), while changing the values of their attributes (like color or shape) until they eventually die.

Reynolds introduced a flocking model [133], that can be viewed as a generalization of Reeves' particle systems. With this approach, agents (called "boids") now often have a shape and a more complex geometrical state that includes orientation. Another important contribution is that, unlike particles in particle systems that do not interact with each other, boids interact strongly in order to flock or move together correctly. The behavior of boids consists of four separate components, one for each "desire": the boid wants to fly in the same direction as its neighbors, be in the center of the local cluster of boids, avoid collisions with close neighbors, and maintain a clear view ahead by skirting around others boids that block its view. Figure 2.2 illustrates each of these components. Each of the components is computed separately, then combined for movement. An important contribution of Reynold's work is the generation of successful overall group behavior while individual agents only sense their local environment and close neighbors. Figure 2.3 shows how a collection of boids self-organize into a flock.

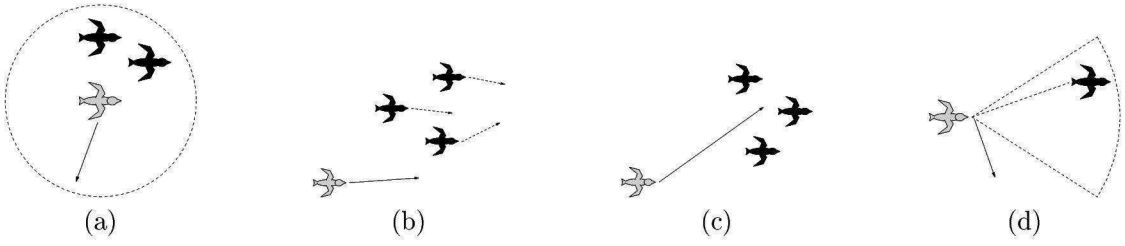


Figure 2.2: Four birds rules: (a) avoid flying too close to others, (b) match the velocity of near neighbors, c) move toward the center of local neighbors, d) attempt to maintain a clear view. Figure from *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation.* Copyright ©1998-2000 by Gary William Flake.

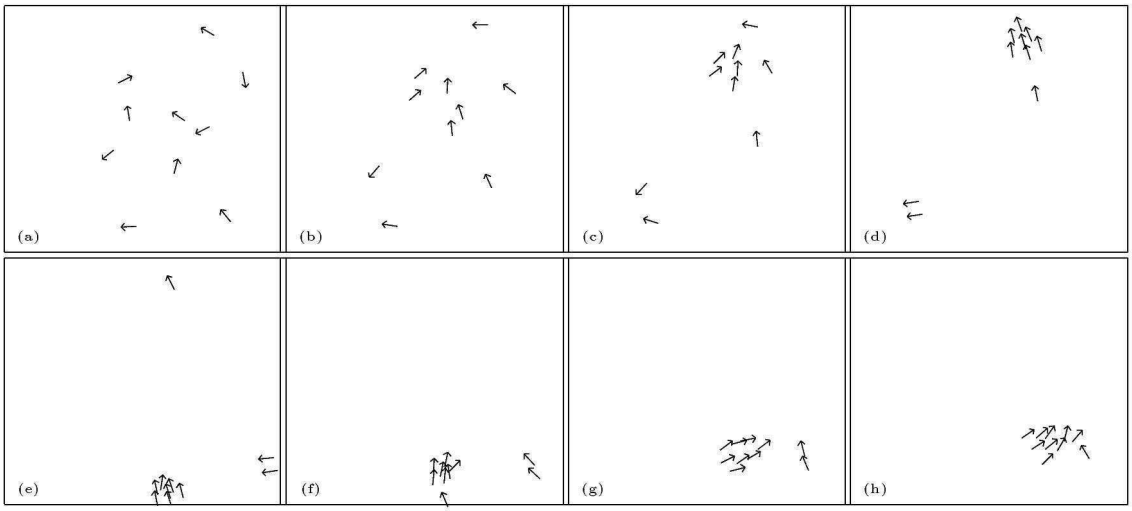


Figure 2.3: A collection of boids self-organize into a flock. Figure from *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation.* Copyright ©1998-2000 by Gary William Flake.

Improvements to this approach have recently been made by Terzopoulos, who developed artificial fish in a 3D virtual physical world [150]. This model not only emulates the appearance, movement, and behavior of individual animals, but also the complex group behaviors evident in many aquatic ecosystems. Emergent collective behaviors include collision avoidance, foraging, preying, schooling, and mating.

During recent years, the term *swarm robotics* emerged as the application of swarm intelligence to multi-robot systems. This area is similar to computer animation but with emphasis on physical embodiment of the entities and realistic interactions among the entities and between the entities and the environment. For example, a robotic approach in the same line of Reynolds' model shows that simple behaviors like avoidance, aggregation and dispersion can be combined to create an emergent flocking behavior in groups of wheeled robots [94, 95, 96]. Examples of tasks to which swarm robotics is currently applied include collective navigation [25][72][94][95], foraging [55][145][146], agent

chain formation [41][106][158], collective transportation[15][82][81][96][108], clustering [8][63][93] and self-assembly [48][68][102]. A particularly interesting application of swarm robotics principles can be found in the Swarm-bots project [40], where distributed adaptive controllers were used to control a group of real, or simulated, robots so that they perform a variety of tasks which require cooperation and coordination. The robot controllers were developed by an extensive use of artificial neural networks shaped by evolutionary algorithms. The robots have performed successfully in simulation and experimental tasks of coordinate motion, hole/obstacle avoidance, adaptive division of labor, cooperative transport and functional self-assembling [40].

2.2.2 Optimization

Another area of swarm intelligence that has been explored is how to transform models of social collective behavior into useful optimization and control algorithms. This line of research concerns the transformation of knowledge about how social organisms (including people) collectively solve problems into artificial problem-solving techniques in which the underlying model of intelligence is the collective intelligence of a social colony. Unlike the methods presented in the previous section, in these algorithms the swarm moves in a cognitive (and often high-dimensional) space where collision of agents is no longer a concern and is even encouraged. An example of a biologically-inspired model along these lines is bacterial chemotaxis [101], a model proposed by analogy to the strategy employed by bacteria to follow chemoattractants in a spatial concentration gradient to a food source. Bacteria, despite their relative simplicity, are able to acquire

information about their environment, orient themselves in this environment, and use this information efficiently to survive. Optimization algorithms based on this model gather information about the environment and use this information to move to an optimum location. They have been successfully applied to the training of artificial neural networks [33], multimodal function optimization [101], the design of aerofoils [101], and the control of robots for environmental monitoring [36]. The most well-known example of optimization algorithm in swarm intelligence is undoubtedly particle swarm optimization [43][73] and it will be described in detail in the next section.

2.3 Particle Swarm Optimization

Particle swarm optimization (PSO) algorithms have been inspired by the flocking behavior of birds and by social psychology [43]. This model operates as follows. A number of simple entities, the particles, are placed in the parameter space of some problem or function, and each evaluates the fitness at its current location. Each particle then determines its movement according to its own experience, and according to the experience of a neighboring particle, making use of the best position encountered by itself and its neighbor. During the search, the particles exchange information about their positions and fitness values. As a result of this communication, the swarm learns and refines its knowledge about the search, and moves towards the better search space areas. This is analogous to flocks of birds flying and searching for food, to social insects such as bees and ants when foraging or nesting, and to humans that affect the minds of each other by interacting socially. These algorithms have been used for numerical optimization,

learning weights for neural networks, tracking dynamic systems, and for tackling multi-objective optimization and constraint optimization problems [43]. This model presents a concept of neighborhood unlike most of the models used in physical space. In this *social neighborhood* the idea is that each agent's related neighbors are assigned initially and viewed as forming a fixed network, regardless of agent's spatial movements. Many neighborhood topologies have been developed, but two basic topologies remain very popular in the literature [43], the ring topology (local neighborhood) and the star topology (global neighborhood). These topologies are shown in Figure 2.4. In the star topology (also known as gbest), all the particles are neighbors of each other; thus, the position of the best overall particle in the swarm (i.e., the global best) is used in the social term of the velocity update equation. It is assumed that gbest swarms converge fast, as all the particles are attracted simultaneously to the best part of the search space. However, if the global optimum is not close to the best particle, it may be impossible to the swarm to explore other areas; this means that the swarm can be trapped in local optima. In the ring topology (also known as lbest), only a specific number of particles (neighbor count) can affect the velocity of a given particle. The swarm will converge slower but can locate the global optimum with a greater chance.

PSO models a set of potential problem solutions as a swarm of particles moving about in a virtual search space [43][73]. Every particle in the swarm begins with a randomized position ($\vec{x}_i$) and a velocity ($\vec{v}_i$) in the n-dimensional search space, where x_{ij} represents the location of particle index i in the j^{th} dimension of the search space. The current position $\vec{x}_i$ can be considered as a set of coordinates describing a point in space.

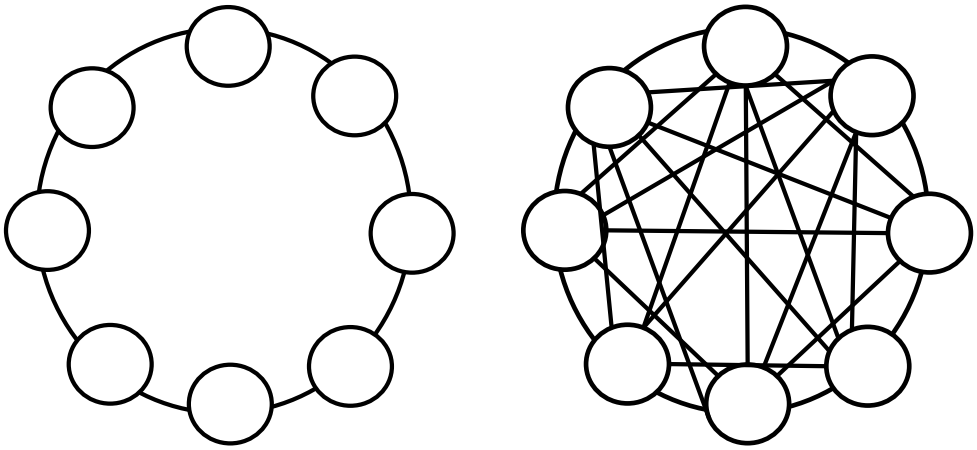


Figure 2.4: Examples of PSO neighborhood topology. a) ring topology (lbest) b) star topology (gbest).

Candidate solutions are optimized by flying the particles through the virtual space, with attraction to positions in the space that yielded the best results. 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 remembers which particle achieved the best overall position in that neighborhood ($\vec{n_i}$).

At each iteration of the algorithm, as shown in Algorithm 4, the position of each particle i is updated with a velocity that is a weighted sum of three components: the old velocity, a velocity component that drives the particle towards the location in the search space where it previously found the best solution so far ($\vec{b_i}$), and a velocity component that drives the particle towards the location in the search space where the neighbor particles found the best solution so far ($\vec{n_i}$). In algorithm 4 the terms $U(0, c_1)$ and $U(0, c_2)$ weight the contributions of b_i and n_i respectively. They are influential in striking a balance between the relative roles of the individual (cognitive) experience (governed by *cognitive-factor* c_1) and of the social communication (governed by *social-factor* c_2). Uniform ran-

Algorithm 4 Traditional PSO

```
for each particle i do
    initialize position x_i and velocity v_i
end for
while stop criteria not met do
    for each particle i do
        evaluate fitness(x_i)
        b_i ← best position found so far by the particle
        n_i ← best position found so far by its neighborhood
    end for
    for each particle i do
        v_i ← ω × v_i + U(0, γ1) × (n_i − x_i) + U(0, γ2) × (b_i − x_i)
        x_i ← x_i + v_i
    end for
end while
```

dom selection $U(i, j)$ of these two parameters avoids any a priori choice of the importance of either of the two sources of information. The inertia weight w acts as a scaling factor over the previous velocity. As a result, the new velocity includes a component of the previous velocity. Inertia values may be changed dynamically over time. By doing so, the early stages of the algorithm are favored towards exploration, while exploitation of local areas occurs at latter stages. A common strategy [142] is to linearly decrease w over time within the range [0,1] . Depending of the nature of the problem being optimized, different stopping criteria may be applied to stop the algorithm. Usually, a PSO algorithm executes for a maximum number of iterations or fitness functions evaluations. Another termination criteria can be when the average velocity update over all particles approximates zero, indicating the particles are no longer moving (i.e., the system is in equilibrium).

Extending PSO to more complex combinatorial search spaces is of great interest but it requires a clear definition of the relationship between particle position and a solution [122]. The definition of a particle as an encoded solution is usually called a *solution*

representation and the method to convert it to a problem specific solution is usually called a *decoding method*. This relationship, 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.

2.3.1 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 solving problems with binary-valued solution elements was presented in [74]. 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 $\{0, 1\}$, the particle's position equation is changed to:

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

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 of 1. The notion of velocity of the standard PSO representing the change of the particle's position x_{ij} is lost.

In a more general case, when integer solutions (not necessarily 0 or 1) are needed, the optimal solution can be determined by rounding off the optimum real values to the nearest integer. This was implemented in integer PSO [111][136]. Integer PSO uses the

same velocity and position equations of Algorithm 4. Once the new position is determined, its value in each dimension is rounded to the nearest integer value. Results of integer PSO were encouraging in seven integer programming test problems, and in some instances it outperformed the branch and bound method [86] with respect to the mean number of required function evaluations [85]. This PSO approach makes the implicit assumption that nearby discrete values are physically related, i.e., a position value of $x_{ij} = 1$ would move to $x_{ij} = 2$ easier than it would to $x_{ij} = 5$. Another recent approach to discrete PSO is multi-value PSO, MVPSO [126], which defines the particles' positions to contain the probabilities of solution elements assuming specific values. This strategy maintains a notion of position and velocity similar to the traditional PSO, but the performance of MVPSO decreased as the number of discrete values increased from binary to ternary [126].

The most common approach to PSO extensions for explicit combinatorial problems, such as the traveling salesman problem, is based on the principle of permutations [23][140][157]. In these algorithms, a particle's position is a specific permutation, and the velocity becomes a swap operator that transforms one permutation into another. Discrete PSO [23] redefines the basic arithmetic operators to work on discrete spaces and particles that represent permutations. A velocity operator is interpreted as a list of permutations of a particle, and a particle represents a candidate solution. For example, given the following particle position $\vec{x}_j = [1, 2, 3, 4, 5]$ and the velocity $\vec{v}_j = [(1, 2), (2, 3)]$. The next position is found by applying the first transposition of v_j to x_j, then the second one to the result, and so on. Thus, the new position in this example is $\vec{x}_j = [3, 1, 2, 4, 5]$.

Various other heuristic rules have been proposed to apply PSO to combinatorial problems while attempting to keep the particles in continuous space. For example, the smallest position value rule, 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 [149]. For example, having a particle position $\vec{x}_j = [1.80, -0.99, 3.01, -0.72, -1.20, 2.15]$ would represent the potential solution $\vec{s}_j = [5, 2, 4, 1, 6, 3]$. A hybrid PSO with this representation plus a local search method is found to be competitive for a sequencing problem [149].

2.3.2 Cooperative PSO

In cooperative PSO approaches, particles are explicitly or implicitly grouped into subswarms. Implicit grouping is achieved on the basis of particle behavior and overall state. Cooperation is mainly in terms of exchanging information about best positions found by different groups. An example of this approach is concurrent PSO, CONPSO [7], where two swarms search concurrently for a solution with frequent message passing of information. Breeding PSO, developed in [91], has implicit cooperation based on the exchange of genetic material between parents of different subswarms. Multi-Phase PSO, MuPSO [71], 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_k, [154], 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 represents the necessary mechanism that provides the glue for the cooperative approach. This context is unique during the evaluation of particular PSO, which means that all particles are evaluated using one template solution. CPSO-S_k also raises a problem in terms of swarm credit assignment. Each PSO swarm needs to be assigned credit for the quality of its participation in the overall solution. The danger is that the algorithm could spend too much time optimizing variables that have little effect on the overall solution. CPSO-S_k gets trapped in pseudo-minima, that is, a minimum that was created as a side effect of the partitioning of the search space [154]. To help alleviate this problem two strategies are proposed. The fist one consists of a second context vector made up of random components (as suggested by [125]) and the second one is to combine traditional PSO with CPSO-S_k, interleaving their execution and combining the results by replacing half of the population of one with the best solutions of the other. CSPSO-S_k was tested in several continuous optimization benchmark problems with promising results [154].

Chapter 3

Cooperative PSO for Combinatorial Problems with Shared Values

A class of problems often dealt with in the field of optimization is the class of the so-called combinatorial problems. *Combinatorial problems* are characterized by the consideration of a selection or permutation of a discrete set of "items" or by an assignment among these. The first cooperative strategy presented in this dissertation aims to solve combinatorial problems where candidate solutions are represented by a vector, *solution* $= \langle c_1, c_2, ..., c_n \rangle$, and each component c_j is selected from an ordered set of k possible values $Q = \{q_1, q_2, ..., q_k\}$.

To take a simple example, suppose we have have to deliver packages to different locations $L = \{l_1, l_2, l_3, l_4\}$. We can hire up to three drivers to deliver these packages, i.e., $D = \{d_1, d_2, d_3\}$. Each driver has his own fee for each location, and not every driver goes to all locations. Each driver has a fixed fee of 200 if he gets hired for the day, plus a fee for each package delivered in each location. The fee schedule is indicated in the following table:

	l_1	l_2	l_3	l_4
d_1	2	3	3	
d_2	5	2		5
d_3	4	3	2	

This means that if we select driver d_1, then the cost of delivering packages in l_1 is \$2 per package, \$3 per package in both locations l_2 and l_3, and no deliveries can be made to locations l_4. Given a set of locations that need deliveries *Input* $= \{l_1, l_2, ..., l_n\}$, we want

to determine the set of drivers D_s which are able to deliver all packages with minimum cost, where

$$\textit{cost} \;=\; 200|D_s| + \sum_{l_j \in \textit{Input}} \min_{\forall d_i \in D_s} \textit{np}_j \times \textit{fee}(d_i, l_j)$$

where np_j is the number of packages that need to be delivered to location l_j. In this example, each component is a location that needs a driver, therefore each solution component has a set of possible values $D = \{d_1, d_2, d_3\}$. A candidate solution is a collection of n drivers, one for each location where deliveries have to be made. Two locations can have the same driver, so the solution is reduced to the set represented by D_s. In this example, let us assume that we need to deliver 100 packages to location l_1 and 100 packages to location l_2 (i.e., $\textit{Input} = \{l_1, l_2\}$). The global minimum is the solution $D_s = \langle d_1 \rangle$ with $\textit{cost} = 200|\{d_1\}| + \textit{np}_1 \times \textit{fee}(d_1, l_1) + \textit{np}_2 \times \textit{fee}(d_1, l_2)$, $\textit{cost} = 200(1) + 100 \times 2 + 100 \times 3$, i.e., $\textit{cost} = 700$.

This chapter presents a new PSO model to deal with this type of combinatorial problem. *Shared-space CCPSO* consists of particles that move in a *shared high-dimensional continuous space*, and based only on local interactions generate a solution as a result of their collective behavior. Abductive diagnosis is the combinatorial task selected for the evaluation of shared-space CCPSO. The goal in abductive diagnosis is to obtain the best or most plausible explanation for the manifestations (symptoms) known to be present in a given case. Such problems are often quite complex due to the number of possible elementary hypotheses for each observation and the many different ways to combine these hypotheses into an explanation. Moreover, the plausibility of solutions depends on the interactions between explanation components in a highly non-linear fashion which makes

it a complex problem. These characteristics of abductive diagnosis makes it an especially attractive task to evaluate the performance of the proposed combinatorial PSO, CCPSO.

3.1 Shared-space CCPSO

In CCPSO, each solution component c_j is represented by a particle p_j. Each possible value in Q is represented by an attractor r_i. Particles move in a k-dimensional space, where $k = |Q|$. One attractor r_i is created for each q_i. Each attractor tries to pull particles to its position, i.e., tries to assign its value q_i to the solution component c_j. The algorithm ends when the system has reached equilibrium, i.e., all particles have arrived at an attractor's position. The solution represented by the swarm is decoded from the particle's positions. Each particle contributes the value represented by the attractor it picked. Note that the attractors are shared by all particles, and several particles can pick the same attractor. The candidate solution is the set of distinct values represented by this collection. For the example presented above, one particle p_j is created for each location that needs a driver. Since $Input = \{l_1, l_2\}$, two particles are created: $\mathit{particles} = \{p_1, p_2\}$. An attractor r_i is created for each possible driver. Since $\mathit{attractors} = \{r_1, r_2, r_3\}$, the space where particles move is three-dimensional. This is depicted in Figure 3.1.

More detailed description of the swarm, attractors and particles are given below.

Swarm A *swarm* represents a candidate solution for the optimization problem, and is defined by the tuple:

$$S = \langle \mathit{particles}, \mathit{sol}, \mathit{fitness} \rangle$$

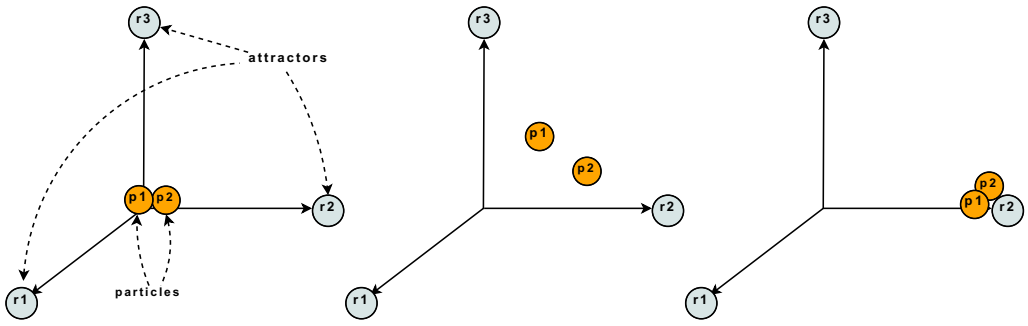


Figure 3.1: Components in shared-space CCPSO model with two particles and three attractors, *particles* $= \langle\, p_1, p_2 \,\rangle$, *attractors* $= \langle\, r_1, r_2, r_3 \,\rangle$ a) initial state with particles at origin, b) intermediate state, c) equilibrium state, all particles are at an attractor position.

where *particles* is the set of particles that belong to the swarm, *sol* is the candidate solution decoded from *particles*, *fitness* is the fitness value of the candidate solution evaluated according to the optimization fitness function.

Attractors An *attractor* represents a value q_i for components in the candidate solution. Attractors are represented by static particles r_i located throughout the k-dimensional space, where k is the number of possible values. Each attractor r_i is defined by the tuple:

$$r_i = \langle \vec{a}_i, \mathit{neighbors}_i, \mathit{st}_i, \mathit{ft}_i \rangle$$

where $\vec{a}_i$ is its location, $\mathit{neighbors}_i$ is the set of particles with which it interacts, st_i is the strength (heuristic parameter) that determines its attractive force, and ft_i is its fitness and represents how successful the attractor is in pulling particles to its position. The position $\vec{a}_i$ is initialized to a unit vector in a direction of the i^{th}coordinate axis perpendicular to $a_h \forall_{h<i}$.

Particles A *particle* represents one component in the candidate solution. Its position is used by the swarm to construct the candidate solution. Each particle p_j is defined by the tuple:

$$p_j = \langle \vec{x}_j, \vec{v}_j, \textit{neighbors}_j \rangle$$

where $\vec{x}_i$ is its current position, $\vec{v}_i$ is its current velocity vector, and $\textit{neighbors}_i$ is the set of attractors with which it interacts. The initial position $\vec{x}_j$ is set to the origin of the k-dimensional space, and its initial velocity vector $\vec{v}_j$ is set to zero. A new velocity is computed for each particle based on its previous velocity and the current force exerted on it, $\vec{F}_j$, where the equation for computing $\vec{F}_j$ is problem dependent. Then each particle's position is updated by simply adding the velocity vector. Parameter $\triangle_t$ regulates the step size. The step size may be changed dynamically over time but in the current implementation is set to 0.1. The updates of the particles are accomplished according to the following equations:

$$\vec{v}_j = \vec{v}_j + \triangle_t \cdot \vec{F}_j \tag{3.1}$$

$$\vec{x}_j = \vec{x}_j + \triangle_t \cdot \vec{v}_j \tag{3.2}$$

Algorithm Algorithm 5 presents the CCPSO pseudocode that combines the components described above. The system is initialized at time $t = 0$ by creating one swarm. One particle is defined per element of the combinatorial problem. An attractor is created for each possible value of the variables. The iterative process starts after this initialization. For each time step, each particle updates its position vector by simply adding a new velocity vector (Equation (3.2)). This new velocity vector is computed based on the influ-

ence exerted by the attractors and then the attractors recompute their fitness values based on their proximity to the particles. The closer particles are to an attractor, the higher its fitness. A key component in the algorithm is this feedback mechanism. It rewards attractors which are successful in pulling particles towards them. This way, each possible value for a solution component (i.e., each attractor) gets feedback that reflects its contribution to the overall solution. The process iterates until all particles have arrived at a stationary position (i.e., the system is in equilibrium) or until a maximum number of time steps is reached.

Algorithm 5 Shared-space CCPSO algorithm

for each element $\in$ *Input* **do**
  create particle j
  initialize position $\vec{x}_j$ to $\overrightarrow{0}$
  initialize velocity $\vec{v}_j$ to $\overrightarrow{0}$
end for
for each $q_i \in Q$ **do**
  create an attractor r_i
  initialize position a_i
  compute st_i
end for
while stop criteria not met **do**
  for each particle j **do**
    compute $\vec{F}_j$ exerted by attractors in $neighbors_i$
    $\vec{v}_j = \vec{v}_j + \triangle_t \times \vec{F}_j$
    $\vec{x}_j = \vec{x}_j + \triangle_t \times \vec{v}_j$
  end for
  for each attractor i **do**
    $ft_i \propto \sum_{p_j \in neighbors_i} proximity_{ij}$
  end for
end while
decode solution sol_s
evaluate $fitness(sol_s)$

3.2 Application: Abductive Diagnosis

Abductive diagnosis is used as a first test of the benefits of shared-space CCPSO. The complex and highly non-linear nature of this task makes it an especially interesting combinatorial problem to evaluate the performance of the proposed combinatorial PSO. *Abduction*, or inference to the best explanation, is a form of inference that goes from data to a hypothesis that best explains this data [69]. The philosopher Pierce first introduced the notion of abduction and distinguish it from deduction and induction [115]. A modern interpretation of this distinction is as follows:

- Deduction, a process based on the application of general rules to a specific case, with the inference of a specific result.
- Induction, reasoning in which from a specific case and a specific result, a general rule can be hypothesized.
- Abduction, inference where from a general rule and a specific result, a general explanation can be hypothesized.

Both induction and abduction involve making and testing hypotheses that cannot be deduced. The difference between these two processes is that in induction learning what is being hypothesized is the general rule (new knowledge) while in abduction it is the specific case (i.e., an application of existing knowledge). Also, in induction there is usually a large number of situations that collectively support the general rule, while in abduction the inference is usually conducted with data about a single situation.

In abduction, given a set of facts, the goal is to find one or more plausible hypotheses that can explain or account for these given facts. This inference process is often viewed as involving two steps: hypothesis generation to construct explanatory hypotheses, and hypothesis disambiguation to pick *the best ones* among all possible hypothesis [119]. The problem arises of how to choose the best among the entire set of possible hypothesis. Peirce gives several criteria that a good explanation should fulfill [115]. First, and foremost, a hypothesis should account for all the facts. Second, it should be the "simplest" hypothesis available. In general, *simplicity* is interpreted as logical simplicity, which means that the hypotheses that contain fewer elements are preferred, but other views exist.

A prominent application of abduction has been diagnostic problem solving. For example, in medicine the diagnosis task can be viewed as producing an explanation (set of disorders) that best accounts for a patient's manifestations (symptoms). Examples of abductive models for diagnostic problem solving are [69][97][124][127][131][132].

Abductive inference has also been used in other fields. For example, it has been used for high-level computer vision applications [26]. In this case, the hypotheses are the objects to be recognized, and the observations are partial descriptions of objects. In natural language understanding, abduction has been used to interpret ambiguous sentences [62][147], the abductive explanations corresponding to the various possible interpretations of such sentences. Another application in the case of natural language understanding is the problem of building a plausible model for a story so that questions about events can be answered [29]. In planning problems, plans can be viewed as "explanations" for the given goal state to be reached [45][141]. In plan recognition, given a theory

describing how actions relate to goals and an observation of agent action, agent goals can be conjectured to account for the actions.

3.2.1 Diagnostic Problem Solving

A diagnostic problem is a problem in which one is given a set of manifestations and must explain why they are present by using one's knowledge about the world. Diagnostic problems can be found in various areas, e.g., diagnosis in clinical medicine, computer program debugging, and fault localization in circuits. In this type of abductive problem, for a set of manifestations (observations or symptoms), diagnostic inference consists of finding the most plausible set of disorders that can explain why the manifestations are present. In general, an individual disorder can explain only a portion of the observations and therefore it is necessary to find a solution which consists of multiples disorders (i.e., a multi-disorder solution) that explain all of the observations that are present. For example, often a medical doctor will need to postulate the presence of multiple disorders to explain a patient's symptoms.

Two kinds of knowledge are usually employed in diagnostic systems: structural and probabilistic knowledge. Structural knowledge (usually in symbolic form in knowledge-based systems) specifies the entities of interest in an application domain and which entities are associated by what kind of associations. Among these different kinds of associations, cause-effect relations are probably the most important in diagnostic problem-solving. For example, in medical diagnosis, the statements "disorder d_i may be a cause of manifestation m_j", or, "m_j may be a manifestation of d_i" associate d_i and m_j together. Probabilistic

knowledge, on the other hand, reflects the uncertain aspects of such associations. Probabilistic knowledge (usually in numeric form in knowledge-based systems) specifies the strength of associations and prior probabilities of individual entities. Probabilities may be represented as actual numbers (0.5, 0.01, etc.) or in non-numeric symbolic form ("very common", "moderately common", "very rare", etc.) [119].

An associative network of causal relationships allows the representation of both structural and probabilistic knowledge. This network is a graph where edges represent direct causal effects, and the edges are labeled with a weight that represents the strength of that relationship. This network thus supports and encompasses causal inferences plus probabilistic reasoning. In these networks, the sets of possible disorders $D = \{d_1, d_2, d_3, ..., d_n\}$ and possible manifestations $M = \{m_1, m_2, m_3, ...m_p\}$ are represented by the nodes of the graph. As illustrated in Figure 3.2, each disorder $d_i \in D$ is associated with a number $p_i \in (0, 1)$ that represents its prior probability, and each causal link is associated with a number $c_{ij} \in (0, 1]$ that represents how frequently d_i causes m_j. Figure 3.2 shows an example where $D = \{d_1, d_2, d_3\}$ and $M = \{m_1, m_2, m_3, m_4, m_5\}$. In this network the link between d_1 and m_1 establishes that disorder d_1 causes manifestation m_1 with a probability of 0.2, and the prior probability that disorder d_1 is present is 0.01.

This representation of the diagnosis problem makes three assumptions: i) disorders d_i are independent of each other, ii) causal strengths are invariant, i.e., d_i causes m_j with the same probability whenever d_i is present, and iii) no manifestation is present without being caused by some disorder. This later assumption can be dropped by including

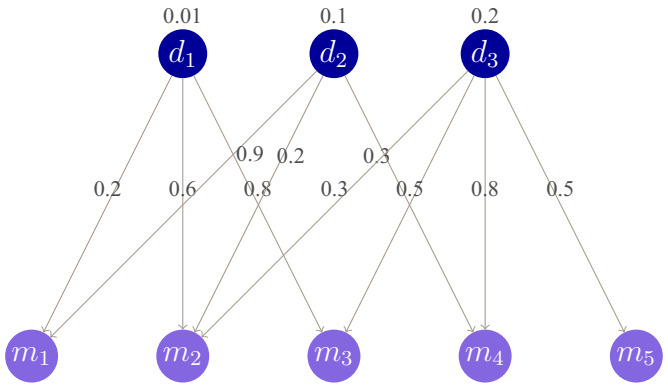


Figure 3.2: A very simple causal network (real-world diagnostic networks are much larger). Disorders are pictured at the top, manifestations at the bottom. Values of prior probabilities for disorders and causal strengths for causal links are indicated.

a "dummy" disorder with links to all manifestations, linking it all manifestations with the probability that m_i is present without any causal disease. An instance of a diagnostic problem consists of a causal network plus a particular set of observed manifestations M^+. For example, $M^+ = \{m_1, m_2\}$ represents the case where m_1 and m_2 are the manifestations observed for which an explanation is needed, while other manifestations are absent.

One abductive theory of diagnosis based only in structural knowledge is the *set covering model* [131][132]. This model is based on the notion of parsimoniously covering a set of observable manifestations, that is, finding the *simplest* explanation for the given set of manifestations. Several parsimony criteria have been proposed for this model [103]. A very simple one is to have a minimum cover, that is, the cardinality of hypothesis is the smallest among all explanations. In other words, this minimal cardinality criteria adopts the notion that smallest hypotheses (containing fewer disorders) are better or more plausible than larger ones. Another often used parsimony criteria is *irredundancy* (sometimes

also called minimality) where a hypothesis is plausible if it does not contain any superfluous disorders. In other words, the removal of any disorder from this set will make the hypothesis no longer capable of explaining all manifestations present. In many diagnostic problems, such as in medicine, it is preferable to take all the plausible diseases into consideration rather than have only one candidate disease for the cause. Therefore irredundant cover has been often used as the parsimony criterion for abductive inference.

A limitation of parsimonious set covering theory is that the solution for a diagnostic problem may include a large number of alternative hypotheses. In order to discriminate further among these potential explanations, some criteria other than parsimony are needed. The *probabilistic causal model* based on Bayesian classification integrates symbolic cause-effect inference with numeric probabilistic information and provides a formal probability theory capable of determining the relative likelihood of multi-disorder hypotheses [116][117]. With this new measure it is possible to select the most-probable hypothesis among all possible hypotheses.

Traditional approaches to abduction can be applied to the diagnosis problem. However, finding a multi-disorder solution that satisfies some criteria for being the best explanation (such as plausibility, explanatory coverage and parsimony) is computationally very expensive. The reason for this high computational complexity is the large size of the space of possible combinations of individual hypothesis. This space is exponentially large [119].

Two approaches have been considered to solve this difficulty. One approach is to develop traditional AI algorithms that focus the diagnostic reasoning in a restricted diag-

nosis space so that combinatorial explosion can be avoided [30][128]. These algorithms can be viewed as a special case of Bayesian/causal network methods [78]. Another approach is to develop problem solving algorithms involving parallel processing, in this respect several connectionist models have been explored, for example [118][164].

3.3 CCPSO for Diagnostic Problem Solving

A mapping from disorders and manifestations in the causal network to the shared-space CCPSO model is needed to be able to apply it to diagnostic problem solving. This mapping is explained below. A solution to a diagnosis problem is defined as set of disorders with maximal explanatory coverage and maximal plausibility. The solution components are the disorders and a candidate solution is a set of disorders, one for each observed manifestation. Disorders and manifestations are represented by two classes of particles that interact in an k-dimensional space, where $k = |D|$, the number of possible disorders. Each attractor representing a disorder interacts only with the set of manifestation particles that it is adjacent to (i.e., that it causes) in the causal network. This defines a fixed neighborhood for each particle. For example, in the causal network in Figure 3.2, the attractor that represents d_1 would only interact with the particles that represent m_1, m_2 and m_3. The particles that represent manifestations will move throughout the space, based on the influences exerted by the attractors (the particles that represent disorders), and at the end of the simulation, the final position of the manifestation particles will be used to obtain the solution to the diagnosis problem.

To keep within the notation used in diagnostic problem solving, attractors will be

referred as *d-particles*, and particles as *m-particles*.

Subswarm → candidate hypothesis A *subswarm* represents a candidate hypothesis D_s for the diagnostic problem. The hypothesis D_s for the diagnostic problem is the set of disorders represented by the attractors selected by each of the particles at the current iteration step.

Disorders → d-particles As noted above, disorders are represented by attractors. For each $d_i \in D$, a d-particle labeled d_i is created. The position $\vec{a}_i$ of d-particle d_i is initialized to a unit vector in a direction of the i^{th}coordinate axis perpendicular to the positions of all previous d-particles. For example, for the causal network of Figure 3.2, $n = 3$, a 3-dimensional space, and the d-particles representing the disorders d_1, d_2 and d_3 would be at $\vec{a}_1 = \langle 1, 0, 0 \rangle$, $\vec{a}_2 = \langle 0, 1, 0 \rangle$ and $\vec{a}_3 = \langle 0, 0, 1 \rangle$. The strength value st_i is determined using a single non-local computation before processing starts based on the prior probability of the disorder d_i and the set manifestations that are expected to be observed when this disorder is present but are not. M^- is the set of manifestations that are not present in the instance. This value is computed by using:

$$st_i = p_i \times \prod_{m_l \in M^-} (1 - c_{il}) \tag{3.3}$$

where $M^- \equiv M - M^+$.

The fitness value ft_i is updated at each iteration based upon the feedback that the m-particles give to the d-particle d_i. All d-particles are initialized with the same fitness

value of 1.0, and this value is recomputed at each time-step based on the success of its interactions with m-particles (i.e., the closer that m-particles get to d_i's position, the higher the fitness of d_i). Specifically,

$$ft_i = \sum_{m_j \in M^+} proximity_{ij} \tag{3.4}$$

where $proximity_{ij} = \frac{1}{\sqrt{(\vec{a}_i - \vec{x}_j)^2}}$.

Manifestations → m-particles Manifestations are represented by particles that move throughout the space. For each $m_j \in M^+$ an m-particle labeled m_j is created. The initial position $\vec{x}_j$ of particle m_j is set to the origin of the k-dimensional space, and its initial velocity vector $\vec{v}_j$ is set to zero. The particles' updates are accomplished according to Equation (3.1) and Equation (3.2). The value of the new influence of other particles, $\vec{F}_i$, is the result of all forces exerted by attractors on particle m_j as shown below in Equation (3.5). Equation (3.6) shows that this attraction force is dependent upon particular d-particle's attributes: its strength (st_i) and its ability to attract d-particles, fitness (ft_i), and upon interactive values such as the causal strength c_{ij} and the directed distance $\vec{u}_{ij}$ between the m-particle and the d-particle. Specifically:

$$\vec{F}_i = \sum_{d_i \in neighbors_j} \vec{F}_{ij} \tag{3.5}$$

$$\vec{F}_{ij} = st_i \times ft_i \times c_{ij} \times \textit{proximity}_{ij} \times \vec{u}_{ij} \tag{3.6}$$

where $\vec{u}_{ij} = \frac{\vec{a}_i - \vec{x}_j}{|\vec{a}_i - \vec{x}_j|}$.

Algorithm As a simple pre-processing step to avoid useless computations, D is reduced to D', where $D' \subset D$ such that only disorders that have a causal relationship with some $m \in M^+$ are included in D' and the rest of the possible disorders are discarded. The system is initialized at time $t = 0$ by creating a particle for each observed manifestation, i.e., for each $m \in M^+$. Then, a d-particle is created for each disorder in D', its position is initialized and its strength value is computed using Equation (3.3).

The iterative process starts after this initialization. For each time step, each m-particle updates its position vector by simply adding a new velocity vector (Equation (3.1)). This new velocity vector is computed based on the influence exerted by neighboring d-particles. Equation (3.5) defines how to get the resultant force of all neighboring d-particles, and Equation (3.6) shows how to compute the individual force of each one. Then all d-particles recompute their fitness value based on the new m-particles' position, following Equation (3.4). The process iterates until all m-particles have arrived at the position of a d-particle, (i.e., the system is in "equilibrium"), or until a maximum number of time steps is reached.

Once the iterative process has terminated, a solution is extracted. If equilibrium has been reached and each m-particle has "picked" a winner attractor, the solution D_s is the set of disorders represented by these winner d-particles. Note that more than one m-particle may move close to the location of the same d-particle, thus picking the same winner. If

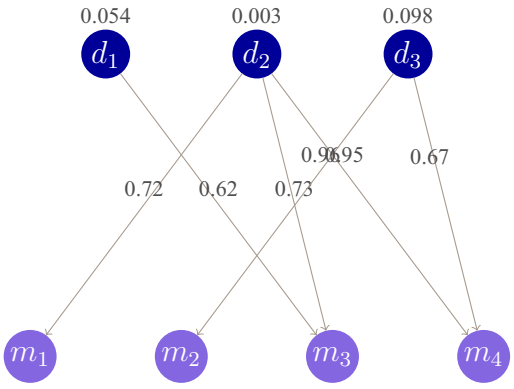


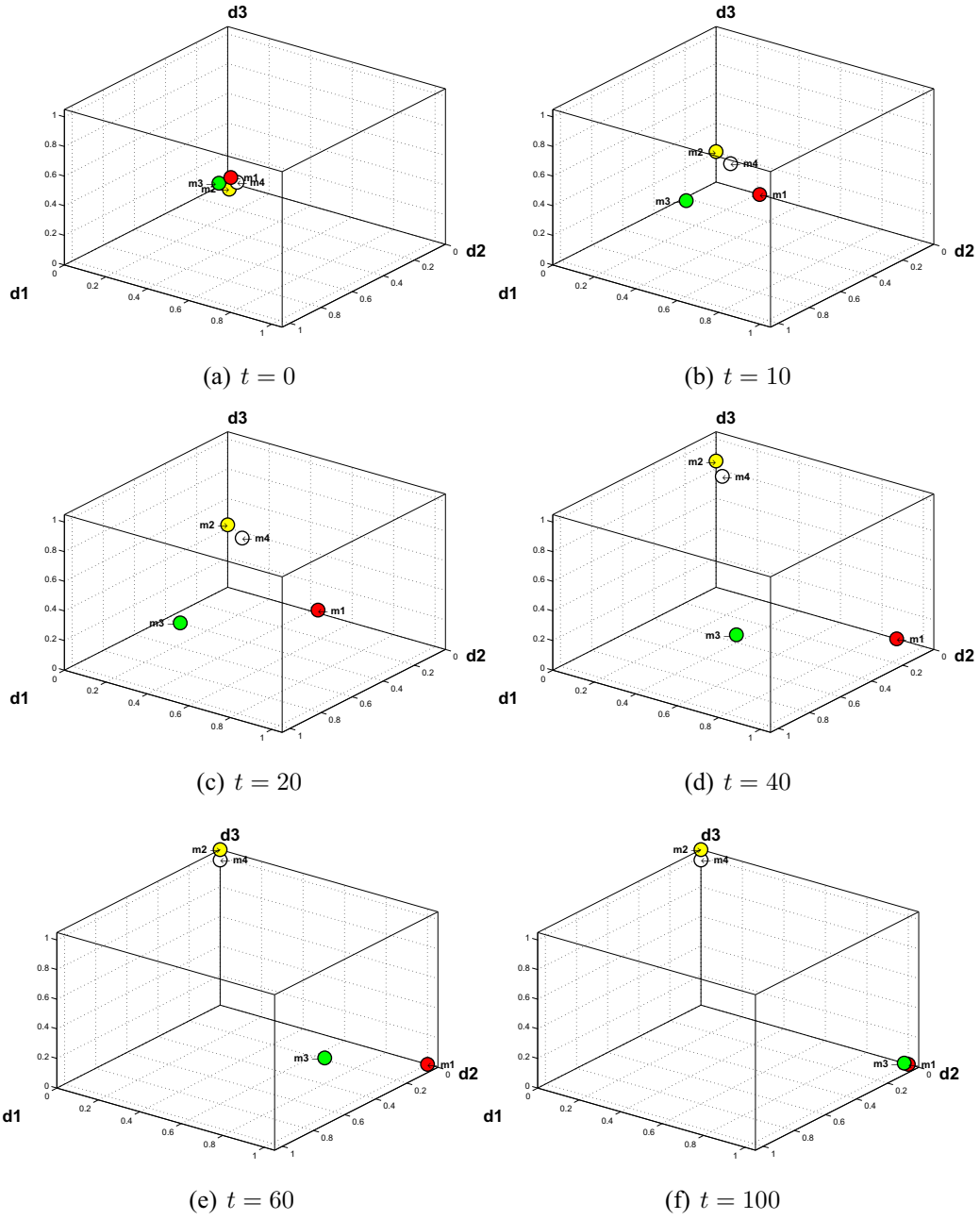
Figure 3.3: Causal network of shared-space CCPSO in Figure 3.4.

one or more of the m-particles is unsettled (not close enough to a d-particle's position) then a solution is not obtained. To determine whether an m-particle is "close enough" to a d-particle, a simple delta function is applied in each dimension of the position vector $\vec{x}$. For dimension k of the position vector $\vec{x}$, this delta function is:

$$\vec{x}_k = \begin{cases} 1 & \text{if } \vec{x}_k > 0.95 \\ 0 & \text{if } \vec{x}_k < 0.05 \\ \textit{unsettled} & \textit{otherwise} \end{cases}$$

Example Figure 3.4 shows an example output of shared-space CCPSO for the randomly generated causal network of Figure 3.3. For this particular instance there are four manifestations present $M^+ = \{m_1, m_2, m_3, m_4\}$. For illustrative purposes, the space in which the particles will interact actually has three dimensions, i.e., $|D'| = 3$; the 3D space is defined by d_1, d_2, and d_3.

The system is initialized at time 0 by deploying the four m-particles m_1, m_2, m_3 and m_4 with initial positions of $\{0, 0, 0\}$ as shown in Figure 3.4a. After a few iterations,



(a) $t = 0$ (b) $t = 10$

(c) $t = 20$ (d) $t = 40$

(e) $t = 60$ (f) $t = 100$

Figure 3.4: Simple example of shared-space CCPSO execution. The graphs show how the m-particles move around the space until they settle. The system is shown at different time steps. Note how particle m_3 initially moves towards d_1 but ends near d_2.

the m-particles are starting to move away from the origin of the coordinate system, as depicted by Figure 3.4b. It can be seen that m_1, m_2 and m_3 seem to have chosen a direction and are moving toward an specific d-particle; m_1 is headed towards d_2, m_2 towards d_3, m_3 towards d_1. But m_4 is moving in between d_2 and d_3, with a small preference for the former. This behavior continues for a few more iterations, as shown by Figure 3.4c at time 20. At time 40, as depicted by Figure 3.4d, there have been some changes. First, m_4 is now moving more clearly towards d_3, the influence of d_2 over this particle seems to have weakened. Secondly, m_3 has changed direction, from a trajectory almost completely towards d_1 and it is now moving towards the space between d_1 and d_2. This change of direction of m_3 can be attributed to the increased fitness value of d_2, which in turn has been getting stronger feedback from m_1. This tendency continues (Figure 3.4e shows the system at time 60), and finally, at time 100 the system reaches equilibrium, as depicted by Figure 3.4f. Solution D_s is obtained from the final "winner" position of the m-particles: m_1 and m_3 have selected d_2 while m_2 and m_4 chose d_3; so the solution is $D_s = \langle d_2, d_3, d_2, d_3 \rangle$, which is simplified to $D_s = \{d_2, d_3\}$.

3.4 Computational Validation of Shared-space CCPSO

Shared-space CCPSO was systematically evaluated using computer simulations with two randomly-generated causal networks and a substantially bigger network taken from a medical domain. Bayesian methods were used to identify the correct answers, i.e., the actual most likely set of disorders D^+ (second most likely, third, etc.) given the prior and conditional probabilities of the network and the manifestations known to be present.

Specifically, the solution to a problem, the most probable hypothesis D^+, is the one with the highest *relative likelihood* (as defined by [119]) among all possible hypotheses. An exhaustive search algorithm was used to identify the most probable hypothesis D^+. Since the presence/absence of disorders is possible, the potential space for searching for a solution is 2^D, an extremely large search space for an exhaustive algorithm to be used in practice. The relative likelihood measure for a hypothesis D_s given M^+ is computed by Equation (3.7):

$$L(D_s, M^+) = \prod_{m_j \in M^+} [1 - \prod_{d_i \in D_s} (1 - c_{ij})] \times \prod_{m_l \in M^-} \prod_{d_i \in D_s} (1 - c_{il}) \times \prod_{d_i \in D_s} \frac{1 - p_i}{p_i}. \quad (3.7)$$

A solution that is one of the best three is considered to be an acceptable approximation (a near-optimal solution), and any other solution to represent a failure, so the exhaustive Bayesian algorithm for D^+ actually computes the three best solutions. The purpose of these experiments is to determine whether shared-space CCPSO would produce a valid solution on a set of multiple winners, and if this proposed solution compares favorably with the most probable hypothesis.

The first evaluation used two randomly generated networks taken from [118]. Each network has 10 manifestations, 10 disorders and a maximum of 40 connections (causal associations) between individual disorders and their manifestations. These networks were randomly generated: for each d_i, its prior probability p_i, how many manifestations are causally associated with it, which ones they are, and what their respective causal strengths are, were all generated randomly and independently. The details of the two causal networks are given in Table 3.1 and Table 3.2. Causal strengths not listed are assumed to be

Table 3.1: Details of random causal Network #1.

$p_1 = 0.026$	$p_2 = 0.014$	$p_3 = 0.054$	$p_4 = 0.060$
$p_5 = 0.003$	$p_6 = 0.023$	$p_7 = 0.048$	$p_8 = 0.079$
$p_9 = 0.098$	$p_{10} = 0.027$		
$c_{12} = 0.31$	$c_{14} = 0.85$	$c_{17} = 0.30$	$c_{27} = 0.50$
$c_{32} = 0.29$	$c_{34} = 0.64$	$c_{35} = 0.15$	$c_{36} = 0.11$
$c_{38} = 0.72$	$c_{39} = 0.62$	$c_{43} = 0.88$	$c_{48} = 0.27$
$c_{51} = 0.72$	$c_{52} = 0.92$	$c_{54} = 0.07$	$c_{55} = 0.47$
$c_{59} = 0.73$	$c_{5,10} = 0.96$	$c_{66} = 0.32$	$c_{67} = 0.26$
$c_{71} = 0.05$	$c_{72} = 0.80$	$c_{75} = 0.73$	$c_{78} = 0.58$
$c_{82} = 0.04$	$c_{84} = 0.26$	$c_{86} = 0.23$	$c_{87} = 0.69$
$c_{8,10} = 0.51$	$c_{92} = 0.12$	$c_{97} = 0.95$	$c_{9,10} = 0.67$
$c_{10,4} = 0.43$	$c_{10,5} = 0.18$	$c_{10,6} = 0.11$	

zero. For example, in Network #1, d_1 has three non-zero causal strengths, 0.31, 0.85, and 0.30, to manifestations m_2, m_4, and m_7, respectively.

In these networks, disorders are assumed to have very small prior probabilities; in Network #1, $0 < p_i < 0.1$. For example, p_1 = 0.026, p_2 = 0.014. For Network #2, the prior probabilities are in the range $0 < p_i < 0.2$. The reason for restricting p_i to small numbers is that in most real-world diagnostic problems, such as those in medicine, prior probabilities of disorders are very small [116][117].

For each network, each of the 2^{10} non-empty sets of manifestations was used to define a problem instance. Thus, 1023 problem instances were tested for each network.

3.4.1 Results in Random Networks

In these experiments all 2046 cases reached equilibrium, and each case produced a set of winning disorders. Simulations converged in less than 200 iterations. The results

Table 3.2: Details of random causal Network #2.

$p_1 = 0.17$	$p_2 = 0.07$	$p_3 = 0.03$	$p_4 = 0.12$
$p_5 = 0.135$	$p_6 = 0.18$	$p_7 = 0.075$	$p_8 = 0.03$
$p_9 = 0.14$	$p_{10} = 0.05$		
$c_{12} = 0.06$	$c_{14} = 0.68$	$c_{16} = 0.10$	$c_{17} = 0.51$
$c_{21} = 0.53$	$c_{23} = 0.81$	$c_{24} = 0.09$	$c_{25} = 0.85$
$c_{28} = 0.13$	$c_{29} = 0.34$	$c_{2,10} = 0.85$	$c_{32} = 0.54$
$c_{35} = 0.45$	$c_{36} = 0.90$	$c_{37} = 0.59$	$c_{3,10} = 0.29$
$c_{42} = 0.74$	$c_{45} = 0.52$	$c_{47} = 0.65$	$c_{49} = 0.32$
$c_{53} = 0.72$	$c_{58} = 0.49$	$c_{63} = 0.09$	$c_{65} = 0.66$
$c_{6,10} = 0.44$	$c_{73} = 0.22$	$c_{74} = 0.46$	$c_{75} = 0.21$
$c_{76} = 0.76$	$c_{7,10} = 0.43$	$c_{81} = 0.29$	$c_{82} = 0.34$
$c_{88} = 0.25$	$c_{91} = 0.39$	$c_{94} = 0.20$	$c_{95} = 0.90$
$c_{96} = 0.48$	$c_{97} = 0.38$	$c_{10,2} = 0.74$	$c_{10,8} = 0.27$

Table 3.3: Results of shared-space CCPSO algorithm in random networks.

	Network # 1		Network # 2		Total	
Optimal	609	59.5%	491	48.0%	1100	53.8%
Near-optimal	291	28.5%	270	26.4%	561	27.4%
Other	123	12.0%	262	25.6%	385	18.8%

for these experiments are given in Table 3.3. This table shows that the solution D_s from shared-space CCPSO agreed with the most probable Bayesian hypothesis D^+ (i.e., $D_s = D^+$) in about 59% of the cases with Network #1 and 48% with Network #2. For those cases where shared-space CCPSO solution did not agree with D^+, the winner hypothesis fulfilled the explanatory coverage requirement, (i.e., all observed manifestations can be accounted for by the disorders in D_s), so these are still all valid solutions. The model finds a near-optimal solution (a solution within the first three most probable hypothesis) in 81.2% of the cases.

3.4.2 Improving Solutions

For the results where shared-space CCPSO did not obtain the optimal solution (i.e., $D_s \neq D^+$), this inconsistency was usually due to some m-particle whose local optimization (i.e., selecting a winner d-particle) did not lead to a globally optimal solution. That is, either a d-particle is selected as a winner and is present in D_s when it does not belong to D^+ (it should not have been selected as a winner), or a d-particle was not selected as a winner and is not present in D_s but it does belong to D^+ (it should have been selected). For example, for the causal network in Figure 3.2 and $M^+ = \{m_1, m_3\}$, the solution is $D_s = \{d_2, d_3\}$ with m_1 picking d_2 as a winner and m_3 picking d_3. This is actually the second best solution, because the optimal solution is $D^+ = \{d_1\}$. In this case what happened is that each m-particle got attracted to its individual most likely disorder (for $M^+ = \{m_1\}$, the optimal solution $D^+ = \{d_2\}$ and for $M^+ = \{m_3\}$ the optimal solution $D^+ = \{d_3\}$), but they failed to generate the solution that was best for the whole set. Initially, d_1 gets some feedback from both m_1 and m_3 (by means of its fitness value ft_1) but this feedback gets quickly overridden by the attraction that d_2 exerts over m_1 and d_3 over m_2. So shared-space CCPSO became trapped in a local optimum.

Post-processing methods can be used to further improve the solution accuracy with a moderate increase in the computational cost. An example is iterative improvement as suggested by [65] in which the current solution is compared with all its neighbor disorder sets (differing by only one disorder instantiation) and moved to the best neighbor if it is better. Another method is the *resettling process* used in connectionist models (e.g., [118]) where the computation is re-executed with the activation of a node clamped to the opposite

Table 3.4: Results of shared-space CCPSO algorithm with partial resettling in random networks.

	Network # 1		Network # 2		Total	
Optimal	964	94.2%	847	82.8%	1811	88.5%
Near-optimal	59	5.8%	153	14.9%	212	10.4%
Other	0	0.0%	23	2.2%	23	1.1%

value found previously, while allowing all others nodes to change as usual. This allows the system to escape some local optima. The second approach is selected here; it is more powerful since the new solutions are obtained by searching a restricted subspace instead of direct assignment.

Taking the resettling process of [118] as inspiration, two local search methods are implemented. The first approach to help shared-space CCPSO avoid being trapped in local optima consists of running it once and obtaining D_s as usual, but then taking every disorder in D_s and re-executing shared-space CCPSO without creating the d-particle that represents that disorder. For the example mentioned above, with $D_s = \{d_1, d_3\}$, shared-space CCPSO is re-executed twice, the first time is executed with $D' = \{d_2, d_3\}$ (thus omitting d_1), and the second time with $D' = \{d_1, d_2\}$.

Adding this *partial local resettling* to the algorithm greatly improves the accuracy of the solution. For Network #1 there is an increase of more than 30%, going from 60% to 94% for the optimal solution, and now all 100% of the cases are near-optimal solutions. For Network #2 there is a similar effect, the percentage of optimal solutions found increases from 48% to 83%, and 97.8% are at least near-optimal. The complete results are given in Table 3.4.

Table 3.5: Results of shared-space CCPSO algorithm with full resettling in random networks.

	Network # 1		Network # 2		Total	
Optimal	989	96.7%	946	92.5%	1935	94.6%
Near-optimal	34	3.3%	73	7.1%	107	5.2%
Other	0	0.0%	4	0.4%	4	0.2%

Expanding this idea, a *full local resettling* method is implemented by re-executing shared-space CCPSO n times (where $n = |D'|$) by reducing D' by one possible disorder each time. Table 3.5 shows the results of this full exploration applied to the two random networks. There is a small improvement in Network #1, where the optimal solution goes from 94% to 96.7%. Network #2 gets a larger benefit: it increases almost 10% going from 82.8% to 92.5%. However, this second post-processing method is significantly more expensive than the previous one, since $|D_s| \ll |D'|+$ for most cases.

3.4.3 Scaling up to a Larger Real-World Network

To examine if this model would scale up to larger and more realistic networks, a substantially larger network with 56 manifestations, 26 disorders and 384 causal links in a medical domain was taken from [156]. This knowledge base includes neurological and psychiatric disorders, such as dominant hemisphere infarct, B12-deficiency, Parkinson's disease, delirium tremens, paranoid schizophrenia and other related disorders. Unlike the random networks, the causal associations, prior probabilities and conditional probabilities were all based on subjective estimates from physicians, and thus this network presumably has a structure more like those that occur in real diagnostic domains.

Table 3.6: Results of shared-space CCPSO algorithm in network #3 (Neuropsychiatric Diagnosis).

	Original algorithm		Partial exploration		Full exploration	
Optimal	561	47.4%	817	69.1%	1074	90.9%
Near-optimal	177	15.0%	195	16.5%	74	6.2%
Other	444	37.6%	170	14.4%	34	2.9%

To understand the size difference between the first two random examples and this latter one, consider the space of potential solutions. A network with 10 disorders represents a search space having only $2^{10} = 1024$ potential solutions, whereas a network with 26 disorders represents a space of $2^{26} = 67,108,864$ potential solutions. Also, this is not an abstract and randomly generated network, but a network that is based on a neuropsychiatric diagnostic application in which the network structure and the causal relationship between disorders and manifestations were designed by physicians. This network will be referred to as Network #3.

For this network, since testing with all possible cases is unfeasible, over 1000 cases were randomly generated. Each case consists of a random choice of either 1,3,5,7 or 9 manifestations. Note that even for this small set it is computationally expensive to compute the Bayesian optimal D^+ with the exhaustive search algorithm that was used for the small random networks in Section 3.4.1.

The results for these experiments are given in Table 3.6. Even though this is a considerably larger and more complex network than the previous examples, shared-space CCPSO is able to reach equilibrium and provide a valid solution in all cases. All cases

converged in less than 200 iterations. In this example, the impact of the post processing process is again evident; the accuracy of solutions goes from 47% when no post processing is done, to almost 91% with full post processing.

3.5 Discussion

This chapter introduced a novel cooperative strategy to extend PSO algorithms to combinatorial problems. This new PSO algorithm preserves the concept of particles that move throughout a continuous high-dimensional space. A new cooperative strategy partitions the search space by splitting the combinatorial problem into components and using one particle to optimize each component. The solution is then constructed cooperatively by all particles in the subswarm. When a solution is split into its components, and each component is optimized separately, a *credit assignment problem* arises. Each particle needs to be assigned credit for the quality of its participation in the overall solution. Shared-space CCPSO deals with this problem by using the attractors fitness as a measure of the contribution of each possible value of this component to the overall solution encoded by the subswarm.

Basically, this cooperative strategy is based on the introduction of static particles, (i.e., attractors) which mediate local/global choices of the particles. Two competing forces guide particles through this decision process by only local interactions. On the one hand, attractors exert a force based on their strength value (an heuristic parameter value) that guides particles towards their local minimizers. On the other hand, the attractor's fitness value directs them towards regions of the space that, while might not be the best solu-

tion for a particular component, are favorable for the overall swarm. Particles initially move independently from each other, guided by the forces exerted by attractors. At the beginning of a simulation all attractors have the same fitness value; therefore, its attractive force depends only on their strength value (an heuristic parameter). Particles move in the direction of the attractor with the highest attractive force. This dynamic continues until a particle changes direction, this change of direction is the result of an increase of fitness value of an attractor. These changes propagate throughout the swarm via local interactions, until equilibrium is reached and the swarm reaches a consensus.

The simulation results presented in this chapter provide substantial support for the efficacy of this approach. Experiments show that the diagnosis problem can be solved by the model with reasonable accuracy, above 90% of exactly correct answers when full post-processing is used. This results are particularly noteworthy for a diagnosis problem, considering only local computations are involved.

As shown by the results, it is feasible to extend particle swarm optimization algorithms to solve combinatorial optimization problems. This communication mechanism was found to be vulnerable to some local optima, causing the swarm to be stuck in a near-optimal solution (typically one of the three best solutions). A resettling process, which forces the swarm to explores a distinct area of the space, allows shared-space CCPSO to escape this local optima. The next chapter of this dissertation extends shared-space CCPSO and introduces a more powerful feedback mechanism between solution components that does not need an explicit resettling process to escape local optima.

Chapter 4

Cooperative PSO for Combinatorial Problems with Independent Values

The research in this chapter explores how to solve a combinatorial optimization problem where each component has its own set of possible values, unlike the combinatorial problem presented in the previous chapter where all components had the same set of possible values. In the type of combinatorial problems explored here, candidate solutions are again represented by a vector, *solution* $= \langle\, c_1, c_2, ..., c_n \,\rangle$, where each component c_j is selected from an ordered set of k_i possible values $Q_j = \{q_{j1}, q_{j2}, ..., q_{jk}\}$ for position j. A typical combinatorial problem of this class is a scheduling problem. A scheduling problem can be seen as an assignment problem in which the solution components are the events to be scheduled, and the values assigned to events correspond to the time at which they should occur. This is a generalization of the problem presented in Chapter 2 where all components have the same set Q of possible values.

A core concept in shared-space CCPSO is the distance between particles and attractors, it serves as a measure of the contribution of an attractor (i.e., a particular value for a solution component) to the overall solution. The closer particles are to an attractor, the higher its fitness value. While this strategy avoids the explicit computation of a fitness value for solutions (which is usually a costly function) it is limited by the semantic value implicitly assigned to this measure. The feedback mechanism in shared-space CCPSO assumes that two particles move to the same attractor if this attractor has a value that is

favorable to both solution components represented by these particles. This is the case in combinatorial problems where all components share the same value, and therefore can reinforce each other, but for combinatorial problems where this distance does not have this semantic value, a new measure needs to be defined.

With this in mind, a new cooperative strategy to handle multi-valued combinatorial problems is presented in this chapter. The multi-space CCPSO model partitions the space by placing each particle in its own high-dimensional continuous space. As in the shared-space CCPSO each individual particle moves around the high dimensional space based solely on the forces exerted by attractors while being guided implicitly toward regions of the space that are more promising to the rest of the swarm. The difference lies in the feedback mechanism. In multi-space CCPSO a candidate solution is constructed cooperatively by decoding each particle's position into a choice for that solution component. Each particle has its own set of attractors and feedback to the attractors is provided by a strength value which considers the fitness of the candidate solution with the current component replaced by the choice represented by the attractor. Particles communicate through this candidate solution sol_s. This candidate solution sol_s of each subwarm is used (as in traditional PSO) to keep a memory of the best solution seen so far by the subswarm (cognitive memory b_s), and the best solution seen so far by a neighboring subswarm (social memory n_s). This memories b_s and n_s reflect a snapshot of the choices of other particles in the subswarm. They serve as a bellwether of the overall status of the subswarm. Each particle can now assess the quality of the contribution of its local choices to these overalls solutions.

Side-chain packing (SCP) is the combinatorial task selected to evaluate the performance of multi-space CCPSO. SCP is an important subproblem of the general protein structure prediction problem, in which the side chains of the protein residues are positioned on a fixed and given protein backbone. Specifically, it consists of selecting a side-chain conformation (from a discrete set of preferred conformations, known as rotamers [42]) for each side chain in the protein such that the the protein energy conformation is minimized. The reasons for the selection of this particular combinatorial problem are twofold. First, like abductive diagnosis, SCP deals with a highly non linear and complex evaluation function. Second, the specific values of each solution component (i.e., the set of possible rotamers for each side-chain position) is distinct, and the dimensionality from one component to the next can vary significantly. This is the key difference with the type of combinatorial problems studied under the previous model.

4.1 Multi-space CCPSO

The cooperative strategy presented in the previous chapter is extended here by allowing each particle to have its own set of attractors. Attractors will still mediate communication between particles, but instead of receiving direct feedback based on the position of neighboring particles now they assess their fitness based on the quality of their contribution to the fitness of the swarm solution. A *subswarm* is defined as a collection of particles, where each one contributes one solution component. Once again, each component c_j is represented by a particle p_j. Each possible value in Q_j is represented by an attractor r_{ji}. Particle p_j moves in a k_j-dimensional space where all its attractors r_{ji} are

located. An attractor tries to pull p_j to its position, i.e., tries to assign its value q_{ji} to c_j. A 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 q_{ji} represented by the attractor r_{ji} that is selected as winner for that particle. This winner is selected randomly using the particle's position, $\vec{x}_j$, as probability distribution. Thus, an attractor that is more successful in pulling particles towards its position will have a higher chance of being selected as part of the decoded solution.

While previous approaches [153][154] attempted to partition the search space and optimize each dimension separately, this was found to introduce pseudo-minima [154]. The strategy implemented in the multi-space CCPSO algorithm aims to solve the same problem and avoid pseudo-minima by keeping the concept of particles being guided by their own candidate solution templates that represent the social and cognitive memories. These cognitive and social 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.

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:

$$S_s = \langle \textit{particles}_s, \textit{sol}_s, \textit{fitness}_s, \vec{b}_s, \vec{n}_s, \textit{neighbors}_s \rangle$$

where $particles_s$ is the set of particles that belong to the subswarm, sol_s is the candidate solution decoded from $particles_s$, $fitness_s$ is the fitness value of the solution, $neighbors_s$ is a set of subswarms with which this subswarm interacts, $\vec{b}_s$ represents the cognitive memory of the subswarm, i.e., the candidate solution at which it achieved the best fitness value, and $\vec{n}_s$ keeps a social memory.

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

$$r_{sji} = \langle \vec{a}_{sji}, st^b_{sji}, st^n_{sji} \rangle \tag{4.1}$$

where $\vec{a}_{sji}$ = is the position of the attractor, st^b_{sji} represents the strength of this attractor to pull particle p_{sj} towards the best solution seen by subswarm s (i.e., $\vec{b}_s$), and similarly st^n_{sji} represents the strength of this attractor to pull particle p_{sj} towards $\vec{n}_s$.

The position $\vec{a}_{sji}$ is initialized to a unit vector in a direction of the i^{th} coordinate axis orthogonal to all previous attractors. An example with three attractors in depicted in Figure 4.1. At each iteration a new candidate solution $\vec{b}'_s$ is constructed by using $\vec{b}_s$ as a template candidate solution, and replacing the value of the component c_j with q_{ji}. The

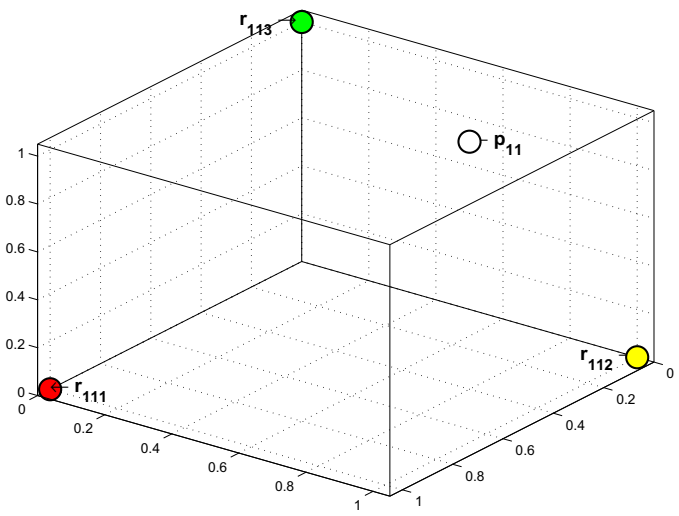


Figure 4.1: Example of a particle and its attractors. Particle p_{11} moves in a 3-dimensional space. Attractors are located in an orthogonal position to one another and try to pull p_{11} towards their locations.

strength value st^b_{sji} is computed proportional to the fitness value of $\vec{b}_s^{'}$. In particular,

$$st^b_{sji} \propto \textit{fitness}(\vec{b}_s^{'}), \text{ where } \vec{b}_s^{'} = \vec{b}_s \text{ with } b_s^{'}[j] = i \tag{4.2}$$

A similar calculation determines st^n_{sji}:

$$st^n_{sji} \propto \textit{fitness}(\vec{n}_s^{'}), \text{ where } \vec{n}_s^{'} = \vec{n}_s \text{ with } n_s^{'}[j] = i. \tag{4.3}$$

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. Equations 4.2–4.3 show how the choices of other particles affect the attractiveness of a particular attractor. This strength value is used as a heuristic to guide the particles towards better areas of the search space. A parameter τ determines the

probability of updating this strength value. At each iteration a biased coin with probability τ is flipped to determine if the strength values need to be updated.

Particles A particle represents one component in a 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 j of subswarm s is defined by the tuple:

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

where $\vec{x}_{sj}$ is its current position and $\vec{v}_{sj}$ is its current velocity vector. The particle's position $\vec{x}_{sj}$ represents the probability of component i taking each possible value. After normalizing $\vec{x}_{sj}$ to sum 1, $\vec{x}_{sj}[i]$ represents the probability of selecting value q_{ji} for component c_j. The selection is made at random using $\vec{x}_{sj}$ as the probability distribution. The initial position $\vec{x}_{sj}$ is set to a random position in the k_i-dimensional space, and its initial velocity vector $\vec{v}_{sj}$ is set to zero. The updates of the particles are accomplished according to Equations (4.4) – (4.5). As with the traditional PSO (refer to Algorithm 4), Equation (4.4) 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 $\vec{v}^b_{sj}$), and a velocity component that drives the particle towards the location of the best neighborhood solution (i.e., a social-velocity $\vec{v}^n_{sj}$). Then each particle's position is updated by simply

adding the velocity vector (Equation (4.5)). Formally:

$$\vec{v}_{sj} \leftarrow \omega \times \vec{v}_{sj} + U(0, \gamma_1) \times \vec{v}^{b}_{sj} + U(0, \gamma_2) \times \vec{v}^{n}_{sj} \tag{4.4}$$

$$\vec{x}_{sj} \leftarrow \vec{x}_{sj} + \vec{v}_{sj} \tag{4.5}$$

where again $U(a, b)$ is a randomly chosen value in (a, b). The value of the cognitive-velocity component $\vec{v}^{b}_{sj}$ is the result of all forces exerted by nearby attractors on particle p_j as shown below in Equation (4.6). Equation (4.6) below shows that this attraction force is dependent upon particular attributes of each attractor: its strength (st^{b}_{sji}), 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 these are called *cognitive-attractors*. The 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 (4.7). Formally, the velocities are computed as:

$$\vec{v}^{b}_{sj} \leftarrow \sum_{j} st^{b}_{sji} \times \vec{a}_{sji} \tag{4.6}$$

$$\vec{v}^{n}_{sj} \leftarrow \sum_{j} st^{n}_{sji} \times \vec{a}_{sji}. \tag{4.7}$$

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

Algorithm 6 Multi-space CCPSO algorithm

for each subswarm s **do**
  for each element $\in$ *Input* **do**
    create particle j
    initialize position $\vec{x}_{sj}$
    initialize velocity $\vec{v}_{sj}$
    for each q_{ji} in Q_j **do**
      create attractor i
      initialize position $\vec{a}_{sji}$
    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 subswarm s
    $\vec{n}_s \leftarrow$ best solution found so far by *neighbors* of subswarm s
    for each particle j **do**
      for each attractor i **do**
        if *flip*(τ)
        $st^b_{sji} \propto$ *fitness*$(\vec{b}'_s)$, where $\vec{b}'_s = \vec{b}_s$ with $b'_s[j] = i$
        $st^n_{sji} \propto$ *fitness*$(\vec{n}'_s)$, where $\vec{n}'_s = \vec{n}_s$ with $n'_s[j] = i$
        end if
      end for
      $\vec{v}^b_{sj} \leftarrow \sum_j st^b_{sji} \times \vec{a}_{sji}$
      $\vec{v}^n_{sj} \leftarrow \sum_j st^n_{sji} \times \vec{a}_{sji}$
      $\vec{v}_{sj} \leftarrow \omega \times \vec{v}_{sj} + U(0, \gamma_1) \times (\vec{v}^b_{sj}) + U(0, \gamma_2) \times (\vec{v}^n_{sj})$
      $\vec{x}_{sj} \leftarrow \vec{x}_{sj} + \vec{v}_{sj}$
    end for
  end for
end while

Algorithm Algorithm 6 presents the multi-space 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

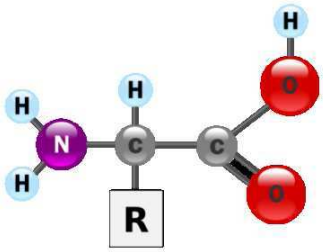


Figure 4.2: General structure of an amino acid. Each amino acid has its own distinctive side chain (R group)

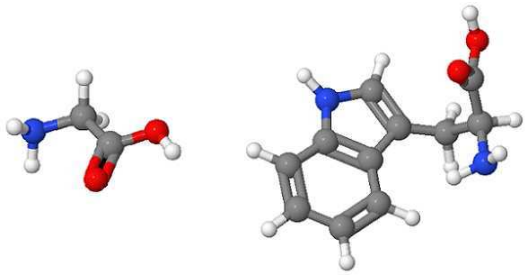


Figure 4.3: Example of two amino acids: Glycine (left) and Tryptophan (right)

(Equation (4.5)). This new velocity vector is computed based on the influence exerted by the cognitive and social attractors. Equations (4.6) - (4.7) define how to get the resultant force of all attractors, and Equations (4.2) - (4.3) show how 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 is reached.

Multi-space CCPSO is validated in an important problem in the computational analysis of protein structure: side-chain packing. This problem is described below.

4.2 Application: Side-chain Packing

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* and it has a distinct *side chain*. A depiction of the general structure of an amino acid is presented in Figure 4.2. Two amino acids are depicted in Figure 4.3. Notice the difference in size between Glycine (on the left) and Tryptophan (on the right). Just as the letters of the

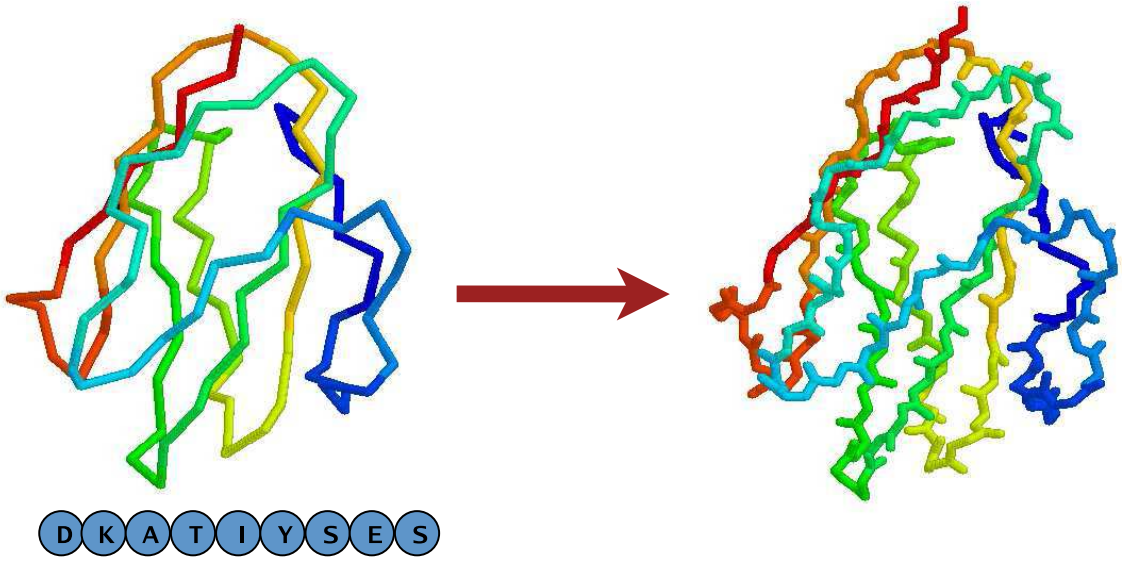


Figure 4.4: Protein side-chain packing task: Given backbone coordinates and the amino acid sequence, model the tertiary structure of the protein

alphabet can be combined to form many different words, the 20 standard amino acids can be linked in varying sequences to form a vast variety of proteins. A protein's sequence of amino acids, called its *primary structure*, determines the 3-D structure of a protein. This structure, which includes both the protein backbone and the conformations of the side chains of the amino acids, 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 [87]. 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. This is depicted in Figure 4.4. The problem of predicting side-chain conformations for each residue given a backbone structure is of central im-

portance in homology modeling and the design of novel protein sequences. Homology modeling is a technique for predicting protein structures from their amino acid sequences that it is based on the assumption that two proteins with similar sequences will have similar tertiary 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 [121]. 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* [123]. The restriction to these discrete configurations implies an important problem reduction. Nevertheless, even this discrete version is known to be NP-hard [120] and hard to approximate [21]. 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.

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 [28]. 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., [22][79][143].

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:

$$E = E_{backbone} + \sum_{i} E(i_r) + \sum_{i<j} E(i_r, j_s) \tag{4.8}$$

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 (4.8) is minimized. Search algorithms for side-chain packing fall into two broad categories: stochastic and deterministic. Stochastic algorithms, including simulated annealing [27] and genetic algorithms [34], semi-randomly sample sequence-structure space and move toward lower energy solutions, while deterministic algorithms, such as dead-end elim-

ination and its extensions [31][35][162], integer programming [2][75], and other graph search approaches [19][163] perform semi-exhaustive searches [121]. 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 $E(i_r, j_s)$. When the interaction energy of rotamer r of residue i and rotamer s of residue j is below a predetermined threshold the rotamers are consider to not interact and there is no edge between the nodes. Note that 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 4.5 shows an example of a SCP graph with three amino acid residues A_1, A_2, and A_3.

4.3 Multi-Space CCPSO for Side-chain Packing

In SCP, an optimal solution is defined as a rotamer assignment with the minimal energy conformation. A solution component is a rotamer for a particular residue, and a candidate solution is an assignment of one rotamer for each residue. In multi-space CCPSO each particle represents a residue and attractors represent the rotamers of

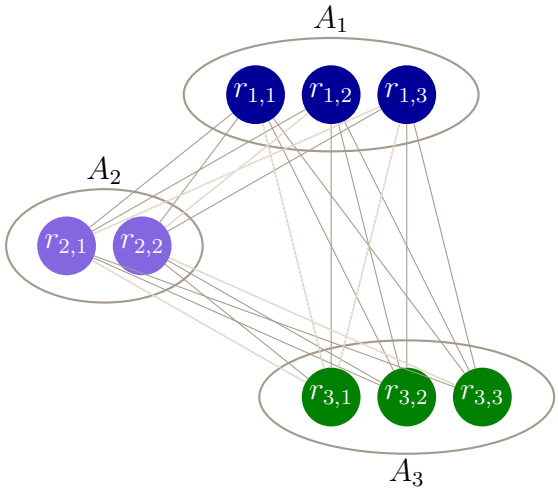


Figure 4.5: An example SCP problem with three amino acid residues A_1, A_2, and A_3. Each amino acid residue is represented by a subgraph that contains all the rotamers that belong to that amino acid, $A_1 = \{R_{1,1}, R_{1,2}, R_{1,3}\}$, $A_2 = \{R_{2,1}, R_{2,2}\}$, $A_3 = \{R_{3,1}, R_{3,2}, R_{3,3}\}$. A possible solution to this SCP is $sol = \langle R_{1,1}, R_{2,2}, R_{3,2}\rangle$.

each residue. In the SCP of Figure 4.5, there are 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, $particles_1$=$\{p_{11}, p_{12}, p_{13}\}$. For residue A_1, a p_{11} that would move in a continuous k_1-dimensional space is created, where k_1 is the number of rotamer choices. Since $k_1 = 3$ for A_1, particle p_{11} 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 4.1.

In a candidate solution for a SCP problem, each component c_j 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_{sj} 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 se-

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

$x_1 \leftarrow$ | $R_{1,1}$ 0.2 | $R_{1,2}$ 0.3 | $R_{1,3}$ 0.5 | $winner(x_1) = R_{1,3}$

$x_2 \leftarrow$ | $R_{2,1}$ 0.7 | $R_{2,2}$ 0.3 | $winner(x_2) = R_{2,1}$

$x_3 \leftarrow$ | $R_{3,1}$ 0.3 | $R_{3,2}$ 0.6 | $R_{3,3}$ 0.1 | $winner(x_3) = R_{3,2}$

obtains the candidate solution

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

Figure 4.6: Solution representation for the SCP problem. A solution sol_1 is obtained by selecting a rotamer from each particle using x_i values as probability distribution.

lected randomly by function *winner* which uses the particle's position $\vec{x}_j$ as distribution:

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

An example of a solution for the SCP defined in Figure 4.5 obtained from a subswarm s is depicted in Figure 4.6. For clarity, the index s of the particles is omitted. Once a solution sol_s for each subswarm s is obtained, computing its fitness value is straightforward. The fitness value of a solution sol_s is computed using:

$$fitness(sol_s) = \frac{1}{energy(sol_s)} \tag{4.9}$$

4.4 Validating multi-space CCPSO on SCP problems

Multi-space CCPSO is evaluated on combinatorial problems resulting from three protein side-chain 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, the same rotamer library and energy function that a previous study used [75] is employed. The rotamer library used is Dunbrack's backbone-dependent rotamer library [42]. For each 10° range of ϕ, ψ backbone angles, this library has 320 rotamers, with the largest number of rotamers, 81, belonging to the amino acids arginine and lysine.

The energy function of the rotamer conformation is computed using Equation (4.8), except that the $E_{backbone}$ term is dropped since the template backbone is fixed. To compute the self-energy terms $E(i_r)$ and the pairwise rotamer energy $E(i_r, j_s)$ the energy functions defined in [75] are used. These energy functions are derived from the AMBER force field [20]. $E(i_r)$, the self-energy of rotamer i, is computed using both a statistical potential and a van der Waals interaction term. 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.

Table 4.1: Multi-space CCPSO parameters for SCP.

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

More details concerning the energy function can be found in [75].

Results are compared to the provably optimal solution obtained by a deterministic method as reported in [75]. The accuracy of multi-space CCPSO results is determined by computing the percentage error. Percentage error is defined as:

$$\textit{error} = 100 \times \frac{|\textit{OPT} - \textit{solution-energy}|}{|\textit{OPT}|} \tag{4.10}$$

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.

Since the method is stochastic, each instance is executed twenty times and the average percentage error is reported, 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.

CCPSO parameters are listed in Table 4.1. These parameters were selected after some preliminary trials.

4.5 Results

The performance of multi-space CCPSO on the three datasets are presented in Tables 4.2 - 4.4. 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. The fourth column points out the energy of the optimal configuration. The 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.

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 4.2 it can be seen that among the 27 proteins with their native backbones, 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

Table 4.2: Results of multi-space CCPSO 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

error $< 1.12\%$. Details are shown in Table 4.3. Again, this indicates that the CCPSO approach can successfully find optimal or near-optimal solutions.

The design dataset contains 25 instances. The goal of 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

Table 4.3: Results of multi-space CCPSO 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

significantly larger. As in [75], 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, each amino acid position in the backbone can 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 4.4. Among the 25 design problems, the CCPSO is able to find the optimal configuration for 10 instances. This indicates that, not surprisingly, the design problem is more difficult to solve for the CCPSO than fitting

Table 4.4: Results of multi-space CCPSO 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

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, the running times of the native and design dataset are compared. 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 number of rotamers per residue. This makes CCPSO particularly appealing for the protein design task as well as side-chain packing with detailed rotamer libraries.

4.6 Discussion

This chapter presents a novel cooperative combinatorial PSO scheme that partitions the combinatorial search space into individual components that are optimized individually. The multi-space CCPSO algorithm preserves the concept of particles that move throughout a continuous high-dimensional space while being guided by its own cognitive and social memories of candidate solutions. The effectiveness of this new combinatorial PSO is evaluated by three side-chain packing tasks, with varying degree of difficulty. Results show that multi-space CCPSO is able to obtain optimal or near-optimal solution in all benchmarks tested. This is the first time that a PSO-based algorithm has been used to solve the side-chain packing problem. Furthermore, multi-space CCPSO's performance, while affected by the number of residues, seems not as susceptible to an increase in ro-

tamer size as other traditional side-chain packing methods, which makes it an attractive method for protein design.

The majority of previous implementations of combinatorial PSO are used in combination with a dedicated local search algorithm such as simulated annealing or fast local search [24] whereas the method presented here does not need any local search algorithm to be able to deal with combinatorial problems.

Equations 4.2–4.3 show how the choices of other particles in the subswarm affect the attractiveness of a particular attractor. These choices are reflected in the cognitive and social memory ($\vec{b}_s$ and $\vec{n}_s$) which are used as a template candidate solution to compute the attractor's fitness value. The attractor's strength value is estimated based on this fitness value. Note that this differs from simple local search algorithms such as the simple iterative improvement algorithm (SIIA) presented in section 2.1.2. In SIAA, at each iteration a new solution is created (independently of previous solutions) and its acceptance is based on its relative fitness with respect to the fitness of the current solution. In CCPSO the attractor's strength value is just one part of the interaction dynamics that define the particle's new position. This new particle's position is then employed to construct a new solution.

The component-by-component optimization of multi-space CCPSO allows fine tuning of each component by each particle when the candidate solution templates (i.e., cognitive and social memories) do not change from one iteration to the next and explores a different part of the space when the memories are modified. This strategy allows it to implicitly balance exploration/explotaition and explore more effectively the search space.

Chapter 5

Multimodal Combinatorial Search

In general, optimization problems are formalized by identifying the function to optimize. When the function to optimize has one global optimum the function is said to be *unimodal*. On the other hand, when such a function presents multiple global optima, or local optima whose values are very close to the global one, it is said to be *multimodal*. Formally, unimodal problems have a single global optimum, $f(x^*) \leq f(X) \forall_{x \in \mathbb{R}^n}$, where $f(X) : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function and n is the dimension of the search space. Multimodal problems have more than one optimum. These optima may all be global optima, or a mixture of global and local optima. A *local optimum*, x_L^*, is subject to: $f(x_L^*) \leq f(X) \forall_{x \in L}$, where $L \subset \mathbb{R}^n$. A *near-optimal optimum*, a x_{OPT}^*, is a special type of local optima where $f(x_{OPT}^*)$ is within ϵ times the value of an optimal solution $f(x^*)$. ϵ is a parameter that specifies the acceptable distance between the global optimum and the near-optimal value. This parameter is problem-dependent.

Multimodal function optimization has been addressed extensively in the recent literature [18][109][110][112]. The traditional class of problems related to this topic focused on developing algorithms to find either the global optimum or all the global optima of the given multimodal function, while avoiding local optima. It is important to make a clear distinction between distinct multimodal scenarios created by mixtures of global and local optima, since very different problems arise in each of these cases. Three typical scenarios

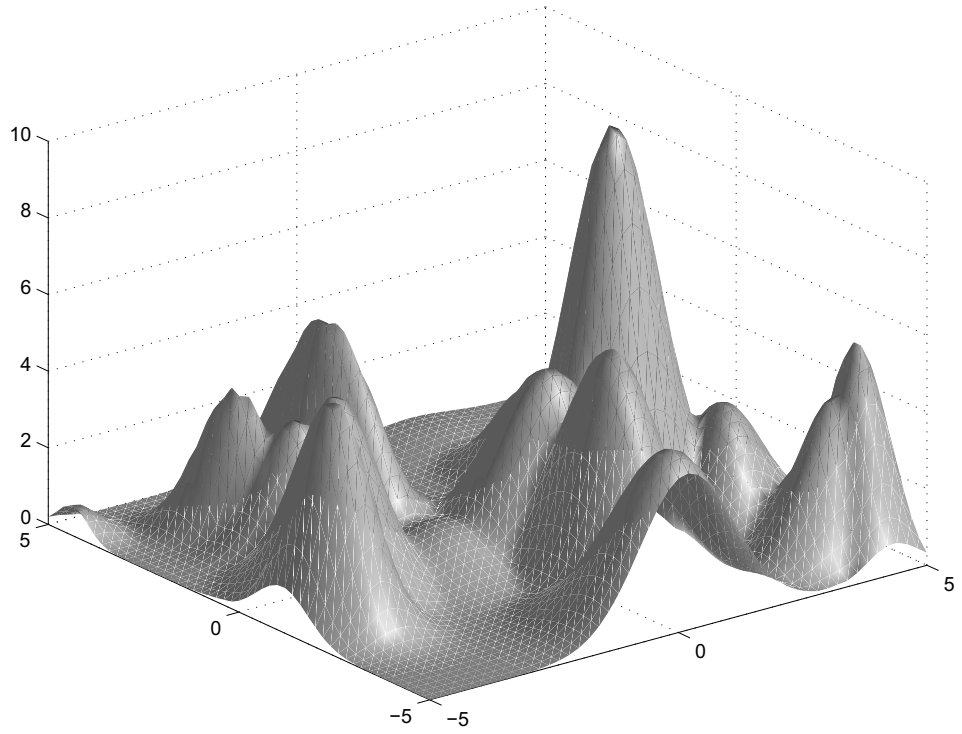


Figure 5.1: Multimodal problem with one global optimum and local optima

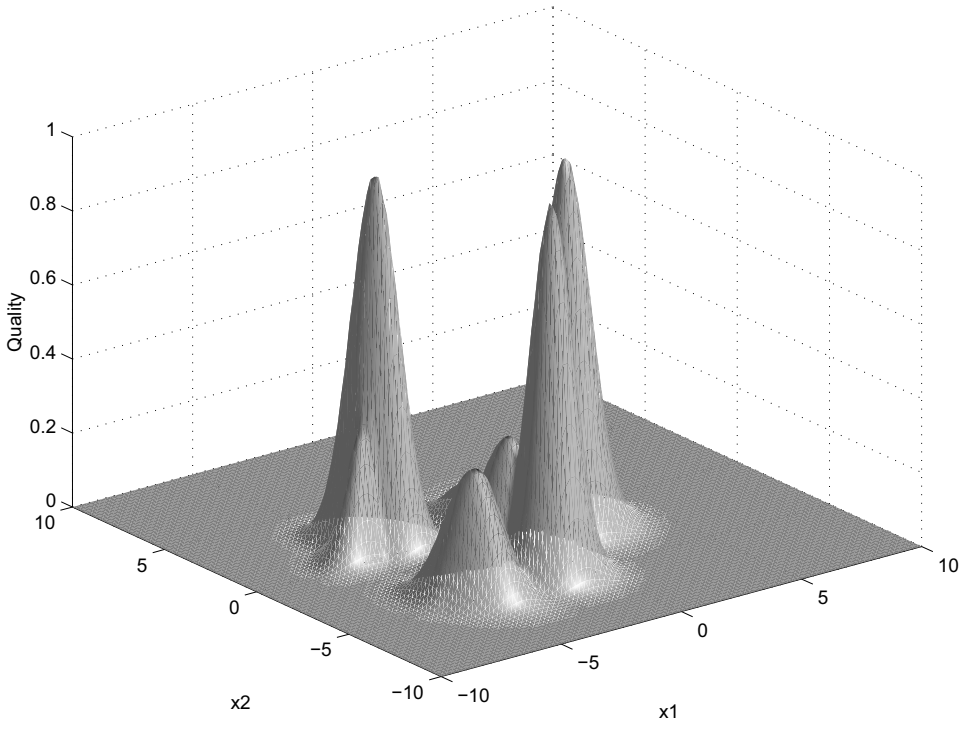


Figure 5.2: Multimodal problem with multiple global optima

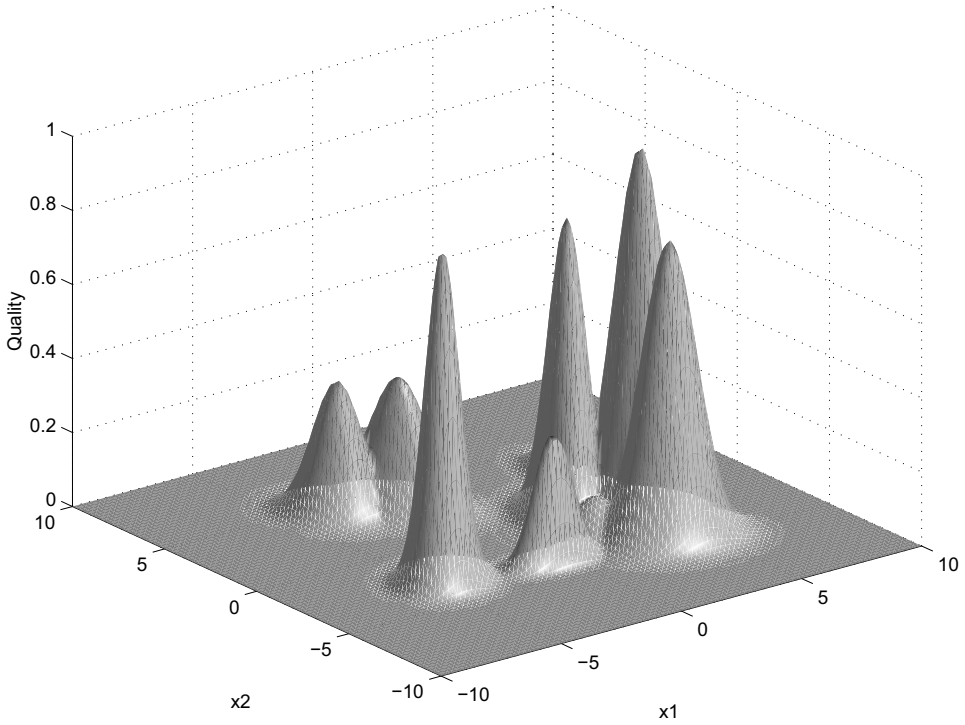


Figure 5.3: Multimodal problem with multiple global optima, near-optima and local optima

are depicted in Figure 5.1–Figure 5.3. Figure 5.1 depicts a multimodal problem when there is only one global optima in the landscape surrounded by local optima. A variety of dedicated algorithms have been developed to increase the probability of locating the global optimum, or at least a very good local optimum. Many of these algorithms are based on the premise that if an optimization algorithm is forced to search in more areas of the search space, then it has higher probability to escape local optima and find the global optimum. Several different PSO algorithms have been proposed with this idea in mind [49][88]. These algorithms are mostly based on existing approaches used in evolutionary algorithms for multimodal optimization [32][56].

Figure 5.2 and Figure 5.3 show multimodal optimization problems with multiple optima having either equal or almost equal function values. These optimization problems

are different from the problem of finding only the global optimum of a multimodal function. In problems with multiple global optima an optimization algorithm should ideally find a set of all the optimal and near-optimal solutions. Figure 5.2 shows an idealized scenario with only global optima and local optima. Figure 5.3, on the other hand, presents a more realistic situation where the task is not only to find multiple global optima, but to distinguish near-optimal from local optima.

Like other meta-heuristics, PSO is usually designed for the goal of finding a single optimal solution for a given problem. However, many scientific and engineering optimization problems have convoluted search spaces with large numbers of optima. Typical PSO algorithms usually assume that there exists only a single best solution in the search space, and they put efforts into isolating it from other spurious solutions. PSO does not guarantee to find the global optimum, thus the solution found could just be a local minimum with no indication of its relative fitness with respect to the optimum. The goal in this class of problems is to avoid deception by the local optima of the multimodal function, which would otherwise result in premature convergence to a suboptimal solution. Multimodality in a search and optimization problem gives rise to several attractors and thereby presents a challenge to any optimization algorithm in terms of finding global optimum solutions [56]. However, the problem is compounded when multiple (global and near-optimal) optima are sought. The knowledge of multiple local and global optima has several advantages such as obtaining an insight into the function landscape and selecting an alternative solution when the dynamic nature of the constraints in the search space makes a previous optimum solution infeasible to implement. These set of solutions can

be used in many different ways, depending on the application field: a specific solution can be selected from the set using a more accurate evaluation function, different solutions can be preferred in different situations, or a combination of multiple solutions can be built. In this case, the goal of the PSO algorithm may be to locate all of them, and standard techniques will usually either favor a single solution, or get confused by the multiple possible solutions and fail to converge to a single one.

An algorithm that looks for a single optimum would arbitrarily pick just one of the optimal solutions, or could be misled by the presence of more than one single optimum and fail to converge. The probability that the conventional PSO algorithm will converge to a sub-optimal position is unacceptably high [104]. Thus, it is necessary either to design specific algorithms, or to modify the generic ones in order to optimize multimodal functions [113].

The research in this chapter explores how to solve combinatorial optimization problem of many optima with the same, or very similar, quality value. In particular, this chapter explores the ability of multi-space CCPSO used in tandem with explicit diversity strategies to discover sets of high-quality and diverse solutions. This idea has been pursued in numerical optimization in several PSO variants [12][17][89][137], but no explicit PSO has been developed to handle multimodal combinatorial problems. Note that even the definition of local optima, i.e., points that are better than all their neighborhood solutions requires both a set of solution points and a neighborhood structure. This is obvious for real-valued problems, but for combinatorial problems the number of local optima depends on the choice of neighborhood structure, i.e., on the velocity operators

used. Therefore, it is of particular interest to analyze how a diversity strategy, aimed at numerical optimization problems, performs in the multi-space CCPSO framework.

Protein design (briefly described in the previous chapter) is a particularly suitable task to validate this model. Its combinatorial complexity along with the highly non-linear energy interactions makes the fitness landscape impressively complex with an extremely large number of possibilities [107]. Protein design is an extension of side-chain packing, with the non-trivial difference that the number of rotamers to consider for each position varies greatly, which makes the search space significantly larger and more complex. Chapter 4 presented the application of multi-space CCPSO to side-chain packing, including the partial redesign of 22 proteins. This chapter goes further into the protein design problem, by focusing on sampling the conformational space and obtaining an ensemble of distinct low-energy side chain conformations, instead of one optimal conformation.

The chapter starts by presenting the new multimodal CCPSO architecture. This includes a description of the diversity strategies implemented, the metric used to measure distance between solutions and the multimodal CCPSO algorithm which integrates all these elements. The next section describes protein design, highlighting its difference to protein structure prediction, and why it motivates the interest in sets of high-diversity low-energy solutions. Finally, all multimodal CCPSO variants are evaluated in the sequence redesign of a small protein, a $\beta\beta\alpha$ motif of the zinc finger DNA binding module [114], along with the results of similar redesign in two in silico proteins. All proteins are redesigned using three different design schemes, which vary the number of amino acids allowed per position. This assesses the ability of multimodal CCPSO to produce an en-

semble of high-quality distinct conformations in three very different fitness landscapes.

5.1 Multimodal CCPSO

5.1.1 Architecture

A modular methodology for multimodal combinatorial optimization problems is presented in multimodal CCPSO. This new PSO extension tries to find a set of optimal and near-optimal solutions by forcing multi-space CCPSO to sample different regions of the search space. Every time multi-space CCPSO detects a solution of high quality (i.e., a potential optimum) it "marks" the region in the search space by storing this solution in an external memory. The external memory is then used by diversity strategies to encourage particles to search different regions of the search space. The algorithm stops after a given number of iterations and at the end the archive contains the detected optima. This new algorithm consists of four main components: 1) the multi-space CCPSO algorithm to solve the combinatorial problem, 2) an external memory (archive) to store the ensemble of solutions, 3) a diversity strategy to encourage sampling of the search space, and 4) a strategy to decide which solutions should be stored. These main components and their interactions are depicted in Figure 5.4. In the following, each one of these components is presented in detail.

Ensemble Memory Depicted as Box (A) in Figure 5.4. CCPSO is augmented with an external archive called *Ensemble Memory* that holds selected past solutions. Ensemble

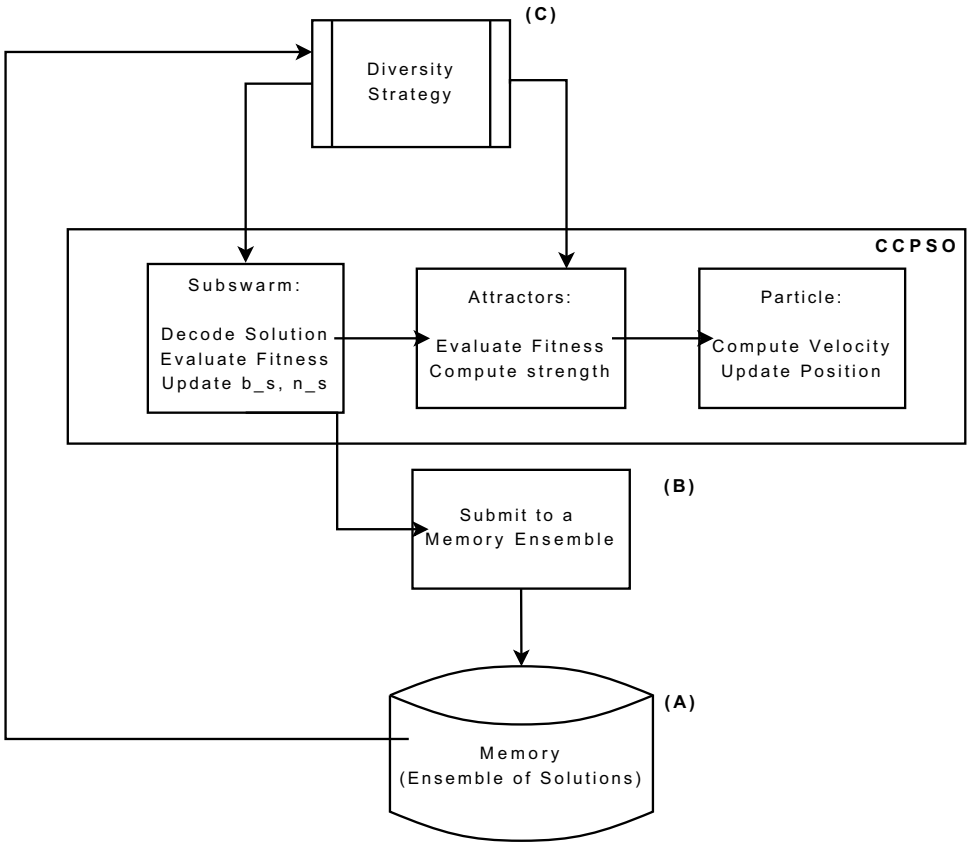


Figure 5.4: Multimodal CCPSO components

memory, *EM*, is defined by the tuple:

$$EM = \langle solutions_s, sol_{best}, fitness_{best}, threshold \rangle$$

where $solutions_s$ is the archive where local optima are stored, sol_{best} is the best solution seen so far, $fitness_{best}$ is the best local minimum found so far, and *thereshold* is a parameter used to define the acceptable energy range of solutions to be stored. Solutions are added to *EM* by the *submission of solutions* strategy described below. If solution with a energy value better than $fitness_{best}$ is added to the ensemble then *EM* is updated, and all solutions that are not within *threshold* of the new $fitness_{best}$ are discarded from the ensemble.

Submission of solutions strategy Depicted as Box (B) in Figure 5.4. Every time a potential optimum is found, a decision needs to be made about keeping or discarding this potential optimum. This strategy uses the *threshold* and $\mathit{fitness}_{best}$ value of *EM* to decide if a solution is "good" enough to be recorded in the ensemble memory and the fitness landscape should be modified to guide the rest of the subswarms away from this solution. Multiple criteria can be used to determine if a potential optimum has been found. This criteria can be fixed: the algorithm executes for a maximum number of iterations or fitness functions evaluations, or adaptive by detecting when particles are no longer moving (i.e., the system is in equilibrium). The average velocity update over all particles or the number of iterations since the last fitness change can be used as measure to detect when particles are no longer moving. In multimodal CCPSO a solution x is considered a potential optimum x^* when the following two criteria apply:

- $f(x)$ has not changed in the last k iterations
- $f(x) \leq f(x')\forall_{x' \in swarm}$

It is reasonable to suppose that a potential optimum can be a local optimum or can be very close to a local optimum. To avoid overloading the ensemble memory with local optima, a solution x^* found by multi-space CCPSO is added to the archive if and only if one of the following situations arrive:

- x^* has a quality value within *threshold* of $\mathit{fitness}_{best}$ and it is distinct from any previously found solution
- x^* is better than any solution x in the archive

Regardless if the solution was stored or discarded, the subswarm that submits this solution to the ensemble is restarted to a random position to encourage it to explore a new part of the space.

5.1.2 Diversity Strategies

Diversity strategies are incorporated in Box (C) in Figure 5.4. These strategies are the mechanism to force multimodal CCPSO to move away from promising regions of the search space that have been already explored. A diversity strategy uses the previously located optima stored in the ensemble memory to penalize any subswarm that moves closed to a previously explored region. This penalization takes into account the distance between the subswarm solution and the memory of the best solution found so far and adjusts solution fitness accordingly. The new fitness value $\textit{fitness'}_s$ is computed with Equation (5.1):

$$\textit{fitness'}_s = h(\textit{fitness}_s,\ \textit{distance}(sol_{best}, sol_s)) \tag{5.1}$$

where each diversity strategy defines its particular function h and $\textit{distance}(sol_{best}, sol_s)$ is a metric used to assess the distance between two solutions and it is problem-dependent. If $\textit{distance}(sol_{best}, sol_s) = 0$, i.e., if the solution is identical to the best solution seen so far, then it is discarded and no adjusted fitness value is computed.

Simple Sequential Niching (SSN) is the simplest way to locate multiple solutions. It consists of repeatedly executing the particle swarm optimization algorithm, each time with a different initial swarm, and to record the solution found [44]. The hope is that

different solutions will be located, but there is no such guarantee. Since this method does not explicitly penalize solutions that are close to previously found solutions, the fitness of solution is left unchanged, as seen in Equation (5.2):

$$h_{SSN}(\mathit{fitness}_s,\ \mathit{distance}(\mathit{sol}_{best}, \mathit{sol}_s)) = \mathit{fitness}_s \tag{5.2}$$

Distance Penalty (DP) A well-know diversity technique is to change the objective function each time a solution is found, to penalize particles that move towards already located optima [70]. The penalty function is presented in Equation (5.3):

$$\begin{aligned} h_{DP}(\mathit{fitness}_s, \quad & \mathit{distance}(\mathit{sol}_{best}, \mathit{sol}_s)) = \\ & \mathit{fitness}_s + \lambda \times \mathit{fitness}_s \times (1 - \mathit{distance}(\mathit{sol}_{best}, \mathit{sol}_s)) \end{aligned} \tag{5.3}$$

where $\mathit{distance}(\mathit{sol}_{best} - \mathit{sol}_s)$ is the distance between subswarm s and the best local minimum found so far, sol_{best}. The constant λ is a scaling factor for the distance penalty. The success of this penalty method depends on the value of λ. If λ is too small, the penalty may no have effect at all, and particles may still move towards previously found solutions. If λ is too large, the penalty becomes too strong, and it might repel particles from any good optima that is relatively close to a found minimum. The value of λ therefore should be optimized depending of the balance between fitness value and distance between solutions desired.

Deflation strategy (DS) An alternative approach to repel particles from already discovered minima, referred to as deflation strategy (DS) is presented in [110][111]. In this case

the derating function for the fitness function is as follows:

$$h_{DS}(\mathit{fitness}_s,\ \mathit{distance}(\mathit{sol}_{best}, \mathit{sol}_s)) = \mathit{fitness}_s + \frac{\mathit{fitness}_s}{\mathit{distance}(\mathit{sol}_{best}, \mathit{sol}_s)} \tag{5.4}$$

Note that both DP and DS change the landscape of the objective function and this is not without problems, since they can introduce false local minima.

Fitness Sharing (FS) Fitness sharing algorithms are free from the undesirable side effect of introducing false local minima. The main idea of FS is to distribute particles along different areas of the fitness landscape [32][56]. When a subswarm s is sharing resources with other subswarms (i.e., when it is close to other subswarms), its fitness $\mathit{fitness}_s$ is degraded in proportion to the number and closeness of the subswarms that surround it. The equation for the adjusted fitness is defined in Equation (5.5) and (5.6)

$$h_{FS}(\mathit{fitness}_s,\ \mathit{distance}(\mathit{sol}_{best}, \mathit{sol}_s)) = \frac{\mathit{fitness}_s}{\sum_{j=0}^{n} \mathit{sharing}_s^j} \tag{5.5}$$

where n is the number of subswarms.

$$\mathit{sharing}_s^j = \begin{cases} 1 - (\frac{\mathit{distance}(\mathit{sol}_j, \mathit{sol}_s)}{\sigma_{\mathit{share}}}) & \text{if } \mathit{distance}(\mathit{sol}_j, \mathit{sol}_s) < \sigma_{\mathit{share}} \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

σ_{share} is a parameter that needs to be adjusted and represents the desired distance between particles.

5.1.3 Algorithm

Diversity strategies can be partitioned into two main classes: iterative and subpopulation methods. In iterative methods, the algorithms are applied several times consecutively to locate each optimum. In the subpopulation methods, the population is divided into parts to search optima simultaneously. Both of these type of diversity strategies are implemented in multimodal CCPSO. Specifically, from the iterative class: simple sequential niching [44], distance penalty [44] and deflation penalty [110][111] are selected; and from the subpopulation class: the popular fitness sharing technique [32][56].

For the iterative techniques, in order to allow as much parallelism as possible, instead of sequentially executing multi-space CCPSO and recording only the best solution found, multimodal CCPSO continually adds any local optima found to *EM* until a termination criteria is met (e.g., maximum number of iterations, or maximum number of solutions in *EM*). Every time a local optima is found, the submission of solutions strategy decides if the solution is good enough to be recorded and if the fitness landscape needs to be updated to guide particles away from this new found local optimum.

In the subpopulation algorithms the population is divided into parts to search for optima simultaneously. This means that the population needs to create and maintain niches that repel other individuals from the location of the niche. This is a successful strategy when the goal is to force the algorithm to explore different areas of the space as an strategy to escape local optima, but it becomes highly inefficient when the goal is to explore as many areas as possible in search of distinct local optima. Therefore, the fitness sharing strategy is modified as follows: instead of creating niches and having to maintain them by

using the PSO population, the solutions stored in the ensemble memory are used as members of existing niches. Every element in the ensemble memory *EM* is treated as an actual member of a current niche, and any subswarm that is near to it, is penalized accordingly. The pseudocode of the multimodal CCPSO algorithm is presented in Algorithm 7. For clarity, the details of multi-space CCPSO that were described in Chapter 4 are omitted. The lines marked with * in the algorithm, compute the adjusted fitness value by using a specific diversity strategy.

Algorithm 7 Multimodal CCPSO algorithm

```
Initialize EM
for each subswarm s do
    Create particles and attractors
end for
while stop criteria not met do
    for each subswarm s do
        decode solution sol_s
        evaluate fitness(sol_s)
        fitness'(sol_s) = h(fitness_s, distance(sol_best, sol_s)) *
        Update b_s, n_s
        if b_s is local optima then
            submit b_s to EM
        end if
        for each particle i do
            For each attractor j compute fitness
            For each attractor j update fitness' = h(fitness_s, distance(sol_best, sol_s)) *
            For each attractor j compute strength
            Update particle's velocity
            Update particle's position
        end for
    end for
end while
```

5.2 Application: Ensembles of Conformations for Protein Design

Protein design aims at selecting the optimal protein sequence for a desired structure. The ability to design proteins that have specific structures and functions will be very valuable to future protein drug discovery. Even now, protein design technology has been successfully applied to stabilize proteins, increase protein-protein binding affinity and create new protein structures [90]. Compared to protein folding, which starts from the sequence with the goal to predict the corresponding three dimensional structure, protein design starts with a target structure (represented by a protein backbone) and searches for sequences that minimize the energy conformation of that backbone. This distinction is depicted in Figure 5.5.

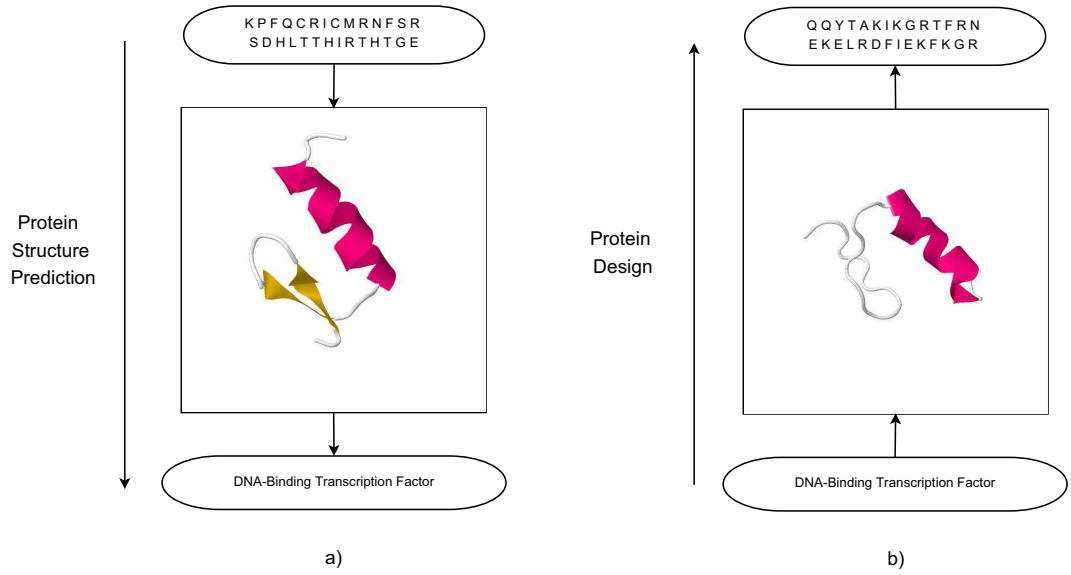


Figure 5.5: a) The protein-fold prediction task is to predicting a protein structure (and ultimately its function) from its sequence. Protein design, on the other hand, starts from the desired protein function which defines the structural elements needed, and then chooses a sequence that will fold into a structure consistent with those elements.

Computational methods play a central role in the rational design of novel proteins.

Developments have been made both toward grand challenges and toward more immediate practical applications. Examples of grand challenges are *de novo* design problems such as the creation of a novel protein fold, or a new binding interface, whereas more immediate practical applications involve the redesign of existing proteins to alter some specific property. One remarkable example of an early attempt at computational protein design was the complete redesign of a zinc finger protein [114] with a fixed backbone [27]. The fixed backbone assumption is incompatible with de novo design because there is no design template available. When a template backbone is available, such as the case of protein redesign, this greatly reduces computational time. Nevertheless, even with a fixed backbone, the protein design problem remains NP-hard [75]. Small proteins can potentially adopt an enormous number of conformations and can encode a large number of sequences. For example, the protein analyzed in this chapter with only 28 residues can have more than 10^{70} conformations. This combinatorial complexity along with the highly non-linear energy interactions make the energy landscape impressively complex with practically infinite possibilities [107]. In fact, the extensive amount of structural diversity accessible in protein design has created a demand for computational resources that can evaluate a multitude of candidate structures. Traditionally, the protein design problem aimed to obtain the global minimum energy conformation (GMEC). However, predicting a set of low-energy sequences is appealing for various reasons [47][51][139]. One of these reasons is that the energy landscape is not modeled with sufficient rigor. There is imprecise knowledge of many aspects of the physical forces in energy functions and rotamers are discretized to reduce combinatorial complexity. Moreover, even if

the lowest-energy conformation could be predicted accurately, this does not guarantee its biological feasibility. Therefore, its more attractive to provide a set of low-scoring conformations that can be reranked by more physically realistic (and computationally more expensive) energy functions, or tested in the wet lab.

In order to avoid rejecting potentially good solutions and to maximize the conformational diversity of the search, multi-space CCPSO has been augmented with the multimodal search techniques described above. The goal is not only to have an ensemble of rotamer conformations with near-optimal energy values, but that these conformations are as distinct from each other as possible. Conformations that are of special interest in protein design are the ones with energy value very close to the energy value of the known optima but with a very low similarity with this optimal conformation.

5.3 Comparing Diversity Strategies for Protein Redesign

The design methodology consists of starting with a protein backbone and attempt to design an amino acid sequence that best fits the given backbone. The method consists of applying multimodal CCPSO to efficiently sample all possible amino acid sequences and within them, to sample the rotamer conformations for local optimal solutions with energy value close to the global minimum. To evaluate the performance of multimodal CCPSO in different fitness landscapes, the four different diversity strategies are evaluated with three different proteins: one real protein and two in silico proteins. Each protein is redesigned with three different design schemes that vary the number of rotamers per position.

Proteins

Real protein The first protein selected is the $\beta\beta\alpha$ motif of the second finger module of the DNA binding protein Zif268 [114], the zinc finger DNA binding module. This protein motif contains the most common secondary structures: sheet, helix and turn structures. This protein has been used in previous protein design studies [27]. This protein will be referred as *zinc-finger*. The protein information was downloaded from the Protein Data Bank [11] and the self-energy terms $E(i_r)$ and the pairwise rotamer energy $E(i_r, j_s)$ are computed with the energy functions defined in [75], which are derived from the AMBER force field [20]. The rotamer library used is Dunbrack's backbone-dependent rotamer library [42]. 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.

In-silico proteins Zinc-finger presents a rich energy landscape with a high (albeit unknown) number of near-optimal solutions. In order to assess the ability of multimodal CCPSO to sample efficiently the energy landscape, two proteins with a controlled number of optimal solutions are randomly generated. These proteins were created with the following methodology. First, a residue interaction graph with the same structure of zinc-finger was generated for each protein. Then, new weights in this interaction graph were generated randomly and independently creating an almost flat energy landscape. Finally, a number of solutions with similar quality value were implanted in this energy landscape to create a multiple global optima scenario like the one depicted in Figure 5.3. The following explains each one of these steps in more detail.

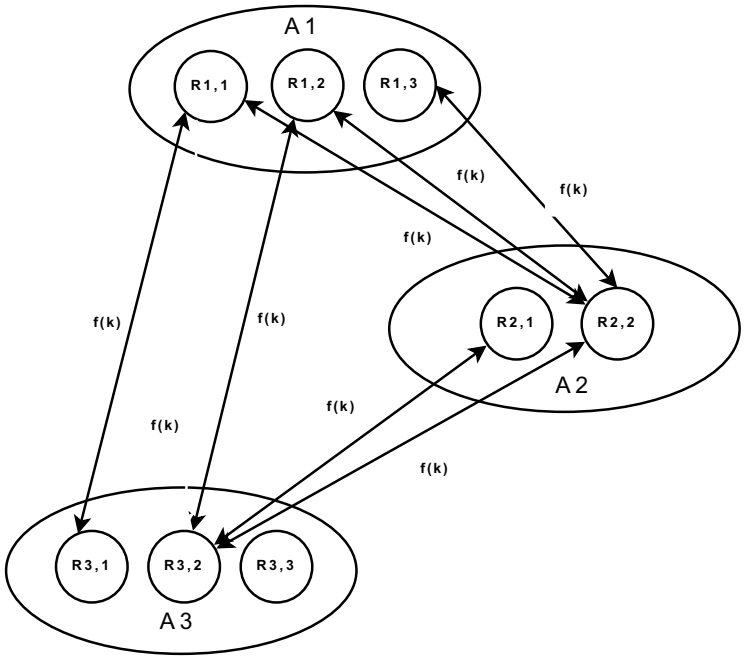


Figure 5.6: Residue interaction graph of in silico proteins. Each amino acid residue is represented by a subgraph that contains all the rotamers that belong to that amino acid. An edge between rotamer i of residue r and rotamer j of residue s has weight $f(k)$ if that weight had a $E(i_r, j_s) <> 0$ in original protein zinc-finger.

The interaction graph of zinc-finger is taken as a starting point to create the new proteins. In this interaction graph, 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 $E(i_r, j_s)$. If an interaction does exist between two rotamers in this interaction graph, then a new weight is created randomly and independently for it with Equation (5.7). This equation assigns a β weight with a gaussian distribution of shape k. Figure 5.6 shows an example of how new weights are generated.

$$f(k) = \beta + \frac{1}{1 + e^k} \tag{5.7}$$

Finally, twenty-five global optima (or near-optimal) solutions sol_{imp} were implanted in these synthetic proteins. The implanted solutions in one protein have an average rotamer pairwise distance value of 0.30. This protein is called *zinc-finger-30-25*. Similarly, the second synthetic protein, named *zinc-finger-75-25*, has an average rotamer pairwise distance in its set of optima solutions of 0.75. This is depicted in Figure 5.7. All interactions between any pair or rotamers that belong to any solution in sol_{imp} are assigned a new weight with Equation (5.8).

$$g(k) = -\beta + \frac{1}{1 + e^k} \tag{5.8}$$

This strategy assigns a low energy to all solutions in sol_{imp} creating a multiple global optima. Since it is of particular interest to test multimodal CCPSO in landscapes with near-optimal solutions a gaussian distribution is used to create solutions of similar quality value.

In summary, the interaction graph of each synthetic protein is generated as follows:

$$E(i_r, j_s) = \begin{cases} f(k) & \text{if } E(i_r, j_s)! = 0 \text{ in zinc-finger} \\ g(k) & \text{if } \exists sol \in sol_{\text{imp}} \text{ such that } i_r \text{ and } j_s \in sol \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

Design model All three proteins are subject to three design schemes with increasing order of difficulty. This is depicted in Figure 5.8. Design model I represents a sequence-fixed redesign, where the task is reduced to choose rotamers for a fixed residue (i.e., this design problem is reduced to side-chain). Design model II represents a partial se-

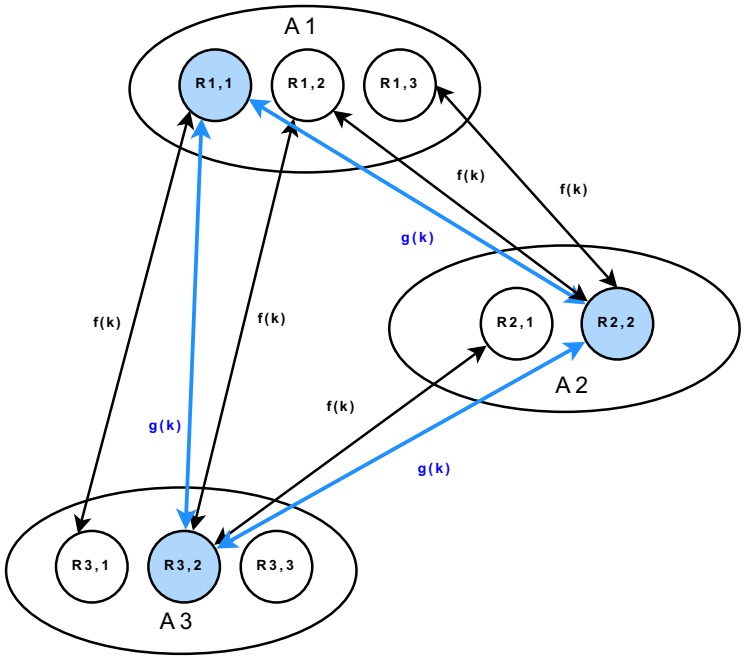


Figure 5.7: Implantation of solutions in synthetic proteins.

quence redesign following the classification of residues into core, surface and boundary classes presented in [27]. The residue positions in the protein structure are partitioned into core, surface and boundary classes. Core residue identities are selected from among the amino acids Ala, Val, Leu, Ile, Phe, Tyr and Trp, while surface residue identities are selected among Ala, Ser, Thr, His, Aps, Asn, Glu, Gln, Lys, and Arg. Boundary residue identities are chosen from the union of these sets. This classification is as follows: one residue position (position number 5) is classified as core, seven residue positions (positions 3,7,12,18,21,22 and 25) are classified as boundary and the remaining twenty residues are assigned to surface. Design model III represents the full sequence design where every position is allow to pick among the twenty amino acids. The three design models are depicted in Figure 5.8. The search space of the three models is presented in Table 5.1.

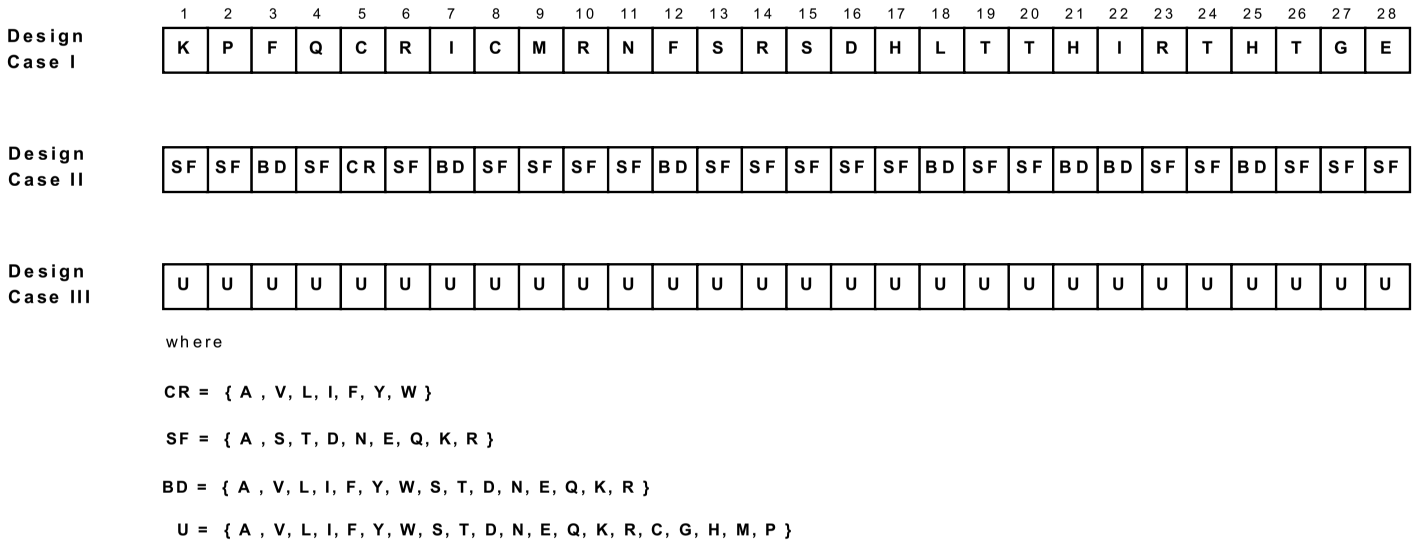


Figure 5.8: Depiction of the three design schemes. Design model I is a fixed sequence redesign, i.e. there is only one possible amino acid per position. Design model II divides the amino acids in three sets, core amino acids (CR) , surface amino acids (SF) and boundary amino acids (BD). Design model III is a full redesign case, and each position is allowed to pick among the twenty amino acids available.

Table 5.1: Benchmark search complexity of protein design schemes

	Model I	Model II	Model III
number of rotamers	587	6512	9016
number of residues	27	26	28
rotamer/residue ratio	21.74	250.46	322
search space size	$4.13509e+26$	$8.47811e+51$	$1.65945e+70$
$\log_{10}$ search space	26.61	61.92	70.22

Distance metrics Two metrics are used to assess the distance between two proteins. The first one computes the distance in rotamer space, which considers all possible rotamer combinations for a fixed sequence length. The second metric computes the distance in sequence space, which considers all possible residue combinations for a fixed sequence length.

Equation (5.10) computes $distance_{rot}$ which measures the distance of two solutions of size n in rotamer space and equation (5.10) estimates the distance of two solutions of size n in residue space, $distance_{res}$.

$$
\begin{aligned}
&distance_{rot}(sol_{s1}, sol_{s2}) = 1.0 - \frac{\sum_{i=1}^{n} same\text{-}rot(sol_{s1}[i], sol_{s2}[i])}{n} \\
&same\text{-}rot(j,k) = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{5.10}
$$

$$
\begin{aligned}
&distance_{res}(sol_{s1}, sol_{s2}) = 1.0 - \frac{\sum_{i=1}^{n} same\text{-}res(sol_{s1}[i], sol_{s2}[i])}{n} \\
&same\text{-}res(j,k) = \begin{cases} 1 & \text{if } res(j) = res(k) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{5.11}
$$

and function $res(x)$ returns the residue to which that rotamer belongs to.

Table 5.2: Details of multimodal CCPSO variants evaluated in protein design task.

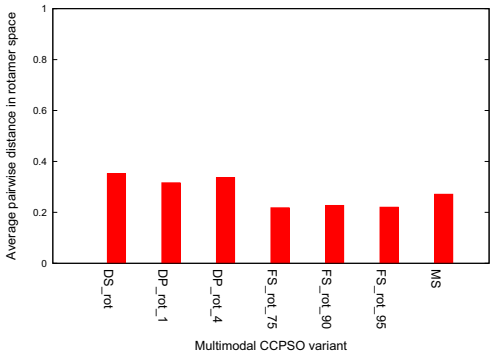
name	diversity strategy	parameters	design model tested
MS	multistart	none	I, II, III
DS_rot	deflation strategy	$distance_{rot}$	I, II, III
DS_res	deflation strategy	$distance_{res}$	II, III
DP_rot_1	deflation strategy	$distance_{rot}$, $\lambda = 1$	I, II, III
DP_rot_4	deflation strategy	$distance_{rot}$, $\lambda = 4$	I, II, III
DP_res_1	deflation strategy	$distance_{res}$, $\lambda = 1$	II, III
DP_res_4	deflation strategy	$distance_{res}$, $\lambda = 4$	II, III
FS_rot_75	fitness sharing	$distance_{rot}$, $\sigma_{share=0.75}$	I, II, III
FS_rot_90	fitness sharing	$distance_{rot}$, $\sigma_{share=0.90}$	I, II, III
FS_rot_95	fitness sharing	$distance_{rot}$, $\sigma_{share=0.95}$	I, II, III
FS_res_75	fitness sharing	$distance_{res}$, $\sigma_{share=0.75}$	II, III
FS_res_90	fitness sharing	$distance_{res}$, $\sigma_{share=0.90}$	II, III
FS_res_95	fitness sharing	$distance_{res}$, $\sigma_{share=0.95}$	II, III

Variants Thirteen variants of multimodal CCPSO are implemented based on the four diversity strategies described in the previous section. Preliminary experiments were run to tune specific parameters to each diversity strategy. The resultant methods are presented in Table 5.2.

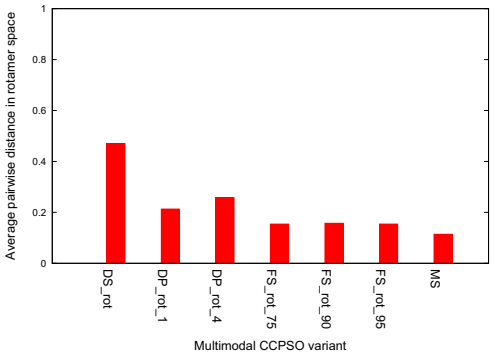
An instance is defined as one protein in one sequence design scheme. Therefore, the benchmark consists of nine instances, each one solved by all multimodal CCPSO variants. Since our method is stochastic, each instance-algorithm pair is executed ten times and average values are reported. The experiments were run on a 2.2 Ghz AMD Quad-Core AMD Opteron(tm) Processor 8356 with 132GB of shared RAM.

5.4 Results

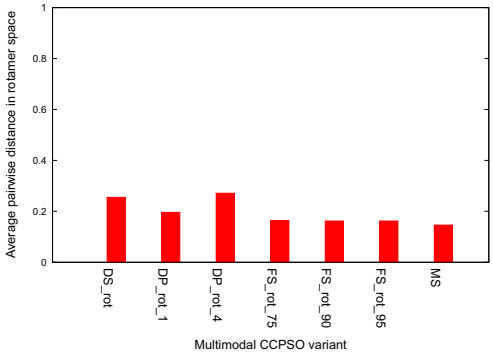
Real protein vs in-silico protein Low-energy ensembles obtained in the real protein, zinc-finger, have a higher average pairwise diversity almost throughout all multimodal CCPSO variants and all design schemes. This corroborates that energy landscapes of real proteins such as zinc-finger have significant more optima than the in silico proteins created. Moreover, we can see in Figures 5.9–5.11 that zinc-finger-30-25 has a higher average pairwise diversity than zinc-finger-70-25 in all design models. This reflects the actual energy landscape of the in silico proteins. Even though both proteins have the same number of global optima (25 solutions each), the optima implanted in zinc-finger-30-25



(a) Results with zinc-finger.



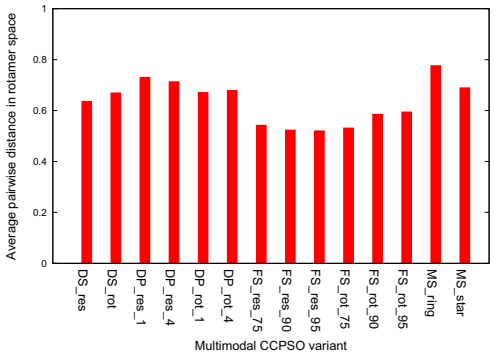
(b) Results with zinc-finger-30-25.



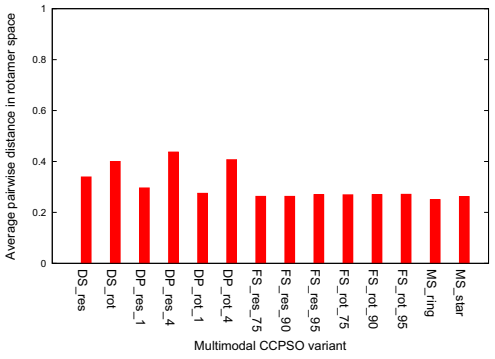
(c) Results with zinc-finger-70-25.

Figure 5.9: Average pairwise diversity in rotamer space for top 100 solutions for each variant of multimodal CCPSO in design case I. Axis X show each one of the methods, and axis Y plots the average pairwise diversity among the top 100 solutions.

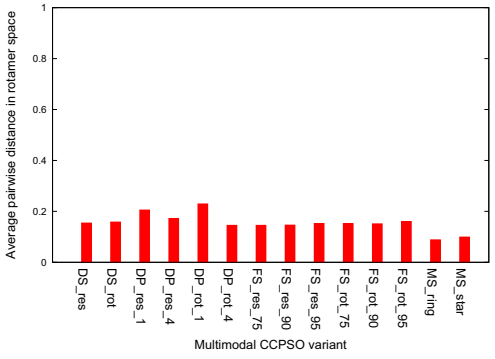
has a lower pairwise similarity. These results provide evidence that multimodal CCPSO is able to distinguish between these three multimodal scenarios. Additionally, while this trend persists across all design models, its effect seems to be modulated by the design model. In design model I (Figure 5.9) the difference between zinc-finger-30-25 and zinc-finger-70-25 is subtle whereas in design model II and II (Figure 5.10 and Figure 5.11 respectively) the effect is more evident. It is also interesting that while zinc-finger has the highest pairwise similarity, these diversity seems to reach its maximum in design model II. In other words, even though design model III provides a considerably larger rotamer space, multimodal CCPSO finds a similar average pairwise diversity in both design cases.



(a) Results with zinc-finger.

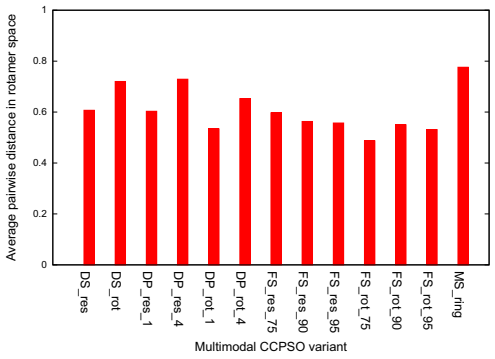


(b) Results with zinc-finger-30-25.

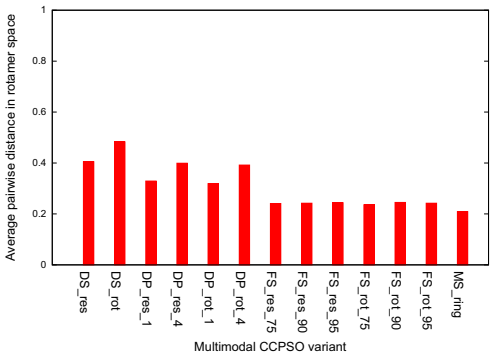


(c) Results with zinc-finger-70-25.

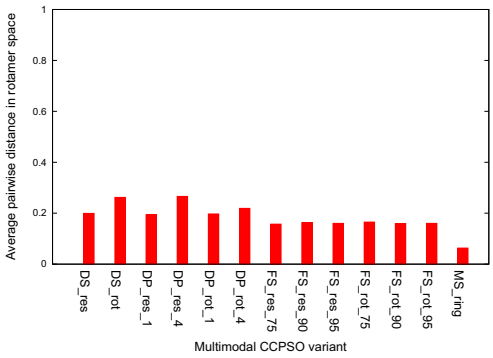
Figure 5.10: Average pairwise diversity in rotamer space for top 100 solutions for each variant of multimodal CCPSO in design case II. Axis X show each one of the methods, and axis Y plots the average pairwise diversity among the top 100 solutions.



(a) Results with zinc-finger.



(b) Results with zinc-finger-30-25.

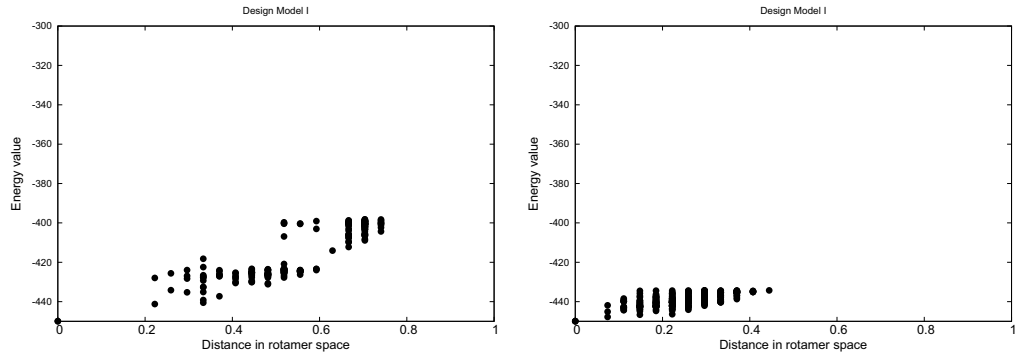


(c) Results with zinc-finger-70-25.

Figure 5.11: Average pairwise diversity in rotamer space for top 100 solutions for each variant of multimodal CCPSO in design case III. Axis X show each one of the methods, and axis Y plots the average pairwise diversity among the top 100 solutions.

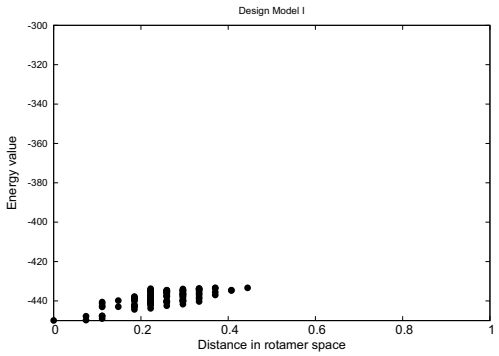
This can be observed in Figure 5.10 a) and 5.11 a). Presumably, this is a feature of the energy landscape and not of the multimodal CCPSO. Finally, it is worth mentioning that all variants are able to make the distinction between the fitness landscape of these three proteins.

High-quality ensembles Figure 5.12–Figure 5.14 shows the quality of the 100 top solutions and their distance to the optimal solution. In design case I, where the search space is the most restricted, multistart and fitness sharing strategy have a similar behavior. Their exploratory power is somewhat limited with a maximum distance to the best solution

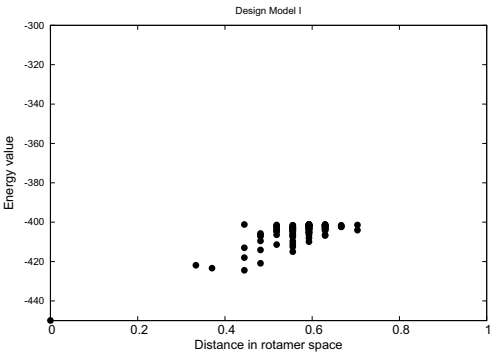


(a) Distance penalty strategy.

(b) Fitness sharing strategy.



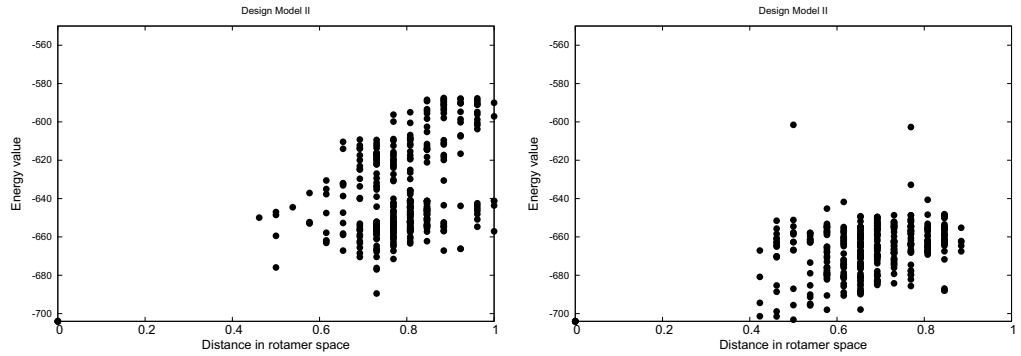
(c) Multistart strategy.



(d) Deflation strategy.

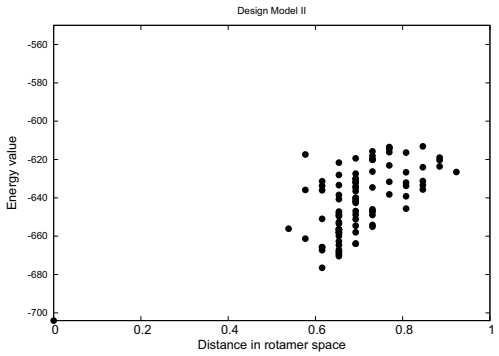
Figure 5.12: Fitness value vs distance in rotamer space for top 100 solutions for each variant of multimodal CCPSO in design case I. Axis X show each the distance to the optimal solution in rotamer space, and axis Y shows the fitness of the solutions.

around 0.4. Both of these methods maintain all their solutions within 5% of the optimal solution. Distance penalty and deflation strategy, on the other hand, are able to find an ensemble of higher pairwise distance but lower quality. In design case II, all variants obtain ensembles of solutions which are relatively far from the global optima, the closest solution is at 0.4. The method that maintains the highest quality ensemble is deflation strategy. This behavior pattern is repeated in design model III, although ensemble quality is higher.

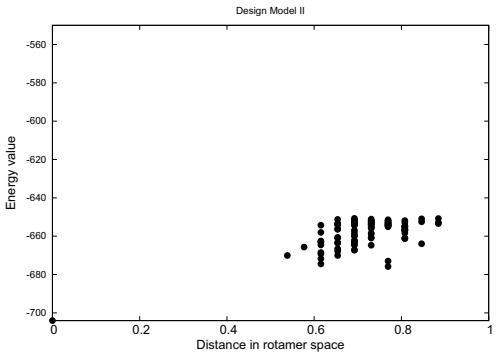


(a) Distance penalty strategy.

(b) Fitness sharing strategy.



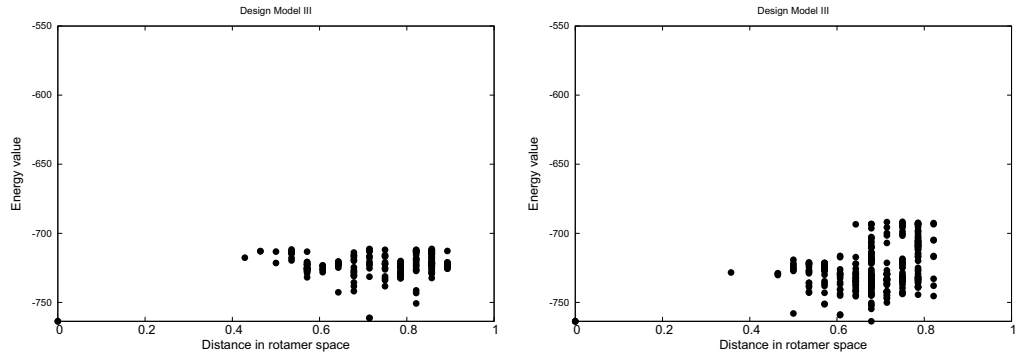
(c) Multistart strategy.



(d) Deflation strategy.

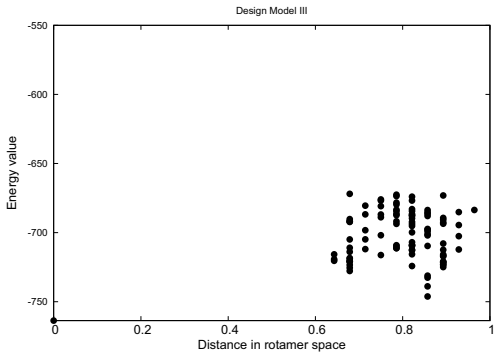
Figure 5.13: Fitness value vs distance in rotamer space for top 100 solutions for each variant of multimodal CCPSO in design case II. Axis X show each the distance to the optimal solution in rotamer space, and axis Y shows the fitness of the solutions.

High-diversity ensembles The deflation and distance penalty diversity strategies consistently produce the highest average pairwise distance in rotamer space. This behavior is consistent throughout the three design schemes in all proteins evaluated. Not surprisingly, the multistart strategy has an inconsistent performance. This method relies too heavily in the stochastic nature of the PSO to explore new regions of the search space. In design case I, the multistart strategy produces a high-quality although low-diversity ensemble while in design case II and III the quality of its ensemble is significantly lower. The fitness sharing technique, on the other hand, has a consistent but not-as-good performance. It produces ensembles of lower diversity than deflation and distance penalty strategies.

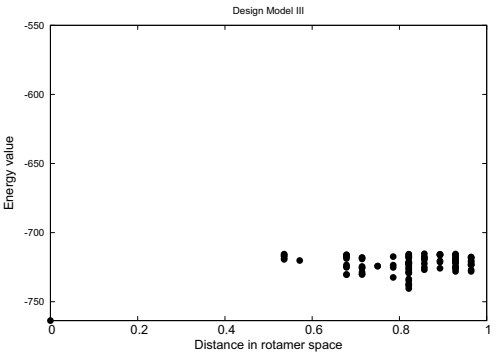


(a) Distance penalty strategy.

(b) Fitness sharing strategy.



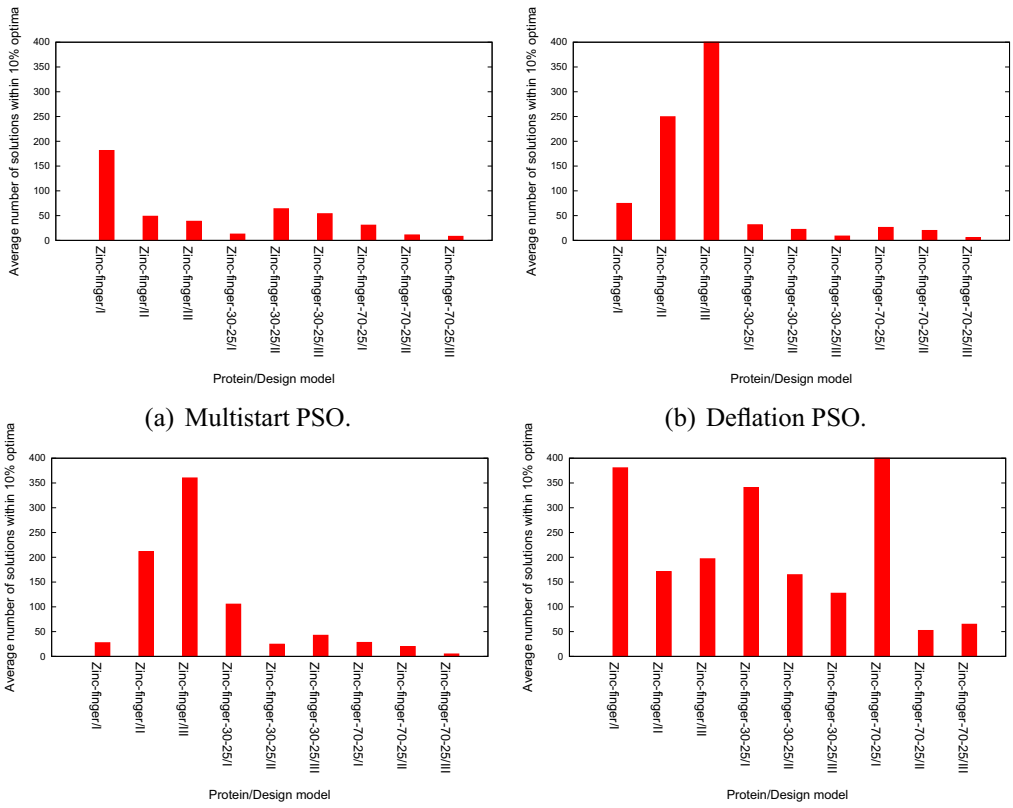
(c) Multistart strategy.



(d) Deflation strategy.

Figure 5.14: Fitness value vs distance in rotamer space for top 100 solutions for each variant of multimodal CCPSO in design case III. Axis X show each the distance to the optimal solution in rotamer space, and axis Y shows the fitness of the solutions.

Number of solutions Figure 5.15 shows the average number of solutions found within 10% of optima by each multimodal CCPSO variant. The fitness sharing strategy is clearly more effective than any other variant in finding "good" near-optima optimal solutions. Distance penalty PSO and deflation strategy have a very similar behavior. Their performance varies greatly based on the specific protein and design model. For protein zinc-finger, there seems to be a trend where the ensemble size increases with the complexity of the design model. Finally, the behavior of the multistart strategy is erratic at best; this algorithm finds more solutions in the more restricted design case I of zinc-finger than in design case III of the same protein.



(a) Multistart PSO.

(b) Deflation PSO.

(c) Distance penalty PSO.

(d) Fitness sharing PSO.

Figure 5.15: Average number of solutions found within 10% of optima for each protein/design model. Axis X show each one of protein/design model combination, and axis Y plots the average number of solutions found.

5.5 Discussion

The previous chapter proved the feasibility of a cooperative combinatorial PSO with a partial-fitness feedback strategy as a search method for the protein structure prediction task. To further study the properties and capabilities of the proposed method, this chapter presents a second set of experiments where multi-space CCPSO faces a different combinatorial problem: to find an ensemble of top-scoring protein sequences. This combinatorial task accentuates the difficulty of the side-chain packing problem not only by considerably increasing the problem size, but also by requiring that multiple local optima of similar quality that the global optimum be detected.

This chapter introduces multimodal CCPSO; in this algorithm multi-space CCPSO is augmented with diversity strategies to extend it to multimodal combinatorial problems. Four diversity strategies were selected based on their previous success in real-value optimization with the intention of characterizing their performance in combinatorial problems.

Multimodal CCPSO is applied to the sequence redesign of a small protein and results show that each diversity strategy has its own performance pattern. The multistart strategy has the less consistent behavior, since it relies in the initial random positions of the particles to find distinct optima in each execution. The fitness sharing strategy has a conservative approach; it keeps a balance between the diversity of solutions and their quality. The distance penalty and deflation strategy are the methods of choice if the interest lies in having an ensemble of solutions with a high average pairwise distance at the cost of having lower-quality solutions. Additionally, the fitness sharing technique seems to be very effective in exploiting good areas of the space. It is capable of finding a large number of multiple optima. Its behavior in the in silico protein is particularly striking. It is the only variant capable of finding a large number of solutions in these proteins. The ensemble found by the fitness sharing technique consists of slight variants of the 25 original solutions implanted in the synthetic protein.

These results corroborate that Multi-space CCPSO, augmented with an ensemble memory and diversity strategies, is an effective particle swarm algorithm to sample multimodal combinatorial problems. Several diversity approaches have previously been employed in particle swarm algorithms [18][110]. These approaches are typically used to

escape local optima, and to try to locate multiple global solutions. There have been several attempts to extend PSO to locate all the global optima. However, these algorithms were tested only in much simpler numerical benchmark function. At present, multimodal CCPSO seems to be the only variant of PSO which has been developed for multimodal combinatorial problems.

Chapter 6

Discussion

PSO algorithms have been very successful in almost all areas where they have been applied, with perhaps the exception of combinatorial optimization problems where further improvements to PSO techniques are still needed before the PSO can compete on par with other techniques [6][122]. A recent study of publications in particle swarm optimization [122] reveals that less than 3.5% of these papers deal with combinatorial optimization problems. Moreover, the majority of current implementations of combinatorial PSO are used to detect promising regions in the search space, which are then passed to a dedicated local search algorithm such as simulated annealing or fast local search [24].

One of the main problems with combinatorial PSO is that it requires one to redefine the interaction between a particle or candidate solution and continuous space. There is no natural representation for combinatorial problems in continuous space (e.g., how to redefine the velocity operator). This has been addressed by previous research by encoding the candidate solution as a permutation and designing specialized velocity operators for this permutation space [23][149][157]. One of the main goals of the research presented here was to provide some insight about the feasibility of an alternative approach. Specifically, it focused on the following questions: "Is particle swarm optimization a viable approach to solve combinatorial problems?", "Does the representation of a particle moving in a continuous space need to be sacrificed for a permutation representation?", and

"Can a cooperative strategy provide a mechanism to design an effective PSO to handle combinatorial problems without sacrificing the concept of particles moving in a continuous space?". In an attempt to answer these questions, I have proposed two cooperative strategies for combinatorial PSO. The central hypothesis is that by allowing a set of particles, rather than each single particle as in most combinatorial PSOs, to represent solutions to problems, one can collectively construct solutions. These cooperative strategies allow each particle to optimize its own choices, while being guided implicitly toward regions of the space that are more promising to the rest of the swarm.

In cooperative algorithms a solution sol_s consists of n components and has a fitness value f. However, there is no explicit fitness value associated with the individual components. This raises the issue of how to reward (or penalize) a particular component for its contribution towards sol_s, i.e., how to know if a particular component is helping or hurting the fitness of the overall solution. This problem is known as the *credit assignment problem*, and it has been pointed out in previous work in cooperative PSO algorithms for numerical optimization [44][153][154]. Of particular interest is the approach presented in [154], namely CPSO-S_k, where each individual solution component is assigned to a particular swarm and optimized *individually*. The major difficulty with CPSO-S_k is that the solution is a plain combination of the swarm's global best. Consequently, solutions are the result of each swarm selecting its own best solution, i.e., each swarm selects its local minimizer, instead of reaching a compromise with each other and selecting components that will produce the global minimum. To help escape these pseudominima a traditional PSO is combined with CPSO-S_k, interleaving their execution and merging their results

by replacing half of the population of one with the best solutions of the other.

My research takes a different direction, and provides two alternatives to deal with this credit assignment problem. The first one, implemented in shared-space CCPSO consists in placing all particles in a n-dimensional space, where each possible value is represented by an attractor-particle. This representation of the space allows a particle's position to be used in a feedback mechanism to reward attractors which are successful in pulling particles towards them. This way, each possible value for a solution component (i.e., each attractor) gets feedback that reflects its contribution to the overall solution. This feedback mechanism acts as *stigmergic* communication between particles.

Multi-space CCPSO, on the other hand, presents a qualitatively different approach. This model partitions the space by placing each particle in its own high-dimensional continuous space. As in the shared-space CCPSO, each individual particle moves around the high dimensional space based solely on the forces exerted by attractors. The difference lies in the feedback mechanism. In multi-space CCPSO, particles communicate by collectively creating a candidate solution sol_s. This candidate solution sol_s of each subwarm is used (as in traditional PSO) to keep a memory of the best solution seen so far by the subswarm (cognitive memory b_s), and the best solution seen so far by a neighboring subswarm (social memory n_s). This memories b_s and n_s reflect a snapshot of the choices of other particles in the subswarm. They serve as a bellwether of the overall status of the subswarm. Each particle can now assess the quality of the contribution of its local choices to these overalls solutions. This component-by-component optimization encourages fine-tuning of the solution by each particle when the candidate solution mem-

ory does not change from one iteration to the next, and exploration of a different part of the space when the memories are modified. This cooperative strategy allows CCPSO to preserve the concept of particles that move throughout a continuous high-dimensional space while keeping their own cognitive and social memories of candidate solutions. In this sense, multi-space CCPSO is closer to the original goal of preserving the key PSO features while extending it to combinatorial optimization.

Multimodal CCPSO extends these combinatorial PSO algorithms to efficiently sample the search space in problems with multiple global optima. This new algorithm extends the ability of multi-space CCPSO to discover sets of high-quality and diverse solutions. Every time multimodal CCPSO detects a solution of high quality (i.e., a potential optimum) it "marks" the region in the search space by storing this solution in an external memory. The external memory is then used by diversity strategies to encourage particles to search different regions of the search space.

Throughout this dissertation, an extension to particle swarm optimization has been presented and demonstrated on varied combinatorial tasks. This extension has the main purpose of facilitating the application of particle swarm optimization to combinatorial problems. CCPSO is aimed at combinatorial problems where the task is the assignment of values among a discrete set of values and it is not intended to permutation problems, such as the traveling salesman problem. Nevertheless, permutation problems can be tackled by CCPSO by finding a suitable mapping between the permutation problem and the "assignment" representation of CCPSO.

After the work on the three applications presented, it is clear that there is more work

to be done. First, one limitation of multi-space CCPSO is the high cost of recomputing the fitness value of each particle's attractor at each iteration where the cognitive or social memory has changed. Multi-space CCPSO deals with this problem by having a noisy evaluation of this attractor's fitness. When an attractor fitness needs to be updated, a biased coin with probability τ is flipped to determine if the fitness value is updated. This noisy calculation of the attractor's fitness is successful in the combinatorial problems presented in this dissertation, but it is not clear whether it would prove successful in other types of combinatorial problems. An alternative solution could be to use the value of the attractor's fitness of nearby particles to avoid useless computations.

Another direction for future research is related to the performance of the diversity strategies employed specifically for ensemble of near-optimal solutions. Each one of the diversity strategies studied in Chapter 5 had a specific behavior pattern, a pattern that was maintained throughout the execution of multimodal CCPSO. An alternative here would be to add an adaptive mechanism to monitor both the quality and pairwise diversity of the solutions stored so far, and based on this information, provide feedback to the diversity strategy. The diversity strategy could then adjust its parameters accordingly to produce either higher quality solutions or higher diversity solutions. Additionally, the techniques explored were aimed to find an ensemble of dissimilar near-optimal solutions. They do not maximize *explicitly* the distance between solutions. It would be interesting to compare the performance in obtaining a high-quality and diverse ensemble of solutions of the multimodal strategies implemented here with multi-objective techniques.

6.1 Contributions

- A new strategy to solve combinatorial problems where all components extract their solution from the same values. The proposed strategy consists in using the position of the particles (which are trying to optimize other components) to give feedback to each other, implicitly guiding everybody else to their preferred area. This feedback mechanism is implemented by the attractors' fitness. Each particle selects its best solution component, but instead of passively communicating this choice, they modify their environment by increasing the fitness of the selected attractor, increasing its chances of being selected by other particles. All attractors are in a sense competing to win all particles, and each partial success (being selected by one particle) increases its ability to attract others.

- A generalization of the previous PSO model, allowing the PSO to solve combinatorial problems where each component optimizes its own choices. This new scheme, multi-space CCPSO, partitions the combinatorial problem into components and optimizes each one of them individually by a particle that moves on its own multidimensional space. The feedback mechanism consists of computing a fitness value for each possible choice of the particle; this fitness value is estimated using the cognitive and social memory of the subswarm as template to assess its contribution to the overall solution. Note that multi-space CCPSO, unlike shared-space CCPSO, preserves the key feature of cognitive and social memories.

- A modular methodology for multimodal combinatorial optimization problems is

presented in multimodal CCPSO. This new PSO extension finds a set of optimal and near-optimal solutions by forcing multi-space CCPSO to sample different regions of the search space. Every time multi-space CCPSO detects a solution of high quality (i.e., a potential optimum) it "marks" the region in the search space by storing this solution in an external memory. The external memory is then used by diversity strategies to encourage particles to search different regions of the search space. The multi-space CCPSO algorithm is augmented with existing diversity strategies, leading to four multimodal CCPSO variants. These new algorithms offer insight about the behavior of multi-space CCPSO in multimodal problems. Additionally, these variants are augmented with an external memory and a storing mechanism to provide an ensemble of high-quality distinct solutions. The final ensemble of solutions produced by each variant are examined by their optimality and pairwise diversity.

- Three important combinatorial problems are tackled by these new models. Experiments in Chapter 3 show the diagnosis problem can be solved by the model with reasonable accuracy, up to 90% of exactly correct answers, which is quite respectable given that only local computations are involved. Chapter 4 shows that Multi-space CCPSO has encouraging performance in Side-Chain Packing, an NP-hard problem, by obtaining optimal or near-optimal solutions in all datasets tested. Finally, Chapter 5 demonstrates the ability of multimodal CCPSO to predicting a set of low-energy sequences for the Protein Design problem.

In summary, throughout this dissertation, an extension to PSO for combinatorial

problems has been presented and demonstrated in different NP combinatorial tasks. This extension has the main purpose of enabling PSO to partition complex combinatorial problems in smaller subtasks, while maintaining active communication between these components.

Bibliography

[1] E. Aarts and J. Lenstra. *Local search in combinatorial optimization*. Princeton Univ Pr, 2003.

[2] 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):597–612, 2002.

[3] R. Arkin and T. Balch. Cooperative multiagent robotic systems. In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, Cambridge, MA, 1998.

[4] M. Atallah. *Algorithms and theory of computation handbook*. CRC, 1999.

[5] T. Bäck and H. P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[6] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, 2008.

[7] S. Baskar and P. N. Suganthan. A novel concurrent particle swarm optimization. In *Proc. Congress on Evolutionary Computation, CEC'2004*, volume 1, pages 792–796, 2004.

[8] R. Beckers, O. E. Holland, and J. L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In *Artificial Life IV: Proceedings of the International Workshop on the Synthesis and Simulation of Living Systems, Third Edition*. MIT Press, 1994.

[9] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[10] G. Beni and J. Wang. Intelligence in cellular robotic systems. In *Proceedings of the NATO Advanced Workshop on Robots and Biological Systems, Il Ciocco, Tuscany, Italy, June 26- 30, 1989*, 1989.

[11] F. Bernstein, T. Koetzle, G. Williams, E. Meyer Jr, M. Brice, J. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The Protein Data Bank: a computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, 112(3):535, 1977.

[12] S. Bird and X. Li. Adaptively choosing niching parameters in a PSO. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 3–10. ACM New York, NY, USA, 2006.

[13] C. Blum and X. Li. Swarm Intelligence in Optimization. *Swarm Intelligence*, pages 43–85, 2008.

[14] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

[15] K. Bohringer, R. Brown, B. Donald, and J. Jennings. Distributed robotic manipulation: Experiments in minimalism. In *ISER, Proceedings of the Internation Symposium on Experimental Robotics*, Stanford, CA, 1995.

[16] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[17] R. Brits, A. Engelbrecht, and F. van den Bergh. A niching particle swarm optimizer. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL02)*, volume 2, pages 692–696, 2002.

[18] R. Brits, A. Engelbrecht, and F. van den Bergh. Locating multiple optima using particle swarm optimization. *Applied Mathematics and Computation*, 189(2):1859–1883, 2007.

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

[20] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods. The AMBER biomolecular simulation programs. *Journal of computational chemistry*, 26(16):1668–1688, December 2005.

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

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

[23] M. Clerc. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In G. C. Onwubolu and B. V. Babu, editors, *New Optimization Techniques in Engineering*, volume 141, pages 219–239. Springer, 2004.

[24] M. Clerc. *Particle Swarm Optimization*. Wiley-ISTE, February 2006.

[25] W. W. Cohen. Adaptive mapping and navigation by teams of simple robots. *Robotics and Autonomous Systems*, 18(4):411–434, October 1996.

[26] L. Console, D. T. Dupré, and P. Torasso. A theory of diagnosis for incomplete causal models. In *IJCAI. Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1311–1317. Morgan Kaufmann, 1989.

[27] B. Dahiyat, C. Sarisky, and S. Mayo. De novo protein design: towards fully automated sequence selection. *Journal of molecular biology*, 273(4):789–796, 1997.

[28] S. Das, A. Abraham, and A. Konar. Swarm intelligence algorithms in bioinformatics. In *Computational Intelligence in Bioinformatics*, volume 94 of *Studies in Computational Intelligence*, pages 113–147. Springer Berlin / Heidelberg, 2008.

[29] V. Dasigi and J. A. Reggia. Parsimonious covering as a method for natural language interfaces to expert systems. *Artificial Intelligence in Medicine*, 1:49–60, 1989.

[30] J. de Kleer. Focusing on probable diagnosis. *Readings in model-based diagnosis*, pages 131–137, 1992.

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

[32] K. Deb and D. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms table of contents*, pages 42–50. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1989.

[33] A. Delgado. Control of nonlinear systems using a self organizing neural network. *Journal of Neural Computing and Applications*, 9:113–123, 2000.

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

[35] 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):539–542, April 1992.

[36] A. Dhariwal, G. S. Sukhatme, and A. A. G. Requicha. Bacterium-inspired robots for environmental monitoring. In *ICRA, IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004.

[37] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–56, 1997.

[38] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[39] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Books, 2004.

[40] M. Dorigo, E. Tuci, R. Groß, V. Trianni, T. Labella, S. Nouyan, C. Ampatzis, J. Deneubourg, G. Baldassarre, S. Nolfi, et al. The Swarm-Bots project. *Lecture Notes in Computer Science*, 3342:31–44, 2005.

[41] A. Drogoul and J. Ferber. From tom thumb to the dockers: some experiments with foraging robots. In *From animals to animats 2. Proceedings of the second international conference on simulation of adaptive behavior*, pages 451–459, Cambridge, MA, USA, 1993. MIT Press.

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

[43] R. C. Eberhart, Y. Shi, and J. Kennedy. *Swarm Intelligence*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, March 2001.

[44] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2006.

[45] K. Eshghi. Abductive planning with event calculus. In *ICSLP. Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 562–579. MIT Press, 1988.

[46] L. J. Fogel. Evolutionary programming in perspective: The top-down view. In I. Robert, J. M. Zurada, and C. J. Robinson, editors, *Computational Intelligence: Imitating Life*, pages 135–146. IEEE Press, Piscataway, NJ, USA, 1994.

[47] M. Fromer and C. Yanover. Accurate prediction for atomic-level protein design and its application in diversifying the near-optimal sequence space. *Proteins: Structure, Function, and Bioinformatics*, 75(3), 2009.

[48] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Structure decision method for self organising robots based on cell structures CEBOT. In *IEEE International Conference on Robotics and Automation, 1989*, volume 2, pages 695–700, 1989.

[49] A. García-Villoria and R. Pastor. Introducing dynamic diversity into a discrete particle swarm optimization. *Computers and Operations Research*, 36(3):951–966, March 2009.

[50] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, October 1970.

[51] I. Georgiev, R. Lilien, and B. Donald. The minimized dead-end elimination criterion and its application to protein redesign in a hybrid scoring and search algorithm for computing partition functions over molecular ensembles. *Journal of Computational Chemistry*, 29(10):1527–1542, 2008.

[52] F. Glover. Tabu search–part I. *INFORMS Journal on Computing*, 1(3):190, 1989.

[53] F. Glover. Tabu search–part II. *INFORMS Journal on Computing*, 2(1):4, 1990.

[54] F. Glover and F. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[55] D. Goldberg and M. J. Mataric. Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. In T. Balch and L. E. Parker, editors, *Robot Teams: From Diversity to Polymorphism*. AK Peters, 2002.

[56] D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application table of contents*, pages 41–49. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1987.

[57] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[58] M. Grötschel and L. Lovász. Combinatorial Optimization. In R. L. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics, Volume II*, pages 1541–1597. Elsevier (North-Holland), 1995.

[59] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[60] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, pages 196–210, 1962.

[61] D. Henderson and S. Jacobson. The Theory and Practice of Simulated Annealing In Handbook of Metaheuristics, ch. 10. *Kluwer Academic Publishers, Boston*, pages 287–319, 2003.

[62] J. R. Hobbs. An integrated abductive framework for discourse interpretation. In *Proceedings, AAAI Symposium on Automated Abduction*, pages 10–12, 1990.

[63] O. Holland and C. Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.

[64] H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Morgan Kaufmann, 2005.

[65] P. James and D. Adnan. Approximating map using local search. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 403–410, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[66] D. Johnson. Local optimization and the traveling salesman problem. In *Proc. 17th Colloq. on Automata, Languages and Programming*, volume 443, pages 446–461. Springer, 1990.

[67] C. Jones and M. J. Mataric. Adaptive division of labor in large-scale minimalist multi-robot systems. In *IROS 2003. Proceedings of International Conference on Intelligent Robots and Systems*, volume 2, pages 1969–1974, 2003.

[68] C. Jones and M. J. Mataric. From local to global behavior in intelligent self-assembly. In *ICRA 2003, Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 721–726, September 2003.

[69] J. R. Josephson and S. G. Josephson, editors. *Abductive Inference: Computation, Philosophy, Technology*. Cambridge University Press, New York, 1994.

[70] I. Kassabalidis, M. El-Sharkawi, and A. da Silva. Dynamic security border identification using enhanced particle swarm optimization. *IEEE Transactions on Power Systems*, 17(3):723, 2002.

[71] B. A. Kazemi and C. K. Mohan. Discrete multi-phase particle swarm optimization. In *Information Processing with Evolutionary Algorithms*, pages 305–327. Springer London, 2005.

[72] I. Kelly and D. Keating. Flocking by the fusion of sonar and active infrared sensors on physical autonomous mobile robots. In *The Third International Conference on Mechatronics and Machine Vision in Practice*, volume 1, pages 1–4, Guimaraes, Portugal, 1996.

[73] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[74] 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*, volume 5, pages 4104–4108, 1997.

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

[76] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[77] S. Kirkpatrick, Jr, and M. P. Vecchi. Optimization by simulated annealing. In *Readings in computer vision: issues, problems, principles, and paradigms*, pages 606–615. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[78] K. B. Korb and A. E. Nicholson. *Bayesian Artificial Intelligence*. CRC Press, 2004.

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

[80] J. R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[81] R. C. Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1/2):85–101, 2000.

[82] R. C. Kube and H. Zhang. Collective robotics: from social insects to robots. *Adaptive Behavior*, 2(2):189–218, 1993.

[83] P. Laarhoven and E. Aarts. *Simulated annealing: theory and applications*. Springer, 1987.

[84] E. Larrosa and J. Alberto. *New heuristics for global optimization of complex bioprocesses*. PhD thesis, Universidad de Vigo, 2008.

[85] E. C. Laskari, K. E. Parsopoulos, and M. N. Vrahatis. Particle swarm optimization for integer programming. In *Proc. of the IEEE Congress on Evolutionary Computation, CEC'02*, pages 1582–1587, 2002.

[86] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations research*, pages 699–719, 1966.

[87] B. A. Lazikani, J. Jung, Z. Xiang, and B. Honig. Protein structure prediction. *Current Opinion in Chemical Biology*, 5(1):51–56, February 2001.

[88] T. Li, C. Wei, and W. Pei. PSO with sharing for multimodal function optimization. In *Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on*, volume 1, pages 450–453 Vol.1, 2003.

[89] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. *Lecture Notes in Computer Science*, pages 105–116, 2004.

[90] S. Lippow and B. Tidor. Progress in computational protein design. *Current Opinion in Biotechnology*, 18(4):305–311, August 2007.

[91] M. Lovbjerg, T. K. Rasmussen, and T. Krink. Hybrid Particle Swarm Optimizer with Breeding and Subpopulation. In *Proc. of the Genetic and Evolutionary Computation Conference, GECCO'2001*, volume 1, pages 469–476. Morgan Kaufmann, 2001.

[92] O. Martin, S. Otto, and E. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 1992.

[93] A. Martinoli, A. J. Ijspeert, and F. Mondada. Understanding collective aggregation mechanisms: From probabilistic modelling to experiments with real robots. *Robotics and Autonomous Systems*, 29:51–63, 1999.

[94] M. J. Mataric. Designing and understanding adaptive group behavior. *Adaptive Behavior*, 4(1):51–80, 1995.

[95] M. J. Mataric. *Interaction and intelligent behavior*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

[96] M. J. Mataric, M. Nilsson, and K. Simsarian. Cooperative multi-robot box-pushing. In *IROS '95: Proceedings of the International Conference on Intelligent Robots and Systems*, volume 3, pages 556–561, Washington, DC, USA, 1995. IEEE Computer Society.

[97] R. A. Miller, H. E. Pople, and J. D. Myers. Internist-1: An experimental computer-based diagnostic consultant for general internal medicine. In W. J. Clancey and E. H. Shortliffe, editors, *Readings in Medical Artificial Intelligence: The First Decade*, pages 190–209. Addison-Wesley, Reading, MA, 1984.

[98] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[99] L. Mitten. Branch-and-bound methods: General formulation and properties. *Operations Research*, pages 24–34, 1970.

[100] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[101] S. D. Müller, J. Marchetto, S. Airaghi, and P. Koumoutsakos. Optimization Based on Bacterial Chemotaxis. *IEEE Transactions on Evolutionary Computation*, 6(1):16–29, 2002.

[102] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proceedings of IEEE International Conference on Robotics and Automation, 1994*, volume 1, pages 441–448, May 1994.

[103] D. S. Nau and J. A. Reggia. Relationships between deductive and abductive inference in knowledge-based diagnostic problem solving. In *Expert Database Workshop*, pages 549–558, 1984.

[104] N. Nedjah and L. de Macedo Mourelle. *Swarm Intelligent Systems*, volume 26 of *Studies in Computational Intelligence*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006.

[105] G. Nemhauser and L. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, 1988.

[106] S. Nouyan and M. Dorigo. Chain formation in a swarm of robots. Technical Report 2004-18, IRIDIA - Universit Libre de Bruxelles, Belgium, February 2004.

[107] J. Onuchic, Z. Luthey-Schulten, and P. Wolynes. Theory of protein folding: the energy landscape perspective. *Annual review of physical chemistry*, 48(1):545–600, 1997.

[108] L. E. Parker. Alliance: an architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *Advanced Robotic Systems and the Real World. Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems '94*, volume 2, pages 776–783, September 1994.

[109] K. Parsopoulos, V. Plagianakos, G. Magoulas, and M. Vrahatis. Stretching technique for obtaining global minimizers through particle swarm optimization. In *Proceedings of the Particle Swarm Optimization Workshop*, volume 29. Citeseer, 2001.

[110] K. Parsopoulos and M. Vrahatis. Modification of the particle swarm optimizer for locating all the global minima. In *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Prague, Czech Republic, 2001*, page 324. Springer Verlag Wien, 2001.

[111] K. Parsopoulos and M. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1(2):235–306, 2002.

[112] K. Parsopoulos and M. Vrahatis. On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):211–224, 2004.

[113] A. Passaro and A. Starita. Particle swarm optimization for multimodal functions: a clustering approach. *Journal of Artificial Evolution and Applications*, 8(2), 2008.

[114] N. Pavletich and C. Pabo. Zinc finger-DNA recognition: crystal structure of a Zif268-DNA complex at 2.1 A. *Science*, 252(5007):809–817, 1991.

[115] C. Peirce, C. Hartshorne, and P. Weiss. *Collected Papers of Charles Sanders Peirce*. Harvard University Press, Cambridge, MA, 1932.

[116] Y. Peng and J. A. Reggia. A probabilistic causal model for diagnostic problem solving, part I: integrating symbolic causal inference with numeric probabilistic inference. *IEEE Transactions on Systems, Man and Cybernetics*, 17(2):146–162, 1987.

[117] Y. Peng and J. A. Reggia. A probabilistic causal model for diagnostic problem solving, part II: Diagnostic strategy. *IEEE Transactions on Systems, Man and Cybernetics*, 17(3):395–406, 1987.

[118] Y. Peng and J. A. Reggia. A connectionist model for diagnostic problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 19(2):285–298, 1989.

[119] Y. Peng and J. A. Reggia. *Abductive inference models for diagnostic problem-solving*. Symbolic Computation. Springer-Verlag New York, Inc., 1990.

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

[121] 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):269–281, May 2001.

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

[123] J. Ponder and F. Richards. Tertiary templates for proteins: Use of packing criteria in the enumeration of allowed sequences for different structural classes. *Journal of Molecular Biology*, 193(4):775–791, February 1987.

[124] H. E. Pople, J. D. Myers, and R. A. Miller. Dialog: A model of diagnostic logic for internal medicine. In *IJCAI. Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 848–855, Tiblisi, USSR, 1975.

[125] M. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature PPSN III*, Lecture Notes in Computer Science, pages 249–257. Springer Berlin / Heidelberg, 1994.

[126] J. Pugh and A. Martinoli. Discrete Multi-Valued Particle Swarm Optimization. In *Proc. IEEE Swarm Intelligence Symposium, SIS '06*, 2006.

[127] W. F. Punch, M. C. Tanner, and J. R. Josephson. Design considerations for PEIRCE, a high-level language for hypothesis assembly. In *Proceedings of the Expert Systems in Government Symposium*, 1986.

[128] O. Raiman, J. de Kleer, and V. A. Saraswat. Critical reasoning. In *Proc. of International Joint Conference on Artificial Intelligence*, pages 18–23, 1993.

[129] I. Rechenberg. Evolution strategy. In J. M. Zurada, R. J. M. II, and C. Robinson, editors, *Computational Intelligence: Imitating Life*, pages 147–159. IEEE Press, Piscataway, NJ., 1994.

[130] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[131] J. A. Reggia, D. S. Nau, and P. Y. Wang. Diagnostic expert systems based on a set covering model. *International Journal on Man-Machine studies*, 19:437–460, 1983.

[132] J. A. Reggia, D. S. Nau, and P. Y. Wang. A formal model of diagnostic inference part I: problem formulation and decomposition. *Information Sciences*, 37(1-3):227–256, 1985.

[133] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[134] A. Rodriguez and J. A. Reggia. Extending self-organizing particle systems to problem solving. *Artificial Life*, 10(4):379–395, 2004.

[135] A. Rodriguez and J. A. Reggia. Collective-movement teams for cooperative problem solving. *Integrated Computer-Aided Engineering. Special Issue: Performance Metrics for Intelligent Systems*, 12(3):217–235, 2005.

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

[137] I. Schoeman and A. Engelbrecht. A parallel vector-based particle swarm optimizer. In *Proceedings of the 7th International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA05)*. Springer, 2005.

[138] H. Schwefel. *Evolution and optimum seeking*. Wiley-Interscience, 1995.

[139] D. Sciretti, P. Bruscolini, A. Pelizzola, M. Pretti, and A. Jaramillo. Computational protein design with side-chain conformational entropy. *Proteins: Structure, Function, and Bioinformatics*, 74(1):176–191, 2009.

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

[141] M. Shanahan. Prediction is deduction but explanation is abduction. In *IJCAI. Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1055–1060. Morgan Kaufmann, 1989.

[142] Y. Shi and R. Eberhart. A modified particle swarm optimizer. *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73, May 1998.

[143] A. Shmygelska and H. Hoos. An Improved Ant Colony Optimisation Algorithm for the 2D HP Protein Folding Problem. In *Advances in Artificial Intelligence*, Lecture Notes in Computer Science, page 993. Springer Berlin / Heidelberg, 2003.

[144] W. M. Spears, K. A. De Jong, T. Back, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In *ECML '93: Proceedings of the European Conference on Machine Learning*, pages 442–459, London, UK, 1993. Springer-Verlag.

[145] L. Steels. Cooperation between distributed agents through self-organization. In Y. Demazeau and J. P. Mueller, editors, *MAAMAW-89, Decentralized Al - Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 175–196, Amsterdam, 1990. Elsevier Science.

[146] L. Steels. The artificial life roots of artificial intelligence. *Artificial Life Journal*, 1(1):89–125, 1994.

[147] M. E. Stickel. Rationale and methods for abductice reasoning in natural-language interpretation. In R. Studer, editor, *Natural Language and Logic. Proceedings of International Symposium Natural Language and Logic*, volume 459 of *Lecture Notes in Computer Science*, pages 233–252. Springer, 1989.

[148] T. Stützle. *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. PhD thesis, Darmstadt University of Technology, 1998.

[149] 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*, volume 2, pages 1412–1419, 2004.

[150] D. Terzopoulos, X. Tu, and R. Grzeszczuk. Artificial fishes: autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artificial Life*, 1(4):327–351, 1994.

[151] D. Thalmann, S. R. Musse, and F. Garat. Guiding and interacting with virtual crowds in real-time. In *Proceedings of Eurographics Workshop on Animation and Simulation*, pages 23–34, Milan, Italy, 1999.

[152] D. Vail and M. M. Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. In A. Schultz, L. Parkera, and F. Schneider, editors, *Multi-Robot Systems*, pages 87–98. Kluwer, 2003.

[153] F. van den Bergh and A. P. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, 26:84–90, 2000.

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

[155] J. von Neumann. *The Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[156] J. Wald, M. Farach, M. Tagamets, and J. A. Reggia. Generating plausible diagnostic hypotheses with self-processing causal networks. *Journal of Experimental and Theoretical Artificial Intelligence*, 1(2):91–112, 1989.

[157] K. P. Wang, L. Huang, C. G. Zhou, and W. Pang. Particle swarm optimization for traveling salesman problem. In *Proc. of Int. Conference on Machine Learning and Cybernetics*, volume 3, pages 1583–1585, 2003.

[158] B. B. Werger and M. J. Mataric. Robotic food chains: Externalization of state and program for minimal-agent foraging. In *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1996. The MIT Press/Bradford Books.

[159] D. Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, 2001.

[160] S. Wolfram. *Theory and applications of cellular automata*. Advanced Series on Complex Systems. World Scientific Publication, 1986.

[161] S. Wolfram. *A new kind of science*. Wolfram Media Inc., 2002.

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

[163] J. Xu. Rapid protein side-chain packing via tree decomposition. In S. Miyano, J. P. Mesirov, S. Kasif, S. Istrail, P. A. Pevzner, and M. S. Waterman, editors, *RECOMB*, volume 3500 of *Lecture Notes in Computer Science*, pages 423–439. Springer, 2005.

[164] Y. Xu and C. Zhang. A neural network model for monotonic diagnostic problem solving. In *Proc. of the 2nd IEEE International Conference on Intelligent Processing Systems*, pages 574–578, 1998.