Chapter 3

MULTISET RULE-BASED PROGRAMMING PARADIGM FOR SOFT-COMPUTING IN COMPLEX SYSTEMS

E. V. Krishnamurthy
Australian National University
and
*Vikram Krishnamurthy**
University of British Columbia

Abstract

This chapter describes a rule-based multiset distributed programming paradigm as a unifying theme for conventional as well as soft and innovative computing, e.g., Markov Chain Monte Carlo (MCMC)-based Bayesian inference; biological, chemical, DNA, dynamical, genetic, immuno-, and membrane computation; and nature-inspired, self-organized criticality and active walker (swarm and ant intelligence) models. The computations are interpreted as the outcome arising out of deterministic, nondeterministic, or stochastic interaction among elements in a multiset object space that includes the environment. These interactions are like chemical reactions, and the evolution of the multiset can mimic biological evolution. Since the reaction rules are inherently parallel, any number of actions can be performed cooperatively or competitively among the subsets of elements so that the elements evolve toward an equilibrium or an emergent state. Practical realization of this paradigm is achieved through a coordination programming language using Multiset and transactions. This paradigm permits carrying out parts or all of the computations independently in a distributed manner on distinct processors and is eminently suitable for cluster and grid computing. Some important applications of this paradigm are described.

* Research was supported by the National Sciences and Engineering Research Council (NSERC) Canada.

1 INTRODUCTION

Most systems we observe in nature are complex dynamical systems that consist of a large number of degrees of freedom. Further, they may contain several inhomogeneous subsystems that are spatially and temporally structured on different scales and characterized by their own dynamics and interacting with each other in a complex manner. Such complex systems often exhibit collective ("Emergence") behavior that is difficult to model deterministically, based entirely on the properties of the individual subsystems. Probabilistic models such as stochastic dynamical systems provide an efficient methodology for modeling such complex systems by capturing the average behavior of the system at different spatial and temporal scales. Despite the relative simplicity of a probabilistic model (in terms of the number of degrees of freedom) compared with a deterministic model, the dynamics of the resulting stochastic system are often highly nonlinear, implying that it is not possible to obtain analytical expressions for the distributions or statistics such as mean and variance. Thus we need to resort to Monte Carlo simulation-based methods to compute estimates of the distribution and statistics. These Monte Carlo estimators are based on the strong law of large numbers, which under suitable regularity conditions (e.g., geometric ergodicity of the dynamical system) states that the arithmetic mean of the simulated random samples converges strongly (with probability one) to the true statistic. This simulation-based approach for computing estimates of the distribution and statistics of a stochastic dynamical system can be viewed as "*soft computation*," since unlike conventional computation, where exactness is our goal, we allow here for the possibility of error and randomness. Soft computation needs to be supported by a suitable choice of a datastructure and an associated programming paradigm. It is the object of this chapter to describe a unifying programming paradigm for carrying out soft computation in complex systems.

A programming paradigm is a class of programs that solve different problems while having the same control structure [6, 7, 70]. This permits us to write a generic program—called a *program shell*—that implements the common control structure. The program includes a few unspecified data types and procedures that vary from one application to another. We can then devise a parallel application program by substituting the types and procedures needed for a specific application.

In this chapter we describe a unified multiset programming paradigm (UMPP) that constitutes a unifying theme for several widely used computational schemes. These include all conventional algorithms, Markov chain Monte Carlo (MCMC) [26], Particle filters [27], *evolutionary algorithms*, such as classifier systems, probabilistic bucket brigade learning [19], *genetic algorithms* [8, 12, 29, 30, 32, 61, 62], *genetic programming* [42], *membrane computing* [74], *immunocomputing* [33], *Self-organized criticality* [10], and *active walker models* (ants with scent or multiwalker-paradigm, where each walker can influence the other through a shared landscape), also called *Stigmergy* [9, 11, 18, 23, 24, 38, 39, 60, 56, 40] *stochastic marked point processes* [78, 81], *random graph models* [15, 25], *biomimicry* [72], and *DNA computing* [2, 58]. Also it is applicable to nonequilibrium systems interacting with surroundings [63, 76] using feedback mechanisms involving catalytic reactions—as, for example, the production of ATP (adenosine triphosphate) from ATP. These lead to the necessity for the *open-world hypothesis* (rather than the closed-world

assumption used in logic) to discover newer unknown possibilities, e.g., self-organization and active walks (swarm intelligence: see [11, 23, 39, 56]).

1.1 Structure of Unified Multiset Programming Paradigm (UMPP)

The UMPP proposed in this chapter consists of the following features [65, 37, 41, 80] that mimic evolutionary behavior in a biological system and innovative aspects of other nonclassical computational schemes [85]:

1. One or more object spaces that contain elements whose information is structured in an appropriate way to suit the problem at hand (e.g., the genomic library).
2. A set of interaction rules that prescribes the context for the applicability of the rules to the elements of an object space. Each rule consists of a left-hand side (a pattern or property or attribute) of named objects and the conditions under which they interact, and a right-hand side that describes the actions to be performed on the elements of the object space, if the rule becomes applicable, based on some deterministic or probabilistic criteria. For example, techniques in the *lock-key paradigm* [20, 21] abstracted from shape matching in molecular biology form the basis for molecular computing. The lock-key paradigm arises from the notions of complementarity and the union of opposites that pervade the entire science and natural philosophy. It is based on the recognition of an object (molecule) through complementary shape matching.
3. A control strategy that specifies the manner in which the elements of the object space will be chosen and interaction rules will be applied, the kinetics of the rule-interference (inhibition, activation, diffusion, chemotaxis), and a way of resolving conflicts that may arise when several rules match at once. This is analogous to the selection strategy (possibly stochastic) in a biological system.
4. A mechanism to evaluate the elements of the object space in order to determine the effectiveness of rule application (e.g., evaluating fitness for survival in a biological system).

Thus, UMPP provides a stochastic framework of *generate and test* for a wide range of problems [92, 62]. Also, the system structure of UMPP, consisting of components and their interaction, is supported by contemporary software architecture design [5].

1.2 Computational Features of UMPP

The UMPP has the following computational features:

1. Interaction-based: The computations are interpreted as the outcome of interacting elements of the object space that produce new elements (or the same elements with modified attributes) according to specific rules. Hence, the intrinsic (genotype) and acquired properties due to interaction (phenotype) can both be incorporated in the object space. Since the interaction rules are inherently parallel, any number of actions can be performed *cooperatively or competitively* among the subsets of elements so that the new elements evolve toward an equilibrium or unstable or chaotic state. Such an evolution may retain certain invariant properties of the attributes of the elements.

2. Content-based activation of rules: The next set of rules to be invoked is determined solely by the contents of the object space, as in the context of chemical reactions. This feature is very powerful, since it provides for automating the discovery of solutions, as in genetic programming [42].
3. Pattern matching: Search takes place to bind the variables in such a way as to satisfy the left-hand side of the rule. It is this characteristic of pattern (or attribute) matching that gives the rule-based paradigm its distinctive capabilities for innovative computing [36, 85, 91].
4. Simplicity of implementation: The implementation involves three basic tasks:
   a. Searching for elements of the object space satisfying the interaction condition
   b. Carrying out the action to these elements, ensuring that certain invariants hold before and after the actions
   c. Evaluation of the new elements in the object space for the required termination or equilibrium or self-organized criticality or emergent states
5. Suitable for deterministic, nondeterministic, and probabilistic evolutionary modes: The object space mentioned above is analogous to phase space in dynamical systems. It permits the introduction of a probabilistic formulation in rule applications. As each element of the ensemble changes over time, its phase point is carried into a new phase point. The evolution of the resulting probability cloud (e.g., probability mass function associated with the discrete phase points) in phase space corresponds to a distributed probabilistic computation. Thus this paradigm is suitable for deterministic exact computation when the initial conditions are exactly specified and the evolution is governed by a deterministic system. It is also suitable for approximate probabilistic computation when the initial conditions and interactions are not complete and not well specified. The probabilistic computation mode is useful when one wants to derive macroscopic or bulk properties of matter from the rules governing a large number of objects, as in a statistical mechanical system interacting with an environment.
6. Choice of objects and actions: We can use strings, arrays, sets, trees and graphs, multisets, tuples, molecules, particles, and even points as the basic elements of computation and perform suitable actions on them by defining a suitable topology, geometry, or metric space. Accordingly, this approach is widely applicable to several innovative computing approaches.

The rest of this chapter is organized as follows. In Sections 2 and 3, general properties and those of rule-based paradigms are developed. In Section 4, we use these properties to give a complete description of the UMPP. In Section 5, examples are given that are completely modeled by UMPP. These are, respectively, Markov Chain Monte Carlo methods, Classifier/Bucket Brigade systems, Genetic Algorithms, Genetic Programming, Oscillatory Chemical Reactions, Swarm and Ant-Colony techniques, and Conrad's Lock-Key Paradigm, membrane and immunocomputing, and quantum field theory. In Section 6, UMPP is interpreted in terms of relational databases. Section 7 explains the molecular DNA computation using multiset datastructure. Section 8 presents some simulation examples of adaptive learning algorithm for Nernst potential, and adaptive spreading code optimization in wireless CDMA systems. Section 9 contains some concluding remarks.

2 DEFINITIONS AND FORMALIZATION

We define the following terms:

System. A system is a set of objects together with relationships between objects and their attributes.

Environment. For a given system, the environment is the set of all objects, a change in whose attributes affects the system, and also those objects whose attributes are changed by the behavior of the system.

Specification. We also define a specification of a *deterministic computation* as a description that, when executed, would transform the given input object space into a desired output object space satisfying the prescribed attributes.

The main feature of the general rule-based paradigm is the specification of the program:

$G(R, A)(M)$ = If there exists elements $a, b, c, \ldots$ in an object space M such that an interaction rule R $(a, b, c, \ldots)$ involving elements a, b, c is applicable, then $G(R, A)((M\text{-}\{a, b, c, \ldots\}) + A(a, b, c, \ldots))$; else M.

Here M denotes the initial object space. This is a multiset or a bag in which a member can have multiple occurrences [14]. The sign "–" denotes the removal (annihilation) of the interacted elements; it is the multiset difference. The sign "+" denotes the insertion (or creation) of new elements after the action A; this is multiset union. Note that R is a condition text (or an interaction condition that is a boolean) that determines when some of the elements of the object space M can interact. The function A is the action text that describes the result of this interaction.

The function R can be interpreted as the query evaluation function in a database M, and the function A can be interpreted as the updating function for a set of database instances. Hence, if one or several interaction conditions hold for several nondisjoint subsets at the same time, the choice made among them can be nondeterministic. This leads to competitive parallelism. However, if the interaction condition holds for several disjoint subsets of elements in the database at the same time, the actions can take place independently and simultaneously. This leads to cooperative parallelism.

Deterministic Iterative Computation. This paradigm is a deterministic iterative dynamic computation consisting of applications of rules that consume the interacting elements of the object space and produce new or modified elements in the multiset. This is essentially *Dijkstra's Guarded Command Program* [45]. It is well known that the Guarded Command approach serves as a universal distributed programming paradigm for all conventional algorithms with deterministic or nondeterministic components [70]. So we will not elaborate on this aspect any further.

Termination. To achieve termination of rule application, the interaction conditions R have to be designed so that the elements in the object space can interact only if they are in opposition to the required termination condition. When all the elements meet the termination condition, the rules are not applicable and the computation halts, leaving the object space in an equilibrium state (or a fixed point of the iterative dynamics).

Nontermination, instability and irreversibility. These cases arise when the rules continue to fire indefinitely. Then the object space can be in a nonequilibrium state. It is also possible that the evolution of the system is chaotic.

As an example, consider the rule-based iterative deterministic dynamical system:

For $X(0)$ in the range $[-1,1]$,
if $X(i) \geq 0$ then $G(X(i+1)) = -2X(i) + 1$;
else $G(x(i+1)) = 2X(i)+1$

The two rules for $X(i) \geq 0$ and $X(i) < 0$ are mutually exclusive and do not compete. This generates a chaotic dynamical system that is unstable and has a dense orbit in the interval $[-1,1]$ [17, 56].

3 TYPES OF RULE-BASED SYSTEMS

Several types of rule-based systems are used in computer science [31, 41, 65, 80]:

1. Monotonic. Here the application of one rule does not prevent or interfere with the application of another rule that could have also been applied at the time when the first rule was selected.
2. Nonmonotonic. Here the application of one rule interferes with the application of another rule.
3. Partially commutative. If the application of a particular sequence of rules transforms the system from state 1 to state 2, then any interleaved set of rules in the sequence would equally well create the same transformation from state 1 to state 2.
4. Commutative. A system that is both monotonic and partially commutative is called *commutative*.

Commutative systems are useful for problems in which changes occur but can be reversed and the order in which operations occur is not critical. Non-partially commutative production systems are useful when irreversible changes occur; here the order is important.

The implementation of a production system operates in three-phase cycles: matching, selecting, and execution. The cycle halts when the elements of the database satisfy a termination condition. The task of match phase is similar to query matching—that is, unification of the rules with the database. This phase returns a conflict set that satisfies the conditions of different rules. In the select phase, we select those compatible rules after conflict resolution. In the execution phase, all selected rules are fired and actions are implemented.

Parallelism can be achieved in the matching and execution phases as follows:
In the matching phase, we can

1. Match in parallel several partitions of the rule set
2. Match several partitions of the object space

In the execution phase, we can

1. Execute several rule actions on the object space elements if these are independent (interrule actions)
2. Execute several instantiations of the same rule simultaneously (intrarule actions)

3.1 Kinetics of the Multiset Rule-based Systems

In order to speed up the use of UMPP, we need to consider how to permit multiple rule execution concurrently. This offers the possibility of carrying out parts or all of computations in parallel on distinct processors or performing multiple simulations simultaneously in a grid or cluster computing environment. Such possibilities would require the analysis of how the rules interfere with each other. There are three ways in which the rules can interfere [35, 37, 41, 55, 43, 45]. We call these interference rules *Turing's kinetic rules*, as they are similar to those enunciated by Turing [88] to describe the development of shape, form, and pattern in organisms (chemical morphogenesis rules: see [69]).

1. **Enabling dependence (ED)**. Rule i and rule j are called *enable dependent* if the application (or firing) of rule i updates (writes, or W) the elements of the object space, and creates the required precondition that is read (R) by rule j and causes it to fire. As a special case, the update can be either insertion ($W+$) or deletion ($W-$) of elements, and the precondition of rule j to fire is respectively the presence ($R+$) or absence (R) of those identical elements. (In parallel programming, these WR, $W+R+$, $W-R-$ types of dependencies are called *dataflow dependence*).
2. **Inhibit dependence (ID)**. Rule i and rule j are called *inhibit dependent* if the application (or firing) of rule i updates (W) the elements of the object space, and disables the required precondition that is read (R) by rule j and prevents it from firing. As a particular case, the updates can be either insertion ($W+$) or deletion ($W-$) of elements, and the precondition of rule j to fire can be respectively the absence ($R-$) or presence ($R+$) of those elements. The WR, $W+R-$, or $W- R+$ types of dependencies are called *inhibit dependencies*.
3. **Opposition dependence (OD)**. Rule i and rule j are said to be *opposition dependent* if the following situation holds. Rule i updates (W) or deletes ($W-$) or adds ($W+$) elements, while rule j respectively overwrites (W) or simply adds ($W+$) or deletes (W) the same elements. (In parallel programming, this WW type of dependence is called *data-output dependence*).

The rules are called *compatible* if they are not inhibit dependent (ID) and not opposition dependent (OD). The communication among the objects takes place through ED and ID. Note that a rule can enable (be autocatalytic) or inhibit itself.

We can relate the parallelism in production rules with vector, pipeline, and data parallelism thus:

1. Vector parallelism. If all the rules are compatible, then we can apply all the rules simultaneously, e.g., a vector addition.
2. Pipeline parallelism. Here multiple rules are fired in parallel and passing data in a pipeline fashion, e.g., multienzyme reactions, where at each membrane an "imprisoned" enzyme performs a given operation and then sends it on to the next stage [76].
3. Data parallelism. Multiple instantiations of the same rule are fired in parallel based on distinct data, e.g., forming matrix products.

The rule-based paradigm can be supported by a database transaction processing system if we identify the condition text with a database query evaluation function (to find those elements or subsets of elements of the database satisfying particular conditions) and the action text with the updating operation in the database. Such identification relates the rule-based programming style and the

database transactional programming style [59]. When one or more reaction conditions hold for several disjoint subsets at the same time, the query or Read (R) operation and the update (W) operation can take place concurrently. This parallelism corresponds to *cooperative parallelism*.

If, however, one or more conditions hold for nondisjoint subsets of the database, then a transaction is chosen among the alternatives either nondeterministically or probabilistically, as dictated by a random number generator. The actions on the chosen subset are executed atomically and committed. In other words, the chosen subset undergoes an *asynchronous atomic update*. This ensures that the process of matching and the follow-up actions satisfy the four important properties called *ACID properties* [49]: Atomicity (indivisibility and either all or no actions carried out), Consistency (before and after the execution of a transaction), Isolation (no interference among the actions), and Durability (no failure). Once all the actions are carried out and committed, the next set of conditions is considered. As a result of the actions followed by commitment, we derive a new database; this may satisfy new conditions of the text, and the actions are repeated by initiating a new set of transactions. This set of transformations halts when there are no more transactions executable or the database does not undergo a change for two consecutive steps, indicating a new consistent state of the database. Such a scheme would correspond to *competitive parallelism*.

The implementation of a rule based-system for mathematical problems requires that the application of the rules eventually terminates. Termination for a rule set is guaranteed if rule processing always reaches a stable state in which none of the rules will be enabled to react. However, note that rule processing does not terminate if rules provide new conditions to fire indefinitely—that is, if actions of *Ri* create the right conditions for *Rj* to fire. This would correspond to cyclic computations and could lead to circularity or a deadlock situation.

3.2 Closed-World Assumption and Closed Systems

In conventional computer programming, we usually choose a commutative production system that is sure to lead us to termination of the program corresponding to a fixed point. This is achieved by choosing the positive world of facts in which any fact that is not present (or cannot be derived from other assertions) is assumed to be false. This assumption is called the *Closed World Assumption (CWA*; [31]). It is an important assumption used in database design that is based on first-order logic. The Gamma paradigm described by Banatre and Me'tayer [6] is based on the closed-world assumption and corresponds to a commutative production system; here we do not check for facts that are not present or derivable. If CWA does not hold, we have a nonmonotonic system. CWA is valid only in first-order logic or for Horn clauses in second-order logic [31].

First-order logic has the three important properties:

1. Completeness. It is complete with respect to the domain of interest. That is, all facts needed to solve a problem are present in the system or can be derived from the given rules.
2. Consistency. All axioms are contradiction free.
3. Monotonicity. The only way it can change is that new facts can be added as they become available.

If these new facts are consistent with all other facts that have been asserted, then nothing ever will be retracted from the set of facts that are known to be true.

Formal reasoning requires the CWA to specify what can be produced from the rules. This means that a formal system, no matter how well constructed, will not be able to model the changes in a nonstationary world. Therefore, if any of the above properties are not satisfied, conventional first-order logic cannot be applied and we need to use nonmonotonic logic.

Classical dynamics (also called *rational mechanics*) uses the laws of reasoning based on CWA and a monotonic production rule system. Given a well-defined initial state, we can precisely compute the evolutionary trajectory in the phase space and a well-defined final state of the system. This is because such a system is characterized by a deterministic set of equations (or rules) of motion, we have complete knowledge about the system, and no other unknown fact exists. Hence it is a closed system that is reversible or invariant under the transformation of a positive to negative time coordinate. Such a closed system does not usually interact with the environment and there is no energy or mass exchange outside the system [76].

CWA assumes that anything that is not necessarily true should be assumed to be false. It has two limitations:

1. It only operates on individual predicates and not on interactions among them.
2. It assumes that all the predicates have their instances listed. This is true perhaps in the database context but not otherwise. But, in general, we cannot completely describe all predicates, and the assumption that the world is closed is not valid—we must assume that the world is open.

Nonmonotonic systems [1] can contain empirical statements and plausible set of rules that make the system open.

3.3 Open Systems

An open system, unlike a closed system, is characterized by a system interacting with an environment. Its evolution is *not governed* by a deterministic set of equations of motion, and we are usually concerned about the average behavior of the system as it evolves. For the open system, we need to introduce probabilities due to the following reasons:

1. Ignorance of the relevant variables and functions involved in the rules to represent a given problem domain.
2. Inadequacy of the rules to model a given system and its environment, since we do not know whether an object belongs to a system or whether it belongs to an environment or how to subdivide the objects to establish a dichotomy of sets of related objects into a system and an associated environment.
3. The introduction of a probabilistic approach permits us to take into account all possible sequences of events into the future, from the most to the least probable.

Open systems interact with the environment accompanied by an exchange of energy, entropy, and matter. These systems are characterized by a nonmonotonic production system in which we need to discover new rules that may violate old rules. Examples of such systems are thermodynamical and nonequilibrium systems, including chaotic and self-organized critical systems; and non-Markovian

active-walker models, where the system and the environment interact with each other. Such systems need probabilistic modes of computation to account for ignorance of some relevant variables or functions and inadequacy of the rule sets due to interaction with the environment. This computation is imprecise, reflecting the average behavior of the system [66].

As described in [71] and [47], closed systems with no dissipation of energy have zero metric entropy, while open systems are dissipative and have a positive metric entropy. Thus we have two major classes of systems or machines, ordinary (O) and dissipative (P), which are based on metric entropy as described below:

Ordinary or Zero Metric Entropy Machines (O)

These are completely structured, deterministic, exact behavior (or algorithmic) machines.

This class contains the machines in Chomskiian hierarchy [43]:

1. *Finite State machines: obey regular grammar or type 3 grammar;*
2. *Push down-stack machines: obey context-free grammar or type 2 grammar;*
3. *Linear bounded automata: obey context-sensitive or type 1 grammars;*
4. *Turing Machines that halt: obey an unrestricted or type 0 grammar, and*
5. *Exactly integrable Hamiltonian flow machines*.

Such machines are, in principle, information lossless; their outputs contain all the required information as dictated by the programs. Further, the fixed point of a terminating Turing computation is an analogue of an attractor or equilibrium point in an integrable Hamiltonian system.

Positive Metric Entropy Machines (P)

The Lyapunov exponent of a dynamical system is a measure of the sensitivity of the state of the system to its initial conditions. A nonlinear dynamical system with an attractor that has a positive Lyapunov exponent exhibits chaotic behavior—the attractor is exponentially sensitive to the initial condition of the system. Such systems are analogues of a problem that stands at the border of computability and noncomputability, where we do not know whether the computation halts or reaches a fixed point. To be at the edge of computability is analogous to entering a route to chaos in dynamics. Thus chaos in dynamical systems and noncomputability can be considered as parallels in their respective domains. Undecidable (nonterminating) partially recursive schemes also exhibit chaoslike behavior, such as lack of predictability and decidability! In fact, it is suspected that deterministic chaos corresponds to Godel's undecidability.

Nonintegrable positive entropy machines exhibit various degrees of irregular dynamics:

1. *Ergodicity*. Here the set of points in phase space behave in such a way that the time-average along a trajectory equals the ensemble average over the phase space. Although the trajectories wander around the whole phase space, two neighboring initial points can remain fully correlated over the entire time of evolution. Ergodicity in a dynamical system is a result of nonintegrable perturbation in an integrable system [57]. The term *ergodicity* in dynamical systems means *statistical homogeneity*. This means the trajectory starting from any initial state can access all other states in the phase space.

2. *Mixing*. The initial points in the phase space can spread out to cover the space in time but at a rate weaker than the exponential (e.g., inverse power of time).
3. *Bernoullicity, K-flow, or chaos*. The trajectories cover the phase space in such a way that the initially neighboring points separate exponentially and the correlation between two initially neighboring points decays with time exponentially. It is with this class of irregular motion that we define classical chaos. These trajectories lie on the border and beyond the Turing computable region; that is, they belong to partial recursive schemes leading to undecidability.
4. *Nonequilibrium systems*. These systems exhibit emergent behavior, such as chemical and biological machines and living systems.

Each of the above properties implies all the properties above, e.g., within a chaotic region the trajectories are ergodic on the attractor and wander around the desired periodic orbit. Classical motion is chaotic if the flow of the trajectories in a given region of phase space has positive Lyapunov exponents that measure the rate of exponential separation between two neighboring points in the phase space. Chaos indicates hypersensitivity on the initial conditions. Also, the system becomes inseparable (metric transitivity), and the periodic points are dense. That is, the whole dynamical system is not simply a sum of parts; it functions as a whole, leading to what is known as *emergence*. Also, strange attractors with fractal dimensions govern such dynamic systems!

Thus, to simulate open systems, we need to combine zero and positive entropy machines to carry out computation and also provide environmental interaction. This can be achieved by the introduction of entropy through random choices.

The introduction of positive entropy through the injection of either chaoticity (deterministic randomness) or stochasticity (statistical randomness) has several advantages [44, 46, 50, 67]:

1. It provides ergodicity of search orbits. This has the property that every point in the set of accessible states is approached arbitrarily closely during the iteration. This property ensures that searching is done through all possible states of the solution space, since there is a finite probability that an individual can reach any point in problem space with one jump.
2. It provides solution discovery capabilities (as in genetic programming) due to embedded randomness [82]. This property arises due to the fact that chaotic orbits are dense and have a positive Lyapunov exponent, two initially close orbits can separate exponentially from each other.
3. It cuts down the average running time of an otherwise worst–case running time-algorithm. We pay for this gain by producing an output that has an error with a small probability. Accordingly, we cannot claim that the solutions would always exist, and even if they exist, they are exact.
4. It can solve problems of high complexity by facilitating cross-fertilization across discipline; e.g., genetics (genetic algorithms), thermodynamics (simulated annealing), statistical mechanics (particle transport), and complex systems (active-walker, self-organization, and percolation models).
5. Also, stochastic mechanisms plays a vital role in many physical processes involving motion of particles: e.g., mechanisms such as diffusion, aimless drift of particles, convection, annihilation of particles from the population, and creation of particles. These mechanisms change the local density of the population. Numerous physical and social systems behave in this manner.

Remarks

1. It is possible that quasi-ergodic behavior arises, resulting in the entrapment of the orbit in isolated regions. This behavior can be avoided by using perturbations to the chaotic orbit that are highly sensitive to initial conditions or by using more than one Markov chain, with the initial states reasonably apart.
2. Prigogine [76] suggests the use of nonunitary transformations, called *star-Hermitean operators*, to extend the capabilities of computational systems to reflect average behavior. That is, the tools of both equilibrium and nonequilibrium quantum statistical mechanics are needed to create open systems.

3.4 Simulating Open Systems

A way to simulate a mixture of the zero and positive entropy machines is by choosing the mode of application and the action set of a rule-based program to be either deterministic, nondeterministic, probabilistic, or fuzzy. Rule application policy in a production system can be modified by

1. Assigning probabilities/fuzziness for applying the rule
2. Assigning strength to each rule by using a measure of its past success
3. Introducing a support for each rule by using a measure of its likely relevance to the current situation.

The above three factors provide for competition and cooperation among the different rules. In particular, the probabilistic rule system can lead to emergence and self-organized criticality. Thus, the capabilities of class O machines can be enlarged by simulating special features of class P machines—using nondeterminism, randomness, approximation, probabilities, equilibrium statistical mechanical (e.g., simulated annealing), and nonequilibrium statistical mechanical (e.g., genetic algorithms and the Ant algorithm) approaches.

We will describe in Section 4 how the probabilistic rule-based paradigm can simulate the open system.

4 THE STOCHASTIC RULE-BASED PARADIGM

In every closed logical, physical, chemical, or biosystem, certain properties (or attributes) do not change (are conserved) or remain invariant when the system evolves over time, moving from one state to another. Such attributes are called *invariants* and play an important role in the specification of the system. A deterministic rule-based paradigm in a closed system ensures that when an interaction triggers an action, certain specified invariants always hold before and after the actions. However, in open systems—such as complex systems, which consist of a large number of simple elements interacting with each other and the environment—new properties such as self–organized criticality and the active walk system can emerge [56]. Such systems are dissipative, do not necessarily satisfy predetermined invariant conditions (such as conservation of certain specific properties), and need to use probabilistic rule selection and modification. Here, the probabilistic rule paradigm plays an important role [83].

In Section 3 we discussed the necessity for the introduction of probabilistic variant of the production rule paradigm for nonmonotonic or open systems. This paradigm is obtained by introducing probabilities for selection when one or more reaction conditions hold for several nondisjoint subsets at the same time. In this case, the choice made among these subsets is determined by a random number generator that selects the ith possible subset with a probability $p(i)$ to perform the required actions, thus providing for probabilistic competition among the different choices. This results in the Unified rule-based Multiset Programming Paradigm (UMPP) and is defined by the function

$PG\,(R(p(i), A)\,(M)$ = if there exists elements $a, b, c,\ldots$ belonging to an object space M (a multiset) such that $R(a, b, c,\ldots)$, then $G(R,A)((M\text{-}\{a, b, c,..\}) + A(a, b, c,..))$, else M,

where each of the possible number of subsets i that satisfy the conditions R is chosen with a probability $p(i)$ and the corresponding text of action A is implemented. Note that the sum of $p(i)$ equals 1. Also, when $p(i)$ is not specified, the choice can be deterministic or nondeterministic. Thus UMPP can contain within itself the deterministic, nondeterministic, and probabilistic components.

The UMPP is useful in many ways:

1. It can be used to realize evolutionary algorithms such as classifier systems, probabilistic, bucket brigade learning, the genetic algorithms [8, 12, 29, 30, 32, 61], self-organized criticality, and active walker models—ants with scent or multiwalker-paradigm, where each walker can influence the other through a shared landscape based on nondeterministic or probabilistic action [11, 23, 56].
2. The multiset datastructure used in UMPP is suitable to describe physical events. It can represent pointlike variables in physics (time, space, velocity, or other quantity) or discreteness of events intrinsic to the physical processes (intrinsic point processes) or arising out of observations (observational point processes) [64, 78, 81]. Also it can represent iterative dynamical systems, including cellular automata [36, 91] and evolving networks [25].
3. It can support the design of a wide variety of programs to seek answers to questions such as:
   a. Which is the most likely state that a system will reach if supplied with a given input sequence?
   b. What is the average survival time of a population that is subject to reproduction and death of its members?
   c. How long does it to take to learn a particular concept?
   d. Can a system reach self-organized criticality?
   e. Can a system become chaotic?

4.1 Properties of UMPP

The nondeterministic as well as probabilistic computations are organized in two phases [44]:

Phase 1: Guessing or tossing (random choice)

Phase 2: Evaluation and verification of the validity of the result

These two phases work interactively. Thus nondeterministic and probabilistic computations are no more than guess-check and toss-check actions. In the guessing

or tossing phase, we apply certain reaction rules probabilistically to individual elements satisfying the required conditions and perform the required actions. In the verification phase, we evaluate either the individual elements of the database or a selected subset or the whole database using some acceptance criteria.

As mentioned in Section 2, the deterministic and nondeterministic UMPP programs are based on two-valued logic, and they terminate when the interaction conditions (guards) are false [45]. But to determine the speed of convergence and termination of the probabilistic UMPP paradigm, we need to use probabilistic arguments. In practice, to detect a fixed point (or equilibrium), we need to use some acceptance criteria, and at the end of each trial evaluate an individual element or a selected subset or the whole object space, to decide whether to repeat the trial or to halt. That is, the evaluation of the object space can take place at different levels of granularity depending upon the problem domain. Also, the acceptance criteria may be chosen dependent on or independent of the number of previous trials, and the choice of probabilities can remain static or can vary dynamically with each trial. Thus, depending upon the evaluation granularity, acceptance criteria, and the manner in which the probability assignments are made, we can devise various strategies. We will give examples of these in Section 5.

4.2 Iterative Dynamics of UMPP

In Section 2 we described the deterministic iterative scheme that can either lead to a fixed point for closed systems or exhibit aperiodic and chaotic behavior or self-organization when applied to open systems. In the stochastic approach, we need to deal with stochastic difference or differential equations. For search and optimization problems, one could use either chaoticity or stochasticity in iterative schemes to create ergodicity. While it is still not known whether chaoticity or stochasticity is superior in computational performance, the stochastic method seems to be more easily amenable for proof techniques and seems to be more robust under dynamic noise. This is a major research area currently [17]. Since UMPP can deal with deterministic, nondeterministic (deterministic random and chaoticity), and stochasticity, it permits simulating a variety of schemes, namely, piecewise deterministic, piecewise stochastic, and nondeterministic systems encountered in time-varying systems, and point processes that have a variety of applications [78, 64, 81].

UMPP provides a suitable model for understanding a large class of evolutionary events. Such a model is applicable to very wide areas in biological and social systems that are characterized by different kinds of attractors belonging to four classes, called *Wolfram classes* [91]:

1. Evolution to a fixed homogeneous state in living systems (limit points in dynamical systems) corresponding to fixed points in programming
2. Evolution to simple separated periodic structures in living systems (or limit cycles in dynamical systems) corresponding to competitive cycles of deadlock or livelock in concurrent computation
3. Evolution to chaotic behavior, yielding aperiodic patterns in living systems (strange attractors in dynamical systems) that have no correspondence in computer science

4. Evolution to complex patterns of localized structures in living systems, which have no analog in dynamical systems or in computer science.

In addition, phase transitions can arise between the various classes. For example, between periodic and chaotic behavior there is a phase transition. While the periodic and chaotic regions are governed by rules, the transition region is not governed by any rules. Thus one proceeds through a complexity hierarchy from simple to complex upto the transition region and beyond that complex to simple dynamics. The phase transition therefore separates the space of computation into an ordered and a disordered regime, which can be thought to correspond with halting and nonhalting computations. The transients grow very rapidly in the vicinity of a transition between ordered and disordered dynamics. Dynamics in the vicinity of the phase transition gives rise to a critical slowing down, and the various complexity classes (constant, linear, polynomial, exponential) are encountered. Critical slowing down of a system appears like the exponential slowdown in computing the solution of an intractable or nonpolynomial time-solution problem.

Phase transition-like phenomenon arises in a wide variety of algorithms and heuristics used for search problems in the NP class or beyond [96] and in random graph models [15]. Although simulation is a poor substitute, there seems to be no other way to guess the threshold of certain properties like phase transition in complex systems. However, no satisfactory conclusions have been arrived at so far to distinguish NP-complete problems or many other similar properties from the phase transition point of view.

5 REALIZATION OF UMPP AND EXAMPLES

Practical realization of the UMPP can be achieved through a coordination programming language using Multiset and transactions; the design details will be published elsewhere.

5.1 Markov Chain Monte-Carlo and Randomized Grid Bayesian Inference

The Markov Chain Monte Carlo (MCMC) methods [79], which include simulated annealing, data augmentation, and Metropolis Hastings-type algorithms, are used to construct the *a posteriori* probability density function of a random dynamical system. The UMPP proposed in Section 4 can realize such MCMC methods as follows: the first condition text (say, *R /pi*) prescribes the nature of random variable (i.e., probability distribution function) to be used for the selection of the elements of the database and also a set of *deterministic criteria* for the acceptance or rejection of the elements. The corresponding action text (*A*) implements these conditions and accepts or rejects the elements as and when they are randomly generated (on the fly). Following the rejection of elements, the second condition text (say, *R* /p*i*) prescribes a *probabilistic criterion* to accept (or reject) some of the earlier rejected elements; the corresponding action text *A** implements these conditions. The condition text *R* /p*i* is then varied as a function of the current number of trials and a parameter called

temperature [61]. The effect of this variation is such that the probability of accepting a new solution that is worse than the current solution decreases with the degree of the deterioration of the solution, and more significantly with the run time of the method.

More recently, a recursive (real-time) MCMC algorithm called a *particle filter* [27], has been developed. This algorithm iteratively updates the currently available set of particles into a new set of particles so that the empirical probability distribution of the particles closely follow the true distribution. That is, the simulated evolution of the particles mimics the real system. This algorithm is based on randomized grids for propagating the conditional density of the state of a dynamical system given noisy observations. Then sequential importance sampling, together with the Bayes rule, is used to update the weights of the grid points from a priori distribution to a posteriori distribution. In order to avoid degeneracy problems, a random resampling method is used to eliminate low-probability points in the grid. This step mimics evolution in the sense that it eliminates most poorly adapted species (selective extinction or death). It is clear that these steps can be completely captured by the UMPP. In MCMC methods, the granularity of the evaluation takes place at the elemental level, and hence these methods permit on-the-fly acceptance of elements, thereby providing high concurrency in the implementation.

The use of Multiset facilitates the realization of the Multiple Particle Filter approach recently described by Yuen and MacDonald [94]. It can be computationally advantageous by splitting a multidimensional problem into multiple low-dimensional ones if there is sufficient degree of independence among the components in the estimation problem.

5.2 Classifier / bucket-Brigade systems

The classifier system [8, 12, 29, 30, 32, 61] is a parallel rule-based production system based on two-valued logic. Each classifier can be regarded as a separate instruction that takes messages as input and produces messages as output. As in a production system, there is a match cycle, where each rule is matched against the state of the short-term memory containing a message list M. If its preconditions are satisfied by at least one message, then the classifier is activated, and an execute cycle carries out the required actions and posts an external message. All external communications are via the message list. Thus all internal control and external communication reside in the same data structure. The classifier system can therefore be realized by the rule-based model, if the messages are represented by a database of appropriate type. When the classifier system is used deterministically, it iterates for a fixed number of times or until the message list does not change for two successive atomic steps. It has been shown in Forrest [29] that

1. any finite function can be computed by some classifier system in a single match-and-execute step, with an arbitrary amount of parallelism, by distributing the representation of the function over enough number of processors, and
2. a classifier system can be made to behave completely sequentially.

These two properties permit the classifier system to behave with maximal parallelism or completely sequentially. Thus a classifier system offers a flexible for-

malism that permits optimization of the three parameters, namely, the number of processors (classifiers), the length of computation, and the amount of interprocessor communication.

In the Bucket-Brigade system (BBS) [8, 12, 29, 30, 32, 34], the classifier system is generalized by introducing two factors, called the *strength* and the *activation* of each rule based on its relevance and support from other rules.

1. *Strength of the individual rule*. The strength of each rule is based on its success.
2. *Support and relevance of the rule*. Each rule is evaluated by its likely relevance to the current situation and support from other rules.

Here, the first rule reflects the success of the individual rule, while the second rule reflects the performance of the collection of the different rules acting as a whole, by evaluating the relevance of each individual and providing support for its activity.

These two factors, namely, the strength and the support, provide for competition and cooperation among the different rules. When a posted message matches a rule, each classifier makes a bid proportional to its strength; hence, highly fit rules are given preference. The bid made by an activated classifier is then proportionately divided and sent as a reward to other classifiers earlier responsible for activating it. Thus if a rule is instrumental in permitting other rules to fire *favorably*, it receives payoff and hence its strength will be increased; otherwise, its strength is decreased due to bidding. In the steady state, the strength remains invariant. BBS is useful to construct a database whose attributes are not known in advance but are adaptive or evolving probabilistically. The UMPP can realize the BBS, if the probabilities p_i are replaced by strengths and at the end of each atomic update, the actions are evaluated and the strengths are reassigned depending upon downstream and upstream payoff.

The ant algorithm and swarm intelligence use an approach similar to BBS (see subsection 5.7 below).

5.3 Genetic Algorithm

The genetic algorithm goes a step further than the BBS: after selecting the most successful rules, these rules are combined by selecting pairs and performing crossover and mutation. The crossover may be thought of as a combination of independent rules, while the mutation may be thought of as an error in input or an approximation.

The genetic algorithm [8, 12, 29, 30, 32, 68, 13, 62, 84, 83, 90] chooses an initial population of objects called *chromosomes*, represented by a multiset M of binary strings with a length of m bits. Then it performs on M three different probabilistic operations that mimic the operations found in nature, namely, selective reproduction, crossover, and mutation. These operations can therefore be represented in the UMPP using four transactions:

Transaction 1 evaluates the given generation for assigning the selection probabilities q to initiate the actions for the creation of a new generation.

Transaction 2 performs reproduction; its condition text enables us to select probabilistically those strings that are fit, and its action text replaces the original multiset with the fittest strings.

Transaction 3 performs crossover; its condition text picks up any two elements and the crossover site; the corresponding action text performs crossover and returns the two offspring back to the multiset.

Transaction 4 performs mutation; its condition text gives the probabilistic conditions to select a string as well as the site for mutation; the action text performs mutation and returns the string back to the multiset. This sequence of transactions is then repeated until the multiset is stable or does not undergo a change.

If the reproduction, crossover, and mutation are performed sequentially, one after another, then the UMPP is given by the composition of the four transactions applied to M. We can introduce concurrency among the three transactions 2, 3, and 4, subject to the constraint that they act on different strings. Such a genetic algorithm is called an *elitist model*, where the parents and offspring together can undergo selective reproduction, crossover, and mutation [32, 61, 84].

Note that the evaluation transaction cannot be interleaved with others. In implementing the paradigm, therefore, we must decide how to interleave tasks of different types and size (granularity). This problem is called *multilevel atomicity*. It is possible to interleave the simulated annealing algorithm with the genetic algorithm; the accepted elements in the annealing algorithm can undergo selection, mutation, and crossover or other new operations listed in Michalewicz [61].

The correspondence between the genetic algorithm and UMPP is summarized below:

Genetic Algorithm	UMPP
Generation	Iteration of the multiset
String	Elements
Crossover, Mutation	Interaction
Selection	Deterministic, Probabilistic Choice
Evaluation	Pattern matching, Evaluation
Fitness	Test
Population Fitness	Global test of the multiset
Family fitness	Local test of sets of varying granularity
History	Computational history

5.4 Genetic Programming

Koza's [42] genetic programming can be ably supported and implemented with the very general multiset data structure by a proper choice of the features described in the Introduction, namely, the choice initial object spaces, set of reaction rules and their probabilities, the self-activation of the rules, the control strategy, and the termination condition for evaluating the fitness of elements.

5.5. Evolutionary Optimization

In nature, highly specialized complex structures emerge when their most inefficient elements are selectively driven to extinction. Evolution progresses by selecting against the few most poorly adapted species, rather than by expressly breeding those species best adapted to the environment. The experimental

approach by Boettcher and Percus [10] uses the extremal optimization (EO) processes in which the least fit variables are progressively eliminated [35]. The EO process uses a different strategy in comparison to simulated annealing. In simulated annealing, the system is forced to equilibrium dynamics by accepting or rejecting local changes. EO, however, takes the system to a far-from-equilibrium position, and persistent selection against the worst fitness (i.e., selective extinction or death) leads to a near-optimal solution. Also, EO differs from the genetic algorithm (GA); whereas GA keeps track of entire gene pools of states from which to select and breed an improved generation of solutions, EO operates only with local updates on a single copy of the system, with improvements achieved instead by elimination of the bad. EO also differs from the greedy strategy, which aims at improving the solution at each step and as a result falls into a local optimum. EO, however, can fluctuate between good and bad solutions and can enable us to cross barriers and approach new regions in configuration space. Note that EO can be simulated using UMPP.

5.6 Oscillatory Chemical Reactions

We mentioned in Section 3 that the rule processing does not terminate if and only if rules provide new conditions to fire indefinitely—that is, actions of a rule *Ri* create the right conditions for another rule *Rj* to fire. This leads to circularity in definition or a deadlock. Usually in computer science we do not want this situation to happen, as it leads to a wasteful consumption of resources and instability. However, such a cyclic system occurs commonly in biology and seems to form the basis of all living systems. For example, the energy-rich molecule adenosine triphosphate (ATP) is produced through a succession of reactions in the glycolytic (sugar-splitting) cycle that involves ATP at the start. To produce ATP, we need ATP. These reactions correspond to catalytic reactions that arise in nonequilibrium systems, where a set of reactants produce a set of products that react with some of the reactants and continue reacting indefinitely as an oscillatory reaction. These reactions, for example, can be of the form in which one of the initial reactants A and another reactant X produce Y, and Y produces Z, and Z produces X. Such reactions have rates that are determined by the concentration of the reactants and the products. Such oscillatory systems are called *nonequilibrium systems* or the Brusselator model [76, 86, 87] and can be realized using UMPP.

The Brusselator model consists of the four production rules:

$$(1)\ A \rightarrow X;\ (2)\ BX \rightarrow YD;\ (3)\ XXY \rightarrow XXX;\ (4)\ X \rightarrow E$$

Thus this model is a multiset $M = \{A, B, X, Y, E\}$ and a set of reaction rules R defined above among the elements. Note that an element can undergo self-mutation on its own (autolysis) or due to interaction with the environment, e.g., in the rules for

$$A \rightarrow X \text{ or } X \rightarrow E.$$

We can also incorporate the rate of reaction within each rule, which can be either deterministic or probabilistic. By varying the reaction rates or rates of application of rules to the system, the number of elements X and Y can diverge, become unstable, oscillate, or converge.

For example, a simple simulation shows that the relative frequencies of application of rule (1) and rule (2) can produce varying kinds of effects in the population (numbers) of X and Y.

5.7 Swarm and Ant Colony Paradigm

A swarm (consisting of birds, ants, cellular automata) is a population of interacting elements that is able to optimize some global objective through cooperative search of space. Interactions that are relatively local are often emphasized [39, 11, 24, 60, 56]. There is a general stochastic tendency in a swarm for individuals to move toward a center of mass in the population on critical dimensions, resulting in convergence to an optimum. In real number space, the parameters of a function space can be conceptualized as a point. Here, individual elements in the multiset are points in space, and change over time is represented as movement of points, representing particles with velocities, and the system dynamics is formulated in UMPP using the following rules:

1. *Stepping rule*. The state of each individual element is updated in many dimensions, in parallel, so that the new state reflects each element's previous best success, e.g., the position and momentum (velocity) of each particle.
2. *Landscaping rule*. Each element is assigned a new best value of its state that depends on its past best value and a suitable function of the best values of its interacting neighbors, with a suitably defined neighborhood topology and geometry.

Remark: The above two rules are similar to the strength and support of rules used in BBS in subsection 5.2 above. The first rule reflects the betterment of the individual, while the second rule reflects the betterment of the collection of the individuals in the neighborhood as a whole by evaluating the relevance of each individual and providing support for its activity.

All elements in the multiset or selected chunks are updated using rules (1) and (2). These two rules permit us to model Markovian random walks, which are independent of the past history of the walk and non-Markovian random walks, which are dependent upon past history—such as self-avoiding, self-repelling, and active random-walker models. This can result in a swarm (a self-organizing system) whose global nonlinear dynamics emerges from local rules due to stochasticity or chaoticity introduced by the parameter variation. In nonlinear dynamics, the beautiful property of superposition (namely, the linear combinations of solutions are also solutions and such solutions form a linear vector space) is lost. As a result, there is no general solution. Also, analytic solutions are rare and nonexistent; solutions may exhibit singularities not present in the equations of motion, and these may be sensitive to initial conditions. Further, interesting new properties may show up: low-dimensional attractors, bifurcations, and chaos. Accordingly, various kinds of attractors (as described in Section 4.2) can arise that result in fractal dimensions, presenting a swarmlike, flocklike appearances depending upon the Jacobian of the mapping [57]. Thus, in nonlinear dynamics, integration (solving differential equations) and finding attractors are the key issues.

5.8 Conrad's Lock–Key Paradigm

Conrad and Zauner [20, 21, 22, 95] suggest the lock–key paradigm as the basis for molecular computing. This paradigm arises from the notions of complementarity and the union of opposites that pervade the entire science and natural philosophy. The lock–key paradigm is based on the recognition of an object (molecule) through complementary shape matching. This assumes the existence of a suitably defined "complement" among pairs of objects. Such a complement can be defined for molecular structures at three levels [16]:

1. Primary level. Here we refer to purely syntactic attributes, as in the Watson–Crick complement of a sequence (got by interchanging purines and pyrimidines: A (Adenine) for T (Thymine) and conversely, and G (Guanine) for C (Cytosine) and conversely; or
2. Secondary level. Here we refer to the structure that describes local internal arrangements (alpha helices and beta sheets); or
3. Tertiary level. Here we deal with the 3-D configuration of the structure, namely, the geometric and topological features that are complementary.

To simulate the lock–key paradigm at the above three levels, we need to have a database of elements with complementary attributes, much as in a relational database. However (unlike in a database), here the query and retrieval take place through molecular reaction when a pair matches (at the primary, secondary, or tertiary level according to a specified rule), followed by the required actions (e.g., chemoreceptors for gustation and olfaction). The rule system can be explicit as in a production system or can be implicit as in a dynamical system, where energy minimization brings the required elements into physical contact to self-assemble and generate the required actions. Thus the complementarity paradigm is applicable to both the computational and dynamical systems. It is also useful in synthetic biology [89, 48].

5.9 Membrane and Immunocomputing

The basic datastructure used in membrane [74] and biomolecular immunocomputing [33] are multisets with a priority relation among rules, and the rules are applied in a conditional manner. UMPP provides all these features.

5.10 Quantum Field Theoretical Computations

In the "occupation number formalism" used in Quantum Field theoretical computation [75], states are characterized by a specification of how many particles there are in each of a complete set of single-particle states [51]. The occupation number notation is a multiset datastructure or a bag that has repetitive elements in a set [70]. The repeated elements correspond to indistinguishable objects in the same state. For example, the shell notation for atoms:

$(1s)^2$, $(2s)^2$ $(2p)^1$ meaning that two electrons are in state 1s, two are in state 2s, and one is in the state 2p. This can be written in the multiset notation as {1s.2,2s.2,2p.1}.

Thus, the occupation number formalism in quantum physical computations can be implemented in UMPP. Also, UMPP can simulate the evolution of states

of quantum systems with arbitrarily many particles described by vectors in the Fock space by appropriate choice of rules and actions that satisfy the desired invariant properties.

Quasi-particles are entities in quantum field theory. These consist of a main particle surrounded by a cloud of associated particles that are in certain definite relationship to the main particle. Thus a quasi-particle provides for a collective operation, and hence it simplifies the analysis of the evolution of a system through its propagation. This is the main reason for the use of quasi-particles in physics, as well as in the mutiset parallel computational paradigm. Thus quasi particles behave like user-defined objects that encapsulate data and procedures that can be mobile, e.g., agents in distributed systems.

6 RELATIONAL DATABASE MODEL AND LOCK–KEY PARADIGM

The lock–key paradigm is closely related to rule-based programming and the relational database model. The relational database model consists of one or more relations, each expressed as a set of tuples, often expressed as a table. Each row of this table represents one of the tuples, and each column represents one of the component positions of all the tuples. The tuples are also called *attributes*.

The following properties [41] of relational database programming make it suitable for biosystems:

1. Entire relations are accepted as inputs, and conceptually complete relations are delivered as results.
2. The contents of relations change as a function of time.
3. The relations are always finite, unlike in ordinary computing, where we deal with infinite relations.
4. Individual elements in tuples are always simple ground constants with no variables or complex structures.

Among the many operators of relational algebra, the most important one, which we need for computation with molecular biosystems, is a variant of the join operation. The join operation accepts two relations and joins them along the columns if the attribute names and domains are common to both. Thus it builds up relationships among the relations. However, we still have ground constants as elements of tuples, limiting the expressive power of the relational algebra to that of the first-order logic. When the join condition involves only comparing equality, it is called *equijoin*. Thus when equijoin is performed, we will have one or more pairs of attributes with identical values. In order to remove the superfluous elements, a new operation called *natural join* is used in relational databases. It is basically an equijoin operation followed by the removal of superfluous elements.

In the lock–key or complementarity paradigm, we indeed need a modification of natural join in which the attributes are not equal but are complementary: after the operation the result is a single tuple. We call this new operation *conjoin* to indicate that two objects are united into one, after the natural join of their complementary attributes. The conjoin of two relations R and S, denoted by $R<>S$, is formed by taking each tuple r from R and each tuple s from S and comparing them. If the component of r for A_i equals the complement of the component of

s for B_j, then we form one tuple from r and s; otherwise, no tuple is formed. This requires a lock–key matching of attribute values that are mutually complementary. While forming the tuple from r and s, we take the components of r followed by the components of s, but indicating the match. Note that the conjoin operation is no more expensive than the natural join, since we look for complementary features in each defined attribute. Using conjoin, we can realize the three types of complementary paradigms—primary, secondary or tertiary—with well-defined attributes. Note that the conjoin operation need not necessarily result in a unique tuple, since matching may not be unique.

6.1 Algorithm Conjoin

Consider conjoining relations R with attributes $\{A,B\}$ and relation S with attributes $\{B^*,C\}$, where B^* is a suitably defined complement of B. The algorithm for conjoin, based on a nested "for" loop, is as follows:

for each tuple r in R **do**
  for each tuple s in S **do**
if r and s are complements on their B and B^* attributes
    then
conjoin the tuple matching r and s on attributes A, B, B^*, and C

The above algorithm would take $O(rs)$ time, since we need to pair all the tuples.

6.2 Conjoin and Self-Assembly

In nature, the conjoin is a fundamental operation carried out by RNA polymerase, the enzyme that synthesizes a complementary RNA copy of one or more genes of a DNA molecule. The RNA serves to direct the synthesis of the proteins encoded by those genes. The polymerase essentially carries out a "for" loop; in each cycle, it takes a small molecule (one of the four nucleotide pyrophosphates, ATP, GTP, CTP, or TTP, whose base is complementary to the base about to be copied on the DNA strand) from the surrounding solution, forms a covalent bond between the nucleotide part of the small molecule and the existing uncompleted RNA strand, and releases the pyrophosphate part into the surrounding solution as a free pyrophosphate molecule (PP). The enzyme then shifts forward one notch along the DNA in order to copy the next nucleotide and repeats. The proofreading of DNA replication and repairing damage requires a similar "for" loop. Conrad [20] proposes a jigsaw puzzle model for self-assembly based on the lock–key paradigm and the energy minimization criteria. In this model, energy plays an important role in directing the computation, unlike the models (such as Turing machine) used in computer science.

7 MOLECULAR DNA COMPUTATION

The DNA molecular computation scheme introduced by Adelman [2] [16, 58, 77, 54, 28] models the actual instance of a problem rather than providing a universal approach to computation. It provides for a fine-grained parallelism based

on the free-energy minimization (annealing) associated with the self-assembly properties of DNA sequences. That is, the physical structure of the machine is specific so as to cater to the need for solving a single problem instance [96].

The actual computation proceeds in three phases:

Phase 1. Encoding phase: Here the input is encoded into a multiset whose elements are DNA sequences and their Watson–Crick complements (WCC) so that the required conditions R for the chemical reactions A driven by chemical free-energy minimisation are feasible.

Phase 2. Action (chemical reaction) phase: The encoded molecules chemically react with each other autonomously to form supermolecular complexes (new elements of the multiset) through the association of WCC sequences. Each of the complexes denotes a partial solution to the problem.

Phase 3. Solution selection and termination: The complexes are selected and tested for the existence of the solution or as to whether they satisfy the termination condition of the solution expected.

The above three phases are the basic steps in UMPP. Note that the updating action A (Phase 2) in the conditional in $G(R,A)(M)$, namely: if there exists elements $a,b,c, \ldots, \ldots$ belonging to a multiset M such that $R(a,b,c,\ldots)$, then

$$G(R,A)((M-\{a,b,c, \ldots\}) + A(a,b,c, \ldots)), \text{ else } M$$

is implemented autonomously by the free-energy minimization. However, Phase 3 requires a test for the solution through external physicochemical means, such as polymerase reaction, gel electrophoresis, and hybridization probing.

7.1 Molecular multiset data structure

Adelman's approach uses the biological–chemical reaction in DNA molecules to solve the Hamilton Path Problem (HPP)–[2, 73]: Given a graph G, is there a path through the graph that visits every node in the graph exactly once? This is a proverbially hard problem for which no polynomial time solutions are as yet known.

The data structure chosen is a chemical graph whose vertices are uniquely labeled by a short even-member sequence of DNA consisting of nucleotides A, G, C, or T. This sequence is called *oligonuleotide* and is a single-stranded DNA molecule. Each edge is an ordered pair of nodes (i,j). Such an edge is denoted by the oligonuleotide that combines the righthalf of the DNA sequence $(R(i))$ representing the vertex i and the left -half of the sequence $(L(j))$ representing the vertex j.

To join any two edges (i,j) and (j,k), a chemical reaction known as *ligation* is used. This consists in tying two edges together by using a WCC oligonuleotide of j (in which A is replaced by T and conversely, and C is replaced by G and conversely). Thus the fusing of two edges denoted by the ordered triplet (i,j,k) is represented by the fused double-stranded DNA sequence $R(i),L(j),R(j),\ L(k)$, in which the middle part $L(j)\ R(j)$ is ligated to WCC $(j) = L(j^*)\ R(j^*)$. This double-stranded sequence is denoted by

$\{R(i), (L(j)\ \&\ L(j^*)), (R(j)\ \&\ R(j^*)), L(k)\}$, where "&" denotes an elementwise nucleotide fusing operation that can be looked upon as conjoin operation earlier described in a relational database context .

Adeleman's algorithm consists of three basic tasks:

1. Using molecules to represent elements of multiset
2. Self-assembly of elements of the multiset satisfying the reaction condition
3. Checking for termination conditions

The above tasks are implemented by Adelman to find the Hamilton path using the following four steps:

1. Synthesize a random path in the graph
2. Retain only those paths that satisfy the condition specified for the beginning and ending vertices of the path.
3. Keep only those paths that contain exactly n different vertices
4. Check for termination by looking for only those paths that go through all the vertices at least once.

7.2 Molecular Chemical Transactions

Adelman uses different types of chemical and physico–chemical operations (these may be called *genetic engineering* operations): copy, paste, amplification, extraction, identification zero-test, sum, for which we use the computer science term *transaction* meaning that these need to have four important properties called ACID properties [49, 95]: Atomicity (indivisibility and either all or no actions carried out), Consistency (before and after the execution of a transaction), Isolation (no interference among the actions), and Durability (no failure).

1. *Copy/ replicate*. This synthesizes a large number of copies of any single-stranded DNA. That is, it creates a large number of copies of the elements of a multiset M.
2. *Paste or create double strands*. This produces a combination of elements of multisets, resulting in a multiset containing new elements that can be ordered sequences containing WCC fused subsequences. Such a fusion creates double strands. This set of elements can be represented by a parallel elementwise operation on subsequences, denoted by "&". This operation is chemically achieved through self-assembly by free-energy minimization.
3. *Selective amplify*. This operation is different from "copy" in that it is a higher-level or macro-copy operation that replicates any selected element (single- or double-stranded DNA) whose attributes are specified. This operation is externally achieved either through cloning or through the more efficient polymerase reaction (PCR). This is analogous to the selective reproduction strategy widely used in the genetic algorithms.
4. *Extraction*. This operation extracts sequences of specified lengths. It is carried out using the physicochemical operation known as *gel electrophoresis*. This operation is similar to the message pattern matching used in classifier and production systems. Also, it is similar to the selection or projection operation used in the theory of recursive functions to select the k elements among n ordered elements.
5. *Identification*. This operation selects those sequences of specified lengths that contain specified subsequences. Using hybridization probes of subsequences that are complementary to the selected subsequence, individual vertices are identified.

6. *Zero test*. this operation determines whether or not there is a DNA strand. This is a fundamental operation required in the recursive functions.
7. *Sum or merge*. Given two multisets M and N, the sum denoted by $M+N$ is the multiset where each element has a multiplicity that is a sum of its multiplicity in M and N.

In DNA computing, knowledge is represented by omission of negative facts. Propositions that are not given are assumed to be false; that is, we use the *Closed world assumption (CWA)*, in which only positive facts are stated. As mentioned earlier, CWA is valid only for Horn clauses; that is, if there are non-Horn clauses, we cannot make the closed-world assumption [31]. For nonequilibrium systems interacting with their surroundings through an entropy flow, this assumption is invalid [76]. Most such systems use feedback mechanisms involving catalytic reactions, as, for example, production of ATP (adenosine triphosphate) from ATP. Such reactions arise from far-from-equilibrium conditions. These lead to the necessity for the *Open-world hypothesis*, which leads to newer phenomena such as self-organization and active walks (swarm intelligence) [11, 23, 39, 56].

Finally, we observe that we can combine neural and molecular computing. The genetic code may be looked upon as a string of spin variables undergoing dynamical development in the course of reproduction. Analogous to Hopfield's spin-glass approach for neural nets, Anderson and others have formulated a model for the evolution of organisms using a fitness landscape. Kuhn's model uses the concept of RNA replication and autocatalysis driven by temperature cycling [38]. A population of organisms (strings of + or − spins $S(i)$) is encouraged to conjugate randomly in pairs during a cooling cycle, and if two shorter strings (RNA polymers) are conjugated end to end on a third longer one, they are allowed to bond to each other, giving a long single RNA polymer. The strings are then subjected to a heating cycle, where they separate and also encounter a fitness function that can be identified by a spin-glass Hamiltonian and that serves as a criterion for survival into the next conjugation. The fitness landscape is a spin-glass function. The use of autocatalysis seems to extend the power of first-order logic to overcome the limitations of the closed-world assumption.

8 DISCRETE ADAPTIVE STOCHASTIC OPTIMIZATION

Searching and exploration form the basis for many types of data analysis, adaptive learning, and pattern classification problems. Adaptive systems need to use some form of search operation to explore a feature space that describes all possible configurations of the system. Usually we are interested in "optimal" or "near optimal" configurations defined with respect to a specific problem domain. Such problem domains are usually high dimensional with no single optimal solution and are multimodal, i.e., they can have many local and global optimal solutions.

In general, finding the global maximum and minimum of multimodal problems with high dimensions and conflicting constraints turnsout to be exponential in complexity, and usually the problems are NP-complete or even NP-hard [46].

Therefore, conventional search techniques are inefficient. We need to use a probabilistic approach that is adaptable to the particular problem domain so that the search space can be sampled to yield near-optimal solutions with a high probability. Any such adaptive (learning) search methods are characterized by taking the following aspects into account:

1. How are solutions (parameters, hypotheses) represented? What data structures are used?
2. What search operators are used in moving from one configuration to the next? How is the adaptive step defined?
3. What type of search is conducted by applying the search operators iteratively? How is the search space explored and exploited?
4. Is the adaptive system supervised (interactive) or unsupervised (noninteractive)?
5. How can problem-specific knowledge be incorporated into the adaptive learning algorithm?

8.1 Example

Consider the following discrete stochastic optimization problem. Let $\Theta = \{1,2,\ldots,S\}$ denote a finite set and consider the following problem: Compute

$$\theta^* = \min_{\theta \in \Theta} E\{X_n(\theta)\},$$

where E denotes mathematical expectation and, for any fixed $\theta \in \Theta$, $\{X_n(\theta)\}$ denotes a sequence of independent and identically distributed (iid) random variables that can be generated for any choice of $\theta \in \Theta$. If the density function of $X_n(\theta)$ is not known, then it is not possible to analytically evaluate the above expectation, and hence θ^*. In such a case, one needs to resort to simulation-based stochastic approximation to compute the optimal solution θ^*.

A brute-force approach of computing the optimal solution to the problem involves exhaustive enumeration over all Θ and proceeds as follows: For each $\theta \in \Theta$, generate a large number N of random samples $X_1(\theta), X_2(\theta),\ldots X_N(\theta)$. Then compute an estimate of $E\{X_n(\theta)\}$ using the sample average (arithmetic mean)

$$G_N(\theta) = (X_1(\theta) + X_2(\theta) + \ldots + X_N(\theta)/N.$$

By Kolmogorov's strong law of large numbers (which is one of the most fundamental consequences of the ergodic theorem for iid processes), $G_N(\theta) \to E\{X_n(\theta)\}$ with probability one as $N \to \infty$. This and the finiteness of Θ imply that

$$\arg\max_{\theta \in \Theta} G_N(\theta) \to \arg\max_{\theta \in \Theta} E\{X_n(\theta)\} \text{ as } N \to \infty.$$

However, the above brute-force procedure is extremely inefficient—evaluating $G_N(\theta)$ at values $\theta \in \Theta$ with $\theta \neq \theta^*$ is wasted effort, since it contributes nothing towards evaluating $G_N(\theta^*)$. What is required is an intelligent dynamic scheduling (search) scheme that decides at each time instant which value of θ to evaluate next, given the current estimates, in order to converge to the maximum θ^* with minimum effort.

There are several different classes of methods that can be used to solve the above discrete stochastic optimization problem [3, 52]. When the feasible set Θ is small (usually 2 to 20 elements), statistical ranking and selection methods and

multiple comparison methods can be used to locate the optimal solution. However, for large Θ, the computational complexity of these methods becomes prohibitive. The above problem can also be viewed as a multiarmed bandit problem, which is a special kind of infinite-horizon Markov decision process with an "indexable" optimal policy. However, as mentioned in Andradottir [4], multiarmed bandit solutions and learning automata procedures often tend to be conservative because they are designed to spend as much time as possible at the optimum solution.

8.2 Stochastic Approximation Algorithm

In recent years a number of discrete stochastic approximation algorithms have been proposed. Several of these algorithms [3], including simulated annealing-type procedures and stochastic ruler, fall into the category of random search. Here we present a globally convergent discrete stochastic approximation algorithm based on the random search procedures in Andradottir [3]. The basic idea is to generate a homogeneous Markov chain, taking values in Θ that spend more time at the global optimum than at any other element of Θ. This generation consists of the following UMPP steps:

Step 0: Initialization. At time n=0, select starting point $\theta_0 \in \Theta$ randomly with uniform probability. Set $D_0 = e_\theta 0$, where e_i denotes the S-dimensional unit vector with 1 in the ith position and zeros elsewhere. Set the initial solution estimate $\hat{\theta}_0 = x_0$.

Step 1: Sampling. At time n, sample $u_n \in \Theta - \{\theta_n\}$ with uniform distribution.

Step 2: Evaluation and acceptance.
Evaluate the random sample costs $X_n(\theta_n)$ and $X_n(u_n)$.
If $X_n(\theta_n) > X_n(u_n)$, then set $\theta_{n+1} = \theta_n$; else set $\theta_{n+1} = u_n$.

Step 3: Update duration time vector at time n+1 as $D_{n+1} = D_n + e_{\theta_n}$

Step 4: Update estimate of maximum at time n as $\hat{\theta}_n = \arg\max_{i \in \{1,2,\ldots,S\}} D_{n+1}(i)$
Set $n \rightarrow n+1$ and go to Step 1.

Then, as proved in Andradottir [3], under suitable conditions (e.g., if the density function with respect to which the expected value is defined above is symmetric), the estimate $\hat{\theta}_n$ generated by the above random search stochastic approximation algorithm converges with probability one to the global optimum θ^*. It is also shown in Andradottir [3] that the algorithm is attracted to the global optimum, i.e., the algorithm spends more time at the global optimum than at any other candidate value. That is, for sufficiently large n, the duration time vector D_n has its maximum element at θ^*.

8.3 Applications

The above discrete stochastic approximation algorithm has several applications. For example, in Krishnamurthy and Chung (2003), it is used to learn the behavior

of an ion channel (large protein molecule) in a nerve cell membrane to estimate the Nernst potential efficiently. In Krishnamurthy, Wang and Yin [53], a recursive version of the algorithm is used to optimize the spreading code of a CDMA spread spectrum transmitter over a fading wireless channel. More recently, in Yin, Krishnamurthy and Ion [93], an adaptive version of the above algorithm is presented that can track a slowly time-varying global optimum. For a weak convergence analysis and complexity aspects of this adaptive algorithm, [93, 52].

9 CONCLUSION

This chapter presents a unified rule-based multiset programming paradigm (UMPP) as a general model and unifying theme for conventional and soft-computing. The introduction of probabilistic choices in a multiset chemical reaction model provides a soft-computational model to study evolutionary biological, chemical, and physical systems based on intermittent feedback from the environment. Unlike conventional computation, where exactness is our goal, in soft computation, we allow the possibility of error and randomness to model features that are inherent in problems arising in nature. The paradigm described here provides a new programming environment based on a distributed architecture for classifier, bucket brigade, genetic, and molecular algorithms as well as ant-algorithms, swarm intelligence, membrane and bio-immunology computing, multiple-particle filtering, adaptive stochastic optimization and self-organized criticality. This paradigm is well suited for cluster and grid computing.

REFERENCES

[1] A.N. Abdallah (1995): *The Logic of Partial Information*, Springer Verlag, New York.

[2] L.M. Adelman (1994): Molecular computation of solutions to combinatorial problems, *Science*, *266*, 1021–1024.

[3] S. Andradottir (1996): A global search method for discrete stochastic optimization, *SIAM Journal of Optimization*, *6*, 2(1), 513–530.

[4] S. Andradottir (1999): Accelerating the convergence of random search methods for discrete stochastic optimization, *ACM Transactions on Modelling and Computer Simulation*, *9*, 4(1), 349–380.

[5] R. Backhouse and J. Gibbons (2003): Generic Programming, *Lecture Notes in Computer Science*, Vol. 2793, Springer Verlag, New York.

[6] J.-P. Banatre, D.L. Me'tayer (1990): The Gamma model and its discipline of programming, *Science of Computer Programming*, *15*, 55–77.

[7] J.-P, Banatre, D.L. Me'tayer (1993): Programming by Multiset transformation, *Comm. ACM*, *36*, 98–111.

[8] R.K. Belew, S. Forrest (1988): Learning and programming in classifier systems, *Machine Learning 3*, 193–223.

[9] T. Blackwell and J. Branke (2004): Multi-swarm optimization in dynamic environments, *Lecture Notes in Computer Science*, Vol. 3005, pp. 489–500, Springer Verlag, New York.

[10] S. Boettcher, and A. Percus (2000): Nature's way of optimizing, *Artificial Intelligence*, 119, 275–286.
[11] E. Bonabeau, M. Dorigo and G. Theraulaz (1999): *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, U.K.
[12] L.K. Booker, D.E. Goldberg, J.H. Holland (1986): Classifier systems and Genetic Algorithms, *Artificial Intelligence*, *40*, 235–282.
[13] J. Branke, H.C. Andersen and H. Schmeck (1996). Global selection methods for massively parallel computers, in *Evolutionary Computing*, T.C. Fogarty, ed., *Lecture Notes in Computer Science*, *1143*, 175–188, Springer Verlag, New York.
[14] C.S. Calude, et al., (2001): Multiset processing, *Lecture Notes in Computer Science*, Vol. 2235, Springer Verlag, New York.
[15] C. Cannings and D.D. Penman (2003): Models of Random graphs and their applications, *Handbook of Statistics*, C.R. Rao, ed., *21*, 51-91, North Holland, Amsterdam.
[16] N. Campbell (1996): *Biology*, Benjamin/Cummings, New York.
[17] K.S. Chan, and H. Tong (2002): *Chaos: A Statistical Perspective*, Springer, New York.
[18] S. Chu, et al., (2003): Parallel ant colony systems, *Lecture Notes In Artificial Intelligence*, *2871*, 279–284, Springer Verlag, New York.
[19] C.A.C. Coello, D.A. Van Veldhuizen, G.B. Lemont (2002): *Evolutionary Algorithm for Solving Multi-objective Problem*, Kluwer, New York.
[20] M. Conrad (1992): Molecular computing paradigms, *Computer*, *25*, 6–68.
[21] M. Conrad, K.-P. Zauner (1997): Molecular computing: From conformational pattern recognition to complex processing networks, in *Bioinformatics, Lecture Notes in Computer Science 1278*, 1–10, Springer Verlag, New York.
[22] M. Conrad, K-P Zauner (1998): DNA as a vehicle for the self-assembly model of computing, *Biosystems*, *45*, 59–66.
[23] M. Dorigo, G.D. Caro and M. Sampels (2002): Ant algorithms, *Lecture Notes in Computer Science*, Vol. 2463, Springer Verlag, New York.
[24] M. Dorigo, and T. Stutzle (2004): *Ant Colony Optimization*, M.I.T. Press, Cambridge, Mass.
[25] S.N. Dorogovtsev, and J.F.F. Mendes, (2003): *Evolution of Networks*, Oxford University Press, Oxford.
[26] A. Doucet et al., (2000): *Sequential Monte-Carlo Methods in Practice*, Springer, New York.
[27] A. Doucet, N. Gordon, V. Krishnamurthy, (2001): Particle filters for state estimation of jump Markov linear systems, *IEEE Trans. Signal Processing*, *49*, 613–624.
[28] J.L. Fernandez-Villacanas, J.M. Fatah, S. Amin (1998): Computing with evolving proteins, *Parallel and Distributed Processing*, J. Rolim, ed. *Lecture Notes in Computer Science*, Vol. 1388, Springer Verlag, New York, pp. 207–215.
[29] S. Forrest (1991a): *Parallelism and Programming in Classifier Systems*, Morgan Kauffman, San Mateo, California.
[30] S. Forrest (1991b): *Emergent Computation*, M.I.T Press, Cambridge, Mass.
[31] M.H. Genesereth, N. Nilsson, (1987): *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, Los Altos, California.

[32] D.E. Goldberg, (1989): *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison Wesley, Reading, Mass.
[33] L. Goncharova, et al., (2003): Biomolecular immunocomputing, *Lecture Notes in Computer Science*, *2787*, 102-110, Springer Verlag, New York.
[34] J.J. Grefenstett, (1988): Credit assignment in rule discovery systems based on genetic algorithms, *Machine Learning*, *3*, 225–245.
[35] J.H. Holland, et al., (1987): *Induction*, M.I.T. Press, Cambridge, Mass.
[36] A. Ilachinski, (2002): *Cellular Automata*, World Scientific, Singapore.
[37] T. Ishida (1991): Parallel, distributed and multiagent production systems, *Lecture Notes in Computer Science*, 890, Springer Verlag, New York.
[38] S.A. Kauffman (1993): *The Origins of Order*, Oxford University Press, Oxford.
[39] J. Kennedy and R.C. Eberhart, (2001). *Swarm Intelligence*, Morgan Kauffman, London.
[40] P. Kevin MacKeown (1997): *Stochastic Simulation in Physics*, Springer, New York.
[41] P.M Kogge, (1991): *The Architecture of Symbolic Computers*, McGraw Hill, New York.
[42] J.R. Koza, (1994): *Genetic Programming II*, M.I.T. Press, Cambridge, Mass.
[43] E.V. Krishnamurthy, (1985): *Introductory Theory of Computer Science*, Springer Verlag, New York.
[44] E.V. Krishnamurthy (1986): Solving problems by random trials, Science and computers, (A volume dedicated to Nicholas Metropolis), G.C. Rota, ed., *Advances in Mathematics*, *10*, 61-81, Academic Press, New York.
[45] E.V. Krishnamurthy, (1989): *Parallel Processing*, Addison Wesley, Reading, Mass.
[46] E.V. Krishnamurthy (1996): Complexity issues in parallel and distributed computing, in *Handbook of Parallel and Distributed Computing*, Chapter 4, A. Zomaya, ed., McGraw Hill, New York.
[47] E.V. Krishnamurthy, (2003): Algorithmic entropy, phase transitions, and smart systems, *Lecture Notes in Computer Science*, *2659*, 333–342, Springer Verlag, New York.
[48] E.V. Krishnamurthy, (2004): Rule-based Multiset Programming Paradigm, Applications to Synthetic Biology, Third Workshop on Non-Silicon Computation, (NSC-3), Munich, in *31st International Symposium on Computer Architecture*, Munich, June 2004.
[49] E.V. Krishnamurthy, V.K. Murthy, (1992): *Transaction Processing Systems*, Prentice Hall, Sydney.
[50] V. Krishnamurthy, and E.V. Krishnamurthy, (1999): Rule-based Programming Paradigm: A formal basis for biological, chemical and physical computation, *Biosystems*, *49*, 205–228.
[51] E.V. Krishnamurthy, and V. Krishnamurthy (2001): Quantum field theory and computational paradigms, *International Journal of Modern Physics*, *12*C, 1179–1201.
[52] V. Krishnamurthy, and S.H. Chung (2003): Adaptive learning algorithms for Nernst potential and I-V curves in nerve cell membrane ion channels modelled as hidden Markov models, *IEEE Transactions NanoBioScience*, *2*(4), 266–278.

[53] V. Krishnamurthy, X. Wang, G. Yin (2004): Adaptive Spreading Code Optimization and Adaptation in CDMA via Discrete Stochastic Approximation, *IEEE Transactions Information Theory*, *50*(9), 1927–1949.
[54] I. M. Kulic (1998): Evaluating polynomials on the molecular level—a novel approach to molecular computers, *Biosystems*, *45*, 45–57.
[55] S. Kuo, D. Moldovan, (1992): The state of the art in parallel production systems, *J. Parallel and Distributed Computing*, *15*, 1–26.
[56] L. Lam (1998): *Nonlinear Physics for Beginners*, World Scientific, Singapore.
[57] A.J. Lichtenberg and M.A. Liberman, (1983): *Regular and Stochastic Motion*, Springer Verlag, New York.
[58] R.J. Lipton (1995): DNA solution to hard computational problems, *Science*, *268*, 542-545.
[59] W. Ma,, E.V. Krishnamurthy and V.K. Murthy (1995): Multran—A coordination programming language using multiset and transactions, *Proc. Neural, Parallel and Scientific Computing*, *1*, 301-304, Dynamic Publishers, Inc., U.S.A.
[60] N. Meuleau and M. Dorigo, (2002): Ant colony optimization and stochastic gradient descent, *Artificial Life*, *8*, 103–121.
[61] Z. Michalewicz (1992): *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag, New York.
[62] Z. Michalewicz and D.B. Fogel (2000): *How to Solve It: Modern Heuristics*, Springer Verlag, New York. (1992,
[63] D. Midgley (2003): *Systems Thinking*, Vols. 1–4, Sage Publications, London.
[64] R.K. Milne (2001): Point processes and some related processes, *Handbook of Statistics*, *19*, 599–641, C.R.Rao, ed., North Holland, Amsterdam.
[65] D.P. Miranker (1991), *TREAT: A New Efficient Match Algorithm for AI Production Systems*, Pitman, London.
[66] B. Misra, I. Prigogine and M. Courbage (1979), From deterministic dynamics to probabilistic descriptions, *Physica*, 98A, 1–26.
[67] R. Motwane and P. Raghavan (1995), *Randomized Algorithms*, Cambridge University Press, Cambridge.
[68] H. Muehlenbein (1991), Evolution in time and space-the parallel genetic algorithm, in *Foundations of Genetic algorithms*, Rawlins, G., ed., Morgan Kaufmann, San Mateo, California, 316–337.
[69] J.D. Murray (2003): *Mathematical Biology*, Springer, New York.
[70] V.K. Murthy and E.V. Krishnamurthy (1995): Probabilistic Parallel Programming based on multiset transformation, *Future Generation Computer Systems*, *11*, 283–295.
[71] V.K. Murthy and E.V. Krishnamurthy, (2003): Entropy and Smart systems, *International Journal of Smart Engineering Systems*, *5*, 481-499.
[72] K.M. Pacino (2002): Biomimicry of bacterial foraging for distributed optimization and control, *IEEE Control magazine*, *22*(3), 52-68.
[73] C.H. Papadimitriou (1985): *Computational Complexity*, Addison Wesley, Reading, Mass.
[74] G. Paun (2003): Membrane computing, *Lecture Notes in Computer Science*, FCT 2003, *2751*, 284–295, Springer Verlag, New York.
[75] D. Petrina, Ya., (1995): *Mathematical Foundations of Quantum Statistical Mechanics*, Kluwer Academic Publishers, London.
[76] I. Prigogine (1980): *From Being to Becoming*, W.H. Freeman, San Fransisco.

[77] N.G. Rambidi (1997): Biomolecular computer: roots and promises, *Biosystems*, *44*, 1–15.
[78] R.D. Reiss (1993): *A Course on Point processes*, Springer Verlag, New York.
[79] C.P. Robert and G. Casella (1999) *Monte Carlo Statistical Methods*, Springer Verlag.
[80] E. Rich, K. Knight (1991): *Artificial Intelligence*, McGraw Hill, New York.
[81] J.D. Scargle and G.J. Babu (2003), Point processes in astronomy, *Handbook of Statistics*, C.R. Rao, ed., 21, 795–825, North Holland, Amsterdam.
[82] R.J. Solomonoff (1995): The discovery of algorithmic probability: A guide for the programming of true creativity, *Lecture Notes in Computer Science*, *904*, 1–22.
[83] J.C. Spall (2003): *Introduction to Stochastic Search and Optimization*, Wiley-Interscience, New York.
[84] W.M. Spears, and K.A. De Jong (1993): An overview of evolutionary computation, *Machine Learning ECLML-93, Lecture Notes in Computer Science*, *667*, 442-459, Springer Verlag, New York.
[85] S. Stepney, J.A. Clark et al., (2003): Artificial Immune System and the grand challenges for non-classical computation, *Lecture Notes in Computer Science*, *2787*, 204–216, Springer Verlag, New York.
[86] D. Straub (1997): *Alternative Mathematical Theory of Nonequilibrium Phenomena*, Academic Press, New York.
[87] Y. Suzuki, et al., (2001): Artificial Life applications of a class of P systems: Abstract rewriting systems on Multisets, *Lecture Notes in Computer Science*, *2235*, 299–346, Springer Verlag, New York.
[88] A.M. Turing (1952): The chemical basis for morphogenesis, *Phil. Trans. Roy. Soc. London*, *237*, 37–79.
[89] W. Wayt Gibbs (2004): Synthetic life, *Scientific American*, *290*(5), 48–55.
[90] D. Whitley T. Starkweather (1990): Genitor: a distributed Genetic algorithm, *J. Experimental and Theoretical Artificial Intelligence*, *2*, 184–214.
[91] S. Wolfram (2002): *A New Kind of Science*, Wolfram Media Inc., Champaign, Ill.
[92] X. Yao, (2003): The evolution of evolutionary computation, *Lecture Notes in Artificial Intelligence*, *2773*, 19–20, Springer Verlag, New York.
[93] G. Yin, V. Krishnamurthy and C. Ion (2004): Regime Switching Stochastic Approximation Algorithms with application to adaptive discrete stochastic optimization, *SIAM Journal of Optimization*, *14*(4), 1187–1215.
[94] D.C.K. Yuen and B.A. MacDonald (2004): Theoretical considerations of multiple particle filters for simultaneous localization and map-building, *Lecture Notes in Computer Science*, *3213*, 203–209.
[95] K.-P. Zauner, M. Conrad (1996): Parallel computing with DNA: toward the Anti-Universal Machine, *Proc. PPSN-IV, Lecture Notes in Computer Science*, 1141, Springer Verlag, New York.
[96] W. Zhang and R. Korf (1996): A study of complexity transitions on the asymmetric travelling salesman problem, *Artificial Intelligence*, *81*, 223–239.