

Autonomy Oriented Computation

Jiming Liu and Kwok Ching Tsui

Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong

COMP-03-003

Release Date: February 5, 2003

Abstract

Examples of autonomous multi-entity systems are plentiful, both in the natural and artificial worlds. Many systems have been studied in depth and some models of these have been built in computational systems for problem solving. Central to these computational systems is the notion of autonomy. This article proposes autonomy oriented computation (AOC) as a complementary paradigm for solving hard computational problems and for characterizing the behaviors of a complex system.

Autonomy Oriented Computation

Jiming Liu, *Senior Member, IEEE*, and Kwok Ching Tsui

Jiming Liu and Kwok Ching Tsui are with Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Kowloon, Hong Kong, E-mail: {jiming,tsuikc}@comp.hkbu.edu.hk.

June 28, 2002

DRAFT

Abstract

Examples of autonomous multi-entity systems are plentiful, both in the natural and artificial worlds. Many systems have been studied in depth and some models of these have been built in computational systems for problem solving. Central to these computational systems is the notion of autonomy. This article proposes *autonomy oriented computation* (AOC) as a complementary paradigm for solving hard computational problems and for characterizing the behaviors of a complex system. .

Index Terms

Autonomy Oriented Computation (AOC); Synthetic autonomy; Emergent autonomy; Self-organization; Autonomous entities; Multi-agent systems.

I. INTRODUCTION

Nature is full of complex systems that exhibit intelligent behavior and they have been studied extensively from different angles and with different objectives. Some researchers want to understand the working mechanism of the complex system concerned. Immunologists, for example, want to know the way human immune system reacts to antigens [1]. Similarly, economists want to know the factors contributing to the ups and downs in share prices. The knowledge gained in this way helps the scientists to predict future system behavior. Other researchers studying complex system behavior want to simulate the observed complex behavior and formulate problem solving strategies for hard computational problems such as global optimization. Computer scientists and mathematicians have formulated various algorithms based on natural evolution for solving their problems at hand. In general, one wants to be able to *explain, predict, reconstruct* and *deploy* a complex system.

Common techniques for complex systems modeling can broadly be divided into top-down and bottom-up approaches. Top-down approaches start from the high-level system and use various techniques such as ordinary differential equations. These methods generally treat every part of a complex system homogeneously and tend to model the average case well, where regional variation in behavior is minimal and can be ignored [2]. However, it seems this is not always applicable. Bottom-up approaches, on the other hand, start from the smallest and simplest element of the system based on the following characteristics of the entities in a complex system:

- **Autonomous:** the systems' elements are rational individuals that will act independently. In other words, a central controller for directing and coordinating individual elements is absent;
- **Emergent:** They exhibit, often not simple, behaviors, that are not present or pre-defined in the behavior of the autonomous entities within complex adaptive systems;
- **Adaptive:** They often change their behavior in response to changes in the environment in which they are situated; and
- **Self-organized:** They are able to organize the elements to achieve the above behaviors.

This article proposes *autonomy oriented computation* (AOC) as a complementary paradigm for solving hard computational problems and for characterizing the behaviors of a complex system. The first goal of AOC is *to*

reproduce life-like behavior in computation. With detailed knowledge of the underlying mechanism, simplified life-like behavior can be used as model for a general-purpose problem solving technique. Replication of behavior is not the end, but rather the means, of these computational algorithms.

The second goal of AOC is *to understand the underlying mechanism of a real-world complex system* by hypothesizing and repeated experimentation. The end product of these simulations is a better understanding of or explanations to the real working mechanism of the modeled system.

The third goal of AOC concerns *the emergence of a problem solver in the absence of human intervention.* In other words, self-adaptive algorithms are desired.

The following is a list of common steps in AOC-based approaches:

- i. observe macroscopic behavior of a natural system ;
- ii. design entities with desired synthetic behavior;
- iii. observe macroscopic behavior of the artificial system;
- iv. validate the behavior of the artificial system against the natural counterpart;
- v. modify (ii) in view of (iv);
- vi. repeat (iii)-(v) until satisfactory;
- vii. find out a model/origin of (i) in terms of (ii) or apply the derived model to solve problems.

A. Organization of the Article

The article is structured as follows. Definitions of the crucial concepts of behavior and autonomy are given (Section II). Autonomy oriented computation (AOC) is then discussed where details of three major approaches to AOC and their examples are given. Also discussed are issues related to the practical usage of AOC such as principles of application, simulation environments and other paradigms related to AOC (Section III). Some research issues, both theoretical and practical, are also described (Section IV). This article concludes with a summary of the main points.

II. CONCEPTS AND TAXONOMIES

Complex systems modeling using a bottom-up approach centers around the external behavior and internal behavior of individual entities. The trickiest of all is finding the relationship between these two behavior. Autonomy oriented computation adds a new dimension to the modeling process, *i.e.*, modeling and deploying autonomy. Broadly speaking, autonomy is an attribute of entities in a complex system and is the building block of an autonomy oriented computation algorithm. This section attempts to differentiate between different types of behavior and their relationships. A definition for autonomy in the context of a computational system is also given below.

A. Types of Behavior

Entities in a complex system have a set of local behaviors and three kinds of external behaviors: purposeful behavior, emergent behavior and emergent purposeful behavior.

Definition 1 *Local behavior is a set of rules that governs how the private attributes of a complex system entity are modified. The rules are triggered based on some internal and/or external factors.*

Definition 2 *Purposeful behavior is the result of one or more entities acting together to achieve a goal. Such behavior can be defined with the attributes and capabilities of the entities concerned.*

Definition 3 *Emergent behavior is some system behavior not inherent in the individual entities. It can only be resulted from the interactions of individual entities or other emergent behaviors. Such behavior cannot be defined simply by the attributes and capabilities of the entities concerned.*

Definition 4 *Emergent purposeful behavior is a goal-directed emergent behavior that is also not achievable purely at the individual entity level.*

Individual entities have their local behavior. Whether or not they are uniformly identical depends on the system concerned. If the entities of a complex system are able to adapt, the local behavior of each entity are bound to be different over time. As a result, the external behavior may also be changed. Worth noting is the fact that emergent behavior may not arise just from the interactions between the basic elements but from the interactions between subsystems.

Take an ant colony as an example, food foraging is an individual task as well as a group task [3]. Thus the wandering around of ants is a purposeful behavior. Their ability to converge on a certain food source is an example of emergent behavior. On the other hand, the ability of slime-molds to change form in the presence of an unfavorable environmental condition is an example of emergent purposeful behavior.

B. Autonomy Defined

According to the American Heritage Dictionary of the English Language, *autonomy* is defined as the condition or quality of being (1) autonomous; independence, (2) self-government or the right of self-government; self-determination and self-directed. All these relate to freedom from the control by others with respect to local or internal affairs. In the context of Artificial Intelligence, autonomy has been one of the key notions in many research fields such as Intelligent Agents and Artificial Life [4], [5], to name but a few. A similar definition can be stated as follows.

Definition 5 *Autonomy is an attribute of a self-governed, self-determined and self-directed entity with respect to its own action and internal state, free from the explicit control of another entity.*

This is an endogenous view of autonomy. In other words, the internal affair of an entity is protected from the influence of others in the way similar to that of an object in the software engineering sense. However, only direct perturbation is prohibited; indirect influence is allowed and encouraged. The underlying assumption is that each

entity is able to make decisions for itself, subject to the limitations of the available information and its self-imposed constraints.

Definition 6 *Synthetic Autonomy is defined as an abstracted equivalent of the desired observable attribute of an entity in a natural complex system and is the fundamental building block of an autonomy oriented computational system.*

Any computational system having synthetic autonomy exhibits its emergent or emergent purposeful behavior. Similar to the messages that are passed between objects in an object-oriented program, an inter-entity interaction is the ‘glue’ that helps to put the building blocks together to form a coherent system.

Definition 7 *Emergent Autonomy is defined as the observable, self-induced attribute of an autonomy oriented computational system built using entities having synthetic autonomy as the basic building block.*

It is not difficult to see that a computational system can be derived from many levels of abstraction. If a human society was to be modeled as a computational system, abstraction can possibly occur at the level of population, individual, biological system, cell, molecule and atom. Note that autonomy according to Definition 5 is present at all these levels. Moreover, the emergent autonomy obtained at, say, the cell level, is the foundation for the emergent autonomy at the biological system level. With the above definitions, autonomy in the context of a computational system can be stated as:

Definition 8 *Autonomy in a computational system is an attribute of a self-governed, self-determined and self-directed computational entity that exhibits emergent autonomy, built from computational entities having synthetic autonomy.*

This multi-level view of autonomy encompasses Brooks’ subsumption architecture [6] in that complex behavior can be built up from multiple levels of simpler, and relatively more primitive, behavior.

III. GENERAL APPROACHES IN AUTONOMY ORIENTED COMPUTATION

Autonomy oriented computation (AOC) refers to computational approaches that employ autonomy as the core model of any complex systems behavior. They aim at reconstructing, explaining and predicting the behavior of such systems, which is hard to model or compute using top-down approaches. Local interaction between the autonomous entities is the primary driving force of AOC. Formulation of an autonomy oriented computational system involves an appropriate analogy, which normally comes from nature. Employing such an analogy, therefore, requires identification, abstraction and reproduction of certain natural phenomenon. The process of abstraction inevitably involves certain simplification of the natural counterpart. An abstracted version of some natural phenomena is the starting point of AOC such that the problem at hand can be recasted. There are three different approaches to AOC and they are described in more detail, with examples, in the following sections.

AOC-by-fabrication aims at replicating certain self-organized behavior observable in the real-world to form a general-purpose problem solver. The operating mechanism is more or less known and may be simplified during the modeling process. Research in Artificial Life is related to this AOC approach up to the behavior replication stage. Nature-inspired techniques such as Genetic Algorithm and Ant system are typical examples of such an extension.

AOC-by-prototyping aims at understanding the operating mechanism underlying a complex system to be modeled by simulating the observed behavior, through characterizing a society of autonomous entities. Examples of this approach include the study of Internet ecology, traffic jams and Web log analysis. This AOC approach relates to multi-agent approaches to complex systems in Distributed Artificial Intelligence.

AOC-by-self-discovery aims at the automatic discovery of a solution. The trial-and-error process of an AOC-by-prototyping algorithm is replaced by autonomy in the system. In other words, the distance measure between the desired emergent behavior and the current emergent behavior of the system in question becomes part of the environmental information that affects the local behavior of an entity. Some evolutionary algorithms that exhibit self-adaptive capability are examples of this approach.

AOC algorithms share a basic form with many possible variants. The following sections attempt to characterize these three classes of AOC algorithms, where more detailed illustrations are given for each case.

A. AOC-by-Fabrication

AOC-by-fabrication is characterized by some known working mechanisms of a natural phenomenon to be abstracted and upon which models are built. Works in the field of Artificial Life (ALife) provide a firm foundation for such endeavor, as the definition of ALife shows.

“The study of man-made systems that exhibit behaviors characteristic of natural living systems.” [7]

“... a field of study devoted to understanding life by attempting to abstract the fundamental dynamical principles underlying biological phenomena, and recreating these dynamics in other physical media – such as computers – making them accessible to new kinds of experimental manipulation and testing. ” [8]

Some well known instances of ALife include visual arts [9], L-System [10], [11] and Tierra [12]. These systems aim at replicating some natural behaviors.

Building on the experience of systems modeling, AOC algorithms are used for solving hard computational problems. For example, in the commonly used version of genetic algorithm [13] in the family of evolutionary algorithms, the process of sexual evolution is simplified to selection, recombination and mutation, without the explicit identification of male and female, for example, in the gene pool. Genetic programming [14] further modifies the chromosome representation to trees. Evolutionary programming [15] and evolution strategy [16], on the other hand, are closer to asexual reproduction with the addition of constraints on mutation and the introduction of mutation operator evolution respectively. Despite these simplification and modification, evolutionary algorithms capture the essence of natural evolution and are proven global optimization techniques. Another successful algorithm that has been applied in similar domains is the Ant System [17]. It mimics the food foraging behavior of ants [3]. Other

examples include the Selfish Gene Algorithm [18] and the Culture Algorithm [19] simulates the Selfish Gene principle [20] and the development of culture in a society, respectively. The following three examples aim to demonstrate the steps in formulating and using AOC-by-fabrication algorithms.

1) *Solving Computational Constraint Problems:* Liu *et al.* [21], [22] have presented an autonomy oriented approach called ERA (*i.e.*, Environment, Reactive rules, and Agents) to solving constraint satisfaction problems (CSPs). This approach is intended to provide a multi-agent formulation that can be used to handle general CSPs and to find approximate solutions without too much computational cost.

Definition 9 A *constraint satisfaction problem (CSP)* consists of:

- A finite set of variables, $\mathbf{X} = \{X_1, X_2, \dots, X_n\}$.
- A domain set, containing a finite and discrete domain for each variable: $\mathbf{D} = \{D_1, D_2, \dots, D_n\}, \forall i \in [1, n], X_i \in D_i$.
- A constraint set, $\mathbf{C} = \{C(R_1), C(R_2), \dots, C(R_m)\}$, where each R_i is an ordered subset of the variables, and each constraint $C(R_i)$ is a set of tuples indicating the mutually consistent values of the variables in R_i .

In the autonomy oriented approach, distributed entities represent variables and a two-dimensional grid-like environment in which the entities inhabit corresponds to the domains of the variables. Thus, the positions of the entities in such an environment constitute the solution to a CSP. The distributed multi-agent system self-organizes itself, as each individual entity follows its behavioral rules, and gradually evolves toward a global solution state.

Autonomy Oriented Models for the Queens: Based on the two general principles of ‘survival of the fittest’ - poor performers will be washed out, and ‘law of the jungle’ - weak performer will be *eaten* by stronger ones, the AOC-by-fabrication method is applied to solve a benchmark constraint satisfaction problem [23]. The N-queen problem aims to allocate N queens on an $N \times N$ chessboard such that no two queens are placed within the same row, column and diagonal. Based on the rules of the problem, a model is formulated in the following manner. Each queen is modeled as an autonomous entity in the system and multiple queens are assigned to each row in the chessboard. This is to allow competition between the queens in the same row such that the queen with the best strategy survives. The system calculates the number of violated constraints for each position on the grid. This represents the environmental information for all queens for making movement decisions, which is restricted to positions on the same row. Queens are given three movement strategies. The ‘randomized move’ strategy allows the queen to select randomly a new position. The *least-move* strategy selects the position with the least number of violations. The *coop-move* strategy promotes cooperation between queens by excluding positions that will attack those queens that one wants to cooperate. These strategies are selected probabilistically.

An initial energy is given to each queen. A queen will ‘die’ if its energy falls below a predefined threshold. Energy will change in two ways. When a queen moves to a new position that violates the set constraint with m queens, it loses m units of energy. This will also cause those queens that attack this new position to lose 1 unit of energy. The intention is to encourage the queens to find a position with the least violations. The ‘law of the jungle’

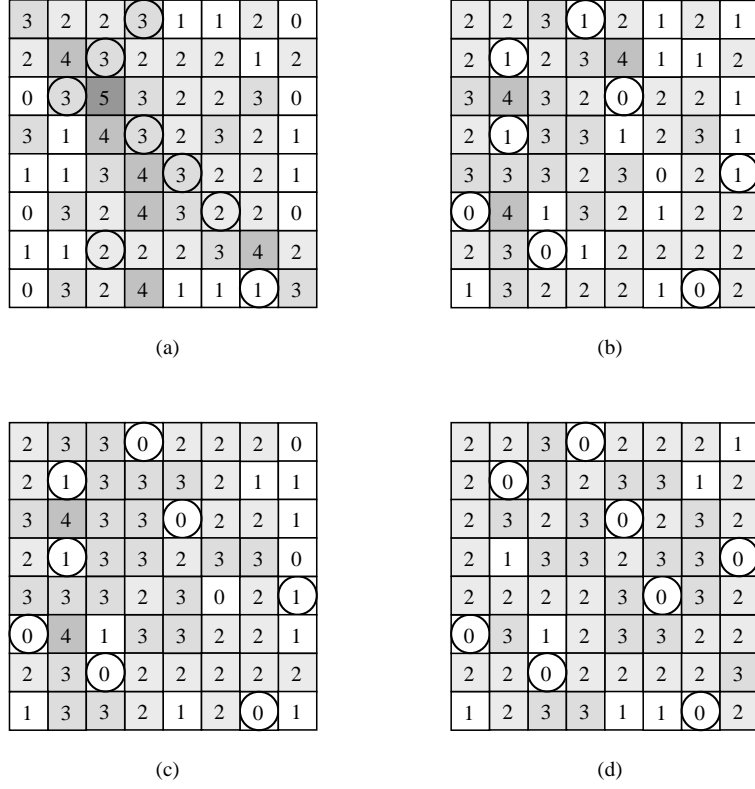


Fig. 1. (a) s_0 at time step 0 (initialization). (b) s_1 at time step 1. (c) s_2 at time step 2. (d) s_3 at time step 3, which is an exact solution state [21].

is implemented by having two or more queens occupying the same grid position to *fight* for the occupancy. The queen with the highest energy will win and eat the loser(s) by absorbing all the energy of the loser(s).

The above model is able to solve a 7,000-queen problem in few seconds using a moderate hardware configuration. Experimental results show that the ‘survival of the fittest’ principle helps to find an optimal solution much more quickly due to the introduction of competition. Moreover, randomized move is indispensable as it helps the system to come out of local minima, although giving a high chance of making randomized moves will lead to chaotic behavior. Finally, the probability of using *least-move* and *coop-move* should be synchronized and increased with the size of the problem.

An Example [21]: Figure 1 presents some snapshots from an 8-queen problem experiment. Here each circle signifies an entity. The number on the lattice gives the corresponding violation number. The darker the color of a lattice, the larger the violation number of that position will be. First, in the initialization of time step 0 in Figure 1(a), eight entities are randomly placed onto the rows. In this particular case, none of the entities is at a *zero-conflict-position*. Five of them are at the positions of *violation*= 3. Two entities are at the positions of *violation*= 2. One

entity is at the position of $violation=1$. Obviously, the assignment according to this state is not a solution. For entity a_1 at position (4, 1), we can observe that entity a_2 at (3, 2) and entity a_3 at (2, 3) are both in the same diagonal as a_1 , and entity a_4 is in the same column as a_1 . So the position where a_1 stays has the violation number of 3.

Between time step 0 and time step 1, most entities can move to a better position that reduces the violation number. At time step 1, eight entities have moved to a better position. In this assignment, four variables (*i.e.*, X_3 , X_6 , X_7 , and X_8) satisfy all the constraints applicable to them. Two pairs, $\langle X_1, X_5 \rangle$ and $\langle X_2, X_4 \rangle$, cannot satisfy the constraints: X_1 and X_5 are in the same diagonal, and X_2 and X_4 are in the same column. Obviously, the assignment in state s_1 is much better than the assignment in state s_0 .

From time step 1 to time step 2, the following moves have occurred: $s_1 \Rightarrow a_4$ stays, a_5 *least-move* to (6, 5), a_3 stays, a_6 stays, a_2 stays, a_7 stays, a_1 stays, and a_8 stays $\Rightarrow s_2$. In this assignment, six variables (*i.e.*, X_1 , X_3 , X_5 , X_6 , X_7 , and X_8) satisfy all the constraints related to them, while the pair of $\langle X_2, X_4 \rangle$ cannot satisfy each other.

From time step 2 to time step 3, the moves can be summarized as follows: $s_2 \Rightarrow a_4$ *least-move* to (8, 4) but all other entities remain at the same positions $\Rightarrow s_3$. An exact solution state is reached.

2) *Feature Search and Extraction:* The specific search problem to be considered in this example is the following: An environment, denoted by \mathcal{S} , contains a homogeneous region with the same physical feature characteristics. This region is referred to as a goal region. The feature characteristics of the goal can be evaluated based on some measurements. Here, the term *measurement* is taken as a generic notion; the specific quantity that it refers to will depend on the nature of applications. For instance, it may refer to the grey-level intensity of an image in the case of image processing. The task of autonomous entities in \mathcal{S} is to search the feature locations of the goal region. The entities can recognize and distinguish feature locations, if encountered, and then decide and execute their next step reactive behavior.

Autonomy Oriented Models in Feature Extraction: In the AOC-based approach, an entity checks its neighboring environments, *i.e.*, small circles as in Figure 2(a), and selects its behavior according to the concentration of a particular region elements. If the concentration is within a certain range, then the current location satisfies a triggering condition. This further activates the reproduction mechanism of the entity.

Taking a border-tracing entity for example, if an entity of border sensitive class reaches a border position, then this entity will inhabit at the border and proceed to reproduce both within its immediate neighboring region and inside a large region, as illustrated in Figures 2(b) and (c).

Image Segmentation: Image segmentation requires identification of a homogeneous region within the image. However, homogeneity can be at varying degree at different parts of the image. This presents problems to conventional methods such as split-and-merge that segments the image by iterative partitioning of non-homogeneous regions and simultaneous merging of homogeneous ones [25], [26]. An autonomy oriented approach has been used to tackle the same task [27]. Autonomous entities are deployed to the 2-D representation of the image. Each entity is equipped with the ability to assess the homogeneity of the region within a predefined locality. Specifically, homogeneity is defined by the relative contrast, regional mean and region standard deviation of the grey-level intensity. When a non-homogeneous region is found, the autonomous entity will diffuse to another pixel in a certain direction

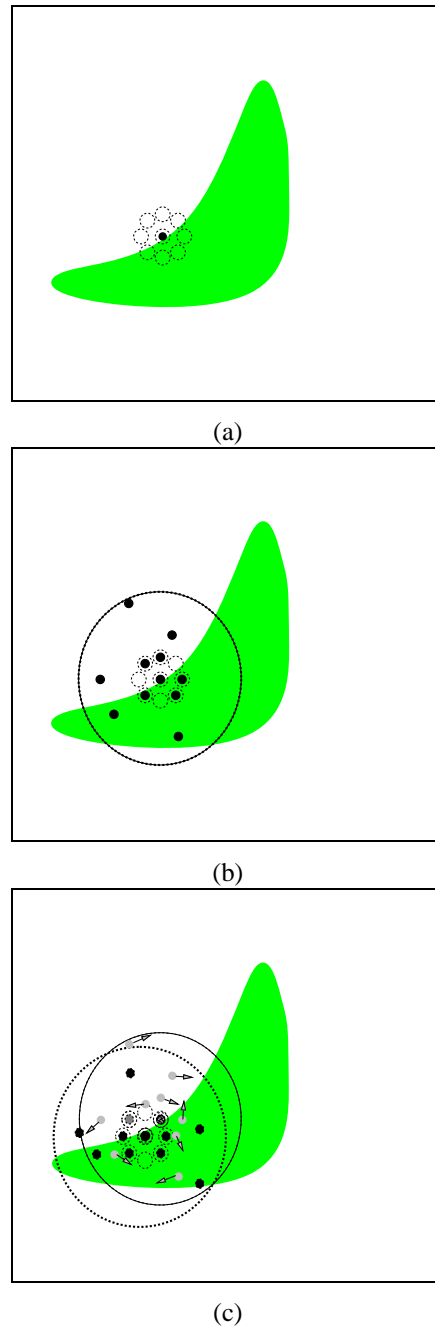


Fig. 2. An illustration of the behaviors of cellular entities. (a) As an entity, which is marked as a solid circle, moves to a new location in its local environment, it senses its neighboring locations, marked by dotted circles in this example. (b) When a triggering condition has been satisfied, as the location of the entity is right next to the border of a shaded region, the entity will self-reproduce some offspring entities within its local region. (c) At the following time step, the offspring will diffuse to new locations. By doing so, some of them will encounter new border feature locations as well and thereafter self-reproduce more entities. On the other hand, the entities that cannot find any border features after a given number of diffusion steps will be automatically turned off [24].

within the local region. In contrast, when an entity locates a homogeneous region within the range of the pixel it presently resides, it replicates (breeds) itself to give rise to certain number of offspring and delivers them to its local region in a certain direction.

The breeding behavior enables the newly created offspring to be distributed near the pixels where the region is found to be homogeneous, so that it is more likely to find the extension to the current homogeneous region. Apart from breeding, the entity will also label the pixel found to be homogeneous. If an autonomous entity fails to find a homogeneous region during its life span (a predefined number of steps) or wandered off the search space during diffusion, it will be marked inactive.

In summary, the stimulus from the pixels will direct the autonomous entities to two different behavioral tracts: breeding and pixel labeling, or diffusion and decay. The directions of breeding and diffusion are determined by their respective behavioral vectors, which contain weights (between 0 and 1) of all possible directions. The weights are updated by considering the number of successful siblings in the respective directions. An entity is considered to be successful if it has found one or more pixels that are within a homogeneous region during its life time. This method of direction selection is somewhat similar to herding behavior that considers only local information. A similar technique has also been applied to a feature extraction tasks such as border-tracing and edge detection [28].

An Example: In order to examine the effectiveness of multiple classes of entities in the simultaneous detection of significant image segments, Liu *et al.* [24], [27], [28] have conducted several experiments in which various classes of entities are defined and used to extract different homogeneous regions from an image, such as the example given in Figure 3 ($t = 0$). For this image segmentation task, 1,500 entities (500 from each of three classes) are randomly distributed over the given image. Figure 3 presents a series of intermediate steps during collective image segmentation. Figure 3 ($t = 50$) gives the resultant markers as produced by the different classes of entities.

Computational Efficiency: In AOC-based image segmentation, the *computational costs* required can be estimated by counting how many active entities are being used over time (*i.e.*, the entities whose ages do not exceed a given life span). For the above-mentioned collective image-segmentation task, we have calculated the total number of active entities in each class that have been involved over a period of 50 time steps, as given in Table I. It can readily be noted that the total number of active entities (*i.e.*, computational steps) involved in extracting a homogeneous region is less than the size of the given image, $526 \times 197 = 103,622$.

3) *Synthesizing Collective Autonomy:* Ant colonies are able to collect objects (such as food or dead ants) and place them in particular places. Collective behaviors in a complex system offer the possibility of enhanced task performance, increased task reliability, and decreased cost cover traditional complex systems. Much work to date in collective robotics focuses on limited cases, such as flocking and foraging. Typical entities in such experiments either use manually-built (non-learning) controllers [29], or perform learning in simulation [30] or relatively simple physical domains/environments [31]. One way to generate robust collective behaviors is to apply biologically-inspired adaptive algorithms at the team level, and on the individuals level, the environment plays a central role in activating a certain basic behavior at any given time (It draws on the idea of providing the robots with

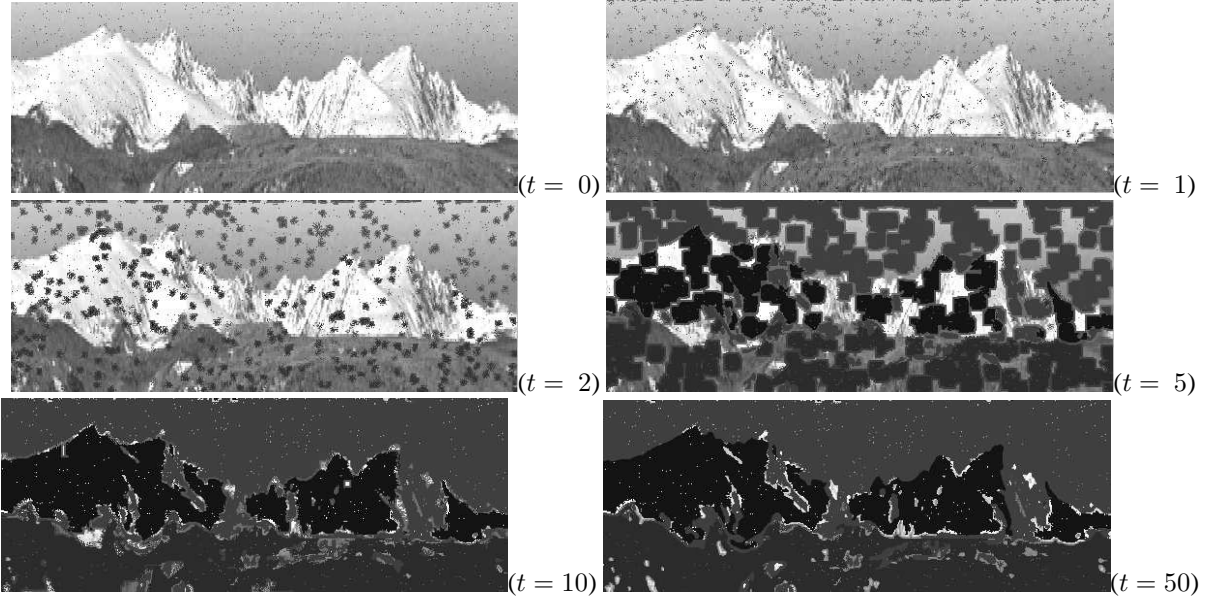


Fig. 3. Segmenting a landscape image that contains three complex-shaped regions [24].

TABLE I

THE NUMBER OF ACTIVE ENTITIES USED IN LABELING THE HOMOGENEOUS REGIONS OF A LANDSCAPE IMAGE.

Class	# of active entities used (time step = 1 ~ 50)
Class-1	47,037
Class-2	75,473
Class-3	48,837

a range of basic behaviors and letting the environment determine which behavior is more suitable as a response to a certain stimulus.). The integration of learning methods can contribute significantly to the design of a team of self-programming robots for some predefined tasks. Those individual robots can automatically program task-handling behaviors to adapt to the dynamic changes in a collective manner in their task environment.

World Modeling: Liu and Wu [32] have developed and evaluated an AOC-based method for collective world-modeling with a group of mobile robots in an unknown or less structured environment. The goal is to enable the group robots to cooperatively perform the map building task with fewer sensory measurement steps, that is, to construct the potential field map as efficiently as possible. The following issues are addressed in developing the proposed world-modeling method:

- How to formally define and represent the reactive behaviors of mobile robots and their underlying adaptation mechanisms to enable the dynamical acquisition of the group behaviors?
- How to solve the problem of collective world-modeling (*i.e.*, potential field map-building) in an unknown

robot environment based on self-organization principles?

Artificial potential field (APF) theory states that for any goal-directed robot in an environment that contains stationary or dynamically moving obstacles, an APF can be formulated and computed, taking into account an attractive pole at the goal position of the robot and repulsive surfaces of the obstacles in the environment. Using APF, any dynamic changes in the environment can be modeled by updating the original artificial potential field. With APF, the robot can reach a stable configuration in its environment by following the negative gradient of its potential field.

An important challenge in the practical applications of the APF methodology is that evolving a stable APF is a time-consuming learning process, which requires a *large* amount of input data coming from the robot-environment interaction. The distributed self-organization method for collective APF-modeling with a group of mobile robots begins with the modeling of local interactions between the robots and their environment, and then applies a global optimization method for selecting the reactive motion behaviors of the individual robots, with an attempt to maximize the overall effectiveness of collectively accomplishing a task.

The main idea behind self-organization based collective task-handling is that multiple robots are equipped with a repository of behaviors in such a way as to create some desirable global order, (*e.g.*, the fulfillment of a given task). For instance, mobile robots may independently interact with their local environment. Based on their performance (*e.g.*, distributed proximity sensory measurements), some global world model of an unknown environment (*i.e.*, global order) can be dynamically and incrementally self-organized.

Collective Autonomy: In the case of collective world-modeling, the self-organization method is designed as follows: Suppose that a robot moves to location \mathbf{P}_0 and measures its distances to the surrounding obstacles of its environment in several directions (\mathcal{N}). These measurements are recorded in a sensing vector, $\mathcal{S}_0 = [D_1^0, D_2^0, \dots, D_i^0, \dots, D_{\mathcal{N}}^0]$, with respect to location \mathbf{P}_0 . The robot will then associate this information to its adjacent locations in the environment by estimating the proximity values in the neighboring locations. The estimated proximity of any location \mathbf{P}_j inside the neighboring region of \mathbf{P}_0 to a sensed obstacle will be calculated as follows: $\hat{D}_i^j = D_i^0 - \rho_j \cdot \cos\beta$ ($i = 1, 2, \dots, \mathcal{N}$), where $\beta = \alpha_0^{(i)} - \alpha_j$. $\alpha_0^{(i)}$ and α_j denote the polar angle of the sensing direction and that of location \mathbf{P}_j , respectively. \hat{D}_i^j is an estimate for \mathbf{P}_j based on the i th direction sensing value. D_i^0 is the current measurement taken from \mathbf{P}_0 in the i th direction. Thus, the estimated proximity values for location \mathbf{P}_j can be written as: $\hat{\mathcal{S}}_j = [\hat{D}_1^j, \hat{D}_2^j, \dots, \hat{D}_i^j, \dots, \hat{D}_{\mathcal{N}}^j]$. We define a confidence weight for each element of $\hat{\mathcal{S}}_j$, that is, a function of the distance between a robot and location \mathbf{P}_j , or specifically, $w_j = e^{-\eta\rho_j^2}$, where η is a positive constant. ρ_j is the distance between the robot and location \mathbf{P}_j .

The potential field estimate at location \mathbf{P}_j will be computed as follows: $\hat{U}_j^t = \sum_{i=1}^{\mathcal{N}} e^{-\lambda\hat{D}_i^j}$, where λ is a positive constant. Thus, at time t , a set of potential field estimates, $\Omega_t^j = \{\hat{U}_j^{t_1}, \hat{U}_j^{t_2}, \dots, \hat{U}_j^{t_k}\}$, can be derived by k robots with respect to location \mathbf{P}_j ; that is,

$$\Omega_t^j \leftarrow \Omega_{t-1}^j \cup \mathcal{Q} \quad (1)$$

where Ω_{t-1}^j denotes the set of potential field estimates for location \mathbf{P}_j at time $t-1$, and $\mathcal{Q} = \hat{U}_j^{t_k}$, where subscript k indicates that the potential value is estimated based on the measurement of the k th robot. Ω_t^j is associated with a confidence weight set: $W_t^j = \{w_j^{t_1}, w_j^{t_2}, \dots, w_j^{t_k}\}$.

Hence at time t , an acceptable potential field value can readily be calculated as follows:

$$U_j^t = \begin{cases} \hat{U}_j^{t_i} & \exists i \in [1, k], w_j^{t_i} = 1, \\ \sum_{i=1}^k \hat{U}_j^{t_i} \cdot \bar{w}_j^{t_i} & \text{otherwise} \end{cases} \quad (2)$$

where $\bar{w}_j^{t_i}$ denotes a normalized weight component of W_t^j , i. e., $\bar{w}_j^{t_i} = \frac{w_j^{t_i}}{\sum_{n=1}^k w_j^{t_n}}$.

Adaptation: In order to optimize the efficiency of the above-mentioned self-organization based collective task-handling, we need an adaptation mechanism for distributed autonomous robots to dynamically generate and modify their group cooperative behaviors based on some group performance criteria. The selected (*i.e.*, high-fitness) cooperative behaviors are used to control the individual robots in their interactions with the environment.

In order to evaluate the group fitness, we identify two situations involved in the evolution: One is spatial diffusion when the inter-distance between robots i and j , d_{ij} , is less than or equal to a threshold, T_2 , and another is area coverage when $d_{ij} > T_2$. In either situation, we can use a unified direction representation of robot proximity, denoted by θ_i , that indicates a *significant proximity direction* of all proximity stimuli to robot i . Having identified the two situations in group robots, we can reduce the problem of behavior evolution into that of acquiring *two individual reactive motion behaviors*: One for spatial diffusion and another for area coverage, respectively. Both reactive behaviors respond to proximity stimuli as defined in terms of a unified significant proximity direction.

The fitness function will consist of two terms: one is called *general* fitness, denoted by f_g , and another is called *special* fitness, denoted by f_s . The general fitness term encourages the group robots to explore the potential field in new, less confident regions, and at the same time, avoid repeating the work of other robots. It is defined as follows:

$$f_g = \prod_{i=1}^m \left\{ (1 - \max\{w_i^{t_k}\}) \prod_{j=1}^{m_e} \sqrt[4]{d_{ij} - T_1} \right\}, \quad (3)$$

where $\max\{w_i^{t_k}\}$ denotes the maximal confidence weight corresponding to the location of robot i . m denotes the number of robots that are grouped together during one evolutionary movement step (of several generations). m_e denotes the number of robots that do not belong to m and have just selected and executed their next behaviors. d_{ij} denotes the distance between robots i and j , which is greater than a predefined distance threshold, T_1 .

Two special fitness terms will be defined corresponding to the performance of spatial diffusion and area coverage:

$$f_{s1} = \prod_{i=1}^{m_d-1} \prod_{j=i+1}^{m_d} \sqrt{d_{ij} - T_2} \quad (\text{spatial_diffusion}) \text{ and} \quad (4)$$

$$f_{s2} = \frac{\sqrt{\Delta\mathcal{V}}}{\prod_{i=1}^{m_c} \zeta_i} \quad (\text{area_coverage}), \quad (5)$$

where m_d denotes the number of spatially-diffusing robots whose inter-distances d_{ij} have become greater than the distance threshold, T_2 . $\Delta\mathcal{V}$ denotes the total number of locations visited by a group of m_c area-covering robots

based on their selected motion directions. ζ_i denotes a significant proximity distance between robot i and other robots in the environment.

4) *Summary:* The AOC-by-fabrication examples shown above have the following common characteristics:

1. There is a population of homogeneous individuals, each is characterized by a set of behavior rules, and individuals differ only by the parameters of the behavior rules;
2. The composition of the population changes over time, by the process analogous to birth (amplification of the desired behavior) and death (elimination of the undesired behavior);
3. Interaction between individuals is local, neither global information nor central executive to control the behavior selection is needed;
4. Behavior selection is probabilistic;
5. The environment is dynamic and acts as the center of information relating to the current status of the problem and as place holder for information sharing between individuals;
6. Local goal of individual drives the selection of local behavior;
7. Global goal is represented by a universal fitness function which gives a measure of the progress of the computation.

B. AOC-by-Prototyping

With the help of a blueprint, engineers can build a model of a system in an orderly fashion. When insufficient knowledge about the mechanism how the system works, it is difficult, if not impossible, to build such a model. Assumptions about the unknown workings have to be made in order to get the process starts. Given some observable behavior of the desired system, designers can verify the model by comparing that of the model with the desired features. This process will have to be repeated several times before a good, probably not perfect, prototype is found. This is AOC-by-prototyping. Apart from obtaining a working model of the desired system, an important by-product of the process is the discovery of the mechanisms that are unknown when the design process first started. This view is shared by researchers developing and testing theories about society and social phenomena [33], [34]. Others have also used similar methods to study highway traffic flow [35], find solutions to traffic jam [36], [37], crowd control in sports grounds [38] and crowd dynamics in a smoky room where fire has broken out [39].

1) *Understanding Self-Organized Regularities:* The Internet is growing in size, *i.e.*, number of Web pages, everyday. At the same time, more and more people are getting ‘connected’. A good understanding of the browsing behavior of all Web surfers has many important implications to network designers and portal engineers.

Regularity Characterization: Researchers have identified several interesting, self-organized regularities related to the Web, ranging from the growth and evolution of the Web to the usage patterns in Web surfing. Many regularities are best represented by characteristic distributions following either a Zipf-like law [40] or a power law. That is, if probability P of a variant taking value k is proportional to $k^{-\alpha}$ where α is from 0 to 2. A distribution presents a heavy tail if its upper tail declines like a power law [41].

Random-walk models [42], [43] and Markov-chain models [44], [45] have been used to simulate statistical regularities as empirically observed from the Web. However, these models do not relate the emergent regularities to the dynamic interactions between users and the Web, nor do they reflect the inter-relationships between user behavior and the contents or structure of the World Wide Web. The issues of user interest and motivation to navigate on the Web are among the most important factors that directly determine the navigation behaviors of users [46].

With an AOC-based approach to regularity characterization, Liu and Zhang [47], [48] have taken one step further by proposing a new computational model of Web surfing that takes into account the characteristics of users, such as interest profiles, motivations, and navigation strategies. By doing so, they attempt to answer the following questions:

- Is it possible to experimentally observe regularities similar to empirical Web regularities if we formulate the aggregation of user motivation? In other words, is it possible to account for empirical regularities from the point of view of motivation aggregation?
- Are there any navigation strategies or decision-making processes involved that determine the emergence of Web regularities, such as the distributions of user navigation depth?
- Will different navigation strategies or decision-making processes lead to different emergent regularities?
- Will the distribution of Web contents as well as page structure affect emergent regularities?

In order to answer the above questions, they have developed a white-box model. The features of this autonomy oriented model are: First, it incorporates the behavioral characteristics of Web users with measurable and adjustable attributes; second, it exhibits the empirical regularities as found in Web log data; and third, the operations in the model correspond to those in the real-world Web surfing.

AOC-Based Regularity Characterization: In the AOC-based regularity characterization, we view users as *information foraging entities* inhabiting in the Web space. The Web space is a collection of websites connected by hyperlinks. Each website contains certain information contents, and each hyperlink between two websites signifies certain content similarity between them. The contents contained in a website can be characterized using a multi-dimensional *content vector* where each component corresponds to the relative information weight on a certain topic. In order to build an artificial Web space that characterizes the topologies as well as connectivities of the real-world Web, we introduce the notion of an artificial website that may cover contents related to several topics and each topic may include a certain number of Web pages. Such a website may also be linked to other websites of similar or different topics through URLs. Thus, the Web space is generated by assigning topics to each Web page according to a certain statistical distribution. The variance of the distribution controls how similar all the pages are.

As users are the main subject to be studied, each of them are simulated in the system by associating with them an interest vector, again generated randomly based on a statistical distribution with certain variance. Therefore, there is a parameter to control the degree of overlap in interest between users.

Motivational Support Aggregation: When an information searching entity finds certain websites in which the content is close to its interested topic(s), it will become more ready to *forage* to the websites at the next level, that is, it gets more *motivated* to *surf* deeper. On the other hand, when the entity does not find any interesting information after some foraging steps or it has found enough contents satisfying its interests, it will stop foraging and leave the Web space. In order to model such a motivation-driven foraging behavior, here we introduce a support function, S_t , which serves as the driving force for an entity to forage further. When the entity has found some useful information, it will get rewarded, and thus the support value will be increased. As the support value exceeds a certain threshold, which implies that the entity has obtained a sufficient amount of useful information, the entity will stop further foraging. In other words, the entity is satisfied with what it has found. On the contrary, if the support value is too low, the entity will lose its motivation to forage further and thus leave the Web space.

Specifically, the support function is defined as follows:

$$S_{t+1} = S_t + \theta \cdot \Delta M_t + \phi \cdot \Delta R_t \quad (6)$$

where

- S_t : support value at time step t ,
- ΔM_t : motivational loss at time step t ,
- ΔR_t : reward received at time step t ,
- θ, ϕ : weights of motivation and reward, respectively.

Validation of Autonomy Oriented Models: In order to validate their autonomy oriented model, Liu and Zhang [47], [48] have conducted several experiments. In the experiments, two measurements are of particular interests: the surfing-depth (step) distribution and the rank-frequency distribution of link clicks. The frequency of link clicks refers to the number of times for which a link is passed through by the entities. It is also called *link-click-frequency*. In the experiments, foraging entities are categorized into three different groups: recurrent user who is familiar with the Web structure, rational user who is new to the website but knows clearly what he/she is looking for, and random user who has no strong intention to get anything and just ‘wander’ around.

Figures 4 and 5 present the statistical distributions of foraging depth and link-click-frequency obtained for recurrent and rational entities, respectively.

It is interesting to observe from Figures 4(b) and 5(b) that the distributions of link-click-frequency exhibit a power law. A very similar result on the distribution of website popularity has been empirically observed and reported in [49].

Liu and Zhang [47], [48] have also used real-world Web log datasets to compare their corresponding empirical distributions with those produced by the information foraging entities. The first dataset is NASA Web log that recorded all HTTP requests received by the NASA Kennedy Space Center Web server in Florida from 23:59:59 August 3, 1995 to 23:59:59 August 31, 1995. The distributions of user surfing depth and link-click-frequency for the NASA dataset are shown in Figures 6(a) and (b), respectively.

From the above-mentioned synthetic and empirical results, we can note that the autonomy oriented models can

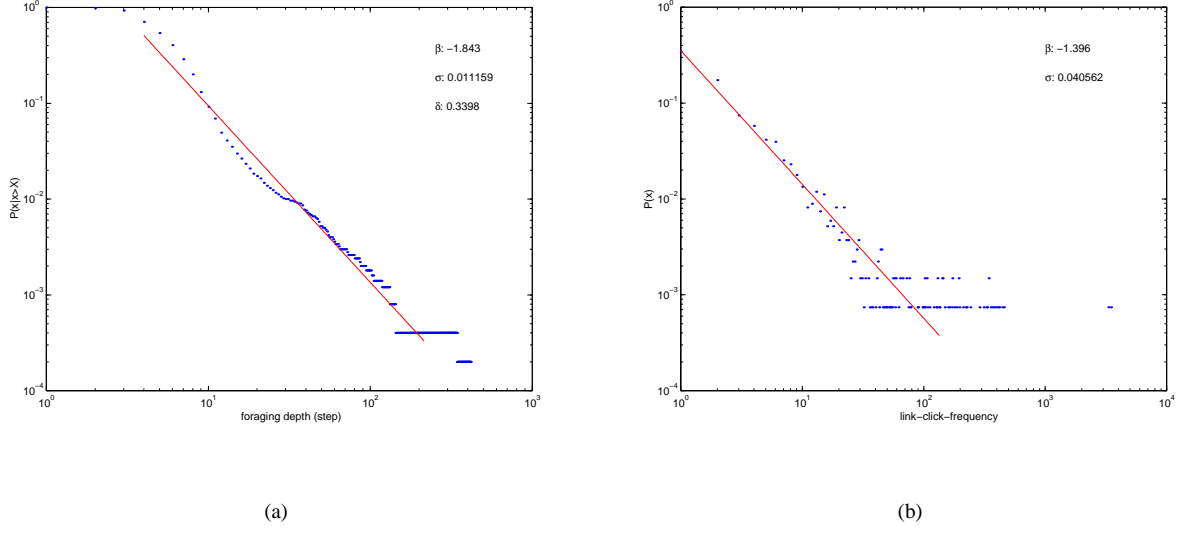


Fig. 4. Recurrent entities. (a) Cumulative distribution of entity foraging depth (step), where ‘.’ corresponds to experimental data and ‘—’ corresponds to a linear-regression fitted line. The tail of the distribution follows a power-law distribution with power $\beta_c = -1.843$ and the residual of linear regression $\sigma = 0.01$. δ denotes entities’ satisfaction rate (*i.e.*, the ratio of the number of satisfied entities to the total number of entities what have surfed on the Web). (b) Distribution of link-click-frequency (link click refers to the total times for which entities pass through a link). The tail follows a power-law distribution with power $\beta_l = -1.396$, as obtained by weighted linear regression [47].

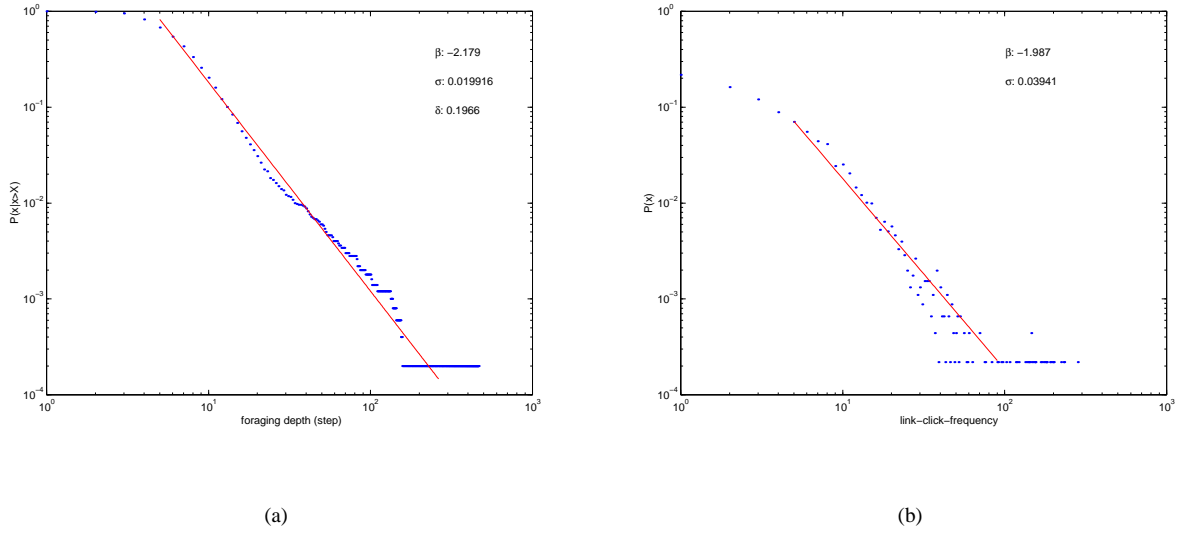


Fig. 5. Rational entities. (a) Cumulative distribution of entity foraging depth (step), where ‘.’ corresponds to experimental data and ‘—’ corresponds to a linear-regression fitted line. The tail of the distribution follows a power-law distribution with power $\beta_c = -2.179$ and the regression residual $\sigma = 0.02$. δ denotes entity’s satisfaction rate. (b) Distribution of link-click-frequency. The distribution follows a power-law distribution with power $\beta_l = -1.987$, as obtained by weighted linear regression [47].

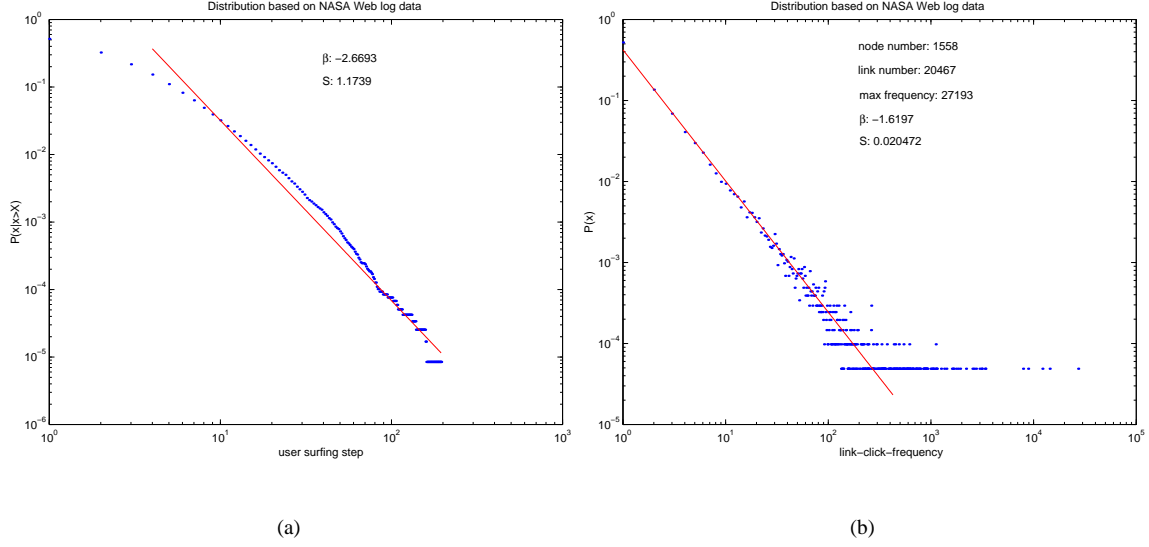


Fig. 6. Distributions based on real-world NASA Web log data. (a) Cumulative distribution of user surfing step. The distribution follows a heavy tail with the tail's scale of $\beta_c = -2.669$. The linear-regression residual s is about 1.17. (b) Distribution of link-click-frequency. It agrees well with a power law of power $\beta_l = -1.62$, as obtained by weighted linear regression [47].

readily generate power-law distributions in surfing step and link-click-frequency, which are similar to those from the real-world, and hence they offer a white-box explanation to the self-organized Web regularities.

In addition to the distributions of user steps in accessing pages and link-click-frequency, they are also interested in the *distribution of user steps in accessing domains or topics* – an issue of great importance that has never been studied before. Figure 7 presents the distributions of steps in accessing domains by recurrent and rational entities, respectively. From Figure 7, we can readily conclude that the cumulative probability distribution of entity steps in accessing domains follows an exponential function.

They have further obtained an empirical dataset that records user behavior in accessing the domains of a website. The dataset is a Web log file for the *Microsoft* corporate website, recording the domains or topics of www.microsoft.com that anonymous users visited in a one-week timeframe in February 1998. The distribution of user steps in accessing domains is shown in Figure 8. If we compare Figure 7 with Figure 8, we can note that the domain-visit regularity generated by our model characterizes the empirically observed domain-visit regularity well.

Using the autonomy oriented models described above, several critical factors that affect a user experience can also be analyzed by altering some parameters of the models. For instance, it has been revealed that there exists an optimal accessibility (distance) between two Web pages.

2) *Summary*: AOC-by-prototyping can be seen as an iterated application of AOC-by-fabrication with the addition of parameter tuning at each iteration. The difference between the desired behavior and the actual behavior of a prototype then becomes the guideline to parameter adjustment. The process can be summarized, with reference to the summary of AOC-by-fabrication in Section III-A.4, as follows:

1. Behavior rules can be changed from one prototype to the next;

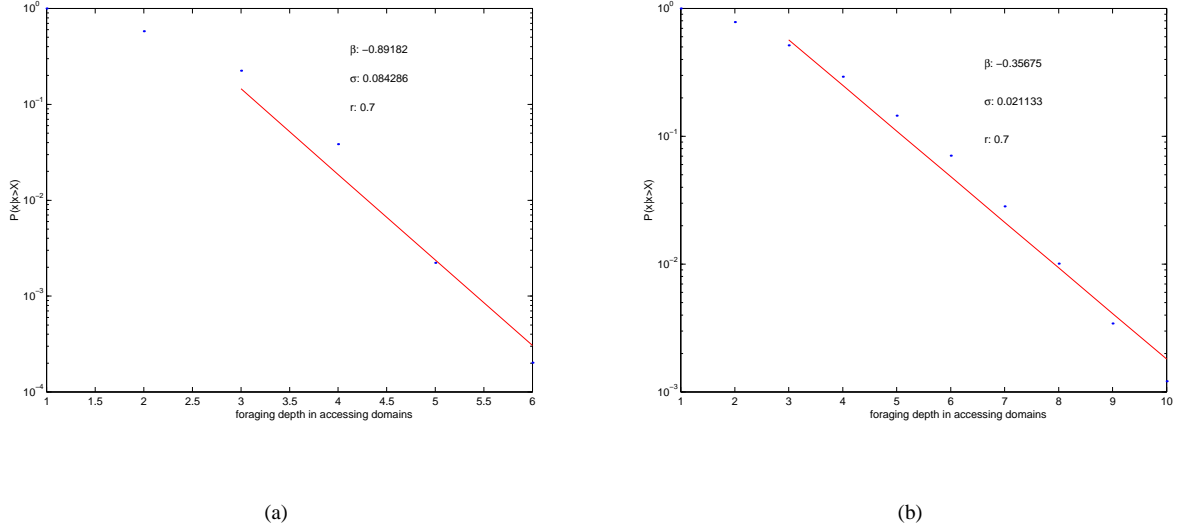


Fig. 7. Entities visiting *domains*. (a) Cumulative distribution of foraging depth in accessing domains by *recurrent* entities, where ‘.’ corresponds to experimental data and ‘—’ corresponds to a linear-regression fitted line. The distribution follows an exponential function with exponent $\beta_d = 0.892$ and residual $\sigma = 0.08$. (b) Cumulative distribution of foraging depth in accessing domains by *rational* entities. The distribution follows an exponential function with a *smaller* exponent $\beta_d = 0.357$ and residual $\sigma = 0.02$ [47].

2. Interaction rules can be changed from one version to the next;
3. There is an additional step to compare the desired behavior with the current emergent behavior;
4. A new prototype is built by adopting steps 1 and 2 above and repeating the whole process.

C. AOC-by-Self-Discovery

AOC-by-self-discovery emphasizes the ability of an AOC-based computational system to find its own way to achieve what AOC-by-prototyping can do. The ultimate goal is to have a fully automated algorithm that can adjust its own parameters for different application domains. In other words, the AOC itself becomes autonomous.

Adaptive evolutionary algorithms [50]-[53] have been proposed that automatically tune some of the parameters related to the algorithms, such as mutation rate [16], [54]-[58], crossover rate [59], [60], crossover operator [61], [62], mutation operator [63]-[66], and population size [58]. They usually track changes in progress measures such as online and offline performance [52], the ratio of average fitness to best fitness, and the ratio of the worst fitness to average fitness, among others.

Meta-EA is another group of self-improving EA that does not rely on the specific instruction of the designer [67]. An EA [68]-[71] or another intelligent system [72]-[75] is used to control a population of EA in the way similar to an EA optimizing the parameters of the problem at hand.

Other examples of AOC-by-self-discovery include the evolution of emergent computation where a GA is used to evolve the rules of a cellular automata for a synchronization task [76]-[78] and for generating test patterns for hardware circuit [79]. A variant of the latter example employs a selfish gene algorithm [80].

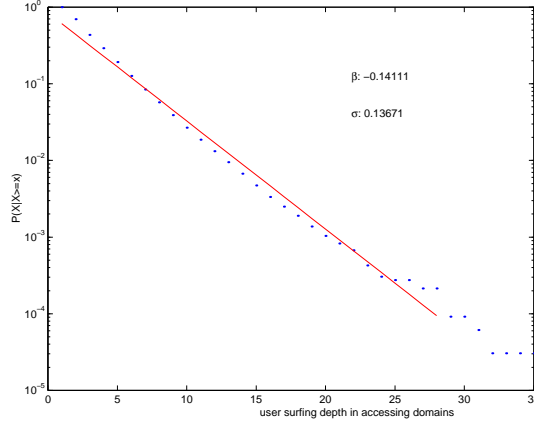


Fig. 8. Real-world *Microsoft* Web log data. Cumulative distribution of user step in accessing domains. The distribution follows an exponential function with $\beta_d = -0.141$. The regression residual σ is about 0.137 [47].

1) *Global Optimization*: Global optimization [81] aims at finding the optimal solution to a function $F(x)$ where $x = \{x_1, x_2, \dots, x_n\}^T$ is an n -dimensional vector representing the parameters of function. The optimal solution is then represented by $F(x^*)$ such that

$$F(x^*) < F(x) \quad \forall x \quad (7)$$

Many algorithms have been developed over the years to tackle this problem [82]-[84]. This is a challenging task because: First, the landscape of the function to be optimized is unknown and search algorithms are likely to be trapped in suboptimal solutions; second, the dimensionality of the function presents problems to a search algorithm. The task is further complicated if one needs to adjust the various parameters of the search algorithms.

Search by Diffusion: Inspired by diffusion in nature, Tsui and Liu [85], [86] have developed an AOC-based method, called *Evolutionary Diffusion Optimization* (EDO), to tackle the global optimization task. A population of entities are used to represent candidate solutions to the optimization problem in hand. The goal is to build a collective view of the landscape of the search space by sharing information between entities. Specifically, each entity performs search in its local proximity and captures the ‘direction’ information of the landscape in a *probability matrix* – the likelihood estimate of success with respect to direction of motion for each object variable.

Autonomy Oriented Evolutionary Diffusion Model: Evolutionary diffusion optimization (EDO) defines three types of local behavior for each entity, namely *diffuse*, *reproduce* and *aging*. Free ranging entities that are searching for a position better than its birthplace are called *active* entities. Those entities that have already become parents are called *inactive* entities.

Entities in EDO explore uncharted locations in the solution space by *diffusion*. *Rational move* refers to the kind of diffusion where an entity modifies its object vector by drawing a random number for each dimension of the object vector. Each random number is then used to choose between the set of fixed steps according to the probability matrix. Adjustment is then made by adding the product of the number of steps and the size of a step to the entry

in question in the object vector. This process is repeated until all dimensions of the object vector are covered. The updated object vector then becomes the new position of the entity in the solution space.

As an entity becomes older, it becomes eagerer to find a better position. It, therefore, will probabilistically decide to act wild and take a *Random Walk* with probability given by $\exp(-\frac{lifespan-age}{\alpha})$ where α is a scaling factor. An entity will first choose randomly a direction of diffusion for each dimension, *i.e.*, either no change, towards the upper bound, or towards lower bound. In case a move is to be made, a new value between the chosen bound and the current value is then picked randomly. The process ends when all the dimensions of the object vector are updated.

At the end of an iteration, the fitness of all active entities are compared with that of their parents, which have (temporarily) become stationary. All entities with higher fitness will *reproduce* via asexual reproduction – a reproducing entity replicates itself a number of times and send the new entities off to a new location by *rational move*. Parents and their offspring share the same probability matrix. It is only when an entity becomes a parent then a new probability matrix will be created for it, which is an exact copy of the parent's updated one. Sharing the probability matrix between parents and siblings enables entities from the same family to learn from each other's successes as well as failures.

Aging is the process by which consistently unsuccessful entities are eliminated from the system. This is controlled by a *lifespan* parameter. Exception are granted to those entities whose age has reached the set limit but has above-average fitness. On the other hand, entities whose fitness is at the lower 25% of the population will be eliminated before their lifespan expire.

Adaptation: Search algorithms need a scheme to implement the strategy that says: 'good' moves need to be rewarded while 'bad' moves should be discouraged. All entities in EDO maintain a close link between parents and offspring via sharing the probability matrix. Therefore, it is very easy for EDO to implement the above strategy and EDO has two feedback mechanisms for updating the probability matrix of parent. *Positive feedback* increases the value of the entry in the probability matrix that corresponds to the 'good' moved. The update rule is:

$$p'_{ij} = \frac{p_{ij} + \delta}{1 + \delta}, \quad (8)$$

where p_{ij} refers to the probability of object variable i and step multiplier j that relates to the previous movement(s) and δ is an fixed adjustment factor. In contrast, *negative feedback* reduces the relevant probabilities that relate to the 'bad' move using:

$$p'_{ij} = p_{ij} \times (1 - \delta). \quad (9)$$

While negative feedback is exercised after each diffusion, positive feedback can only take place after an entity has become a parent. Note also that all probabilities are normalized using their respective sum after updating.

EDO also adapts the *step_size* parameter, which decides the amount of change during *diffusion*, over time based on the performance measurement of the population. *Step_size* is reduced if the population has not improved over a period of time. Conversely, if the population has been improving continuously for some time, *step_size* is increased. The rationale is that the entities in the neighborhood of a minimum need to make finer steps for careful exploitation, while using a larger *step_size* during a period of continuous improvement attempts to speed up the search.

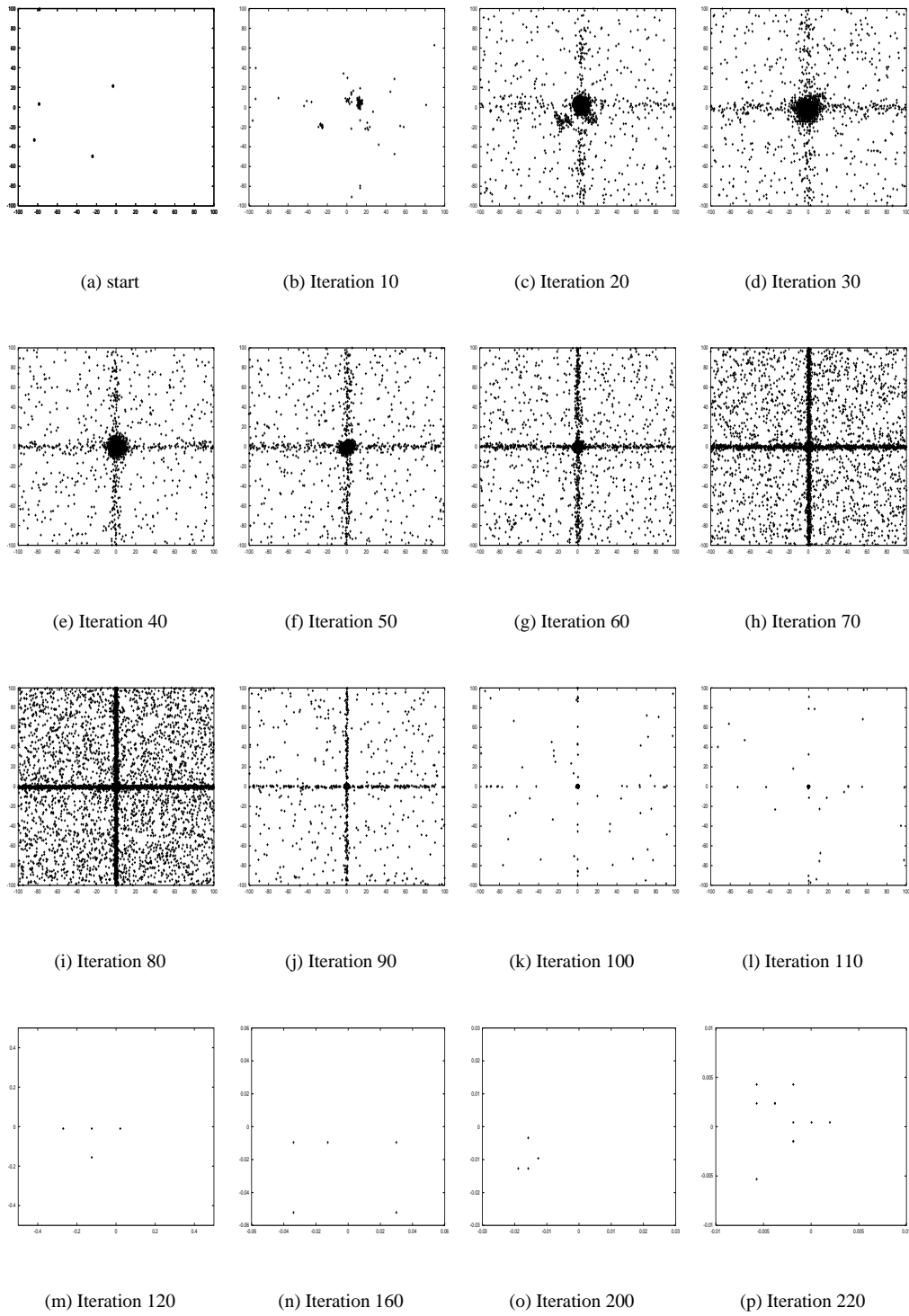


Fig. 9. Distribution of entities in the solution space for optimizing the 2-D version of F_1 . Plot (a)-(l) shows the range between ± 100 in both axes. The range for plots (m)-(p) are ± 0.5 , ± 0.06 , ± 0.02 and ± 0.01 , respectively. The number of entities shown in plots (m)-(p) are 4 out of 5, 5 out of 5, 4 out of 6 and 11 out of 14, respectively.

An Example: Figure 9 shows the distribution of entities used to optimize $F_1(x) = \sum_{i=1}^2 x_i^2$. Initially, five entities were positioned randomly in the solution space - three with x-value around -80, one around -20 and one around zero - but none is within the central 20 by 20 zone around the origin. As the search progresses to generation 10, more and more entities appear inside the central zone. This trend continues between generation 10 and generation 80.

The diameter of the occupied area increases between generations 10 and 30 due to entities exploring the solution space, and shrinks between generations 30 and 80 when the direction information is being learned. It can also be observed that there is an increase in the number of entities along the axes, which can be considered as suboptimal solutions since one of the two variables is at its lowest value with respect to F_1 .

The number of entities peaks at around generation 80 but dropped drastically in the next 10 generations. This is because entities not around the origin or along the axes have much lower fitness than the population average and are gradually eliminated. The second phase of search sees entities along the axes being eliminated as they are occupying a suboptimal position. As the entities continue to move towards the origin, entities that are further away from the origin are eliminated. The last four plots in Figure 9 zoom in to the origin with an increasing scale where the majority of the best entities are located.

2) *Summary:* Full automation of the prototyping process is achieved by having an autonomous entity to control another level of autonomous entities. The examples described above show that AOC-by-self-discovery is indeed a viable proposition. The steps for engineering this kind of AOC algorithm is the same as those stated in Section III-A.4 with the addition of one rule:

1. Systems parameters are adapted according to some performance measurement.

D. Related Work

Autonomy oriented computation certainly shares commonality with other research areas. This section attempts to draw attentions to some clear distinctions between them.

Artificial Life emphasizes the simulation of life in a computer setting. It, therefore, falls short of its use as a computational method for problem solving. On the other hand, an AOC does not necessarily need to produce life-like behavior as natural phenomena are usually abstracted and simplified. *Agent Based Simulation* (ABS) shares a similar goal with AOC-by-prototyping – finding an explanation to observed phenomena. Again, there is no computational problem to be solved in ABS.

There are at least three examples of multi-entity computational system. *Multi-agent Systems* for distributed decision making [87], [88] is one example. Works in this field attempt to solve a computational task by delegating responsibilities to groups of entities. These entities usually are *heterogeneous* entities and different groups have different roles. For example, in the Zeus collaborative agent building framework, entities are divided into utility entities, such as name server, facilitators and visualizer, and domain-level agents [89]. In relation to this, behavior of individual is preprogrammed and can sometimes be complex. Complicated issues such as negotiation and coor-

dination are of paramount importance. All these are usually part of the systems design and hence require a lot of human intervention.

Another example of multi-entity computation is *Ecology of Computation* [90], [91] where a population of heterogeneous computational system shares partial solutions to the common problem at hand. These individual problem solvers tackle the problem with different methods and, therefore, have different internal representation of the problem/solution. While it is shown that this approach is efficient in solving a problem, the coordination needs to be articulated carefully so that different internal representations can be translated properly.

Distributed Constraint Satisfaction Problem (distributed CSP) [92]-[94] was proposed to solve CSP problems in a distributed manner. Specifically, the asynchronous backtracking algorithm assigns a variable to an agent and a directed graph called a constraint network is constructed to represent the relationships between the variables. The asynchronous weak-commitment search algorithm enhanced the above algorithm by adding a dynamic priority hierarchy. In the event of conflicts, the hierarchy structure is changed so that a sub-optimal solution can be constructed first and then the final solution incrementally. This is different from the AOC approach where competition and behavior amplification are used to discover the best strategy for an autonomous entity. Furthermore, AOC does not need this hierarchy to find a solution as all autonomous entities are treated equally and only local neighbors (might change over time) matter in checking conflicts. While the distributed CSP algorithms are event-driven (act upon the receipt of messages from other agents), AOC is clock-driven with synchronous update.

Swarm Intelligence [95], [96] is another good demonstration of AOC-by-fabrication in that it emphasizes the social insect metaphor for problem solving through local interaction of relatively simple entities. It has not yet been applied to areas that other AOC approaches have been used, such as using it to explain complex behavior and discovery of problem solvers.

E. Engineering Issues

Autonomy oriented computation can be viewed as a methodology for engineering a system for solving hard computational problems. Various approaches of AOC systems differ, in terms of engineering, in at least five different aspects. Firstly, it is often the case in AOC that analogies are drawn from the natural or physical world. Therefore, the working mechanism becomes the key basis of modeling. Secondly, having a detailed knowledge of the working mechanism means more detailed work by the designer of the AOC systems in implementing the algorithms. Thirdly, the uncertainty with the outcome of the computational system varies according to the knowledge of the actual mechanism. Fourthly, the computational cost of any AOC algorithm is expected to be higher than a customized algorithm for a particular problem as AOC algorithms tend to be problem independent. Finally, the whole problem solving process depends upon the complexity of the problem as well as the degree of understanding of the problem. Figure 10 contrasts the relative rank of the AOC approaches according to the above five criteria.

Following the AOC-by-fabrication approach requires detailed knowledge of the working mechanism and high degree of designer's involvement to implement it. On the other hand, systems built this way require the least

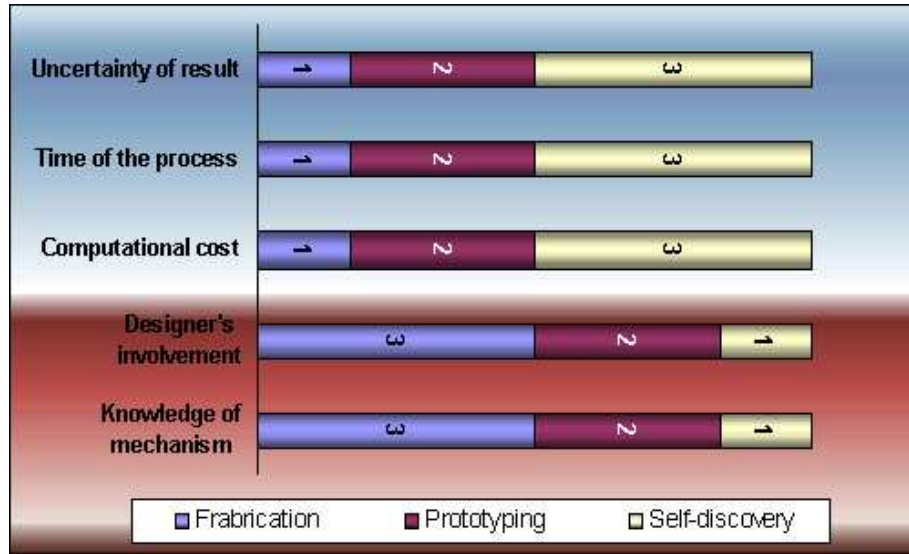


Fig. 10. The relative rank of each AOC approach according to different engineering issues, 1 being the easiest and 3 the hardest.

computational cost and it is safe to say that they will work. The overall time needed to build such systems is also at the minimum.

Building an AOC-by-self-discovery system, on the other hand, does not require much knowledge about the working mechanisms because most of them are not known. As a result, the designer does not need to spend too much time crafting the system. The cost is shifted to computational time where the system discovers for itself what need to be tuned and it naturally takes a long time to achieve the task. It follows that the risk of failure (not able to find a solution) is the highest with an AOC-by-self-discovery system.

The level of difficulty in constructing an AOC-by-prototyping system is in between the aforementioned two, as some information is already known. The need for designer's involvement is also limited by this. More computational power and time than an AOC-by-prototyping system are generally needed and incur higher risk of failing to find a solution.

Any computational algorithms can be assessed in at least the following areas: efficiency, generality, robustness, completeness and computational cost. Efficiency measures the effectiveness of an algorithm to find an optimal solution and how quickly such a solution is found. Generality measures the applicability of the algorithm to different problem domains. Robustness concerns the sensitivity of an algorithm in terms of its parameter settings. Completeness of an algorithm assesses its ability to search the solution space. Finally, computational cost refers to the algorithm's requirement on computer cycles before a solution is found. It can be a combination of CPU time and memory.

Autonomy oriented computation is a general problem solving methodology as it is not designed for a particular problem. AOC algorithms can also be considered complete as it is able to cover the 'more promising' areas of the solution space effectively. On the front of efficiency, AOC algorithms can usually find a 'good enough' solution

within a very short period of time. Given enough time, the global solution could be found. Directly related to this is the computational cost. AOC algorithms usually have a population of elements and requires a lot of computation cycles for evaluating each candidate and accumulate enough positive feedback to make a solution stand out. Robustness is usually high but sometimes is hampered by the formulation of the solution and the fitness function, as it directly affects the quality measure of a solution and indirectly the progress.

F. Guiding Principles

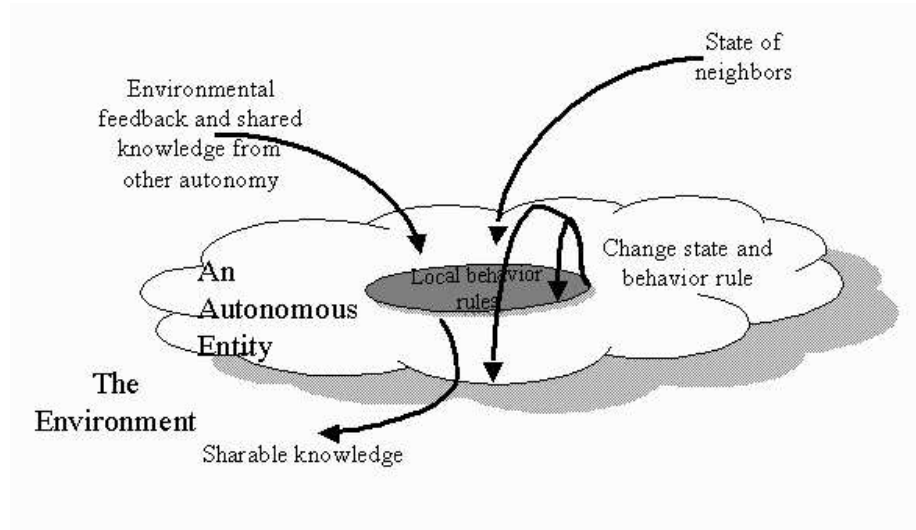


Fig. 11. Schematic diagram of an autonomous entity.

1) *Elements of an AOC System:* An AOC algorithm contains, naturally, a population of autonomous entities and the rest of the system is referred to as the environment. An autonomous entity consists of a detector (or a set of it), an effector (again, there can be a set of it) and a repository of local behavioral rules (see Figure 11).

The detector receives information related to the neighbors and the environment. In the simulation of a flock of birds, this information includes speed, direction they are heading and the distance between the entities in question. Details of the content and format of this information need to be defined according to the system to be modeled or the problem to be solved. The notion of neighbor can be defined in terms of position (the bird in front, to the left and to the right), distance (a radial distance of two grids), or both (the birds up to 2 grids in front). Environmental information conveys the status of certain feature that is of interest to the autonomous entity, for example, the presence of food. The environment can also help to carry sharable local knowledge.

The effector of an autonomous entity refers collectively to the device for expressing actions. These actions can be either changes in an internal state, displaying certain behavior externally or making changes to the environment in which the entity inhabits. An important role of the effector, as part of the local behavior model, is to facilitate implicit information sharing between autonomous entities.

Central to an autonomous entity is the behavior rules that governs how it should act or react to the information

collected by the detector from the environment and its neighbors. These rules decide to what state should this entity change and also what local knowledge should be released via the effector to the environment. An example of sharable local knowledge is the pheromone in the Ant system. This information is untargeted and the communication via the environment is undirected; any ant can pick up the information and react according to its own behavior model.

In order to adapt itself to the problem without being explicitly told in advance, an autonomous entity needs to modify its behavior rules over time. This is the learning capability of the individual.

It is worth noting that randomness plays a part in the decision making process of an autonomous entity, despite the presence of a rule set. This is to allow an autonomous entity to explore uncharted territories even in the presence of mounting evidence that it should exploit a certain path. On the other hand, randomness helps the entity to resolve conflict in the presence of equal support to the suggestions to act in different manner, and avoid being stuck by choosing an action randomly.

The environment acts as the domain in which autonomous entities roam. This is a static view of the environment. The environment of an AOC algorithm can also act as the ‘noticeboard’ where the autonomous entities can post and read local information. In this dynamic view, the environment is changing all the time. For example, StarLogo [97] has patches in the environment that know how to grow food or evaporate ants’ pheromone. Sometimes it provides feedback to the entity regarding their actions. For example, in the N-queen constraint satisfaction problem, the environment can tell a particular queen how many constraints are violated in its neighborhood after a move is taken. This, in effect, translates a global goal to a local goal for individual entities. The environment also keeps the central clock that helps to synchronize the actions of all autonomous entities, if necessary.

2) *Characteristics of AOC Algorithms:* While the three approaches to autonomy oriented computation varies in their goals, they represent a natural progression from detailed engineering work to an automated process. The following principles attempts the characterize the major features in an AOC-based system.

Homogeneity	Complex systems are made up of self-similar individuals that have similar behavior rules. Their rule-set may differ only in the parameters of the rules but not in the structure of the rules.
Simplicity	Behavior model of each individual in the population is simple.
Locality	Interaction is strictly local although the notion of locality can be physical as well as logical.
Implicit	Another form of interaction comes from the implicit knowledge sharing between individuals via a common environment. This environment can also have dynamical behaviors such as spreading the shared news or aging the news contributed by the individuals.
Uncertainty	Behavior such as rule selection is not purely deterministic. There is usually a certain degree of randomness in the decision-making process.
Amplification	Desirable behavior is amplified while undesirable ones are eliminated via mechanisms such as birth and death. This is also the result of positive feedback from the environment.
Recursion	Complex behaviors are aggregated from simpler ones through many iterations.
Openness	New types of entities can be added seamlessly into the complex systems.

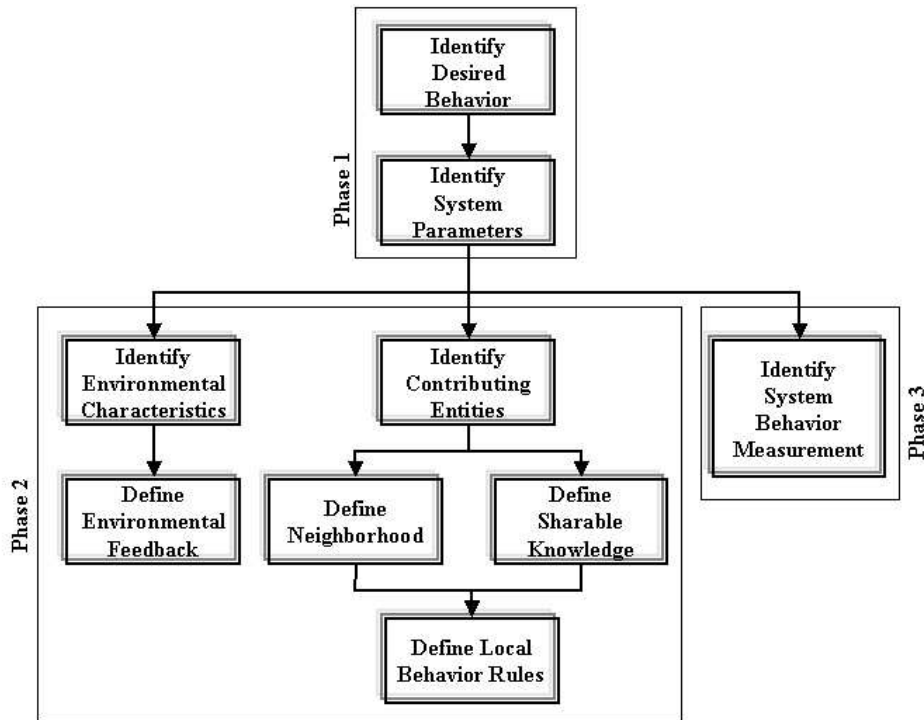


Fig. 12. Block diagram of major components of an AOC that need to be defined.

3) *The Autonomy-Modeling Process*: Formulation of an AOC algorithm involves three phases (see Figure 12). The first phase, natural system identification, can be viewed as the precursor to actual systems modeling and concerns the selection of an appropriate analogy from the natural and physical world. There are two tasks involved: *identify desired behavior* and *identify system parameters*. Choosing the right analogy is the key to the success of the AOC-based system and the right system usually presents itself through its behaviors. Once an appropriate analogy is chosen, details such as the number of entities to run and the length of time to run the simulation need to be decided.

The second phase, artificial system construction, involves all elements in the AOC-based system. This phase is divided into two major sub-phases: autonomous entity modeling and environment modeling. The *identify contributing entities* task is the first and the most important task in this phase. Designers are required to choose the level of detail to be modeled, that is appropriate to the problem at hand. The *define neighborhood* task defines a distance measurement in the solution space within which local interactions can occur and local information can be collected. The *define sharable knowledge* task defines what kind of information and in what form will the entities share with others in their neighborhood. The last task concerning the entities is *define local behavior rules*, which defines the ways an autonomous entity reacts to various information it collected within its neighborhood and the way it adapts its behavior rules.

The tasks that concerns the environment are *identify environment characteristics* and *define environmental feedback*. The former task concerns the role the environment plays in conveying the knowledge shared between the

autonomous entities. The latter task handles the results of the actions an autonomous entity enacted onto the environment.

The third phase, performance measurement, concerns the evaluation criteria for comparing the artificial system manifested by the AOC-based system with its natural counterpart. This relates to problem-solving and provides an indication to modify the current set of individual behavior rules. The end of this phase will trigger the next cycle, if necessary, and involves modifications to some or all AOC elements defined in the previous cycle.

G. Measure of Performance

Randomness is an important factor in any self-organizing system such as those formulated using the AOC principles. On the one hand, it helps the autonomous entities to explore new areas in the solution space. On the other hand, it introduces certain degree of uncertainty regarding the outcome of the simulations. Therefore, it is important to have a concrete assessment of the progress in an AOC-based system.

Emergence is a property of an AOC that is not pre-programmable. Therefore, it is not possible to measure it directly from the parameters that characterize the system and its elements. Wright *et al.* [98] have suggested a measure of emergence by likening a self-organized system with a set of non-linear springs and dampers to represent the local interaction between the entities. It was argued that factors such as spatial and velocity coherence in a flock of birds are two observable behaviors due to the same underlying mechanism. It was further argued that there is a strong correlation between abrupt behavior changes and emergent behaviors while the system parameters are being changed smoothly. Therefore, these factors need to be considered together. A Hamiltonian model using Shannon entropy is formulated and a dimensionality measure Ω is defined based on the entropy, which is linked directly to the phase transition hypothesis and has been used as feedback to a GA to guide the adaptation.

Ronald *et al.* have defined a qualitative emergence test [99] based on two different viewpoints and a surprise factor. Specifically, from the designer's point of view, a design language \mathcal{L}_1 is used to describe the local interactions. From the observer's point of view, the observed behavior is described by an observation language \mathcal{L}_2 . Surprise is defined as the difference between the expected outcome of the \mathcal{L}_1 as perceived by the observer and the observation by the observer using \mathcal{L}_2 .

Complexity of an AOC may have direct implication on whether a problem is solvable or not. Standish [100] has pointed out that the complexity of a complex system is context dependent. It can be measured by the entropy of the system that is characterized by the length of the description of the system, the size of the alphabet needed to encode the description and the size of all equivalent descriptions. This principle has been applied to measure the complexity of the artificial life system Tierra [101]. Nehaniv has also defined a hierarchical complexity measure for a biological system [102]. A maximal complexity measure is defined as the least number of active computing levels required in order to build a finite transformation semigroup hierarchically from simple components. It increases as the computing power increase but in a bounded manner.

Evolvability of an AOC-based system refers to its ability to evolve an optimal solution [103]. Zhang and Shimohara [104] have defined an index (entropy measure) in the experiments with Tierra to measure the evolutionary

action over time. This index is a weighted entropy of size distribution of Tierran organism. The weight function is set to be the ratio of Tierran size in adjacent time steps. The experimental results show that during the period of strong evolutionary action, the entropy increases. The entropy would then drop to a low level and stay there when evolutionary action dampens. Nehaniv has proposed measuring evolvability as the rate of complexity (defined above) increase [105]. By considering complexity in longer terms, it was shown that the proposed evolvability measure is bounded between one and one plus the complexity of the ancestor of the individual in question, depending on the type of steps that have occurred during the evolution from the said ancestor to the said individual.

H. Simulation Environments

Performing experiments with AOC requires either writing tailor-made programs or using a simulation environment. This section reviews two publicly available and widely used environments.

*StarLogo*¹ [97] is a simulation environment for explorations of collective behavior. It is a parallel implement of the Logo programming language. StarLogo was first developed on a Mac platform but a Java-based version has been released. It is designed to run on a single machine. A StarLogo world consists mainly of two classes of objects: the environment and the creatures. The environment is a grid of patches that are dynamic and can perform functions like diffusing pheromone. The patches can be inhibited by many autonomous creatures called turtles, or other species named by the programmer. Each turtle as well as patch can be programmed with certain behavior. The status of any patch and turtle can be queried and the objects related to patches can be altered by any turtle. Status update of all object in StarLogo is synchronous. The latest version of StarLogo comes with a visualizer where users can design the world of the simulation and interact with the system using buttons and slidebars. Many simulations have been implemented such as termites, slime molds, traffic jam, among others. It is a good starting point to experiment with AOC. However, computational systems have not yet been reported.

*SWARM*² [106], [107] is a larger scale simulation environment where users can perform simulations of complex systems. The basic unit is a swarm, which is a collection of entities with a schedule of events over the entities. Many swarms can be defined in the system and a swam may consists of other swarms. This is similar to the hierarchical relationship between emergent autonomy and synthetic autonomy without the requirement of emergent behavior. Everything in the *SWARM*, including the environment, are entities with specific behaviors. This provides high degree of flexibility for engineers to experiment with different setups. There are three free libraries in *SWARM* and they make the modeling process easier by hiding a lot of simulation-specific technicalities such as order of action execution and visualization, allowing modelers to concentrate on the problem-specific issues change as agent behavior and events of a particular simulation.

IV. RESEARCH ISSUES

Research in Autonomy Oriented Computation has been active in the past decade or so with many successful applications. However, many areas need to be explored in order to exploit the benefits of AOC. This section

¹website: <http://www.media.mit.edu/starlogo>

²website: <http://www.swarm.org>

sketches some directions, both theoretical and practical, of future research.

A. Theoretical Issues

1) *Concept, Methodology & Theory*: New approaches to system dynamics and performance measurement are particularly needed so that clearer guidelines can be developed to help practitioners gain better insights into AOC, and AOC-by-self-discovery in particular. Measures of emergence, evolvability and self-organization, tractability and scalability of AOC are useful for tracking the progress of AOC. Theory on the formation of roles and social structure in a community of AOC-based systems would expand the capability of a homogeneous AOC-based systems to heterogeneous ones.

2) *Algorithm Comparison*: Strengths and weaknesses of AOC need to be formally assessed by comparing with other multi-agent paradigms to establish a clear insight into the benefits of AOC. Benchmark problems should also be identified for this purpose.

B. Practical Issues

There are a few decisions need to be made when implementing autonomy oriented computation. They include: hardware and software environment, update schedule, management services and visualization. This section discusses the relevancy of these issues to the three approaches of AOC.

1) *Hardware and Software Environment*: Autonomy oriented computation usually involves more than one object in the system. These objects include the autonomous entities and the dynamic environment. Implementing the computational system in a single processor machine requires the support of virtual parallelism. Modern operating systems and programming languages support multi-threading. This allows slicing up CPU cycles and allocates them to the individual processes that represent objects in the system. When multiple processors are available, individuals can be allocated to different processor on the same machine or across the network of processors with the support of some facilities such as Parallel Virtual Machine (PVM)³ [108] and Message Passing Interface (MPI)⁴ [109]. However, it requires a central control program to coordinate task allocation and result consolidation. This makes the parallel implementation of, for example, a genetic algorithm possible without requiring an expensive parallel machine.

With the popularity of network programming, mobile agents can be sent to run on any machine over the Internet, provided permissions are granted by the host. Running simulated evolution in this way has been attempted [110] and is a good starting for those pursuing this line of research.

2) *Update Schedule*: An individual entity changes its state at every step based on its current state and that of its neighbors. In a synchronous update scenario, the current state of all individuals are frozen to allow all individuals to obtain state information and change their state, if appropriate. The current state of the whole system is then updated and the system clock ticks, marking the beginning of the next time step. In a parallel implementation,

³website: <http://www.epm.ornl.gov/pvm/>

⁴website: <http://www.erc.msstate.edu/labs/hpcl/projects/mpi/>

synchronization might become an overhead too big to handle. Asynchronous update can therefore allow processes on each processor to proceed as if it were a single processor machine. However, the choice of an update schedule and the choice of a hardware platform are related. If a multi-process hardware environment is chosen, synchronous update would slow the simulation down as all processes have to start and stop at the same time.

3) *Management Services:* With the large number of autonomous entities in the system, autonomy oriented computation needs to keep track of the creation and deletion of objects. Moreover, a centralized messaging mechanism is needed to facilitate message passing between objects. A central clock is also required to help the entities to synchronize with state updates, no matter it is synchronous or asynchronous. Some centralized whiteboard may also be needed if the entities were to share information in an implicit way and to contain a global view of the system's status. This whiteboard may also be used to simulate the dynamic environment in systems such as the Ant system [17].

4) *Visualization:* Visualization is a good way for people running simulations to 'see' what is going on with the experiment. Items of interest that are related to autonomy include actual movement, state, actions taken, fitness, age, etc. On the other hand, some global information such as population size, best fitness, average fitness, etc, alongside any progress measurements such as measure of emergence, evolvability, diversity and convergence, are also of interest to modelers. Visual display of all such information is of tremendous help to modelers to obtain a quick view of the system.

V. SUMMARY

We have outlined in this article a new computational paradigm called Autonomy Oriented Computation. The usage of AOC is vast as judged from many examples described above. AOC has three general approaches with different objectives. AOC-by-fabrication is similar to construction with a blueprint where a known phenomenon is abstracted and replicated as a computational problem solving technique. AOC-by-prototyping presents a trial-and-error approach to finding explanations to some observations via an autonomy oriented system. Human involvement is certainly intensive to fine-tune system parameters. AOC-by-self-discovery, on the other hand, is an autonomous problem solver that can tune its own settings to suit the problem at hand. It requires less human intervention than AOC-by-prototyping but represents the highest degree of uncertainty as not until a solution is found, it is difficult to tell when it will stop.

This article has also discussed issues related to the implementation of an AOC algorithm and has compared AOC with some related work. There are still many issues to be addressed by the research community, some of which have also been described. This article has just scratched the tip of an iceberg and hope to stimulate more interests in this excitingly new field.

Acknowledgement

The authors wish to sincerely thank Dr. David Fogel, Editor-in-Chief, the anonymous reviewers, and the Associate Editor for their valuable comments and detailed suggestions that have helped us improve the presentation of

this article. The authors would also like to acknowledge the experimentation assistance by Yichuan Cao, Huijian Zhou, Jing Han, Shiwu Zhang, and Jianbing Wu in some of the research projects mentioned in the article. Thanks are also due to Xialong Jin for painstakingly proofreading the manuscript. The present project has been in part supported by HKBU FRG grants.

REFERENCES

- [1] Yoram Louzoun, Sorin Solomon, Henri Atlan, and Irun R. Cohen, "The emergence of spatial complexity in the immune system," Los Alamos Physics Archive arXiv:cond-mat/0008133 (<http://xxx.lanl.gov/html/cond-mat/0008133>), 2000.
- [2] John Casti, *Would-Be Worlds: How Simulation is Changing the Frontiers of Science*, John Wiley & Son, 1997.
- [3] S. Goss, R. Beckers, J. L. Deneubourg, S. Aron, and J. M. Pasteels, "How trail laying and trail following can solve foraging problems for ant colonies," in *Behavioural Mechanisms of Food Selection*, R. N. Hughes, Ed., vol. G 20 of *NATO-ASI Series*. Springer-Verlag, 1990.
- [4] N. R. Jennings and M. Wooldridge, "Software agents," *IEEE Review*, vol. 42, no. 1, pp. 17–21, January 1996.
- [5] Pattie Maes, "Modeling adaptive autonomous agents," in *Artificial Life: An Overview*, Christopher G. Langton, Ed., pp. 135–162. MIT Press, 1995.
- [6] R. A. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, pp. 139–159, 1991.
- [7] C. G. Langton, "Artificial life," in *Artificial Life*, C. G. Langton, Ed., pp. 1–47. Addison-Wesley, Redwood City, CA, 1989, Volume VI of SFI Studies in the Sciences of Complexity.
- [8] C. G. Langton, "Preface," in *Artificial Life II*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds., pp. xiii–xviii. Addison-Wesley, Redwood City, CA, 1992, Volume X of SFI Studies in the Sciences of Complexity.
- [9] Karl Sims, "Artificial evolution for computer graphics," *Computer Graphics*, vol. 25, no. 4, pp. 319–328, 1991.
- [10] Przemyslaw Prusinkiewicz, Mark Hammel, and Radomir Mech, "Visual models of morphogenesis: A guided tour," Online document, May 1997, at <http://www.cpsc.ucalgary.ca/Redirect/bmv/vmm-deluxe/Titlepage.html>.
- [11] Przemyslaw Prusinkiewicz and Aristid Lindenmayer, *The Algorithmic Beauty of Plants*, Springer-Verlag, NY, 1990.
- [12] Thomas S. Ray, "An approach to the synthesis of life," in *Artificial Life II*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds., pp. 371–408. Addison-Wesley, Redwood City, CA, 1992, Volume X of SFI Studies in the Sciences of Complexity.
- [13] John H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, 1992.
- [14] John R. Koza, *Genetic Programming : on the programming of computers by means of natural selection*, MIT Press, 1992.
- [15] Lawrence Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulation Evolution*, Wiley, NY, 1966.
- [16] H.-P. Schwefel, *Numerical Optimization of Computer Models*, Wiley : Chichester, 1981.
- [17] Marco Dorigo, Vittorio Maniezzo, and Alberto Colomi, "The ant system: Optimization by a colony of cooperative agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, no. 1, pp. 1–13, 1996.
- [18] F. Corno, M. Sonza Reorda, and G. Squillero, "The selfish gene algorithm: A new evolutionary optimization strategy," in *Proceedings of SAC'98: 13th Annual ACM Symposium on Applied Computing*, February 1998, pp. 349–355.
- [19] R. G. Reynolds, "An introduction to cultural algorithms," in *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, A. V. Sebald and L. J. Fogel, Eds. 1994, pp. 131–139, World Scientific, River Edge, NJ.
- [20] Richard Dawkins, *The Selfish Gene*, Oxford University Press, 1989.
- [21] J. Liu and J. Han, "Multi-agent oriented constraint satisfaction," *Artificial Intelligence*, 2002, in press.
- [22] J. Liu and J. Han, "ALIFE: A multi-agent computing paradigm for constraint satisfaction problems," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 15, no. 3, pp. 475–491, 2001.
- [23] Jing Han, Jiming Liu, and QingSheng Cai, "From ALife agents to a kingdom of N queens," in *Intelligent Agent Technology: Systems, Methodologies and Tools*, Jiming Liu and Ning Zhong, Eds. 1999, pp. 110–120, World Scientific.
- [24] J. Liu, *Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization, and Adaptive Computation*, World Scientific, 2001.
- [25] T. Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Science Press, 1992.
- [26] I. Pitas, *Digital Image Processing Algorithms*, Prentice Hall, Englewood Cliffs, N.J., 1993.
- [27] Jiming Liu, Y. Y. Tang, and Y. C. Cao, "An evolutionary autonomous agents approach to image feature extraction," *IEEE Transaction on Evolutionary Computation*, vol. 1, no. 2, pp. 141–158, July 1997.

- [28] Jiming Liu and Yuan Y Tang, "Adaptive image segmentation with distributed behavior-based agents," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 6, pp. 544–551, June 1999.
- [29] T. Balch and R. C. Arkin, "Motor schema-based formation control for multiagent robot teams," in *Proceedings of the First International Conference on Multi-Agent Systems*. 1995, pp. 10–16, AAAI Press, Menlo Park, CA.
- [30] T. Balch, "Learning roles: Behavioral diversity in robot teams," in *Proceedings of the AAAI-97 Workshop on Multiagent Learning*, 1997, pp. 7–12.
- [31] M. J. Mataric, "Reward functions for accelerated learning," in *Proceedings of the 11th International Conference on Machine Learning*, W. W. Cohen and H. Hirsh, Eds. 1994, pp. 181–189, Morgan Kaufmann Publishers, San Francisco, CA.
- [32] J. Liu and J. Wu, *Multi-Agent Robotic Systems*, CRC Press LLC, 2001.
- [33] Rosaria Conte and Nigel Gilbert, "Introduction: Computer simulation for social theory," in *Artificial Societies*, Nigel Gilbert and Rosaria Conte, Eds., chapter 1, pp. 1–15. UCL Press, London, 1995.
- [34] Jim Doran and Nigel Gilbert, "Simulating societies: An introduction," in *Simulating Societies: The Computer Simulation of Social Phenomena*, Nigel Gilbert and Jim Doran, Eds., chapter 1, pp. 1–18. UCL Press, London, 1994.
- [35] Dirk Helbing and Bernardo A. Huberman, "Coherent moving states in highway traffic," *Nature*, vol. 396, pp. 738–740, 24/31 December 1998.
- [36] S. Rasmussen and C. Barrett, "Elements of a theory of simulation," Tech. Rep. Working Paper 95-04-040, Santa Fe Institute, 1995.
- [37] Kenneth R. Howard, "Unjamming traffic with computers," *Scientific American*, October 1997.
- [38] G. K. Still, *Crowd Dynamics*, Ph.D. thesis, Mathematics Department, Warwick University, August 2000.
- [39] Dirk Helbing, Illés Farkas, and Tamás Vicsek, "Simulating dynamic features of escape panic," *Nature*, vol. 407, pp. 487–490, 28 September 2000.
- [40] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Cambridge, MA, 1949.
- [41] M. E. Crovella and M. S. Taqqu, "Estimating the heavy tail index from scaling properties," *Methodology and Computing in Applied Probability*, vol. 1, no. 1, pp. 55–79, 1999.
- [42] B. A. Huberman, P. L. T. Pirolli, J. E. Pitkow, and R. M. Lukose, "Strong regularities in World Wide Web surfing," *Science*, vol. 280, pp. 96–97, April 3, 1997.
- [43] R. M. Lukose and B. A. Huberman, "Surfing as an real option," in *Proceedings of the International Conference on Information and Computation Economics*, October 1998, pp. 45–51.
- [44] M. Levene and G. Loizou, "Computing the entropy of user navigation in the Web," Research Note RN/99/42, Department of Computer Science, University College London, 1999.
- [45] M. Levene, J. Borges, and G. Loizou, "Zipf's law for Web surfers," *Knowledge and Information Systems*, vol. 3, pp. 120–129, 2001.
- [46] A. Thatcher, "Determining interests and motives in WWW navigation," in *Proceedings of the Second International Cyberspace Conference on Ergonomics (CybErg1999)*, 1999.
- [47] J. Liu and S. Zhang, "Characterizing Web usage regularities with information foraging agents," *IEEE Transactions on Knowledge and Data Engineering*, 2002, to appear.
- [48] J. Liu and S. Zhang, "Unveiling the origin of Web surfing regularities," in *Proceedings of INET 2001*, 2001.
- [49] B. A. Huberman and L. A. Adamic, "Growth dynamics of the world-wide web," *Nature*, vol. 410, pp. 131, September 9, 1999.
- [50] A. J. Angeline, "Adaptive and self-adaptive evolutionary computations," in *Computation Intelligence: A Dynamic Systems Perspective*, M. Palaniswami, Y. Attikiouzel, R. Marks, D. Fogel, and T. Fukuda, Eds., pp. 152–163. IEEE Press, Piscataway, NJ, 1995.
- [51] Thomas Bäck, "Self-adaptation," in *Handbook of Evolutionary Computation*, Thomas Bäck, David B Fogel, and Zbigniew Michalewicz, Eds., pp. C7.1:1–15. Institute of Physics Publishing and Oxford University Press, 1997.
- [52] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.
- [53] Robert Hinterding, Zbigniew Michalewicz, and A. E. Eiben, "Adaptation in evolutionary computation: A survey," in *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, T. Bäck, Zbigniew Michalewicz, and X. Yao, Eds., Berlin, 1997, pp. 65–69, IEEE Press.
- [54] T. Bäck, "The interaction of mutation rate, selection, and self adaptation within a genetic algorithm," in *Parallel Problem Solving From Nature*, 1992, pp. 85–99.

- [55] Thomas Bäck and Martin Schutz, "Intelligent mutation rate control in canonical genetic algorithms," in *Proceedings of International Symposium on Methodologies for Intelligent Systems*, 1996, pp. 158–167.
- [56] H. R. Gzickman and K. A. P. Sycara, "Self-adaptation of mutation rates and dynamic fitness," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, Cambridge, MA, August 1996, vol. 2, p. 1389, MIT Press.
- [57] Michele Sebag and Marc Schoenauer, "Mutation by imitation in Boolean evolution strategies," in *Parallel Problem Solving from Nature - PPSN IV*, Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, Eds., Berlin, September 1996, pp. 356–365, Springer-Verlag.
- [58] E. A. Williams and W. A. Crossley, "Empirically derived population size and mutation rate guidelines for a genetic algorithm with uniform crossover," in *WSC2: 2nd Online World Conference on Soft Computing in Engineering Design and Manufacture*, June 1997.
- [59] Eun-Joung Ko and O. N. Garcia, "Adaptive control of crossover rate in genetic programming," in *Proceedings of the Artificial Neural Networks in Engineering (ANNIE'95)*, New York, NY, 1995, ASME Press.
- [60] Jim E. Smith and Terence C. Fogarty, "Adaptive parameterised evolutionary systems: Self adaptive recombination and mutation in genetic algorithm," in *Parallel Problem Solving from Nature - PPSN IV*, Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, Eds., Berlin, September 1996, pp. 441–450, Springer-Verlag.
- [61] Peter J. Angeline, "Two self-adaptive crossover operators for genetic programming," in *Advances in Genetic Programming*, Peter J. Angeline and K. E. Kinnear, Jr., Eds., pp. 89–110. MIT Press, Cambridge, MA, USA, 1996.
- [62] Myung-Sook Ko, Tae-Won Kang, and Chong-Sun Hwang, "Adaptive crossover operator based on locality and convergence," in *Proceedings of the 1996 IEEE International Conference on Intelligence and Systems*, Los Alamitos, CA, November 1996, pp. 18–22, IEEE Computer Society Press.
- [63] K.-H. Liang, X. Yao, and C. Newton, "Dynamic control of adaptive parameters in evolutionary programming," in *Proceedings of Simulated Evolution and Learning (SEAL98): Second Asia Pacific Conference*, 1998, pp. 42–49, Springer-Verlag.
- [64] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 2nd (extended) edition, 1994.
- [65] A. K. Swain and A. S. Morris, "A novel hybrid evolutionary programming method for function optimization," in *Proceedings of Congress on Evolutionary Computation (CEC2000)*, 2000, pp. 1369–1376.
- [66] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transaction on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.
- [67] Bernd Freisleben, "Metaevolutionary approaches," in *Handbook of Evolutionary Computation*, Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, Eds., pp. C7.2:1–8. Institute of Physics Publishing and Oxford University Press, 1997.
- [68] Thomas Bäck, "Parallel optimization of evolutionary algorithms," in *Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. 1994, pp. 418–427, Springer, Berlin.
- [69] K. A. DeJong, *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. thesis, Department of Computer and Communication Sciences, University of Michigan, 1975.
- [70] B. Freisleben and M. Härtfelder, "In search of the best genetic algorithm for the traveling salesman problem," in *Proceedings of the 9th International Conference on Control Systems and Computer Science*, 1993, pp. 485–493.
- [71] B. Freisleben and M. Härtfelder, "Optimization of genetic algorithms by genetic algorithms," in *Artificial Neural Networks and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. 1993, pp. 392–399, Springer, Wien.
- [72] Chan-Jin Chung and Robert G. Reynolds, "Knowledge-based self-adaptation in evolutionary search," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 14, no. 1, pp. 19–33, 2000.
- [73] F. Herrera and M. Lozano, "Adaptive genetic operators based on coevolution with fuzzy behaviors," Tech. Rep. DECSAI-98-01-05, Department of Computer Science and Artificial Intelligence, University of Granada, March 1998.
- [74] Michael A. Lee and Hideyuki Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed. 1993, pp. 76–83, Morgan Kaufmann, San Mateo.
- [75] A. G. Tettamanzi, "Evolutionary algorithms and fuzzy logic: A two-way integration," in *Proceedings of the Second Joint Conference on Information Sciences*, 1995, pp. 464–467.
- [76] James P. Crutchfield and Melanie Mitchell, "The evolution of emergent computation," *Proceedings of the National Academy of Sciences, USA*, vol. 92, no. 23, pp. 10742–10746, 1995.

- [77] Rajarshi Das, James P. Crutchfield and Melanie Mitchell, and James E. Hanson, "Evolving globally synchronized cellular automata," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995.
- [78] Win Hordijk, James P. Crutchfield, and Melanie Mitchell, "Mechanisms of emergent computation in cellular automata," in *Parallel Problem Solving from Nature - PPSN V*, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds., Berlin, 1998, pp. 613–622, Springer-Verlag.
- [79] S. Chiusano, F. Corno, P. Prinetto, and M. Sonza Reorda, "Cellular automata for sequential test pattern generation," in *Proceedings of IEEE VLSI Test Symposium*, April, 1997, pp. 60–65.
- [80] F. Corno, M. Sonza Reorda, and G. Squillero, "Exploiting the selfish gene algorithm for evolving hardware cellular automata," in *Proceedings of Congress of Evolutionary Computation (CEC2000)*, 2000, pp. 1401–1406.
- [81] Aimo Torn and Antanas Zilinskas, *Global Optimization*, Springer-Verlag, 1989.
- [82] Reiner Horst and Hoang Tuy, *Global Optimization : Deterministic Approaches*, Springer-Verlag, 1990.
- [83] Reiner Horst and Panos M. Pardalos, Eds., *Handbook of Global Optimization*, Kluwer Academic Publishers, 1995.
- [84] Jonas Mockus, *Bayesian Approach to Global Optimization : Theory and Applications*, Kluwer Academic, 1989.
- [85] Kwok Ching Tsui and Jiming Liu, "Evolutionary diffusion optimization, part I: Description of the algorithm," in *Proceedings of the Congress on Evolutionary Computation*, 2002, to appear.
- [86] Kwok Ching Tsui and Jiming Liu, "Evolutionary diffusion optimization, part II: Performance assessment," in *Proceedings of the Congress on Evolutionary Computation*, 2002, to appear.
- [87] Edmund H. Durfee, "Distributed problem solving and planning," in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Gerhard Weiss, Ed., pp. 121–164. MIT Press, 1999.
- [88] Tuomas W. Sandholm, "Distributed rational decision making," in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Gerhard Weiss, Ed., pp. 201–258. MIT Press, 1999.
- [89] H. S. Nwana, D. T. Ndumu, and L. C. Lee, "ZEUS: An advanced tool-kit for engineering distributed multi-agent systems," in *Proceedings of PAAM98*, March 1998, pp. 377–391.
- [90] Tad Hogg and Bernardo A. Huberman, "Better than the best: The power of cooperation," in *SFI 1992 Lectures in Complex Systems*, L. Nadel and D. Stein, Eds., pp. 163–184. Addison-Wesley, 1993.
- [91] B. A. Huberman, Ed., *The Ecology of Computation*, North-Holland, Amsterdam, 1988.
- [92] Makoto Yokoo, *Distributed Constraint Satisfaction Foundations of Cooperation in Multi-Agent Systems*, Springer Series on Agent Technology. Springer-Verlag, Berlin, 2001.
- [93] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 5, pp. 673–685, 1998.
- [94] Makoto Yokoo and Katsutoshi Hirayama, "Algorithms for distributed constraint satisfaction: A review," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 2, pp. 185–207, June 2000.
- [95] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NJ, 1999.
- [96] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for optimization from social insect behaviour," *Nature*, vol. 406, pp. 39–42, 6 July 2000.
- [97] Mitchel Resnick, *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds*, MIT Press, 1994.
- [98] W. A. Wright, R. E. Smith, M. Danek, and P. Greenway, "A measure of emergence in an adapting, multi-agent context," in *Proceedings Supplement of SAB'2000*, 2000, pp. 20–27.
- [99] Edmund M. A. Ronald, Moshe Sipper, and Mathieu S. Capcarrère, "Design, observation, surprise! a test of emergence," *Artificial Life*, vol. 5, no. 3, pp. 225–239, 1999.
- [100] Russell K. Standish, "On complexity and emergence," Los Alamos Physics Archive arXiv:nlin.AO/0101006 (<http://xxx.lanl.gov/abs/nlin/0101006>), 2001.
- [101] Russell K. Standish, "Some techniques for the measurement of complexity in terra," in *Advances in Artificial Life: 5th European Conference, ECAL 99*, Dario Floreano, Jean-Daniel Nicoud, and Francesco Mondada, Eds., Berlin, 1999, pp. 104–108, Springer.
- [102] Christopher L. Nehaniv and John L. Rhodes, "The evolution and understanding of hierarchical complexity in biology from an algebraic perspective," *Artificial Life*, vol. 6, no. 1, pp. 45–67, 2000.
- [103] Christopher L. Nehaniv, "Preface," In Nehaniv [111], pp. iii–iv, Published as University of Hertfordshire Technical Report 351.

- [104] Yongguang Zhang and Katsunori Shimohara, "A note on evolvability in tierra," In Nehaniv [111], pp. 62–65, Published as University of Hertfordshire Technical Report 351.
- [105] Chrystopher L. Nehaniv, "Measuring evolvability as the rate of complexity increase," In Nehaniv [111], pp. 66–68, Published as University of Hertfordshire Technical Report 351.
- [106] C. G. Langton, N. Minar, and R. Burkhart, "The Swarm simulation system: A tool for studying complex systems," reprint, Santa Fe Institute, March 1995.
- [107] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, "The swarm simulation system: A toolkit for building multi-agent simulations," <http://www.santafe.edu/projects/swarm/overview/overview.html>, June 1996.
- [108] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam, *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994.
- [109] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra, *MPI: The Complete Reference*, MIT Press, 1996.
- [110] R. E. Smith and N. Taylor, "A framework for evolutionary computation in agent-based systems," in *Proceedings of the 1998 International Conference on Intelligent Systems*, C. Looney and J. Castaing, Eds. 1998, pp. 221–224, ISCA Press.
- [111] Chrystopher Nehaniv, Ed., *Proceedings of the Evolvability Workshop at the Seventh International Conference on the Simulation and Synthesis of Living Systems (Artificial Life 7)*, number 351, August 2000, Published as University of Hertfordshire Technical Report 351.