Evolutionary System Induction through Multi Agent Algorithm

Dr. Nada M. A. Al Salami
E-mail: dr_nada71@yahoo.com

Abstract

This research solves system induction problems by using multi agent technique. System's specifications are used to direct solution space. Such specification gives system behaviors rather than its structure. Finite State Automata are constructing in terms of input(s), state, and output(s. The evolutionary process adapts Ant Colony Optimization algorithm to search for a good Finite State Automata that efficiently satisfies input output specifications of the problem.

Keywords: Evolutionary Computation; ACO; Multi agent; System theory.

I. Introduction

The main idea is to use the self-organizing principles to coordinate populations of artificial agents that collaborate to solve computational problems. Self-organization is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components. The rules specifying the interactions among the system's constituent units are executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system by an external ordering influence. For example, the emerging structures in the case of foraging in ants include spatiotemporally organized networks of pheromone trails. Self-organization relies on four basic ingredients [1][2]:

1. Positive feedback (amplification) is constituted by simple behavioral rules that promote the creation of structures.

2. Negative feedback counterbalances positive feedback and helps to stabilize the collective pattern: it may take the form of saturation, exhaustion, or competition.

3. Self-organization relies on the amplification of fluctuations (random walks, errors, random task-switching).

4. All cases of self-organization rely on multiple interactions. They should be able to make use of the results of their own activities as well as others' activities.

When a given phenomenon is self-organized, it can usually be characterized by a few key properties [3][4]:

1. The creation of spatiotemporal structures in an initially homogeneous medium. Such structures include nest architectures, foraging trails, or social organization.
2. The possible coexistence of several stable states (multi stability). Because structures emerge by amplification of random deviations, any such deviation can be amplified, and the system converges to one among several possible stable states, depending on the initial conditions.
3. The existence of bifurcations when some parameters are varied. The behavior of a self-organized system changes dramatically at bifurcations.

Ant Colony Optimization (ACO) is a population-based approach for solving combinatorial optimization problems that is inspired by the foraging behavior of ants and their inherent ability to find the shortest path from a food source to their nest. ACO is the result of research on computational intelligence approaches to combinatorial optimization originally conducted by Dr. Marco Dorigo, in collaboration with Alberto Colorni and Vittorio Maniezzo. The fundamental approach underlying ACO is an iterative process in which a population of simple agents repeatedly construct candidate solutions; this construction process is probabilistically guided by heuristic information on the given problem instance as well as by a shared memory containing experience gathered by the ants in previous iteration. ACO has been applied to a broad range of hard combinatorial problems. Problems are defined in terms of components and states, which are sequences of components. Ant Colony Optimization incrementally generates solutions paths in the space of such components, adding new components to a state. Memory is kept of all the observed transitions between pairs of solution components and a degree of desirability is associated to each transition depending on the quality of the

solutions in which it occurred so far. While a new solution is generated, a component y is included in a state, with a probability that is proportional to the desirability of the transition between the last component included in the state, and y itself [4].

II. Theoretical Definition

We propose theoretical model is purports to describe the behavior of a program in terms of input(s), states, and output(s). Proposed method shares some feature with two of the main approaches to system analysis: data flow, and control flow. It equates what a system means with what it does. The meaning of system **S** can be specified by set of functions from states to states; hence **S** effects a transformation:

$$\mathbf{(S)\ X_{initial} \longrightarrow X_{final}}$$

on a state vector **X,** which consists of an association of the variable manipulated by the system and their values, thus it's length equal to the number of system variables. Two features characterize state transition function **z**:

1- $z(-, -, t) = (X_{initial}, 1)$,

if t = 0

2- z (f, X, t) = z (f, z (f(t-1), X, t-1))

if t ≠ 0

To execute system **S**, transition functions are firing starting from, t = 0. Execution terminate when t > T. The concepts of reusable parameterized subsystems can be implemented by restricting the transition functions of the main system, so that it has the ability to call and pass parameters to one or more such sub-systems. Suppose we have sub-system 'P, and main-system P, then they can be defined by the following 9-tuples:

*f is a special function we call it sub-SFSA function to distinguish it from other primitive functions in the set F. Also, we call the sub-system **˙S**, sub-SFSA, to distinguish it from the main SFSA. Formally, a system **˙S** is a sub-system of a system **S**, iff: **•x ⊆ x, ˙T ⊆ T, ˙I ⊆ I, ˙O ⊆ O, ˙γ** must be the restriction of **γ** to **˙O**, and **˙F ⊆ N**, where **N** is the set of restrictions of **F to ˙T. If (˙f, ˙X, ˙t)** is an element of **˙F × ˙X × ˙T**, then there exists **f Є F**, such that the restriction of f to **˙T** is **˙f**, and **˙z (˙f, ˙X, ˙t) is z (f, X, t)**.

The idea of recursive function could be simply applied with the proposed method using mathematical induction. The principle of mathematical induction can be used to construct system as well as proofs. Consider the following definition of the recursion function $\mathbf{f}_r$, which is highly reminiscent of proofs by mathematical induction:

1. Input-Output Specification (IOS):

An **IOS** is a modification for input-output specification used with system design theory (see Ref. [5]). **IOS** is establishing the input-output boundaries of the system. It describes the inputs that the system is designed to handle and the outputs that the system is designed to produce. An **IOS** is not a system, but it determines the set of all systems that satisfy the **IOS**. It is a 6-tuples: **IOS = (T, I, O, Ti, To, η)**. Where T, is the time scale of **IOS**, I is the set of inputs, O is a set of outputs, Ti is a set of input trajectories defined over T, with values in I, T,, is a set of output trajectories defined over T, with values in O, and η is a function defined over T_i whose values are subset of T_o; that is, η matches with each given input trajectories T_i the set of all output trajectories that might, or could be, or eligible to be produced by some systems as output, experiencing the given input trajectory T_i. A system P satisfies IOS if there is a state X of P, and some subset U not empty of the time scale T of P, such that for every input trajectory g in T_i, there is an output trajectory h in T_o matched with g by η such that the output trajectory generated by S, started in the state X .

III. Ant Colony Algorithm for System Induction

A combinatorial optimization problem is a problem defined over a set **C** = c_1, ..., c_n of basic *components*. A subset **S** of components represents a *solution* of the problem; **F** $\subseteq 2^C$ is the subset of *feasible solutions*, thus a solution **S** is feasible if and only if **S** $\in$ **F**. A *cost function z* is defined over the solution domain, z : 2^C **à** **R**, the objective being to find a minimum cost feasible solution S*, i.e., to find S*: S* $\in$ **F** and z(S*) $\leq$ z(**S**), $\forall$**S**$\in$**F** [8]. Given this, the functioning of an ACO algorithm can be summarized as follows (see also [9]). A set of computational concurrent and asynchronous agents (a colony of ants) moves through states of the problem corresponding to partial solutions of the problem to solve. They move by applying a stochastic local decision policy based on two parameters, called *trails* and *attractiveness*. By moving, each ant incrementally constructs a solution to the problem. The ACO system contains two rules:

1. Local pheromone update rule, which applied whilst constructing solutions.
2. Global pheromone updating rule, which applied after all ants construct a solution.

Furthermore, an ACO algorithm includes two more mechanisms: trail evaporation

and, optionally, daemon actions. Trail evaporation decreases all trail values over time, in order to avoid unlimited accumulation of trails over some component. Daemon actions can be used to implement centralized actions which cannot be performed by single ants, such as the invocation of a local optimization procedure, or the update of global information to be used to decide whether to bias the search process from a non-local perspective [6]

At each step, each ant computes a set of feasible expansions to its current state, and moves to one of these in probability. The probability distribution is specified as follows. For ant k, the probability of moving from state t to state n depends on the combination of two values [7][8]:

- the attractiveness of the move, as computed by some heuristic indicating the priori desirability of that move;
- the trail level of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an a posteriori indication of the desirability of that move.

Following this general ant colony optimization procedure, the problem of system induction is solved. In the proposed algorithm a colony of ants moves through system states **X** **=$\{X_{initial}, \ldots\ldots, X_{final}\}$**.They move by applying the transition function Z, as defined in section II. This transition is policy based on two parameters, called trails and input-output specifications of the problem, i.e. data trajectory sets. By moving, each ant incrementally constructs a solution to the problem, in other words construct the transformation:

(S) $\mathbf{X}_{\mathbf{initial}} \longrightarrow \mathbf{X}_{\mathbf{final}}$

In the initial iteration of **ACO** algorithm, all ant begin from $\mathbf{X}_{\mathbf{initial}}$, and randomly move to each possible system states, as shown in figure:1. The number of system states in **ACO** depends on the number of system variables: $\mathbf{2^x}$. In the rest iterations, each ant use, it's memory to move from it's current stat to next state may be any of $\mathbf{2^x}$ states including itself, i.e. loop state. System states change by applying **z** $\in$**Z,** where: **z (f, X, t) = (•X, •t).** These mean when ant move, system state and time are changed,

outputs are produced as the type of readout function. In the next section, an example is given where mealy readout function is used, thus outputs are produced in the target states. During ants' movements, trails are always modified toward satisfying input-output specifications. When an ant complete a solution, or during the construction phase, it evaluate the solution and modify the trail value on the components used in its solution. This pheromone information will direct the search of the future ants.

Algorithm

An ACO consists of two main sections: ***initialization*** and a ***main loop***. The main loop runs for a user-defined number of iterations. These are described below:

♦***Initialization:***

a. Set initial parameters that are system:

b. Set initial pheromone trails value.

♦***While termination conditions not meet do***

a. Construct Ant Solution:

b. Apply Local Search

c. Best Tour check:

d. Update Trails

End While

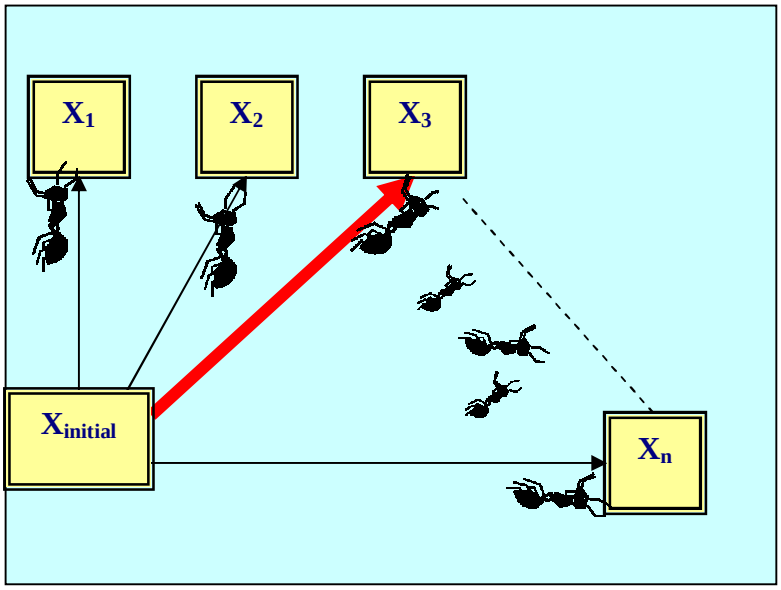

Figure 1: Initial Iteration ACO Algorithm.

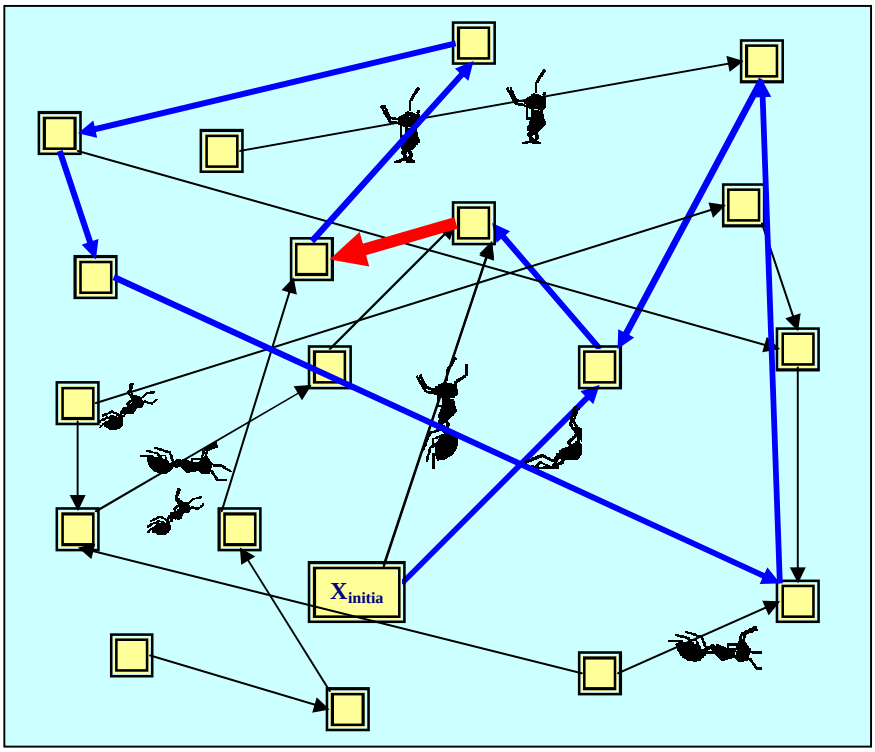

Figure 3: Final Solution of ACO Algorithm.

References:

[1] J. Holland, "Adaptation in Natural and Artificial Systems", Ann Arbor: University of Michigan Press, 1975.

[2] George Rzevski, Petr Skobelev, "Emergent Intelligence in Large Scale Multi- Systems", International Journal of Education and Information Technology Issue 2, Volume 1, 2007 64, http://www.naun.org/journals/educationinformation/eit-11.pdf.

[3] Michae J. Wooldridge, "Multi Agent Systems", john wiley sons Ltd, 2002.

[4] Nicholas R. Jennings, and Michael J. Wooldeidge, "Agent Technology", UNICOM, 2002.

[5] M. Dorigo, M. Birattari, and T. Stitzle, "Ant Colony Optimization: Arificial Ants as a Computational Intelligence Technique, IEEE computational intelligence magazine, November, 2006.

[6] A. W. Wymore, "Theory of System", Handbook of Software Engineering, CBS Publishers, pp. 119-133, 1986.

[7] Nada M. A. AL-salami, Saad Ghaleb Yaseen, "Ant Colony Optimization", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.6, pp 351-357, June, 2008

[8] M. Dorigo, T. Stützle. "The ant colony optimization metaheuristic: Algorithms, applications and advances", in F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, Kluwer Academic Publishers, To appear in 2002.

.