Journal of Computer Science 5 (5): 380-387, 2009
ISSN 1549-3636

System Evolving using Ant Colony Optimization Algorithm

Nada M.A. AL-Salami
Department of Management Information Systems, Faculty of Economic and Business,
Al Zaytoonah University of Jordan, Jordan, Amman

Abstract: Problem statement: The goal of automatic programming system is to create, in an automated way, a computer program that enables a computer to solve a problem. It is difficult to build an automatic programming system: They require carefully designed specification languages and an intimate knowledge base. Determine the relevance of mathematical system theory to the problems of automatic programming and find automatic programming methodology, where a computer program evolved to solve problem by using problem's input output specifications only. **Approach:** Problem behavior was described as a finite state automata based on its meaning, also problem's input-output specifications were described in theoretical manner, based on its input and output trajectories information, then a program was evolved to solve the problem. Different implementation languages can be used without significantly affecting existing problem specification. Evolutionary process adapts ant colony optimization algorithm to find good finite state automata that efficiently satisfies input-output specifications. **Results:** By moving from state to states, each ant incrementally constructs sub-solution in an iterative process. The algorithm converged to the optimal final solution, by accumulating most effective sub-solutions; main problem will appeared in solving problem with little input-output specifications. Fixed and dynamic input-output specifications were used to mimic chaotic behavior of real world. **Conclusion:** These results indicated that theoretical bases can enhance efficiency and performance of automatic programming system, leading to an increase in the system productivity and letting the concentrate to be done on problem specification only. Also, the collective behavior emerging from the interaction of the different ants had proved effective in solving problem; finally, in dynamic input-output specification chaos theory, especially "butterfly effect", can be used to control the sensitivity to initial configuration of trajectory information.

Key words: Automatic programming, ACO, multi-agent

INTRODUCTION

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[1]. 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[2-4]. Self-organization relies on four basic ingredients:

- Positive feedback (amplification) is constituted by simple behavioral rules that promote the creation of structures
- Negative feedback counterbalances positive feedback and helps to stabilize the collective pattern: It may take the form of saturation, exhaustion, or competition
- Self-organization relies on the amplification of fluctuations (random walks, errors, random task-switching)
- 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[5-7]:

- The creation of spatiotemporal structures in an initially homogeneous medium. Such structures include nest architectures, foraging trails, or social organization
- 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
- The existence of bifurcations when some parameters are varied. The behavior of a self-organized system changes dramatically at bifurcations

Automatic Programming is the area in which Artificial Intelligent and programming come the most closely together: it refers both to the fully computerized generation of programs from initial problem specifications and to automated improvement of program efficiency. In particular, it is desirable that the user not be required to pre specify the architecture of the ultimate solution of his problem. Problem specification might take the form of an interactive dialogue, or they might appear in graphical form, or it may be written in a specific language. Although automatic programming holds much promise for problem-solving strategies, the technology has met with less success in systems and real time programming. This research is an attempt to find general and standard Automatic Programming methodology, where a computer program is evolved to solve problem by using it's input output specifications. Problems are defined in terms of states and transition between them, which are sequences of transformation. Ant Colony Optimization incrementally generates solutions paths in the space of such transformation, adding new components to a state. Convergence to the optimal final solution is occurred by accumulating most effective sub-solutions. Fixed and dynamic input-output specifications are used to mimic chaotic behavior of real world.

MATERIALS AND METHODS

Theoretical model is proposed to describe the behavior of a program in terms of input (s), states and output (s). It equates what a program means with what it does. The word "System" in our presentation mean "Program". The meaning of system P can be specified by set of functions from states to states; hence P effects a transformation:

$$(P)\ X_{initial} \rightarrow X_{final}$$

on a state vector X, which consists of an association of the variable manipulated by the system and their values. A system P can be defined as 9- tuples, called Semantic Finite State Automata (SFSA):

$$P = (x, X, T, F, Z, I, O, \gamma, X_{initial})$$

Where:

x = The set of system variables
X = The set of system states $X = \{X_{initial},...,X_{final}\}$
T = The time scale, $T = [0, \infty)$
F = The set of primitive functions
Z = The state transition function, $Z = \{(f, X, t): (f, X, t) \in F\times X\times T, z (f, X, t) = (\bullet X, \bullet t)\}$
I = The set of inputs
O = The set of outputs
γ = The readout function
$X_{initial}$ = The initial state of the system, $X_{initial} \in X$

The sets involved in the definition of P are arbitrary, except T and F. Time scale T must be some subset of the set $[0, \infty)$ of nonnegative integer numbers, while the set of primitive function F must be a subset of the set C_L (F_L) of all computable functions in the language L and sufficient to generate the remainder functions. To execute system P, transition functions are

firing starting from, t = 0. Execution terminate when t>T. Two features characterize state transition function:

- $z(-, -, t) = (X_{initial}, 1)$, if $t = 0$
- $z(f, X, t) = z(f, z(f(t-1), X, t-1))$ if $t \neq 0$

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 $\dot{P}$ and main-system P, then they can be defined by the following 9-tuples:

$P(x, X, T, F, Z, 1, 0, X_{initial}, \gamma)$

$\dot{P}(\bullet x, \dot{X}, \dot{T}, \dot{F}, \dot{Z}, \dot{I}, \dot{O}, \dot{X}_{initial}, \dot{\gamma})$

where, $\bullet x \subseteq x$, $\dot{X}_{initial} \in X$, then there exit $*f \in F$, $z \in Z$, $\dot{f}, \in F$ and $\bullet z \in \dot{Z}$ and h is a function defined over $\dot{Z}$ with value in $\dot{X}$ is defined as follows:

$h = \dot{z}(\dot{f}, \dot{X}_{initial}, 1) = X_h, t_i$

$z(*f, X, t) = z(h, X, t) = X_h, t$

*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 $\dot{S}$, sub-SFSA, to distinguish it from the main SFSA. Formally, a system $\dot{S}$ is a sub-system of a system S, iff: $\bullet x \subseteq x$, $\dot{T} \subseteq T$, $\dot{I} \subseteq I$, $\dot{O} \subseteq O$, $\dot{\gamma}$ must be the restriction of γ to $\dot{O}$ and $\dot{F} \subseteq N$, where N is the set of restrictions of F to $\dot{T}$. If $(\dot{f}, \dot{X}, \dot{t})$ is an element of $\dot{F} \times \dot{X} \times \dot{T}$, then there exists $f \in F$, such that the restriction of f to $\dot{T}$ is $\dot{f}$ and $\dot{z}(\dot{f}, \dot{X}, \dot{t})$ 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 f_r, which is highly reminiscent of proofs by mathematical induction:

$f_r(X) = X, t = t_{max}+1$ if $X = 0$ (base of induction)

$fr(X) = X_{initial} = X, t = 0$ otherwise (induction step)

where, $T = [0, t_{max}]$.

Input-Output Specification (IOS): An IOS establishing input-output boundaries of the system. It describes inputs those the system is designed to handle and outputs those 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:

$IOS = (T, I, O, Ti, To, \eta)$

Where:

T	=	The time scale of IOS, I is the set of inputs
O	=	A set of outputs
Ti	=	A set of input trajectories defined over T, with values in
I, To	=	A set of output trajectories defined over T, with values in
O and η	=	A function defined over Ti whose values are subset of To

That is, η matches with each given input trajectories 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. 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 Ti, there is an output trajectory h in To matched with g by η such that the output trajectory generated by P, started in the state X is:

$$\gamma(Z(f(g), X, t) = \eta(h(t)) \quad \text{For every } T \epsilon U$$

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, $z: 2^C \rightarrow R$, 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]. 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:

- Local pheromone update rule, which applied whilst constructing solutions
- 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[1,10]

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[9,11,12]:

- 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

In the proposed algorithm a colony of ants moves through system states $X = \{X_{initial},...,X_{final}\}$. They move by applying the transition function Z, which is based on two parameters: d trails values and input-output specifications of the problem. By moving from state to state, each ant incrementally constructs a solution to the problem, in other words construct the transformation:

(S) $X_{initial} \rightarrow X_{final}$

In the initial iteration of ACO algorithm, all ant begin from $X_{initial}$ and use input-output specification only to move to each possible system states, as shown in Fig. 1. The number of system states in ACO depends on the number of system variables: 2^x. In the rest iterations, each ant use, it's memory as well as input-output specifications to move to next state(may be any of 2^x states including itself, i.e., loop state, as shown in Fig. 2 and 3). System states change by applying z ЄZ, where: z (f, X, t) = (•X, •t). These mean when ant move, system's state and time are changed and outputs are produced (as the type of readout function), if any. 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.

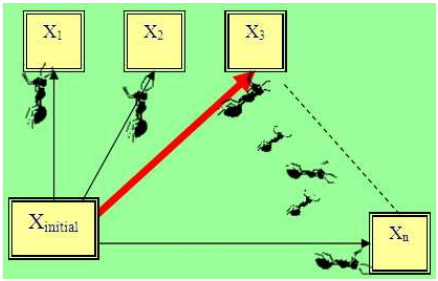

Fig. 1: Initial iteration ACO algorithm

In Fig. 2 and 3, red line denote the currently selected transition, blue lines denote the most efficient path previously stored in the memory of system, while black lines denote unselected poor transitions. Thus, at each iteration, an ant select only one transition (red) and try to append it with previously constructed path. The algorithm is defined as follow:

ACO algorithm: An ACO algorithm consists of two main parts: initialization and a main loop.

Initialize:

- Set initial parameters of the system: Variable, states, function, input, output, input trajectory, output trajectory
- Set initial pheromone trails value
- The current state of each ant is: $X_{initial}$, with empty memory

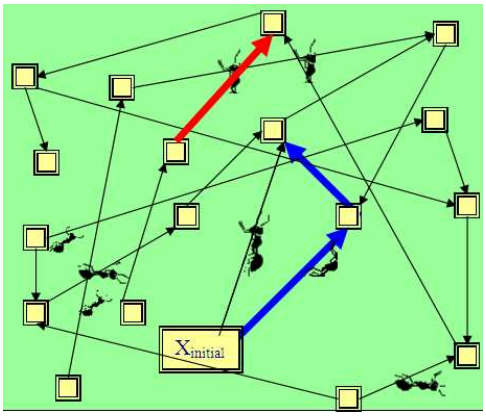

Fig. 2: After n iterations of ACO algorithm

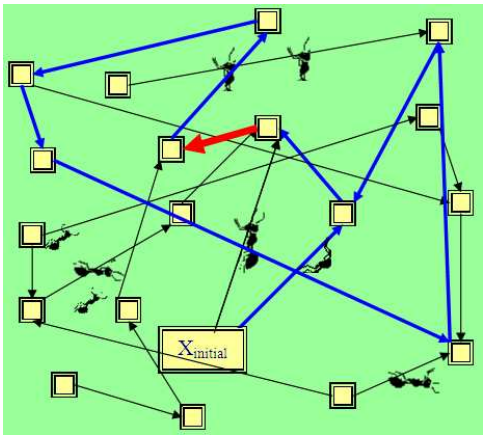

Fig. 3: Final solution of ACO algorithm

While termination conditions not meet do:

- Construct ant solution: Each ant constructs a path by successively applying the transition function zЄZ. The probability of moving from state to state depend on: Data trajectory sets (as the attractiveness of the move) and the trail level of the move.
- Apply local search.
- Best tour check: For each ant, construct data trajectory sets tour and compare to the best trajectory sets, by using the function:

$$\gamma (Z (f (g), X, t) = \eta(h(t))$$

If there is an improvement, update it.

- Update trails:
  - Evaporate a fixed proportion of the pheromone on each road
  - For each ant perform the "ant-cycle" pheromone update
  - Reinforce the best tour with a set number of "elitist ants" performing the "ant-cycle

End while:

The ACO meta-heuristic can be applied to discrete optimization problems characterized as follows:

- C = {c1; c2; : : : ; c_{NC}} is a finite set of components
  - L = {l_{cicj} | (c_i; c_j) Є ~ C}; |L|≤N^2_C is a finite set of possible connections/transitions among the elements of ~ C, where ~ C is a subset of the Cartesian product C×C
- J_{cicj} ≡ J(l_{cicj}; t) is a connection cost function associated to each l_{cicj} Є L, possibly parameterized by some time measure t
- Ω ≡ Ω(C; L; t) is a finite set of constraints assigned over the elements of C and L
- s = <c_i, c_j,….., c_k,……> is a sequence over the elements of C (or, equivalently, of L). A sequence s is also called a state of the problem. If S is the set of all possible sequences, the set ~ S of all the (sub) sequences that are feasible with respect to the constraints Ω(C; L; t), is a subset of S. The elements in ~ S define the problem's feasible states. The length of a sequence s, that is, the number of components in the sequence, is expressed by |s|
- Given two states s_1 and s_2 a neighborhood structure is defined as follows: the state s_2 is said to be a neighbor of s_1 if both s_1 and s_2 are in S and the state s_2 can be reached from s1 in one logical step (that is, if c_1 is the last component in the sequence determining the state s1, it must exists c2 2 C such that l_{c1c2} Є L and $s_2 \equiv <s_1, c_2>$). The neighborhood of a state s is denoted by N_s
- Ψ is a solution if it is an element of ~S and satisfies all the problem's requirements. A multi-dimensional solution is a solution defined in terms of multiple distinct sequences over the elements of C
- J_Ψ (L, t) is a cost associated to each solution Ψ. J_Ψ(L, t) is a function of all the costs J_{cicj} of all the connections belonging to the solution Ψ

Ants of the colony have the following properties:

- An ant searches for minimum cost feasible solutions^J Ψ = $\min_\Psi$^J Ψ (L, t)
- An ant k has a memory M^k that it can use to store information on the path it followed so far. Memory can be used to build feasible solutions, to evaluate the solution found and to retrace the path backward
- An ant k in state $s_r = <s_{r-1}, i>$ can move to any node j in its feasible neighborhood N^i_k, defined as $N^i_k = \{j| (j Є N_i)^(<s_r, j> Є ~S)\}$
- An ant k can be assigned a start state s^k_s and one or more termination conditions e^k. Usually, the start state is expressed as a unit length sequence, that is, a single component
- Ants start from the start state and move to feasible neighbor states, building the solution in an incremental way. The construction procedure stops when for at least one of the termination conditions e^k is satisfied
- An ant k located on node i can move to a node j chosen in N^k_i. The move is selected applying a probabilistic decision rule

The ants' probabilistic decision rule is a function of:

- The values stored in a node local data structure $A_i = [a_{ij}]$ called ant-routing table, obtained by a functional composition of node locally available pheromone trails and heuristic values
- The ant's private memory storing its past history
- The problem constraints
- When moving from node i to neighbor node j the ant can update the pheromone trail τ_{ij} on the arc (i, j). This is called online step-by-step pheromone update

Once built a solution, the ant can retrace the same path backward and update the pheromone trails on the

traverse arcs. This is called online delayed pheromone update.

RESULT

System behavior can be represented as transition graphs because it is easier to understand graphical notations; also ant colony algorithm is clearly understood. Such graph is a collection of four things:

- A finite set of states X, to represent the graph nods
- The sets of inputs I and outputs O
- The set N = ((f, t): (f, t) Є F×T)
- A transition table that shows for each state X and each pair (f, t) Є N. what output (if any) are produced and what states are reached next

Every state must have exactly one outgoing edge for each possible pair. Edge traveling is determined by the transition table: While traveling on edges; system outputs (if any) must be produced by applying the read out function γ. To produce efficient systems the number of it's states can be reduced and thus reduces it's time T. To perform state minimization, assigns the same time t, to all consecutive transitions iff they all have same effect on the state vector X. Only consecutive transitions are taken into consideration, so as to keep system behaviors unchanged. Figure 5 shows the minimized State Transition Graph of that showed in Fig. 4.

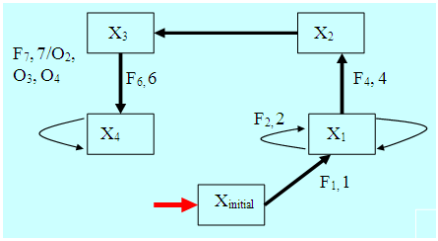

Fig. 4: State Transition Graph of the example system, where T = [1,7] and No. of states = 5

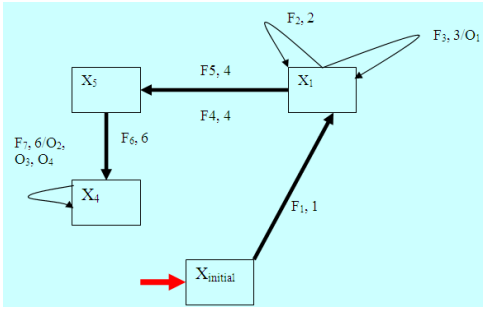

Fig. 5: Minimized STG of graph in Fig. 2, where T = [1,6] and No. of states = 4

The State Minimization Algorithm:

- Construct a subset Π of the state vector X_j> = 0 (for j = 0, $X_j = X_{initial}$). Π Consist of all entries in the state vector X_j, which occurs as arguments in the current executed function f_i, i> = 1, thus $\Pi \subseteq X_j$.
- Construct a new subset Π_{new}, it will consists of all entries in the state vector X_{j+1} which occurs as arguments in the next executed function f_{i+1}, $\Pi_{new} \subseteq X_{j+1}$. Note that each subset is either an empty set or consists only of program variable
- If $\Pi \cap \Pi_{new}$, = empty set, then delete X_j from the set X. All edges which have X_j as an end node are now redirected to X_{j+1}, to reflect the change in states
- If $Xj_{+1} = X_{final}$, go to final step
- $\Pi = \Pi_{new}$, go to second step
- Repeat above steps until non-empty set are produced from the intersection of step 3 for a complete iteration.

Example: Assume we our system is defined as following:

X = {$X_{initial}$, x1, x2, x3, x4},
T = [0, 7], F = {f_1, f_2, f_3, f_4}
I = {i}, O = {O1, O2, O3, O4}
Y = Mealy readout function
Z = {z_0(-, -, 0) = $X_{initial}$, 1
z_1 (f_1, $X_{initial}$, 1) = X_1, 2
z_2(f_2, X_1, 2) = X_1, n
z_3 (f_3, X_1, 3) = X_1, 4/O_1
z_4 (f_4, X_1, 4) = X_2, 5
z_5 (f_5, X_2, 5) = X_3, 6
z_6 (f_6, X_3, 6) = X4, 7
z_7 (f_7, X_4, 7) = X_4, 8/O_2, O_3, O_4}

DISSCUSION

It clear that the evolutionary process of our system is highly depends on input-output specifications, more precisely input and output trajectory sets and η function. Figure 6, specify clearly that system with high trajectory information converge to the solution in less time than these populations with little trajectory information. From the Fig. 6, little data trajectory information always may lead to un convergence state, (maximum iteration number allowed her is 10000). Two types of trajectory sets are used: Fixed and dynamic sets. Work with fixed system specification is usually easy since ACO algorithm is only focus on selecting transitions which highly satisfy these specification.

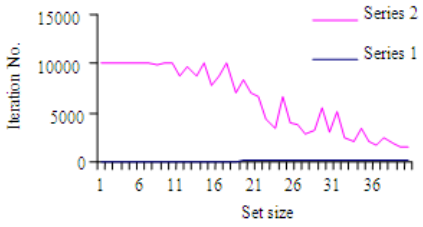

Fig. 6: Converge time versus trajectory .information of the problem

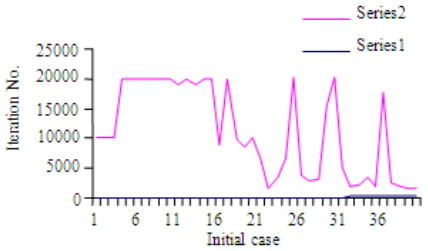

Fig. 7: Variant converge for the same problem, but different initial trajectory sets

Unfortunately, when we deal with complex systems and real live problem, strong feedback (positive as well as negative) and many interactions exist: i.e., chaotic behavior. Thus, we need to find a way to control chaos, to understand and predict what may happen long term. In these cases input and output specifications are self organized, which mean that trajectory data are collected and enhanced over time, when evolutionary process runs again and again. The algorithm begins with initial version of input and output trajectory sets and η function. Then change them over time to reflect input-output characteristic of the required system. Therefore, main problem will appear if the system has little fixed trajectory information, from experienced work we note that problems with self-modified trajectory information are difficult and take more time to converge, so the maximum iteration number was scaled up. Although trajectory data are changed over time, but by experiment, it still sensitive to initial set. This is one of the most important characteristic of a chaotic system (sensitivity to the initial conditions: “butterfly effect”). As shown in Fig. 7, for the same problem, when initial configurations of data trajectory sets are changed, there are big changes in the behaviors of ACO algorithm, even that they are small variations.

CONCLUSION

- The collective behavior emerging from the interaction of the different agents has proved effective in solving combinational optimization problems. System induction by using such interaction is more effect than induction based on formal specifications
- A colony of ants moves through system states X, by applying the transition function Z. These movements are based on two parameters: Trails and input-output specifications of the problem, i.e., data trajectory sets. By moving, each ant incrementally constructs a solution to the problem. When an ant complete solution, or during the construction phase, the ant evaluates the solution and modifies the trail value on the components used in its solution
- An artificial ant builds a solution for system induction by traversing the fully connected construction graph, represented as STG, G (C, L), where C is a set of vertices and L is a set of edges. Since Ant colony algorithm may produce redundant states in the graph, its better to minimize such graphs to enhance the behavior of the inducted system
- The proposed algorithm works with fixed and dynamic input-output specification. However, it work better with fixed specification. Thus, main problem will appeared in solving problem with little input-output specifications. More efficient learning algorithm, such as neural network, my be used to enhance the work. Further more, chaos theory, especially “butterfly effect”, can be used to control the sensitivity to initial configuration of trajectory information (in dynamic specification cases)

REFERENCES

1. Dorigo, M., M. Birattari and T. Stitzle, 2006. Ant Colony optimization: Artificial ants as a computational intelligence technique. IEEE. Comput. Intell. Mag., 1: 28-39. http://www.citeulike.org/user/rizzoli/article/1145653
2. Dorigo, M., G. Di Caro and L.M. Gambardella, 1999. Ant algorithm for discrete optimization. Artifi. Life, 5: 137-172. http://portal.acm.org/citation.cfm?id=338955
3. Holland, J.H., 1992. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. 2nd Edn., The MIT Press, USA., ISBN: 10: 0262581116, pp: 228.
4. George Rzevski and Petr Skobelev, 2007. Emergent intelligence in large scale multi- systems. Int. J. Educ. Inform. Technol., 1: 64-64. http://www.naun.org/journals/educationinformation/eit-11.pdf

5. Wooldridge, M.J., 2002. Multi Agent Systems. John Wiley Sons Ltd., USA., pp: 225-233.
6. Jennings, N.R. and Wooldeidge, M.J., 2002. Agent Technology. UNICOM, pp: 139-203.
7. Odell, J., H.V.D. Parunak and B. Bauer, 2001. Representing agent interaction protocols in ML. Proceeding of the 1st International Workshop Agent-Oriented Software Engineering, June 10-10, Springer Berlin, Heidelberg, pp: 201-218.
8. Nada, AL-salami, M.A. and S.G. Yaseen, 2008. Ant colony optimization. Int. J. Comput. Sci. Network Secur., 8: 351-357. http://search.ijcsns.org/02_search/02_search_03.php?number=200806049
9. Dorigo, M. and G. Di Caro, 1999. The Ant Colony Optimization Meta-Heuristic. In: New Ideas in Optimization, D. Corne *et al*. (Eds.). McGraw Hill, London, UK., pp: 11-32.
10. Dorigo, M., V. Maniezzo and A. Colorni, 1996. The ant system: Optimization by a colony of cooperating agents. IEEE. Trans. Syst. Man Cybernet., 26: 29-41. DOI: 10.1109/3477.484436
11. Stephen, G. and M. Dras, 2005. Understanding the pheromone system within ant colony optimization. Lecture Notes Comput. Sci., 3809: 786-789. DOI: 10.1007/11589990_81
12. Dorigo, M. and T. Stützle, 2002. The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances. In: Handbook of Metaheuristics, Glover, F. and G. Kochenberger (Eds.). Kluwer Academic Publishers, ISBN: 978-0-306-48056-0, pp: 250-285.