# ARTIFICIAL LIFE, ADAPTIVE BEHAVIOR, AGENTS AND ANT COLONY OPTIMIZATION

## Poster Papers

**Karthik Balakrishnan and Vasant Honavar, chairs**

# Ant Algorithm for Construction of Evlutionary Tree

**Shin Ando, Hitoshi Iba**
School of Engineering, University of Tokyo, Tokyo, Japan

## 1 Introduction

This paper proposes an implemetation of ant algorithms for constructing evolutionary tree.

An evolutionary tree analyzes the nucleotide and amino acid sequencesis to infer phylogenetic relationships and evolutionary hypotheses. It is an unrooted binary tree. The sequence of the leaves are not given on the internal nodes. An evolutionary tree with $n$ leaves has $n-2$ internal nodes and $m = 2n-3$ edges. The number of possible trees with n leaves is $\prod_{i-3}^{n}(2i-5)$. Most version of evolutionary trees problems are NP complete and exploration of such tree structure-space is both computationally demanding and time-consuming.

## 2 Suffix Representation

An evolutionary tree is represented in following suffix representation:
$Tx0\ Tx1\ Tx2 + Tx3 + +Tx4+$.
An ant must visit N leaves and internal vertices N-1 times in between. The vertices are not labeled, indicated as +. The constraints on suffix representation uses the idea of Stack Count .
We assign leaves with the Stack Value of +1 and internal vertices with -1. The Stack Count of a tree is the sum of the Stack Value of all its vertices. The Stack Count of any complete tree and subtree is 1. While adding up the vertices of a valid representation in sequential order, Stack Count never subceed 1. The ants are prohibited from constructing incomplete tree by constraints of Stack Count. The standard probability for choosing next city, pheromone update is used . A specific rule is applied for the probability that the ant will bind the branches at the vertex to form a larger branch.

| D | FITCH | Neighbor | Ant |
|---|---|---|---|
| -3 | 85% | 78% | 82% |
| 3-6 | 60% | 33% | 78% |
| 6- | 35% | 12% | 64% |

Table 1: 8-leaf simulation comparison

## 3 Scoring a Tree

The circular tour length, $C(S)$ , along the tree edges is calculated from distance matrix $[\delta_{ij}]$ without an explicit knowledge of the correct evolutionary tree. $C(S)$ is used for evaluating tree structure that the ants have found. The details of conventional methods for determining tree structure is found in .

## 4 Simulated Experiment

Based on a specific evolutionary tree, random sequences of nucleotide are generated by Seq-Gen . The parameters are, maximum distance in the matrix and sequence length. The 8-leaf and 16-leaf trees were simulated.The inputs were fed to FITCH and Neighbor algorithm of the PHYLIP programs for comparison.

Results of the simulation are summarized in Table1. In the simulated experiment, the algorithms showed comparable results against existing software.

### 4.1 Multiple Alignment Examples

The method was tested on 15 species of Cytochrome P450 CYP050A (http://cpd.ibmh.msk.su/). The comparison is made to FITCH and Neighbor programs. The Ant algorithms gave the best score and created a very feasible structure.

# Behavioural Selection Pressure Generates Hierarchical Genetic Regulatory Networks

**Josh C. Bongard** **Rolf Pfeifer**
Artificial Intelligence Laboratory
University of Zürich
CH-8057 Zürich, Switzerland
[bongard|pfeifer]@ifi.unizh.ch

## Introduction

The field of 'evo-devo'—evolutionary developmental biology—is making rapid inroads to biological questions that encompass phylogenetic evolution and ontogenetic development, specifically in regards to genetic regulatory networks (GRNs). However, there is relatively little understanding so far of how selection pressure shapes GRNs (Carroll 2000). We have shown that by enhancing evolutionary algorithms with genetic regulatory networks, it is possible to not only evolve simulated agents that can perform behavioural tasks, but it is also possible to analyze both evolved GRNs, and the evolutionary history of them in the evolving population (Bongard 2002). Here we show that successful evolutionary runs produce heirarchical GRNs: there is a dominant unidirectional flow in gene regulation, and relatively few cyclical gene regulation pathways. Artificial Ontogeny extends the genetic algorithm to include ontogenetic development. In the results presented below, agents are tested for how fast they can travel over an infinite horizontal plane during a pre-specified time interval. The fitness determination is a two-stage process: the agent is first grown from a GRN (the growth phase), and then evaluated in its virtual environment (the evaluation phase). See (Bongard 2002) for methodological details.

## Results

Sixty independent evolutionary runs of 300 generations each were conducted, using a variable length, floating-point genetic algorithm with a population size of 300. The best fitness curve of the most successful run is plotted in Fig. 2. The GRN of the fittest agent in each generation was transformed into a directed graph, and Warshall's algorithm was used to compute how many of the genes in each GRN were part of a cyclical genetic pathway, and scaled to the range $[0, 1]$. Finally, for each of these GRNs, 10 random graphs were generated with the same number of nodes and directed edges. The proportion of nodes lying along a cyclical path was computed using the same method as for the GRNs, and divided by the total number of nodes, to determine the random graph's cyclicality. The cyclicalities were averaged for each set of 10 random graphs, and are also plotted in Fig. 2. It can be seen that after generation 15, when the agents in the population begin to move, and thus are selected based on their behaviour, the evolved GRNs begin to exhibit much lower cyclicality than the random graphs of the same size, which indicates this is an evolved response to selection pressure.

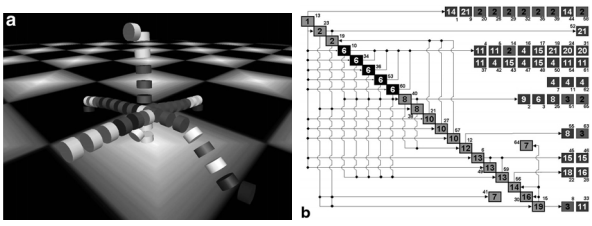


Figure 1: **a:** Most successful, evolved agent. **b:** Its GRN viewed as a graph.

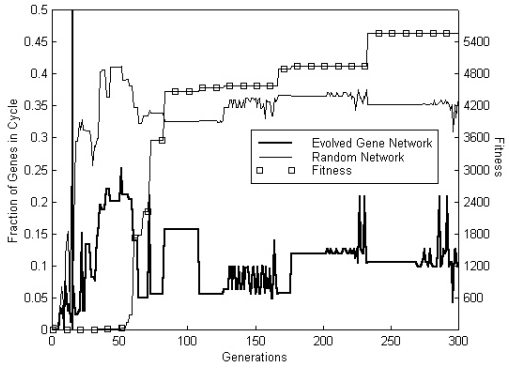


Figure 2: Evolutionary change of gene networks.

### References

J. Bongard (2002). Evolving modular genetic regulatory networks. To appear in *IEEE Congress on Evolutionary Computation, CEC 2002*, Honolulu, USA.

S. B. Carroll (2000). Endless forms: the evolution of gene regulation and morphological diversity. In *Cell* **101**:577–580.

# Solving Approximation Problems by Ant Colony Programming

**Mariusz Boryczka**
University of Silesia, Sosnowiec, Poland
e-mail: boryczka@us.edu.pl

**Zbigniew J. Czech**
Silesia University of Technology, Gliwice, and
University of Silesia, Sosnowiec, Poland
e-mail: zjc@polsl.gliwice.pl

## Abstract

A method of automatic programming, called genetic programming, assumes that the desired program is found by using a genetic algorithm. We propose an idea of ant colony programming in which instead of a genetic algorithm an ant colony algorithm is applied to search for the program. The test results demonstrate that the proposed idea can be used with success to solve the approximation problems.

## 1 INTRODUCTION

Given a problem one usually builds an appropriate computer program to solve the problem. Automatic programming makes possible to avoid a tedious task of creating such a program. In automatic programming the program is obtained by specifying first the goals which are to be realized by the program. Then, based on this specification, the program is constructed automatically. A method of automatic programming, called genetic programming, was proposed by Koza (Genetic programming: On the programming of computers by natural selection, MIT Press, Cambridge, MA, 1992). In genetic programming a desired program is found by using a genetic algorithm.

## 2 ANT COLONY PROGRAMMING

This work introduces an idea of ant colony programming in which instead of a genetic algorithm, an ant colony algorithm is applied to search for the program. We consider approximation problems which consist in a choice of an optimum function from some class of functions. Such a function should approximate in a best way another, known function, or some values of an unknown function specified in a finite number of points from its domain. Approximation problems are encountered in analysis of numerical data, modeling physical phenomena, analysis of statistical observations etc.

While solving an approximation problem by ant colony programming we use two approaches. In the first, expression approach, we search for an approximating function in the form of an arithmetic expression represented in the prefix notation. The artificial ants (agents) build the expression as a tree consisted of terminal symbols and functions, communicating with each other through the pheromone trails. In the second, program approach, the desired approximating function is built as a computer program, i.e. a sequence of assignment instructions which evaluates the function. The process of program generation consists in expanding the program by consecutive instructions taken from some predefined set.

To date ant colony programming has not been applied to automatic programming.

## 3 CONCLUSIONS

The test results demonstrate that the ant colony programming approaches are effective, especially the program approach. There are still some issues which remain to be solved. The most important is the issue of constants which regards the choice of constants which are to be enclosed in the set of terminal symbols. These constants are crucial for the work of the ant colony algorithm. The future work on ant colony programming includes an extension of the solutions to multivariate approximation problems.

### Acknowledgments

The work presented in this paper is carried out under the State Committee for Scientific Research (KBN) grant no 7 T11C 021 21.

# Evolution of Asynchronous Cellular Automata: Finding the Good Compromise

**Mathieu S. Capcarrere**
Logic Systems Laboratory
School of Computer and Communication Sciences
Swiss Federal Institute of Technology, Lausanne
CH-1015 Lausanne, Switzerland
E.mail: mathieu.capcarrere@epfl.ch

## ABSTRACT

One of the prominent features of the Cellular Automata (CA) model is its synchronous mode of operation, meaning that all cells are updated simultaneously. But this feature is far from being realistic from a biological point of view as well as from a computational point of view. Past research has mainly concentrated on studying Asynchronous CAs in themselves, trying to determine what behaviors were an "artifact" of the global clock. In this paper, I propose to evolve Asynchronous CAs that compute successfully one of the well-studied task for regular CAs: The synchronization task. As I will show evolved solutions are both unexpected and best for certain criteria than a perfect solution.

**The model used** is fully asynchronous. Each cell has the same probability $p_f$ of not updating its state at each step.

## THE ADVANTAGES OF REDUNDANT CELLULAR AUTOMATA

The extremely weak capabilities of binary CAs to cope with even limited asynchrony called for the use of redundant CA to deal with full asynchrony. I call redundant CA, a CA which uses more states in the asynchronous mode than is necessary in the synchronous mode to solve the same task. The idea behind redundancy is that the information in a CA configuration is not only the current state and the topology, but also the *timing*.

### A Simple and Perfect Time-Stamping Method

If we are looking for a method to perfectly correct asynchrony, then all information should be maintained. That is to say, all the 3-tuples $(c, i, t)$, where $c$ is the state of cell $i$ at time $t$, of the synchronous case should be reconstructible in the asynchronous case. A time-stamp added to each cell so that the cell may know if it is ahead of one of its neighbors does the trick. The minimum value of the time-stamp, not to confuse between being ahead or being late, is 3. If each cell stores both its current and last state, the new CA is both able to know if it can update and how it should update. Thus we can design a $3 * q^2$ state CA that simulates perfectly, whatever $p_f$, a $q$-state CA.

## CO-EVOLUTION OF SYNCHRONIZING CELLULAR AUTOMATA

The evolution of binary CAs does not produce very good results on real asynchrony. In the previous section, a simple method to deal perfectly with full asynchrony was designed using $3 * q^2$ states. However if we consider a task like the synchronization task, we have a perfect example of a lossy task, i.e., a task where there is no need to maintain absolutely the full information present in the synchronous case to solve the problem in the asynchronous case. The question is then to find the good compromise between the number of states needed, i.e., between the $q$ states necessary in the synchronous case, and the $3 * q^2$ we know to be sufficient to simulate perfectly the synchronous CA in the asynchronous mode. I thus proposed here to try to evolve 4-state CAs, following the cellular programming approach developed by Sipper.

Globally the evolutionary runs are very successful, and if we consider a fitness of 0.98, for $p_f < .01$, as equivalent to a fitness of 1.0 in the synchronous case[1], the success rate is equivalent to the evolution of binary CAs in the synchronous case.

## CONCLUDING REMARKS

CA asynchrony was often studied in itself in the past literature and it was often concluded that the global behavior from a CA, the emergent behavior, was an artifact of the global clock. This conclusion was not wrong in itself but rather the wrong standpoint on a reality. Time is part of the visual information contained in a CA. Now if we tackle the asynchrony problem with this idea of restoring all the information, then as we saw, we can easily design a totally asynchronous CA that simulates exactly, with no loss of information, any synchronous CA. However this presents two main problems. First, the required number of states is quite higher the original number of states. Second, it is visually different from the original CA. The visual efficiency of the original CA is lost. Evolution may then be used to limit both these problems. As presented, the cellular programming algorithm was very successful at finding 4-state solutions that were both economic and still visually efficient. Actually, it's all a question of the possible compromise between the information loss and the necessity to maintain that information.[2].

[1]The faults introduce necessarily some cells in the wrong state.

[2]Details on this work may be found in Mathieu S. Capcarrere. *Cellular Automata and Other Cellular Systems: Design & Evolution*. Phd Thesis No 2541, Swiss Federal Institute of Technology, Lausanne, 2002.

# An Ant Colony Approach for The Steiner Tree Problem

**Sanjoy Das**
Elect. & Comp. Eng. Dept.
Kansas State University
Manhattan, KS 66506
sdas@ksu.edu
(785) 532-4642

**Shekhar V. Gosavi**
Mech. Eng. Dept
Kansas State University
Manhattan, KS 66506
shekharg@ksu.edu
(785) 532-2621

**William H. Hsu**
Comp. & Info. Sci. Dept..
Kansas State University
Manhattan, KS 66506
bhsu@cis.ksu.edu
(785) 532-6350

**Shilpa A. Vaze**
Elect. & Comp. Eng. Dept.
Kansas State University
Manhattan, KS 66506
shilpag@ksu.edu
(785) 532-4595

An instance of the *Steiner tree* problem consists of:

1. A graph $G(V,E)$, where $V$ is a set of vertices (or nodes) and $E \subset V \times V$ is a set of edges.
2. A weight associated with each edge, where the weight is a mapping,
$$w : E \to \Re^+$$
3. A set of terminal nodes, $T \subseteq V$.

The problem is to compute a minimum Steiner tree, i.e. an acyclic subset $S(V_S, E_S)$ of $G$, with $T \subset V_S$, such that the vertices included in $T$ are all mutually reachable, with the smallest possible cost function.,

$$w(S) = \sum_{e \in E_S} w(e). \quad (1)$$

We propose an ant colony approach to compute minimal Steiner trees (Dorigo, Gambardella, 1997).

One ant is placed initially at each of the given terminal vertices that are to be connected. In each iteration, an ant is moved to a new location via an edge, determined stochastically, but biased in such a manner that the ants get drawn to the paths traced out by one another. Each ant maintains its own separate list of vertices already visited to avoid revisiting it. When any ant collides with another ant, or even with the path of another, it merges into the latter. An ant $m$, currently at a vertex $i$, selects a vertex $j$ not in its tabu list $T^{(m)}$, to move to, only if $(i, j) \in E$. In order to ensure that the ants merge with one another as quickly as possible, we define a potential for each vertex $j$ in $V$, with respect to an ant $m$ as follows,

$$\psi_j^{(m)} = \min_k \{d(j,k)\}, \quad (2a)$$

where,

$$k \in \bigcup_{m' \neq m} T^{(m')}. \quad (2b)$$

Here, $m'$ is any other ant, and $d(j,k)$ is the shortest distance from the two vertices, $j$ and $k$. The potential of a vertex is, therefore, a measure of the minimum possible additional cost required to join it with any of the partially completed trees. Our algorithm tries to place ants with lower potentials, but it also considers the actual cost of moving an ant from its present location to the other vertices. We define the desirability of a vertex with respect to an ant $m$, currently in vertex $i$ as,

$$\eta_j^{(m)} = \frac{1}{w(i,j) + \gamma \psi_j^{(m)}}. \quad (3)$$

The quantity $\gamma$ is a constant. The ant's position may be updated using the following equation,

$$p_{i,j} = \frac{[\tau_{i,j}]^\alpha [\eta_{i,j}^{(m)}]^\beta}{\sum_{k \notin T^{(m)}} [\tau_{i,k}]^\alpha [\eta_{i,k}^{(m)}]^\beta}. \quad (4)$$

Here, $\tau_{ij}$ and $p_{ij}$ are the trail intensity of edge $(i, j)$ and the probability of an ant using that edge to move. Trail updating rules and parameters were borrowed from known work (Dorigo, Gambardella, 1997). The results obtained are shown in Table 1.

Table 1: Results from ten test runs

| Graph Size | | | Tree Weight $w(S)$ | | |
|---|---|---|---|---|---|
| \|V\| | \|E\| | \|T\| | Avg | Best | $w^*$ |
| 50 | 100 | 13 | 61 | 61 | 61 |
| 50 | 63 | 9 | 82 | 82 | 82 |
| 50 | 63 | 25 | 138 | 138 | 138 |
| 75 | 150 | 19 | 89 | 88 | 88 |
| 100 | 125 | 25 | 235.3 | 235 | 235 |
| 100 | 200 | 50 | 225.5 | 224 | 218 |

## REFERENCES

M. Dorigo, L. M. Gambardella, Ant Colony Systems: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation*, 1(1): 53-66, 1997.

# An Individual-Based Approach to Multi-level Selection

**T. Lenaerts** tlenaert@vub.ac.be
**A. Defaweux,** adefaweu@vub.ac.be
**P. van Remortel,** pvremort@vub.ac.be
**B. Manderick** bmanderi@vub.ac.be

**Computational Modeling Lab – Computer Science Department (DINF)**
Vrije Universiteit Brussel – Belgium
http://como.vub.ac.be/

When trying to solve a complex problem, it is a natural reflex to divide this problem in a number of subproblems and solve them separately. Afterwards, the solutions to these subproblems can be combined in order to solve the entire problem. This approach is called divide-and-conquer and is applied in many circumstances.

Evolutionary Algorithms (EAs) are in this case counter-intuitive since they try to evolve a solution for the entire problem as a whole. EAs may show improvement when they can create more complex evolutionary units through some form of cooperative combination of sub-solutions similar to divide-and-conquer. In other words, instead of trying to evolve a single solution for the problem, solutions may try to cooperate to solve the problem. This cooperation results in groups of individuals which have more functionality than an isolated individuals. Hence, complexity, in terms of the structure of the solution, becomes an emergent property of the evolutionary system.

Inspiration on how to produce these cooperating groups can be found in natural systems investigated in the biological theories on Evolutionary Transitions (Maynard Smith and Szathmáry, 1997; Michod, 1997) and multi-level selection (Sober and Wilson, 1998; Keller, 1999). These theories capture the abstract principles of how higher-level structures can evolve from cooperative interactions between lower-level entities.

Similar characteristics can be observed in these transitions; In a collection of competing elements cooperative groups can emerge through spatial structuring or localisation of these individuals and their offspring. As a result higher level units, which consist of cooperating lower-level entities can emerge and hence complexity can increase. This cooperative behaviour could not have emerged in a single population due to the maladaptiveness of this behaviour. As a result of this localisation in groups and the fact that the fitness of each constituent of the group depends on the group composition, cooperative individuals were able to survive and spread in the population.

We developed a model based on a set of necessary and sufficient conditions for the emergence of cooperation proposed by Sober and Wilson (Sober and Wilson, 1998). The model consists of four iteratively repeated steps; *dispersal* of the individuals in the population into groups, *reproduction* in these isolated groups, *merging* of these different groups in some fitness proportional manner and *shrinking* of the new population to maintain the predefined population capacity.

This model results in a system where selection takes place at two different levels, i.e. at the level of the individual and the level of the group. The obtained results depend strongly on the iteratively executed disperse step. Under the right conditions, the experiments show that individual cooperative behaviour can only emerge when the selective force at the higher level can counter the selective force at the lower level. The statistical effects of these forces can be observed through the use of the Price covariance equation.

## References

L. Keller, editor (1999). *Levels of Selection in Evolution.* Monographs in Behaviour and Ecology, Princeton University Press. Princeton, New Jersey.

R.E. Michod (1997). *Darwinian Dynamics; Evolutionary Transitions in Fitness and Individuality*, Princeton Paperbacks, New Jersey.

J.M. Smith and E. Szathmáry (1997). *The Major Transitions in Evolution*, Oxford University Press

E. Sober and D.S. Wilson (1998). *Unto Others, The Evolution and Psychology of Unselfish Behaviour.* Harvard University Press. Cambridge. MA