Chapter 12
Verification of Attributes in Linked Lists Using Ant Colony Metaphor

Soumya Banerjee
Birla Institute of Technology, International Center, Mauritius

P. K. Mahanti
University of New Brunswick, Canada

ABSTRACT
The chapter describes the validation of the attributes of linked list using modified pheromone biased model (of Ant colony) under complex application environment mainly for kernel configuration and device driver operations. The proposed approach incorporates the idea of pheromone exploration strategy with small learning parameter associated while traversing a linked list. This process of local propagation on loop and learning on traversal is not available with the conventional validation mechanism of data structure using predicate logic. It has also been observed from simulation that the proposed ant colony algorithm with different pheromone value produces better convergence on linked list.

1. INTRODUCTION
Program verification has been a major area of research and development since last couple of years (Baresel, Binkley, Harman, & Korel, 2004; Manevich, Yahav, Ramalingam, & Sagiv, 2005; Lahiri & Qadeer, 2005). Different hybrid intelligent predicate and finite state machine based tools put a significant contribution to this progress. For a number of programs, the verification condition whose validity establishes the correctness of the program is expressible in a combination of first order theories such as arithmetic, un-interrupted functions, and propositional logic. However the simple first order predicate logic fails to prove properties towards a crucial data structure e.g. linked list and its all variants. The main problem with reasoning about a data structure, such as a linked list, is that it is impossible to express an invariant about all members of a list using simple first-order logic. This requires a propagation and ripple loop in predicate logic to verify the attributes of linked list successfully.

It has been observed that kernel of operating systems and device drivers manipulate a variety of data structures, such as arrays, singly and doubly
linked lists, and hash tables frequently. Current verification tools focus on control-dominated properties and are consequently unable to handle such programs in which the control flow interacts with the data in subtle ways and thus failed to create propagation and ripple loops as well.

Over the last two decades Ant Colony Optimization (ACO) has emerged as a leading nature-inspired metaheuristic method for the solution of Combinatorial Optimization (CO) problems (Yu & Zhang, 2009). The prime components of ACO are identified as the indirect communication between ants mediated by modifications of the environment and differential path length which signifies that the time spent on a path is proportional to the actual path length. The first property is called as stigmergy.

The chapter proposes an algorithm that relies on differential path length and stigmergy, incorporating a small learning component to improve the performance of the program verification. The property of learning and local propagation on loop is not possible using first order logic predicate.

We consider the following implementation of function Test_Cycle that iterates over an Acyclic list pointed to by the variable Head and sets Data field of each list element to 0.

```c
void Test_Cycle() {
    iter = head;
    while (iter!=null){
        itr.data = 0;
        itr= itr.nxt;
    }
}
```

Here, the linked list acts as set and perform several iterations. In order to achieve a loop independent code, we need to establish a loop and propagation logic based on certain a mapping of cell for the next iteration until it encounters Null field.

In order to include the linked list traversal (to explore search path or reachability) under defined conditions, the ant colony optimization with different pheromone bias has been presented in this work.

In classical ant colony, an artificial ant builds a solution by traversing the fully connected construction graph \( G_C (V, E) \), where \( V \) is a set of vertices and \( E \) is a set of edges (Neumann, Sudholt, & Witt, 2007). This graph can be obtained from the set of solution components \( C \) in two ways: components may be represented either by vertices or by edges. Artificial ants move from vertex to vertex along the edges of the graph, incrementally building a partial solution. Additionally, ants deposit a certain amount of pheromone on the components; that is, either on the vertices or on the edges that they traverse. The amount \( \Delta \tau \) of pheromone deposited may depend on the quality of the solution found.

At each iteration, the pheromone values are updated by all the ants that have built a solution in the iteration itself. The pheromone \( \tau_{ij} \), associated with the edge joining two vertex \( i \) and \( j \), is updated as follows:

\[
\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta \tau_{ij}^k
\] (1)

Hence the next iteration is decided by modifying the existing bias of ant colony pheromone deposition mechanism. Where \( \rho \) is the evaporation rate, \( m \) is the number of ants, and \( \Delta \tau_{ij}^k \) is the quantity of pheromone laid on edge \((i, j)\) by ant \( k \).

Let \( M \) denote the total number of ant agents to be applied on the data structure of interest. At the initial time \( n = 0 \), all the agents are on the source node graph. Whenever ant \( k \) is at node \( S \) at time \( n \), it probabilistically chooses an edge \( i \) to traverse at time \( n + 1 \). As mentioned earlier each agent traverses an edge one segment at a time. At every segment agent \( k \) decides to “wait” with probability \( p \) or move ahead with probability \( 1-p \), for some small \( p \). Therefore the case code if modified like below: