Chapter 4: Ant Algorithms

Download CD Content

In this chapter, we'll look into an interesting multi-agent simulation that's useful in solving a wide variety of problems. Ant algorithms, or Ant Colony Optimization (the name coined by its inventor Marco Dorigo), uses the natural metaphor of ants and stigmergy to solve problems over a physical space [Dorigo 1996]. Nature has provided a number of different methods space [Dorigo 1996]. Nature has provided a number of different methods and ideas in the space of optimization (as illustrated by other chapters in this book such as "Simulated Annealing" and "Introduction to Genetic Algorithms"). Ant algorithms are particularly interesting in that they can be used to solve not only static problems, but also highly dynamic problems such as routing problems in changing networks.

Natural Motivation

Although ants are blind, they navigate complex environments and can find food some distance from their nest and return to their nest successfully. They do this by laying pheromones while they navigate their environment. This process, known as stigmergy, modifies their environment to permit communication between the ants and the colony as well as memory for the return trip to the nest.

4장 개미 알고리즘, 최적화

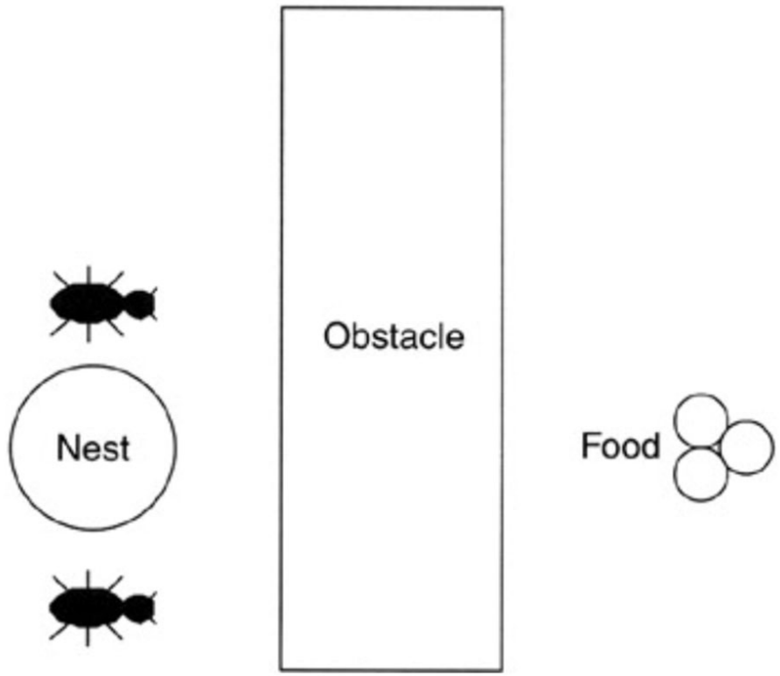


Figure 4.1: Initial configuration (T_0).

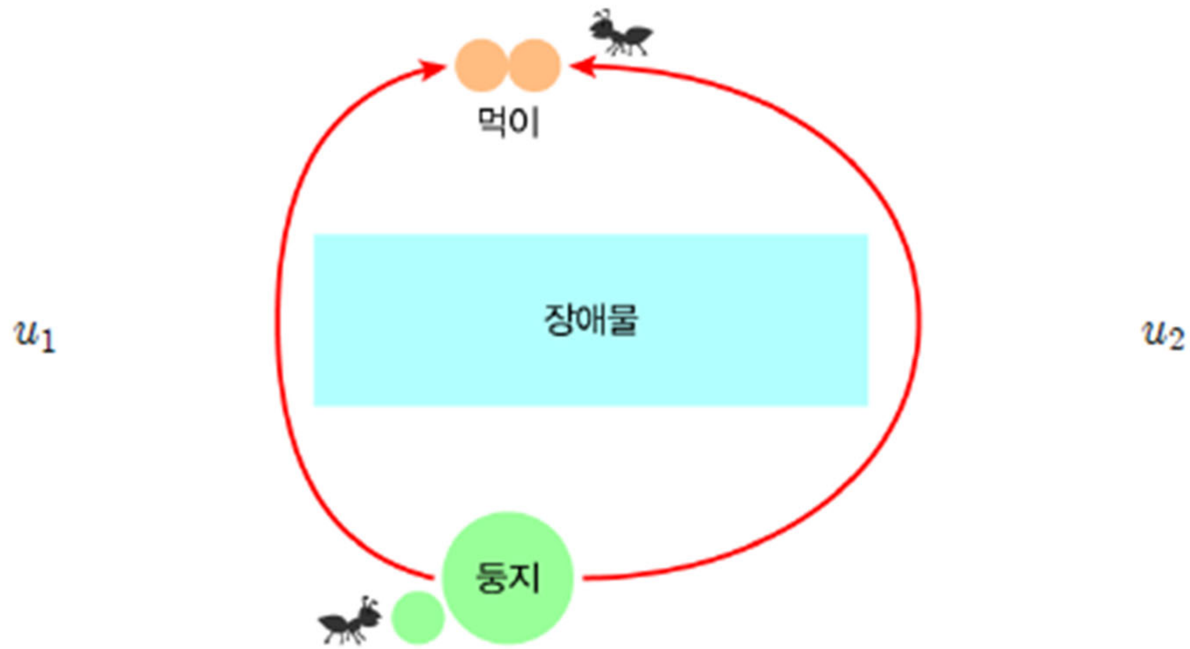
Ant Algorithm

Let's now look at the ant algorithm in detail to better understand how it functions for a specific problem.

Network

For this discussion, the environment in which our ants will exist is a fully connected bidirectional graph. Recall that a graph is a set of nodes (or vertices) connected via edges (or lines). Each edge has an associated weight, which we'll identify as the distance between the two nodes connected by the edge. The graph is bidirectional so that an ant can traverse the edge in either direction (see Figure 4.4).

4장 개미 알고리즘, 예제



$Q = 10$, $|u_1| = 10$, $|u_2| = 20$, $\rho = 0.6$, $\alpha = 3.0$, $\beta = 1.0$

4장 개미 알고리즘, 예제

$$P_{ru_1} = \frac{\tau(r,u_1)^{\alpha} \times \eta(r,u_1)^{\beta}}{\tau(r,u_1)^{\alpha} \times \eta(r,u_1)^{\beta} + \tau(r,u_2)^{\alpha} \times \eta(r,u_2)^{\beta}} = \frac{1.0^{3.0} \times 0.1^{1.0}}{1.0^{3.0} \times 0.1^{1.0} + 0.5^{3.0} \times 0.05^{1.0}}$$

$$= 0.9412$$

$$P_{ru_2} = \frac{\tau(r,u_2)^{\alpha} \times \eta(r,u_2)^{\beta}}{\tau(r,u_1)^{\alpha} \times \eta(r,u_1)^{\beta} + \tau(r,u_2)^{\alpha} \times \eta(r,u_2)^{\beta}} = \frac{0.5^{3.0} \times 0.05^{1.0}}{1.0^{3.0} \times 0.1^{1.0} + 0.5^{3.0} \times 0.05^{1.0}}$$

$$= 0.0588$$

Sample Problem

For a sample problem to apply the ant algorithm, we'll look at the Traveling Salesman Problem (or TSP). The goal of the TSP is to find the shortest tour through a graph using a Hamiltonian path (a path that visits each node only once). Mathematicians first studied the general TSP in the 1930s (such as Karl Menger in Vienna), though problems related to it were treated in the 1800s, particular by Irish mathematician Sir William Rowan Hamilton.

Source Code Discussion

The following listings will illustrate the ant algorithm to find good to optimal solutions for the Traveling Salesman Problem.

First we'll look at the data structures for both the cities on the plane and the ants that will traverse the cities. Listing 4.1 contains the types and symbolic constants used to represent the cities and ants.

Listing 4.1: Types and Symbolic Constants for City/Ant Representation.

```
#define MAX_CITIES          30

#define MAX_DISTANCE        100

#define MAX_TOUR    (MAX_CITIES * MAX_DISTANCE)
```

Sample Runs

Let's now look at a couple of sample runs of the ant algorithm for TSP.

The first run provides a solution for a 30-city TSP (see Figure 4.7). The parameters for this problem were $\alpha = 1.0$, $\beta = 5.0$, $\rho = 0.5$, and Q = 100.

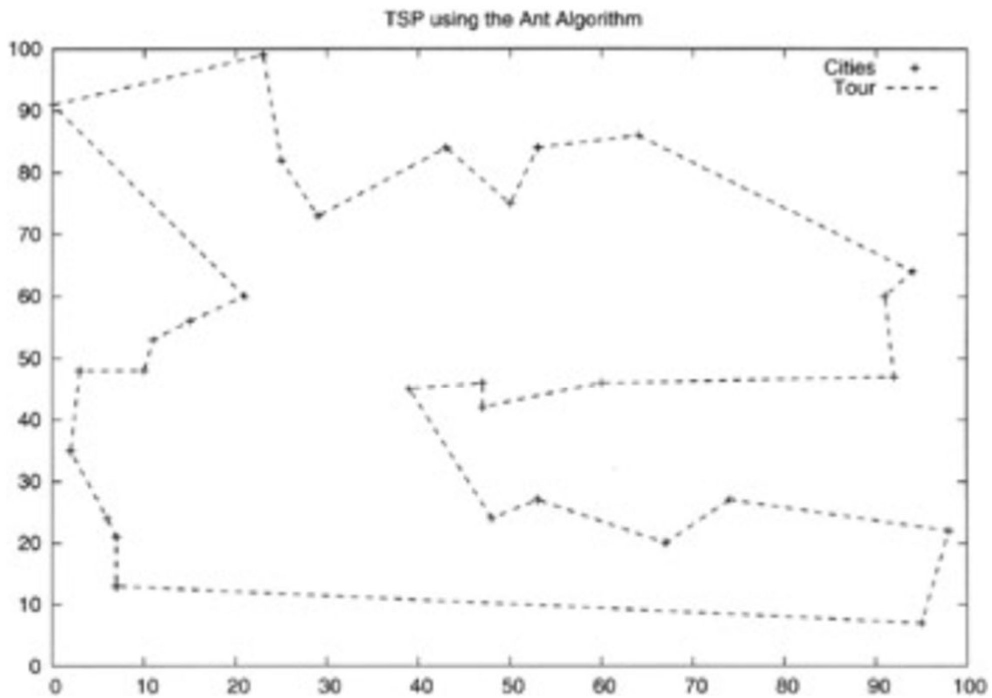


Figure 4.7: Sample solution for the 30-city TSP.