Swarm intelligence

In **swarm intelligence**, we deal with **swarms**: large groups of N individuals. (Think of flocks of birds or schools of fish.) Each individual has its own behavior and goals. And although the behavior of each individual might be simple, the whole swarm often behaves itself in a complicated yet effective way. This phenomenon is called **emergence**.

Using swarm intelligence has several advantages. All the individuals, called agents, can be produced in series. This saves costs. Also, the swarm is robust: if one agent fails, the swarm still functions. Finally, the swarm is easily scalable: you simply add more agents.

In this chapter, we'll examine three methods that use swarm intelligence. Let's start off by examining particle swarm optimization.

1 Particle swarm optimization

A particular application of swarm intelligence is **particle swarm optimization** (PSO). In PSO, we want to find the value $\mathbf{x}$ which minimizes a function $f(\mathbf{x})$. To do this, we create an n-dimensional search space, with n the size of the vector $\mathbf{x}$. In it, we put N particles. Every particle i has a (randomly initialized) position $\theta_i(k)$ and a velocity $\mathbf{v}_i(k)$ at time k. At every time step, the position of each particle is updated using

$$\theta_i(k+1) = \theta_i(k) + \mathbf{v}_i(k). \tag{1.1}$$

Also, the velocity is updated. This is done using

$$\mathbf{v}_i(k+1) = w(k)\mathbf{v_i}(k) + c_1 r_1(k)\left(\theta_{i,pbest}(k) - \theta_i(k)\right) + c_2 r_2(k)\left(\theta_{i,lbest}(k) - \theta(k)\right). \tag{1.2}$$

Let's walk through the terms in this equation. The first term is the **momentum term**. It causes particles to keep on going in the same direction as they currently are moving. The goal of this is to prevent particles from converging to a local minimum too quickly. By giving them momentum, they search the entire search space. Thus, the constant $w(k)$ is initially relatively big. (That is, almost equal to 1.) But as the algorithm proceeds, the constant is reduced.

The second term in the above equation is the **cognitive component**. The parameter $\theta_{i,pbest}(k)$ is the **personal best position**: the best position (with lowest $f(\theta)$) which particle i has found so far. This term thus causes the particle to be pulled back to its personal best. c_1 is a constant and $r_1(k)$ is a random variable, often uniformly distributed in the interval $[0, 1]$.

The third term, called the **social component**, is similar to the cognitive component. However, this time the particle compares its position to the **global best position** $\theta_{i,lbest}(k)$: the best position found by all particles together so far. The rest of the term works similarly.

When applying PSO, the particles start at random positions. Initially, they all move across the whole search space. But as time progresses, they should converge to minima of the function. When the algorithm is stopped, the actual solution is simply equal to $\theta_{i,lbest}(k)$: the best position found by all particles so far.

2 Artificial potential fields

Another way to search a space is by using an approach with **artificial potential fields**. The basic idea is that we have several particles with position $\mathbf{x}_i$. We now want to find the minimum for the function $\sigma(\mathbf{x})$. We then simply let each particle 'flow down'. This is done by applying a force on every particle of

$$\mathbf{u_i} = -\nabla_{\mathbf{x}}\sigma(\mathbf{x}_i). \tag{2.1}$$

Next to this, we also don't want particles to come closer together. It's no use if multiple particles search exactly the same space. To ensure that they don't, we use artificial potential fields. Every particle has a potential field around it, which repels other particles. This gives an additional force

$$\mathbf{u_{i,apf}} = \sum_{j=1, j\neq i}^{M} g_j(\mathbf{x_i} - \mathbf{x_j}). \tag{2.2}$$

To shorten notation, we usually write $\mathbf{y}_{ij} = \mathbf{x_i} - \mathbf{x_j}$. Examples of functions $g_j(\mathbf{y}_{ij})$ are

$$g(\mathbf{y}_{ij}) = -\mathbf{y}_{ij}\left(a - b\exp\left(-\frac{||\mathbf{y}_{ij}||^2}{c}\right)\right). \tag{2.3}$$

Next to this, obstacles might also be added. Just like particles, these obstacles do not move. They only repel other particles. Obstacles are useful if we want to restrict the search parameters to certain values.

3 Ant colony optimization

Ant colony optimization is a method to find the shortest path to a certain destination. Let's suppose that we have a graph. On a node i in this graph is an ant. This ant needs to select which arc he is going to walk on. The chance that he select an arc j at time k is given by

$$p_{ij}(k) = \frac{(\tau_{ij}(k))^{\alpha}(\eta_{ij})^{\beta}}{\sum_l (\tau_{il}(k))^{\alpha}(\eta_{il})^{\beta}}. \tag{3.1}$$

In this equation, $\tau_{ij}(k)$ denotes the **pheromone level** of the arc at time k. η_{ij} is a (constant) heuristic; for example the inverse of the length of the arc.

But we don't have one ant. We have N_a ants. Every ant chooses its arc at time step k in this way. After every ant has walked along its arc, the pheromone levels are updated. This is done using

$$\tau_{ij}(k+1) = (1-\rho)\tau_{ij}(k) + \sum_{a=1}^{N_a} \Delta\tau_{ij,a}(k). \tag{3.2}$$

Here, ρ is the **pheromone decay rate**. Also, we have

$$\Delta\tau_{ij,a}(k) = \begin{cases} F(s_a) & \text{if arc } (i,j) \text{ is used by ant } a, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Here, $F(s_a)$ is the **fitness function** of the node s at which ant a is. It can be seen as the amount of food at this node. What happens now is that arcs that lead to food sources (i.e. high fitness functions) get relatively high pheromone levels. So, they will be selected relatively often in the future as well. If, however, a path to a food source is found that is faster, then the ants will start to travel along that path more. And because it takes less time to walk along this path, more pheromone can be dropped along it. This route will thus become more preferable. In this way, the ants will find the fastest routes between food sources.

Extensions of the ant colony optimization method are also possible. For example, it can be combined with fuzzy logic. Now, an ant can be for a part in one node, and for another part in another node. And, although this can be a very interesting method, we won't go into depth on it here.