Chapter 4

Ant colony optimization

Ants are among the most widespread living organisms on the planet, and they are native to all regions except Antarctica and certain islands (such as Greenland). Ants display a multitude of fascinating behaviours, and one of their most impressive achievements is their amazing ability to co-operate in order to achieve goals far beyond the reach of an individual ant. An example, shown in Fig. 4.1, is the dynamic bridge-building exhibited by Weaver ants. Ants of this species build nests by weaving leaves together. In many situations, leaves may be too far for one ant to put them together. In such cases, Weaver ants form a bridge using their own bodies. Another example is the collective transport of heavy objects, exhibited by many species of ants. In this case, groups of ants co-operate to move an object (i.e. a large insect) that is beyond the carrying capacity of a single ant. While perhaps seemingly simple, this is indeed a very complex behaviour. First of all, the ant that discovers the object must realize that it is too heavy to transport without the help of other ants. Next, in order to obtain the help of additional ants, the first ant must carry out a process of **recruitment**. It does so by secreting a substance from a poison gland, attracting nearby ants (up to a distance of around 2 m). Then, the transport must be co-ordinated regarding, for example, the number of participating ants, their relative positions around the object, etc. Furthermore, there must be a procedure for overcoming deadlocks, in cases where the transported object becomes stuck. Studies of ants have shown that their capability for collective transport is indeed remarkable. As an example, groups of up to 100 ants, carrying objects weighing 5,000 times the weight of a single ant have been spotted [50].

The complex cooperative behaviours of ants have inspired researchers and engineers. For example, the dynamic bridge-building just mentioned has inspired research on rescue robots. In this application, the scenario is as follows: freely moving robots are released in a disaster area to search for injured or unconscious victims of the disaster. If, during the search, a robot encounters an insurmountable obstacle,

Figure 4.1: *A group of Weaver ants bringing leaves together, using their own bodies as a dynamic bridge. Photo by Václav Skoupý. Reproduced with permission.*

it may recruit other robots to form a bridge, or some other suitable structure, to pass the obstacle, and then continue the search.

Ants are also capable of remarkably efficient foraging (food gathering). Watching ants engaging in this behaviour, one may believe that the ants are equipped with very competent leaders and are capable of communicating over long distances. However, neither of these assertions would be true: ants self-organize their foraging, without any leader, and they do so using mainly short-range communication. In fact, the essential features of ant foraging can be captured in a rather simple set of equations that underlie the **ant colony optimization** (ACO) algorithms that will be studied in this chapter. However, we shall begin with a brief discussion of the biological background of ACO.

4.1 Biological background

In general, co-operation requires communication. Even though direct communication (e.g. by sound) occurs in ants, their foremost means of communication is indirect. Ants are able to secrete substances known as **pheromones**, and thus to modify the environment in a way that can be perceived by other ants. This

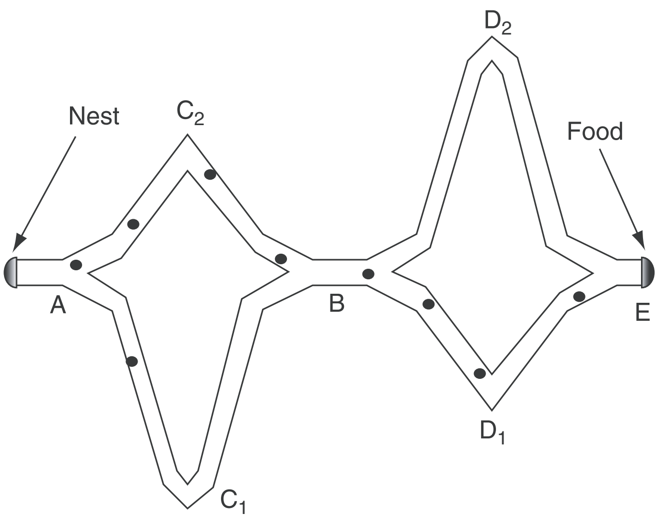

Figure 4.2: *A schematic view of the experimental setup used by Deneubourg et al. [15].*

form of indirect communication through modification of the local environment is known as **stigmergy** and it is exhibited not only by ants but by many other species (e.g. termites) as well.

During foraging, ants traverse very large distances, in some cases up to 100 m, before returning to the nest carrying food. This is an amazing feat, considering that the size of a typical ant is of the order of 1 cm. How is it achieved? The answer is that, while moving, an ant will deposit a trail of pheromone.[1] Other ants that happen to encounter the pheromone scent, are then likely to follow it, thus reinforcing the trail. The trail-making pheromones deposited by ants are volatile hydrocarbons. Hence, the trail must be continually reinforced in order to remain detectable. Therefore, when the supply of food (from a given source) drops, the trail will no longer be reinforced, and will eventually disappear altogether. The local communication mediated by pheromones is thus a crucial component in ant behaviour.[2]

The stigmergic communication in ants has been studied in controlled experiments by Deneubourg *et al.* [15]. In their experiments, which involved ants of the species *Linepithema humile* (Argentine ants), the ants' nest was connected to a food source by a bridge, shown schematically in Fig. 4.2. As can be seen from the figure,

[1] The rate of pheromone laying is not uniform, however. Typically, inbound ants (moving towards the nest) deposit more pheromone than outbound ones.

[2] In fact, pheromones have many other uses as well, both in ants and other species. A fascinating example is that of slave-making ants: about 50 of the 12,000 or so species of ants are slave-makers, i.e. they invade the nests of other ants and capture pupae containing workers of the invaded species. The exact invasion procedure varies between species, but some species are capable of using pheromones in a kind of chemical warfare: releasing their pheromones, they cause members of the other species to fight among themselves, allowing the invading species to go about its business.

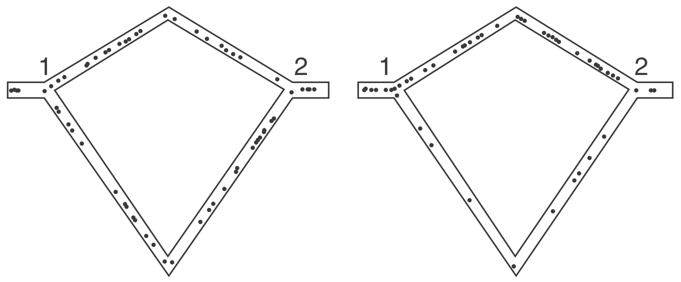

Figure 4.3: *A simplified setup, used in numerical experiments of ant navigation. The ants move from the nest at the left edge, following either of two possible paths to the food source at the right edge, before returning to the nest, again selecting either of two possible paths. The left panel shows a snapshot from the early stages of an experiment, where the probability of an ant choosing either path is roughly equal. Later on, however (right panel), the simulated ants show a clear preference for the shorter path.*

an ant leaving the nest is given a choice of two different paths, at two points (A and B) along the way towards the food source. The second half of the path might seem unnecessary, but it was included to remove any bias. Thus, at the first decision point, the short path segment is encountered if a left turn is made whereas, at the second decision point, an ant must make a right turn in order to find the short path segment. Furthermore, the shape of the branches (at the decision points A, B and E) was carefully chosen to avoid giving the ants any guidance based on the appearance of the path. Both branches involved an initial 30° turn, regardless of which branch was chosen.

Thus, the first ants released into this environment made a random choice of direction at the decision points: some ants would take a long path (A–C_1–B–D_2–E), some a short path (A–C_2–B–D_1–E) and some a path of intermediate length. However, those ants that happened to take the shortest path in both directions would, of course, return to the nest before the other ants (assuming roughly equal speed of movement). Thus, at this point, ants leaving the nest would detect a (weak) pheromone scent, in the direction towards C_2, upon reaching A, and would therefore display a preference for the shorter path. During their own trek, they would then deposit pheromone along this path, further reinforcing the trail. In the experiment, it was found that, a few minutes after the discovery of the food source, the shortest path was clearly preferred.

In addition to carrying out the experiment, Deneubourg *et al.* also introduced a simple numerical model for the dynamics of trail formation, which we will now consider briefly. Since the model deals with artificial ants, there is no need to use the elaborate path considered in the experiment with real ants. Thus, a simplified path with a single decision point in each direction, shown in Fig. 4.3, can be used. The numerical model accounts for the fact that *L. humile* deposit pheromones both on the outbound and the inbound part of the motion.

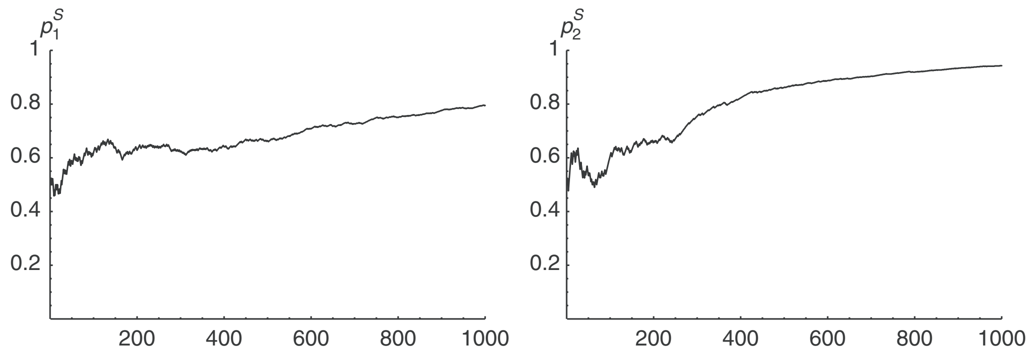

Figure 4.4: *The probabilities p_1^S (left panel) and p_2^S plotted as functions of the number of ants passing each decision point (1 and 2, respectively) for a typical run. The details will, of course, vary from run to run, since the ants select paths probabilistically.*

Thus, upon arriving at decision point 1, while moving towards the food source, an ant chooses the short path with a probability p_1^S, empirically modelled as

$$p_1^S = \frac{(C+S_1)^m}{(C+S_1)^m + (C+L_1)^m}, \tag{4.1}$$

where S_1 and L_1 denote the amount of pheromone along the short and long paths, respectively, at the decision point, and C and m are constants. Furthermore, before proceeding, the ant deposits a unit of pheromone on the chosen branch. Since the ants must choose one of the two paths, the probability p_1^L of selecting the long path equals $1-p_1^S$. Similarly, while inbound (i.e. moving towards the nest), arriving at decision point 2 the ant chooses the short path with probability

$$p_2^S = \frac{(C+S_2)^m}{(C+S_2)^m + (C+L_2)^m}, \tag{4.2}$$

and then again deposits a unit of pheromone on the chosen branch.

In the experiments with real ants, traversing a short branch (e.g. A–C_2–B in Fig. 4.2) took around 20 s, whereas an ant choosing a long branch would need $20r$ s, where r is the length ratio of the two branches. It was found in Ref. [15] that a good fit to experimental data (for various values of r in the range $[1,2]$) could be found for $C=20$ and $m=2$.

Example 4.1

The model for ant navigation described in eqns (4.1) and (4.2) was implemented in a computer program, and the experiments described in Ref. [15] were repeated. The results obtained, using the parameter values $C=20$, $m=2$ and $r=1.5$, are shown in Figs. 4.3 and 4.4. The number of simulated ants was equal to 100. The left panel of Fig. 4.3 shows a typical distribution of ants (shown as black dots) in the early stages of a simulation. As can be seen in the figure, the distribution is rather uniform, reflecting the fact that, initially, the simulated ants have no preference when selecting paths. However, in the right panel, a snapshot taken after a couple of hundred ants had traversed the paths, it can be seen clearly that the short path is preferred. In Fig. 4.4, the probabilities p_1^S and p_2^S are plotted, for a typical run, as functions of the number of ants having passed each point. ■

4.2 Ant algorithms

Given the biological background of ACO, i.e. the foraging behaviour of ants, it is, perhaps, not surprising that ACO algorithms operate by searching for paths (representing the solution to the problem at hand) in a graph. Thus, applying ACO is commonly a bit more demanding than using a GA (or PSO, for that matter; see Chapter 5) since, in order to solve a problem using ACO, it is necessary to formulate it as the problem of finding the optimal (e.g. shortest) path in a given graph, known as a **construction graph**. Such a graph, here denoted as $\mathcal{G}$, can be considered as a pair $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ are the nodes (vertices) of the graph, and $\mathcal{E}$ are the edges, i.e. the connections between the nodes. In ACO, artificial ants are released onto the graph $\mathcal{G}$, and move according to probabilistic rules that will be described in detail below. As the ants generate their paths over $\mathcal{G}$, they deposit pheromone on the edges of the graph. Thus, each edge e_{ij}, connecting node j (denoted ν_j) to node i is associated with a pheromone level τ_{ij}. Note that, in ACO, the construction graphs are, in general, directed, so that τ_{ij} may be different from τ_{ji}.

The exact nature of the construction graph $\mathcal{G}$ varies from problem to problem. The left panel of Fig. 4.5 shows a typical construction graph for one of the most common applications of ACO, namely the **travelling salesman problem** (TSP). The aim, in TSP, is to find the shortest path that visits each city once (and only once) and, in the final step, returns to the city of origin. The number of possible paths, which equals $(n-1)!/2$ where n is the number of nodes, grows very rapidly as the size of the problem is increased. Returning to the left panel of Fig. 4.5, in standard TSP, the nodes $\nu_i \in \mathcal{N}$, represent the cities, and the edges e_{ij} the (straight-line) paths between them. Thus, for this problem, the construction graph has a very straightforward interpretation: it is simply equivalent to the **physical graph** in which the actual movement takes place. In fact, because of its straightforward interpretation, the TSP is an ideal problem for describing ACO algorithms, and we will use it in the detailed description below.

However, in many problems, the construction graph is an abstract object, and functions merely as a vehicle for finding a solution to the problem at hand. The right panel of Fig. 4.5 shows a **chain construction graph**, which generates a binary string that can then, for example, be used as the input to a Boolean function. A chain construction graph that generates a binary string of length L consists of $3L+1$ nodes and $4L$ edges. Starting from the initial node ν_1, either an up-move or a down-move can be made, the decision being based on the pheromone levels on the edges e_{21} and e_{31}. If an up-move is made (to ν_2), the first bit of the string is set to 1, whereas it is set to 0 if a down-move (to ν_3) is made. The next move, to the centre node ν_4, is deterministic, and merely serves as a preparation for the next bit-generating step, which is carried out in the same way as the first move. Thus, upon reaching the end of the chain, the artificial ant will have generated a

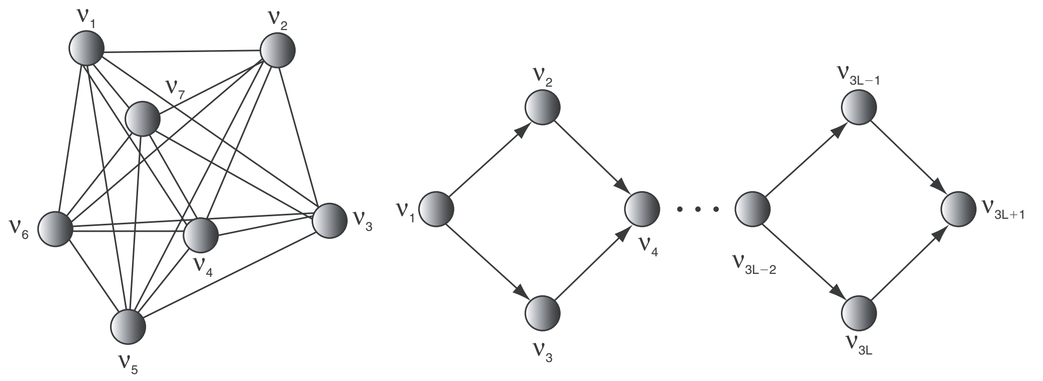

Figure 4.5: *Construction graphs for ACO. Left panel: a typical construction graph for the TSP. Right panel: a chain construction graph.*

string consisting of L bits. Chain construction graphs will be considered further in connection with our discussion of some of the mathematical properties of ACO algorithms in Appendix B. Let us now return to the TSP, and introduce ACO in detail. Since the introduction of ACO by Dorigo [17, 18], several different versions have been developed. Here, we will consider two versions, namely Ant system (AS) and Max–min ant system (MMAS), using the special case of the TSP in the description of the algorithms.

4.2.1 Ant system

AS was the first ACO algorithm introduced [18]. In order to solve TSP over n nodes, a set of N artificial ants are generated and released onto the construction graph $\mathcal{G}$. The formation of a candidate solution to the problem involves traversing a path that will take an ant to each city (node) once, and then return to the origin. Thus, each ant starts with an empty candidate solution $S = \emptyset$ and, for each move, an element representing the index of the current node is added to S. In order to ascertain that each node is visited only once, each ant maintains a memory in the form of a **tabu list** $L_T(S)$, containing the (indices of the) nodes already visited. Thus, initially, the node of origin is added to L_T. As mentioned above, each edge e_{ij} of the graph is associated with a pheromone level τ_{ij}. In any given step, an ant chooses its move probabilistically, based on a factor depending on the pheromone level as well as the distances between the current node v_j and the potential target nodes. More specifically, the probability of the ant taking a step from node j to node i is given by

$$p(e_{ij}|S) = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{v_l \notin L_T(S)} \tau_{lj}^{\alpha}\eta_{lj}^{\beta}}, \tag{4.3}$$

where, in the TSP, $\eta_{ij} = 1/d_{ij}$ and d_{ij} is the (Euclidean) distance between node j and node i. α and β are constants that determine the relative importance (from the point of view of the artificial ants) of the pheromone trails and the problem-specific information η_{ij}, referred to as the **visibility** in the case of TSP.

In each step of the movement, the current node is added to the tabu list L_T. When all nodes but one have been visited, only the as yet unvisited node is absent from the tabu list, and the move to that node therefore occurs with probability 1, as obtained from eqn (4.3). Next, the tour is completed by a return to the city of origin (which is obtained as the first element of L_T).

eqn (4.3) specifies, albeit probabilistically, the movement of the ants, given the pheromone levels τ_{ij}. However, the equation says nothing about the variation of pheromone levels. In order to complete the description of the algorithm, this crucial element must also be introduced. In AS, an **iteration** consists of the evaluation of all N ants. At the end of each iteration, all ants contribute to the pheromone levels on the edges of the construction graph. Let D_k denote the length of the tour generated by ant k. The pheromone level on edge e_{ij} is then increased by ant k according to

$$\Delta\tau_{ij}^{[k]} = \begin{cases} \frac{1}{D_k} & \text{if ant } k \text{ traversed the edge } e_{ij}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The total increase in the pheromone level on edge e_{ij} is thus given by

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \Delta\tau_{ij}^{[k]}. \tag{4.5}$$

Obviously, with eqn (4.5), pheromone levels will increase indefinitely. However, taking a cue from the properties of real pheromone trails, AS introduces pheromone evaporation, and the complete update rule thus takes the form

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}, \tag{4.6}$$

where $\rho \in]0,1]$ is the **evaporation rate**. Note that eqn (4.6) is applied to *all* edges in the construction graph. Hence, for edges e_{ij} that are not traversed by any ant, the pheromone update consists only of evaporation. AS is summarized in Algorithm 4.1.

The free parameters in AS are the constants α, β and ρ, as well as N (the number of ants). The optimal values of these parameters will, of course, vary between problems. However, experimental studies have shown that, over a large range of problems, the values $\alpha = 1$, $\beta \in [2,5]$, $\rho = 0.5$ and $N = n$ (i.e. the number of nodes in the construction graph) yield good performance. A remaining issue is the initialization of the pheromone trails. In AS, it is common to initialize the pheromone levels as $\tau_{ij} = \tau_0 = N/D^{\text{nn}}$, where D^{nn} is the **nearest-neighbour tour** obtained by starting at a random node and, at each step, moving to the nearest unvisited node.

1. Initialize pheromone levels:

   $$\tau_{ij} = \tau_0, \quad \forall i, j \in [1, n].$$

2. For each ant k, select a random starting node, and add it to the (initially empty) tabu list L_T. Next, build the tour S. In each step of the tour, select the move from node j to node i with probability $p(e_{ij}|S)$, given by:

   $$p(e_{ij}|S) = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{v_l \notin L_T(S)} \tau_{lj}^{\alpha}\eta_{lj}^{\beta}}.$$

   In the final step, return to the node of origin, i.e. the first element in L_T. Finally, compute and store the length D_k of the tour.
3. Update the pheromone levels:

   3.1. For each ant k, determine $\Delta\tau_{ij}^{[k]}$ as:

   $$\Delta\tau_{ij}^{[k]} = \begin{cases} \frac{1}{D_k} & \text{if ant } k \text{ traversed the edge } e_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$

   3.2. Sum the $\Delta\tau_{ij}^{[k]}$ to generate $\Delta\tau_{ij}$:

   $$\Delta\tau_{ij} = \sum_{k=1}^{N} \Delta\tau_{ij}^{[k]}.$$

   3.3. Modify τ_{ij}:

   $$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}.$$

4. Repeat steps 2 and 3 until a satisfactory solution has been found.

Algorithm 4.1: *Ant system (AS), applied to the case of TSP. See the main text for a complete description of the algorithm.*

Example 4.2

Algorithm 4.1 was implemented in a computer program, and was then applied to the TSP problem illustrated in Fig. 4.6. In this instance of TSP, there were 50 cities to visit. Several parameter settings were tried. For each parameter setting, 100 runs were carried out. The number of ants was equal to 50, and each run lasted 200 iterations, so that the number of evaluated paths was equal to 10,000 for each run. The results are summarized in Table 4.1. The distribution of results for any given parameter setting could be approximated rather well

Chapter 5

Particle swarm optimization

5.1 Biological background

The tendency to form swarms[1] appears in many different organisms, for instance, (some species of) birds and fish. Swarming behaviour offers several advantages, for example protection from predators: an animal near the centre of a swarm is unlikely to be captured by a predator. Furthermore, the members of a swarm may confuse predators through coordinated movements, such as rapid division into subgroups. Also, because of the large concentration of individuals in a swarm, the risk of injury to a predator, should it attack, can be much larger than when it is attacking a single animal. On the other hand, it is not entirely evident that swarming is always beneficial: once a predator has discovered a swarm, at least it knows where the prey is located. However, by aggregating, the prey effectively presents the predator with a needle-in-haystack problem and the benefits of doing so are apparently significant, since swarming behaviour is so prevalent in nature. Swarms are also formed for other reasons than protection from predators. For example, swarming plays a role in efficient reproduction: it is easier to find a mate in a large group. However, also in this case, there are both benefits and drawbacks, since competition obviously increases as well. Swarming is also a prerequisite for cooperation that, in turn, plays a crucial role in, for example, the foraging (food gathering) of several species, particularly ants, bees and termites, but also in other organisms such as birds. Here, the principle is that many eyes are more likely to find food than a single pair of eyes.

[1] Even though, normally, several different terms are used to describe different kinds of swarms, e.g. schools of fish, flocks of birds, herds of buffalo, etc., in this chapter the generic term *swarm* will be used throughout.

The search efficiency provided by swarming is what underlies **particle swarm optimization** (PSO) algorithms. Before introducing PSO in detail, however, we shall briefly consider a model for swarming in biological organisms.

5.1.1 A model of swarming

In many instances of swarming in animals, there is no apparent leader that the other members of the swarm follow. Thus, the swarm must be a result of *local* interactions only, and an interesting question follows: how can a coherent swarm result from such interactions? This issue has been addressed by Reynolds [62], who introduced a model for numerical simulation of the swarming of bird-like objects (or **boids** as they were called), which we will consider next.

In the description of PSO below, the i^{th} member of a swarm, referred to as a *particle*, will be denoted p_i. Thus, in order to keep a unified notation throughout the chapter, we shall use the symbol p_i also for the boids considered here. Let **S**, defined as,

$$\mathbf{S} = \{p_i, \quad i = 1, \ldots, N\}. \tag{5.1}$$

denote a swarm of N boids. In this model, the boids have limited visual range and can therefore only perceive nearby swarm mates. Thus, for each boid i, a **visibility sphere** $\mathbf{V}_i$ is introduced, defined as

$$\mathbf{V}_i = \{p_j : \|\mathbf{x}_j - \mathbf{x}_i\| < r, \quad j \neq i\}, \tag{5.2}$$

where r is a global constant, i.e. a property of the swarm. Note that vision is isotropic in this basic model. Of course, the model can easily be modified to include directionality of vision. In each time step of the simulation, the positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$ of the boids are updated using standard Euler integration, i.e.

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \mathbf{a}_i \Delta t, \quad i = 1, \ldots, N, \tag{5.3}$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i \Delta t, \quad i = 1, \ldots, N, \tag{5.4}$$

where $\mathbf{v}_i$ denotes the velocity of boid i, $\mathbf{a}_i$ its acceleration, and Δt is the time step. Each boid is influenced by three different movement tendencies (or steers) that together determine its acceleration, namely **cohesion, alignment** and **separation**. Cohesion represents the tendency of any given boid to stay near the centre of the swarm. Let ρ_i denote the centre of density of the boids within the visibility sphere of boid i, i.e.

$$\rho_i = \frac{1}{k_i} \sum_{p_j \in V_i} \mathbf{x}_j, \tag{5.5}$$

where k_i is the number of boids in $\mathbf{V}_i$. The steering vector representing cohesion is defined as

$$\mathbf{c}_i = \frac{1}{T^2}(\rho_i - \mathbf{x}_i), \tag{5.6}$$

where T is a time constant, introduced in order to give $\mathbf{c}_i$ the correct dimension (acceleration). If $k_i = 0$, i.e. if no boids are within the visibility sphere of boid i, then $\mathbf{c}_i$ is set to $\mathbf{0}$. Alignment, by contrast, is the tendency of boids to align their velocities with those of their nearby swarm mates. Thus, the alignment steering vector is defined as

$$\mathbf{l}_i = \frac{1}{Tk_i} \sum_{p_j \in V_i} \mathbf{v}_j. \tag{5.7}$$

As in the case of cohesion, if $k_i = 0$, then $\mathbf{l}_i$ is set to $\mathbf{0}$. Separation, finally, is needed in order to avoid collision with nearby boids, and the corresponding steering vector is obtained as

$$\mathbf{s}_i = \frac{1}{T^2} \sum_{p_j \in V_i} (\mathbf{x}_i - \mathbf{x}_j). \tag{5.8}$$

If $k_i = 0$, then $\mathbf{s}_i = \mathbf{0}$. Finally, combining the steering vectors, the acceleration of boid i is obtained as

$$\mathbf{a}_i = C_c\mathbf{c}_i + C_l\mathbf{l}_i + C_s\mathbf{s}_i, \tag{5.9}$$

where the constants C_c, C_l and C_s ($\in [0,\ 1]$) measure the relative impact of the three steering vectors. An example of steering vectors in a boids simulation is shown in Fig. 5.1.

A crucial factor in simulations based on this model is the initialization; if the initial speed of the boids is too large, the swarm will break apart. Thus, commonly, swarms are instead initialized with $\mathbf{v}_i \approx \mathbf{0}$, $i = 1, \ldots, N$. Furthermore, each boid should (at initialization) be within the visibility sphere of at least one other boid.

The simple model presented above leads, in fact, to very realistic swarm behaviour, and the algorithm has (with some modifications) been used both in computer games and in movies (e.g. Jurassic Park). An illustration of the boids model is given in Example 5.1.

Example 5.1

The boids model was implemented in a computer program, with three-dimensional visualization of the swarm. Even though the equations described above do generate swarm behaviour in principle, the equations must, in practice, be supplemented with limits on accelerations in order for the swarm to remain coherent. Here, the magnitude of each steer was limited to $a_{\max}$. Furthermore, due to random fluctuations, a swarm is likely, over time, to drift away indefinitely. Thus, in order to keep the swarm in roughly the same area, an additional acceleration component defined as

$$\mathbf{e}_i = \begin{cases} -\frac{1}{T^2}\mathbf{x}_i & \text{if } \|\mathbf{x}_i\| > R_{\max}, \\ 0 & \text{otherwise}, \end{cases} \tag{5.10}$$

was added to $\mathbf{a}_i$. $R_{\max}$ is a constant determining (roughly) the maximum allowed distance, from the origin, of any boid. Finally, the overall velocities of the boids were limited as $\|\mathbf{v}_i\| < v_{\max}$, $i = 1, \ldots, N$.

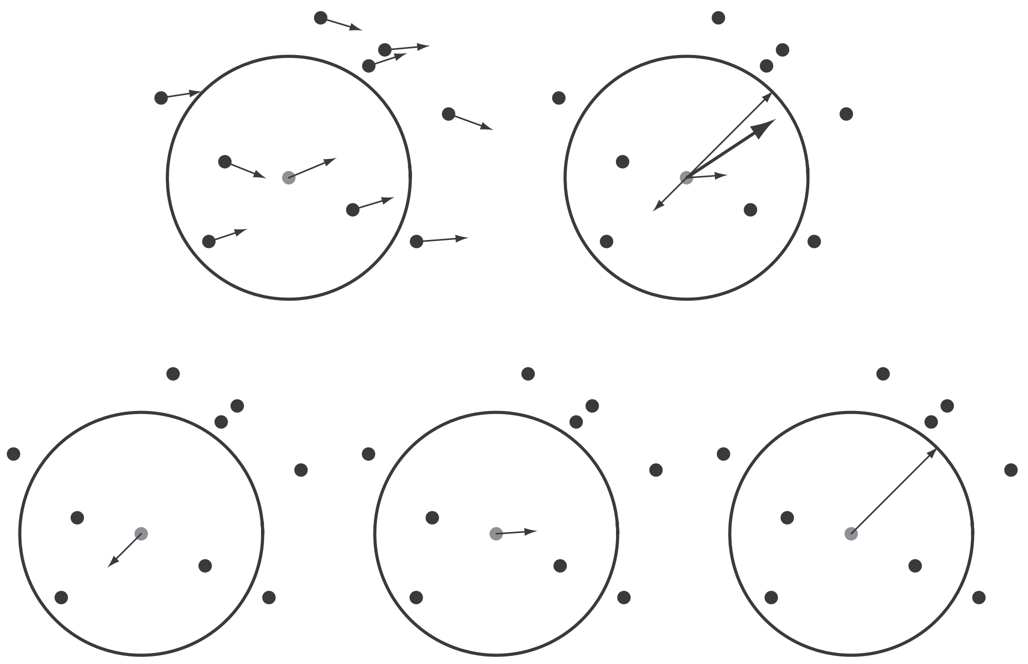

Figure 5.1: *A two-dimensional example of steering vectors: The upper left panel shows the positions and velocities of a swarm of boids, and the upper right panel shows the resulting acceleration vector (thick arrow) for the boid in the centre, as well as its components* **c**, **l** *and* **s**. *The lower panel shows, from left to right, the individual components* **c**, **l** *and* **s**, *respectively. In this example, the adjustable parameters were set to* $T = C_c = C_l = C_s = 1$. *The circles indicate the size of the visibility sphere.*

Several runs were made, with different parameter values. In one of the best runs, depicted in Fig. 5.2, the number of boids was set to 50. At initialization, the boids were randomly distributed in a sphere of radius 2.5, and given random velocities such that $\|\mathbf{v}_i\| < 0.03$, $i = 1, \ldots, N$. The radius of the visibility sphere (r) was set to 2.0, and the time constant T was set to 5.0. The constants C_c, C_l and C_s were set to 1.0, 0.1 and 0.75, respectively. Finally $a_{\max}$, $v_{\max}$ and $R_{\max}$ were set to 0.03, 0.15 and 6.0, respectively. ■

5.2 Algorithm

Particle swarm optimization (PSO) [36] is based on the properties of swarms. As is the case with ACO, PSO algorithms (of which there are several versions, as we shall see) attempt to capture those aspects of swarming that are important in optimization, namely, the search efficiency attributable to a swarm. Essentially, in PSO, each particle[2] is associated both with a position and a velocity in the search

[2] As mentioned earlier in the chapter, in PSO, the candidate solutions, corresponding to individuals in EAs, are referred to as *particles*.

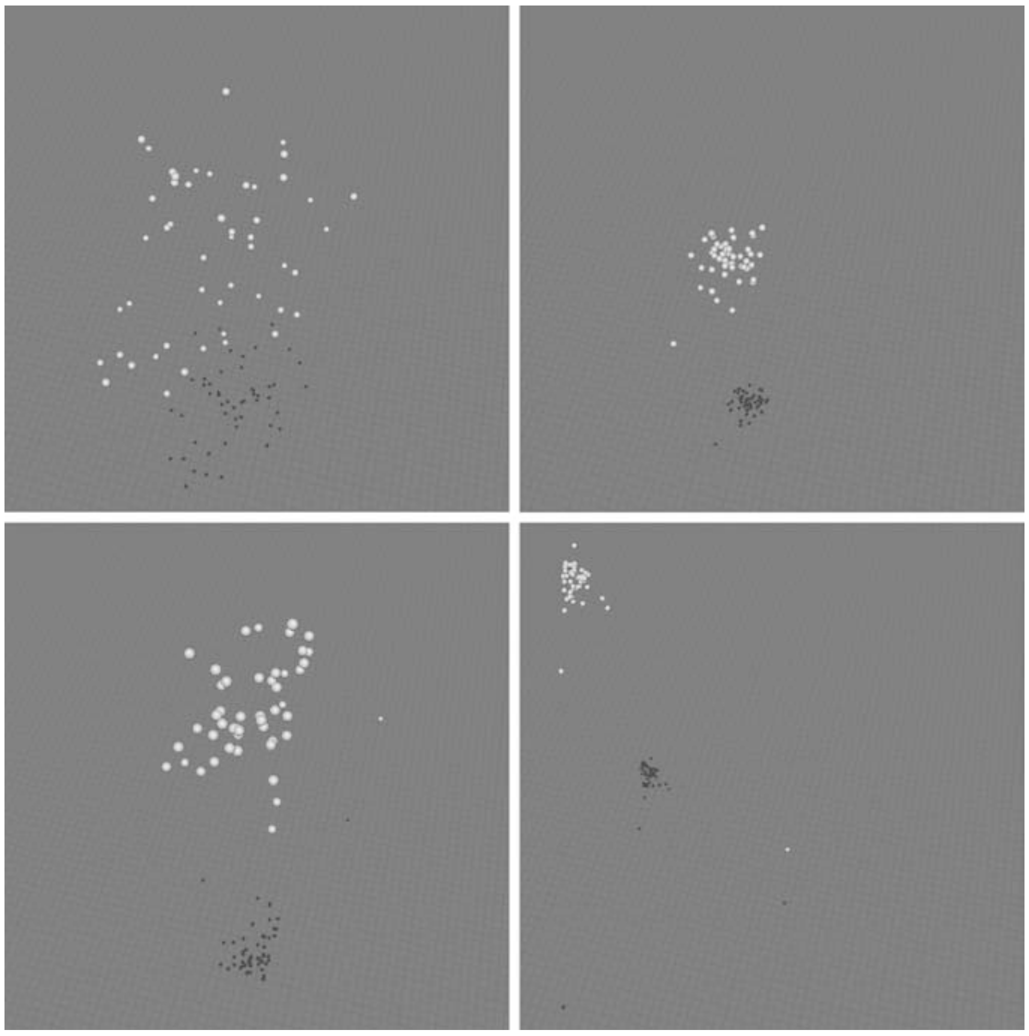
Figure 5.2: *Four snapshots from the boids simulation described in Example 5.1. The upper left panel shows the initial distribution of the boids, and the upper right panel shows the results of the early contraction that occurred since the magnitudes of the initial velocities were quite small. The lower panels depict the swarm at two later instances, showing that it remains coherent, with the exception of a few outliers.*

space, as well as a method for determining the changes in velocity depending on the performance of the particle itself and that of other particles. Thus, a clear difference compared, for example, with EAs is the introduction of a velocity in the search space. A basic PSO algorithm is described in Algorithm 5.1. The first step is initialization of the position $\mathbf{x}_i$ and the velocity $\mathbf{v}_i$ of each particle $p_i, i = 1, \ldots, N$. The appropriate number of particles will vary from problem to problem, but is typically smaller than the number of individuals used in EAs. Common values of N are 20–40. Positions are normally initialized randomly, using uniform sampling in a given range $[x_{\min}, x_{\max}]$, i.e.

$$x_{ij} = x_{\min} + r\,(x_{\max} - x_{\min}), \quad i = 1, \ldots, N, \; j = 1, \ldots, n \tag{5.11}$$

where x_{ij} denotes the j^{th} component of the position of particle p_i and r is a uniform random number in the range $[0, 1]$. N denotes the size of the swarm, corresponding to the population size in an EA, and n, as usual, is the dimensionality of the problem (the number of variables). Velocities are also initialized randomly, according to

$$v_{ij} = \frac{\alpha}{\Delta t}\left(-\frac{x_{\max} - x_{\min}}{2} + r(x_{\max} - x_{\min})\right), \quad i = 1, \ldots, N, \; j = 1, \ldots, n \tag{5.12}$$

where v_{ij} denotes the j^{th} component of the velocity of particle p_i and r again is a random number in the range $[0, 1]$. α is a constant in the range $[0, 1]$ (often set to 1), and Δt is the time step length which, for simplicity, commonly is set to 1. In the common special case where $x_{\min} = -x_{\max}$ eqn (5.12) is reduced to

$$v_{ij} = \alpha\frac{x_{\min} + r(x_{\max} - x_{\min})}{\Delta t}, \quad i = 1, \ldots, N, \; j = 1, \ldots, n \tag{5.13}$$

Once initialization has been completed, the next step is to evaluate the performance of each particle. As in an EA, the detailed nature of the evaluation depends, of course, on the problem at hand. Also, the sign of the inequalities in Algorithm 5.1 depends on whether the goal is to maximize or minimize the value of the objective function (here, minimization has been assumed).

Next, the velocities and positions of all particles should be updated. As the aim is to reach optimal values of the objective function, the procedure for determining velocities should, of course, keep track of the performance of the particles thus far. In fact, two such measures are stored and used in PSO, namely, the best position $\mathbf{x}_i^{\text{pb}}$ so far, of particle i and the best performance $\mathbf{x}^{\text{sb}}$ so far, of *any* particle in the swarm. Thus, after the evaluation of a particle p_i, the two performance tests described in step 3 in Algorithm 5.1 are carried out. The first test is straightforward and simply consists of comparing the performance of particle p_i with its previous best performance. The second test, however, can be carried out in different ways, depending on whether the *best performance of any particle in the swarm* is taken to refer to the *current* swarm or *all* particles considered thus far, and also depending on whether the comparison includes *all* particles of the swarm or only particles in a neighbourhood (a concept that will be further discussed below) of particle p_i. In Algorithm 5.1, it is assumed that the comparison in step 3.2 involves the whole swarm, and that the best-ever position is used as the benchmark. Thus, in this case, after the first evaluation of all particles, $\mathbf{x}^{\text{sb}}$ is set to the best position thus found. $\mathbf{x}^{\text{sb}}$ is then stored, and is updated only when the condition in step 3.2 of the algorithm is fulfilled.

```
1. Initialize positions and velocities of the particles $p_i$:

    1.1 $x_{ij} = x_{\min} + r(x_{\max} - x_{\min}),\ i = 1,\ldots,N,\ j = 1,\ldots,n$
    1.2 $v_{ij} = \frac{\alpha}{\Delta t}\left(-\frac{x_{\max} - x_{\min}}{2} + r(x_{\max} - x_{\min})\right),\ i = 1,\ldots,N,\ j = 1,\ldots,n$

2. Evaluate each particle in the swarm, i.e. compute $f(\mathbf{x}_i),\ i = 1,\ldots,N$.
3. Update the best position of each particle, and the global best position. Thus,
   for all particles $p_i,\ i = 1,\ldots,N$:

    3.1 if $f(\mathbf{x}_i) < f(\mathbf{x}_i^{\text{pb}})$ then $\mathbf{x}_i^{\text{pb}} \leftarrow \mathbf{x}_i$.
    3.2 if $f(\mathbf{x}_i) < f(\mathbf{x}^{\text{sb}})$ then $\mathbf{x}^{\text{sb}} \leftarrow \mathbf{x}_i$.

4. Update particle velocities and positions:

    4.1 $v_{ij} \leftarrow v_{ij} + c_1 q\left(\frac{x_{ij}^{\text{pb}} - x_{ij}}{\Delta t}\right) + c_2 r\left(\frac{x_j^{\text{sb}} - x_{ij}}{\Delta t}\right),\ i = 1,\ldots,N,\ j = 1,\ldots,n$
    4.2 Restrict velocities, such that $|v_{ij}| < v_{\max}$.
    4.3 $x_{ij} \leftarrow x_{ij} + v_{ij}\Delta t,\ i = 1,\ldots,N,\ j = 1,\ldots,n$.

5. Return to step 2, unless the termination criterion has been reached.
```

Algorithm 5.1: *Basic particle swarm optimization algorithm. N denotes the number of particles in the swarm, and n denotes the number of variables in the problem under study. It has been assumed that the goal is to minimize the objective function $f(\mathbf{x})$. See the main text for a complete description of the algorithm.*

Given the current values of $\mathbf{x}_i^{\text{pb}}$ and $\mathbf{x}^{\text{sb}}$, the velocity of particle p_i is then updated according to

$$v_{ij} \leftarrow v_{ij} + c_1 q\left(\frac{x_{ij}^{\text{pb}} - x_{ij}}{\Delta t}\right) + c_2 r\left(\frac{x_j^{\text{sb}} - x_{ij}}{\Delta t}\right), \quad j = 1,\ldots,n, \tag{5.14}$$

where q and r are uniform random numbers in $[0,1]$, and c_1 and c_2 are positive constants, typically both set to 2, so that the statistical mean of the two factors $c_1 q$ and $c_2 r$ is equal to 1. The term involving c_1 is sometimes referred to as the **cognitive component** and the term involving c_2 the **social component**. The cognitive component measures the degree of self-confidence of a particle, i.e. the degree to which it trusts its own previous performance as a guide towards obtaining better results. Similarly, the social component measures a particle's trust in the ability of the other swarm members to find better candidate solutions. Once the velocities have been updated, restriction to a given range $|v_{ij}| < v_{\max}$ is carried out, a crucial step for maintaining the coherence of the swarm, i.e. to keep it from expanding indefinitely