# 12 The Associative Search Network: Landmark Learning and Hill Climbing

*M. Bota and A. Guazzelli*

## 12.1 Introduction

In 1981, Barto and Sutton showed how a simple network could be used to model landmark learning. Their work was based on a previous paper by Barto et al. (1981) which defined the associative search problem and presented the associative search network theory. The associative network described by Barto and Sutton (1981) controls locomotion in a spatial environment composed of distinct olfactory gradients, which are produced by landmarks. In this chapter, we show how this simple network and associated task can be easily implemented in NSL. Further discussion of this example is provided in Barto and Sutton (1981).

Figure 12.1 shows a NSL window, which was used to depict the network and its environment. For didactic reasons, from now on, we assume that a simple robot, which contains an associative search network is actually the agent in the environment. The robot's only task it to move from its current position to a tree located at the center. Four additional landmarks, located at the cardinal points exist in this rather simple and imaginary world.

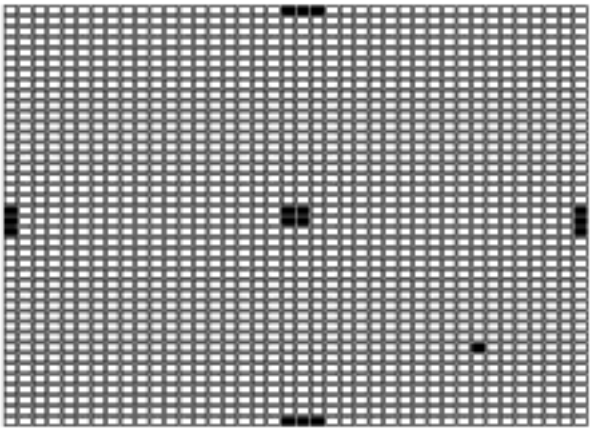


**Figure 12.1**
A window representing the robot's environment (40 x 40 small rectangles). Three consecutive filled rectangles located at the cardinal points represent the four landmarks: North, South, West, and East. The one filled rectangle on the Southeast quadrant represents the initial position of the robot. The four filled rectangles in the middle represent the tree.

Not only the tree, but also the landmarks emit each a distinctive odor, whose strengths decay with distance. However, only the odor emitted by the tree is attractive to the robot. The odors emitted by the landmarks can only be used as a cue to location in space.

## 12.2 Model Description

It can be shown that by using a hill-climbing algorithm, the robot can find its way towards the tree, even without an associative network or landmarks. If we imagine that the tree is located on the top of a hill, we can use a measure, a payoff function *z*, that tells the robot how high up in the hill it is each time it moves one step. Since the goal is to get to the tree, the higher the robot climbs, the closer it will get to its goal. In this case, the payoff function reaches its maximum at the goal, i.e. the top of the hill, and decreases smoothly with increasing distance from the tree. Note that the robot itself does not know

how far it is from the goal. Its only concern is to maximize the value of the payoff function. In formal terms, at time $t$ the robot takes one step in direction $d(t)$, moving from a position with payoff $z(t)$ to a new position with payoff $z(t+1)$. If $z(t+1) > z(t)$, then the robot will continue to move in the same direction, $d(t+1) = d(t)$, with a high probability. However, if $z(t+1) < z(t)$, then $d(t+1)$ is chosen randomly. This is like a goal-seeking strategy used by simple organisms, like *the bacterial chemotaxis* strategy used by several types of bacteria.

While this hill-climbing strategy alone can give the robot the capacity of eventually getting to the top of the hill, its trajectory, as we can imagine, will look rather clumsy and inefficient. Nevertheless, we can improve the robot's goal-seeking behavior by using Barto and Sutton's associative search network (figure 12.2). The network is composed of four input and four output units. Each input unit $i$, where $i$ = *North*, *South*, *East*, and *West*, receives an input $x_i(t)$ from their respective landmark. Moreover, each input unit is fully connected with all four output units $j$, where $j$ = *North*, *South*, *East*, and *West*. This allows each input unit to adapt four connection weights $w_{ji}(t)$ in the connection matrix. Each weight encodes a degree of confidence that, when the robot is near landmark $i$, it should proceed in direction $j$ to get closer to the tree. An extra input unit (depicted in figure 12.2 as a triangle), represents the specialized payoff pathway $z$, which has no associated weights. The payoff function can also be seen as a reinforcement signal.

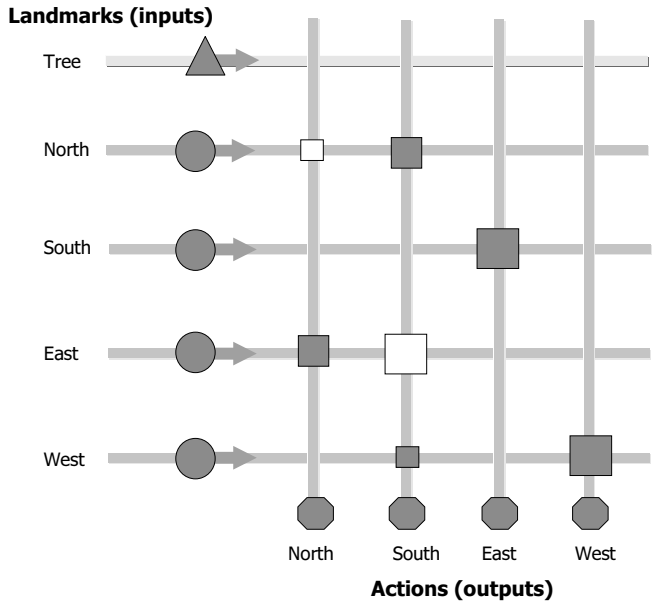



**Figure 12.2**
The associative search network. The tree and the four additional landmarks are labeled vertically on the left. Each landmark releases an odor. The five distinct odors give rise to five different input pathways (input units are depicted as filled circles and as a triangle). At the bottom of the network, four distinct actions (representing the direction to be taken at the next step) give rise to four output pathways (output units are depicted as filled hexagons). Adaptable weights are depicted as rectangles: bigger weights are represented as bigger rectangles (negative weights are depicted as hollow rectangles; positive weights as filled rectangles). See text for more details.

When the robot is at a particular location in its environment, it is able to sense its distance from each of the landmarks. The degree of confidence $s_j(t)$ for a move in direction $j$ is determined by the sum of the products of the current weights and the current signals received from the four landmarks:

$$s_j(t) = w_{0j}(t) + \Sigma_i \, w_{ji}(t) \, x_i(t) \qquad (12.1)$$

where $w_{0j}(t)$ can be seen as a bias term to be further described below. If we assume that the connection matrix contains appropriate weights, we can also assume that the chosen direction $j$ is also appropriate. If, for example, our robot is close to the *Northern* landmark, the output unit *South* will be activated and the robot's next step will be towards *South*. Moreover, if the robot is in the *Southwest* quadrant, output units *North* and *East* will be activated and the robot's next step will be towards *Northeast*.

However, since it is still too early for us to assume that the network contains suitable weights, a noise term is added to $s_j(t)$, setting the output of unit $j$ at time $t$ to be

$$y_j(t) = 1 \text{ if } s_j(t) + NOISE_j(t) > 0, \text{ else } 0 \tag{12.2}$$

where each $NOISE_j(t)$ is a normally distributed random variable with zero mean. If $s_j(t)$ is bigger than 0 when noise is added, the robot's next step will be towards direction $j$. If, on the other hand, $s_j(t)$ is smaller than 0, a random direction is chosen.

At this point, however, the biggest challenge for the robot is to learn appropriate weights. For this reason, a learning rule has to be implemented. This follows the following equation:

$$w_{ji}(t+1) = w_{ji}(t) + c[z(t) - z(t-1)]\, y_j(t-1)\, x_i(t-1) \tag{12.3}$$

where $c$ is a positive constant determining the learning rate. In the simulations depicted below, $c = 0.25$. According to this rule, a connection weight $w_{ji}$ will only change if a movement towards direction $j$ is performed ($y_j(t\text{-}1) > 0$) and if the robot is near an $i$-landmark ($x_i(t\text{-}1) > 0$). If we return to the view of $z(t)$ as height on a hill, we can see that $w_{ji}$ will increase if $z$ increases, which implies that direction $j$ moves the robot uphill. In this situation, a $j$-movement will be more likely to happen again. If, on the other hand, $w_{ji}$ decreases, $z$ decreases, which implies the robot is moving downhill. In this case, a $j$-movement will be less likely to occur.

### 12.3 Model Implementation

In NSL, this learning rule is implemented by the following code:

```
NslDouble2 W(4,4);   // weight matrix
NslDouble1 Y(4);     // output vector
NslDouble1 X(4);     // input vector
double     tmp;
double     z;
double     z1;
…
tmp = 0.25 * (z1 - z);
W=W+tmp*Y*X;
for (i = 0; i < 4; i++){
  for (j = 0; j < 4; j++) {
      if ((W[i][j] >= (0.5))
         W[i][j] = 0.5;
      if (W[i][j]<= (-0.5))
         W[i][j] = -0.5;
   }
}
```

where z1 = z(t) and z = z(t-1). As it can be seen above, in the computations we performed, the weights are bounded inside the interval [-0.5, 0.5].

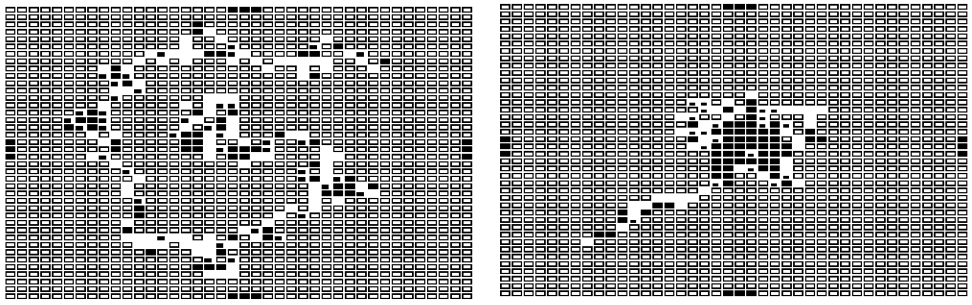
The weights $w_{0j}$ (formerly described as biases) are updated as follows:

$$w_{0j}(t+1) = f[w_{0j}(t) + c_0[z(t) - z(t\text{-}1)]\, y_j(t\text{-}1)], \tag{12.4}$$

where $f(x)$ = BOUND if $x >$ BOUND, 0 if $x < 0$, $x$ otherwise (this will bound each $w_{0j}$ to the interval [0, BOUND]), $c_0 = 0.5$, and BOUND = 0.005. Moreover, this learning rule is necessary only to permit the robot to climb the hill in the absence of landmark input information $x_i$.

## 12.4 Simulation and Results

By providing our robot with an associative search network and the appropriate values for its connection weights, we are giving the robot the chance of conducting hill climbing in weight space, instead of physical space. The simulations below try to show this newly acquired capacity. Figure 12.3a shows a NSL window containing the histogram of places occupied by the robot in its first attempt to get to the goal (training phase). Figure 12.3b shows the histogram of the robot's second attempt to get to the tree (testing phase), this time starting from a different position. If we compare (a) and (b), we can clearly see that there is a major improvement in the trajectory taken by the robot, since in 3b, it is using its long-term store acquired during the training phase. Both histograms are depicted over the environment as shown in figure 12.1.



(a) (b)

**Figure 12.3**

Two NSL windows containing the histogram of places occupied by the robot while in search of the tree during (a) the training phase and (b) the testing phase. In the test phase, the robot starts its trajectory from a different position than the one used for the training phase. Empty rectangles with a bold contour mean that the robot did not entered at that position. Empty rectangles with a light contour mean that the robot entered at that position at most one time. Each window displays a total of 400 steps for each phase.

This landmark-guided hill-climbing example illustrates how the results of explicit searches can be transferred to an associative long-term store so that in future encounters with similar (but not identical) situations the system need only access the store to find out what to do. As pointed out by Arbib (1989), the associative search network shows how all of this can be accomplished without centralized control. It is thus an improvement over a non-learning search method, and it also has the important property that the optimal responses need not be known a priori by the environment, the system, or the system's designer.

The NSL environment built to illustrate the robot's search for the tree is also composed by additional windows than the ones showed above. Figure 12.4 depicts a typical run. The main window is composed of six small windows. From left to right and top to bottom, the first window (x) shows the distance on the horizontal plane from the robot to the tree during the training phase. The second window (y) shows the distance on the vertical plane during the training phase. One can see that during the search, both distances are converging to 0. This is reflected in the third window (D), which shows the computed Euclidean distance between the robot and the tree during the training phase. The fourth window (Dtest) shows the computed Euclidean distance during the testing phase. The fifth window (Weight) shows the connection weights in the same way as shown in figure 12.2. The magnitude of the rectangles reflect the weights obtained at the end of the training phase. The next two windows (histtrain and histtest) show the histograms for the training and testing phases as in figure 12.3. In the present case,

however, the robot started its trajectory from the same position in both phases. The last two windows (pathtest and pathtrain) show the trajectories of the robot on its path from the initial position to the goal. Each window displays a total of 400 steps.

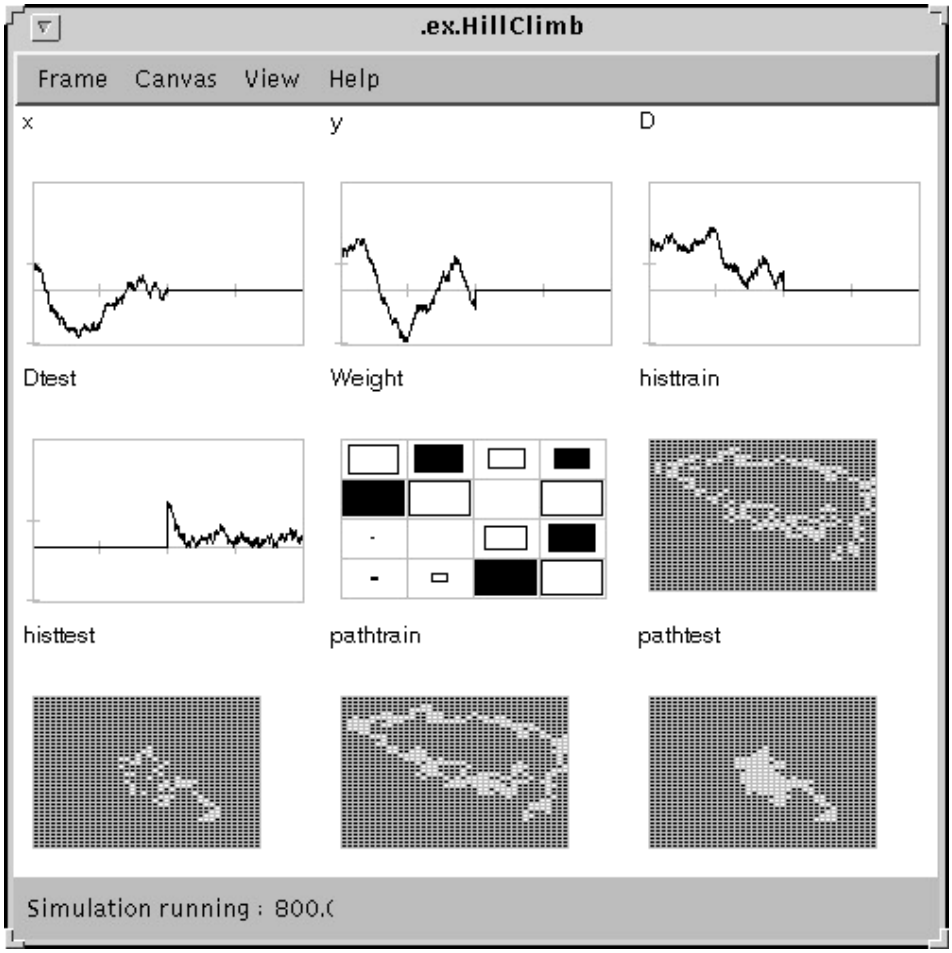



**Figure 12.4**
The NSL interface window used to simulate Barto & Sutton's (1981) landmark learning task. See text for details.

## 12.5 Summary

With this model we have shown how the hill-climbing strategy can give a robot the capability of getting to the top of a hill. However, we have also shown how to make the algorithm more efficient by improving the robot's goal-seeking behavior by using Barto and Sutton's associative search network (figure 12.2). Also, by using the NSL 3.0 simulation system we were able to easily encapsulated some of our more complex mathematical computation, and we were able to easily debug the model. We also used the NSL "Train and Run" feature to separate out the learning phase from the execution phase of the model. Finally, we were able to use NSL dynamic plot capability to plot the variables we were interested in and print the results for this book.

## Notes

1. The Associative Search Network model was implemented and tested under NSLJ.