Learning Real-Time A* (LRTA*) Search Algorithm

When only one agent is solving a path-finding problem, it is not always possible to perform local computations for all nodes. For example, autonomous robots may not have enough time for planning and should interleave planning and execution. Therefore, the agent must selectively execute the computations for certain nodes. Given this requirement, which node should the agent choose? One intuitively natural way is to choose the current node where the agent is located. It is easily to imagine that the sensing area of an autonomous robot is always limited. First, the agent updates the h value of the current node, and then moves to the best neighboring node. This procedure is repeated until the agent reaches a goal state. This method is called the *Learning Real-Time A** (LRTA*) algorithm.

More precisely, in the LRTA* algorithm, each agent repeats the following procedure (we assume that the current position of the agent is node i). As with asynchronous dynamic programming, the agent records the estimated distance $h(i)$ for each node.

1. Lookahead

Calculate $f(j) = k(i, j) + h(j)$ for each neighbor j of the current node i, where $h(j)$ is the current estimate of the shortest distance from j to goal nodes, and $k(i, j)$ is the link cost from i to j.

2. Update

Update the estimate of node i as follows:

$$h(i) \leftarrow \min_{j} f(j)$$

3. Action selection

Move to the neighbor j that has the minimum $f(j)$ value. Ties are broken randomly.

One characteristic of this algorithm is that the agent determines the next action in a constant time, and executes the action. Therefore, this algorithm is called an *on-line, real-time* search algorithm.

In the LRTA*, the initial value of *h* must be optimistic, i.e., it must never overestimate the true value. Namely, the condition $h(i) \leq h^*(i)$ must be satisfied. If the initial values satisfy this condition, $h(i)$ will not be greater than the true value $h^*(i)$ by updating.

We call a function that gives the initial values of *h* a *heuristic function*. For example, in the 8-puzzle, we can use the number of mismatched tiles, or the sum of the Manhattan distances (the sum of the horizontal and vertical distances) of the mismatched tiles, for the heuristic function (the latter is more accurate). In the maze problem, we can use the Manhattan distance to the goal as a heuristic function.

A heuristic function is called *admissible* if it never overestimates. The above examples satisfy this condition. If we cannot find any good heuristic function, we can satisfy this condition by simply setting all estimates to 0.

In LRTA*, the updating procedures are performed only for the nodes that the agent actually visits. Therefore, if the initial value of node *i* is larger than the true value, it is possible that the agent never visits node *i*; thus, $h(i)$ will not be revised.

The following characteristic is known: In a finite number of nodes with positive link costs, in which there exists a path from every node to a goal node, and starting with non-negative admissible initial estimates, LRTA* is *complete*, i.e., it will eventually reach a goal node.

Furthermore, since LRTA* never overestimates, it learns the optimal solution through repeated trials, i.e., if the initial estimates are admissible, then over repeated problem solving trials, the values learned by LRTA* will eventually converge to their actual distances along every optimal path to the goal node.

References

M. Yokoo, T. Ishida, *Search Algorithms for Agents*, in G. Weiss (ed.), Multiagent systems, The MIT Press, 1999.