# Online Search

Now we are going to state what the difference between offline search and online search is.

* Offline Search => Agents that implement an offline search algorithm for solving a search problem firstly compute the solution to the problem and then set foot in the real world to apply the solution.
* Online Search => Agents that implement an online search algorithm for solving a search problem interleave computation and action. The agent first takes an action, then it observes the environment and computes the next action.

Online search is a necessary idea for unknown environments, where the agent does not know what its actions do. In this state of ignorance, the agent faces an **exploration problem** and must use its actions as experiments in order to learn enough to make deliberation worthwhile. The canonical example of online search is a robot that is placed in a new building and must explore it to build a map that it can use for getting from A to B. Methods for escaping from labyrinths—required knowledge for aspiring heroes of antiquity—are also examples of online search algorithms. Spatial exploration is not the only form of exploration, however. Consider a newborn baby: it has many possible actions but knows the outcomes of none of them, and it has experienced only a few of the possible states that it can reach. The baby’s gradual discovery of how the world works is, in part, an online search process.

Thus, the main difference between offline search agents and online search agents is that the latter does not know the transition model (what state s’ is reached if performing action a in state s) and so it needs to explore and perform actions in order to gain an understanding of their outcomes. Therefore, it is not possible to compute the solution to the problem beforehand as in offline search settings.

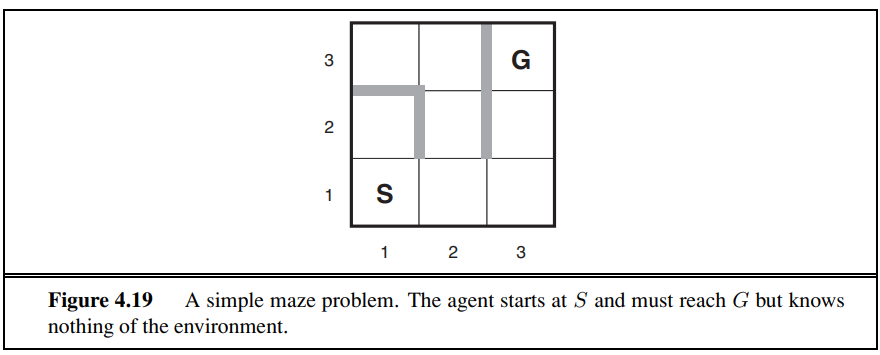
## Online search problem

In what follows, we are going to assume that the environment of online search problems is deterministic and fully observable.

In an online search problem, an agent is provided with only the following information:

* Actions(s) => list of actions allowed in state s
* Goal-test(s) => allows the agent to know whether a state s is a goal state

Note in particular that the agent cannot determine RESULT(s, a) except by actually being in s and doing a. For example, in the maze problem shown in Figure 4.19, the agent does not know that going Up from (1,1) leads to (1,2); nor, having done that, does it know that going Down will take it back to (1,1).



It is important to notice that the environment is fully observable which means that the agent that is placed in the maze of figure 4.19 knows where the goal state and what states must be in before reaching state G. However, he does not know how to reach state G because he does not know the outcome of its actions.

Typically, the agent’s objective is to reach a goal state while minimizing cost. (Another possible objective is simply to explore the entire environment.) The cost is the total path cost of the path that the agent actually travels. It is common to compare this cost with the path cost of the path the agent would follow if it knew the search space in advance—that is, the actual shortest path (or shortest complete exploration). In the language of online algorithms, this is called the competitive ratio; we would like it to be as small as possible.

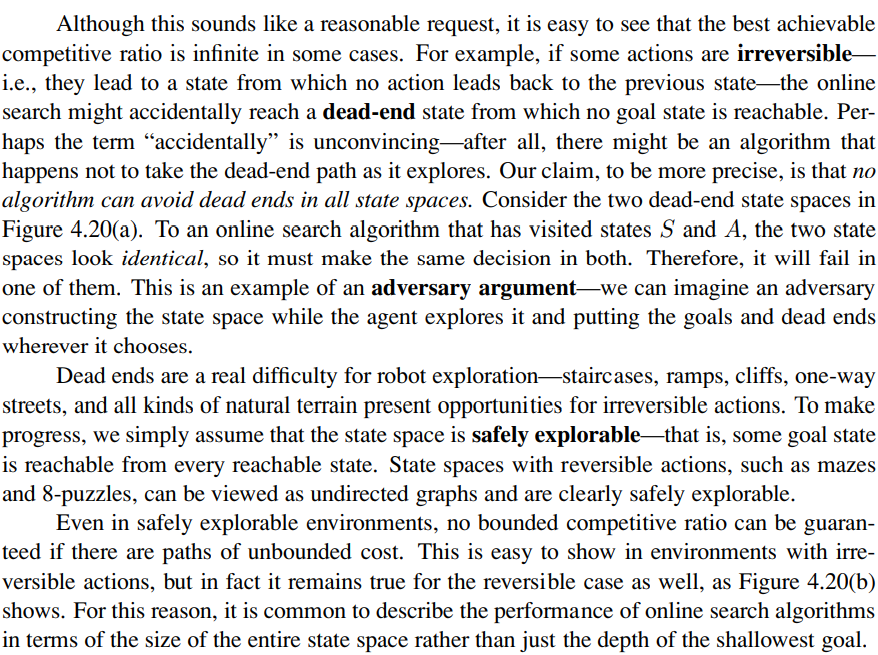
In other words, the performance of online search agents is measured through the **competitive ratio**.

The competitive ratio is equal to the following:

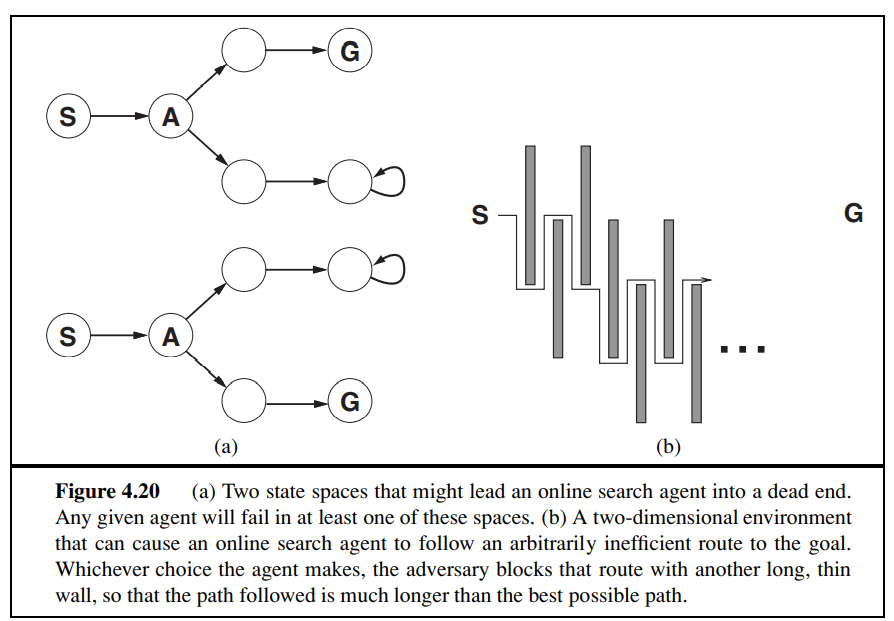
competitive ratio = actual distance travelled by the agent / shortest path from S to G

where S = starting state and G = goal state

Thus, the performance of online search agents is put in relation with the performance of an offline search agent for the same environment.



In other words, the competitive ratio could be infinite when the agent is placed in an environment that allows irreversible actions. In other words, it is not possible for the agent to go back to the parent state of the current state. The current state is called DEAD-END. NO ALGORITHM CAN AVOID DEAD ENDS. However, in most cases we assume that the environment is safely explorable and so there are no dead ends. When an environment always allows reversible actions then it can be represented as an undirected graph rather than a directed one. Figure 4.20 (a) proves that it is not possible to avoid dead ends. It is important to notice that an agent does not know beforehand that a dead end is a dead end because it does not know the transition function from that state.



## An implementation of an online search agent

After each action, an online agent receives a percept telling it what state it has reached; from this information, it can augment its map of the environment. The current map is used to decide where to go next. This interleaving of planning and action means that online search algorithms are quite different from the offline search algorithms we have seen previously. For example, offline algorithms such as A∗ can expand a node in one part of the space and then immediately expand a node in another part of the space, because node expansion involves simulated rather than real actions. An online algorithm, on the other hand, can discover successors only for a node that it physically occupies. To avoid traveling all the way across the tree to expand the next node, it seems better to expand nodes in a local order (as the objective it to minimise the distance travelled by the agent). Depth-first search has exactly this property because (except when backtracking) the next node expanded is a child of the previous node expanded. An online depth-first search agent is shown in Figure 4.21. This agent stores its map in a table, RESULT[s, a], that records the state resulting from executing action a in state s. Whenever an action from the current state has not been explored, the agent tries that action. The difficulty comes when the agent has tried all the actions in a state. In offline depth-first search, the state is simply dropped from the queue; in an online search, the agent has to backtrack physically. In depth-first search, this means going back to the state from which the agent most recently entered the current state. To achieve that, the algorithm keeps a table that lists, for each state, the predecessor states to which the agent has not yet backtracked. If the agent has run out of states to which it can backtrack, then its search is complete. We recommend that the reader trace through the progress of ONLINE-DFS-AGENT when applied to the maze given in Figure 4.19. It is fairly easy to see that the agent will, in the worst case, end up traversing every link in the state space exactly twice. For exploration, this is optimal; for finding a goal, on the other hand, the agent’s competitive ratio could be arbitrarily bad if it goes off on a long excursion when there is a goal right next to the initial state. An online variant of iterative deepening solves this problem (as it makes the agent proceeds level by level). Because of its method of backtracking, ONLINE-DFS-AGENT works only in state spaces where the actions are reversible. There are slightly more complex algorithms that work in general state spaces, but no such algorithm has a bounded competitive ratio.

