Finding Promising Exploration Regions by Weighting Expected Navigation Costs

Mark B. Ring
Mark.Ring@GMD.de
GMD — German National Research Center for Information Technology
Schloß Birlinghoven, D-53 754 Sankt Augustin
Germany

April 11, 1996

Abstract

In many learning tasks, data-query is neither free nor of constant cost. Often the cost of a query depends on the distance from the current location in state space to the desired query point. Much can be gained in these instances by keeping track of (1) the length of the shortest path from each state to every other, and (2) the first action to take on each of these paths. With this information, a learning agent can efficiently explore its environment, calculating at every step the action that will move it towards the region of greatest estimated exploration benefit by balancing the exploration potential of all reachable states encountered so far against their currently estimated distances.

1 Tasks with Distance Relationships

In many learning tasks, data-query is neither free nor of constant cost. Often the cost of a query depends on the distance from the current location in state space to the desired query point. This is easiest to visualize in robotics environments where a robot must physically move to a location in order to learn something there. The cost of this learning is the time and effort it takes to reach the new location. Furthermore, this cost is characterized by a *distance relationship*: When the robot moves as directly as possible from a source state to a destination state, the states through which it passes are closer (i.e., cheaper to reach) than is the destination state.

Distance relationships hold in many real-world, non-robotics tasks also: whenever information from any desired state cannot be queried at will. Optimizing the performance of a chemical plant, for example, requires the adjustment of analog controls which have a continuum of intermediate states. Querying possibly optimal regions of state space in these environments is inadvisable if the path to the query point intersects a region of known volatility.

In continuous environments, some first-order approximations to such distance-dependent active learning has already been done [2, 5, 7, 8]. In these cases, the learning agent follows a gradient towards promising learning areas by taking the action at each step that maximizes a local ignorance measure. These techniques have no explicit way of balancing the exploration of mildly promising local areas with greatly promising distant areas.

2 Keeping Track of Navigation Costs

In discrete environments with small numbers of states, it is possible to keep track of precisely where and to what degree learning has already been done sufficiently and where it still needs to be done. It is also possible to keep best known estimates of the distances from each state to each other (see Kaelbling, 1993). Kaelbling's DG-learning algorithm is based on Floyd's all-pairs shortest-path algorithm [1] and is just slightly different from that used here. These "all-goals" algorithms (after Kaelbling) can provide a highly satisfying representation of the distance/benefit tradeoff.

The following describes a minor variation of the DG algorithm that is perhaps a bit more direct. Associated with every pair of states x and y are two quantities: D_{xy} and A_{xy}. D_{xy} is the *distance* (navigation cost) from state x to state y, and A_{xy} is the *originating action* the agent takes in state x to move most directly (most cheaply) to state y. This information can be learned incrementally and *completely*: that is, if a path from any state x to any state y is deducible from the state transitions seen so far, then the following three conditions can be guaranteed. (1) The algorithm will know a path from x to y. (2) The current value for D_{xy} will be the best deducible value from all data seen so far. (3) No next action could be found, given the data seen so far, that moves the agent towards state y from state x more directly then A_{xy}.

The modified DG-algorithm for deterministic environments is given in Figure 1. All

1 If $D[S_{t-1}][S_t] > d_t$ then
2   FROM $\Leftarrow \{\}$
3   TO $\Leftarrow \{\}$
4     $D[S_t][S_{t+1}] \Leftarrow d_t$
5     $A[S_{t-1}][S_t] \Leftarrow a_t$
    /* See if this improves any paths to S_t. */
6   For all x in S (except S_{t-1}):
7   if $D[x][S_t] > D[x][S_{t-1}] + d_t$
8             $D[x][S_t] \Leftarrow D[x][S_{t-1}] + d_t$
9             $A[x][S_t] \Leftarrow A[x][S_{t-1}]$
10   FROM $\Leftarrow$ FROM $\cup\ x$
    /* See if this improves any paths from S_{t-1}. */
11   For all y in S (except S_t):
12   if $D[S_{t-1}][y] > d_t + D[S_t][y]$
13             $D[S_{t-1}][y] \Leftarrow d_t + D[S_t][y]$
14             $A[S_{t-1}][y] \Leftarrow a_t$
15   TO $\Leftarrow$ TO $\cup\ y$
    /* Improve paths from all $x \in$ FROM to all $y \in TO$. */
16   For all x in FROM:
17   for all y in TO:
18   if $D[x][y] > D[x][S_{t-1}] + d_t + D[S_t][y]$
19                 $D[x][y] \Leftarrow D[x][S_{t-1}] + d_t + D[S_t][y]$
20                 $A[x][y] \Leftarrow A[x][S_{t-1}]$

Figure 1: Modified all-goals learning algorithm.

costs are stored in the two-dimensional array $D[x][y]$, which is initialized with infinite costs from every state x to every state y. S_τ is the state the agent is in at step τ, so S_t is the agent's current state and S_{t-1} was its previous state. Steps 2 through 20 are only executed if the action just taken (a_t) has revealed information that can reduce the cost of at least one known path. If the previously stored cost from S_{t-1} to S_t is greater than the cost just experienced (d_t), then this new cost is stored instead (line 4), and the action the agent took to achieve it is also stored (line 5). Lines 6 through 20 search for other paths that might be reduced with this new knowledge. Lines 6 through 10 search for all states x whose stored path to S_t can be improved by first going to S_{t-1} (and then taking action a_t). Any such states found are then stored in a set called "FROM" (line 10), and the stored distances from these states to S_t are updated (line 8). The originating actions from these states are also updated (line 9). Lines 11 through 15 do exactly the same thing in the opposite direction, finding all states y to which paths from S_{t-1} can be improved by going first to S_t. These states are collected into a set named

"TO" (line 15). Lines 16 through 20 update all remaining paths that might be improved by going through S_{t-1} and S_t.

So long as the D and A arrays accurately reflected the best deducible distances and actions at $t-1$ (before the last action was taken), no paths can be improved other than (1) those whose distances to S_t are reduced by going first to S_{t-1} (the FROM set), (2) those to which the distance from S_{t-1} can be improved by going first to S_t (the TO set), and (3) those originating at a FROM state and terminating at a TO state that can be improved by passing through S_{t-1} and S_t.

The proof (by contradiction) is very simple. For the proof, index D according to when it was last modified: D^{t-1} represents the state of the matrix just before taking action a_t. Since the information gained from the last action is only useful if it *reduces* an entry in D, only paths that can be *improved* by taking the last action are affected. Assume that after following the procedure given above, there is still some path, p_{xy} (from some state x to some state y) that can be found using the current knowledge to have a total cost of less than that stored in D^t_{xy}. That is, D^t_{xy} is not minimal given knowledge at t. Since the only new information introduced at time t is d_t, the path p must pass through S_{t-1} and S_t. For the assumption to be true, then, either x was not in the FROM set, or y was not in the TO set. That is, although it is the case that D_{xy} can be improved by going through S_{t-1} and S_t:

$$D^t_{xy} > D^{t-1}_{xS_{t-1}} + d_t + D^{t-1}_{S_t y},$$

one of the following is also true: either x was not in the FROM set,

$$D^{t-1}_{xS_t} \leq D^{t-1}_{xS_{t-1}} + d_t, \tag{1}$$

or y was not in the TO set,

$$D^{t-1}_{S_{t-1}y} \leq d_t + D^{t-1}_{S_t y}. \tag{2}$$

In the case of (1), D^{t-1}_{xy} could have been reduced to $D^{t-1}_{xS_t} + D^{t-1}_{S_t y}$ at $t-1$. In the case of (2) D^{t-1}_{xy} could have been reduced to $D^{t-1}_{xS_{t-1}} + D^{t-1}_{S_{t-1}y}$. However, since D^{t-1} already held the best deducible values, neither (1) nor (2) can be true, and the assumption is therefore false.

3 Full Exploration Guarantee

If nothing is known about the environment, a simple strategy will guarantee complete exploration: move next towards any known state whose actions have not yet all been tested. This is due to the following property that can be proven for the agent described above in any deterministic environment, regardless of the actions available:

> If there are any *reachable* states that have not yet been fully explored, then there is also a *known* path from the current state to a state with unexplored actions.

More formally, define the following sets: R, K, and U. R is the set of all states reachable from the current state. K is the set of all states to which a path from the current state is known ($K \subset R$). U is the set of all underexplored states (i.e., states in which there is at least one action that has not yet been tried).

Theorem 1 $[\exists x : x \in U \cap R] \rightarrow [\exists y : y \in U \cap K]$.

Proof: Since x is reachable, a path exists from the current state to x. The first underexplored state y encountered on this path (i.e., $y \in U$) will be x if no earlier such state is encountered. All states on the path before y are fully explored. Therefore, all actions on the path from the current state to y have been tried. Since all actions from the current state to y have been tried, a path from the current state to y is deducible and therefore (from above) such a path is also known, i.e., $y \in K$.

As a result, the agent will always know a path to an underexplored state until all reachable states have been explored. (The existence of this proof was also implied by Rivest and Schapire [6].)

4 Locality and Exploration Benefit

Though the strategy just given will guarantee *complete* exploration, it does not imply *efficient* exploration when certain common properties about the environment might be known in advance. One of these is the property of *locality*. Locality is exhibited by environments in which an action taken in a state will not generally take the agent very far away. That is, after taking action a in state x, the agent will be taken to a state y where D_{yx} is small. The smaller this value is on average, the greater the environment's locality. This property of locality is very high, for example, in most robotics applications.

Another aspect of the environment that may be known is the degree to which certain states may be more interesting than others — for example, the degree to which they offer a greater exploration benefit. In these cases, the agent can better organize its exploration strategy if it has access to a quantity E_x that specifies the exploration benefit of visiting state x. With this information, decisions about which areas to explore next can be based on both the benefit resulting from the exploration as well as the cost of reaching each area.

4.1 Weighting Actions by Exploration Benefit and Navigation Costs

If the effects of all actions in all states were known before exploration began, planning the route that most efficiently visited every action in every state would be NP-hard, being equivalent to TSP. However, this information is not known in advance, which makes the optimization problem somewhat easier. In the general case, planning is useless: there is no better exploration strategy than to always move towards the closest known state that has an unexplored action, since there is no way to predict where that action could next take the agent.

If the environment exhibits some degree of locality, however, then it can be split into regions, one for each action available in S_t, such that all the states in each region have the same originating action from S_t. The agent can then take the action that moves it into the region with the best tradeoff between proximity and exploration benefit, since the chances are high that unexplored actions will leave the agent within approximately the same region. A value, w_a is computed for each region a that balances the benefits of visiting the states in the region against their distances from the current state.

For each action a in state S_t

$$w_a = \sum_{\{y|a=A_{S_t y}\}} f(D_{S_t y}, E_y)$$

The action taken next is that which leads in the direction of greatest accumulated exploration benefit (i.e., the one whose weight is the largest). Ties are broken arbitrarily. Different functions f can be used to find different ways of balancing small local benefit against large but distant benefit. And, of course, even when all nearby regions are fully explored, the agent will automatically take the least-cost path to distant regions of highest potential exploration benefit.

4.2 Balancing Exploitation with Exploration

Most often, exploration is not the only or even the central goal of a learning agent. Usually the agent has a primary goal — a value to maximize or a state to reach — toward which exploration is only of long-term benefit. In these cases, the agent must have some other measure besides E that specifies the intrinsic goodness of taking certain actions in certain states. Fortunately, the E values can be modified to explicitly balance exploitation with exploration, that is, to take advantage of short-term goal opportunities while also pursuing the long-term benefits of exploring new actions.

Some actions, for example, may provide short-term benefit but as a consequence may bring the agent to a place far from its previous state (i.e., the return distance is great.) While returning to achieve another short-term benefit, it may be worthwhile for the agent to explore adjacent areas. The E values might therefore be a sum of the intrinsic benefit and the exploration benefit of visiting a state.

5 Some Results

When implemented in a two-dimensional grid world with random obstacles, the benefits of the exploration approach described above become evident. In these environments there are four possible actions in each state, and each state is accessible from every other. A lower bound on the number of actions needed to fully explore these environments is NA (where A is the average number of actions available in each state), though this can only be achieved if the topology of the environment is known in advance (or by great luck). Random exploration performs particularly poorly. With about 85 states in a 10x10 grid (i.e., about 15% of the grid is occupied by obstacles), about 4200 actions are required on average before all state/action pairs have been tried, or roughly 1100% above the lower bound (which is 340 in this environment). Better performance is achieved when

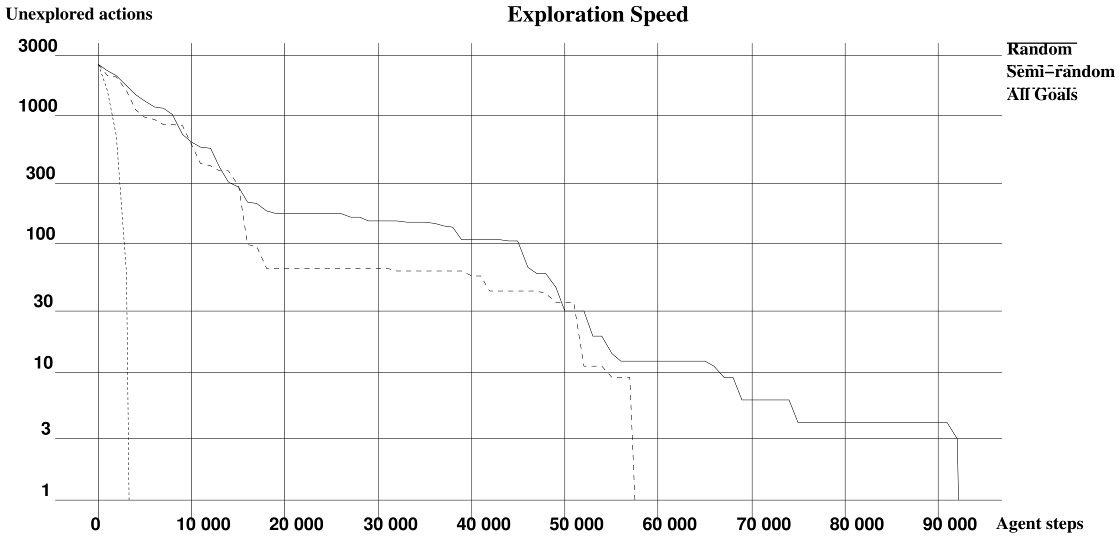

Figure 2: Comparisons of the three exploration methods mentioned in the text. The vertical access shows the number of actions left to explore. The horizontal axis shows the number of steps the agent has taken so far. The all-goals method explores all actions quickly and without delay. Both the random strategy and the semi-random strategy (where exploration is locally intelligent) get slower as the number of unexplored actions decreases. Notice that the vertical axis is logarithmic, so that approximately the bottom third of the graph is devoted to the last 10 actions (less than 0.4%).

actions are locally intelligent: untested actions in a state are tried before already-tested actions are retried. This yields about 2200 actions to fully explore 85 states, or about 550% above the lower bound. In contrast, the method described in Section 4.1 requires approximately 410 actions in an environment of 85 states when $f(x, y) = y/x^4$. This is about 20% above the lower bound. Of course, the performance of the two random approaches scales miserably. The all-goals approach scales extremely well: even with 4500 states, the number of actions taken was still less than 20% above the lower bound.

Figure 2 graphs the performance of the three methods in a small sample environment of 631 states and 2524 actions. The two random methods are dominated by the time spent finding a small number of remaining unexplored actions.

6 Compression Strategies

By far the biggest drawback of using this method is the overhead imposed by the $O(N^2)$ storage requirement, where N is the number of states. The most space-saving representation of the environment is a single-step model that only keeps track of the immediate costs and results of each action. Though the space complexity drops to $O(NA)$, the time required to find the distance between two states also rises to $O(NA)$.

An alternative method for compressing the D matrix after a complete exploration of the environment is as follows:

1. Choose the state, V, that lies on the greatest number of shortest paths, (i.e., where $D_{xy} = D_{xV} + D_{Vy}$ for the greatest number of pairs x, y.)

2. Create two sets: V_I and V_F:
V_I = set of all initial states, I, for each of the paths in 1.
V_F = set of all final states, F, for each of the paths in 1.

3. For each x in V_I and each y in V_F, store its distance to or from V respectively: D'_{xV} and D'_{Vy}. Also store the originating actions: A'_{xV} and A'_{Vy}.

4. Remove all entries D_{xV}, D_{Vy}, and D_{xy} from D, where $x \in V_I$, and $y \in V_F$.

Repeat steps 1 through 4 until no more pairs remain in D. To find the distance from state x to state y, find the state V where $x \in V_I$ and $y \in V_F$ such that $D'_{xV} + D'_{Vy}$ is minimal. Experimental results in 2-dimensional grid worlds demonstrate a final size of about $O(N lg^2 N)$ for D' using this method, but the cost is the access time $O(lg N)$ to find the distance between two states. Worse still, the time needed for the compression is $O(N^4)$, mostly due to the first line of the algorithm.

An alternate scheme replaces line 1 with a heuristic. For each state V, the heuristic finds two values: (1) the number of entries in D that go either to or from V; (2) the sum of these entries' values. The first divided by the second squared provides a rating for V. The state with the highest rating is chosen. This seems to work fairly well and results in an average space requirement of $O(N^{1.5})$ for D' while reducing the run time of the algorithm to $O(N^3)$.

Unfortunately, neither of these schemes could be said to require less than $O(N^2)$ space, since the complete D matrix is needed from the outset. So far, none of the attempts to find an incremental version of the algorithm have yielded a satisfactory result. Kaelbling [3] has introduced some methods that use less space while approximating the D matrix using neural networks and hierarchies of distances. Though promising, this approach has not yet yielded a practical solution to the problem.

One highly pleasing, very intuitive idea that unfortunately does not seem to improve the situation at all, is to assign each state a coordinate in some multidimensional space and then to apply a distance function to the coordinates of the two states to determine their distance. It can easily be proven that no Euclidean distance function will suffice in an environment with obstacles, regardless of the number of dimensions used. The best non-Euclidean function I have yet discovered is that described by the method above, where a state's coordinates are its distances in all the sets V_I and V_F.

7 Conclusions

The general principle of using all-goals learning in unknown deterministic environments has many positive characteristics. Chief among these is the efficiency with which the environment can be explored. Of equal importance is the intuitive way in which exploration can be balanced with exploitation. Also of major importance is the fact that, once the knowledge is gained, it can be used by the agent to reach any goal from any starting state in the provably minimal number of steps. For small state spaces (given today's computer capacities, this means not more than ten thousand states), the advantages of the technique far outweigh its drawbacks.

There are drawbacks, however, in terms of space and time complexity. The two major obstacles that keep this approach from being applicable to large or continuous state spaces are (1) the $O(N^2)$ space requirement, and (2) the $O(N)$ time required at every step to choose the best action. Furthermore, the method described above deals only with deterministic environments. The amount of storage required for stochastic environments is even higher. Should the next stage of research — reducing these requirements while still learning incrementally — be successful, the resulting algorithm will provide an enormously powerful way to explore any unknown environment.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, 1983.

[2] David Cohn. Neural network exploration using optimal experiment design. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 679–686, San Mateo, California, 1994. Morgan Kaufmann Publishers.

[3] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Machine Learning: Proceedings of the tenth International Conference*, pages 167–173. Morgan Kaufmann Publishers, June 1993.

[4] Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1094–1098, Chambéry, France, 1993. Morgan Kaufmann.

[5] Alexander Linden and Frank Weber. Implementing inner drive through competence reflection. In J. A. Meyer, H. Roitblat, and S. Wilson, editors, *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 321–326. MIT Press, 1993.

[6] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, April 1993.

[7] Jürgen Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI–149–91 (revised), Technische Universität München, Institut für Informatik, April 1991.

[8] Sebastian B. Thrun and Knut Möller. Active exploration in dynamic environments. In J. E. Moody, S. J. Hanson, and R. P. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 531–538, San Mateo, California, 1992. Morgan Kaufmann Publishers.