Learning to Act Using Real Time Dynamic Programming - Barto et. al [1993]

Monica Patel (260728093), Pulkit Khandelwal (260717316)

3-Feb-2017

McGill University

Synopsis of the Paper

- Bridge between AI research on real-time planning and control AND concepts from control theory
- Understanding on-line off-line decision making.
- Introduces Real-Time Dynamic Programming [RTDP]: a learning algorithm based on DP by using Asynchronous DP
- Compares RTDP, Adaptive RTDP, Q-Learning and Conventional DP algorithms.

Dynamic Programming

Given a complete and accurate model of the environment in the form of an MDP, it is possible to compute optimal policies i.e. solve the decision problem off-line by using a DP algorithm.

Understanding Shortest Path Problem

- World in the problem are divided discretely into grids.
- Set of start and goal points are give to the driver of the car.
- Cost of traveling from one node to its neighbor node is constant $= 1$
- Set if allowed actions for car are up, down, right, left, diagonally to adjacent grid node
- Car does not actually performs control action with certainty.
- Driver has to reach goal in minimum cost.

Synchronous Dynamic Programming (Shortest Path Problem)

- Involves operations over the entire state set of the MDP, i.e., they require sweeps of the complete state set.
- The following update is done for each state in state set, until f converges to f*

$$f_{k+1} = \min_{\forall u}[c_i(u) + \gamma \sum_{\forall j} P_{ij}(u) f_k(j)]$$

- During the update only old values of states are considered and not the updated values, meaning update of one state is independent of new updated value of other state and so the process can be parallelized.
- After the optimal value function is calculated greedy policy will lead to goal.
- Computationally expensive and sometimes infeasible if problem is large and unstructured.
- Algorithms in Synchronous:
  - Value Iteration
  - Policy Iteration

Policy Evaluation

- Iterate using *pi* instead of max(a) for every state S:

$$V_{k+1}(s) = C(\pi(s), s) + \gamma \sum_{\forall s'} P(s'|\pi(s), s) V_k(s') \tag{1}$$

Policy Iteration

- Initialise (randomly)

$$\pi_o$$

- Iterate:
    - Policy Evaluation: Compute

$$V_{\pi_k}$$

    - Policy Improvement:

$$\pi_{k+1}(s) = argmax_a Q_{\pi_k}(a, s)$$

Value Iteration

Given the Bellman Equation:

$$v_*(s) = max_a[C(s,a) + \gamma \sum_{\forall s'} P(s'|s,a)v_*(s')] \tag{2}$$

Iterate (for every S):

$$v_{k+1}(s) = max_a[C(s,a) + \gamma \sum_{\forall s'} P(s'|s,a)v_k(s')] \tag{3}$$

Stopping Criterion:

$$max_s|V_{k+1}(s) - V_k(s)| <= eps \tag{4}$$

Gauss-Seidal Dynamic Programming

- Gauss-Seidal differs from synchronous DP in the way the costs are updated for each state
- In synchronous version sequential ordering of the backups were irrelevant to the results.
- In Gauss-Seidal Costs are backed up one state at a time in sequential sweep.
- Assuming that the states are ordered in number, the sweep proceeds in number.
- Unlike synchronous DP while updating value of next state updated values of previous states are considered and not the old values.
- It can be seen by the following equation:

$$f_{k+1} = \min_{\forall u}[c_i(u) + \gamma \sum_{\forall j} P_{ij}(u) f_k(j)]$$

where,

$$f(j) = f_{k+1}(j), j < i$$
$$f(j) = f_k(j), otherwise$$

Prioritized Sweeping

- Similar to Gauss-Seidel Value Iteration, but the sequence of states in each iteration is proportional to their update magnitudes (Bellman errors).
- Bellman Error is the change in the state's value after the most recent update
$$E[s; V_t] = |V_{t+1}(s) - V_t(s)| \tag{5}$$
that is the change of ss value after the most recent update

Asynchronous Dynamic Programming

- Like Gauss-Seidal Asynchronous DP does not back up the cost simultaneously
- Additionally Asynchronous DP can use parallel computation power unlike Gauss-Seidal.
- Sweep of the state set is not done sequentially but in asynchronous DP subset the state set is selected and updated.
- Next sweep uses the updated values of the state from previous update.
- If all states are updated infinitely, convergence is still guaranteed.
- It can be seen by the following:

$$f_{k+1}(i) = \min_{\forall u} Q_{f_k}(i, u), i \in chosenSubset$$

$$f_{k+1}(i) = f_k, otherwise$$

Real Time Dynamic Programming

- The methods described so far are off-line methods of solving the MDP.
- Meaning the successive steps in the algorithms are not related to actual time step.
- In real time dynamic programming controller performs asynchronous DP concurrently with taking actual control process.
- This works in following way
    - Control decision are taken based on most recent update of cost of states in state space.
    - Subset chosen from state set for asynchronous DP is influenced by the control.

LRTA*

Korf's Heuristic search problem can be applied to state-space search problem with following additional properties:

- Each time there is unique current state of system being controlled. (Not going back to state which is already seen)
- At each bounded time interval, one action should be choses.
- System changes state at the end of the time interval.

Algorithm

Simplest form of LRTA*

- Calculate cost for neighbors of current state by:

$$cost_f(i, u) = c_i(u) + \gamma \sum_{\forall j} P_{i,j}(u) h(j) \tag{6}$$

- Update the estimate of node i as:

$$h_{new}(i) = \min_j h(j) \tag{7}$$

- Update current state with the state with minimum cost.

This algorithm can also be modified to do forward search up-to certain depth d, given that computation is bounded by the real world step time T and then selecting the state which gives least cost to the depth d.

Features

- Notice that while checking the heuristic of the nodes, only neighbors are checked, that is only those nodes are check where control action action will likely lead instead of complete state space.
- Unlike off-line control design this is gives advantage to controller to make decision for a subregion of the state space when it can be reached by actual control and incorporate any changes that happened at previous times.
- Some times complete control design is not possible off-line due to large or unstructured nature of control problem.

Whats happening in RTDP

- Instead of just backing up the value of current Node, Node on which driver currently is, in RTDP forward search can be performed up-to certain depth and values of all the nodes in forward search are updated like asynchronous DP
- To put it in algorithm:
  - Synchronously backup the cost of leaf nodes (Max depth nodes)
  - Synchronously backup the cost of predecessor nodes of leaf node (Max depth nodes) till current node
  - Asynchronously Backup the whole set cost.
  - Then take the control action.
  - Note that computation time of above is bounded by the time between two control actions.

Implementation of Synchronous DP for Grid World

Statement of Contribution

- *Monica Patel*: Application of Synchronous DP, Gauss-Seidel, LRTA* and RTDP to shortest path problem on a grid world and corresponding explanation in presentation.
- *Pulkit Khandelwal*: Application of Synchronous DP: Policy Evaluation, Policy Iteration, Value Iteration; Asynchronous DP: Gauss-Seidel Value Iteration on a grid world and corresponding explanation in presentation.

POINT DRIVER MADE IT TO GOAL

THANK YOU