Dynamic Programming

Asynchronous Dynamic Programming

Sarwan Ali

Department of Computer Science
Georgia State University

Advanced Dynamic Programming Techniques

Today's Learning Journey

Motivation: Limitations of Synchronous DP

Synchronous Dynamic Programming Challenges

- **Computational Burden**: Updates all states in each sweep
- **Memory Requirements**: Requires two value function arrays
- **Uniform Updates**: Equal attention to all states regardless of importance
- **Sequential Dependency**: Must complete full sweep before next iteration

Key Question

Can we be more **selective** and **efficient** in our updates?

↓

Asynchronous Dynamic Programming

What is Asynchronous Dynamic Programming?

Definition (Asynchronous DP)

Dynamic programming algorithms that update states **selectively** and **in-place**, without requiring systematic sweeps through the entire state space.

Synchronous DP

- Updates all states
- Fixed order
- Two arrays needed
- Sweep-based

Asynchronous DP

- Selective updates
- Flexible order
- In-place updates
- Continuous process

Fundamental Principles of Asynchronous DP

1. **In-Place Updates**
   - Use updated values immediately
   - No need for separate arrays
   - $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
2. **Selective State Updates**
   - Focus on important/relevant states
   - Skip states that don't need updating
   - Prioritize based on value changes
3. **Convergence Guarantee**
   - All states must be updated infinitely often
   - No state can be permanently ignored
   - $\lim_{k\to\infty} \max_s |V_k(s) - V^*(s)| = 0$

Mathematical Foundation

Bellman Optimality Equation (Reminder)

$$V^*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V^*(s')]$$

Asynchronous Update Rule

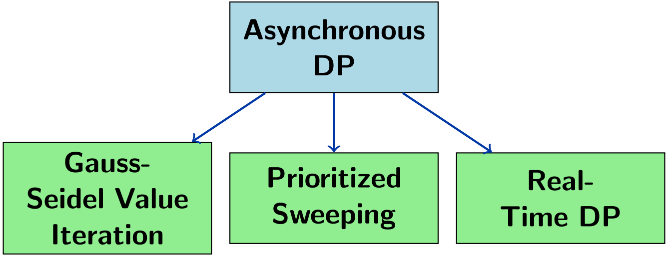
For any state s at any time: $V_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]$
Where $V_k(s')$ uses the **most recent** value available.

Theorem (Convergence of Asynchronous DP)

If all states are updated infinitely often, then:

$$\lim_{k \to \infty} V_k(s) = V^*(s) \quad \forall s \in \mathcal{S}$$

Types of Asynchronous DP Algorithms



- **Gauss-Seidel**: Sequential in-place updates
- **Prioritized Sweeping**: Update states based on priority
- **Real-Time DP**: Update states as they are visited

Gauss-Seidel Value Iteration

Algorithm Concept

Update states sequentially using the most recent values available.

Initialize $V(s) = 0 \ \forall s$
repeat
  for each state $s \in \mathcal{S}$ **do**
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)$
      $[r + \gamma V(s')]$
  end for
until convergence

Key Features

- Uses updated $V(s')$ immediately
- Only one value array needed
- Often faster convergence
- Order of updates matters

Prioritized Sweeping

Core Idea

Update states in order of their **priority** - how much their value is expected to change.

Definition (Priority)

Priority of state s: $\text{Priority}(s) = \left|\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] - V(s)\right|$

Initialize priority queue Q, $V(s) = 0\ \forall s$
repeat
    $s \leftarrow$ state with highest priority from Q
    Update $V(s)$ using Bellman equation
    for each predecessor $\bar{s}$ of s **do**
        Compute priority of $\bar{s}$
        if priority $>$ threshold **then**
            Add $\bar{s}$ to Q
        end if
    end for
until Q is empty or convergence

Real-Time Dynamic Programming

Motivation

Update only states that are **actually visited** during execution or simulation.

Initialize $V(s) = 0\ \forall s$
$s \leftarrow$ start state
repeat
  Update $V(s)$ using Bellman equation
  Choose action a (e.g., ϵ-greedy)
  $s \leftarrow$ next state
until termination

Advantages

- Focuses on relevant states
- Suitable for large state spaces
- Can run during execution
- Natural for online learning

Trade-off

May not find globally optimal policy if some states are never visited.

Convergence Requirements

Theorem (Asynchronous DP Convergence)

Asynchronous DP converges to V^* if and only if:

1. *All states are updated infinitely often*
2. *Updates use the Bellman optimality operator*
3. *The MDP satisfies standard assumptions (finite states, bounded rewards)*

Practical Implications

- **No state can be permanently ignored**
- Updates can be in any order
- Can skip states temporarily
- Convergence rate depends on update strategy

Convergence Rate Analysis

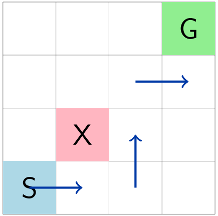
Factors Affecting Convergence Speed

1. **Update Order**
   - Some orders converge faster than others
   - Gauss-Seidel often faster than Jacobi
2. **State Prioritization**
   - Prioritized sweeping focuses on important changes
   - Can achieve faster practical convergence
3. **Problem Structure**
   - Connectivity of state space
   - Distribution of optimal paths

No Universal Best Order

The optimal update order is problem-dependent and often unknown a priori.

Example: Grid World with Asynchronous DP



Synchronous vs Asynchronous

Synchronous:

- Update all 16 states
- Multiple sweeps needed

Asynchronous:

- Focus on path states
- Prioritize by value change
- Faster convergence

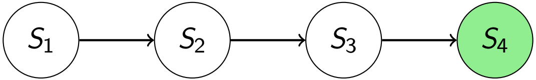
Update Strategy

Start from goal and work backwards, or follow trajectories from start state.

Numerical Example: Prioritized Sweeping

Simple Chain MDP

States: $\{S_1, S_2, S_3, S_4\}$, Actions: $\{\text{left}, \text{right}\}$
Rewards: $R(S_4) = +10$, all others $= 0$



Priority Order:

1. S_4: Priority $= 10$
2. S_3: Priority $= \gamma \times 10$
3. S_2: Priority $= \gamma^2 \times 10$
4. S_1: Priority $= \gamma^3 \times 10$

Update Sequence:

- Update S_4 first
- Then S_3 (affected by S_4)
- Then S_2 (affected by S_3)
- Finally S_1 (affected by S_2)

Advantages of Asynchronous DP

Computational Benefits

- **Memory Efficient**: In-place updates
- **Faster Convergence**: Often requires fewer computations
- **Flexible**: Can adapt to problem structure
- **Scalable**: Better for large state spaces

Practical Benefits

- **Online Learning**: Can update during execution
- **Anytime Algorithm**: Can stop and resume
- **Prioritization**: Focus on important states
- **Real-time**: Suitable for time-constrained environments

Key Advantage: Efficiency without sacrificing optimality

Limitations and Challenges

Theoretical Challenges

- **Convergence Guarantee**: Must update all states infinitely often
- **Order Dependency**: Convergence rate depends on update order
- **No Universal Strategy**: Best approach is problem-dependent

Practical Challenges

- **Implementation Complexity**: More complex than synchronous versions
- **Debugging Difficulty**: Harder to track and debug
- **Priority Computation**: Additional overhead for prioritization
- **Memory Access Patterns**: May not be cache-friendly

When to Avoid

- Small, simple problems where synchronous DP is sufficient
- When deterministic, predictable behavior is required

Implementation Best Practices

Data Structures

- **Priority Queue**: For prioritized sweeping (heap-based)
- **State Tracking**: Keep track of when states were last updated
- **Predecessor Lists**: For efficient backward propagation

Algorithmic Considerations

- **Threshold Values**: When to add states to priority queue
- **Termination Criteria**: When to stop updating
- **Update Scheduling**: How to ensure all states are updated
- **Numerical Stability**: Handle floating-point precision issues

Performance Optimization

- **Sparse Representations**: For large, sparse state spaces
- **Parallel Updates**: When states are independent
- **Incremental Computation**: Reuse computations when possible

Pseudocode: Prioritized Sweeping Implementation

Complete Algorithm

```
function PrioritizedSweeping(MDP, threshold):
    Initialize V(s) = 0 for all s
    Initialize priority queue PQ
    // Initial population of queue
    for each state s:
        priority = |BellmanUpdate(s) - V(s)|
        if priority > threshold:
            PQ.insert(s, priority)
    while PQ is not empty:
        s = PQ.extractMax()
        V(s) = BellmanUpdate(s)
        for each predecessor p of s:
            priority = |BellmanUpdate(p) - V(p)|
            if priority > threshold:
                PQ.insert(p, priority)
    return V
```

Advanced Asynchronous DP Variants

Bounded Real-Time DP

- Limits computation time per decision
- Updates multiple states per action selection
- Balances planning time with action quality

Focused Dynamic Programming

- Uses reachability analysis
- Only considers states reachable from start state
- Efficient for problems with many irrelevant states

Parallel Asynchronous DP

- Multiple processors update different states
- Requires careful synchronization
- Can achieve significant speedup

Connection to Modern RL

Relationship to Temporal Difference Learning

- TD learning can be viewed as asynchronous DP with sampling
- Both use immediate updates of value estimates
- Asynchronous DP uses full model, TD uses sample transitions

Dyna Architecture

- Combines learning and planning
- Uses prioritized sweeping for background planning
- Updates model and values asynchronously

Modern Deep RL

- Experience replay can be seen as prioritized update mechanism
- Prioritized experience replay directly inspired by prioritized sweeping
- Asynchronous methods in deep RL (A3C, etc.)

Key Takeaways

Core Concepts

- **Flexibility**: Asynchronous DP provides flexible, efficient alternatives to synchronous methods
- **Convergence**: Guaranteed convergence with proper update requirements
- **Efficiency**: Often faster convergence with less memory usage

Practical Impact

- **Scalability**: Enables DP for larger problems
- **Real-time Applications**: Suitable for online and real-time scenarios
- **Foundation**: Basis for modern RL algorithms

Asynchronous DP: Efficiency meets Optimality

Next Steps

What's Coming Next

- **Monte Carlo Methods**: Model-free approaches
- **Temporal Difference Learning**: Combining ideas from DP and MC
- **Policy Gradient Methods**: Direct policy optimization

Study Recommendations

- Implement prioritized sweeping on a grid world
- Compare convergence rates of different asynchronous methods
- Explore the connection to modern deep RL methods

Questions?