Complexity Comparison

Let:

- $|S|$: Number of states (10,000 in the code).
- $|A|$: Number of actions (3 in the code).
- $|\phi|$: Feature dimension in ALP (12 in the code, from `Phi.shape[1]`).
- k: Number of iterations for DP convergence.
- c: Number of constraints in LP/ALP.

1. Dynamic Programming (Value Iteration)

- **Time Complexity**:
    - Each iteration updates $V(s)$ for all $|S|$ states.
    - For each state, compute the max over $|A|$ actions, each involving a sum over possible next states (in this deterministic case, one next state per action, so $O(1)$ per action).
    - Total per iteration: $O(|S| \cdot |A|)$.
    - Assuming k iterations: $O(k \cdot |S| \cdot |A|)$.
    - In the code: $|S| = 10,000$, $|A| = 3$, $k = 226$, resulting in a runtime of 64.78s.
- **Space Complexity**:
    - Stores $V(s)$ and $\pi(s)$: $O(|S|)$.
    - Transition matrix T: $O(|S| \cdot |A|)$ (sparse, one next state per action).
    - Reward matrix R: $O(|S| \cdot |A|)$.
    - Total: $O(|S| \cdot |A|)$.

- In the code: Peak memory is 0.18MB, reflecting efficient storage (sparse T).

2. Linear Programming (LP)

- **Time Complexity**:
  - LP has $|S|$ variables (one $V(s)$ per state) and up to $|S| \cdot |A|$ constraints (one per state-action pair).
  - Solving an LP with interior-point methods (e.g., "highs") has complexity approximately $O((|S| + c)^{1.5} \cdot c)$, where $c \approx |S| \cdot |A|$.
  - In practice, the solver handles sparse matrices efficiently, but the large number of constraints $(10,000 \cdot 3 = 30,000)$ leads to a runtime of 9.02s.
- **Space Complexity**:
  - Stores constraint matrix A_{ub}: $O(|S| \cdot |A| \cdot |S|)$ (sparse, with $O(1)$ non-zeros per row).
  - Variables $V(s)$: $O(|S|)$.
  - Total: Dominated by A_{ub}, approximately $O(|S| \cdot |A|)$ for sparse storage.
  - In the code: Peak memory is 7207.77MB, indicating significant memory usage due to the dense constraint matrix construction.

3. Approximate Linear Programming (ALP)

- **Time Complexity**:
  - ALP has $|\phi|$ variables (feature weights θ) and up to $|S| \cdot |A|$ constraints.
  - LP solver complexity: $O((|\phi| + c)^{1.5} \cdot c)$, where $c \approx |S| \cdot |A|$.
  - Since $|\phi| \ll |S|$ (12 vs. 10,000), the variable count is reduced, but the constraint count remains high, leading to a runtime of 11.12s.
- **Space Complexity**:
  - Feature matrix Φ: $O(|S| \cdot |\phi|)$.
  - Constraint matrix A_{ub}: $O(|S| \cdot |A| \cdot |\phi|)$ (sparse).

- Variables θ: $O(|\phi|)$.
- Total: $O(|S| \cdot |\phi|) + O(|S| \cdot |A| \cdot |\phi|)$.
- In the code: Peak memory is 26.21MB, much lower than LP due to the reduced variable count and sparse matrix usage.

Summary Table

Method	Time Complexity	Space Complexity	Runtime	Memory
DP	$O(k \cdot \|S\| \cdot \|A\|)$.	$O(\|S\| \cdot \|A\|)$	64.78s	0.18 MB
LP	$O((\|S\| + c)^{1.5} \cdot c)$, where $c \approx \|S\| \cdot \|A\|$	$O(\|S\| \cdot \|A\|)$ for sparse storage	9.02s	7207.77 MB
ALP	$O((\|S\| \cdot \|\phi\|)$	$O(\|S\| \cdot \|\phi\|) + O(\|S\| \cdot \|A\| \cdot \|\phi\|)$	11.12s	26.21 MB

Comparison Insights

- **Time**:
  - **DP** is slower (64.78s) due to multiple iterations ($k = 226$), but each iteration is computationally lightweight.
  - **LP** is faster (9.02s) as it solves the problem in one optimization step, leveraging efficient LP solvers.
  - **ALP** is slightly slower than LP (11.12s) due to the overhead of feature matrix computation, but it benefits from fewer variables ($|\phi| \ll |S|$).
- **Space**:
  - **DP** is highly memory-efficient (0.18MB) due to sparse storage of T and R.

    - **LP** consumes excessive memory (7207.77MB) because of the dense constraint matrix, despite theoretical sparsity.
    - **ALP** is more memory-efficient than LP (26.21MB) due to the reduced variable count and sparse matrix usage.
- **Performance** (from code output):
    - **DP** achieves the highest average reward (-8914.96), serving as the baseline.
    - **LP** is close (-8927.24), with a max error of 1999.98 and 80/10,000 states differing in policy.
    - **ALP** has the lowest reward (-9061.88), likely due to the mismatched feature set (Taxi problem features), but still only differs in 80/10,000 states.

Conclusion

- **DP** is suitable for problems with moderate state spaces and offers exact solutions with low memory usage but requires many iterations.
- **LP** provides a faster alternative for exact solutions but demands significant memory for large state spaces.
- **ALP** is a scalable compromise, reducing the variable count but potentially sacrificing accuracy if features are poorly chosen (as seen in the code). For large MDPs, ALP is promising if feature engineering is improved.