NEURO-DYNAMIC PROGRAMMING AN OVERVIEW

Dimitri Bertsekas
Dept. of Electrical Engineering and Computer Science
M.I.T.

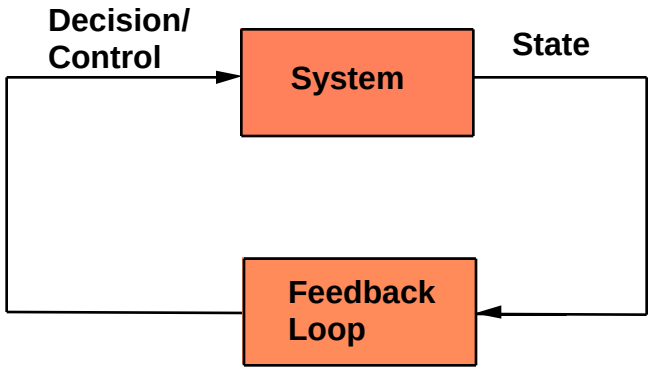
January 2001

OUTLINE

- Main NDP framework
- Relations to other methodologies
- Rollout algorithms
- Actor-critic methods
- Book references:
  - Neuro-Dynamic Programming (Bertsekas + Tsitsiklis)
  - Reinforcement Learning (Sutton + Barto)
  - Dynamic Programming: 2nd Edition (Bertsekas)
- Papers can be downloaded from http://web.mit.edu/dimitrib/www/home.html

DYNAMIC PROGRAMMING / DECISION AND CONTROL

- **Main ingredients:**
  - State evolving over time (e.g., a Markov chain)
  - Decision/control applied at each time
  - Reward is obtained at each time period
  - There may be noise & model uncertainty
  - There is state feedback used to determine the control



KEY DP RESULT: BELLMAN'S EQUATION

- **Optimal decision at the current state maximizes the expected value of**

  Current stage reward + Future stages reward starting from the next state (using opt. policy)

- **Extensive mathematical methodology**
- **Applies to both discrete and continuous systems (and hybrids)**
- **Dual curses of dimensionality/modeling**

KEY NDP IDEA

- Use an “approximate” reward function
- At the current state select decision that maximizes expected value of

  Current stage reward + Approximate reward of the next state
- Important issues:

  How to construct the approximate reward of a state - simulation is a key approach

  How to quantify the effects of approximation

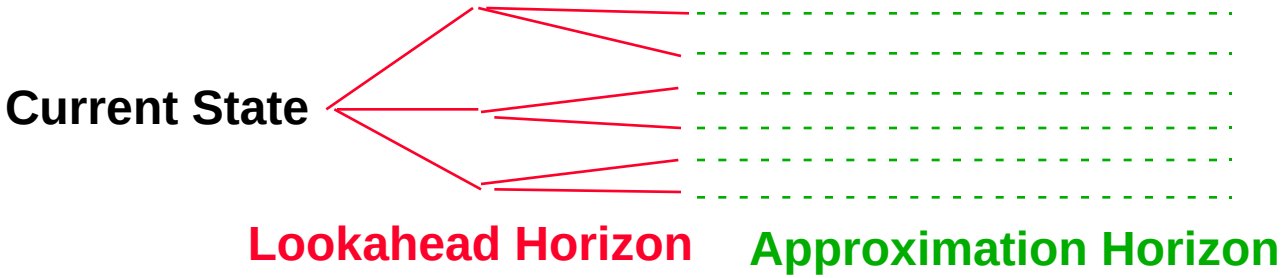
METHODS TO COMPUTE AN APPROXIMATE REWARD

- **Rollout algorithms (on-line)**
  - **Simulate the system under some (good heuristic) policy starting from the state of interest**
- **Parametric approximation algorithms (off-line)**
  - **Use a functional approximation to the optimal (DP-obtainable) reward function and select the weights of the approximation - connection with "neural networks"**
  - **Hand-tuning and trial and error**
  - **Systematic DP-related policy and value iteration methods (TD-Lambda, Q-learning, etc) based on repeated simulation of the sytem and forms of "least squares fit"**

SIMULATION AND LEARNING

- Simulation (learning by experience) to compute the (approximate) reward-to-go is a key distinctive aspect of NDP
- Important advantage: A detailed model of the system may not be necessary - a simulator can be used instead
- In case of a rollout algorithm: on-line learning is used (we learn only the reward values needed by on-line simulation)
- In case of parametric approximation: off-line learning

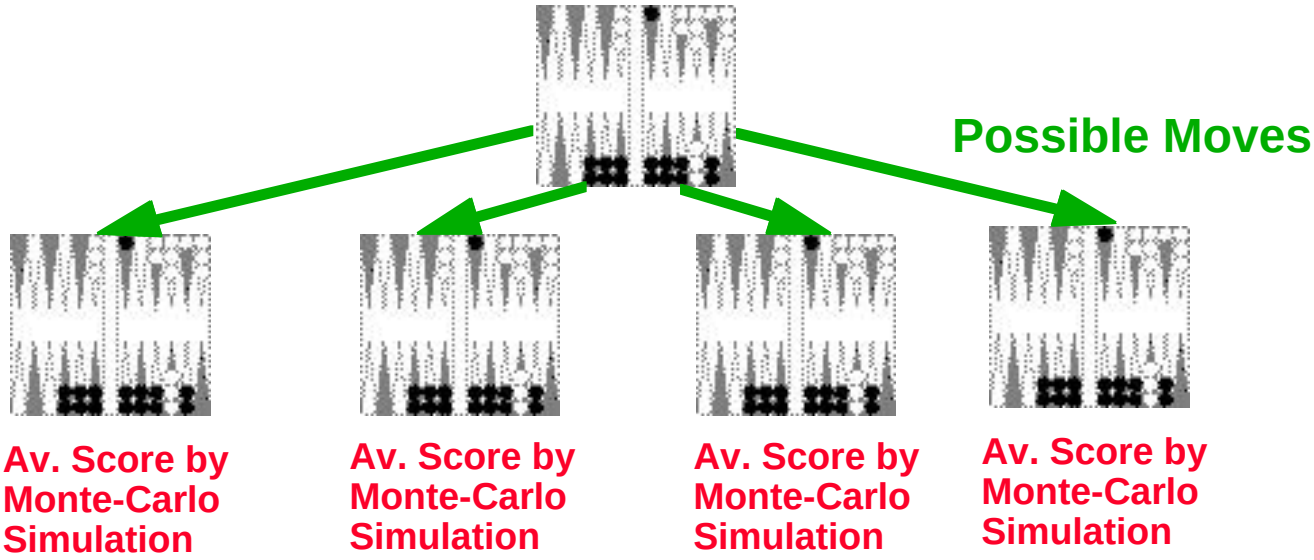
RELATION TO LIMITED LOOKAHEAD AND ROLLING HORIZON METHODS



- As in limited lookahead methods:
  - Search exhaustively within the lookahead horizon
  - Approximate reward-to-go within the approximation horizon
- As in rolling horizon methods:
  - The approximate reward-to-go may be based on a fixed number of steps after the lookahead horizon
- Connection with model predictive control

ROLLOUT POLICIES: BACKGAMMON PARADIGM

- On-line (approximate) reward-to-go calculation
- Simulation using a (simple) backgammon player (Tesauro 1996)
- Generic performance improvement result

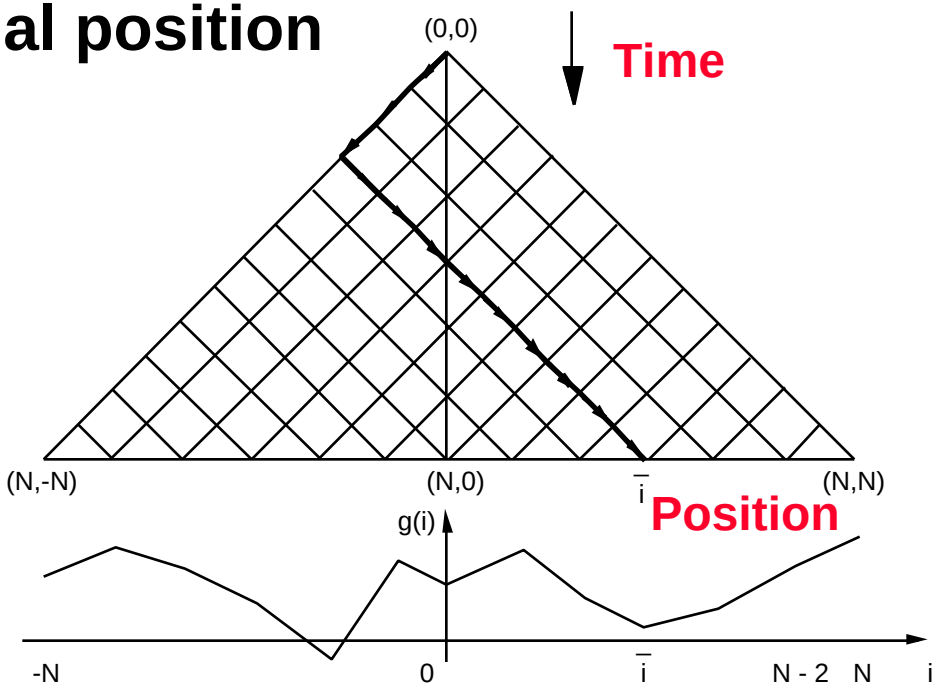


DETERMINISTIC PROBLEMS

- ONLY ONE simulation trajectory needed
- Use a heuristic for approximate reward-to-go calculation
  - At each state, consider all possible next states, and run the heuristic (once) from each
  - Select the next state with best heuristic reward
- Straightforward to implement
- Performance improvement result: Rollout algorithm improves on the performance of the heuristic (Bertsekas, Tsitsiklis, Wu, 1997)

EXAMPLE (ONE DIMENSIONAL WALK)

- N steps to left or right
- Heuristic: Always go right
- Optimize final position



ROLLOUT ALGORITHM PROPERTIES

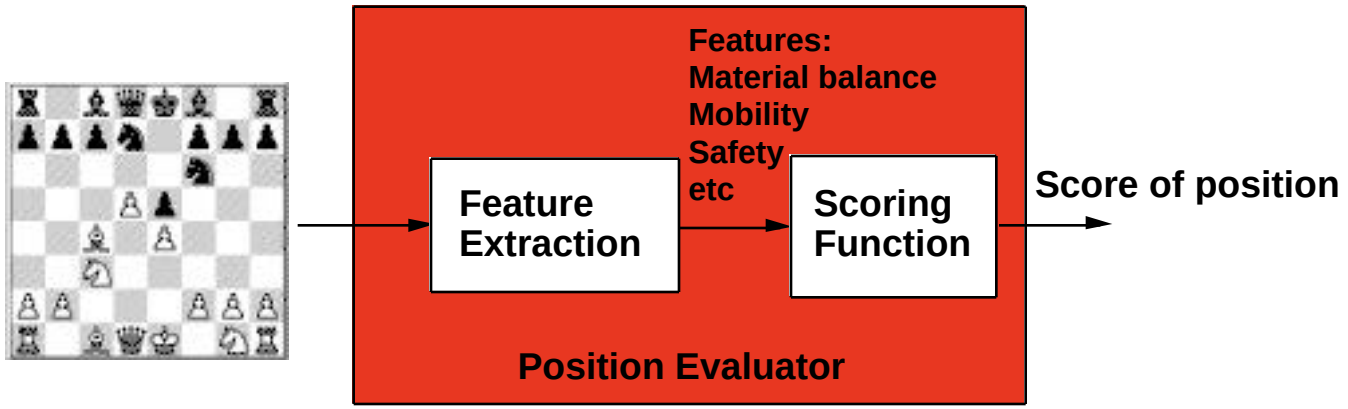
- Forward looking (the heuristic runs to the end)
- Self-correcting (the heuristic is reapplied at each time step)
- Suitable for on-line use
- Suitable for replanning
- Suitable for situations where the problem data are a priori unknown
- Substantial positive experience with many types of optimization problems, including combinatorial (e.g., scheduling)

STOCHASTIC PROBLEMS/ CERTAINTY EQUIV. APPROX

- Idea: Avoid Monte-Carlo
- Calculates score of next state assuming future unknown quantities are fixed at some typical values (a "certainty equivalence" scenario)
- Advantage : Single simulation run per next state (no averaging over stochastic uncertainty)
- Some loss of optimality
- Extension to multiple scenarios. Possibility of training (Bertsekas and Castanon, Bertsekas and Wu)

PARAMETRIC APPROXIMATION: CHESS PARADIGM

- Chess playing computer programs
- State = board position, Score of position: calculated in terms of the "important features" of the position, appropriately weighted



TRAINING

- In chess the weights of the architecture are "hand-tuned"
- In more sophisticated methods the weights are more systematically determined by using simulation-based training algorithms
- TD(λ), Q-Learning, λ-policy iteration, etc
- All of these methods are based on DP ideas of policy iteration and value iteration

POLICY IMPROVEMENT PRINCIPLE

- **Given a current policy, define a new policy as follows:**

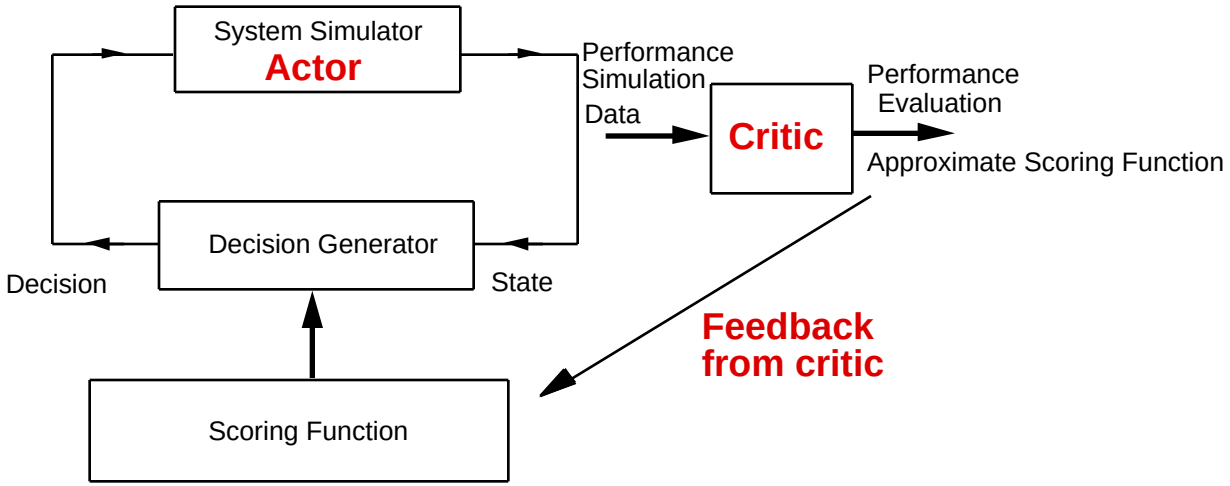
  At each state i maximize

  Current stage reward + Reward-to-go of current policy (starting from the next state)

- **Policy improvement result: New policy has improved performance over current policy**
- **If the reward-to-go is evaluated approximately, the improvement is "approximate"**

ACTOR/CRITIC SYSTEMS

- **Mechanism for policy improvement**
- **Actor implements current policy**
- **Critic evaluates the performance of the current policy, and passes feedback to the actor, who accordingly changes his policy**



FEATURES OF ACTOR-CRITIC SYSTEMS

- A lot of mathematical analysis, insight, and practical experience are now available
- Typically, a sequence of improved policies is obtained early on, and then the method oscillates around the optimal
- Training is far more challenging and time-consuming than in rollout algorithms
- On-line computation requirements are small
- Less suitable when problem data changes frequently