Topics in Reinforcement Learning: AlphaZero, ChatGPT, Neuro-Dynamic Programming, Model Predictive Control, Discrete Optimization, Applications

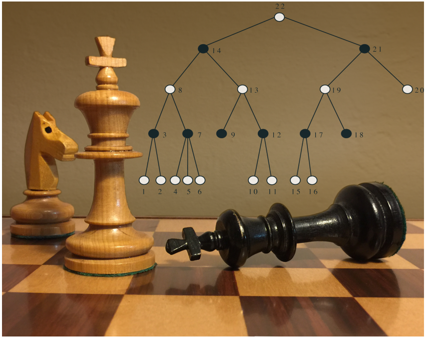
Arizona State University
Course CSE 691, Spring 2025

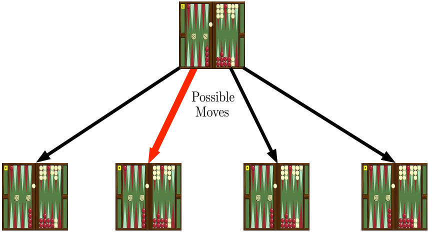
Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

Prof. Dimitri P. Bertsekas (dimitrib@mit.edu)
and
Dr Yuchao Li (yuchaoli@asu.edu)

Lecture 1
Introduction to Sequential Decision Making
(Reinforcement Learning, Dynamic Programming, Decision and Control)

Outline





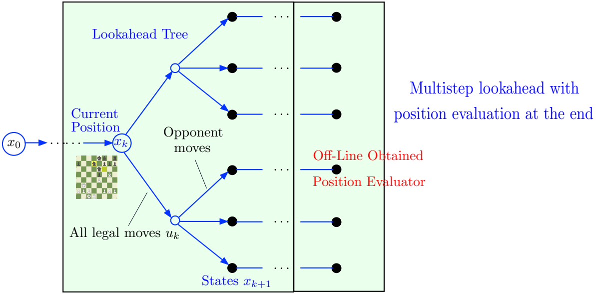
Both AlphaZero (2017) and TD-Gammon (1996) involve two algorithms:

- Off-line training of value and/or policy neural network approximations
- On-line play by multistep lookahead, rollout, and cost function approximation

Strong connections to DP, policy iteration, and RL-type methodology

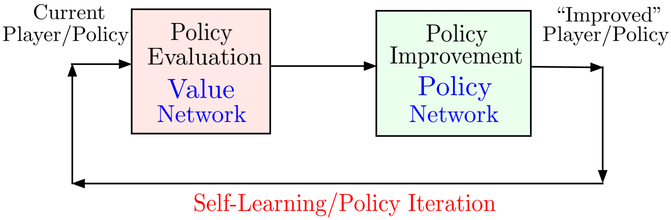
- We aim to understand this methodology, so it applies far more generally
- For example, in control system design (model predictive and adaptive control), large language models, and discrete optimization

On-Line Play in AlphaZero: Approximation in Value Space



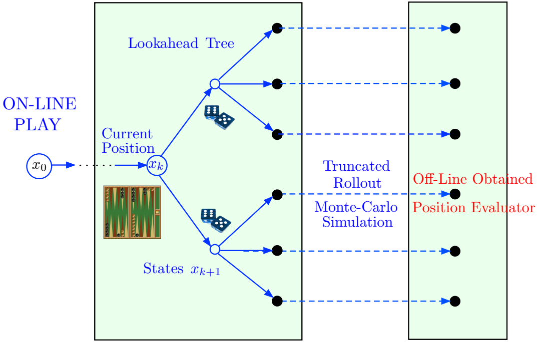
- Multistep lookahead with position evaluation at the end
- On-line play uses the result of off-line training, which is the position evaluator
- An example of a fundamental RL approach that we will use extensively in this course: Approximation in value space
- Strong similarities with Model Predictive Control (MPC) except that:
  - State is discrete in chess, but continuous in MPC (usually)
  - In chess the lookahead tree is usually “pruned", while in MPC the lookahead optimization is usually exact (more on this later)
  - Another difference is that chess is a two-player game. More on this later, but think of chess against a fixed opponent (this makes chess a one-player game)

Off-Line Training in AlphaZero: Approximate Policy Iteration (PI) Using Self-Generated Data



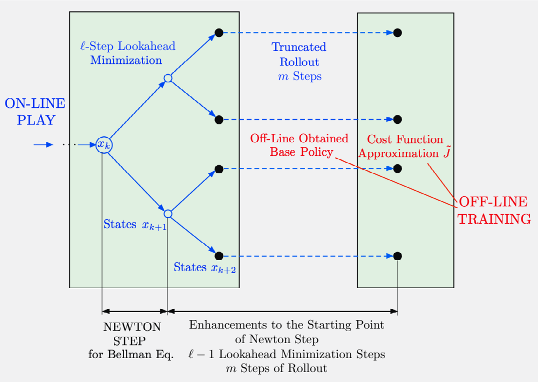
- The current player is used to train an improved player, and the process is repeated
- The current player/policy is "evaluated" by playing many games
- Its evaluation function is represented by a value neural net through training
- The "improved player" is represented by a policy neural net through training
- The "improvement" is done by using a form of approximate multistep lookahead minimization, called Monte-Carlo Tree Search (MCTS)
- The results of off-line training are the "final player" and its corresponding position evaluator

On-Line Play in TD-Gammon: Approximation in Value Space with Rollout



- On-line play uses the results of off-line training, which are: A position evaluator and a (base) player
- It aims to improve the base player by:
  - Generating forward a lookahead tree involving several moves
  - Simulating the base player for some more moves at the tree leaves (rollout)
  - Approximating the effect of future moves by using the terminal position evaluation
  - Calculating the “values" of the available moves at the root and playing the best move
- An important connection: Discrete optimization by rollout (using a base heuristic)

Some Major Empirical Observations



The AlphaZero on-line player plays much better than the off-line-trained player

TD-Gammon plays much better with truncated rollout than without rollout

We will aim for explanations, insights, and generalizations through abstract Bellman operators, visualization, and a deep theoretical insight:

Approximation in value space is connected to Newton's method

Framework of the Course

We aim to unify several areas of large scale computation:

- Reinforcement learning (RL) as practiced by the AI community
- Approximate dynamic programming (DP) as practiced by parts of the optimization/OR community
- Model predictive and adaptive control as practiced by the control systems community
- Parts of discrete optimization as practiced by the algorithms/CS community
- Parts of the emerging area of large language models as practiced by the LLM community

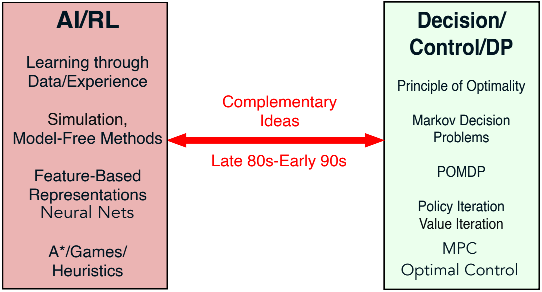
We rely on:

- The algorithmic theory of exact, approximate, and abstract DP
- The paradigm of AlphaZero/TD-Gammon and similar design architectures
- Intuitive visualization based on Bellman operators and Newton's method

We aim, through unification, to:

- Bridge the gap between cultures of different communities
- Bring to bear the power of RL to a very broad range of applications

Evolution of Approximate DP/RL: A Fruitful Synergy



Historical highlights

- Exact DP, optimal control (Bellman, Shannon, and others 1950s ...)
- AI/RL and Decision/Control/DP ideas meet (mid 80s-mid 90s)
- First major successes: Backgammon programs (Tesauro, 1992, 1996)
- Algorithmic progress, analysis, applications, first books (mid 90s ...)
- Machine Learning, BIG Data, Robotics, Deep Neural Networks (mid 2000s ...)
- AlphaGo and AlphaZero (DeepMind, 2016, 2017)
- Large Language Models, ChatGPT (OpenAI, 2022)

Approximate DP/RL Methodology is now Ambitious and Universal

Exact DP applies (in principle) to a very broad range of optimization problems

- Deterministic <—-> Stochastic
- Combinatorial optimization <—-> Optimal control w/ infinite state/control spaces
- One decision maker <—-> Two player games
- ... BUT is plagued by the curse of dimensionality and need for a math model

Approximate DP/RL overcomes the difficulties of exact DP by:

- Approximation (use neural nets and other architectures to reduce dimension)
- Simulation (use a computer model in place of a math model)

State of the art:

- Broadly applicable methodology: Can address a very broad range of challenging problems. Deterministic-stochastic-dynamic, discrete-continuous, games, etc
- There are no methods that are guaranteed to work for all or even most problems
- There are enough methods to try with a reasonable chance of success for most types of optimization problems
- Objective of our course: Structure mathematically the methodology, guide the art, delineate the sound ideas (from the crazy and unhinged ideas)

A Relevant Quotation from $\sim$ 30 Years Ago

From preface of Neuro-Dynamic Programming, Bertsekas and Tsitsiklis, 1996

A few years ago our curiosity was aroused by reports on new methods in reinforcement learning, a field that was developed primarily within the artificial intelligence community, starting a few decades ago. These methods were aiming to provide effective suboptimal solutions to complex problems of planning and sequential decision making under uncertainty, that for a long time were thought to be intractable.

Our first impression was that the new methods were ambitious, overly optimistic, and lacked firm foundation. Yet there were claims of impressive successes and indications of a solid core to the modern developments in reinforcement learning, suggesting that the correct approach to their understanding was through dynamic programming.

Three years later, after a lot of study, analysis, and experimentation, we believe that our initial impressions were largely correct. This is indeed an ambitious, often ad hoc, methodology, but for reasons that we now understand much better, it does have the potential of success with important and challenging problems.

This assessment still holds true!

Aims and References

This course is research-oriented. It aims:

- To explore the state of the art of approximate DP/RL at a graduate level
- To explore in depth some special research topics (rollout, policy iteration, etc)
- To provide the opportunity for you to explore research in the area

Main references:

- Bertsekas, Reinforcement Learning and Optimal Control, 2019
- Bertsekas, Rollout, Policy Iteration, and Distributed Reinforcement Learning, 2020
- Bertsekas, Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control, 2022 (on-line; focus on Newton step view of approximation in value space)
- Bertsekas: A Course in Reinforcement Learning, 2nd Ed., 2024 (text on-line)
- Slides, papers, and videos from the 2019-2023 ASU courses; check my web site

Supplementary references

- Exact DP: Bertsekas, DP and Optimal Control, Vols. I, II, 2017, Abstract DP 2022
- Bertsekas and Tsitsiklis, Neuro-Dynamic Programming, 1996
- Sutton and Barto, 1998, Reinforcement Learning (2nd edition 2018, on-line)
- Machine Learning/Deep Learning books (e.g., Bishop and Bishop, 2024)

Terminology in RL/AI and DP/Control

RL uses Max/Value, DP uses Min/Cost

- Reward of a stage = (Opposite of) Cost of a stage.
- State value = (Opposite of) State cost.
- Value (or state-value) function = (Opposite of) Cost function.

Controlled system terminology

- Agent = Decision maker or controller.
- Action = Decision or control.
- Environment = Dynamic system.

Methods terminology

- Learning = Solving a DP-related problem using simulation.
- Self-learning (or self-play in the context of games) = Solving a DP problem using simulation-based policy iteration.
- Planning vs Learning distinction = Solving a DP problem with model-based vs model-free simulation.

Notation in RL/AI and DP/Control

- RL poses problems as stochastic and uses transition probability notation

$$p(s, a, s')$$

(s, s' are states, a is action); standard in stochastic finite-state problems (MDP)

- Control theory uses discrete-time system equation

$$x_{k+1} = f(x_k, u_k, w_k)$$

which is standard in continuous spaces problems

- Operations research uses both notations [typically $p_{ij}(u)$ for transition probabilities]

These two notational systems are mathematically equivalent but:

- Transition probabilities are cumbersome for deterministic problems and continuous spaces problems
- System equations are cumbersome for finite-state problems

We will use both notational systems, depending on the context

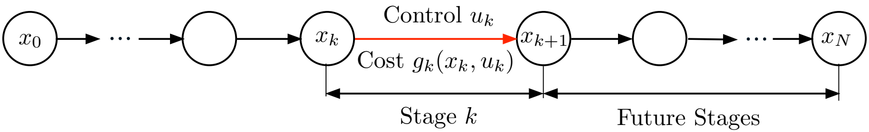
A Fifteen-Minute Break

All our lectures will have a 15-minute break, somewhere in the middle

Catch our breath and think about issues relating to the first half of the lecture.

A short discussion/questions/answers period will follow each break.

Finite Horizon Deterministic Optimal Control Model



- System

$$x_{k+1} = f_k(x_k, u_k), \qquad k = 0, 1, \ldots, N-1$$

where x_k: State (lives in some space), u_k: Control chosen from some set $U_k(x_k)$

- Cost function:

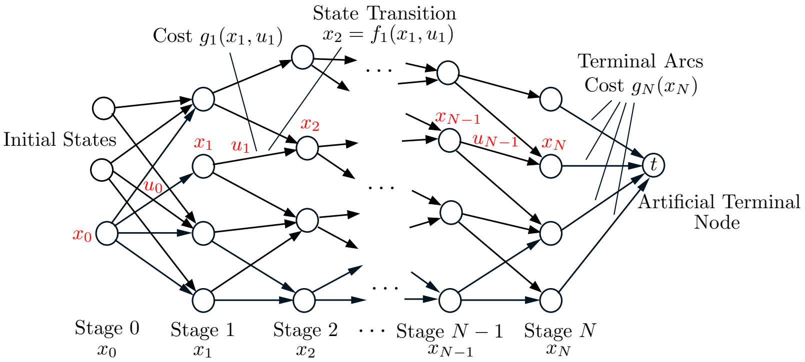
$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$$

- For given initial state x_0, minimize over control sequences $\{u_0, \ldots, u_{N-1}\}$

$$J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$$

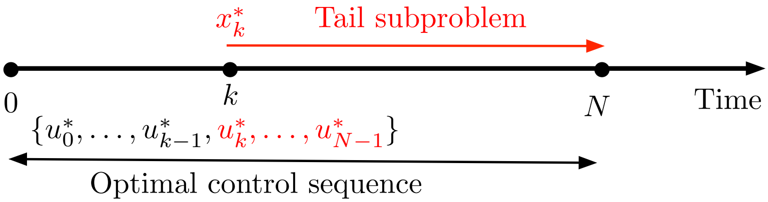
- Optimal cost function $J^*(x_0) = \min_{\substack{u_k \in U_k(x_k) \\ k=0,\ldots,N-1}} J(x_0; u_0, \ldots, u_{N-1})$

A Special Case: Finite Number of States and Controls



- Nodes correspond to states x_k
- Each arc corresponds to a state-control pair (x_k, u_k) [start node is x_k; end node is $x_{k+1} = f_k(x_k, u_k)$]
- Arcs (x_k, u_k) have cost $g_k(x_k, u_k)$ - "terminal arcs" have cost $g_N(x_N)$.
- The cost to optimize is the sum of the arc costs from the initial node/state x_0 to a terminal node t.
- The problem is equivalent to finding a minimum cost/shortest path from x_0 to t.

Principle of Optimality: A Very Simple Idea



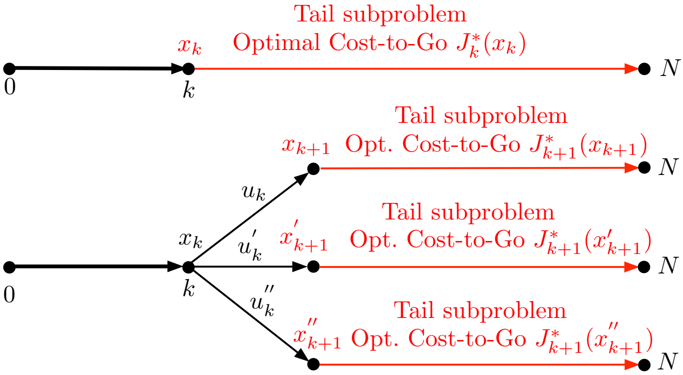
Principle of Optimality

THE TAIL OF AN OPTIMAL SEQUENCE IS OPTIMAL FOR THE TAIL SUBPROBLEM

Let $\{u_0^*, \ldots, u_{N-1}^*\}$ be an optimal control sequence with corresponding state sequence $\{x_1^*, \ldots, x_N^*\}$. Consider the tail subproblem that starts at x_k^* at time k and minimizes over $\{u_k, \ldots, u_{N-1}\}$ the "cost-to-go" from k to N,

$$g_k(x_k^*, u_k) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N).$$

Then the tail optimal control sequence $\{u_k^*, \ldots, u_{N-1}^*\}$ is optimal for the tail subproblem.



By the principle of optimality: To solve the tail subproblem that starts at x_k

- Consider every possible u_k and solve the tail subproblem that starts at next state $x_{k+1} = f_k(x_k, u_k)$. This gives the "cost evaluation of u_k"
- Optimize over all possible u_k

DP Algorithm: Solves All Tail Subproblems Efficiently by Using the Principle of Optimality

Idea of the DP algorithm

Solve all the tail subproblems of a given time length using the solution of all the tail subproblems of shorter time length

DP Algorithm: Produces the optimal costs $J_k^*(x_k)$ of the x_k-tail subproblems

Start with

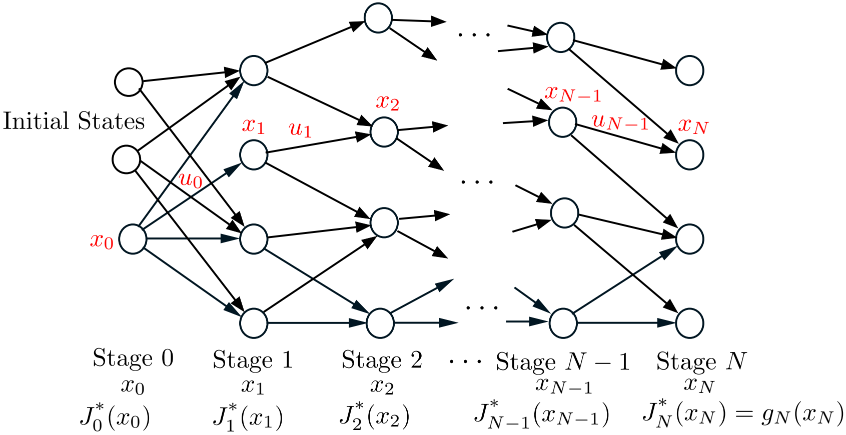
$$J_N^*(x_N) = g_N(x_N), \qquad \text{for all } x_N,$$

and for $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big], \qquad \text{for all } x_k.$$

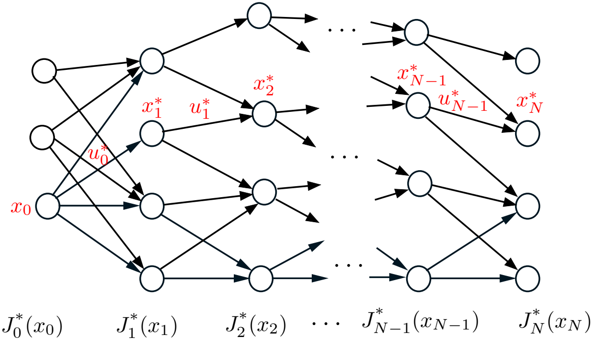
The optimal cost $J^*(x_0)$ is obtained at the last step

DP Algorithm for Generic Finite-State Problem. 1st Phase: Compute $J_k^*(x_k)$, the Optimal Costs-to-Go



$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big], \qquad \text{for all } x_k$$

2nd Phase: Construct the Optimal Control Sequence $\{u_0^*, \dots, u_{N-1}^*\}$



Start with

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + J_1^*\big(f_0(x_0, u_0)\big) \Big]$$

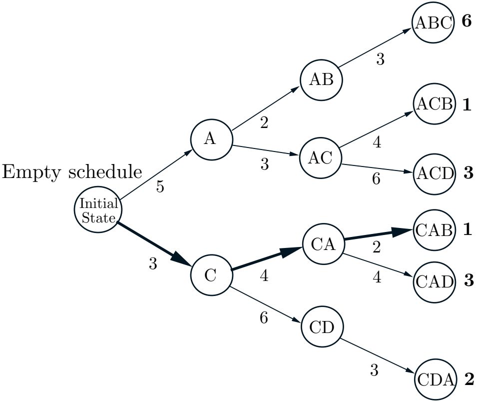
This takes you to

$$x_1^* = f_0(x_0, u_0^*).$$

Sequentially, going forward, for $k = 1, 2, \dots, N-1$, set

$$u_k^* \in \arg \min_{u_k \in U_k(x_k^*)} \Big[g_k(x_k^*, u_k) + J_{k+1}^*\big(f_k(x_k^*, u_k)\big) \Big], \qquad x_{k+1}^* = f_k(x_k^*, u_k^*).$$

Discrete-State Deterministic Scheduling Example

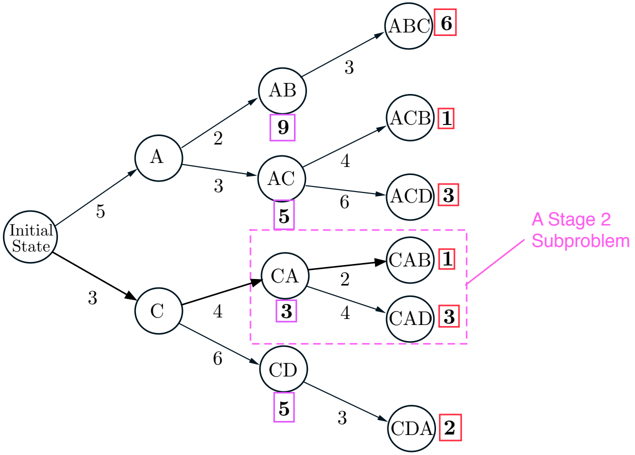


Find optimal sequence of operations A, B, C, D (A must precede B and C must precede D)

DP Problem Formulation

- States: Partial schedules; Controls: Stage 0, 1, and 2 decisions; Cost data shown along the arcs
- Recall the DP idea: Break down the problem into smaller pieces (tail subproblems)
- Start from the last decision and go backwards

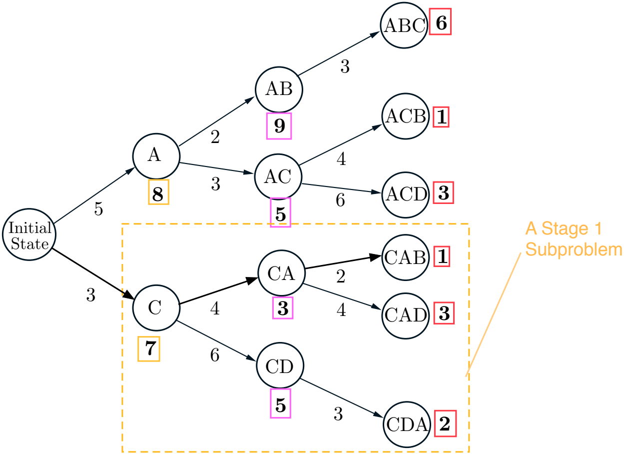
DP Algorithm: Stage 2 Tail Subproblems



Solve the stage 2 subproblems (using the terminal costs - in red)

At each state of stage 2, we record the optimal cost-to-go and the optimal decision

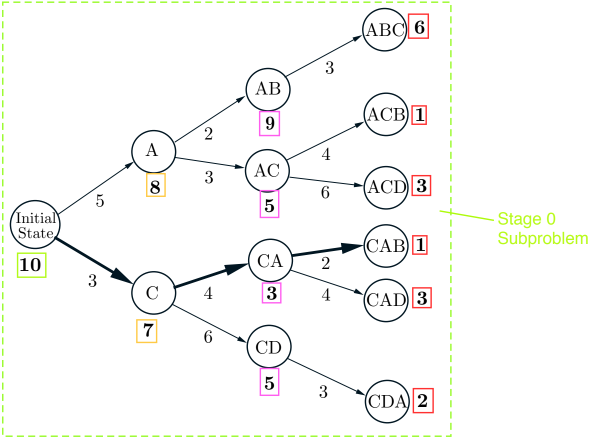
DP Algorithm: Stage 1 Tail Subproblems



Solve the stage 1 subproblems (using the optimal costs of stage 2 subproblems - in purple)

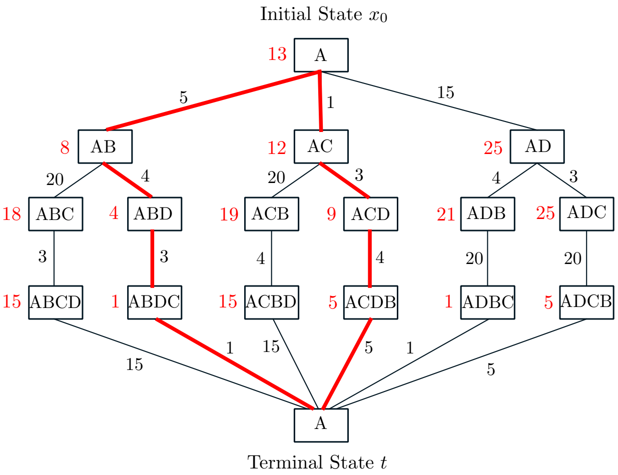
At each state of stage 1, we record the optimal cost-to-go and the optimal decision

DP Algorithm: Stage 0 Tail Subproblems



Solve the stage 0 subproblem (using the optimal costs of stage 1 subproblems - in orange)

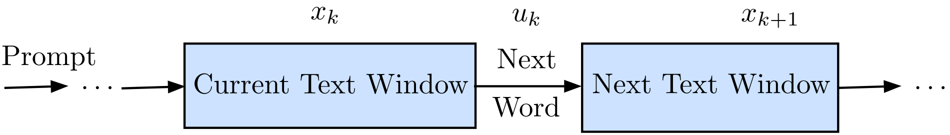
- The stage 0 subproblem is the entire problem
- The optimal value of the stage 0 subproblem is the optimal cost J^*(initial state)
- Construct the optimal sequence going forward



Matrix of Intercity Travel Costs

	5	1	15
5		20	4
1	20		3
15	4	3	

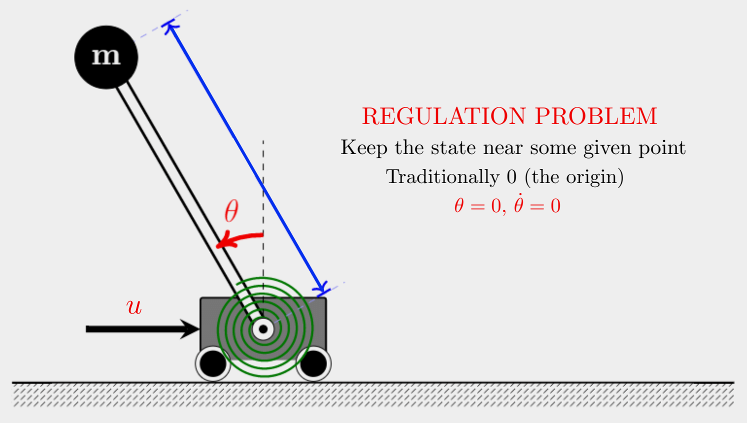
Large Language Models - Next Word Prediction (*n*-Gram - ChatGPT)



- x_{k+1} and x_k are n-word strings differing by the single word u_k
- System $x_{k+1} = f(x_k, u_k)$ (deterministic)
- Cost function: $g_N(x_N)$ (encodes the "quality" of the final text string)
- A trained GPT/NN can generate trajectories of such a system, i.e., state-control sequences $\{x_0, u_0, x_1, u_1, \ldots, u_{N-1}, x_N\}$.
- A GPT can be viewed as a heuristic/suboptimal control generation method (we will call it a "policy" or "base heuristic" in the next lecture).
- x_0 includes the user-supplied prompt - Possibility of "prompt engineering"

Exact DP will find the optimal GPT, but this is totally intractable!
The conceptual DP principles apply and can form the basis for approximations

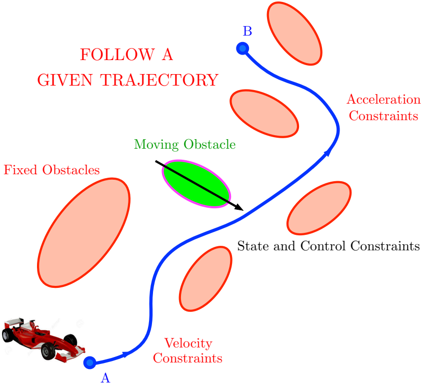
Classical Control Problem I: Control Around a Reference Point



Control theory has many applications:

Space exploration, chemical process control, robotics, self-driving cars

Classical Control Problem II: Path Planning



Example: Self-driving cars. Note the computational challenges:

- Unpredictable and changing environment
- Safety constraints
- Need for on-line replanning
- Tight on-line computational budget constraint
- Approximations are essential

Exact DP algorithm - Optimal control generation: Start with

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + J_1^*\big(f_0(x_0, u_0)\big) \Big]$$

This takes you to

$$x_1^* = f_0(x_0, u_0^*).$$

Sequentially, going forward, for $k = 1, 2, \ldots, N-1$, set

$$u_k^* \in \arg \min_{u_k \in U_k(x_k^*)} \Big[g_k(x_k^*, u_k) + J_{k+1}^*\big(f_k(x_k^*, u_k)\big) \Big], \qquad x_{k+1}^* = f_k(x_k^*, u_k^*).$$

Approximation in value space - Use some $\tilde{J}_k$ in place of J_k^* (off-line training)

Start with

$$\tilde{u}_0 \in \arg \min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + \tilde{J}_1\big(f_0(x_0, u_0)\big) \Big]$$

This takes you to

$$\tilde{x}_1 = f_0(x_0, \tilde{u}_0).$$

Sequentially, going forward, for $k = 1, 2, \ldots, N-1$, set (on-line play)

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{x}_k)} \Big[g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}\big(f_k(\tilde{x}_k, u_k)\big) \Big], \qquad \tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k).$$

Extensions

Stochastic finite horizon problems

The next state x_{k+1} is also affected by a random parameter (in addition to x_k and u_k). More difficult than deterministic (not equivalent to a shortest path problem).

Infinite horizon problems

The exact DP theory is mathematically more complex, but also more elegant.

Stochastic partial state information problems

We will convert them to problems of perfect state information, and then apply DP. Very hard to solve even approximately ... but offer great promise for applications.

Minimax/game problems

The exact DP theory is substantially more complex ... but the most spectacular successes of RL involve games. We will discuss RL methods and the context of computer chess in particular.

Course Aims and Requirements

Our principal aim: To help you to think about how RL applies to your research interests

Requirements:

- Homework (30%): A total of 3-4
- Research-oriented term paper (70%). A choice of:
  - A mini-research project. You may work in teams of 1-3 persons. You are encouraged to try. Selected class presentations at the end.
  - A read-and-report term paper based on 2-3 research publications (chosen by you in consultation with the instructors)
- Attendance in person is a requirement (assuming no hint of illness).

Notation: People in AI/RL, Control Theory, and Operations Research focus on different problems and use different notations

- AI/RL and OR focus on discrete/finite-state problems which are stochastic [Markovian Decision Problems (MDP)]. Use transition probabilities $p_{ij}(u)$ to describe the uncertainty.
- Control theorists use system equation notation $x_{k+1} = f_k(x_k, u_k, w_k)$. This notation is well-suited for continuous-state problems and deterministic problems.
- You are strongly encouraged to use the notation and terminology of the course.

Mathematical Requirements

Math requirements for this course are simple and modest

Calculus, elementary probability, minimal use of vector-matrix algebra. Our objective is to use math to the extent needed to develop insight into the mechanism of various methods, and to be able to start research.

However:

- A math framework is essential for DP problem formulation, understanding, and solution.
- DP relies on substantial math theory, particularly for infinite horizon problems.

Syllabus I (Approximate)

Algorithmic Topics

- Introduction to exact and approximate dynamic programming
- Approximation in value and policy space
- Off-line training, on-line play, and Newton's method
- Rollout and approximate policy iteration
- Model predictive and adaptive control
- Multiagent reinforcement learning
- Discrete optimization using rollout
- Sequential estimation and Bayesian optimization
- Training of feature-based approximation architectures and neural networks

Syllabus II (Approximate)

Application Topics

- Robotics and autonomous systems in multiagent environments
- Large language models
- Inference and optimization of Hidden Markov Models
- Data association
- Two-person games and computer chess
- Infrastructure networks and supply chains
- Cybersecurity applications
- Health care applications

Homework - Future Lectures

Homework due by Tuesday, January 21, midnight

Solve Exercise 1.1(a) of the textbook, ONLY PART (a)

Lectures

The first four lectures will aim to provide an introduction and overview of the subject, which will facilitate selecting and focusing on some research area. The remaining lectures will develop the topics listed above in greater depth.

In the 2nd lecture we will cover:

- DP algorithm for stochastic problems
- Approximation in value space

PLEASE READ AS MUCH OF THE TEXTBOOK AS YOU CAN

Watch the video of Lecture 2 of the 2024 or 2023 offering of the class at
http://web.mit.edu/dimitrib/www/RLbook.html