

Topics in Reinforcement Learning:
AlphaZero, ChatGPT, Neuro-Dynamic Programming,
Model Predictive Control, Discrete Optimization, Applications
Arizona State University
Course CSE 691, Spring 2025

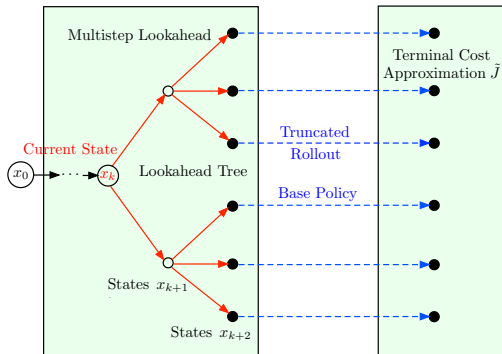
Links to Textbook, Videolectures, and Slides at
<http://web.mit.edu/dimitrib/www/RLbook.html>

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

Lecture 6
Deterministic Problems: Multistep Approximation in Value Space, Constrained
Rollout, Multiagent Rollout

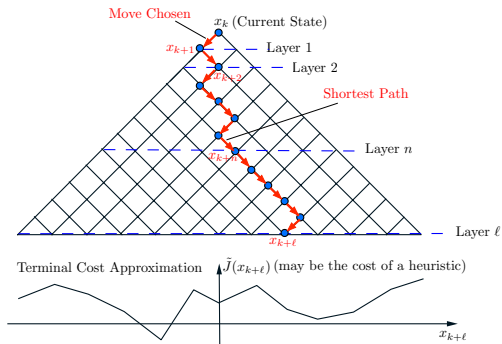
- 1 Deterministic Problems: Approximation in Value Space with Multistep Lookahead
- 2 Constrained Rollout for Deterministic Optimal Control
- 3 Multiagent Problems

Multistep Approximation in Value Space - The General Case



- Special case 1: **No rollout**. The general multistep approximation in value space scheme.
- Special case 2: **Pure multistep rollout**. No terminal cost and no truncation.
- WE TAKE IT AS FACT: **Longer lookahead improves performance** (but is costly).
- OUR STRATEGY: **Extend the lookahead as much as the comp. budget allows**.
- **One idea**: Truncated rollout (a cheap extension of the lookahead length).
- **Another computation-saving idea**: Selectively prune the lookahead tree.

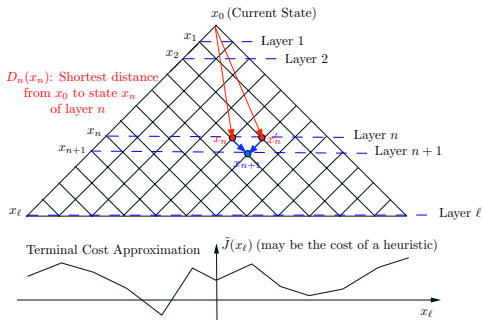
Multistep Lookahead in Deterministic Problems - No Rollout



We obtain a trajectory $\{x_k, x_{k+1}, \dots, x_{k+\ell}\}$ that minimizes the distance from x_k to $x_{k+\ell}$ PLUS $\tilde{J}(x_{k+\ell})$. We then use the first move $x_k \rightarrow x_{k+1}$.

- The shortest path problems from x_k to all $x_{k+\ell}$ can be solved by **backward DP** (start from layer ℓ go towards x_k).
- An important alternative is the **forward DP** algorithm.
- It is the same as the backwards DP algorithm with the **direction of the arcs reversed** (start from x_k go towards layer ℓ - see the next slide).

Forward DP Algorithm and Iterative Deepening [$\tilde{J}(x)$ is given for all x]

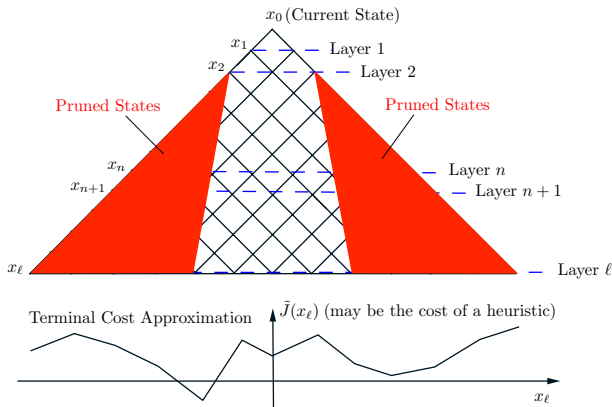


- The “forward” DP algorithm: The shortest distances $D_{n+1}(x_{n+1})$ to layer $n+1$ states are obtained from the shortest distances $D_n(x_n)$ to layer n states as follows:

$$D_{n+1}(x_{n+1}) = \min_{x_n} \left[(\text{Cost } x_n \rightarrow x_{n+1}) + D_n(x_n) \right]$$

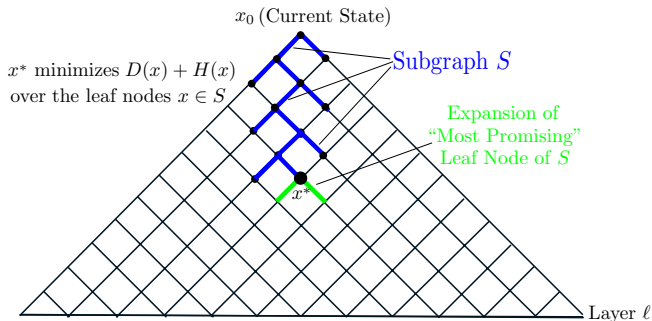
- Solution of the ℓ -step lookahead problem: The shortest path to the state x_ℓ^* of layer ℓ that minimizes $D_\ell(x_\ell) + \tilde{J}(x_\ell)$.
- Iterative deepening: Solve the n -step lookahead problem before solving the $(n+1)$ -step lookahead problem.
- This is an “anytime” algorithm (returns a feasible solution even if it is interrupted).

Iterative Deepening with Tree Pruning



- Iterative deepening can be “enhanced” by pruning states \hat{x}_n such that the n -step lookahead cost $D_n(\hat{x}_n) + \tilde{J}(\hat{x}_n)$ is “far from the minimum” over x_n .
- We prune as we go: Prune states in layer n before pruning states in layer $n + 1$.
- Runs the risk of overpruning: Some pruned states may be “good” in hindsight.
- Should we go back and check for overpruning? How?

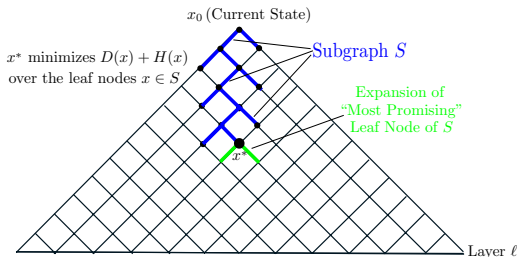
Incremental Multistep Rollout - Flexible Pruning/Iterative Deepening



We use a less regular graph, which is expanded at each iteration based on a shortest path computation

- At the start of an iteration, we have an **acyclic connected subgraph S** rooted at x_0 .
- **We compute the shortest distance $D(x)$ from x_0 to all $x \in S$** , going through S .
- **We find the "most promising" leaf node $x^* \in S$** , i.e., one that minimizes $D(x) + H(x)$, where $H(x)$ is a "heuristic distance" from x to layer ℓ (some form of rollout?).
- **Expand x^* to enlarge S and start the next iteration** (or stop if x^* is in layer ℓ).

Incremental Multistep Rollout - Some Details

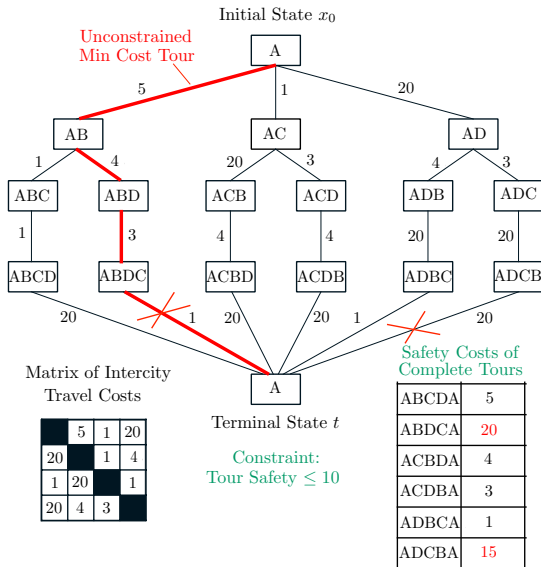


- At the start of an iteration, we have an **acyclic connected subgraph S** rooted at x_0 .
- We minimize $D(x) + H(x)$ over all leaf nodes $x \in S$.
- We expand the "most promising" minimizing node x^* to form the new subgraph.
- The computation of the shortest distances $D(x)$ is done progressively with the **forward DP algorithm** as the subgraph S expands.
- Example of $H(x)$: The **cost of a base heuristic** that starts from x and ends at some node x_ℓ of layer ℓ , plus $\tilde{J}(x_\ell)$, plus **an extra term that favors paths with few hops that encourages backtracking** e.g., $\delta \cdot (\text{number of hops from } x_0 \text{ to } x)$, where $\delta > 0$.
- For $\delta = 0$, we get **max pruning**: S ends up being "long and skinny". For $\delta \approx \infty$, we get **min pruning**: S ends up being as "fat" as possible.

Applies to problems with **additional** constraints on the entire optimal trajectory

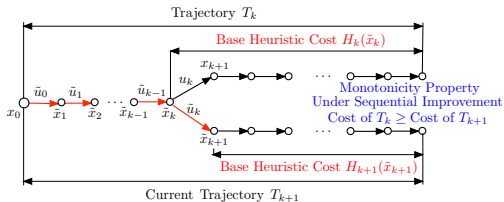
- **Greatly expands the range of applications of rollout**
- For example it applies to intractable discrete optimization problems (e.g., shortest path problems with a limit on the number of hops).
- It is similar to unconstrained rollout: As we expand the rollout path, **we exclude from consideration the Q-factors that correspond to constraint violation.**
- **Guarantees cost improvement over the base heuristic** under appropriate conditions (modified versions of sequential consistency, sequential improvement, or use of a fortified version).

Traveling Salesman: Example of a Trajectory Constraint



Find a minimum cost tour subject to a safety constraint

Deterministic Rollout with Trajectory Constraint $T \in C$: Basic Idea



Review of the unconstrained rollout algorithm:

- Construct sequence of trajectories $\{T_0, T_1, \dots, T_N\}$ with monotonically nonincreasing cost (assuming a sequential improvement condition).
- For each k , the trajectories T_k, T_{k+1}, \dots, T_N share the same initial portion $(x_0, \tilde{u}_0, \dots, \tilde{u}_{k-1}, \tilde{x}_k)$.
- **The base heuristic is used to generate candidate trajectories** that correspond to the controls $u_k \in U_k(x_k)$.
- The next trajectory T_{k+1} is the candidate trajectory that has min cost.

To deal with a trajectory constraint $T \in C$, **we discard all the candidate trajectories that violate the constraint, and we choose T_{k+1} to be the best of the remaining trajectories.**

Deterministic Problems with Trajectory Constraints: Definition and Reformulation

- Consider a deterministic optimal control problem with system $x_{k+1} = f_k(x_k, u_k)$.
- A complete trajectory is a sequence

$$T = (x_0, u_0, x_1, u_1, \dots, u_{N-1}, x_N)$$

- Problem is to $\min_{T \in C} G(T)$ where G is a given cost function and C is a given constraint set of trajectories.

For a constrained problem, x_k cannot serve as state, so we use state augmentation

- Redefine the state to be the partial trajectory

$$y_k = (x_0, u_0, x_1, \dots, u_{k-1}, x_k)$$

- Partial trajectory evolves according to a redefined system equation:

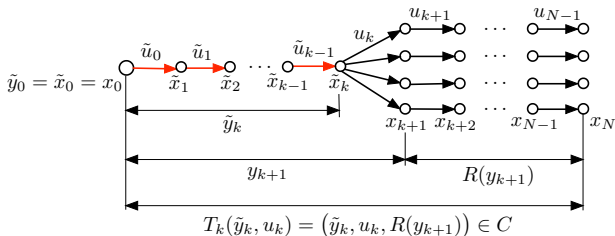
$$y_{k+1} = (y_k, u_k, f_k(x_k, u_k))$$

- The problem becomes to minimize $G(y_N)$ subject to the constraint $y_N \in C$.

A Fifteen-Minute Break

Catch our breath and think about issues relating to the first half of the lecture.
Ask questions when you return.

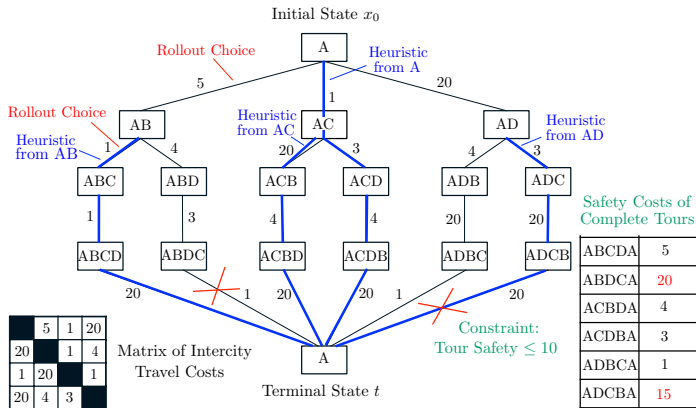
Rollout Algorithm - Partial Trajectory-Dependent Base Heuristic



- Given $\tilde{y}_k = \{\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \tilde{u}_1, \dots, \tilde{u}_{k-1}, \tilde{x}_k\}$ consider all controls u_k and corresponding next states x_{k+1} .
- Extend \tilde{y}_k to obtain the partial trajectories $y_{k+1} = (\tilde{y}_k, u_k, x_{k+1})$, for $u_k \in U_k(x_k)$.
- Run the base heuristic from each y_{k+1} to obtain the partial trajectory $R(y_{k+1})$.
- Join the partial trajectories y_{k+1} and $R(y_{k+1})$ to obtain complete trajectories denoted by $T_k(\tilde{y}_k, u_k) = (\tilde{y}_k, u_k, R(y_{k+1}))$
- Find the set of controls $\tilde{U}_k(\tilde{y}_k)$ for which $T_k(\tilde{y}_k, u_k)$ is feasible, i.e., $T_k(\tilde{y}_k, u_k) \in C$
- Choose the control $\tilde{u}_k \in \tilde{U}_k(\tilde{y}_k)$ according to the minimization

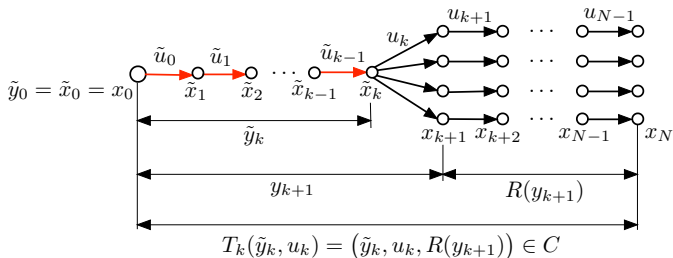
$$\tilde{u}_k \in \arg \min_{u_k \in \tilde{U}_k(\tilde{y}_k)} G(T_k(\tilde{y}_k, u_k))$$

Constrained Traveling Salesman Example



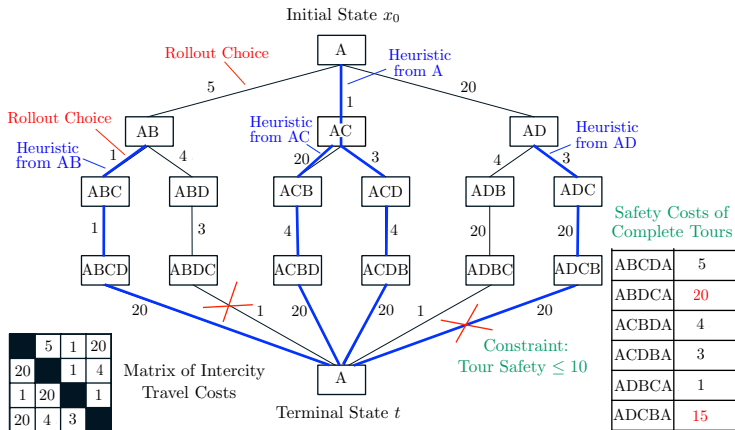
- Rollout at A:** Considers partial tours AB, AC, and AD; Obtains the complete tours ABCDA, ACBDA, and ADCBA; **Discards ADCBA as being infeasible**; Compares ABCDA and ACBDA, finds ABCDA to have smaller cost, and selects AB.
- Rollout at AB:** Considers the partial tours ABC and ABD; Obtains the complete tours ABCDA and ABDCA; **Discards ABDCA as being infeasible**; Selects the complete tour ABCDA.

Constrained Rollout Algorithm Properties



- The notions of **sequential consistency** and **sequential improvement** apply. Their definition includes that the set of “feasible” controls $\tilde{U}_k(\tilde{y}_k)$ is nonempty for all k .
- **Sequential improvement condition**: The min heuristic Q-factor over $\tilde{U}_k(\tilde{y}_k)$ is no larger than the heuristic cost at \tilde{y}_k (see the “Course in RL” textbook).
- **Fortified version** (if sequential improvement does not hold; see the notes):
 - ▶ **Maintains the “tentative best” trajectory**, and follows it up to generating a better trajectory through rollout.
 - ▶ **Has the cost improvement property**, assuming the base heuristic generates a feasible trajectory starting from the initial condition $\tilde{y}_0 = x_0$.
- **Multiagent version**: Selects one-control-component-at-a-time (apply constrained rollout to the equivalent reformulation, i.e., the one with control space “unfolded”).

Example of Sequential Consistency and Sequential Improvement



- The heuristic is **not sequentially consistent at A**, but it is sequentially improving.
- If we change the $D \rightarrow A$ cost to 25, the heuristic is **not sequentially improving at A**, and the cost improvement property is lost.
- If we change the $D \rightarrow A$ cost to 25 and we add fortification, **the rollout algorithm at A sticks with the initial tentative best trajectory ACDBA**, and rejects ABCDA.

Structural components

- (1) **Trajectories T** consisting of a sequence of decisions defined by a layered/optimal control graph
- (2) **A cost function $G(T)$** to rank trajectories
- (3) **A constraint $T \in C$** to determine feasibility of trajectories
- (4) **A base heuristic** that starts from a partial trajectory and generates a complete trajectory

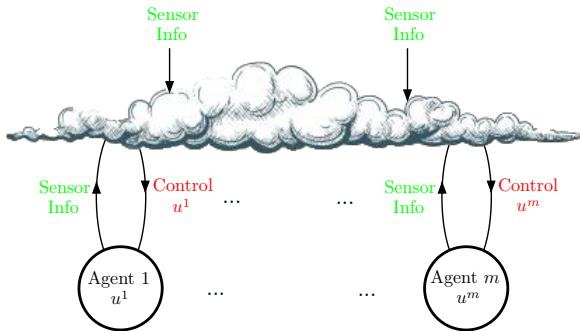
Given (1)

The choices of (2), (3), and (4) are independent of each other

In particular, given (1)-(3):

We can try several different base heuristics or a superheuristic

Multiagent Problems: Review



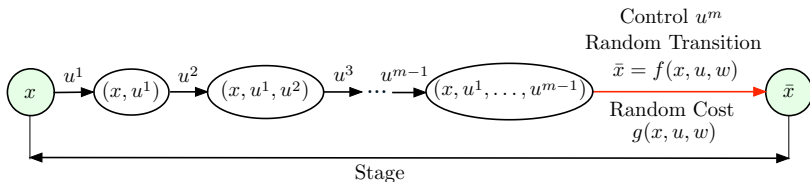
Classical information pattern

At each time: Agents have exact state info; choose their controls as function of state

Model: A discrete-time (possibly stochastic) system with state x and control u

- Decision/control has m components $u = (u^1, \dots, u^m)$ corresponding to m "agents"
- "Agents" is just a metaphor - the important math structure is $u = (u^1, \dots, u^m)$
- We will reformulate the problem so that rollout can be done much faster

Reformulation Idea: Trading off Control and State Complexity (B+T NDP Book, 1996)



An equivalent reformulation - "Unfolding" the control action

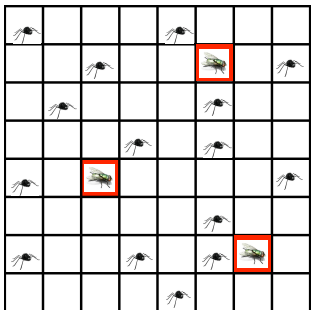
- The control space is simplified at the expense of $m - 1$ additional layers of states, and corresponding $m - 1$ cost functions

$$J^1(x, u^1), J^2(x, u^1, u^2), \dots, J^{m-1}(x, u^1, \dots, u^{m-1})$$

- **Allows far more efficient rollout (one-agent-at-a-time).** This is just standard rollout for the reformulated problem (so it involves a Newton step)
- The increase in size of the state space does not adversely affect rollout (only one state and its successors are looked at each stage during on-line play)
- Complexity reduction: **The one-step lookahead branching factor is reduced from n^m to $n \cdot m$,** where n is the number of possible choices for each component u^i

Spiders-and-Flies Example

(e.g., Vehicle Routing, Maintenance, Search-and-Rescue, Firefighting)



15 spiders move in 4 directions with perfect vision

3 blind flies move randomly

Objective is to

Catch the flies in minimum time

- In the original problem, at each time we must consider $\approx 5^{15}$ joint moves
- In the reformulated problem, **we break down the control into a sequence of one-spider-at-a-time moves**
- Thus, we need to consider only $5 \cdot 15 = 75$ (while maintaining the rollout cost improvement property)
- For more discussion, including illustrative videos of spiders-and-flies problems, see <https://www.youtube.com/watch?v=eqbb6vVIN38&t=1654s> Also Section 2.9 of the course textbook

Will be presented in class

Will be presented in class

Will be presented in class

In the next lecture we will cover:

- Applications of approximation in value space.
- Multiagent path finding.
- Multiobject tracking, data association.