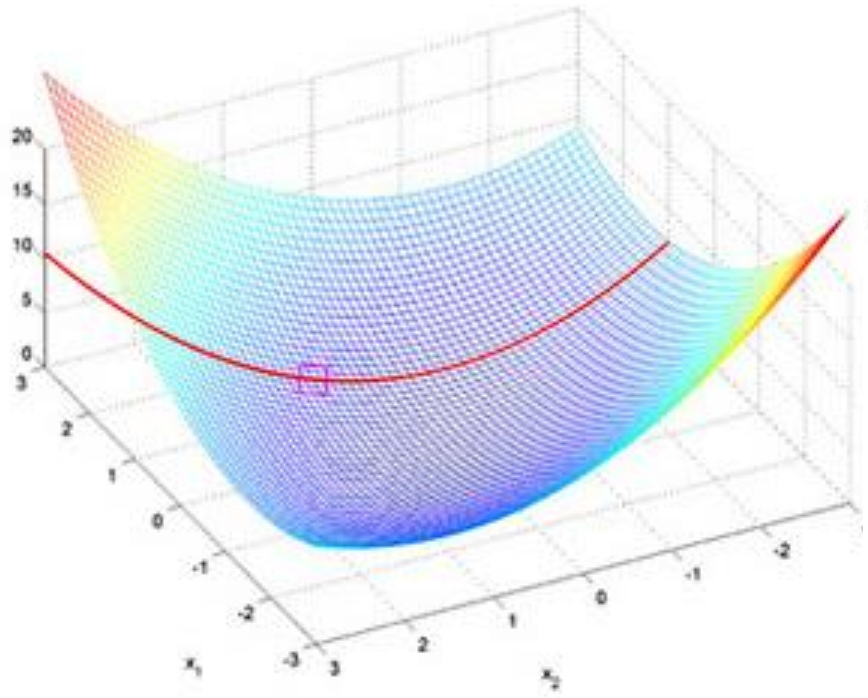
MEGR 3090/7090/8090: Advanced Optimal Control

UNC CHARLOTTE



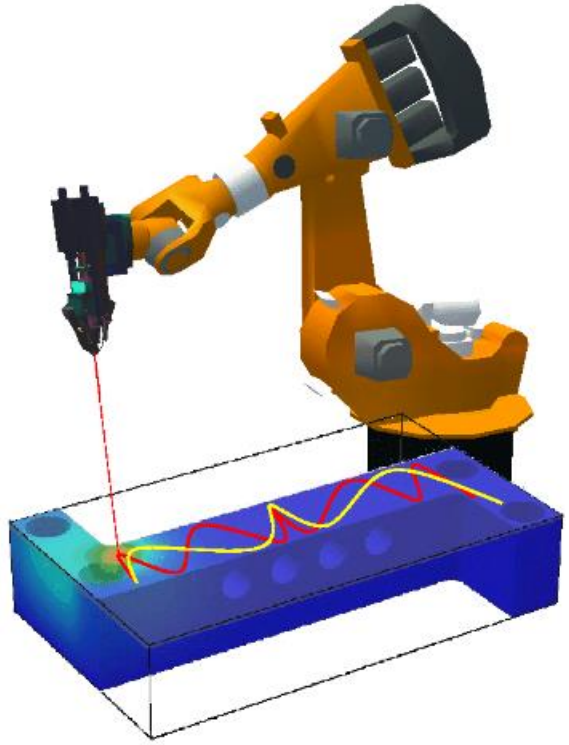
$$V_n(\mathbf{x}_n) = \min_{\{\mathbf{u}_n, \mathbf{u}_{n+1}, \cdots, \mathbf{u}_{N-1}\}} \left[\frac{1}{2} \sum_{k=n}^{N-1} \left(\mathbf{x}_k^T \mathbf{Q}_k \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right) + \frac{1}{2} \mathbf{x}_N^T \mathbf{Q}_N \mathbf{x}_N \right]$$

$$V_n(\mathbf{x}_n) = \min_{\{\mathbf{u}_n, \mathbf{u}_{n+1}, \cdots, \mathbf{u}_{N-1}\}} \left[\frac{1}{2} \sum_{k=n}^{N-1} \left(\mathbf{x}_k^T \mathbf{Q}_k \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right) + \frac{1}{2} \mathbf{x}_N^T \mathbf{Q}_N \mathbf{x}_N \right]$$

$$= \min_{\mathbf{u}_n} \left[\frac{1}{2} \left(\mathbf{x}_n^T \mathbf{Q}_n \mathbf{x}_n + \mathbf{u}_n^T \mathbf{R} \mathbf{u}_n \right) + \underbrace{\min_{\{\mathbf{u}_{n+1}, \cdots, \mathbf{u}_{N-1}\}} \left[\frac{1}{2} \sum_{k=n+1}^{N-1} \left(\mathbf{x}_k^T \mathbf{Q}_k \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right) + \frac{1}{2} \mathbf{x}_N^T \mathbf{Q}_N \mathbf{x}_N \right]}_{V_{n+1}(\mathbf{x}_{n+1})} \right]$$

$$= \min_{\mathbf{u}_n} \left[\frac{1}{2} \left(\mathbf{x}_n^T \mathbf{Q}_n \mathbf{x}_n + \mathbf{u}_n^T \mathbf{R} \mathbf{u}_n \right) + V_{n+1}(\mathbf{x}_{n+1}) \right]$$

$$V_n(\mathbf{x}_n) = \min_{\mathbf{u}_n} \left[\frac{1}{2} \left(\mathbf{x}_n^T \mathbf{Q}_n \mathbf{x}_n + \mathbf{u}_n^T \mathbf{R} \mathbf{u}_n \right) + V_{n+1}(\mathbf{x}_{n+1}) \right]$$



Lecture 14

October 5, 2017

Review – Convex Optimization Formulations

UNC CHARLOTTE

Linear program (LP): Minimize: $J(\mathbf{u}) = \boldsymbol{k}^T\mathbf{u}$ Subject to: $A_1\mathbf{u} - \mathbf{b}_1 \leq \mathbf{0}$, $A_2\mathbf{u} - \mathbf{b}_2 = \mathbf{0}$

- LP solution – simplex method (linprog in MATLAB – see lecture 10)

Quadratic program (QP): Minimize: $J(\mathbf{u}) = \mathbf{u}^T Q\mathbf{u} + R\mathbf{u}$ Subject to: $A_1\mathbf{u} - \mathbf{b}_1 \leq \mathbf{0}$, $A_2\mathbf{u} - \mathbf{b}_2 = \mathbf{0}$

- QP solution – interior point method (quadprog in MATLAB – see lecture 11)

Sequential quadratic programming (SQP):

- A ***local*** solution technique to ***general nonlinear programming*** (NLP) problems:

$$\text{Minimize } J(\mathbf{u}) \qquad \text{Subject to: } g(\mathbf{u}) \leq \mathbf{0},\ h(\mathbf{u}) = \mathbf{0}$$

- 3 steps: (i) Set up a QP subproblem via a Taylor series approximation of the Lagrangian; (ii) Solve the QP subproblem; (iii) Move in the direction of the QP problem's solution
- Can be performed via fmincon in MATLAB (see lecture 13)

Convex Optimization Tools – Merits and Drawbacks

UNC CHARLOTTE

Merits of convex optimization tools:

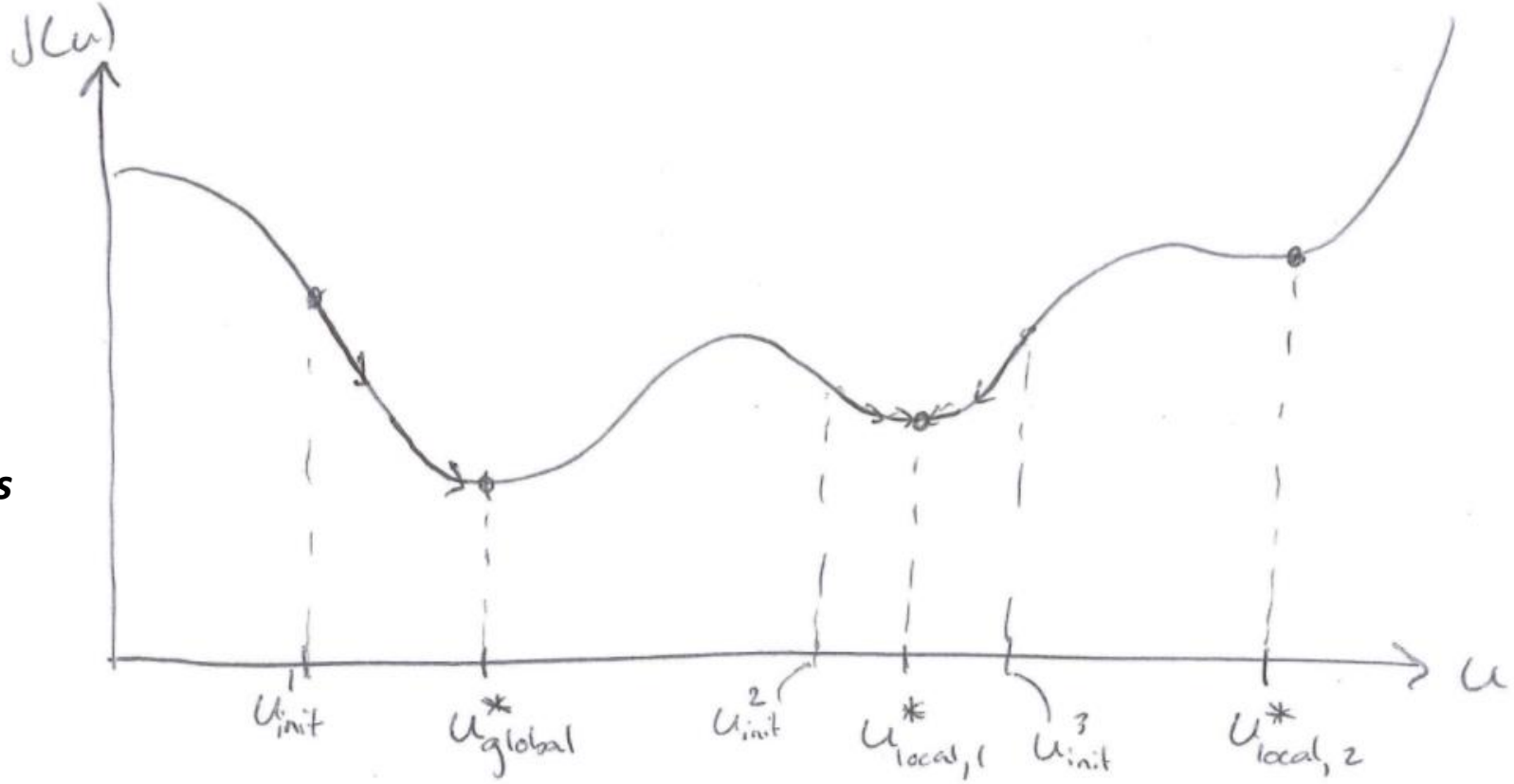
- ***Computational efficiency*** – Convex optimization tools typically result in fast solutions relative to other tools we are about to learn about
- ***Easy setup*** – Off-the-shelf solvers are widely available…just collect all of your control signals in a vector, and encode the objective function and constraints

Drawback of convex optimization tools:

- ***Main drawback*** – ***locality of the solution***: If the optimization problem is not globally convex, then the solution determined by convex techniques will be a ***local*** one
- ***Minor drawback*** – We are solving for a ***sequence*** of control decisions using ***design optimization*** tools, which don't leverage the fact that the control decisions are implemented in a particular order
    - In special (restrictive) cases, such as the linear quadratic regulator (LQR), accounting explicitly for the sequence can lead to a closed-form solution

Convex Optimization Tools – Merits and Drawbacks

A **multi-start strategy** can be used to ***alleviate*** some of the issues associated with the locality of SQP, but it ***does not guarantee convergence to a globally optimal solution, regardless of how many initial guesses are tried***.



Multi-start strategy:

1) Choose multiple u_{init}

2) Perform SQP for each u_{init}

Global Optimization Techniques – Exhaustive Search

UNC CHARLOTTE

Optimization problem: Minimize $J(\mathbf{u})$ Subject to: $g(\mathbf{u}) \leq \mathbf{0}$

$h(\mathbf{u}) = \mathbf{0}$

where: $\mathbf{u} = [u_0 \quad \dots \quad u_{N-1}]^T$, $u_i \in \{v_1, \dots, v_p\}, i = 0 \dots N-1$

Each control (decision) variable is ***quantized*** – It must assume one of p discrete values

Exhaustive search process:

- Evaluate $J(\mathbf{u})$ and verify constraint satisfaction for ***every possible combination*** of $u_0 \dots u_{N-1}$
- Questions: How many candidate control sequences must be considered in computing the optimal control sequence? How does this number depend on the horizon length and number of quantization levels?
- Note: The exhaustive search ***does not distinguish*** between design optimization and control optimization!

Global Optimization Techniques – Exhaustive Search

UNC CHARLOTTE

Quantization refers to the process of assigning/allocating a finite number of values to a variable.

$u(i) \in \{-2, -1, 0, 1, 2\} \quad \forall i$

5 levels

Suppose $u(i) \in \{v_1, \ldots, v_p\}$, $i = 0 \ldots N-1$

$\Rightarrow p^N$ combinations of $\underline{u}$

Exhaustive Search – Optimal Control Example

UNC CHARLOTTE

System dynamics: $x(k+1) = 2x(k) + 0.5u(k)$

Optimization problem: Minimize $J(\mathbf{u}, x(0)) = \sum_{i=0}^{4} 5x(i+1)^2 + u(i)^2$

Subject to: $-1 \leq x(i) \leq 1, i = 1 \dots 5$

where $x(0) = 1$ and $u(i) \in \{-3, -2, -1, 0, 1, 2, 3\}, i = 0 \dots 4$

- Determine the optimal control sequence, $\mathbf{u}^*$…Sample code available on Canvas
- How many (total) cost function evaluations are required? How would this change if the horizon length were increased to 10?

Exhaustive Search – Optimal Control Example

UNC CHARLOTTE

Suppose $u(i) \in \{v_1, \ldots, v_p\}$, $i = 0 \ldots N-1$

$\Rightarrow p^N$ combinations of $\underline{u}$

$u(i) \in \underbrace{\{-3, -2, -1, 0, 1, 2, 3\}}_{7 \text{ levels}}$, $N = 5$

$\Rightarrow$ # of total cost fn. evals. $= 7^5 = 16,807$

If $N = 10$, # of fn. evals. $= 282,475,249$

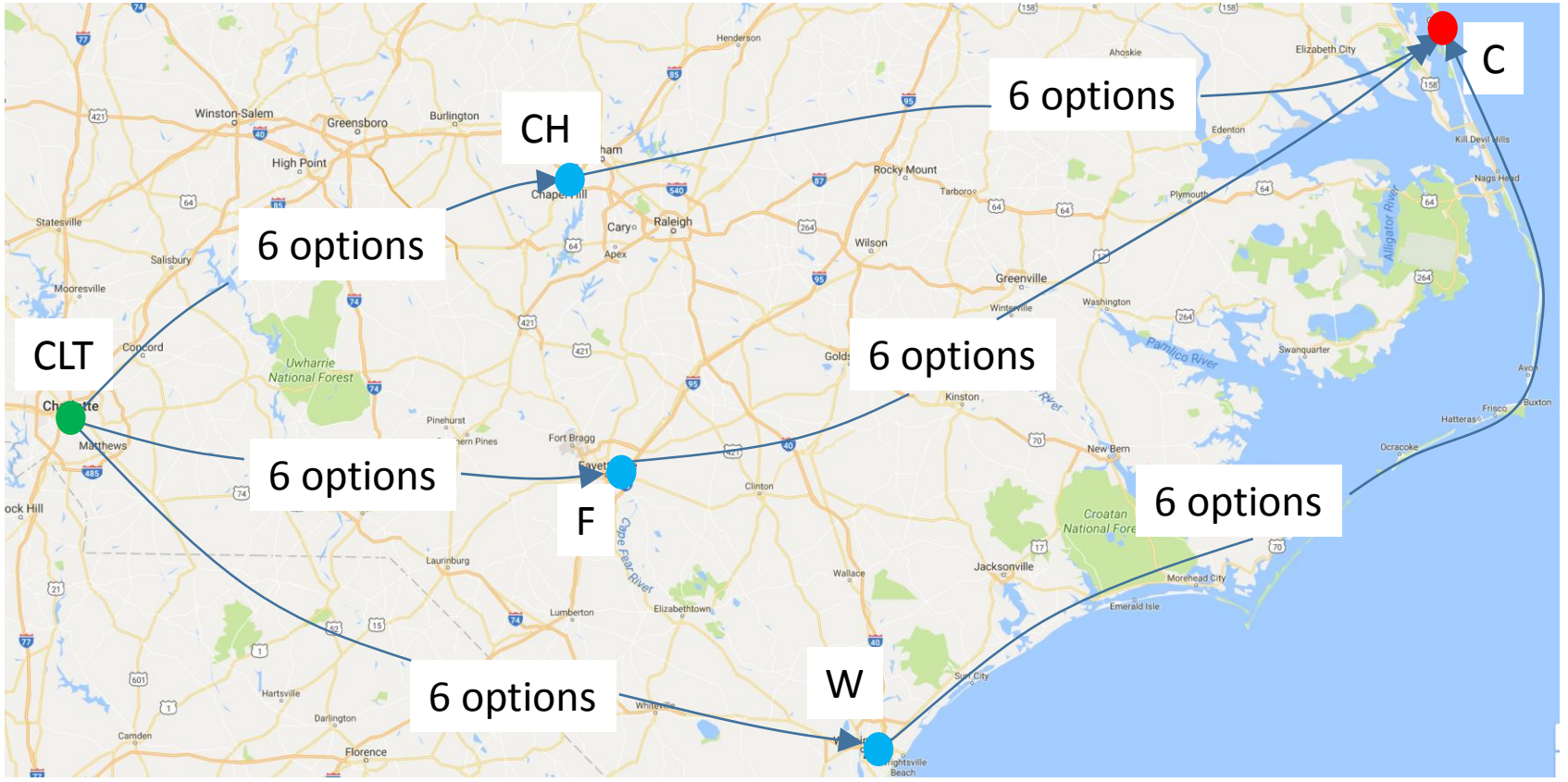
The computational requirement does not scale well with N!

Another Optimal Control Example

Consider an optimal routing problem, where you are planning a road trip from Charlotte (CLT) to Corolla (C) in the Outer Banks

- You want to stop in either Chapel Hill (CH), Fayetteville (F), or Wilmington (W) along the way

- For any choice of intermediate point, there are 6 routes to choose from
- Your goal is to minimize fuel consumption – you know the fuel consumption, u_{AB}^{i}, for each candidate route segment (i) that connects generic point A to generic point B.



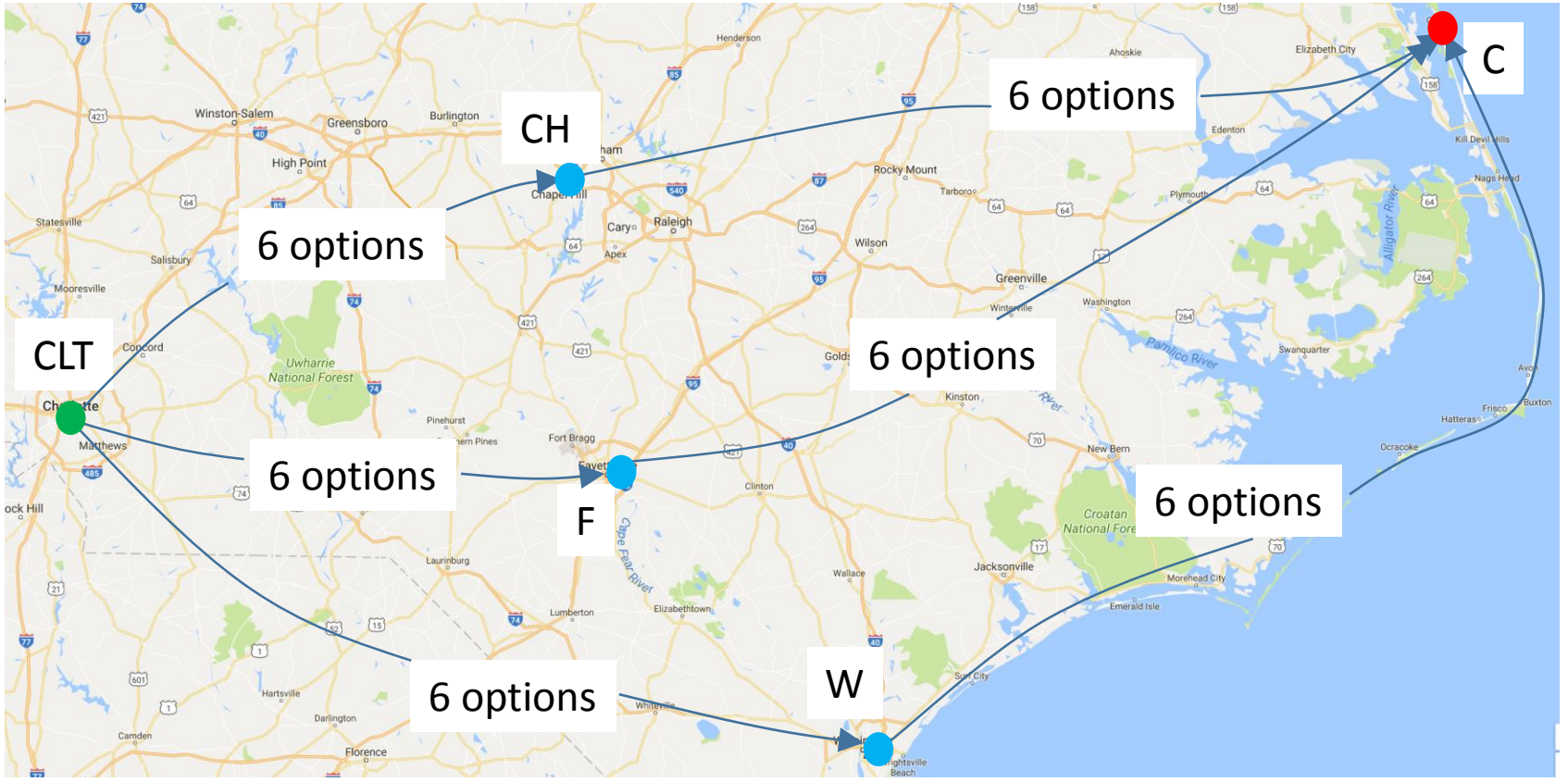
Another Optimal Control Example

Answer the following questions:

- How many control sequences (combinations of control inputs) must be considered in an exhaustive search? This corresponds to the **total number of cost function evaluations required**

- Is there a more computationally efficient solution available for finding the optimal route?

- How many cost function evaluations are required by the simpler approach?



Another Optimal Control Example

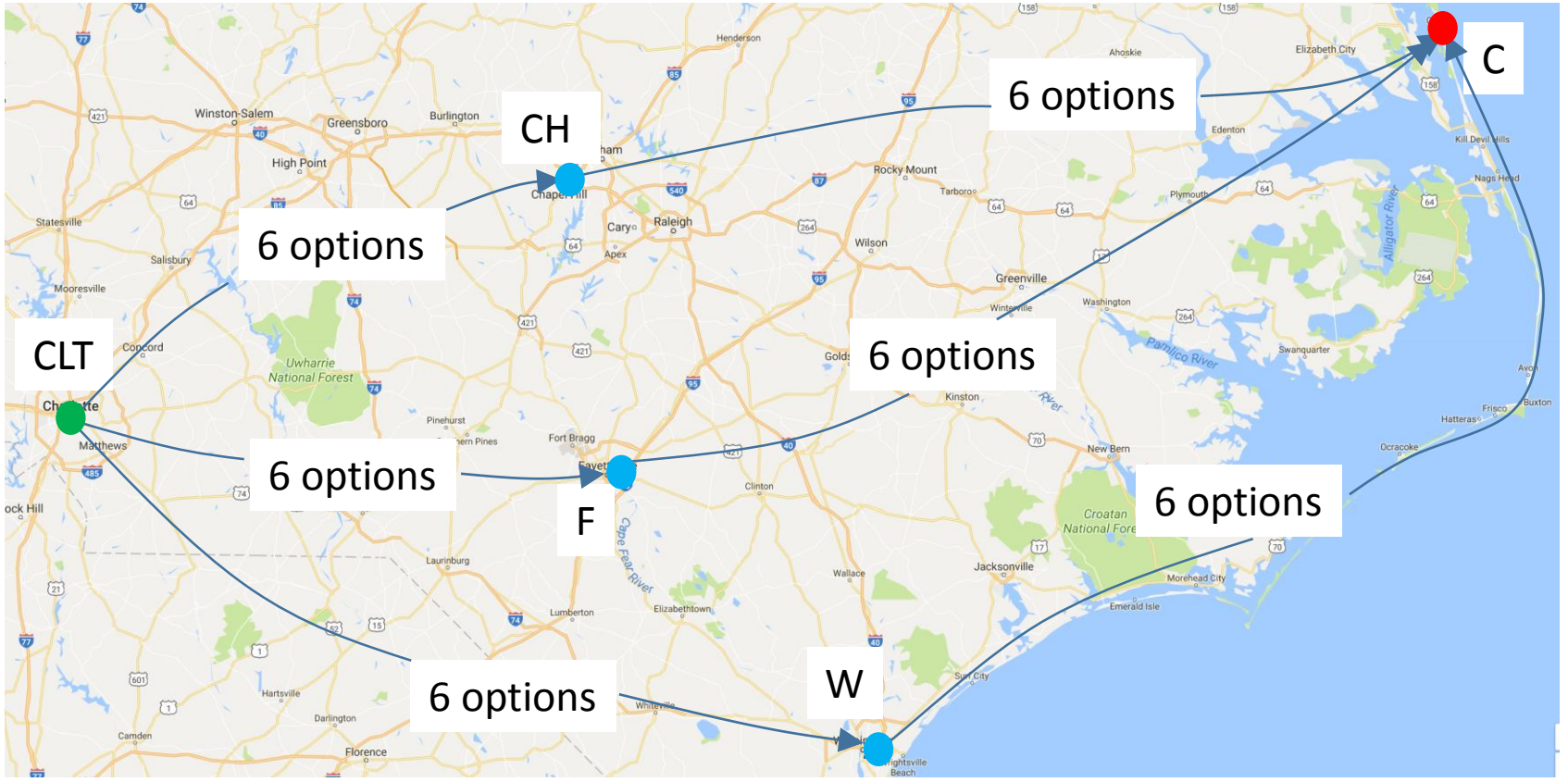
UNC CHARLOTTE

Answer the following questions:

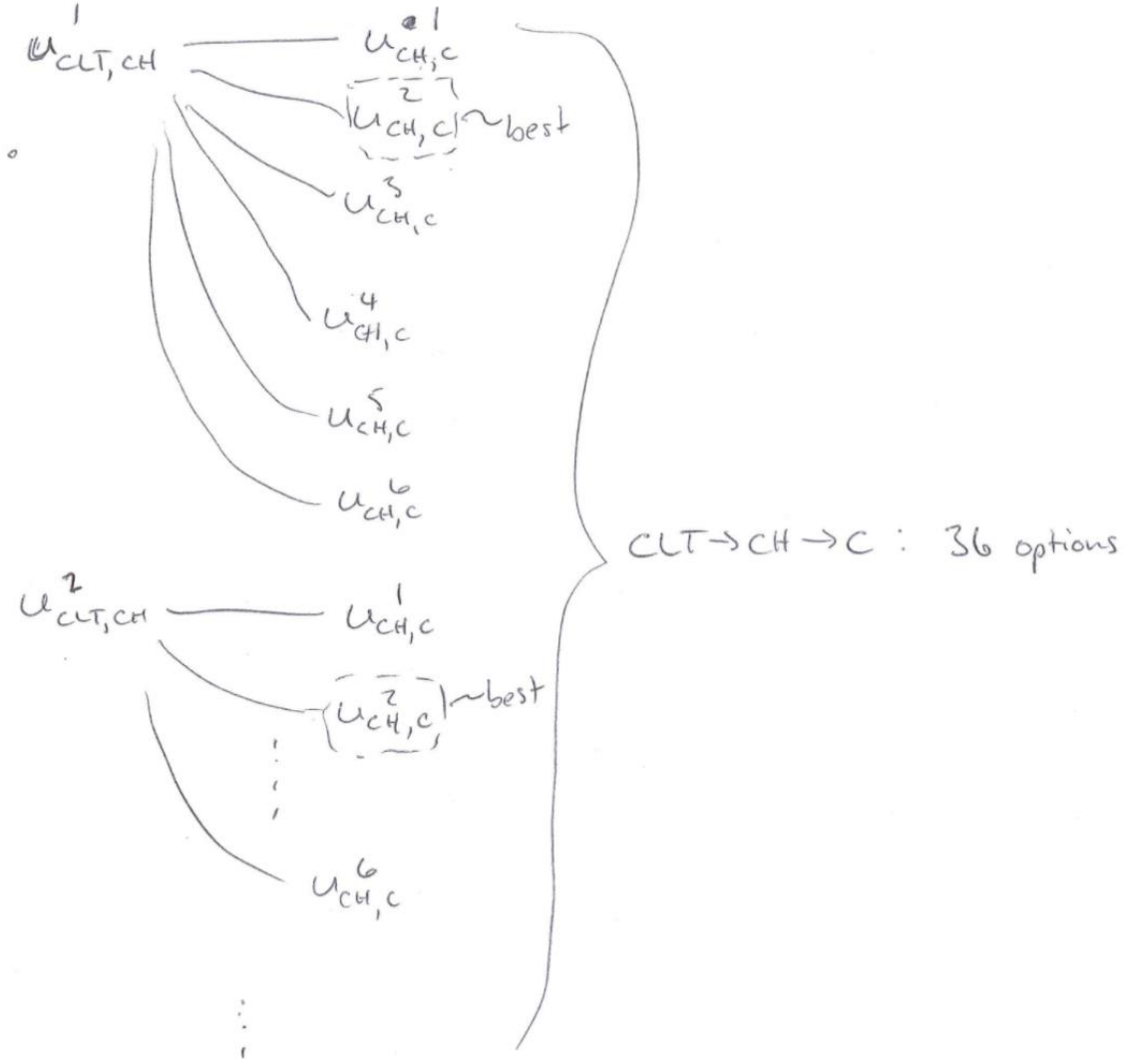
- How many control sequences (combinations of control inputs) must be considered in an exhaustive search? $3 \times 6^2 = 108$. This corresponds to the **total number of cost function evaluations required**

- Is there a more computationally efficient solution available for finding the optimal route? Yes.

- How many cost function evaluations are required by the simpler approach? We're about to find out.



Another Optimal Control Example

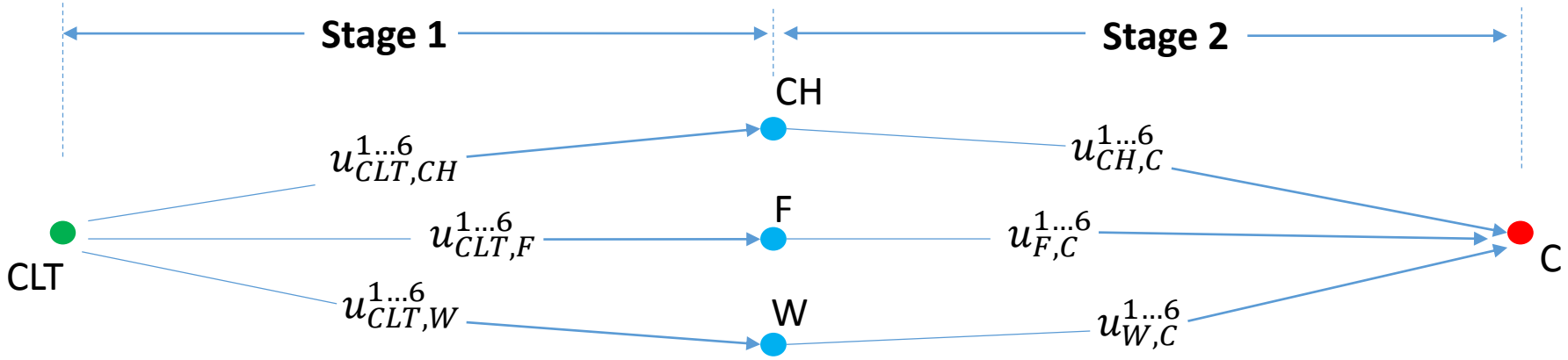


Another Optimal Control Example – Better Solution via *Backward Recursion*

UNC CHARLOTTE

Key observations:

- The trip can be divided into 2 **stages**, which occur in **sequence** (stage 1 happens before stage 2)
- After a control signal is applied at each **stage**, the system arrives at a **state** (in this case, the ***states*** are ***cities***, which can take on values of CLT, CH, F, W, or C)

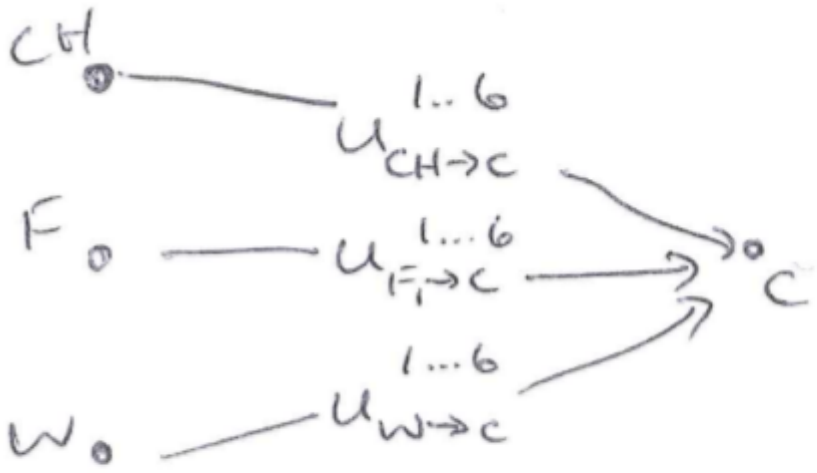


Main idea of backward recursion:

- Compute all ***stage costs*** for the last stage ($J(u_{CH,C}^{1\ldots6}), J(u_{F,C}^{1\ldots6}), J(u_{W,C}^{1\ldots6})$) – **18 cost function evaluations**
- For each originating state at the last stage (CH, F, W), determine the control input that minimizes the cost. Denote the corresponding minimum cost from each origin state as $J_{2,CH}^*, J_{2,F}^*, J_{2,W}^*$
- Move to stage 1. Compute all ***stage costs*** for this stage ($J(u_{CLT,CH}^{1\ldots6}), J(u_{CLT,F}^{1\ldots6}), J(u_{CLT,W}^{1\ldots6})$), and add these to $J_{2,CH}^*, J_{2,F}^*, J_{2,W}^*$, respectively – **18 cost function evaluations**. The lowest resulting cost (and corresponding control sequence) is the global optimum (and optimizer)!

Another Optimal Control Example – Better Solution via *Backward Recursion*

Stage 2:



Need: $u^*_{CH\to C}$, $J^*_{2,CH}$ – 6 fn. evals

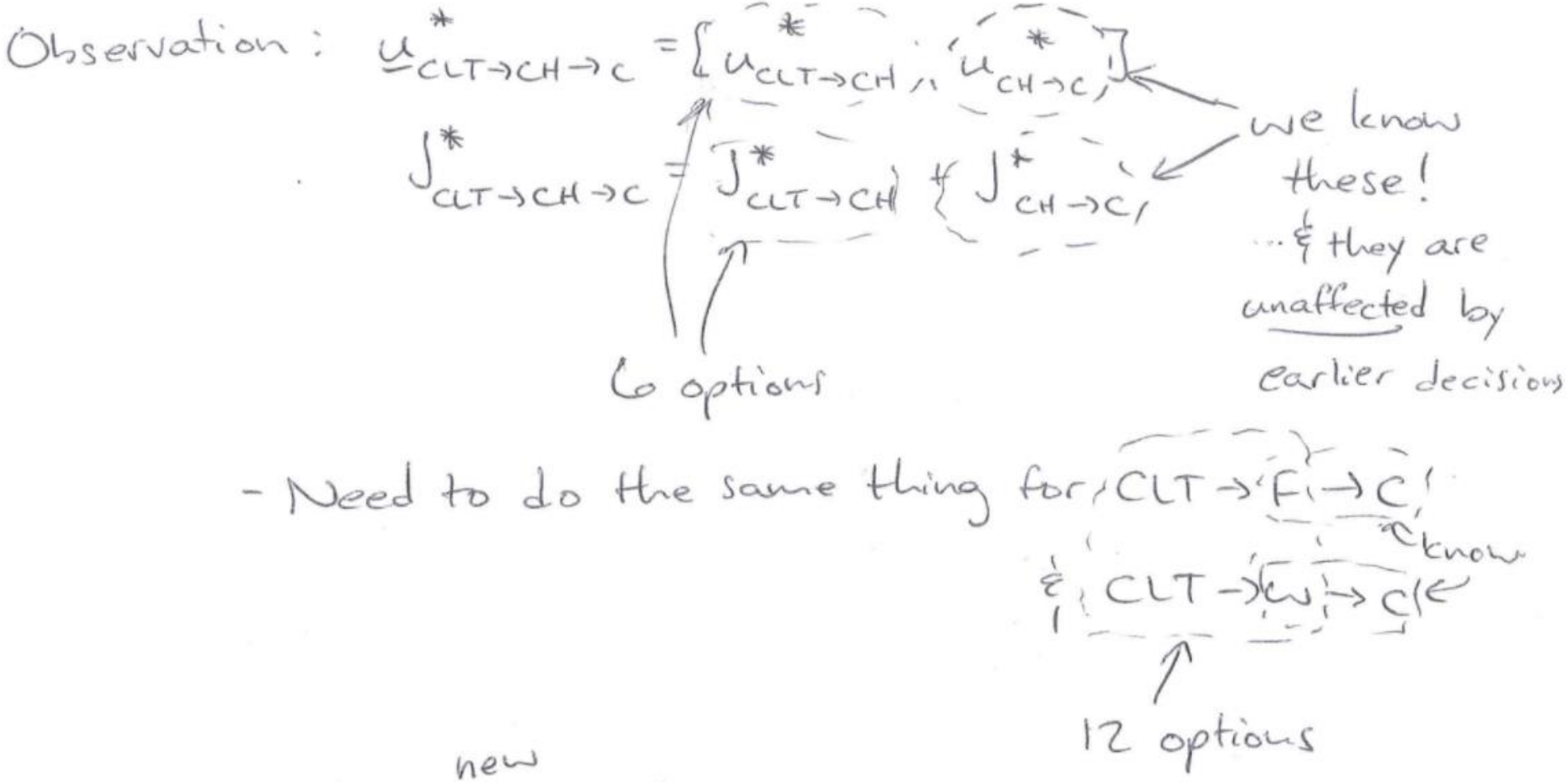
$u^*_{F\to C}$, $J^*_{2,F}$ – " "

$u^*_{W\to C}$, $J^*_{2,W}$ – " "

18 fn. evals.

Another Optimal Control Example – Better Solution via *Backward Recursion*

Stage 1:

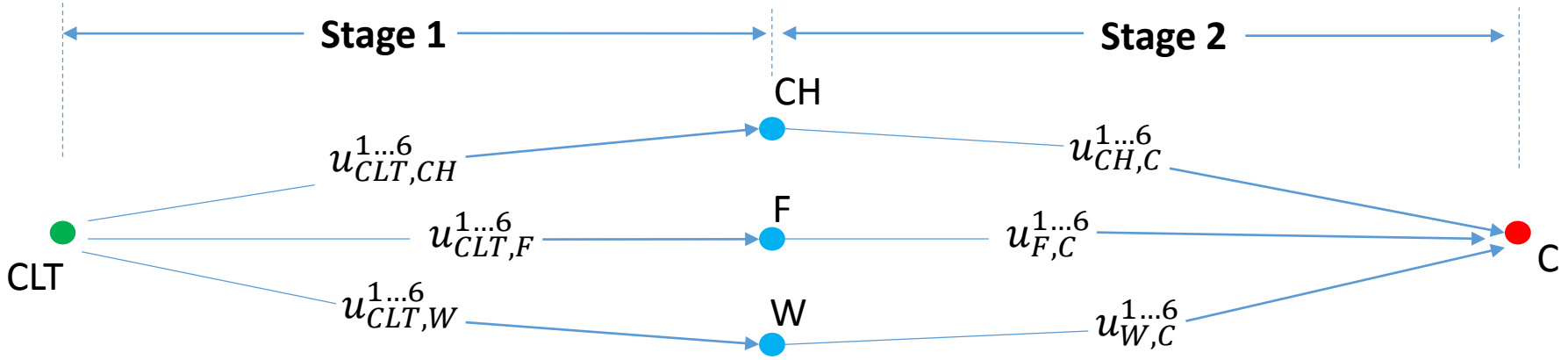


Another Optimal Control Example – Better Solution via *Forward Recursion*

UNC CHARLOTTE

Key observations:

- The trip can be divided into 2 **stages**, which occur in **sequence** (stage 1 happens before stage 2)
- After a control signal is applied at each **stage**, the system arrives at a **state** (in this case, the ***states*** are ***cities***, which can take on values of CLT, CH, F, W, or C)



Main idea of forward recursion:

- Compute all ***stage costs*** for the first stage $(J(u^{1\ldots6}_{CLT,CH}), J(u^{1\ldots6}_{CLT,F}), J(u^{1\ldots6}_{CLT,W}))$ – **18 cost function evaluations**
- For each destination state at the first stage (CH, F, W), determine the control input that minimizes the cost. Denote the corresponding minimum cost to each destination state as $J^*_{1,CH}, J^*_{1,F}, J^*_{1,W}$
- Move to stage 2. Compute all ***stage costs*** for this stage $(J(u^{1\ldots6}_{CH,c}), J(u^{1\ldots6}_{F,C}), J(u^{1\ldots6}_{W,C}))$, and add these to $J^*_{1,CH}, J^*_{1,F}, J^*_{1,W}$, respectively – **18 cost function evaluations**. The lowest resulting cost (and corresponding control sequence) is the global optimum (and optimizer)!

Analysis of our Example

UNC CHARLOTTE

Critical question: *Why did the recursive approaches allow us to obtain a globally optimal solution with fewer objective function evaluations than an exhaustive search?*

Answer: The key to understanding this lies in the concept of **stages** and **states**…Once the system has reached a given **state** at the end of a **stage**, the cost associated with applying a particular control input at the next stage ***does not depend on how the system reached the originating state.***

Implication (example): Your choice of how to go from Charlotte to Chapel Hill has no bearing on the cost (fuel consumption) of going from Chapel Hill to Corolla. So if you find that $u^3_{CH,C}$ is the best way to go from Chapel Hill to Corolla, then it is the best way to proceed from Chapel Hill to Corolla from Charlotte, ***regardless of how you got from Charlotte to Chapel Hill***. That means that you don't need to consider control sequences that involve $u^1_{CH,C}$, $u^2_{CH,C}$, $u^4_{CH,C}$, $u^5_{CH,C}$, or $u^6_{CH,C}$.

The above is a conceptual description of ***Bellman's principle of optimality***, a famous result that is at the heart of ***dynamic programming***.

Analysis of our Example

UNC CHARLOTTE

Dynamic programming is built on the concept of

Sequences and states

↑ give rise to stages

A state characterites all of the information needed to predict the future evolution of the system for a given input.

If $\underline{x}(k)$ is the state at step k & $u(i)$ is known for $i = k \ldots N$, then it is possible to predict $x(i)$, $i = k+1 \ldots N$

Bellman's Principle of Optimality

UNC CHARLOTTE

Conditions: Bellman's principle applies to discrete-time systems (also can be used for continuous-time systems) of the form:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), u(k))$$

(Successor state ($\mathbf{x}(k+1)$) depends on the originating state ($\mathbf{x}(k)$) and the control signal at stage k)

Notation: Given an objective function structure $J\big(u, \mathbf{x}(0)\big) = \sum_{i=0}^{N-1} g(\mathbf{x}(i), u(i))$:

- $J^*_{\mathbf{x}_i(j)\to\mathbf{x}_f(N)}$ = Optimal *cost to go* from intermediate state x_i at step j to final state $\mathbf{x}_f$ at step N
- $J^*_{\mathbf{x}_o(j-M)\to\mathbf{x}_i(j)}$ = Optimal *cost to arrive* at intermediate state $\mathbf{x}_i$ at step j from originating state $\mathbf{x}_o$ at step j-M (when M = 1, often the optimal cost to arrive is the ***only*** cost to arrive, since there is only one way to get from $\mathbf{x}_o$ to $\mathbf{x}_i$ in one step)
- $J^*_{\mathbf{x}_o(j-M)\to\mathbf{x}_i(j)\to\mathbf{x}_f(N)}$ = Optimal cost to go from originating state $\mathbf{x}_o$ at step j-1 to final state $\mathbf{x}_f$ at step N, through intermediate state $\mathbf{x}_i$ at step j

Bellman's principle of optimality: $J^*_{\mathbf{x}_o(j-M)\to\mathbf{x}_i(j)\to\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(j-M)\to\mathbf{x}_i(j)} + J^*_{\mathbf{x}_i(j)\to\mathbf{x}_f(N)}$

Bellman's Principle of Optimality

UNC CHARLOTTE

$$\underline{x}(k+1) = f(\underline{x}(k), u(k))$$

$$J(\underline{u}; \underline{x}(0)) = \sum_{i=0}^{N-1} \underbrace{g(\underline{x}(i), u(i))}_{\text{"stage cost"}}$$

Define: $J^*_{\underline{x}_{inter}(j) \to \underline{x}_{final}(N)}$ = optimal cost to go from $\underline{x}_{inter}$ at j to $\underline{x}_{final}$ at N

$J^*_{\underline{x}_{orig}(j-M) \to \underline{x}_{inter}(j)}$ = optimal cost to arrive at $\underline{x}_{inter}$ at j, starting at $\underline{x}_{orig}$ at $j-M$

Bellman's principle: $J^*_{\underline{x}_{orig}(j-M) \to \underline{x}_{inter}(j) \to \underline{x}_{final}(N)} = J^*_{\underline{x}_{inter}(j) \to \underline{x}_{final}(N)} + J^*_{\underline{x}_{orig}(j-M) \to \underline{x}_{inter}(j)}$

Dynamic Programming – Backward Recursion

UNC CHARLOTTE

Getting set up:

- Quantize control variables into p discrete values
- Quantize state variables into q discrete values

Solution algorithm:

- Start at step N-1. For each of the q allowable state variables, calculate the stage cost ($g(\mathbf{x}(N-1), u(N-1))$ for each of the p allowable control variables that lead to constraint satisfaction. Control variables that do not satisfy constraints are termed ***inadmissible***. Record the optimal control signals and corresponding stage costs for each originating state.
- Move to step N-2. For each of the q allowable state variables, calculate the stage cost ($g(\mathbf{x}(N-2), u(N-2))$ and associated intermediate state, $\mathbf{x}_i(N-1)$, for each of the p allowable control variables that lead to constraint satisfaction. To determine which control variable is optimal, compute the total cost to go as $J^*_{\mathbf{x}_o(N-2)\to\mathbf{x}_i(N-1)\to\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(N-2)\to\mathbf{x}_i(N-1)} + J^*_{\mathbf{x}_i(N-1)\to\mathbf{x}_f(N)}$
- Move to step N-3 and repeat the process (total cost to go is now $J^*_{\mathbf{x}_o(N-3)\to\mathbf{x}_i(N-2)\to\mathbf{x}_f(N)} = J^*_{\mathbf{x}_o(N-3)\to\mathbf{x}_i(N-2)} + J^*_{\mathbf{x}_i(N-2)\to\mathbf{x}_f(N)}$). Keep stepping backward in time until step 0.

Dynamic Programming – Backward Recursion

Consider a system $\underline{x}(k+1) = f(\underline{x}(k), u(k))$

Cost fn: $J(\underline{u}, \underline{x}(0)) = \sum_{i=0}^{N-1} g(\underline{x}(i), u(i))$

Suppose that $\underline{x}(k+1)$ & $g(\underline{x}(k), u(k))$ are given by the following table:

$u(i) \in \{u_1, u_2, u_3\}$

$\underline{x}(i) \in \{\underline{x}_1, \underline{x}_2, \underline{x}_3\}$

$\underline{x}(k)$ \ $u(k)$	u_1	u_2	u_3
$\underline{x}_1$	$\underline{x}_2$, 10	$\underline{x}_3$, 4	$\underline{x}_1$, 5
$\underline{x}_2$	$\underline{x}_1$, 8	$\underline{x}_3$, 7	$\underline{x}_2$, 4
$\underline{x}_3$	$\underline{x}_2$, 7	$\underline{x}_1$, 5	$\underline{x}_3$, 8

Dynamic Programming – Backward Recursion

UNC CHARLOTTE

Stage N-1
(Step N-1)

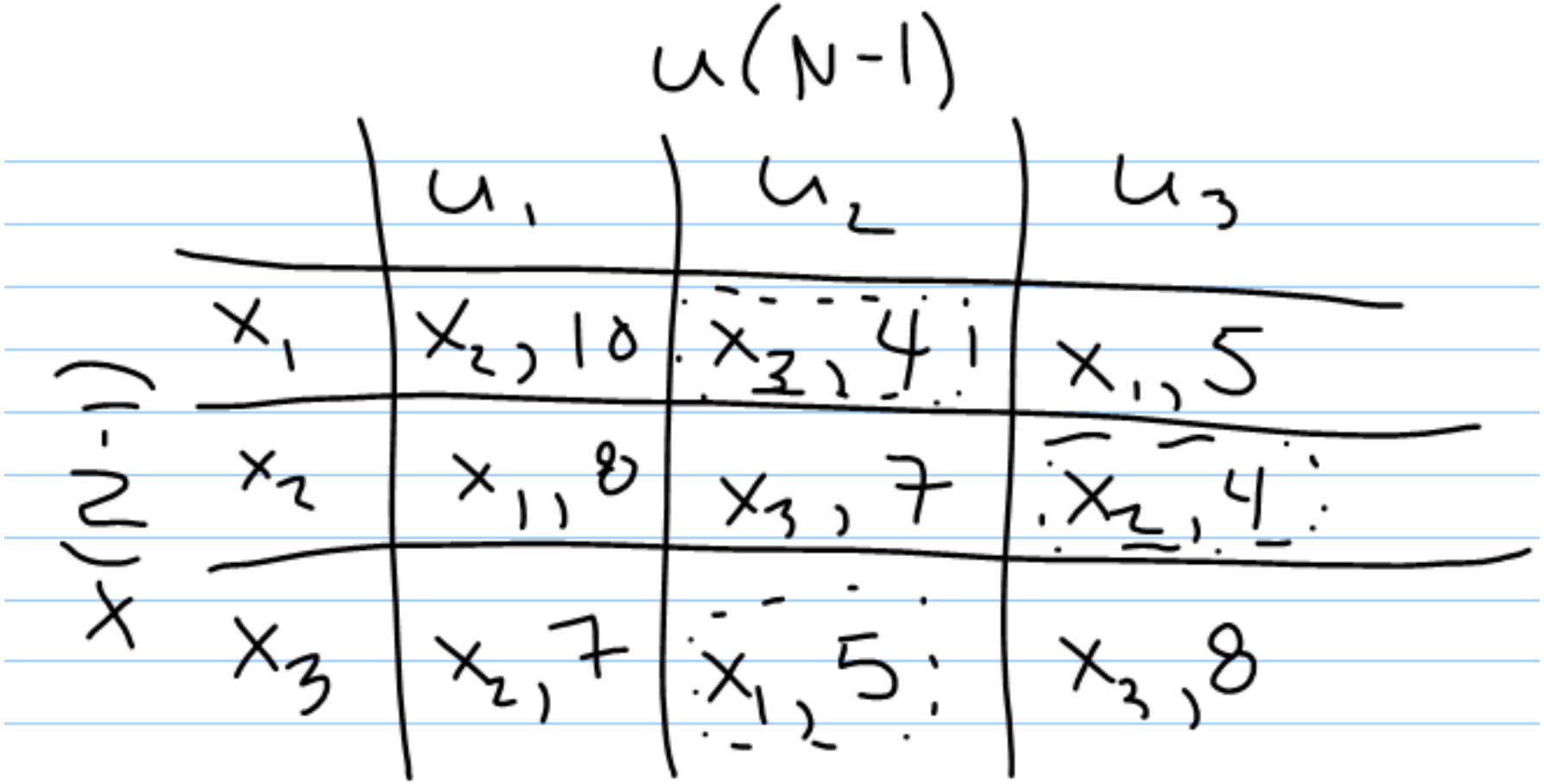
I am going to note...

i) For every $x(N-1), u(N-1)$ combo, I will list $x(N)$, $\underbrace{g(x(N-1), u(N-1))}_{J_{x(N-1)\to x(N)}}$

ii) For every $x(N-1)$, I'll draw a box around the "best" option (lowest g)

$\Rightarrow J^*_{x(N-1)\to x(N)}$ } optimal "cost to go".

Dynamic Programming – Backward Recursion



Dynamic Programming – Backward Recursion

UNC CHARLOTTE

Stage N-2: For every $x(N-2), u(N-2)$, I will list: i) $x(N-1)$

ii) $g(x(N-2), u(N-2)) + J^*_{x(N-1) \to x(N)}$

$= J^*_{x(N-2) \to x(N-1)} + J^*_{x(N-1) \to x(N)}$

For given $u(N-2)$

Dynamic Programming – Backward Recursion

UNC CHARLOTTE

I will draw a box around the "best choice"
For given $x(N-2)$, the box will encircle

$$\text{Over all possible } u(N-2) \rightarrow J^*_{x(N-2)\rightarrow x(N-1)} + J^*_{x(N-1)\rightarrow x(N)}$$

$x(N-2)$ \ $u(N-2)$	u_1	u_2	u_3
x_1	x_2, 14	(x_3), 9	x_1, 9
x_2	x_1, 12	x_3, 12	(x_2), 8
x_3	x_2, 11	(x_1), 9	x_3, 13

$g(x_1, u_1) = 10$

$J^*_{x(N-1)=x_2 \rightarrow x(N)} = 4$

Dynamic Programming – Backward Recursion

UNC CHARLOTTE

Stage N-3: We will list i) $x(N-2)$

ii) $g(x(N-3), u(N-3)) + J^*_{x(N-2) \to x(N)}$

We will draw a ⊡ around best option

for each x (x_1, x_2, x_3), which will tell us

$J^*_{x(N-3) \to x(N)}$

	$u(N-3)$ u_1	u_2	u_3
x_1	x_2, 18	x_3, 13	x_1, 14
x_2	x_1, 17	x_3, 16	x_2, 12
x_3	x_2, 15	x_1, 14	x_3, 17

$g(x(N-3), u(N-3)) = 10$

$J^*_{x(N-2)=x_2 \to x(N)} = 8$

$\Rightarrow J^*_{x(N-3) \to x(N-2) \to x(N)} = 18$

when $u(N-3) = u_1$

Dynamic Programming – Backward Recursion

UNC CHARLOTTE

i) At every stage, I make q cost fn. calcs.

q = state quantization level, p = control quantization

# of computations per stage = pq

⇒ Total # of computations = Npq

linear in N

ii) At step 0, we now have J^* & $\underline{u}^*$

for all initial conditions

Dynamic Programming – Backward Recursion Example

UNC CHARLOTTE

System dynamics: $x(k+1) = 2x(k) + 0.5u(k)$

Optimization problem: Minimize $J(\mathbf{u}, x(0)) = \sum_{i=0}^{N-1} 5x(i+1)^2 + u(i)^2$

Subject to: $-1 \leq x(i) \leq 1, i = 1 \ldots N$

where $x(0) = 1$ and $u(i) \in \{-3, -2, -1, 0, 1, 2, 3\}, i = 0 \ldots N-1$

- Determine the optimal control sequence, $\mathbf{u}^*$, using dynamic programming, with state variables quantized as $x_i \in \{-1, -0.5, 0, 0.5, 1\}, i = 1 \ldots N$, for $N = 5$ and $N = 10$ …Sample code available on Canvas

- How many cost function evaluations are required? How does this compare with that of an exhaustive search, and how does the number depend on N?

Dynamic Programming – Backward Recursion Example

UNC CHARLOTTE

$p = 7, \quad q = 5, \quad N = 5$

# of req'd fn. evals $= 5 \cdot 5 - 7 = \boxed{175}$

Dynamic Programming – Backward Recursion Example – Continuation

UNC CHARLOTTE

System dynamics: $x(k+1) = 2x(k) + 0.5u(k)$

Optimization problem: Minimize $J(\mathbf{u}, x(0)) = \sum_{i=0}^{N-1} 5x(i+1)^2 + u(i)^2$

Subject to: $-1 \leq x(i) \leq 1, i = 1 \ldots N$

$u(i) \in \{-3, -2, -1, 0, 1, 2, 3\}, i = 0 \ldots N-1$

- For $N = 10$, determine the optimal control sequence, $\mathbf{u}^*$, using dynamic programming, with state variables quantized as $x_i \in \{-1, -0.5, 0, 0.5, 1\}, i = 0 \ldots N-1$, for $N = 5$ and $N = 10$, ***for all five possible initial states***…Sample code available on Canvas
- How many cost function evaluations are required? **350**. How does this compare with that of an exhaustive search, and how does the number depend on N? **Linearly**
- **Key point:** Dynamic programming provides $\mathbf{u}^*$ ***for all possible initial conditions*** – this information can be stored on a control computer and referenced during operation

Dynamic Programming – Forward Recursion

UNC CHARLOTTE

Getting set up:

- Quantize control variables into p discrete values
- Quantize state variables into q discrete values
- Usually the initial state, $\mathbf{x}(0)$, is specified (you don't get to choose your initial state)

Solution algorithm:

- Start at step 1. For each of the q allowable values of $\mathbf{x}(1)$, back-compute the value of $\mathbf{x}(0)$ and stage cost that results from each of the p allowable control values. For each admissible control value (whose associated $\mathbf{x}(0)$ satisfies initial condition requirements), determine the stage cost– the lowest stage cost, denoted by $J^*_{\mathbf{x}_0(0)\to\mathbf{x}_t(1)}$, is the ***optimal cost to arrive*** at state $\mathbf{x}_t(1)$.
- Move to step 2. For each of the q allowable values of $\mathbf{x}(2)$, back-compute $\mathbf{x}(1)$ and the associated stage cost for each of the p allowable control variables that lead to constraint satisfaction. To determine which control sequence is optimal, compute the total cost to arrive as $J^*_{\mathbf{x}_o(0)\to\mathbf{x}_i(1)\to\mathbf{x}_f(2)} = J^*_{\mathbf{x}_o(0)\to\mathbf{x}_i(1)} + J^*_{\mathbf{x}_i(1)\to\mathbf{x}_f(2)}$
- Move to step 2 and repeat the process. Keep stepping forward in time until step N.

Dynamic Programming – Forward Recursion

UNC CHARLOTTE

Idea of forward recursion:

Given $\underline{x}(k+1)$, $u(k)$, can compute $\underline{x}(k)$

The following chart gives $x(k-1) = g(x(k-1), u(k-1))$ for a given $x(k)$ & $u(k-1)$

Control:

	u_1	u_2	u_3
x_1	$x_2, 8$	$x_3, 3$	$x_1, 5$
x_2	$x_3, 7$	$x_1, 9$	$x_2, 4$
x_3	$x_1, 6$	$x_2, 7$	$x_3, 8$

Suppose $x(0) = x_3$

Dynamic Programming – Forward Recursion

UNC CHARLOTTE

Step 1 (Stage 0): Chart entries will show i) $g(x(0), u(0))$ ii) $J^*_{x(0) \to x(1)}$

	u_1	u_2	u_3
x_1	$x_2, 8$	$x_3, 3$	$x_1, 5$
x_2	$x_3, 7$	$x_1, 9$	$x_2, 4$
x_3	$x_1, 6$	$x_2, 7$	$x_3, 8$

Dynamic Programming – Forward Recursion

UNC CHARLOTTE

Step 2 (stage 1):

Chart entries will show $g(x(1),u(1)) + J^*_{x(0)\to x(1)}$

for every u, x combo

Boxed "best option" will give us $J^*_{x(0)\to x(2)}$ for each x

	u_1	u_2	u_3
x_1	$x_2, 15$	$x_3, 11$	$x_1, 8$
x_2	$x_3, 15$	$x_1, 12$	$x_2, 11$
x_3	$x_1, 9$	$x_2, 14$	$x_3, 16$

top left:

$g(x(1),u(1)) = 8$

$J^*_{x(0)\to x(1)=x_2} = 7$

$8+7 = 15$

Dynamic Programming – Forward Recursion Example – First Steps

UNC CHARLOTTE

System dynamics: $x(k+1) = 2x(k) + 0.5u(k)$

Optimization problem: Minimize $J(\mathbf{u}, x(0)) = \sum_{i=0}^{N-1} 5x(i+1)^2 + u(i)^2$

Subject to: $-1 \leq x(i) \leq 1, i = 1 \ldots N$

where $x(0) = 1$ and $u(i) \in \{-3, -2, -1, 0, 1, 2, 3\}, i = 0 \ldots N-1$

- ***Try*** to perform the first two steps of forward recursion (i.e., step 1 and one iteration of step 2, per slide 16), with state variables quantized as $x_i \in \{-1, -0.5, 0, 0.5, 1\}, i = 1 \ldots N$

- What goes wrong when you try to perform these steps? Do you think this is restricted to forward recursion?

Dynamic Programming – Forward Recursion Example – First Steps

UNC CHARLOTTE

$x(k+1) = 2x(k) + 0.5u(k)$

Stage 1 (step 2): $x(2) = 2x(1) + 0.5u(1)$

suppose $u(1) = 3$, $x(2) = -1$

$\Rightarrow 2x(1) = x(2) - 0.5u(1) = -0.5$

$\Rightarrow x(1) = -0.25$ ← not on the chart

In the example above, the predecessor state is not one of the quantized values!

Dynamic Programming – Modified Backward Recursion Example – Last Steps

UNC CHARLOTTE

System dynamics: $x(k+1) = 1.5x(k) + 0.5u(k)$

Optimization problem: Minimize $J(\mathbf{u}, x(0)) = \sum_{i=0}^{N-1} 5x(i+1)^2 + u(i)^2$

Subject to: $-1 \leq x(i) \leq 1, i = 1 \ldots N$

where $x(0) = 1$ and $u(i) \in \{-3, -2, -1, 0, 1, 2, 3\}, i = 0 \ldots N-1$

- ***Try*** to perform the last (N-1) and second to last (N-2) steps of backward recursion, with state variables quantized as $x_i \in \{-1, -0.5, 0, 0.5, 1\}, i = 1 \ldots N$
- What goes wrong? Why does nothing go wrong at step N-1?

Dynamic Programming – Modified Backward Recursion Example – Last Steps

$$x(k+1) = 1.5\,x(k) + 0.5\,u(k)$$

Stage N-2: Suppose $u(N-2)=0$, $x(N-2)=0.5$

$\Rightarrow x(k+1) = 1.5 \cdot 0.5 = 0.75$

not one of the quantized values!

Note: Nothing goes wrong at stage N-1, since there is no cost to go from step N…Therefore, there is no requirement that the state at step N belong to the set of quantized values.

Preview of Upcoming Lectures

UNC CHARLOTTE

More dynamic programming:

- **Interpolation techniques** for situations where one (or more) of the p allowable control values leads to a state value that lies ***between*** the quantized states
- Simple forward and backward recursion examples with interpolation
- More sophisticated examples