Approximation in Value Space

- Let's begin by writing the DP algorithm:

$$J_N(x_n) = g_N(x_N)$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} \left\{ \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)) \right] \right\}, \forall k \in \{0, ..., N-1\}$$

- Where we can compute a sub-optimal admissible policy by using an approximate cost-to-go function $\tilde{J}_{k+1}(x_{k+1})$:

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \left\{ \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right] \right\}, \forall k \in \{0, ..., N-1\}$$

- $\tilde{J}_{k+1}(x_{k+1})$ is also called an **approximate value function.**

Multistep Lookahead

- As we saw last time, we can let the approximate function $\tilde{J}_{k+1}(x_{k+1})$ be itself a l-step version of the DP problem:

$$\tilde{J}_{k+1}(x_{k+1}) = \min_{(\mu_{k+1},\ldots,\mu_{k+l-1})} \mathbb{E}_{w_{k+1},\ldots,w_{k+l-1}} \left[\tilde{J}_{k+l}(x_{k+l}) + \sum_{i=k+1}^{k+l-1} g_i(x_i, \mu_i(x_i), w_i) \right]$$

- Where $\tilde{J}_{k+l}(x_{k+l})$ is yet another (albeit simpler) approximate value function.

- The multi-step Lookahead can be summarized as follows:

First l-steps "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+l-1}} \mathbb{E}_{w_{k+1},\ldots,w_{k+l-1}} \left[g_k(x_k, u_k, w_k) + \sum_{i=k+1}^{k+l-1} g_i(x_i, \mu_i(x_i), w_i) + \tilde{J}_{k+l}(x_{k+l}) \right]$$

DP Minimization

Lookahead minimization

Cost-to-go Approximation

Multistep Lookahead

- Ideally the lookahead length is “large-enough” so that we may use very simple approximate functions at the end of the lookahead period. For example:

$$\tilde{J}_{k+l}(x_{k+l}) \equiv 0, \text{ or } \quad \tilde{J}_{k+l}(x_{k+l}) = g_N(x_{k+l})$$

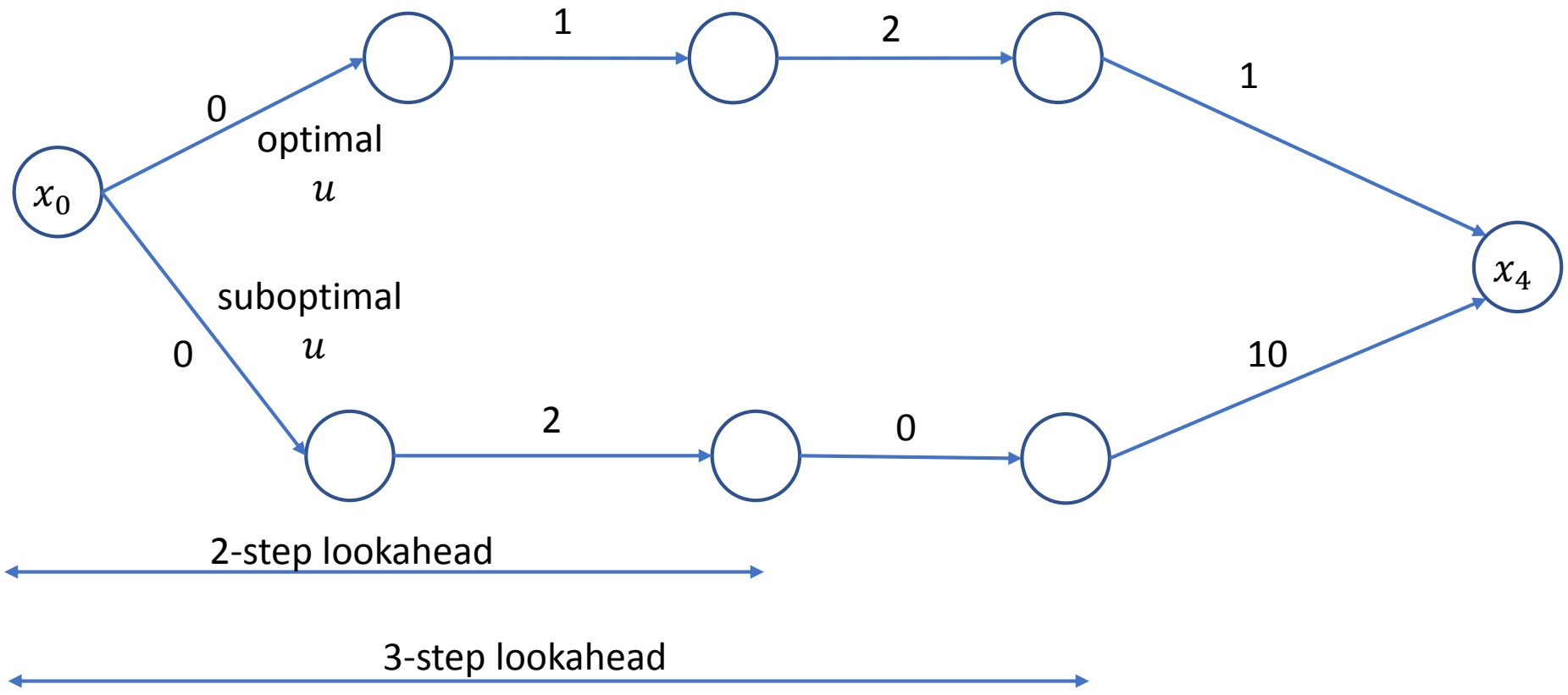
- In the multistep lookahead we compute a single control u_k that get’s applied at stage k, and a sequence of functions $(\mu_{k+1}(x_{k+1}), \dots, \mu_{k+l-1}(x_{k+l-1}))$ which gets **discarded.**

- After applying u_k and obtaining the next state x_{k+1}, the whole system “rolls forward” and we have to re-solve the multistep-lookahead problem.

- This approach is called the **Rolling Horizon** approach, which is quite intuitive.

Example: Pitfall of Multistep Lookahead

- Consider this simple 4-stage Shortest-path problem:



- So, a larger lookahead window may decrease performance.

Example: Partially Deterministic Lookahead

- The multistep lookahead is very flexible. Consider again the l-step problem:

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \left\{ \mathbb{E}_{w_k} \left[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right] \right\}, \forall k \in \{0, \ldots, N-1\}$$

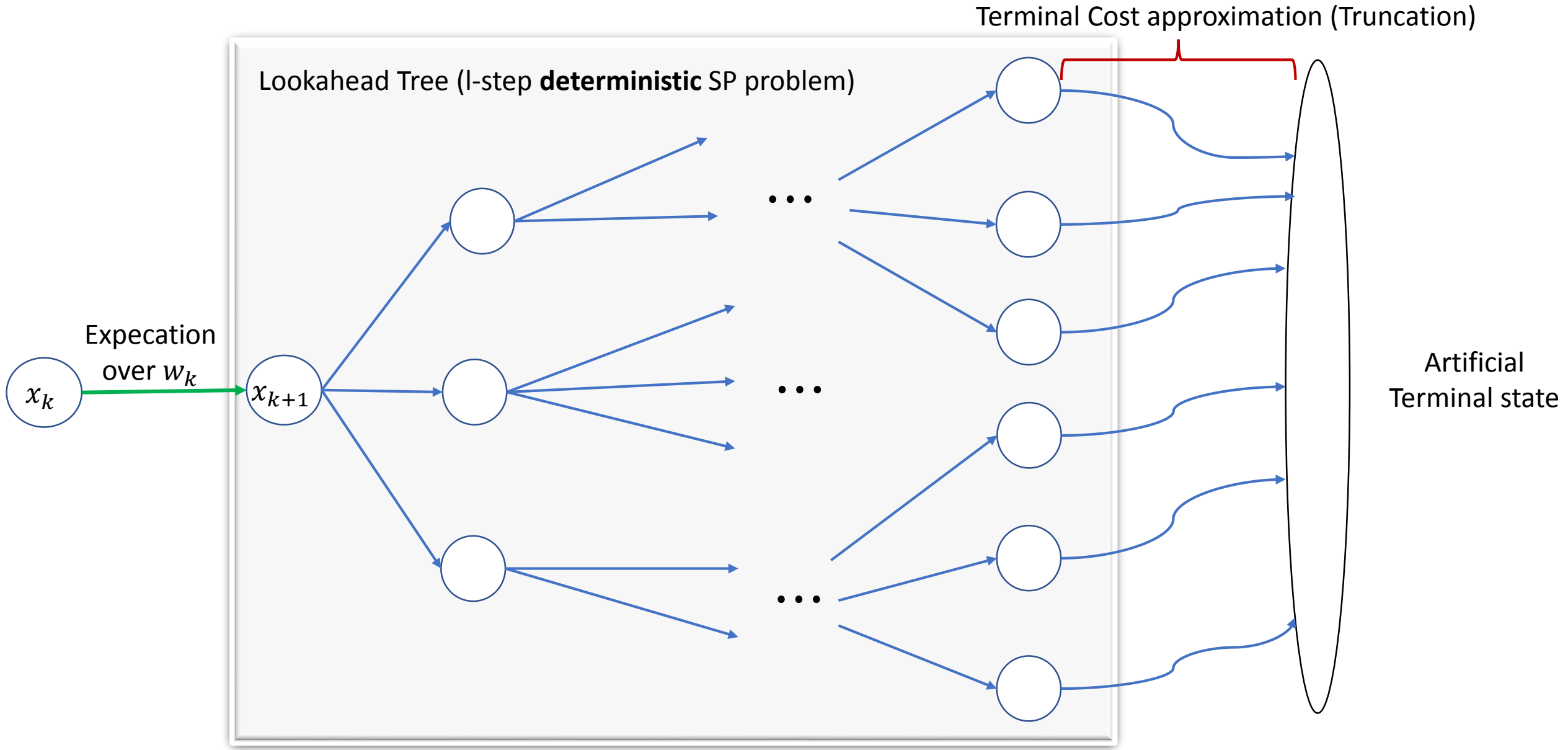
$$\tilde{J}_{k+1}(x_{k+1}) = \min_{(\mu_{k+1}, \ldots, \mu_{k+l-1})} \mathbb{E}_{w_{k+1}, \ldots, w_{k+l-1}} \left[\tilde{J}_{k+l}(x_{k+l}) + \sum_{i=k+1}^{k+l-1} g_i(x_i, \mu_i(x_i), w_i) \right]$$

- Suppose that we apply the *Certainty Equivalence* principle where we replace the disturbances $(w_{k+1}, \ldots, w_{k+l})$ by their average ("typical") values $(\bar{w}_{k+1}, \ldots, \bar{w}_{k+l})$:

$$\bar{w}_{k+1} = \mathbb{E}[w_{k+1} | x_{k+1}, u_{k+1}]$$

- Then the approximate value function $\tilde{J}_{k+1}(x_{k+1})$ can be computed by a Deterministic Shortest Path Algorithm.

Partially Deterministic Lookahead



Scenario-based Lookahead

- Note that the Certainty Equivalence approach can be generalized by using multiple scenarios in computing the expectation in:

$$\tilde{J}_{k+1}(x_{k+1}) = \min_{(\mu_{k+1},\ldots,\mu_{k+l-1})} \mathbb{E}_{w_{k+1},\ldots,w_{k+l-1}} \left[\tilde{J}_{k+l}(x_{k+l}) + \sum_{i=k+1}^{k+l-1} g_i(x_i, \mu_i(x_i), w_i) \right]$$

- Suppose we have in our disposal a simulator that is able to generate a *scenario* sequence given the state x_{k+1}:

$$w^s(x_{k+1}) = (w^s_{k+1}, \ldots, w^s_{k+l-1}), s \in \{1, .., S\}$$

- And let $C^s(x_{k+1})$ be the solution of the **deterministic lookahead problem** under scenario s.

Scenario-based Lookahead

- Then we compute a sample of average approximation (SAA):

$$\tilde{J}_{k+1}(x_{k+1}) = \sum_{s=1}^{S} r_s C_s(x_{k+1})$$

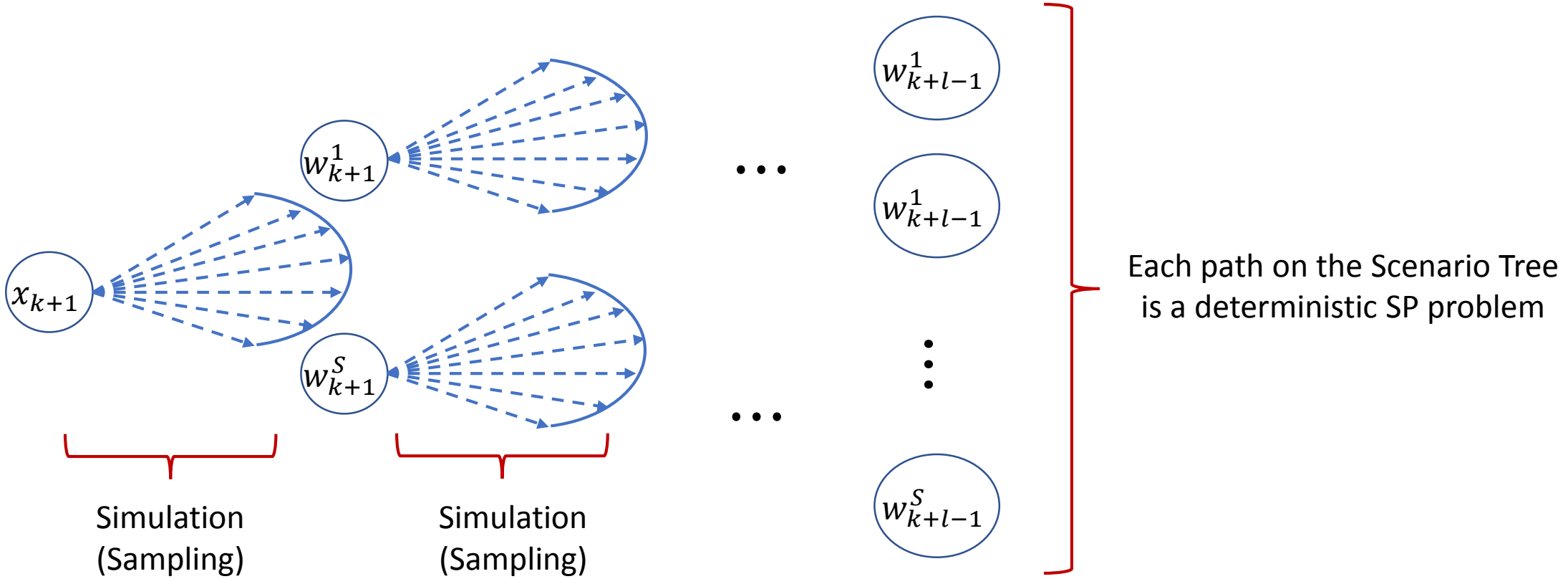
- Where $r = (r_1, \ldots, r_S)$ are some probability distribution vector, representing the "frequency" of each scenario, often called the scenario weights.

- Note that each scenario is independent.

- This idea of replacing required expectations by SAA and using simulation will be pervasive in most (if not all) algorithms in Approximate DP and Reinforcement Learning.

Scenario-based Lookahead

- The scenarios can be represented by a Scenario Tree (also called Monte-Carlo Tree):



Rollout Algorithm: Deterministic Case

- Let's focus on the **Deterministic** 1-step lookahead algorithm (for simplicity):

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \left\{ g_k(x_k, u_k) + \tilde{J}_{k+1}(f_k(x_k, u_k)) \right\}, \forall k \in \{0, ..., N-1\}$$

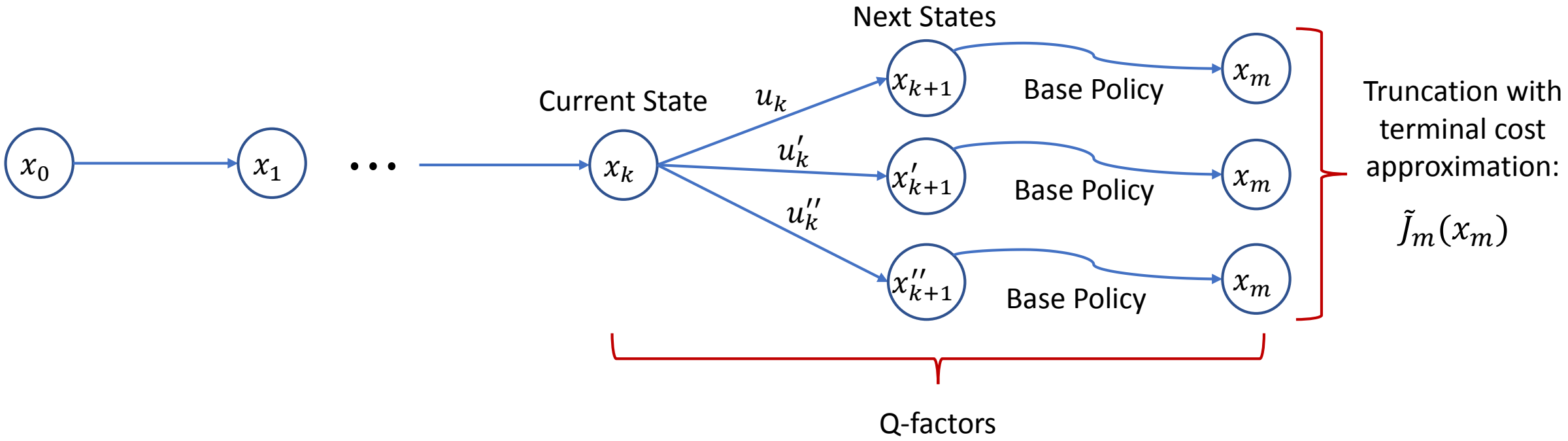
- Where the approximate function $\tilde{J}_{k+1}(x_{k+1})$ is given as the total cost of some *Base Policy* (or *Base Heuristic*) $\hat{\pi} = (\hat{\mu}_{k+1}, \dots, \hat{\mu}_{N-1})$:

$$\tilde{J}_{k+1}(x_{k+1}) = g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \hat{\mu}_i(x_i))$$

- Note that we can truncate the rollout at some stage $m < N$, and use some approximation for the terminal cost (like we did for the lookahead problems).

Rollout Algorithm: Deterministic Case

- Schematically, we can represent the Deterministic Rollout Algorithm as follows:



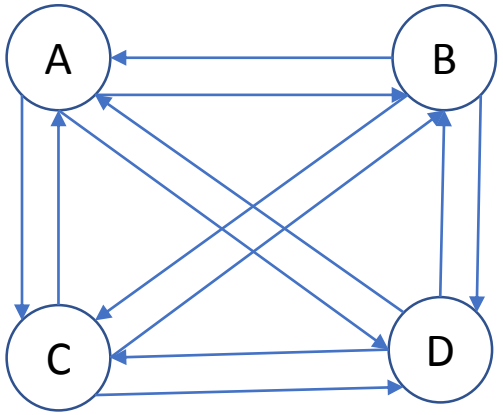
- The **Rollout Policy**:

$$\tilde{\mu}_k(x_k) = \arg \min_{u_k \in U_k(x_k)} \{\tilde{Q}_k(x_k, u_k)\}, \forall k \in \{0, ..., N-1\}$$

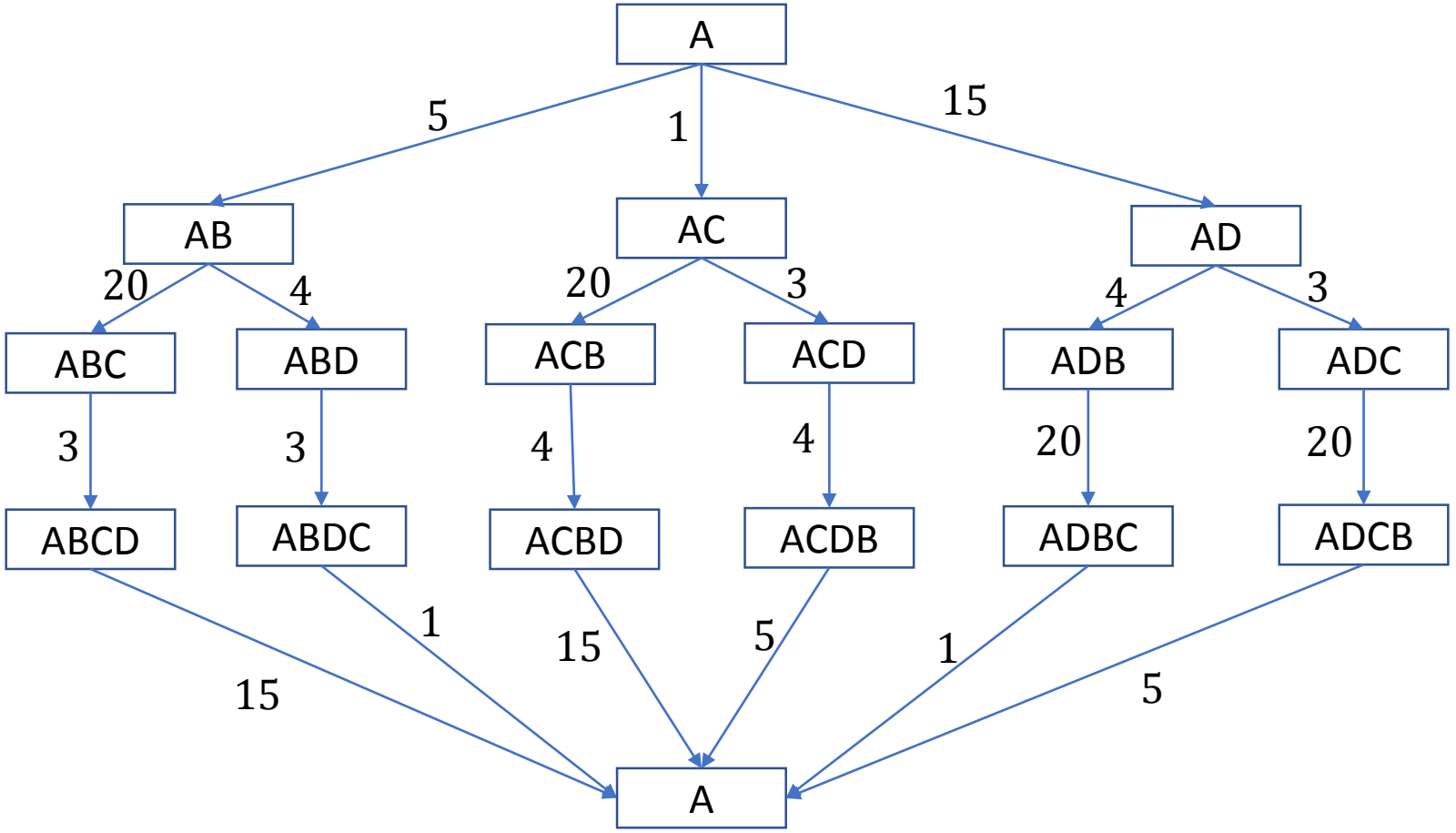
$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + \tilde{J}_{k+1}(f_k(x_k, u_k)), \forall k \in \{0, ..., N-1\}$$

Example: Traveling Salesman Problem

- Recall from last time the TSP:



Distances: $\begin{bmatrix} 0 & 5 & 1 & 15 \\ 5 & 0 & 20 & 4 \\ 1 & 20 & 0 & 3 \\ 15 & 4 & 3 & \end{bmatrix}$



Example: Traveling Salesman Problem

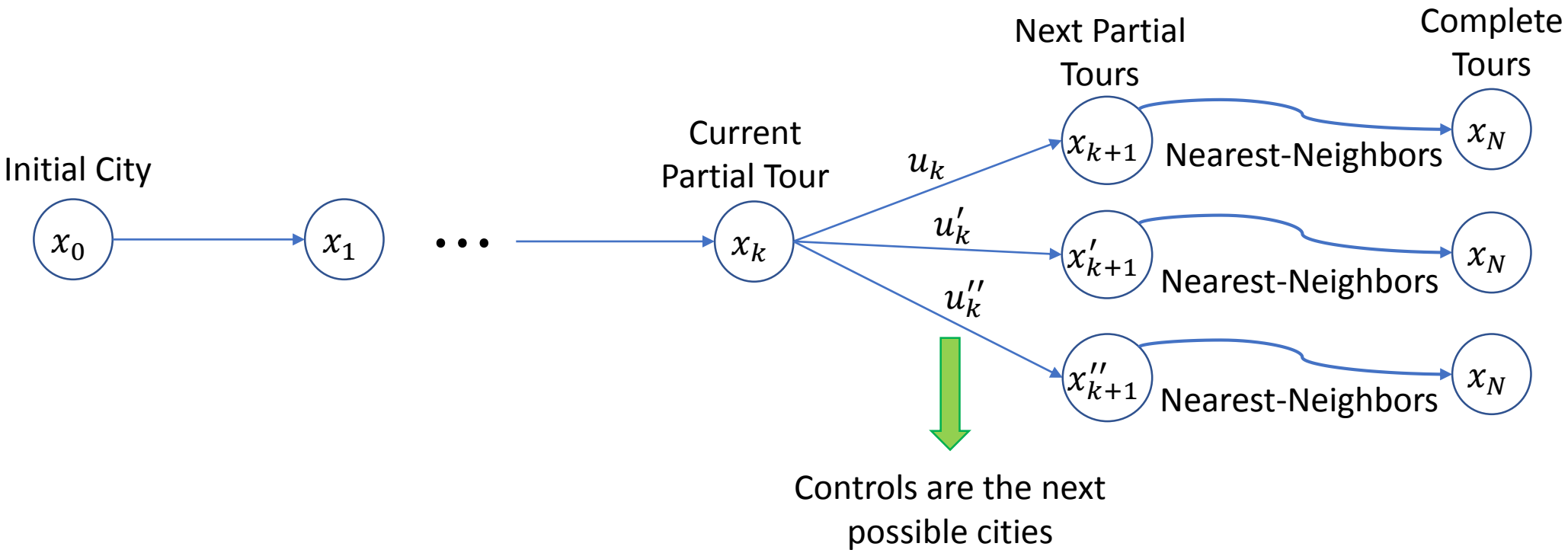
- The simple *Nearest-Neighbor Heuristic* can be seen as a special case of the Rollout Algorithm, where the Base Policy adds a new city to a partial tour that greedily minimize the aggregated cost and does not form a cycle.

- In our formulation x_k is a partial tour with k cities: $x_k = (c_0, \dots, c_k)$.

- The nearest-neighbor heuristic adds a new city c_{k+1} to the partial tour if $g(c_k, c_{k+1})$ is minimized for all cities $c_k \neq c_0, \dots, c_k$, Thus forming a larger partial tour: $x_{k+1} = (c_0, \dots, c_k, c_{k+1})$.

- The Rollout Algorithm implements the nearest-neighbor heuristic as the Base Policy.

Example: Traveling Salesman Problem



- After each iteration we re-run the Nearest-Neighbors heuristic from the next partial tours

Rollout Algorithm: Deterministic Case

- A key question in the Rollout Algorithm is whether the new *rollout policy* $\tilde{\pi}$ is better than the *base policy* $\hat{\pi}$.

- This can be achieved if the Base Policy (Heuristic) is *Sequentially Improving*, that is:

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + \tilde{J}_{k+1}(f_k(x_k, u_k)), \forall k \in \{0, ..., N-1\}$$

$$\tilde{J}_{k+1}(x_{k+1}) = g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \hat{\mu}_i(x_i))$$

- and

$$\min_{u_k \in U_k(x_k)} \{\tilde{Q}_k(x_k, u_k)\} \leq \tilde{J}_k(x_k)$$

- In words: "The minimal approximate Q-factor at x_k is smaller then the cost-to-go of the Base Policy at x_k".

Rollout Algorithm: Deterministic Case

- Suppose that the Base Policy is sequentially improving. We want to show that:

$$J_{k,\tilde{\pi}}(x_k) \leq \tilde{J}_k(x_k)$$

- In words: " The cost-to-go of the rollout policy at $x_k$is smaller than the cost-to-go of the Base Policy".

- If that inequality holds we say that the rollout policy achieves **Policy Improvement.**

- It turns out that if base Policy is sequentially improving, than the rollout policy achieves policy improvement.

- Example: In the TSP, the nearest neighbors heuristic is sequentially improving.

Rollout Algorithm: Deterministic Case

- Note that the Rollout Algorithm is amenable to several base policies. Suppose we have m different base policies and at a given state x_{k+1}, the m'th base policy produces the trajectory:

$$\hat{T}_{k+1}^{m} = \{x_{k+1}, \hat{u}_{k+1}^{m}, \ldots, \hat{u}_{N-1}^{m}, \hat{x}_{N}^{m}\}$$

- And corresponding cost: $C(\hat{T}_{k+1}^{m})$

- And the approximate function $\tilde{J}_{k+1}(x_{k+1})$ is given by:

$$\tilde{J}_{k+1}(x_{k+1}) = \min_{m}\{C(\hat{T}_{k+1}^{m})\}$$

- And it follows that if every base policy is sequentially improving than the *superheuristic*, that produces the cost $\tilde{J}_{k+1}(x_{k+1})$ given x_{k+1} is also sequentially improving.

Deterministic Tree Search

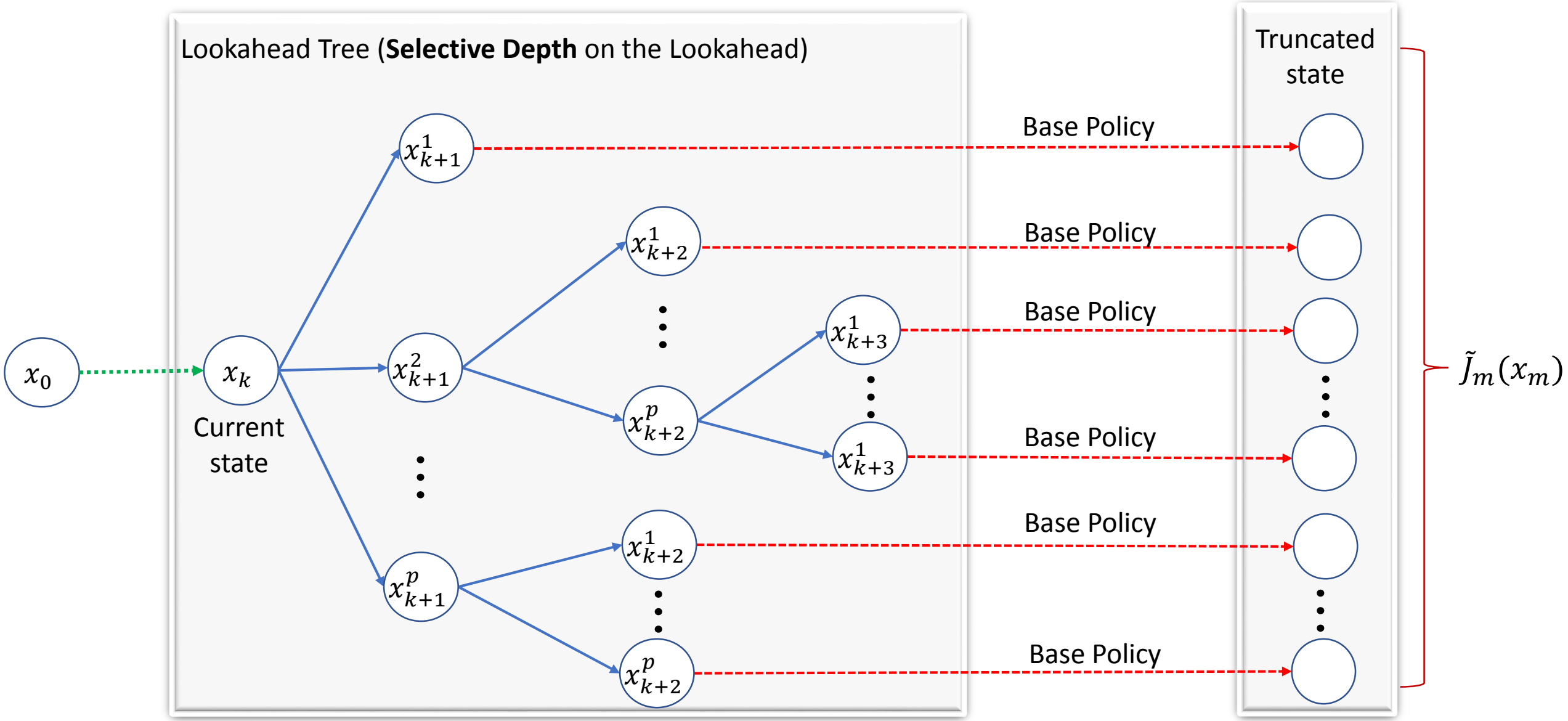
- Now let's combine everything together!
  - Multistep lookahead
  - Rollout Policy
  - Truncation with terminal cost approximation

- The idea is to use a multistep lookahead model where we selectively use base policies to provide approximations for the value functions.

- If the base policies are sequentially improving, then the policy that comes out of the lookahead problem equipped with the rollout policy will achieve policy improvement.

- This procedure leads to a *Tree Search Algorithm*, which is a powerful approximation method (specially when the base policies are Deep Neural Networks!)

Deterministic Tree Search



Deterministic Tree Search

- This formulation already exposes a central idea in approximate DP:
  - **Exploration vs Exploitation**
  - The idea of the Tree Search algorithm is to perform a selective depth lookahead
  - Pruning vs Expansion of the tree nodes.

- For example, we can prune a node at some state x_{k+1} is some score function $S_{k+1}(x_{k+1})$ falls below some threshold α_{k+1}.

- We can use the 1-step lookahead problem equipped with the Base Policy to provide the scoring function:

$$S_{k+1}(x_{k+1}) = \min_{u_{k+1} \in U_{k+1}(x_k+1)} \left\{ g_{k+1}(x_{k+1}, u_{k+1}) + \tilde{J}_{k+2}(f_{k+1}(x_{k+1}, u_{k+1})) \right\}$$

$$\tilde{J}_{k+2}(x_{k+2}) = \tilde{J}_m(x_m) + \sum_{i=k+2}^{m-1} g_i(x_i, \hat{\mu}_i(x_i))$$

Rollout Algorithm: Stochastic Case

- On the stochastic case the notion of policy improvement carries out nicely, and we still have:

$$J_{k,\tilde{\pi}}(x_k) \leq \tilde{J}_k(x_k)$$

- But the key question now is how to execute the Base Policy $\hat{\pi}$ in order to obtain the rollout trajectories.

- We do so by using simulation/sampling. Namely starting from x_k, we compute:

$$x_{i+1} = f_i(x_i, \hat{\mu}_i(x_i), w_i), \ \forall i = k+1, \ldots, N-1$$

- For sampled sequences $(w_k, \ldots, w_{N-1})$.

Rollout Algorithm: Stochastic Case

- We can use SAA, to compute the expected Q-factors as follows:

$$\tilde{Q}_k(x_k, u_k) = \mathbb{E}_{w_k}\left[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, \hat{\mu}_k(x_k), w_k))\right]$$

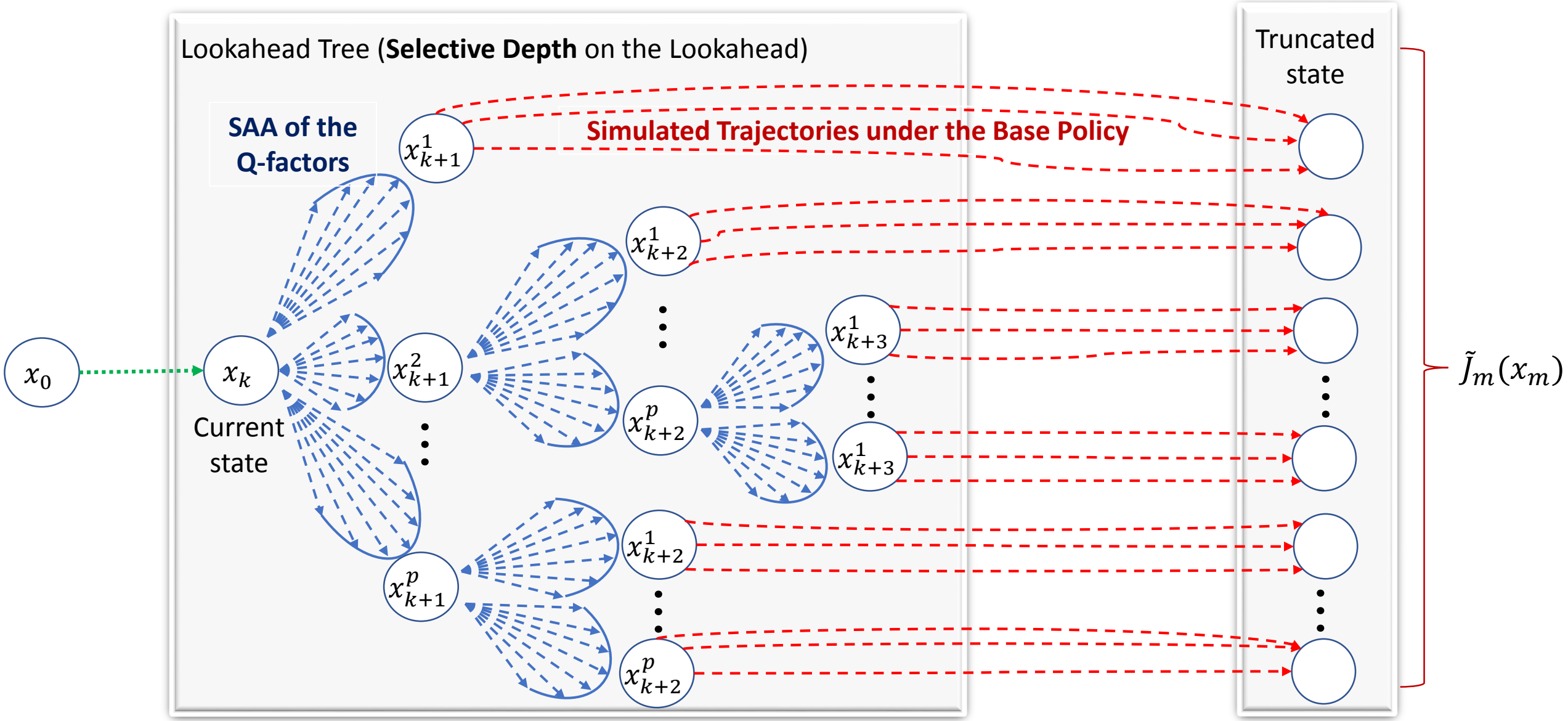
$$\tilde{Q}_k(x_k, u_k) \approx \sum_{s=1}^{S} r_s\left(g_k(x_k, u_k, w_k^s) + \tilde{J}_{k+1}(f_k(x_k, \hat{\mu}_k(x_k), w_k^s))\right)$$

- Where S is the number of samples.

- Then the **rollout policy** becomes:

$$\tilde{\mu}_k(x_k) \in \min_{u_k \in U_k(x_k)} \left\{ \sum_{s=1}^{S} r_s\left(g_k(x_k, u_k, w_k^s) + \tilde{J}_{k+1}(f_k(x_k, \hat{\mu}_k(x_k), w_k^s))\right)\right\}$$

Monte-Carlo Tree Search (MCTS)



Monte-Carlo Tree Search (MCTS)

- The Monte-Carlo Tree Search is a very powerful and flexible framework to perform Approximations in the Value Space.

- With the implementation of such schemes, several questions can be raised:
  - (1) How can we score good control options now? **Exploration vs Exploitation**
  - (2) How to obtain reasonable terminal costs at the truncation states?
  - (3) How can we efficiently obtain samples?
  - (4) How to avoid performing so much simulation online?
  - (5) How can we “control” the simulation error? What about the Variance?
  - (6) What would be a good Base Policy?

- Successful cases, such as AlphaGo, provide answers to the above questions. We will continue next lecture, with potential answers!