Supplementary for "P-MDP: A Framework to Optimize NFPs of Business Processes in Uncertain Environments"

Anonymous

Abstract

This technically supplementary document elaborates on P-MDP design and implementation.

Contents

1 Details for Explanation of Hierarchy Stack

This section is a supplement for Sect. 2 of the submitted paper.

Process models often contain nested concurrent blocks (initiated by split parallel gateways, `spg`) with multiple concurrent flows. To manage these structures, ensuring flow separation and a consistent hierarchical context for all vertices (including `ste` in an implicit root block), the hierarchy stack H_v tracks the position of each vertex in the concurrency hierarchy.

Hierarchy Stack H_v: This stack, whose entries consist of `spg` vertices (split parallel gateways) and their concurrency details, tracks the concurrency hierarchy for each vertex v in the process model. Each entry records:

- The split parallel gateway vertex v_{spg} marking the start of the concurrent block containing the current vertex v
- The concurrent flow index k of the current vertex v within this block
- The total number of concurrent flows n created by the split at v_{spg}

The stack is structured as an ordered sequence where the top element corresponds to the innermost concurrent block, with subsequent elements denoting outer blocks (ordered leftward as blocks progress outward). This structure enables the precise tracking of nested parallel execution paths.

1.1 Initialization

For the start event `ste` of the process model, the hierarchy stack is initialized as:

$$H_{ste} \leftarrow \text{Push}(\text{Root}, 1, 1)$$

This operation establishes the outermost implicit root block (single-flow), with no active concurrency blocks—aligning with the uniform hierarchical context for vertices like `ste` that lack explicit concurrency blocks.

1.2 Construction Rules

For an edge from vertex v_i to vertex v_j in the process model, the hierarchy stack H_{v_j} is constructed based on the types of v_i and v_j:

1. **Inheritance Rule (IR)**: If $T_{v_i} \neq \text{spg}$ and $T_{v_j} \neq \text{mpg}$, then:

$$H_{v_j} \leftarrow H_{v_i}$$

The hierarchy stack remains unchanged when transitioning between non-splitting and non-merging vertices.

2. **Split Rule (SR)**: If $T_{v_i} = \text{spg}$, indicating a split into n concurrent flows:

$$H_{v_j} \leftarrow H_{v_i} \quad \text{followed by} \quad \text{Push}(v_i, k, n)$$

Here, k is the flow index of v_j within the n flows created by v_i. This operation adds a new entry to the stack, reflecting the start of a new concurrency block.

3. **Merge Rule (MR)**: If $T_{v_j} = \text{mpg}$, indicating a merge of n concurrent flows:

$$H_{v_j} \leftarrow H_{v_i} \quad \text{followed by} \quad H_{v_j} \leftarrow \text{Pop}(H_{v_j})$$

Here, v_i is *any* predecessor (all share identical stack states), and a single Pop suffices to end the innermost block, yielding the correct stack state H_{v_j}.

1.3 Illustrative Construction Example

Consider the graph representation of a travel agency process shown in Fig. A1, which is reproduced from Fig. 3 in the submitted paper.

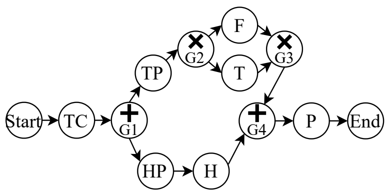


Figure A1: Graph Representation of the Travel Agency Process

The construction of each vertex's H_v through key transitions is as follows:

1. **Start Event** (`ste`): Initialization via Inheritance Rule:$H_{\texttt{ste}} = [(\text{Root}, 1, 1)]$
2. **Transition** `ste` $\rightarrow$ `TC`: Inheritance Rule for `ste` $\rightarrow$ `oth`:$H_{\texttt{TC}} = [(\text{Root}, 1, 1)]$
3. **Transition** `TC` $\rightarrow$ `G1`: Inheritance Rule for `oth` $\rightarrow$ `spg`:$H_{\texttt{G1}} = [(\text{Root}, 1, 1)]$

4. **Split at $\texttt{G1}$ (spg)**: Concurrent flows spawned via Split Rule (SR):

$$H_{\texttt{TP}} = [(\text{Root}, 1, 1), (\texttt{G1}, 1, 2)] \quad \text{(Concurrent flow 1: Transportation)}$$
$$H_{\texttt{HP}} = [(\text{Root}, 1, 1), (\texttt{G1}, 2, 2)] \quad \text{(Concurrent flow 2: Accommodation)}$$

5. **Transitions Between $G1$ and $G4$**: Each intermediate vertex inherits its predecessor's stack via the Inheritance Rule.

6. **Merge at $\texttt{G4}$ (mpg)**: Stack reduction via Merge Rule (MR):$H_{\texttt{G4}} = [(\text{Root}, 1, 1)]$

7. **Later Transitions**: Subsequent vertices retain their respective stacks via the Inheritance Rule.

2 Details for Formalization of Temporal and Logical Dependencies

This section is a supplement for Sect. 2 of the submitted paper.

In process graph $G(V, E)$, sequence flows define execution order (temporal dependencies), while gateways introduce exclusive/concurrent (logical dependencies). These dependencies determine which preceding vertices' states are observable to a vertex v—a relationship formalized by the observability set O (capturing feasible paths and their variables). This formalization is critical for isolating concurrent flows (via H_v) – especially in handling concurrency – and simplifying P-MDP environment construction.

Temporal and Logical Dependencies of Vertices. Such dependencies, induced by sequence flows and gateways in $G(V, E)$, enable each vertex v to observe the states resulting from what happened before it along its execution path. To formalize this observability, we define the following constructs:

2.1 Observability Sets

Let $O = \{\, o_v \mid v \in V\}$ denote the collection of observability sets for all vertices, where $o_v = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ for each v. Each path $\sigma \in o_v$ includes all variables observable at the vertices along that path—specifically, from the first vertex of both v's innermost concurrency block and its concurrent flow, up to v itself. Exclusive gateways may generate multiple paths, so m equals the number of topologically feasible paths leading to v.

To construct O via topological traversal of the graph, each o_{v_j} is defined recursively based on its predecessor vertices v_i, following rules categorized by vertex type:

$$o_{v_j} = \left\{ \begin{array}{lll} \emptyset & \text{if } T_{v_j} = \text{ste} & \text{(1a)} \\ \mathfrak{o}_{v_i} & \text{if } T_{v_j} \in \{\text{oth}, \text{seg}, \text{spg}\} \text{ and } T_{v_i} \neq \text{spg} & \text{(1b)} \\ \emptyset & \text{if } T_{v_j} \in \{\text{oth}, \text{seg}, \text{spg}\} \text{ and } T_{v_i} = \text{spg} & \text{(1c)} \\ \displaystyle\bigcup_{arc_{i,j} \in \varepsilon} \mathfrak{o}_{v_i} & \text{if } T_{v_j} = \text{meg} & \text{(1d)} \\ o_{v_h} \otimes \left(\coprod_{arc_{i,j} \in \varepsilon} \mathfrak{o}_{v_i} \right) & \text{if } T_{v_j} = \text{mpg} & \text{(1e)} \end{array} \right.$$

where:

- $\mathfrak{o}_{v_i} \triangleq \{\sigma \cup C_{v_i} \cup U_{v_i} \mid \sigma \in o_{v_i}\}$, representing the observable states after executing v_i
- ε denotes the set of incoming *arc*s to v_j
- v_h is the split parallel gateway (`spg`) vertex of the innermost concurrency block, obtained via $\text{GetTop}(H_{v_j})$
- $\otimes$ denotes the unordered Cartesian product over execution paths
- $\coprod_{arc_{i,j} \in \varepsilon} \mathfrak{o}_{v_i} \triangleq \mathfrak{o}_{v_1} \otimes \mathfrak{o}_{v_2} \otimes \cdots \otimes \mathfrak{o}_{v_n}$ for incoming edges $\varepsilon = \{arc_{1,j}, arc_{2,j}, \ldots, arc_{n,j}\}$

2.2 Rule Interpretation

1. **Start vertex (Eq.1a)**:
   - This vertex has no preceding vertices, so $o_{v_j} = \emptyset$.
   - Example: The initial vertex of a process model.
2. **Vertices Inheriting and Extending Observability** (Eq.1b):
   - For vertices v_j of type `oth`, `seg`, or `spg` with a non-`spg` predecessor v_i, o_{v_j} extends v_i's observability by appending v_i's variables (C_{v_i}, U_{v_i}) to all paths in o_{v_i}.
   - Example: A task following another task in a sequence flow, extending the prior task's observable variables.
3. **Vertices Initializing Concurrent Flows** (Eq.1c):
   - For vertices v_j of type `oth`, `seg`, or `spg` with a `spg` predecessor v_i, it starts a new concurrent flow in the concurrency block, so $o_{v_j} = \emptyset$ to isolate concurrency hierarchy.
   - Example: The first activity after a parallel split gateway (`spg`), initiating a new concurrent flow in the concurrency block.
4. **Vertices Merging Exclusive Paths** (Eq.1d):

- These vertices consolidate paths from incoming exclusive paths. o_{v_j} is formed by taking the union of extended observability sets from all predecessors v_i ($\bigcup_{arc_{i,j} \in \varepsilon} \mathtt{o}_{v_i}$).
- Example: An exclusive merge gateway (`meg`) unifying paths from an earlier exclusive split, aggregating observable states from all paths.

5. **Vertices Synchronizing Concurrent Flows** (Eq.1e):

   - For vertices v_j of type `mpg`:

     Preprocessing. Insert n empty `oth` vertices—each dedicated to a concurrent flow and positioned between that flow's end and v_j—to buffer timing differences (arising from asynchronous execution of concurrent flows) via self-transitions, a mechanism in P-MDP. This ensures v_j synchronizes only after all n flows complete (as specified in the submitted paper's Synchronization Transition $\mathcal{SP}$).

     Observability. o_{v_j} integrates two components:

     – Cross-tier inherited observability: The `spg` vertex o_{v_h} – co-tier with the v_j – retains state information from v_h's tier, ensuring cross-tier continuity of observability.

     – Intra-tier aggregated observability: $\coprod_{arc_{i,j} \in \varepsilon} \mathtt{o}_{v_i}$ aggregates intra-tier flow observability, respecting sub-flow isolation.

   - Example: An `mpg` (v_j) synchronizing two flows split by a prior `spg` (v_h):

     – Empty `oth` vertices delay v_j until both flows complete, preventing premature synchronization.

     – o_{v_j} unifies cross-tier and intra-tier observability to enable consistent synchronization.

2.3 Mechanisms for gateway-induced complexity

Leveraging the observability set O simplifies the construction of P-MDP environments under gateway-induced complexity:

- **Hierarchical Scope Isolation**: Each $o_v \in O$ is confined to the innermost concurrent block of v's flow, enabling hierarchical observability isolation and avoiding cross-flow variable tracking via path-based dependencies.
- **Modeling and Handling Execution Order Uncertainty**: The uncertainty of execution order across concurrent flows is explicitly modeled via the transition parameter ρ in the ordinary transition of P-MDP $\mathcal{SP}$ and handled via the multidimensional state awareness of the framework.

3 Details for Basic Operator System for User-defined NFPs

This section is a supplement for Sect. 2 of the submitted paper.

In P-MDP, we introduce a flexible operator system to formally define and compute user-defined non-functional properties (NFPs) across different business scenarios. This system allows users to express complex metrics and constraints through a combination of arithmetic, logical, and optimization operators.

3.1 Operator Precedence and Associativity

The operator system is structured with well-defined precedence levels and associativity rules, ensuring consistent interpretation of complex expressions. **Table A1** summarizes the basic operators, their precedence, associativity, and application categories.

Table A1: Priority and Associativity of Operators in P-MDP. M = Metric, HC = Hard Constraint, SC = Soft Constraint.

Priority	**Operator**	**Assoc.**	**Description**	**Category**
0	( )	Left	Parentheses	M, HC, SC
1	[,]	Left	Vector Parentheses	M
2	!	Right	Logical Not	HC, SC
2	mm	Left	Dynamic max-min normalization	M
2	max x	Right	Maximize x	SC
2	min x	Right	Minimize x	SC
2	RB(s)	Right	Maximum in Set s	M
2	LB(s)	Right	Minimum in Set s	M
2	log	Right	Logarithm	M
2	abs(x)	Right	Absolute Value of x	M
2	$\sqrt[n]{x}$	Right	n-th root of x	M
3	$\times$	Left	Multiplication	M
3	$\div$	Left	Division	M
3	$\cdot$	Left	Vector Dot Product	M
4	$+$	Left	Addition	M
4	$-$	Left	Subtraction	M
5	$>$	Left	Greater Than	HC, SC
5	$\geq$	Left	Greater/Equal	HC, SC
5	$<$	Left	Less Than	HC, SC
5	$\leq$	Left	Less/Equal	HC, SC
6	$=$	Left	Equal To	HC, SC
6	$\neq$	Left	Not Equal	HC, SC
7	$\wedge$	Left	Logical AND	HC, SC
8	$\vee$	Left	Logical OR	HC, SC
9	,	Left	Comma Operator	M

3.2 Operator Categories and Applications

Operators are categorized based on their primary use cases:

3.2.1 Metrics (M)

Metrics are modeled as multivariate functions using operators such as:

- Arithmetic operators (e.g., $+$, $-$, $\times$, $\div$) for basic computations
- Statistical functions (e.g., RB, LB, log, abs) for data aggregation
- Max-min Normalization operators (mm) for scaling metrics

3.2.2 Hard Constraints (HC)

Hard constraints define mandatory business rules using operators such as:

- Relational operators (e.g., $>$, $\geq$, $<$, $\leq$, $=$, $\neq$) for value comparisons
- Logical operators (e.g., $\wedge$, $\vee$, !) for combining conditions

3.2.3 Soft Constraints (SC)

Soft constraints represent optimizable objectives using operators such as:

- Optimization operators (max, min) for defining goals
- Logical and relational operators for formulating preferences

3.3 Extensibility and Customization

P-MDP's operator system is extensible, with new operators integrated by updating solely the parsing layer – aligning with its support for user-defined multivariate functions (*udm*) and constraints (*udc*). Users can define new operators via plugins to compose complex *udm* and *udc* for novel business rules and scenarios. This is enabled by *the constraint-oriented lazy reward mechanism*, which defers constraint evaluation until relevant variables are assigned, allowing seamless integration of extended operators without modifying the core components.

4 Formal Definition of P-MDP Horizon Space

This section is a supplement for Sect. 3.1 of the submitted paper.

Each P-MDP time-step corresponds to a configuration of currently active vertices (i.e., which vertices are active as the process enacts) during decision-making, each defining the state for that time-step. These configurations – capturing sequence, parallelism, exclusive choices, and nesting – collectively form the horizon space $\mathcal{H}$.

Horizon Space of P-MDP. The horizon space $\mathcal{H}$ of P-MDP is defined as the set of all active-vertex configurations in the process graph $G(V, E)$. Formally:

$$\mathcal{H} = \{\mathbb{v} \mid \mathbb{v} \text{ is an active-vertex configuration in } G\}$$

An active-vertex set $\mathbb{v}$ can take the following forms:

1. A singleton $\{v\}$ representing a sequential step,
2. A set $\{v_1, v_2, \ldots, v_k\}$ representing concurrent flows,
3. A singleton chosen from alternatives (exclusive choice),
4. Nested combinations of the above forms.

These configurations define the time-steps of P-MDP, capturing sequence, concurrency, choice, and nesting structures of business processes. The horizon space is organized using the Horizon Process Structure Tree (HPST), a variant of the Refined Process Structure Tree (RPST) [6] where:

- Leaf nodes are replaced by vertices in G;
- Non-leaf nodes represent sequence, parallel (AND), and exclusive (XOR) blocks.

4.1 Horizon Process Structure Tree (HPST)

The HPST organizes vertices into a hierarchical structure, enabling the definition of finite horizon spaces. Figure A2 shows an example HPST for a travel agency process, and Table A2 summarizes the construction rules for horizon space nodes.

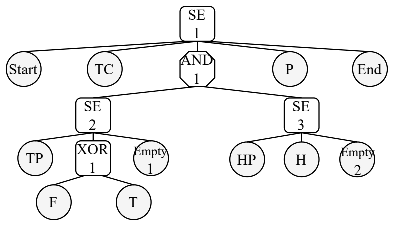


Figure A2: The HPST of the travel agency process in Fig. 2b of the submitted paper.

4.2 Key Properties of $\mathcal{H}$

- **Finite Horizon**: Since the process graph G has a limited number of vertices, the horizon space $\mathcal{H}$ is finite.

Table A2: Horizon Space Rules in HPST. SM_i denotes splitting and merging vertices of the gateway

Node Class	Time-steps
$\textcircled{v}$ Leaf	$\{v\}$
$\boxed{\overset{\text{SE}}{i}}$ SEQUENCE	$\bigcup_{\mathbb{v} \in \text{SE}_i} \mathbb{v}$
$\boxed{\overset{\text{XOR}}{i}}$ XOR	$\left(\bigcup_{\mathbb{v} \in \text{XOR}_i} \mathbb{v}\right) \cup SM_i$
$\boxed{\overset{\text{AND}}{i}}$ AND	$\left(\coprod_{\mathbb{v} \in \text{AND}_i} \mathbb{v}\right) \cup SM_i$

- **Hierarchical Organization**: Vertices are organized by hierarchy, with nested structures represented through the HPST.
- **Terminal States**: The *Start* vertex marks the initial state, and the *End* vertex marks the terminal state, with all others being intermediate states.

The HPST-based horizon space allows P-MDP to model both the temporal and logical dependencies in business processes, providing a structured framework for modeling the time-step of P-MDP.

5 Formal Definition of Prioritized Noisy Double Deep Q-Networks with Multi-Step Learning for P-MDP

This section is a supplement for Sect. 3.2 of the submitted paper.

Here we provide additional details on how we integrate P-MDP into a tailored Deep Q-Network (DQN) [3] architecture – with the overall structure shown in Fig. A3 – as well as the specific techniques (multi-step learning [5], Double DQN [2], Prioritized Experience Replay (PER) [4], and Noisy Nets [1]) used to address high-dimensional action space and delayed rewards.

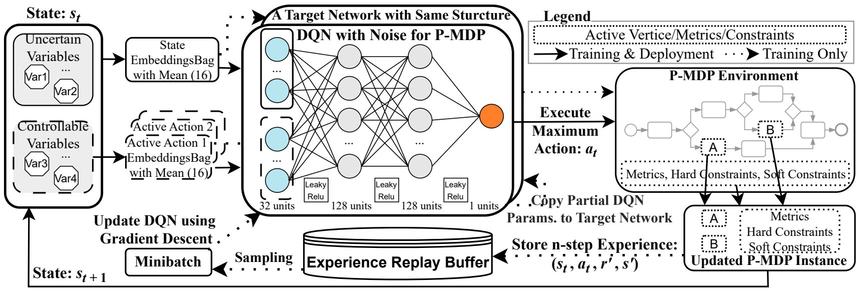


Figure A3: The integrated architecture of Prioritized Noisy Double DQN with multi-step learning for P-MDP in Fig. 4 of the submitted paper.

5.1 Architecture Overview

As shown in Fig. A3, our DQN takes as input both the NFP state embedding and an *active-action mask* — which encodes the set of feasible actions (i.e., active controllable variables) at each time-step. This allows us to:

- **Factorize the action space**: To address the exponential growth of the action space with controllable variables, we decompose the global action space into per-step sub-spaces. This allows the DQN to output scalar Q-values for active actions, rather than a monolithic joint action.
- **Embed active-action masks into the state**: The DQN takes as input both the NFP state embedding and an *active-action mask*, which encodes the set of feasible actions at each time-step, ensuring only valid variables are considered in Q-value calculation.

5.2 Accelerating Delayed Reward Propagation via Multi-step Learning

P-MDP's constraint-oriented lazy reward mechanism – where each constraint is only evaluated upon full assignment – gives rise to a credit-assignment challenge: delayed rewards make it difficult to associate early actions with their long-term consequences. To mitigate this, we adopt the *forward-view* n-step return:

$$r' = \sum_{k=0}^{n-1} \gamma^k r_{t+k}$$

as a key component of the TD target. This n-step return, when integrated into the target calculation (see Eq. 3), accelerates the propagation of delayed rewards back to earlier time-steps.

5.3 Reducing Bias with Double DQN

To mitigate the overestimation inherent in Q-learning, we maintain two networks:

$$\begin{cases} \text{Online network: } Q(s,a;\theta), \\ \text{Target network: } Q(s,a;\theta^-), \end{cases} \tag{2}$$

and compute the n-step TD error

$$\delta_{t:n} = Q(s_t,a_t;\theta) - \underbrace{\Big(r' + \gamma^n\, Q\big(s_{t+n},\, \arg\max_{a'} Q(s_{t+n},a';\theta)\,;\,\theta^-\big)\Big)}_{\text{TD target}}. \tag{3}$$

Here, the underlined term in Eq. (3) serves as the TD target, which combines the n-step return r' with the target network's estimate of future Q-values to stabilize the learning process. The target parameters θ^- are updated via a soft-update rule:

$$\theta^- \leftarrow \tau\,\theta + (1-\tau)\,\theta^- \tag{4}$$

with a hyperparameter $\tau \in (0, 1)$ (typically set to 0.003) that controls the blending of the online and target network parameters.

5.4 Prioritizing Infrequent but Critical Experiences via PER

Delayed rewards often manifest in infrequent yet critical experiences (e.g., constraint violations or satisfactions after multi-step actions) – easily overshadowed in uniform sampling. Initially, these experiences exhibit high TD-errors; as learning progresses, their influence propagates backward, leading earlier related experiences to also exhibit high TD-errors.

In PER, n-step tuples (s_t, a_t, r', s_{t+n}) are stored in an experience buffer, with sampling probabilities dynamically adjusted based on TD errors. The normalized sampling probability is computed as:

$$P(i) = \frac{|\delta^i|^\omega + \epsilon}{\sum_{k=1}^{N} (|\delta^k|^\omega + \epsilon)} \tag{5}$$

where δ^i denotes the TD error of the i-th experience, calculated as the difference between the predicted Q-value and the target Q-value (see Eq. (8)), $\omega \in [0, 1]$ controls prioritization strength (0 for uniform sampling, 1 for fully prioritized), and the small constant $\epsilon > 0$ prevents samples with zero error from being ignored.

To correct for non-uniform sampling bias, PER applies importance-sampling weights:

$$\alpha^i = \alpha \frac{(P(i)N)^{-\beta}}{\max_k ((P(k)N)^{-\beta})} \tag{6}$$

where α is the global learning rate, α^i denotes the adjusted learning rate for sample i, $\beta \in [0, 1]$ increases during training to gradually reduce bias, and N denotes the current number of samples in the buffer.

By prioritizing experiences with high TD-errors – instead of uniformly sampling all experiences – PER enhances the learning efficiency of infrequent yet critical delayed-reward experiences.

5.5 Adaptive Exploration via Noisy Nets

To avoid hand-tuning an ε-greedy schedule and encourage better exploration,we adopt the Noisy Nets, which replaces deterministic network parameters with learnable noise:

$$\widetilde{\theta} = \theta + \theta_{\text{noisy}}, \quad \theta_{\text{noisy}} = W_{\text{noisy}} \odot \epsilon^w \, x \; + \; b_{\text{noisy}} \odot \epsilon^b, \tag{7}$$

where $\epsilon^w, \epsilon^b \sim \mathcal{N}(0, 1)$ and $\odot$ is element-wise multiplication. By default, the noise controllers W_{noisy} and b_{noisy} are initialized to 0.017. These noise parameters are jointly trained through gradient descent, thereby yielding an adaptive exploration–exploitation trade-off.

5.6 Loss Function and Parameter Update

The loss function $L(\widetilde{\theta})$ based on the n-step TD error $\delta_{t:n}$ (see Eq. (3)):

$$L(\widetilde{\theta}) = \tfrac{1}{2}\big[\,Q(s_t, a_t; \widetilde{\theta}) \;-\; (r' + \gamma^n\, Q(s_{t+n}, a^*; \widetilde{\theta}^-))\big]^2 = \tfrac{1}{2}\,\delta_{t:n}^2 \tag{8}$$

where $a^* = \arg\max_{a'} Q(s_{t+n}, a'; \widetilde{\theta})$.

The online network parameters $\widetilde{\theta}$ are updated via gradient descent with importance-weighted learning rates:

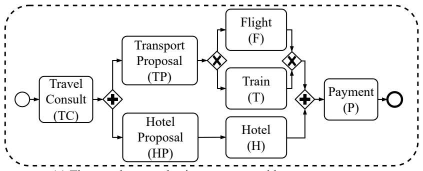
$$\widetilde{\theta} \;\leftarrow\; \widetilde{\theta} \;-\; \alpha^i\; \delta_{t:n}\; \nabla_{\widetilde{\theta}}\, Q(s_t, a_t; \widetilde{\theta}) \tag{9}$$

where α^i is the importance-sampled learning rate defined in Eq. (6). This weighting corrects for non-uniform sampling bias by scaling each sample's gradient contribution proportionally to its importance.

6 Details for the Travel Agency Experiment Example

This section is a supplement for Sect. 4.2 of the submitted paper.

The TravelAgency example, a simplified business process with gateway patterns, simulates dynamic decision-making dilemmas in travel agencies' service composition under uncertainty. Its core elements, expanded from the submitted paper, are detailed below with reference to Fig. A4.



(a) The travel agency business process with gateway patterns

Type \ Vertex	TC	TP	HP	F	T	H	P
Controllable Variable Set (*C*)	DepartureDate (***D***), CheckInDate(***CI***)	ReturnDate (***R***), TransportSelector Vertcor (***TV***)	CheckOutDate(***CO***)	Flight No. (***F***)	Train No. (***T***)	Hotel No. (***H***)	Extra Profit (***EP***)
Uncertain Variable Set (*U*)	UserBudget (***UB***)	$\emptyset$	$\emptyset$	FlightPrice~ (***FP***), FlyingTime~(***FT***)	TrainPrice~ (***TP***), RunningTime~ (***RT***)	HotelPrice~ (***HP***), HotelClass$^+$ (***HC***)	ServiceFee (***SF***), PreferenceVerctor (***PV***)

(b) Variables in the business process and their associated activities, where certain variables with superscripts, ~ indicates the smaller the better and + the opposite.

Metrics
- $TripDays := R - D$
- $TransportCost := [FP, TP] \cdot TV$
- $TotalCost := (TransportCost + HP + SF) \times EP$
- $TotalCostRatio := TransportCost \div TotalCost$
- $TransportQoS := [FP^{mm} + FT^{mm}, TP^{mm} + RT^{mm}] \cdot TV$
- $TravelQoS := [TransportQoS, HP^{mm} + HC^{mm}] \cdot PV$

Hard Constraints
- $hc_1 := 7 \leq TripDays \leq 10$
- $hc_2 := TransportCostRatio \leq 0.2$
- $hc_3 := 4 \leq HC$
- $hc_4 := D \leq 5$
- $hc_5 := D = CI \wedge R = CO$

Soft Constraints:
- $sc_1 := TotalCost \leq UB$
- $sc_2 := \max\ TravelQoS$
- $sc_3 := \max\ EP$

(c) User-defined NFPs for the business process, where certain variables in metrics with ***mm*** indicating dynamic max-min normalization

Figure A4: A detailed example of the travel agency business process in the Fig. 2b of the submitted paper, including functional properties (a) and user-defined NFPs (b-c). Panel (b) illustrates the topological relationship between variables and their corresponding activities.

6.1 Process Structure and Functional Properties (FPs)

The process is modeled as a sequence of interconnected activities (e.g., realized through web services) and gateways (e.g., parallel or exclusive gateways), as shown in Fig. A4a. These form the **functional properties (FPs)** of the process, defining the basic workflow required to achieve the business goal: assisting customers with travel arrangements (e.g., booking transportation, reserving hotels, and coordinating itineraries).

6.2 User-defined NFPs

Beyond the structural FPs, the process involves a series of **user-defined metrics** that capture quality, optimization objectives, and variables influencing decision-making, along with a set of constraints tied to these metrics (Fig. A4b-c). These constraints, as practical business requirements, are categorized as follows:

- **User-defined Hard constraints** (must be satisfied):
  - hc_1: The trip should last between 7 and 10 days.
  - hc_2: The Transportation cost should not exceed 30% of total costs to ensure the comfort of the hotel.
  - hc_3: Hotel must be 4-star or higher.
  - hc_4: The trip must depart within 5 days.
  - hc_5: The hotel reservation dates must align with the transportation dates.
- **User-defined Soft constraints** (to be optimized as much as possible):
  - sc_1: Keep the total cost within budget.
  - sc_2: Maximize travel quality (e.g., shorter transit times, higher service ratings).
  - sc_3: Maximize additional profit for the agency (e.g., from commissions/service fee on hotel bookings or transportation reservations).

These NFPs are optimized based on variables such as real-time service prices, qualities, and transit times. Notably, some of these variables (e.g., price fluctuations for hotels and flights, flight/train delays affecting transit times) model environmental dynamics—their values are uncertain in each execution and dynamically instantiated during process enactment, directly impacting constraint satisfaction.

6.3 Uncertainty Sources and Their Dynamic Impacts

Real-world scenarios vary in requirements, leading to distinct types of NFPs – this form of uncertainty is addressed through user-defined NFPs tailored to specific needs. Beyond this, inherent uncertainties introduce volatility into the NFPs during execution, requiring adaptive decision-making during process enactment:

- **NFP's runtime uncertainty from the external environment**: For example, fluctuations in fuel prices, holiday demand surges, or sudden changes in supplier pricing (e.g., weekend premiums or coupon discounts) directly affect sc_1 (cost) and hc_3 (hotel availability).

- **FP's structural uncertainty induced by gateways**: This uncertainty arises from their parallel and exclusive behavioral patterns.

  For parallel gateways, they initiate concurrent flows (e.g., booking transportation and accommodation) with no fixed execution order between activities of different flows. Simply handling these flows via serialization leads to issues: prioritizing one (e.g., booking a flight) may limit the other (e.g., narrowing hotel choices, violating hc_3), while prioritizing the other (e.g., hotel bookings) may raise costs (violating sc_1) – and such serialization disrupts original concurrent behavior (violating FPs).

  For exclusive gateways, which involve selective execution of alternative branches (e.g., choosing between flights or trains for transportation), uncertainty arises from unforeseen trade-offs across branches. Isolating each branch for optimization paralyzes global optimality: focusing solely on flight options may overlook cost savings from trains (violating sc_1), while fixating on trains may miss time-sensitive constraints (violating hc_4).

  In both cases, the correct approach is to comprehensively model gateway structures (rather than fragmenting paths) to retain original FPs, and balance trade-offs between constraints to maximize expected satisfaction – thus avoiding the solution space fragmentation of isolated optimization.

6.4 Role of the Travel Agency Experiment Example

The travel agency experiment, supported by its synthetic dataset, serves as a comprehensive testbed to validate the proposed method, encapsulating both gateway structures (FPs) and user-defined NFPs to reflect real-world scenarios.

For FPs, the travel agency process models key structural behaviors (Fig. A4a):

- *Parallel Gateway*: Concurrent flows (e.g., simultaneous booking of transportation and accommodation) are simulated via a 30% self-transition probability ($\rho = 0.3$), enabling temporary pausing of one flow (e.g., during flight checks) while another proceeds (e.g., hotel comparisons) – mimicking real-world unblocked parallelism.

- *Exclusive Gateway*: Alternative branches (e.g., selecting flights vs. trains) are modeled through probabilistic branching from exclusive gateway vertices, with path selection treated as a controllable variable (transport selector, TS) to support adaptive optimization.

For NFPs, the dataset emphasizes user-defined complexity (Fig. A4b-c): 100 records per supplier capture runtime uncertainties (e.g., fluctuating prices, transit times, 15-day temporal dynamics like early bird discounts), aligning with soft objectives (sc_1-sc_3) and hard constraints (hc_1-hc_5) – such as sc_1 (total cost) affected by price fluctuations and hc_4 (depart within 5 days) constrained by early bird discounts. These NFPs are dynamic interactions (e.g., price affecting cost while availability impacts date) mirror real-world trade-offs in user-defined constraints.

In summary, this example and dataset integrate FPs (concurrency, exclusive choices) and user-defined NFPs (runtime uncertainty, interdependent constraints) to validate two core capabilities of the proposed method: (1) handling complex process structures without fragmentation (preserving FPs), and (2) optimizing user-defined NFPs under uncertainty (eliminating assumptions of independence/monotonicity). By aligning with both structural and non-functional dynamics, they demonstrate the method's practical relevance to real-world scenarios.

References

[1] Fortunato, M., et al.: Noisy networks for exploration. In: International Conference on Learning Representations (2018)

[2] Hasselt, H.: Double q-learning. Advances in neural information processing systems **23** (2010)

[3] Hessel, M., et al.: Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the AAAI conference on artificial intelligence. vol. 32 (2018)

[4] Schaul, T., et al.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)

[5] Sutton, R.S., et al.: Reinforcement learning: An introduction. MIT press (2018)

[6] Vanhatalo, J., et al.: The refined process structure tree. Data Knowledge Engineering **68**(9), 793–818 (2009)