Operator Theory-Driven Autoformulation of MDPs for Control of Queueing Systems

Anonymous authors
Paper under double-blind review

Abstract

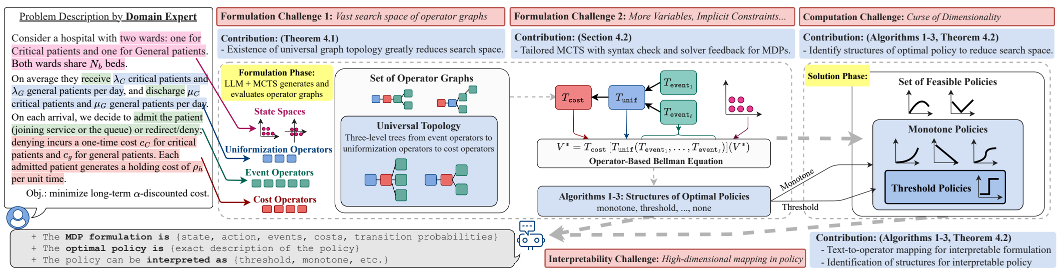
Autoformulation is an emerging field that uses large language models (LLMs) to translate natural-language descriptions of decision-making problems into formal mathematical formulations. Existing works have focused on autoformulating mathematical optimization problems for *one-shot* decision-making. However, many real-world decision-making problems are *sequential*, best modeled as *Markov decision processes* (MDPs). MDPs introduce unique challenges for autoformulation, including a significantly larger formulation search space, and for computing and interpreting the optimal policy. In this work, we address these challenges in the context of queueing problems—central to domains such as healthcare and logistics—which often require substantial technical expertise to formulate correctly. We propose a novel operator-theoretic autoformulation framework using LLMs. Our approach captures the underlying decision structure of queueing problems through constructing the Bellman equation as a graph of *operators*, where each operator is an *interpretable* transformation of the value function corresponding to certain *event* (e.g., arrival, departure, routing). Theoretically, we prove a universal three-level operator-graph topology covering a broad class of MDPs, significantly shrinking the formulation search space. Algorithmically, we propose customized Monte Carlo tree search to build operator graphs while incorporating self-evaluation, solver feedback, and intermediate syntax checking for early assessment, and present a provably low-complexity algorithm that automatically identifies structures of the optimal policy (e.g., threshold-based), accelerating downstream solving. Numerical results demonstrate the effectiveness of our approach in formulating queueing problems and identifying structural results.

1 Introduction

Autoformulation with large language models (LLMs) aims to translate natural-language descriptions of decision-making problems into formal optimization models with minimal human intervention (Zhang et al., 2025b). It democratizes the access to advanced operations research (OR) modeling tools for non-OR domain experts and facilitates rapid prototyping and adaptation for OR practitioners (Gurobi Optimization, 2023; Wasserkrug et al., 2025).

Existing works on autoformulation have been focusing on *mathematical optimization*, which models *one-shot* decision-making (Ramamonjison et al., 2023; Xiao et al., 2023; AhmadiTeshnizi et al., 2024; Astorga et al., 2025; Bertsimas & Margaritis, 2024; Liang et al., 2025; Yang et al., 2025; Lu et al., 2025; Zhang et al., 2025a; Huang et al., 2025). However, many real-world scenarios evolve dynamically and stochastically, thus requiring *sequential* decision-making over time. These problems are naturally modeled as *Markov decision processes* (MDPs) (Puterman, 2014). Autoformulating MDPs presents unique challenges that cannot be addressed by current works on autoformulating optimization problems (see Table 4 and the examples in Appendix A for a more detailed breakdown).

Formulation Challenges. Similar to autoformulating optimization, autoformulating MDPs requires searching the vast space of possible formulations. Moreover, MDPs have *additional components* (e.g., states, transition probabilities) and *implicit constraints* (e.g., nonnegative states, state-dependent action sets) that are not present in optimization and often omitted in the problem description. To ensure the accuracy, autoformulation must identify and infer these hidden structures (e.g., figuring out state transition probabilities of a queue from arrival and service rates).


Problem Description by **Domain Expert**

Consider a hospital with two wards: one for Critical patients and one for General patients. Both wards share N_b beds.

On average they receive λ_C critical patients and λ_G general patients per day, and discharge μ_C critical patients and μ_G general patients per day.

On each arrival, we decide to admit the patient (joining service or the queue) or redirect/deny; denying incurs a one-time cost c_C for critical patients and c_g for general patients. Each admitted patient generates a holding cost of ρ_h per unit time.

Obj.: minimize long-term α-discounted cost.

```
+ The MDP formulation is {state, action, events, costs, transition probabilities}
+ The optimal policy is {exact description of the policy}
+ The policy can be interpreted as {threshold, monotone, etc.}
```

Formulation Challenge 1: *Vast search space of operator graphs*

Contribution: (Theorem 4.1)
- Existence of universal graph topology greatly reduces search space.

Formulation Challenge 2: *More Variables, Implicit Constraints...*

Contribution: (Section 4.2)
- Tailored MCTS with syntax check and solver feedback for MDPs.

Computation Challenge: *Curse of Dimensionality*

Contribution: (Algorithms 1-3, Theorem 4.2)
- Identify structures of optimal policy to reduce search space.

Interpretability Challenge: *High-dimensional mapping in policy*

Contribution: (Algorithms 1-3, Theorem 4.2)
- Text-to-operator mapping for interpretable formulation
- Identification of structures for interpretable policy

Formulation Phase:
LLM + MCTS generates and evaluates operator graphs
State Spaces
Uniformization Operators
Event Operators
Cost Operators
Set of Operator Graphs
Universal Topology
Three-level trees from event operators to uniformization operators to cost operators
T_{cost}
T_{unif}
T_{event_1}
T_{event_ℓ}
$V^* = T_{\text{cost}}\,[T_{\text{unif}}(T_{\text{event}_1}, \ldots, T_{\text{event}_\ell})](V^*)$
Operator-Based Bellman Equation
Algorithms 1-3: Structures of Optimal Policies
monotone, threshold, ..., none
Monotone
Threshold
Solution Phase:
Set of Feasible Policies
Monotone Policies
Threshold Policies

Figure 1: Challenges in formulation, solution, and interpretation, and our contributions in addressing them. ► Text-to-operator mappings improve *accuracy* and the *interpretability of problem formulation*. ► The operator-based Bellman equation reveals the *structures* of the optimal policy (e.g., monotone or threshold) and the value function (e.g., convex), enhancing both *interpretability* and *computational tractability* (by informing the choice of an appropriate solver).

Computational Challenges. Many optimization problems (e.g., convex optimization) are considered solved once formulated (Boyd & Vandenberghe, 2004). In contrast, MDPs are notorious for the *curse of dimensionality* (Puterman, 1994). Although formulating and solving are two distinct phases, we advocate for autoformulation that is amenable to discovering *structural properties* of the optimal policy (e.g., optimal action is monotone in state value) *prior to* the solving phase and therefore mitigating computational challenges (e.g., search among monotone policies instead of all policies).

Interpretability Challenges. Optimization models tend to be more readable, because variables and constraints in the problem formulation often have semantic meanings, and the optimal decision variables are easier to understand. In comparison, the optimal policy of an MDP is a mapping over multi-dimensional state-action pairs, which can be hard to interpret. Therefore, identifying structures of the optimal policy (e.g. monotonicity of action in state) makes the solution interpretable, which is important for decision-support systems (Hajek, 1984; Koole, 1995; Zhou et al., 2015).

Our Solution. Fig. 1 illustrates how our framework addresses the above challenges. Our work builds on the *operator theory* (Koole, 1998; 2007), which views the Bellman equation as a concatenation of *operators*. Each operator is an *interpretable* transformation of the value function that corresponds to certain event (e.g., arrival, departure) in the context of controlling queue systems. The operator-based Bellman equation provides an interpretable problem formulation, as well as theoretical foundations for identifying structures of the optimal policy, which can potentially reduce the complexity of solving MDPs and enhance the interpretability of the solution. We make significant contributions to fulfill the potential of operator theory-driven autoformulation of MDPs. ► *Operator graph and universal topology:* We are the *first* to represent Bellman equations as directed acyclic graphs (DAGs) of *operators*, namely the *operator graph*, and prove the existence of a *universal topology* for a large class of MDPs (Theorem 4.1). This greatly reduces the autoformualtion search space from all possible operator graphs to graphs with the fixed universal topology (**formulation challenge**). ► *Tailored Monte Carlo tree search (MCTS):* We propose a customized MCTS with LLMs to generate and evaluate operator graphs (Sec. 4.2), while incorporating LLM self-evaluation, solver feedback, and intermediate syntax checking to improve the accuracy and efficiency of autoformulation *without expensive fine-tuning of LLMs* (**formulation challenge**). ► *Automatic identification of structures:* We propose a low-complexity algorithm (Algorithms 1–3) that is guaranteed to identify theoretically known structures of the optimal policy (Theorem 4.2), thus mitigating the curse of dimensionality (**computation challenge**) and enhancing solution interpretability (**interpretability challenge**).

Contributions. ① *Conceptually*, we propose an operator theory-driven framework that *for the first time*, jointly automates the formulation of sequential decision-making problems from natural language and the discovery of the structures of the optimal policies (Sec. 4). Our novel view of Bellman equations as operator graphs addresses not only formulation challenges but also computation and interpretability challenges in the formulation phase. ② *Theoretically*, we rigorously prove the existence of a universal operator graph topology for a large class of MDPs, greatly reducing the search space of operator graphs (Theorem 4.1). ③ *Algorithmically*, we tailor MCTS for autoformulation of event-based MDPs by incorporating dense rewards and integrating the feedback from the solver for improved accuracy and efficiency (Sec. 4.2), and propose a provably low-complexity algorithm to automatically uncover structural results from the operator graph (Theorem 4.2). ④ *Empirically*, we create the first dataset on autoformulation of queueing problems, containing natural-language problem descriptions labeled with the optimal policies and their structures, and demonstrate the accuracy and efficiency of our framework.

2 Related Works

Autoformulation of mathematical optimization. There have been considerable efforts in creating datasets containing natural-language description of optimization problems (Ramamonjison et al., 2023; Yang et al., 2025) and developing LLMs and agents fine-tuned for optimization autoformulation (Xiao et al., 2023; AhmadiTeshnizi et al., 2024; Liang et al., 2025; Lu et al., 2025; Zhang et al., 2025a; Huang et al., 2025). Recent works have shown that through prompting and efficient MCTS, open-source LLMs can achieve comparable or better performance without the cost of fine-tuning (Bertsimas & Margaritis, 2024; Astorga et al., 2025). Our framework also uses MCTS without fine-tuning. However, we make significant contributions in addressing the formulation challenges *specific to MDPs*, and computation and interpretability challenges that *these works do not face*.

Autoformulation of dynamic programming. The most related is the recent work on autoformulating dynamic programming problems (Zhou et al., 2025). It focuses on synthetic dataset generation and LLM fine-tuning, but did not consider computation challenges and interpretability challenges.

Addressing computation challenges in MDPs. A large body of research addresses efficient solving of MDPs, notably through approximate dynamic programming (Bertsekas, 2012), reinforcement learning (Sutton, 2018), and exploiting structural properties of the solution (Yang, 2020; Koutas et al., 2025). These approaches are complementary to ours, and can be used in conjunction with our work after structural properties are identified in the formulation phase.

Operator-based Bellman equations for control of queueing systems. Significant OR research is devoted to uncovering structural properties of the optimal solution (Zhuang & Li, 2010; Hsu et al., 2015; Çil et al., 2011). Despite the unifying operator theory framework Koole (1998; 2007), such practices are still on a manual, case-by-case basis. In our attempt to autoformulate MDPs, we are *the first* to view the operator-based Bellman equation as an *operator graph*, and prove the novel result on the existence of a universal graph topology. In addition, we propose a low-complexity algorithm to automate the process of identifying structural results.

Table 1: Comparison with existing works on autoformulation.

Representative work	Problem Formulated	Method	Formulation Challenge	Computation Challenge	Interpretability Challenge
ORLM (Huang et al., 2025)	optimization	fine-tuning	specific to optimization	N/A	N/A
Autoformulator (Astorga et al., 2025)	optimization	prompting	specific to optimization	N/A	N/A
DPLM (Zhou et al., 2025)	(discrete-time) dynamic programming	fine-tuning	✓	×	×
Our Work	discrete-time and continuous-time MDP	prompting	✓	✓	✓

3 Problem Formulation

Our framework autoformulates and solves discrete-time MDPs and continuous-time MDPs (through their equivalent discrete-time *embedded* MDPs). Throughout the paper, we illustrate our framework using examples from healthcare (Chan et al., 2025; Bekker et al., 2017). But our framework can be applied to a variety of applications such as inventory management (Schwarz & Daduna, 2006), logistics (Adelman, 2007), transportation (Stidham, 1985; Ebben et al., 2004), and telecommunication (Koole & Mandelbaum, 2002; Bhulai & Koole, 2003). As shown in Appendix M, our prompts include no contextual information on application domains or even queuing systems in general.

3.1 Preliminaries

Since the results here are established, we provide detailed derivations in Appendix D.1 and a walk-through example in Appendix D.2.

Control of Queuing Systems as Continuous-Time MDPs. We consider a continuous-time MDP specified by six elements (Lippman, 1975; Serfozo, 1979): (1) a countable state space $\mathcal{S}$, (2) a finite set of eligible actions $\mathcal{A}_s$ at each state $s \in \mathcal{S}$, (3) a cost $\hat{c}(s,a)$ incurred at state s when taking action a, (4) the state transition probability $\hat{P}(s'|s,a)$, (5) the *random* transition time τ from state-action pair (s,a) to a different state, which follows an exponential distribution with rate $\lambda(s,a)$, and (6) a discount rate $\alpha \geq 0$ that discounts the cost at time t by $e^{-\alpha t}$.

Optimization Criteria. For a stationary policy $\pi : \mathcal{S} \to \mathcal{A}_s$, the α*-discounted cost* is (Serfozo, 1979)

$$\hat{V}_{\alpha,\pi}(s) \triangleq \mathbb{E}_\pi \left[\textstyle\sum_{i=0}^{\infty} e^{-\alpha t_i} \hat{c}(s_i, a_i) \,|\, s_0 = s\right], \tag{1}$$

where t_i is the time of the i-th state transition. The *average cost* is (Sennott, 2009; Serfozo, 1979)

$$\hat{J}_\pi(s) \triangleq \limsup_{t\to\infty} \mathbb{E}_\pi \left[\textstyle\sum_{i=0}^{I_t} \hat{c}(s_i, a_i)/t \,|\, s_0 = s\right], \tag{2}$$

where $I_t = \max\{i : t_i \leq t\}$ is the number of state transitions that occur in time t.

Discrete-Time Embedded MDPs and Standard Bellman Equations. For an arbitrary upper bound of state transition rates $\Lambda > \sup_{s,a} \lambda(s,a)$, we define a discrete-time MDP with discount factor $\gamma = \frac{\Lambda}{\Lambda+\alpha}$, and state transition probabilities and the cost function as

$$P(s'|s,a) = \begin{cases} \lambda(s,a) \cdot \hat{P}(s'|s,a)/\Lambda, & \text{if } s' \neq s \\ 1 - \lambda(s,a)/\Lambda, & \text{if } s' = s \end{cases} \quad \text{and} \quad c(s,a) = \frac{\lambda(s,a)+\alpha}{\Lambda+\alpha} \cdot \hat{c}(s,a). \tag{3}$$

The discrete-time embedded MDP is obtained by setting a Poisson clock with rate Λ and sampling the continuous-time process when the clock ticks. So the state may remain the same ($s' = s$).

Given a stationary policy π, the γ-discounted cost is $V_{\gamma,\pi}(s) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^n c(s_i, a_i) \,|\, s_0 = s\right]$, and the average cost is $J_\pi(s) = \limsup_{I\to\infty} \mathbb{E}_\pi \left[\sum_{i=0}^{I-1} c(s_i, a_i)/I \,|\, s_0 = s\right]$.

The discrete-time MDP $(\mathcal{S}, \mathcal{A}_s, c, P, \gamma)$ is *equivalent* to the continuous-time MDP $(\mathcal{S}, \mathcal{A}_s, \hat{c}, \tau, \hat{P}, \alpha)$, in the sense that $\hat{V}_{\alpha,\pi}(s) = V_{\gamma,\pi}(s)$ and $J_\pi(s) = \hat{J}_\pi(s)/\Lambda$ for any stationary policy π (Serfozo, 1979). We solve the discrete-time MDP by solving the *standard* Bellman equation:

$$V_{n+1,\gamma}(s) = \min_{a\in\mathcal{A}_s} \left\{c(s,a) + \gamma \cdot \textstyle\sum_{s'\in\mathcal{S}} P(s'|s,a) V_{n,\gamma}(s')\right\}, \tag{4}$$

where $V_{n,\gamma}(s)$ is the *minimum* discounted cost during the last n state transitions when starting from s.

Under mild conditions (Puterman, 2014), the *minimum* discounted cost $V_\gamma(s) = \inf_\pi V_{\gamma,\pi}(s)$ is the limit of $V_{n,\gamma}(s)$ when $n \to \infty$ (Sennott, 2009, Proposition 4.3.1), and the *minimum* average cost $J(s) = \inf_\pi J_\pi(s)$ is the limit of $(1-\gamma)V_\gamma(s)$ when $\gamma \to 1$ (Sennott, 2009, Proposition 6.2.3). Therefore, it suffices to focus on the n-transition discounted cost $V_{n,\gamma}$ in the Bellman equation (4). For the remainder of the paper, we omit the discount factor in the subscript of $V_{n,\gamma}$ and use V_n.

3.2 Event-Based MDP and Operator-Based Bellman Equation

Definition 3.1. *Event-based MDPs* are MDPs whose state $s = (x, e)$ has two components: a controllable component x (e.g., number of patients in the system) and an exogenous, uncontrollable component e (e.g., arrivals), with transition probabilities decomposed as:

$$P\left[(x', e') \,|\, (x, e),\, a\right] = P_{\mathrm{x}}\left[x' \,|\, (x, e),\, a\right] \cdot P_{\mathrm{e}}(e' \,|\, x'). \tag{5}$$

Event-based MDPs are general enough to model decision-making problems in various applications (see Appendix B for a comprehensive list). Many problems in control of queuing systems are special cases of event-based MDPs, where transitions of the queuing state x are deterministic.

Definition 3.2. (Koole, 2007, Definition 3.1) Let $\mathcal{X}$ be the set of controllable state components x and $\mathcal{V}$ be the set of all functions from $\mathcal{X}$ to $\mathbb{R}$. An *operator* is a mapping

$$T : \mathcal{V}^\ell \to \mathcal{V}, \quad \ell \geq 1. \tag{6}$$

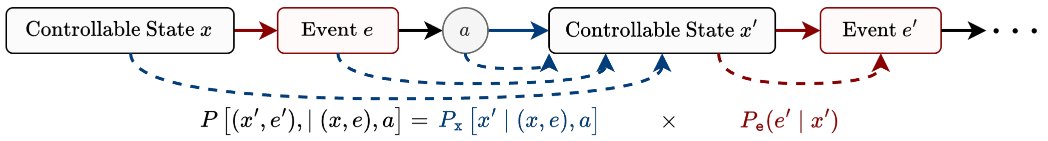

Controllable State x
Event e
a
Controllable State x'
Event e'
$P\left[(x', e'), \mid (x, e), a\right] = P_{\mathrm{x}}\left[x' \mid (x, e), a\right] \times P_{\mathrm{e}}(e' \mid x')$

Figure 2: State transition dynamics of event-based MDPs, where dashed arrows indicate dependency.

The definition of an operator is general, giving us flexibility to express the Bellman equation as concatenation of operators. Given the context, the operators also have physical meaning. For our example introduced in Fig 1, we can define operators related to optimal responses to events, namely the controlled arrivals (CA) and uncontrolled departures (D) for Critical (C) and General (G) patients. The details can be found in Appendix D.2 and Appendix E. In short, we have:

$$T_{\mathtt{CA,C}}\left[V_n^*(x)\right] \quad = V_{n+1}(x, \mathtt{A}_C), \quad T_{\mathtt{CA,G}}\left[V_n^*(x)\right] = V_{n+1}(x, \mathtt{A}_G), \tag{7}$$

$$T_{\mathtt{D,C}}\left[V_n^*(x)\right] \quad = V_{n+1}(x, \mathtt{D}_C), \quad T_{\mathtt{D,G}}\left[V_n^*(x)\right] = V_{n+1}(x, \mathtt{D}_G), \tag{8}$$

with $\mathtt{A}_{C/G}$ and $\mathtt{D}_{C/G}$ corresponding to arrival and departure events for Critical and General patients. We also define a uniformization operator (with $\lambda = \lambda_C + \lambda_G + \mu_C + \mu_G$)

$$T_{\mathtt{unif}}[U_1(x), U_2(x), U_3(x), U_4(x)] = \frac{\lambda_C}{\lambda} \cdot U_1(x) + \frac{\lambda_G}{\lambda} \cdot U_2(x) + \frac{\mu_C}{\lambda} \cdot U_3(x) + \frac{\mu_G}{\lambda} \cdot U_4(x),$$

and a cost operator

$$T_{\mathtt{cost}}\left[U(x)\right] = \rho_h \cdot (x_C + x_G)/(\lambda + \alpha) + \gamma \cdot U(x). \tag{9}$$

Then the Bellman equation (38) can be rewritten as V_n^* going through an *operator graph* to get V_{n+1}^*:

$$V_{n+1}^*(x) = T_{\mathtt{cost}}\left\{T_{\mathtt{unif}}\left(T_{\mathtt{CA,C}}\left[V_n^*(x)\right], T_{\mathtt{CA,G}}\left[V_n^*(x)\right], T_{\mathtt{D,C}}\left[V_n^*(x)\right], T_{\mathtt{D,G}}\left[V_n^*(x)\right]\right)\right\}. \tag{10}$$

Note that for event-based MDPs, we often study the value function $V_n^*(x)$ defined on the controllable state component x, in addition to the standard value function $V_n(s)$ defined on the state s.

4 Method

Our framework has two steps: autoformulation of operator-based Bellman equations and identification of structural results (see Fig. 1 for overview).

4.1 Theoretical Foundation: Universal Topology of Operator Graphs

We can view the Bellman equation (10) as an operator graph with input V_n^* and output V_{n+1}^*. If we view the process of problem formulation as searching in the space of all operator graphs, the search space is vast due to the variety of operators (Koole, 2007) and the many ways they can be connected (i.e., graph topology). Specifically, the number of possible DAGs with N nodes (operators) and 1 out-point (the operator that outputs V_{n+1}^*) grows in the order of 2^{N^2} (Robinson, 1973). To have a sense of how fast this number grows with N, the numbers of possible DAGs for $N = 2, \ldots, 9$ are $1,\ 2,\ 15,\ 316,\ 16885,\ 2174586,\ 654313415,\ 450179768312$.

We prove the existence of a *universal* graph topology for *all* event-based MDPs. This allows us to fix the graph topology, thus significantly reducing the search space.

Theorem 4.1. *For any event-based MDP with event set $\{\mathtt{e}_1, \ldots, \mathtt{e}_\ell\}$, its Bellman equation can be constructed by the following operator graph (the universal topology in Fig. 1):*

$$V_{n+1}^*(x) = T_{cost}\left\{T_{unif}\left(T_{\mathtt{e}_1}\left[V_n^*(x)\right], \ldots, T_{\mathtt{e}_\ell}\left[V_n^*(x)\right]\right)\right\}, \tag{11}$$

where $T_{cost}\left[U(x)\right] = c(x) + \gamma \cdot U(x)$, $T_{unif}\left[U_1(x), \ldots, U_\ell(x)\right] = \sum_{j=1}^{\ell} P(\mathtt{e}_j \,|\, x) \cdot U_j(x)$, and $T_{\mathtt{e}_j}\left[V_n^*(x)\right] = V_{n+1}(x, \mathtt{e}_j)$.

Proof. See Appendix F. □

Theorem 4.1 reduces the search space to *three-level trees with T_{cost} as the root, T_{unif} as the single child of the root, and event operators as leaves*. Hence, in addition to specifying the universal topology, it also reduces the search space by specifying the *types of operators at each level of the tree*.

Takeaway: Theorem 4.1 is crucial for reducing the search space.
The original search space includes all the operator graphs with any topology and any operators as nodes, which is huge. Theorem 4.1 proves that there exists a universal topology: a three-level tree with certain types of operators at each level. This significantly reduces the search space.

4.2 MCTS for Autoformulating Event-Based MDPs.

Now that we know the universal graph topology, we aim to identify the correct nodes of the operator graph. Due to dependencies among components, MCTS is well-suited for this hierarchical search (see Appendix M for prompts). We decompose the search into four layers: m_1 (problem parameters, e.g., queue sizes), m_2 (state variables and constraints), m_3 (events, actions, costs, and their probabilities), and m_4 (operators). This structure reflects the dependency hierarchy and guides exploration. Our MCTS follows the standard loop—*selection*, *expansion*, *evaluation*, and *backpropagation*—with two key modifications during backpropagation: (1) terminal nodes receive rewards from a combination of LLM preference and solver feedback, and (2) intermediate nodes are evaluated for syntax validity to provide dense supervision and penalize early errors.

Terminal Rewards. Every full rollout is scored relative to a baseline defined as the initial rollout. The LLM provides a preference score $\texttt{score}_{\text{LLM}} \in [0, 1]$. To reduce bias from LLM self-evaluation, we incorporate solver convergence $\texttt{score}_{\text{converged}} \in \{0, 1\}$, and compute the final reward as $\texttt{score}_{\text{final}} = \texttt{score}_{\text{LLM}} \times \texttt{score}_{\text{converged}}$.

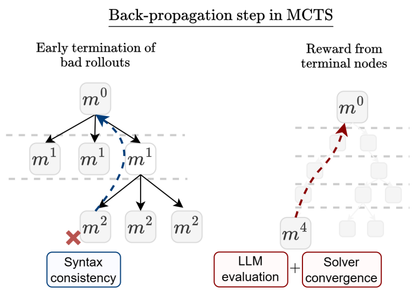


Figure 3: **MCTS for constructing operator graphs.** (1) Syntax check at intermediate nodes detects errors early, preventing failed full rollouts. (2) Solver feedback complements LLM self-evaluation for more objective rewards.

Intermediate Rewards. Inspired by AlphaZero (Silver et al., 2018), we assign rewards to intermediate nodes based on syntactic correctness. If a partial formulation violates syntax constraints, the rollout is terminated early with a zero reward. This enables faster pruning of invalid branches and accelerates convergence.

Iterative Prompting. Syntax errors are often local and should not always be penalized with a zero reward. We allow the LLM up to five attempts to fix syntax issues, using the error message as context. If correction fails, the error is attributed to earlier steps, and a zero reward is backpropagated. The backpropagation mechanism is illustrated in Figure 9, with further details about our MCTS provided in Appendix J.1.

4.3 Automatic Identification of Structures of Optimal Policies Using Dynamic Programming

Given the formulation of operator-based Bellman equations, we aim to identify the structures of optimal policies. This is usually done by identifying the properties of the value function $V^*(x)$. For example, if $V^*(x)$ is convex in x, the optimal policy π^* is decreasing in x. For any operator T, we say that it *propagates* a property if, whenever V_n satisfies the property, then $T[V_n]$ also satisfies it. For example, the linear cost operator $T_{\texttt{cost}}[U(x)] = \beta x + \gamma U(x)$ propagates monotonicity and convexity, because $T_{\texttt{cost}}[U(x)]$ is monotone (convex) if $U(x)$ is monotone (convex). In the following and in the Appendix, we also say that an operator propagates A, where A is the *space* of functions having a certain property (for instance, all convex functions), and we may refer to A as a "property" by abuse of language. There is some common wisdom regarding which properties certain typical operators propagate. However, since $V_n^*(x)$ needs to go through the operator graph, the challenge is to find the properties that are propagated by *all* operators in the graph. A bruteforce approach would require checking an exponentially growing number of possibilities.

To illustrate, consider two operators T_1 and T_2 and six spaces of functions A–F with certain properties. For example, A can be the space of convex functions, B the space of increasing functions, and $A \cap B$ the space of convex increasing functions. Operator T_1 propagates $A \cap B$, $E \cap C$, and D, while operator T_2 propagates $C \cap A$, $D \cap F$, and B. In addition, we have $B \cap C \subset E$.

By computing the closure under intersection, T_1 is found to propagate, for example, $A \cap B \cap D$ and $E \cap C \cap D$. Identifying the smallest common space propagated by both T_1 and T_2 requires leveraging the inclusion relationship; in this case, the minimal shared space is $A \cap B \cap C$, which is not immediately evident from the original lists of properties that each operator propagates. A more detailed explanation of this example can be found in Appendix G.1.

We introduce a general dynamic programming algorithm to address this problem, providing a detailed explanation in Appendix G.

Theorem 4.2 (Identification of Structural Results)**.** *Given an operator graph $\mathcal{G}$, execution of Algorithms 1–3 gives us the set of properties propagated by all operators, with memory and time complexity of $\mathcal{O}(N|\mathcal{G}|)$ and $\mathcal{O}(N|\mathcal{G}|^2)$, where $|\mathcal{G}|$ is the number of operators in the graph, $N = \max_{T \in \mathcal{G}} n_T$, and n_T is the number of properties propagated by operator T.*

Proof. See Appendix G.3. □

A direct bruteforce approach would construct, for each operator, the full family of spaces obtained by closing its propagated properties under intersection, but this closure grows exponentially. A key observation in our algorithm is that any common propagated space can be written as the intersection of a subset of the properties that each operator initially propagates. Thus, instead of generating the full closure, we focus on identifying the properties that may appear in the intersection defining the smallest common propagated space. This is achieved by iteratively removing any property that cannot belong to this intersection. In the example above, F never appears in any space propagated by T_1 and is not implied by any inclusion relationship, so no valid common space can involve F in its intersection representation. Consequently, properties such as $D \cap F$ propagated by T_2 can be discarded. Repeating this pruning step yields a stable family of properties whose intersection is guaranteed to be both a common propagated space and the *smallest* such space. Proof and details on the treatment of inclusion relationships are provided in Appendix G.3.

Takeaway: Theorem 4.2 guarantees that, given an operator graph, Algorithm 1 can identify *all* structural properties detectable within our framework. Thus, our ability to recover structure from a problem description depends entirely on the operator graph generated by the LLM.

5 Experiments

Dataset. We constructed a dataset of 36 natural language descriptions of queueing control problems, varying in difficulty by size and shape of state spaces and number of event types. To assess performance in structure identification and support future research, the dataset includes three categories: (1) problems with provable structural results (e.g., Example 1); (2) problems with empirically observed, but unprovable, structures (e.g., Example 2); and (3) problems with no structural results. All problems are adapted from papers addressing realistic issues from domains such as hospital management (Bekker et al., 2017), telecommunications (Koole & Mandelbaum, 2002; Bhulai & Koole, 2003; Bekker et al., 2011; Zhang et al., 2025c), freight dispatching (Schwarz & Daduna, 2006; Amjath et al., 2023), assembly lines (Adeyinka & Kareem, 2018), and traffic control (Boon et al., 2023).

Experimental Setup. For each problem, we perform multiple MCTS roll-outs and select the best candidate by greedily following the highest-scoring path. Each formulation is evaluated by running a dynamic programming solver; if it fails to converge, it is deemed incorrect. The resulting value function is compared to the ground truth and accepted if within a predefined tolerance. We apply our structure analysis algorithm to each roll-out. Results are summarized in the following tables and interpreted with respect to the challenges in Appendix K, focusing on correctness, tractability, and interpretability. The code and dataset are available here.

5.1 Accuracy of Autoformalization and Error Analysis

Table 2 shows that our autoformulation framework outperforms baseline methods. Single-prompt methods fail entirely to solve the task, even with CoT prompting. CoT is both more computationally demanding and less effective than MCTS as a test-time scaling strategy. In contrast, MCTS achieves better performance with the same level of feedback (LLM, solver feedback, or syntax check). Although the first rollout is relatively costly, MCTS becomes increasingly efficient by reusing prior computations. Notably, it achieves comparable or better performance using fewer computational resources than baseline methods.

Ablation study—comparing with MCTS without SF and/or SC—shows that the incorporation of syntax checks significantly enhances performance, as all formulations proposed by MCTS are executable by the solver—effectively addressing the challenge of *Syntactic Validity*.

Table 2: Comparison with baselines and ablation study. Values in parentheses denote the number of completion tokens. Targeted prompts split the task into successive prompts, mirroring the steps of MCTS that can be found in Appendix M. SF: solver feedback, SC: syntax check.

Method	**1 Rollout**	**5 Rollouts**	**12 Rollouts**
GPT-4o (single prompt w. SF)	0% (2k)	0% (14k)	0% (38k)
CoT (single prompt w. SF)	0% (4k)	2.7% (20k)	2.7% (50k)
GPT-4o (targeted prompts w. SF)	5.5% (2k)	8.3% (16k)	8.3% (42k)
CoT (targeted prompts w. SF)	8.3% (6k)	11.0% (36k)	11.0% (85k)
MCTS (w/o. SF & SC)	11.0% (10k)	13.0% (43k)	16.0% (85k)
MCTS (w. SF, w/o. SC)	8.3% (10k)	36.1% (44k)	41.6% (86k)
GPT-4o (targeted prompts w. SF & SC)	44.4% (6k)	63.8% (30k)	72.0% (80k)
CoT (targeted prompts w. SF & SC)	52.7% (14k)	72.2% (74k)	75.2% (180k)
MCTS (w. SF & SC)	**63.8%** (11k)	**77.7%** (47k)	**83.3%** (96k)

The majority of remaining errors arise from *Semantic Misunderstanding*, including issues with *variable definitions*, *missing constraints*, *incorrect uniformizations*, and *misused operators*.

Figure 4: Accuracy against number of rollouts. Our method improves formulations continuously during search.

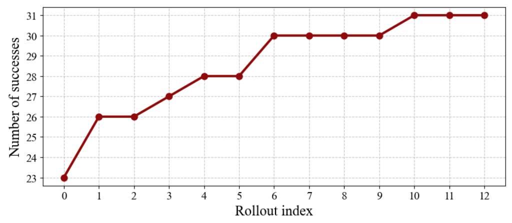


Table 3: Error types in failed roll-outs.

Type of error	Occurrence
Parameter identification (m^1)	5%
Variable definitions (m^2)	26%
Missing constraints (m^2)	24%
Incorrect events dynamics (m^3)	12%
Incorrect uniformization (m^3)	33%

► *Variable definition errors* often occur when the LLM introduces unnecessary queues. In hospital scenarios, for example, beds represent the queue, but the LLM may incorrectly distinguish between patients in beds and those waiting, effectively modeling two queues instead of one. ► *Missing constraints* arise from implicit assumptions in the problem description—e.g., ensuring patient counts remain non-negative. Solver feedback helps detect such issues by identifying unbounded state spaces. ► *Incorrect uniformizations* stem from deeper semantic misunderstandings. For instance, treatment probabilities differ when teams work in parallel vs. sequentially. These subtleties are primarily caught by the LLM signal. ► *Incorrect event dynamics* is another issue. While most events are identified, the LLM may omit actions in large action spaces or invent spurious events (e.g., to model per-time-unit costs). These are typically detected via a combination of solver and LLM signals. ► *Parameter Identification* accounts for few errors and generally fails when the problem description includes irrelevant or distracting information that misleads the LLM. These cases are caught exclusively via the LLM signal.

5.2 Computational tractability and interpretability

Structure Identification. When the operators identified by MCTS share a known propagated space, the second phase of our algorithm (Sec. 4.3) consistently recovers it, demonstrating that the *Structural Inference* challenge can be effectively addressed within our framework. Structural properties were identified in 74% of cases, also indicating strong performance on the second challenge: *Expressiveness of the Formulation*. Examples of successful structure extraction, along with a discussion of their interpretability, are provided in Appendix I. All failure cases fall into four categories: (i) *Incorrect problem translation by MCTS*, due to formalization errors discussed in Section 5.1, and not revisited here. (ii) *Operator mislabeling*, where the LLM correctly models state dynamics but misnames operators. This is the only remaining bottleneck for *Structural Inference* in our framework. (iii) *Limited structural expressiveness*, where the formulation is valid but does not expose the structure. This reveals that some correct formulations are less amenable to structural analysis. We illustrate this in Example 1. (iv) *Structural results beyond Koole's framework*, where certain properties cannot be captured regardless of the operator graph. These cases expose fundamental limitations of the current framework and suggest directions for future extensions. See Example 2.

Takeaway: Quantitative evaluation across the dataset shows that our method correctly identifies 74% of the structural properties, *prior to* solving the problem. Therefore, we can reduce the computational complexity by calling specialized solvers for a large portion of the problems.

Example 1 (Equivalent problem formulations with different structural expressiveness)**.** *Our hospital has 1 ward that manages 2 types of patients with* ***shared*** *healthcare teams. There are N_b beds in total. The average arrival rates of the patients are λ_1/hour and λ_2/hour respectively. The teams take care of patients* ***in parallel*** *with an average rate that depends on their type : μ_1/hour and μ_2/hour respectively. When a patient arrive we can refuse it, it occurs a cost of c_1 for the first type of patients and c_2 of the others.*

Key challenges: We cannot obtain structural results from the straightforward problem formulation. How to find an equivalent combination of operators that allow us to obtain structural results?

Straightforward problem formulation. The natural events of this problem are controlled arrivals and departures of the two types of patients, leading to the operator graph (found by MCTS):

$$V^*_{n+1} = T_{\texttt{cost}} \left\{ T_{\texttt{unif}} \left[T_{\texttt{CA}(1)}(V^*_n), T_{\texttt{CA}(2)}(V^*_n), T_{\texttt{D}(1)}(V^*_n), T_{\texttt{D}(2)}(V^*_n), V_n \right] \right\}$$

In this formulation, the probabilities in T_{unif} depend on the state. For instance,

$$p_{\texttt{D}(1)} = (\mu_1 n_1)/(\lambda_1 + \lambda_2 + \mu N_b) \quad \text{with} \;\; \mu = \max(\mu_1, \mu_2).$$

Due to state dependent probabilities in T_{unif}, we cannot obtain any structural result.

Equivalent problem formulation. We define a new departure operator $T_{\texttt{D}_{\text{modified}}}$: $(\Gamma = \lambda_1 + \lambda_2 + \mu N_b)$

$$T_{\texttt{D}_{\text{modified}}(i)} f(x) = \frac{2\mu_1 n_1}{\Gamma - (\lambda_1 + \lambda_2)} f((x - e_i)^+) + \left(1 - \frac{2\mu_i n_i}{\Gamma - (\lambda_1 + \lambda_2)}\right) f(x)$$

With the new departure operator, the probabilities in T_{unif} are independent of the state:

$$p_{\text{CA}(i)} = \lambda_i / \Gamma, \quad p_{\text{D}_{\text{modified}}(i)} = \frac{1}{2} \left[1 - (\lambda_1 + \lambda_2) / \Gamma\right].$$

Structural results. From the equivalent problem formulation, we can identify the *monotonicity* property of the optimal policy.

Takeaway: Finding equivalent problem formulation with higher structural expressiveness is critical to identify structural results.

Example 2 (Problem with intractable structural results)**.** *Our hospital has 3 wards arranged sequentially, with capacities of 5, 15, and 15 beds, respectively. Each ward has its own healthcare team and manages its own patients. On average, new patients arrive at rates of 3, 20, and 5 patients/day in the respective wards. The wards serve patients one at a time at rates of 10, 5, and 3 patients/day,* *respectively. After being served in the first or second ward, we can transfer to the next ward at a cost of 2 per transfer or keep them in the current ward. Patients served in the third ward leave the hospital. Additionally,* ***patients can be moved back*** *from ward 2 to ward 1 at a rate of 3 patients/day or from ward 3 to ward 2 at a rate of 1 patient/day, each transfer incurring a cost of 2. Incoming patients can also be refused, incurring costs of 5, 10, and 15 for wards 1, 2, and 3, respectively.*

Key challenges: The solution to the problem exhibits structural properties, but these cannot be anticipated regardless of the choice of operator graph.

Problem formulation. Our autoformulator correctly output the operator graph:

$$\begin{aligned} V_{n+1} = T_{\text{cost}} \{ & T_{\text{unif}} [T_{\text{CA},1}(V_n), T_{\text{CA},2}(V_n), T_{\text{CA},3}(V_n), \\ & T_{\text{CTD},(1,2)}(V_n), T_{\text{CTD},(2,3)}(V_n), T_{\text{CTD},(2,1)}(V_n), T_{\text{CTD},(3,2)}(V_n), T_{\text{D1},3}(V_n)] \} . \end{aligned}$$

Structural results. We can observe the structures (e.g., monotone switching curve) of the optimal policy empirically (see Figure 6 in App. I). However, we have yet to find an equivalent problem formulation that would express these structural results.

Takeaway: The current method for identifying structural results fails on certain problems, as it depends on limited theoretical results.

6 Discussion

We propose the first-ever autoformulator of event-based MDPs, a class of sequential decision-making problems encountered in various domains. Key to our framework is representation of the Bellman equation as an operator graph, which ensures interpretability and improves accuracy. By proving a universal operator graph topology for event-based MDPs, we significantly reduce the search space of autoformulation. We also propose a low-complexity algorithm to identify structural results based on the operator graph. To construct the operator graph, we make significant modifications to MCTS by evaluating intermediate nodes for denser rewards and utilizing solver feedback for more objective evaluations. Experimental results demonstrate the effectiveness of our approach in accurately formulating problems and uncovering key structural insights. We also create the first dataset on autoformulating queueing control problems, with a wide variety of labeled problems.

Autoformalising broader classes of operator graphs. A natural direction for future work is to extend autoformulation beyond event-based queueing models. The most immediate step is to consider broader families of MDPs that can be represented as graphs of operators, typically discrete-state MDPs with richer dynamics than those found in event-based systems, such as models with continuously available actions or deterministic events occurring at fixed frequencies. While the universal topology of Theorem 4.1 may no longer hold in these settings, the operator-graph viewpoint remains applicable, and the structural-identification component of our framework applies to any operator graph without modification. The main challenge lies in designing a construction algorithm capable of generating arbitrary operator-graph topologies.

Beyond analytic properties of the value function. This paper focuses on analytic structural properties, such as convexity of the value function, that can be propagated through the operator graph. Many MDPs, however, do not admit such low-level analysis due to complex dynamics or irregular state spaces. We argue that the broader philosophy of designing autoformulators that are both computation-aware and interpretability-aware naturally extends to higher-level structural patterns beyond the analytical properties considered here. For example, hierarchical RL exploits the hierarchical structure inherent in certain tasks, illustrating how higher-level structure can guide algorithmic design even when low-level analytic results are unavailable. Incorporating such perspectives would enable autoformulation in domains with substantially more complex dynamics.

References

Daniel Adelman. Price-directed control of a closed logistics queueing network. *Operations Research*, 55(6):1022–1038, 2007.

Adekanmi Adeyinka and Buliaminu Kareem. The application of queuing theory in solving automobile assembly line problem. *International Journal of Engineering Research and*, V7, 06 2018. doi: 10.17577/IJERTV7IS060206.

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024.

Mohamed Amjath, L. Kerbache, Adel Elomri, and James Smith. Optimal buffers allocation in an inter-facility material transfer system modelled as a closed queueing network and analysed through a simulation-optimisation approach. pp. 705–714, 03 2023. doi: 10.46254/AN13.20230208.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. Autoformulation of mathematical optimization models using LLMs. In *Forty-second International Conference on Machine Learning (ICML)*, 2025.

René Bekker, GM Koole, Bo Friis Nielsen, and Thomas Bang Nielsen. Queues with waiting time dependent service. *Queueing Systems*, 68:61–78, 2011.

René Bekker, Ger Koole, and Dennis Roubos. Flexible bed allocations for hospital wards. *Health Care Management Science*, 20:453–466, 2017.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning, 2017. URL https://arxiv.org/abs/1707.06887.

Richard Bellman. On the theory of dynamic programming. *Proceedings of the national Academy of Sciences*, 38(8):716–719, 1952.

Saif Benjaafar, M. ElHafsi, and Tingting Huang. Optimal control of a production-inventory system with both backorders and lost sales. *Naval Research Logistics*, 57(3):252–265, 2010a. doi: 10.1002/nav.20399.

Saif Benjaafar, Jean-Philippe Gayon, and Seda Tepe. Optimal control of a production–inventory system with customer impatience. *Operations Research Letters*, 38(4):267–272, 2010b. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2010.03.008. URL https://www.sciencedirect.com/science/article/pii/S016763771000043X.

Jeroen Berrevoets, Ahmed Alaa, Zhaozhi Qian, James Jordon, Alexander ES Gimson, and Mihaela van der Schaar. Learning queueing policies for organ transplantation allocation using interpretable counterfactual survival analysis. In *International Conference on Machine Learning*, pp. 792–802. PMLR, 2021.

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

Dimitris Bertsimas and Georgios Margaritis. Robust and adaptive optimization under a large language model lens, 2024. URL https://arxiv.org/abs/2501.00568.

Dimitris Bertsimas, Nathan Kallus, Alexander M Weinstein, and Ying Daisy Zhuo. Personalized diabetes management using electronic medical records. *Diabetes care*, 40(2):210–217, 2017.

Sandjai Bhulai and Ger Koole. A queueing model for call blending in call centers. *IEEE Transactions on Automatic Control*, 48(8):1434–1438, 2003.

David Blackwell. Discounted dynamic programming. *The Annals of Mathematical Statistics*, 36(1): 226–235, February 1965.

Blai Bonet and Hector Geffner. Action selection for mdps: Anytime ao* vs. uct, 2019. URL https://arxiv.org/abs/1909.12104.

Marko Boon, Guido Janssen, Johan van Leeuwaarden, and Rik Timmerman. Optimal capacity allocation for heavy-traffic fixed-cycle traffic-light queues and intersections. *Transportation Research Part B: Methodological*, 167:79–98, 2023. ISSN 0191-2615. doi: https://doi.org/10.1016/j.trb.2022.11.010. URL https://www.sciencedirect.com/science/article/pii/S0191261522001916.

Stephen P Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

Timothy CY Chan, Simon Y Huang, and Vahid Sarhangian. Dynamic control of service systems with returns: Application to design of postdischarge hospital readmission prevention programs. *Operations Research*, 73(4):2242–2263, 2025.

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition, 1999. URL https://arxiv.org/abs/cs/9905014.

Mark Ebben, Durk-Jouke van der Zee, and Matthieu van der Heijden. Dynamic one-way traffic control in automated transportation systems. *Transportation Research Part B: Methodological*, 38 (5):441–458, 2004.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning, 2019. URL https://arxiv.org/abs/1906.05253.

David A. Freedman. The optimal reward operator in special classes of dynamic programming problems. *The Annals of Probability*, 2(5):942–949, 1974. URL http://www.jstor.org/stable/2959317. Accessed: 27 Nov. 2025.

Nakul Gopalan, Marie desJardins, Michael Littman, James MacGlashan, Sarah Squire, Stefanie Tellex, John Winder, and Lawson Wong. Planning with abstract markov decision processes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, pp. 480–488, 2017. doi: 10.1609/icaps.v27i1.13867.

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *CoRR*, abs/1106.1822, 2011. URL http://arxiv.org/abs/1106.1822.

Gurobi Optimization. 2023 State of Mathematical Optimization Report. https://www.gurobi.com/resources/report-state-of-mathematical-optimization-2023/, 2023.

B. Hajek. Optimal control of two interacting service stations. *IEEE Transactions on Automatic Control*, 29(6):491–499, 1984. doi: 10.1109/TAC.1984.1103577.

James E. Helm, Sheir AhmadBeygi, and Mark P. Van Oyen. Design and analysis of hospital admission control for operational effectiveness. *Production and Operations Management*, 20(3):359–374, 2011. doi: 10.1111/j.1937-5956.2011.01231.x.

Yu-Pin Hsu, Navid Abedini, Natarajan Gautam, Alex Sprintson, and Srinivas Shakkottai. Opportunities for network coding: To wait or not to wait. *IEEE/ACM Transactions on Networking*, 23(6): 1876–1889, 2015. doi: 10.1109/TNET.2014.2347339.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, May 2025.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 10026–10039. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/531db99cb00833bcd414459069dc7387-Paper.pdf.

Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon mdps. *Proceedings of the International Conference on Automated Planning and Scheduling*, 23:135–143, 06 2013. doi: 10.1609/icaps.v23i1.13557.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (eds.), *Machine Learning: ECML 2006*, pp. 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.

Ger Koole. A simple proof of the optimality of a threshold policy in a two-server queueing system. *Systems Control Letters*, 26(5):301–303, 1995. ISSN 0167-6911. doi: https://doi.org/10.1016/0167-6911(95)00015-1. URL `https://www.sciencedirect.com/science/article/pii/0167691195000151`.

Ger Koole. Structural results for the control of queueing systems using event-based dynamic programming. *Queueing Systems*, 30:323–339, 1998. URL `https://api.semanticscholar.org/CorpusID:17274845`.

Ger Koole. Monotonicity in Markov reward and decision chains: Theory and applications. *Foundations and Trends® in Stochastic Systems*, 1(1):1–76, 2007.

Ger Koole and Avishai Mandelbaum. Queueing models of call centers: An introduction. *Annals of Operations Research*, 113:41–59, 2002.

Daniel Koutas, Daniel Hettegger, Kostas G. Papakonstantinou, and Daniel Straub. Convex is back: Solving belief MDPs with convexity-informed deep reinforcement learning, 2025. URL `https://arxiv.org/abs/2502.09298`.

Kuo Liang, Yuhang Lu, Jianming Mao, Shuyi Sun, Chunwei Yang, Congcong Zeng, Xiao Jin, Hanzhang Qin, Ruihao Zhu, and Chung-Piaw Teo. LLM for Large-Scale Optimization Model Auto-Formulation: A Lightweight Few-Shot Learning Approach. `http://dx.doi.org/10.2139/ssrn.5329027`, June 2025. Available at SSRN: `https://ssrn.com/abstract=5329027`.

Steven A Lippman. Applying a new device in the optimization of exponential queuing systems. *Operations research*, 23(4):687–710, 1975.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. Optmath: A scalable bidirectional data synthesis framework for optimization modeling, 2025. URL `https://arxiv.org/abs/2502.11102`.

Pinhas Naor. The regulation of queue size by levying tolls. *Econometrica: journal of the Econometric Society*, pp. 15–24, 1969.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

Martin L Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pp. 189–203. PMLR, 2023.

Robert W Robinson. Counting labeled acyclic digraphs. *New Directions in the Theory of Graphs*, pp. 239–273, 1973.

Maike Schwarz and Hans Daduna. Queueing systems with inventory management with random lead times and with backordering. *Mathematical Methods of Operations Research*, 64(3):383–414, 2006.

Linn I Sennott. *Stochastic dynamic programming and the control of queueing systems*. John Wiley & Sons, 2009.

Richard F Serfozo. An equivalence between continuous and discrete time Markov decision processes. *Operations Research*, 27(3):616–620, 1979.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

Shaler Stidham. Optimal control of admission to a queueing system. *IEEE Transactions on Automatic Control*, 30(8):705–713, 1985.

Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

A.C.C. van Wijk, I.J.B.F. Adan, and G.J. van Houtum. Optimal lateral transshipment policies for a two location inventory problem with multiple demand classes. *European Journal of Operational Research*, 272(2):481–495, 2019. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2018.06.033. URL https://www.sciencedirect.com/science/article/pii/S037722171830568X.

Segev Wasserkrug, Leonard Boussioux, Dick den Hertog, Farzaneh Mirzazadeh, Birbil Ş. Ilker, Jannis Kurtz, and Donato Maragno. Enhancing decision making through the integration of large language models and operations research optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(27):28643–28650, Apr. 2025. doi: 10.1609/aaai.v39i27.35090. URL https://ojs.aaai.org/index.php/AAAI/article/view/35090.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-Experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2023.

Yu Xiong, Gendao Li, Yu Zhou, Kiran Fernandes, Richard Harrison, and Zhongkai Xiong. Dynamic pricing models for used products in remanufacturing with lost-sales and uncertain quality. *International Journal of Production Economics*, 147:678–688, 2014. ISSN 0925-5273. doi: https://doi.org/10.1016/j.ijpe.2013.04.025. URL https://www.sciencedirect.com/science/article/pii/S0925527313001898. Interdisciplinary Research in Operations Management.

Insoon Yang. A convex optimization approach to dynamic programming in continuous state and action spaces. *Journal of Optimization Theory and Applications*, 187(1):133–157, September 2020. ISSN 1573-2878. doi: 10.1007/s10957-020-01747-1. URL http://dx.doi.org/10.1007/s10957-020-01747-1.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve llms for optimization modeling, 2025. URL https://arxiv.org/abs/2407.09887.

Shuyu Yin, Qixuan Zhou, Fei Wen, and Tao Luo. A priori estimates for deep residual network in continuous-time reinforcement learning, 2024. URL https://arxiv.org/abs/2402.16899.

Huizhen Yu, A. Rupam Mahmood, and Richard S. Sutton. On generalized bellman equations and temporal-difference learning, 2018. URL https://arxiv.org/abs/1704.04463.

Bowen Zhang, Pengcheng Luo, Genke Yang, Boon-Hee Soong, and Chau Yuen. OR-LLM-Agent: Automating modeling and solving of operations research optimization problems with reasoning LLM, 2025a. URL https://arxiv.org/abs/2503.10009.

Lunjun Zhang, Ge Yang, and Bradly C. Stadie. World model as a graph: Learning latent landmarks for planning, 2021. URL https://arxiv.org/abs/2011.12491.

Yisong Zhang, Ran Cheng, Guoxing Yi, and Kay Chen Tan. A systematic survey on large language models for evolutionary optimization: From modeling to solving. *arXiv preprint arXiv:2509.08269*, 2025b.

Ziyong Zhang, Tao Dong, Jie Yin, Yue Xu, Zongyi Luo, Hao Jiang, and Jing Wu. A particle swarm optimization-based queue scheduling and optimization mechanism for large-scale low-earth-orbit satellite communication networks. *Sensors*, 25(4), 2025c. ISSN 1424-8220. doi: 10.3390/s25041069. URL https://www.mdpi.com/1424-8220/25/4/1069.

Bo Zhou, Ying Cui, and Meixia Tao. Stochastic throughput optimization for two-hop systems with finite relay buffers. *IEEE Transactions on Signal Processing*, 63(20):5546–5560, October 2015. ISSN 1941-0476. doi: 10.1109/tsp.2015.2452225. URL http://dx.doi.org/10.1109/TSP.2015.2452225.

Chenyu Zhou, Jingyuan Yang, Linwei Xin, Yitian Chen, Ziyan He, and Dongdong Ge. Auto-formulating dynamic programming problems with large language models. *arXiv preprint arXiv:2507.11737*, 2025.

Weifen Zhuang and Michael Z.F. Li. A new method of proving structural properties for certain class of stochastic dynamic control problems. *Operations Research Letters*, 38(5):462–467, 2010. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2010.05.008. URL https://www.sciencedirect.com/science/article/pii/S0167637710000714.

Eren Başar Çil, Fikri Karaesmen, and E. Lerzan Örmeci. Dynamic pricing and scheduling in a multi-class single-server queueing system. *Queueing Systems*, 67(4):305–331, 2011. ISSN 1572-9443. doi: 10.1007/s11134-011-9214-5. URL https://doi.org/10.1007/s11134-011-9214-5.

Contents

A AUTOFORMULATING OPTIMIZATION PROBLEMS VERSUS MDPs

Here is an example natural-language description of two parallel M/M/1 queues with controlled arrival and uncontrolled departure. It is adapted from problems in hospital admission control (Bekker et al., 2017; Naor, 1969). Some representative **formulation challenges** are annotated.

We consider a hospital with two wards: one for Critical patients and one for General patients, each staffed by a dedicated team. Both wards share N_B beds. On average, λ_C Critical and λ_G General patients arrive per day [State-dependent action sets: admission decision only at an arrival, readmission decision only at a departure.]. Treatment rates are μ_C and μ_G patients per day for Critical and General wards, respectively, with each team serving one patient at a time [Transition probabilities need to be inferred from the arrival and departure rates.]. Treated patients leave the system, releasing their beds. Upon arrival, a patient may be admitted if a bed is available; otherwise, the patient is rejected, incurring a penalty cost c_C (Critical) or c_G (General). Each admitted patient generates a holding cost ρ_h per unit time. The objective is to minimize the long-run average operating cost, with discount factor α. [Throughout the description, the implicit constraint of queue length being nonnegative was not mentioned.]

Computation challenges. The optimal policy maps queue lengths to admission decisions: $a_{\texttt{CA,C}} \in \{0, 1\}$ for Critical patients and $a_{\texttt{CA,G}} \in \{0, 1\}$ for General patients. Standard dynamic programming enumerates all queue lengths to determine the optimal decisions, repeating this process until convergence. However, if the optimal admission policy can be shown to follow a threshold structure, the search reduces to identifying the threshold values. Likewise, if the value function is convex, specialized dynamic programming solvers converges faster. Thus, uncovering structural properties of optimal policies and value functions before solving significantly reduces computational complexity.

Interpretability challenges. It is also desirable if the policy is interpretable: “admit only when the queue length is smaller than this threshold”.

Next, we inspect a hospital logistics problem in the NL4Opt dataset (Ramamonjison et al., 2023).

A hospital can transport their patients either using a type II ambulance or hospital van. The hospital needs to transport 320 patients every day. A type II ambulance is mounted on a truck-style chassis and can move 20 patients every shift and costs the hospital (including gas and salary) \$820. A hospital van can move 15 patients and costs the hospital \$550 every shift. The hospital can have at most 60% of shifts be hospital vans due to union limitations of the type II ambulance drivers. How many of shift using each type of vehicle should be scheduled to minimize the total cost to the hospital?

In terms of formulation challenges, there is no notion of states or transition probabilities, and the decision variables are two scalars. In terms of computation challenges, this is a linear program that can scale with tens of thousands of decision variables. In terms of interpretability challenges, the optimal decisions are the numbers of vehicles of each type, which is easy to understand.

Table 4 provides a side-by-side comparison of optimization and MDP autoformulation.

Table 4: **Autoformulating mathematical optimization versus MDPs.**

Challenge	**Aspect**	**Optimization**	**MDP**
Formulation	Variables	Often explicitly defined	Need to derive states/actions from context
Formulation	Constraints	Often explicitly defined	Sometimes implicit and omitted
Formulation	Stochastic dynamics	Usually non-existent	Important to model
Computation	Scalability	Scales well for convex problems	“Curse of dimensionality”
Interpretability	Problem formulation	Easy to inspect variable, objectives, constraints	High-dimensional components (e.g., transition probabilities) hard to inspect
Interpretability	Solution	Values of decision variables directly understandable	Policy mapping hard to interpret

B Event-Based MDPs in Various Applications

Table 5 provides a non-exhaustive list of event-based MDPs in a variety of applications domains. We focus on problems that can be modeled as control of queueing systems.

Table 5: **Event-based MDPs in diverse application domains.**

Domain	**Problem**	**Representative Works**
Healthcare	diabetes management	Bertsimas et al. (2017)
	organ transplant	Berrevoets et al. (2021)
	hospital admission control	Bekker et al. (2017)
Business	inventory management, logistics	Schwarz & Daduna (2006); Adelman (2007)
	assembly lines	Adeyinka & Kareem (2018)
	freight dispatching	Schwarz & Daduna (2006); Amjath et al. (2023)
Telecommunications	call center management	Koole & Mandelbaum (2002); Bhulai & Koole (2003); Bekker et al. (2011); Zhang et al. (2025c)
Transportation	intersection management	Stidham (1985); Ebben et al. (2004)
	traffic control	Boon et al. (2023)

C Extended related work

In Section 2, we mostly discussed work on autoformulation. We now provide additional background on two technical dimensions central to our approach: operator-based formulations of Bellman equations and DAG-based structures for the study of MDPs. Our goal is to situate our operator-graph perspective within these broader lines of research.

C.1 Operator-based formulations of Bellman equations

A substantial body of work has examined Bellman equations through the lens of operators acting on value functions. Classical dynamic programming texts already present the Bellman update as a *contraction operator* Freedman (1974); Blackwell (1965). However, the formal meaning of ”operator” varies significantly across the literature and is used in conceptually different ways.

The operators employed in the present work are representative of one such interpretation. Koole introduced a collection of *elementary* or *event-based* operators Koole (1998; 2007) from which the Bellman update can be constructed *by composition*. These atomic operators are not Bellman updates themselves; rather, they model primitive system events (arrivals, departures, etc.). The key insight is that one can establish structural or monotonicity properties at the level of these elementary operators, and then deduce analogous results for broad classes of Bellman equations without repeating the full analysis. This philosophy has been adopted in subsequent work, which introduces additional operators to establish monotonicity or dominance properties in diverse stochastic control problems Helm et al. (2011); Xiong et al. (2014); van Wijk et al. (2019); Benjaafar et al. (2010b;a). Once such properties are proved at the operator level, they can be reused whenever those operators appear as components of a Bellman equation.

Other definitions of operators focus on variants or generalizations of the canonical Bellman update. One example is the *generalized Bellman equation* Yu et al. (2018), which defines an operator not only on value functions but on the parameter space of the learning algorithm itself (here, temporal-difference learning). Another example is the *distributional Bellman equation* Bellemare et al. (2017), where the operator acts on the *distribution* of returns rather than the value function (i.e., the expected return). Finally, in parts of the literature the term *operator* is used more informally for individual pieces of the Bellman update, without the systematic decomposition seen in Koole’s framework. For instance, in one of Bellman’s original formulations Bellman (1952) the operator is essentially the full update except for the $\max$ or $\min$ optimization step, whereas in Yin et al. (2024) it is essentially the opposite, with the operator reduced to that optimization step alone.

C.2 DAG-based structures for the study of MDPs

One classical use of DAGs in MDPs that is closely related to our operator DAGs is the AND/OR search graph, where nodes are time-indexed states and edges represent possible transitions, often weighted by their probabilities Bonet & Geffner (2019). In DP or RL this structure is typically implicit, as it is algebraically collapsed into the Bellman equation. In contrast, several algorithms—especially for finite-horizon problems—operate directly on this graph, such as THTS Keller & Helmert (2013) and the MCTS algorithm used in this paper. Our operator DAG can be viewed as an intermediate representation between the full AND/OR search graph and its complete collapse into the Bellman equation. It factorizes the large AND/OR graph using the analytical structure of the Bellman update, retaining enough of the original graph topology to decompose the Bellman update into more atomic analytical transformations, the operators.

Another use of DAGs in MDPs, less related but potentially confusable with ours, arises in hierarchical reinforcement learning (HRL), where a decision problem is decomposed into higher-level tasks and lower-level subtasks. Each node in this hierarchy is itself a sequential decision problem, and the structure reshapes both the control problem and the learning dynamics of the optimal policy Dietterich (1999); Gopalan et al. (2017). The DAGs we consider in this paper are fundamentally different. Our operator DAG is an analytic hierarchy that specifies the order in which operations are applied to the value function within a single Bellman update. Although some nodes may involve a choice, they correspond to a one-shot evaluation of the policy, not to sub-MDPs with their own temporal evolution or repeated policy execution. HRL thus defines a hierarchy of behaviors unfolding over many transitions, whereas our operator DAG defines a hierarchy of operations applied within one transition.

Beyond these settings, DAGs appear in many other parts of the MDP and RL literature, reflecting the general versatility of graph-based abstractions. These uses, however, are conceptually quite different from our operator DAGs. For example, factored MDPs represent the transition model as a dynamic Bayesian network Guestrin et al. (2011), exploiting conditional-independence structure to obtain compact transition models and more efficient computations. Other work employs graphs as high-level planning abstractions Eysenbach et al. (2019); Zhang et al. (2021), for instance by performing Dijkstra-style shortest-path computations on an abstract task graph while delegating low-level control to an RL policy, as in Jothimurugan et al. (2021).

D Detailed Derivations of Results in Section 3

In this section, we provide detailed proofs and derivations of the results in Sec. 3. In the first part, we provide some key results on continuous-time and discrete-time MDPs to make the paper self-contained. In the second part, we go through the entire process of solving the example of parallel M/M/1 queues with controlled arrivals and shared beds.

D.1 Key Results on Continuous-Time and Discrete-Time MDPs

D.1.1 Bellman Equations for Continuous-Time MDPs

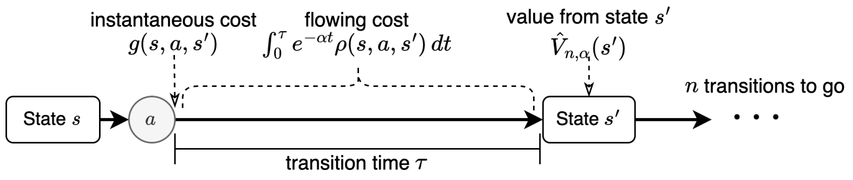
To solve for the minimum cost, we define $\hat{V}_{n,\alpha}(s)$ as the *minimum* α-discounted cost during the last n state transitions when starting from state s. Defining $\hat{V}_{0,\alpha}(s) \equiv 0$, we have the following Bellman equation (Lippman, 1975; Serfozo, 1979)

$$\hat{V}_{n+1,\alpha}(s) = \min_{a \in \mathcal{A}_s} \left\{ \hat{c}(s,a) + \frac{\lambda(s,a)}{\lambda(s,a)+\alpha} \cdot \textstyle\sum_{s' \in \mathcal{S}} \hat{P}(s'|s,a) \hat{V}_{n,\alpha}(s') \right\}, \tag{12}$$

where $\frac{\lambda(s,a)}{\lambda(s,a)+\alpha} = \mathbb{E}_{\tau \sim \texttt{Exp}[\lambda(s,a)]}\left[e^{-\alpha\tau}\right]$ is the expected discounting of the next state value $\hat{V}_{n,\alpha}(s')$.

The Bellman equation in (12) is not standard due to the (s,a)-dependent discounting $\frac{\lambda(s,a)}{\lambda(s,a)+\alpha}$. We can remedy this issue by studying the discrete-time embedded MDP (Serfozo, 1979; Sennott, 2009; Lippman, 1975).

We first derive the Bellman equation of the continuous-time MDP in equation 12.



State transition dynamics and costs during a sample path of a continuous-time MDP.

The figure above shows the state dynamics and the costs incurred during the transition from state s to state s'. The expected total cost starting from state s consists of two parts: the expected cost accumulated until the transition and the expected total cost starting from the next state. The cost during the transition can be calculated as

$$\begin{aligned}
\hat{c}(s,a) &= \mathbb{E}_{\tau \sim \texttt{Exp}[\lambda(s,a)],\, s' \sim \hat{P}(\cdot|s,a)} \left[g(s,a,s') + \int_0^\tau e^{-\alpha t} \rho(s,a,s') dt \right] && (13) \\
&= \mathbb{E}_{\tau \sim \texttt{Exp}[\lambda(s,a)],\, s' \sim \hat{P}(\cdot|s,a)} \left[g(s,a,s') + \rho(s,a,s') \cdot \int_0^\tau e^{-\alpha t} dt \right] && (14) \\
&= \mathbb{E}_{\tau \sim \texttt{Exp}[\lambda(s,a)],\, s' \sim \hat{P}(\cdot|s,a)} \left[g(s,a,s') + \rho(s,a,s') \cdot \frac{1-e^{-\alpha\tau}}{\alpha} \right] && (15) \\
&= \mathbb{E}_{s' \sim \hat{P}(\cdot|s,a)} \left\{ g(s,a,s') + \rho(s,a,s') \cdot \mathbb{E}_{\tau \sim \texttt{Exp}[\lambda(s,a)]} \left[\frac{1-e^{-\alpha\tau}}{\alpha} \right] \right\} && (16) \\
&= \mathbb{E}_{s' \sim \hat{P}(\cdot|s,a)} \left[g(s,a,s') + \rho(s,a,s') \cdot \frac{1}{\lambda(s,a)+\alpha} \right] && (17) \\
&= \sum_{s' \in \mathcal{S}} \left[g(s,a,s') + \frac{\rho(s,a,s')}{\lambda(s,a)+\alpha} \right] \hat{P}(s'|s,a) && (18)
\end{aligned}$$

Note that without discounting ($\alpha = 0$), the term $\frac{\rho(s,a,s')}{\lambda(s,a)}$ is the cost rate $\rho(s,a,s')$ times the expected transition time $\frac{1}{\lambda(s,a)}$. With discounting, $\frac{1}{\lambda(s,a)+\alpha} = \mathbb{E}_{\tau \sim \texttt{Exp}[\lambda(s,a)]} \int_0^\tau e^{-\alpha t}\, dt$ can be interpreted as the expectation of "discounted" transition time.

The second part is the cost starting from the next state s', discounted based on the transition time τ:

$$\mathbb{E}_{\tau\sim\mathrm{Exp}[\lambda(s,a)]}\left[e^{-\alpha\tau}\cdot\hat{V}_{n,\alpha}(s')\right]=\mathbb{E}_{\tau\sim\mathrm{Exp}[\lambda(s,a)]}\left[e^{-\alpha\tau}\right]\cdot\hat{V}_{n,\alpha}(s')=\frac{\lambda(s,a)}{\lambda(s,a)+\alpha}\cdot\hat{V}_{n,\alpha}(s'). \tag{19}$$

Note that this is the cost *if* the next state is s'. So we need to take the expectation over s' to get the expected cost starting from state s and action a:

$$\hat{Q}_{n,\alpha}(s,a)=\hat{c}(s,a)+\frac{\lambda(s,a)}{\lambda(s,a)+\alpha}\cdot\sum_{s'\in\mathcal{S}}\hat{P}(s'|s,a)\hat{V}_{n,\alpha}(s'). \tag{20}$$

Therefore, starting from state s, the *minimum* α-discounted cost during the last $n+1$ transitions is

$$\hat{V}_{n+1,\alpha}(s)=\min_{a\in\mathcal{A}_s}\hat{Q}_{n,\alpha}(s,a), \tag{21}$$

which is the Bellman equation in equation 12.

D.1.2 Equivalent Discrete-Time MDPs and Various Performance Criteria

In Sec. 3, we introduced several notions of *value functions* for continuous-time MDPs:

- $\hat{V}_{\alpha,\pi}$: discounted cost of a continuous-time MDP with discount rate α and under policy π;
- $\hat{V}_{\alpha}$: minimum discounted cost of a continuous-time MDP with discount rate α;
- $\hat{V}_{n,\alpha}$: minimum α-discounted cost of a continuous-time MDP during the last n transitions;
- $\hat{J}_{\pi}$: average cost of a continuous-time MDP under policy π;
- $\hat{J}$: minimum average cost of a continuous-time MDP.

The counterparts in discrete-time MDPs are $V_{\gamma,\pi}, V_\gamma, V_{n,\gamma}, J_\pi, J$, where we use regular letters and use the discount factor γ in the subscript.

The relationship between these value functions are

$$\begin{tikzcd}
 & & \hat{V}_{n,\alpha} \arrow[d, "n\to\infty"'] & & \\
\text{continuous-time:} & \hat{V}_{\alpha,\pi} \arrow[r, "\inf_\pi"] \arrow[d, Leftrightarrow] & \hat{V}_{\alpha} \arrow[d, Leftrightarrow] & \hat{J} \arrow[d, Leftrightarrow] & \hat{J}_{\pi} \arrow[l, "\inf_\pi"'] \arrow[d, Leftrightarrow] \\
\text{discrete-time:} & V_{\gamma,\pi} \arrow[r, "\inf_\pi"] & V_{\gamma} \arrow[r, "\lim_{\gamma\to 1}(1-\gamma)V_\gamma"] & J & J_{\pi} \arrow[l, "\inf_\pi"'] \\
 & & V_{n,\gamma} \arrow[u, "n\to\infty"] & &
\end{tikzcd} \; . \tag{22}$$

In equation 22, the equivalence "$\Leftrightarrow$" between the continuous-time MDP and its discrete-time embedded MDP defined in equation 23 is due to the result in Serfozo (1979), which we restated here.

Theorem D.1 (Theorem in Serfozo (1979))**.** *Let $(\mathcal{S},\mathcal{A}_s,\hat{c},\tau,\hat{P},\alpha)$ be a continuous-time MDP with bounded state transition rates (i.e., $\sup_{s,a}\lambda(s,a)<\Lambda$). Let $(\mathcal{S},\mathcal{A}_s,c,P,\gamma)$ be the discrete-time MDP with discount factor $\gamma=\frac{\Lambda}{\Lambda+\alpha}$, and state transition probabilities and the cost function defined as*

$$P(s'|s,a)=\begin{cases}\lambda(s,a)\cdot\hat{P}(s'|s,a)/\Lambda, & \textit{if } s'\neq s\\ 1-\lambda(s,a)/\Lambda, & \textit{if } s'=s\end{cases}\quad \textit{and}\quad c(s,a)=\frac{\lambda(s,a)+\alpha}{\Lambda+\alpha}\cdot\hat{c}(s,a). \tag{23}$$

For any stationary policy π, we have $V_\gamma=\hat{V}_\alpha$ and $J_\pi=\hat{J}_\pi/\Lambda$.

This transformation, often referred to as the *uniformization* technique, allows us to reduce a continuous-time MDP with exponential transition times into an equivalent discrete-time MDP with adjusted transition probabilities and costs.

The relationship "$\rightarrow$" between other value functions are well-established results. The minimum discounted cost $V_\gamma(s)$ as the limit of $V_{n,\gamma}(s)$ when $n \to \infty$ is due to (Sennott, 2009, Proposition 4.3.1), and the minimum average cost $J(s) = \inf_\pi J_\pi(s)$ as the limit of $(1-\gamma)V_\gamma(s)$ when $\gamma \to 1$ is due to (Sennott, 2009, Proposition 6.2.3). Since these are not the focus of our paper, we omit the formal statements of these results and the conditions under which they hold (usually satisfied in practice).

As we can see from equation 22, solving any value function can be done by solving the value function $V_{n,\gamma}$ for the discrete-time n-period discounted cost case. Suppose, for example, that we want to solve the value function $\hat{J}$ for the continuous-time average cost case. Due to the equivalence established by uniformization, we can instead solve for the discrete-time average cost J, which is obtained by solving $V_{n,\gamma}$ under a sufficiently large discount factor γ and then taking the limit as $n \to \infty$.

D.2 Detailed Walk-Through of An Example : Parallel M/M/1 Queues with Controlled Arrivals and Shared Beds

D.2.1 Uniformization of the Continuous-Time MDP

We illustrate the model components using a simple example of two parallel M/M/1 queues with controlled arrivals (CA) and uncontrolled departures (D). This constitutes a generalization of the problem introduced in Fig. 10.

We consider a hospital with two wards: one for Critical patients and one for General patients, each staffed by a dedicated team. Both wards share N_B beds. On average, λ_C Critical and λ_G General patients arrive per day. Treatment rates are μ_C and μ_G patients per day for Critical and General wards, respectively, with each team serving one patient at a time. Treated patients leave the system, releasing their beds. Upon arrival, a patient may be admitted if a bed is available; otherwise, the patient is rejected, incurring a penalty cost c_C (Critical) or c_G (General). Each admitted patient generates a holding cost ρ_h per unit time. The objective is to minimize the long-run average operating cost, with discount factor α.

Let $x = (x_C, x_G)$ denote the current queue lengths and let $e \in \{\mathtt{A}_C, \mathtt{A}_G, \mathtt{D}_C, \mathtt{D}_G\}$ denote the most recent event: arrival of a Critical patient, arrival of a General patient, departure of a Critical patient, or departure of a General patient. We define the post-event state as $s = (x, e)$. The action set is empty for departure events, but for arrivals the decision is to accept (1) or reject (0). Hence

$$\mathcal{A}_{(x,e)} = \{0, 1\}, \quad e \in \{\mathtt{A}_C, \mathtt{A}_G\}, \quad \text{if } x_C + x_G < N_B,$$

and

$$\mathcal{A}_{(x,e)} = \{0\}, \quad \text{if } x_C + x_G \geq N_B.$$

Interarrival and service times are naturally modeled as exponential (Poisson processes), even if not explicitly stated. Thus, the state transition time is exponentially distributed with rate

$$\lambda = \lambda_C + \lambda_G + \mu_C + \mu_G.$$

When the system is empty, "departure" events may still occur but do not alter the state (see 27, 28), allowing the same transition formulation to apply.

The state transition probability decomposes as

$$\hat{P}\left[(x', e') \,|\, (x, e), a\right] = \hat{P}_{\mathrm{x}}[x' \,|\, (x, e), a] \cdot \hat{P}_{\mathrm{e}}(e' \,|\, x'), \tag{24}$$

with transition probabilities for the queue lengths (with the notation $y^+ = \max(y, 0)$):

$$\begin{aligned}
\hat{P}_{\mathrm{x}}[x' \,|\, (x, \mathtt{A}_C), a] &= \mathbf{1}_{x'_C = x_C + a}, && (25)\\
\hat{P}_{\mathrm{x}}[x' \,|\, (x, \mathtt{A}_G), a] &= \mathbf{1}_{x'_G = x_G + a}, && (26)\\
\hat{P}_{\mathrm{x}}[x' \,|\, (x, \mathtt{D}_C), \emptyset] &= \mathbf{1}_{x'_C = (x_C - 1)^+}, && (27)\\
\hat{P}_{\mathrm{x}}[x' \,|\, (x, \mathtt{D}_G), \emptyset] &= \mathbf{1}_{x'_G = (x_G - 1)^+}. && (28)
\end{aligned}$$

The event probabilities are (this discretization step corresponds to *uniformization*, as introduced in Theorem D.1):

$$\hat{P}_{\mathrm{e}}(\mathtt{A}_C \,|\, x') = \frac{\lambda_C}{\lambda}, \quad \hat{P}_{\mathrm{e}}(\mathtt{A}_G \,|\, x') = \frac{\lambda_G}{\lambda}, \quad \hat{P}_{\mathrm{e}}(\mathtt{D}_C \,|\, x') = \frac{\mu_C}{\lambda}, \quad \hat{P}_{\mathrm{e}}(\mathtt{D}_G \,|\, x') = \frac{\mu_G}{\lambda}. \tag{29}$$

The cost structure consists of one-time rejection penalties c_C, c_G and a holding cost ρ_h per patient per unit time. The cost is given by

$$\hat{c}\big[(x, \mathtt{A}_{C/G}), 1\big] = c_{C/G} + \frac{\rho_h\,(x_C + x_G + 1)}{\lambda + \alpha}, \tag{30}$$

$$\hat{c}\big[(x, \mathtt{A}_{C/G}), 0\big] = c_{C/G} + \frac{\rho_h\,(x_C + x_G)}{\lambda + \alpha}, \tag{31}$$

$$\hat{c}[(x, \mathtt{D}_C)] = \frac{\rho_h\,(\max(x_C - 1, 0) + x_G)}{\lambda + \alpha}, \tag{32}$$

$$\hat{c}[(x, \mathtt{D}_G)] = \frac{\rho_h\,(\max(x_G - 1, 0) + x_C)}{\lambda + \alpha}. \tag{33}$$

D.2.2 Derivation of the Bellman Equation

Using the discrete-time embedded MDP of the original continuous-time problem we can write the following optimal Bellman equations (we recall that $x = (x_C, x_G)$).

$$V^*_{n+1}(x, \mathtt{A}_C) = \min\left\{V^*_n((x_C + 1, x_G)),\ c_C + V^*_n(x)\right\}, \tag{34}$$

$$V^*_{n+1}(x, \mathtt{A}_G) = \min\left\{V^*_n((x_C, x_G + 1)),\ c_G + V^*_n(x)\right\}, \tag{35}$$

$$V^*_{n+1}(x, \mathtt{D}_C) = V^*_n(((x_C - 1)^+, x_G)), \tag{36}$$

$$V^*_{n+1}(x, \mathtt{D}_G) = V^*_n((x_C, (x_G - 1)^+)). \tag{37}$$

where $V^*_n(x)$ is the value function defined on the queuing state x:

$$V^*_n(x) \triangleq \frac{\rho_h \cdot (x_C + x_G)}{\lambda + \alpha} + \gamma\left[\frac{\lambda_C}{\lambda}\,V^*_n(x, \mathtt{A}_C) + \frac{\lambda_G}{\lambda}\,V^*_n(x, \mathtt{A}_G) + \frac{\mu_C}{\lambda}\,V^*_n(x, \mathtt{D}_C) + \frac{\mu_G}{\lambda}\,V^*_n(x, \mathtt{D}_G)\right]. \tag{38}$$

It is worth noting the similarity between the post-event value functions $V^*_n(x, e)$ and the Q-functions commonly used in reinforcement learning, which represent post-action value functions.

E Details on Operator Theory

We provide an overview of the operator theory in constructing the operator-based Bellman equations. We focus on the results used in this paper. We refer the readers to Koole (2007) for a comprehensive discussion of this topic.

The central idea is to decompose the Bellman equation of an MDP into a sequence of operators Koole (2007). Each operator intuitively captures a distinct type of dynamic that arises as time progresses, such as randomness, decision-making, state transitions, or incurred costs.

E.1 An Illustrative Example

We recall the example presented in Appendix D.2:

We consider a hospital with two wards: one for Critical patients and one for General patients, each staffed by a dedicated team. Both wards share N_B beds. On average, λ_C Critical and λ_G General patients arrive per day. Treatment rates are μ_C and μ_G patients per day for Critical and General wards, respectively, with each team serving one patient at a time. Treated patients leave the system, releasing their beds. Upon arrival, a patient may be admitted if a bed is available; otherwise, the patient is rejected, incurring a penalty cost c_C (Critical) or c_G (General). Each admitted patient generates a holding cost ρ_h per unit time. The objective is to minimize the long-run average operating cost, with discount factor α.

This model involves several distinct dynamics:

- **Randomness**: Patient arrivals and departures occur at rates $\lambda_{C/D}$ and $\mu_{C/D}$, respectively.
- **Decision-making**: The system must decide whether to accept or reject a patient upon arrival.
- **State transitions**: The queue length may increase, stay the same, or decrease depending on the decisions made and event type.
- **Costs**: Rejection incurs a penalty, and holding patients generates a time-dependent cost.

These elements are all incorporated into the full Bellman equation. In operator theoery, we view the Bellman equation as a combination of operators, each capturing a specific aspect of the dynamics.

For the example above, the Bellman equation reads:

$$\begin{aligned} V_n^*(x) \triangleq\ & \frac{\rho_h \cdot (x_C + x_G)}{\lambda + \alpha} \\ & + \gamma \Bigg[\frac{\lambda_C}{\lambda} V_n^*(x, \mathtt{A}_C) + \frac{\lambda_G}{\lambda} V_n^*(x, \mathtt{A}_G) \\ & \qquad + \frac{\mu_C}{\lambda} V_n^*(x, \mathtt{D}_C) + \frac{\mu_G}{\lambda} V_n^*(x, \mathtt{D}_G) \Bigg]. \end{aligned} \tag{39}$$

with

$$V_{n+1}^*(x, \mathtt{A}_C) = \min\left\{V_n^*((x_C+1, x_G),\ c_C + V_n^*(x)\right\}, \tag{40}$$

$$V_{n+1}^*(x, \mathtt{A}_G) = \min\left\{V_n^*((x_C, x_G+1),\ c_G + V_n^*(x)\right\}, \tag{41}$$

$$V_{n+1}^*(x, \mathtt{D}_C) = V_n^*(((x_C-1)^+, x_G)), \tag{42}$$

$$V_{n+1}^*(x, \mathtt{D}_G) = V_n^*((x_C, (x_G-1)^+)). \tag{43}$$

We can identify and separate the stochastic dynamics into the uniformization operator $T_{\mathtt{unif}}$, the decision-making upon arrival with its corresponding impact and cost into $T_{\mathtt{CA}}$, and the patient departure mechanism into $T_{\mathtt{D}}$. The holding cost and discount factor are captured by the cost operator $T_{\mathtt{cost}}$. These are formally defined as follows:

$$T_{\mathtt{CA,C}}\left[V_n^*(x)\right] = V_{n+1}(x, \mathtt{A}_C), \quad T_{\mathtt{CA,G}}\left[V_n^*(x)\right] = V_{n+1}(x, \mathtt{A}_G), \tag{44}$$

$$T_{\mathtt{D,C}}\left[V_n^*(x)\right] = V_{n+1}(x, \mathtt{D}_C), \quad T_{\mathtt{D,G}}\left[V_n^*(x)\right] = V_{n+1}(x, \mathtt{D}_G), \tag{45}$$

$$\begin{aligned} T_{\mathtt{unif}}[U_1(x), U_2(x), U_3(x), U_4(x)] = & \tfrac{\lambda_C}{\lambda} \cdot U_1(x) + \tfrac{\lambda_G}{\lambda} \cdot U_2(x) \\ & + \tfrac{\mu_C}{\lambda} \cdot U_3(x) + \tfrac{\mu_G}{\lambda} \cdot U_4(x). \end{aligned} \tag{46}$$

$$T_{\mathtt{cost}}\left[U(x)\right] = \rho_h \cdot (x_C + x_G)/(\lambda + \alpha) + \gamma \cdot U(x). \tag{47}$$

Then the Bellman equation equation 39 can be rewritten as V_n^* going through an *operator graph* to get V_{n+1}^*:

$$V_{n+1}^*(x) = T_{\mathtt{cost}}\left\{T_{\mathtt{unif}}\left(T_{\mathtt{CA,C}}\left[V_n^*(x)\right], T_{\mathtt{CA,G}}\left[V_n^*(x)\right], T_{\mathtt{D,C}}\left[V_n^*(x)\right], T_{\mathtt{D,G}}\left[V_n^*(x)\right]\right)\right\}. \tag{48}$$

The motivation for introducing operators as a tool for autoformalism is twofold. First, it offers a structured framework wherein identifying the Bellman equation reduces to two subtasks: (i) identifying the relevant operators, and (ii) specifying the operator graph. In this work, we focus on a class of problems for which the second task—the structure of the graph—is solved in advance (Theorem 4.1). Thus, the only remaining task is to identify the appropriate operators in this graph. Second, the operator-based framework enables the automatic derivation of structural properties of the value function, by analyzing the propagation behavior of the operators within the graph.

For instance, consider the space of convex value functions $Conv$, defined by the property that $2V_n^*(x+1) \leq V_n^*(x) + V_n^*(x+2)$ for all $x \geq 0$. We can show that under any parameter values—provided the refusal cost is positive and the holding cost $\rho_h(x)$ is convex—the previously defined operators preserve convexity:

$$V^* \in Conv \implies T_{\texttt{CA}}(V^*), T_{\texttt{D}}(V^*), T_{\texttt{unif}}(V^*), T_{\texttt{cost}}(V^*) \in Conv.$$

Consequently, we have the propagation result:

$$V_n^* \in Conv \quad \implies \quad V_{n+1}^* = T_{\texttt{cost}}\Big\{T_{\texttt{unif}}\Big(T_{\texttt{CA,G}}[V_n^*],\ T_{\texttt{CA,C}}[V_n^*],$$
$$T_{\texttt{CD,G}}[V_n^*],\ T_{\texttt{CD,C}}[V_n^*]\Big)\Big\} \in Conv. \tag{49}$$

From this, it follows that $V^* \in Conv$, which in turn implies that the optimal policy π^* is *threshold*. Specifically, there exist two thresholds $n_{\text{T},C}$ and $n_{\text{T},G}$ such that an arriving Critical (resp. General) patient is accepted if and only if $x_C + x_G \leq n_{\text{T},C}$ (resp. $n_{\text{T},G}$)

This approach is generalizable: whenever the Bellman equation can be decomposed into known operators for which we have propagation results, structural properties of the value function—and hence of the optimal policy—can be deduced, provided these operators share a common invariant function space. In the following sections, we list the operators considered and detail their respective propagation properties.

E.2 Operators (Used in This Work)

We introduce the operators used in this work; additional operators can be found in Koole (2007). In the following, ε_i denotes the canonical basis of $\mathbb{R}^k$, where k represents the number of queues.

- $T_{\texttt{cost}}\,[V^*(x)] = C(x) + \gamma V^*(x)$
- $T_{\texttt{unif}}\,[V_1^*(x), \ldots, V_J^*(x)] = \sum_{j=1}^{J} p(j) V_j(x) \quad \text{with } \sum_{j=1}^{J} p(j) = 1$
- $T_{\texttt{CA(i)}}\,[V^*(x)] = \min\{(V^*(x) + c_1, V^*(x+\varepsilon_i) + c_2)\}$
- $T_{\texttt{D1(i)}}\,[V^*(x)] = V^*((x-\varepsilon_i)^+)$
- $T_{\texttt{D(i)}}\,[V^*(x)] = \mu(x_i)V^*((x-\varepsilon_i)^+) + (1-\mu(x_i))V^*(x))$
- $T_{\texttt{CD}}\,[V^*(x)] = \begin{cases} \min\{(c_1 + V^*(x), c_2 + V^*[(x-\varepsilon_i)^+]\} & \text{if } x_i > 0 \\ c_1 + V(x) & \text{otherwise.} \end{cases}$
- $T_{\texttt{TD1(i,j)}}\,[V^*(x)] = \begin{cases} V^*(x - \varepsilon_i + \varepsilon_j) & \text{if } x_i > 0 \\ V^*(x) & \text{otherwise.} \end{cases}$
- $T_{\texttt{CTD(i,j)}}\,[V^*(x)] = \begin{cases} \min\{c_1 + V^*(x), c_2 + V^*(x - \varepsilon_i + \varepsilon_j)\} & \text{if } x_i > 0 \\ V^*(x) & \text{otherwise.} \end{cases}$

The operator $T_{\texttt{cost}}$ represents the cost operator, while $T_{\texttt{unif}}$ corresponds to the uniformization operator.

The operator $T_{\text{CA},i}$ represents controlled arrivals to queue i, whereas $T_{\text{D1},i}$ models a standard departure. The operator $T_{\text{D},i}$ represents departures in a multi-server queue and is specifically used in Example 1. Controlled departures from queue i are denoted by $T_{\text{CD},i}$.

To model tandem queues, we use the operator T_{TD1}, which describes the transition of a customer from queue i to queue j. Similarly, T_{CTD} represents controlled tandem departures.

E.3 Set of Special Functions

We introduce the set of functions used in the propagation results.

– $V \in I(i)$ if
$$V(x) \leq V(x+e_i)$$
for all $x \in \mathbb{N}^k, 1 \leq i \leq k$,
– $I = I(1) \cap \cdots \cap I(m)$,
– $V \in \text{UI}(i)$ if
$$V(x+e_{i+1}) \leq V(x+e_i)$$
for all x and $1 \leq i < k$,
– $\text{UI} = \text{UI}(1) \cap \cdots \cap \text{UI}(k-1)$,
– $V \in Cx(i)$ if
$$2V(x+e_i) \leq V(x) + V(x+2e_i)$$
for all x and $1 \leq i \leq k$,
– $Cx = Cx(1) \cap \cdots \cap Cx(k)$,
– $V \in \text{Super}(i,j)$ if
$$V(x+e_i) + V(x+e_j) \leq V(x) + V(x+e_i+e_j)$$
for all x and $1 \leq i < j \leq k$,
– $\text{Super} = \cap_{1 \leq i < j \leq k} \text{Super}(i,j)$,
– $V \in \text{Sub}(i,j)$ if
$$V(x) + V(x+e_i+e_j) \leq V(x+e_i) + V(x+e_j)$$
for all x and $1 \leq i < j \leq k$,
– $\text{Sub} = \cap_{1 \leq i < j \leq k} \text{Sub}(i,j)$,
– $V \in \text{SuperC}(i,j)$ if
$$V(x+e_i) + V(x+e_j+e_i) \leq V(x+e_j) + V(x+2e_i)$$
for all x and $1 \leq i,j \leq k, i \neq j$
– $\text{SuperC} = \cap_{1 \leq i,j \leq k : i \neq j} \text{SuperC}(i,j)$,
– $V \in \text{SubC}(i,j)$ if
$$V(x+e_i) + V(x+e_j+e_i) \leq V(x) + V(x+2e_i+e_j)$$
for all x and $1 \leq i,j \leq k, i \neq j$
– $\text{SubC} = \cap_{1 \leq i,j \leq k : i \neq j} \text{SubC}(i,j)$,
– $V \in \text{MM}(i,j)$ if
$$V(x) + V(x+d_i+d_j) \leq V(x+d_i) + V(x+d_j)$$
for all x and $1 \leq i < j \leq k$ such that $x+d_i, x+d_j \in \mathbb{N}^k$,
with $d_1 = e_1, d_k = -e_k + e_{k+11}, k = 2, \ldots, k-1$, and $d_k = -e_k$,
– $\text{MM} = \cap_{1 \leq i < j \leq k} \text{MM}(i,j)$.

Further explanations for each can be found in Koole (2007). The following inequalities hold for these sets:

$$\begin{aligned}
&\text{Super}(i,j) \cap \text{SuperC}(i,j) \subset Cx(i) \\
&\text{Sub}(i,j) \cap \text{SubC}(i,j) \subset Cx(i) \\
&\text{MM} \subset \text{Super} \cap \text{SuperC} \subset \text{Super} \cap Cx
\end{aligned}$$

We construct a non-trivial inclusion basis that satisfies the rules introduced in Appendix G.3 and corresponds to the given inequalities. We define $\mathcal{B}$ as the union of all the spaces introduced above, excluding superspaces such as *SuperC*. Instead, we retain only the sets of the form SuperC(i, j) to prevent non-trivial equalities that cannot be derived solely through *intersection* or *augmentation*.

Next, we decompose inequalities to ensure they follow the standard form $A \subset \{i\}$. For example, an inclusion such as

$$\text{MM} \subset \text{Super} \cap \text{SuperC} \tag{50}$$

is rewritten as multiple separate inequalities, such as MM $\subset$ Super(i, j) for specific indices i, j. The resulting set of inequalities constitutes the non-trivial inclusion basis, which satisfies the necessary constraints.

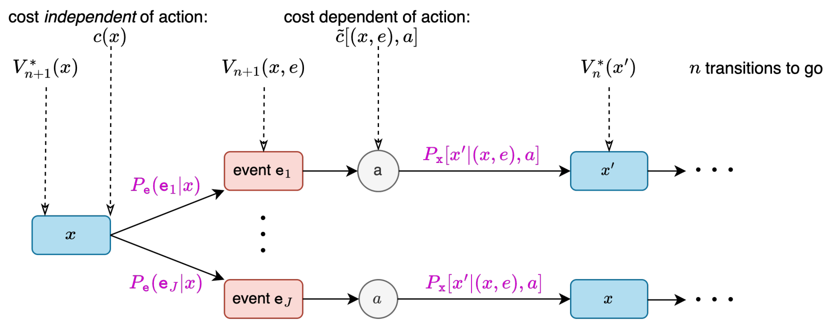
E.4 Propagation Results

We present some of the propagation results for the operators used in this work:

$- T_{\text{CA(i)}} : I \to I,\ \text{UI} \to \text{UI},\ Cx(i) \to Cx(i), \text{Super}(i,j) \to \text{Super}(i,j),\ \text{Sub} \to \text{Sub},$
$\text{Super}(i,j) \cap \text{SuperC}(i,j) \to \text{SuperC}(i,j), \text{Super}(i,j) \cap \text{SuperC}(j,i) \to \text{SuperC}(j,i),$
$\text{Sub}(i,j) \cap \text{SubC}(i,j) \to \text{SubC}(i,j), \text{Sub}(i,j) \cap \text{SubC}(j,i) \to \text{SubC}(j,i),\ \text{MM} \to \text{MM for } i = 1;$
$- T_{D1(i)} : I \to I,\ I \cap \text{UI} \to \text{UI for } i = m,\ I(i) \cap Cx(i) \to Cx(i),\ Cx(j) \to Cx(j)$
$\text{for } j \neq i,\ \text{Super} \to \text{Super}, \text{Sub} \to \text{Sub},\ \text{SuperC}(j,k) \to \text{SuperC}(j,k)\ \ (j,k \neq i),$
$I(i) \cap \text{SuperC}(i,j) \to \text{SuperC}(i,j)\ \ (i \neq j),\ Cx(j) \cap \text{SuperC}(j,i) \to \text{SuperC}(j,i)\ \ (j \neq i),$
$\text{SubC}(j,k) \to \text{SubC}(j,k)\ \ (j,k \neq i),\ I(i) \cap \text{SubC}(i,j) \to \text{SubC}(i,j)(j \neq i),$
$Cx(j) \cap \text{SubC}(j,i) \to \text{SubC}(j,i), \text{UI} \cap \text{MM} \to \text{MM for } i = m;$
$- T_{\text{TD1(i)}} : I \to I,\ \text{UI} \to \text{UI for } i < m, \text{UI} \cap \text{MM} \to\ \text{for } i < m, \text{UI} \cap Cx \cap \text{Super} \to$
$Cx \text{ for } i < m, \text{UI} \cap Cx \cap \text{Super} \to \text{Super for } i < m;$

For the rest of the operators, again refer to Koole (2007).

F Proof of Theorem 4.1



State transition dynamics and costs during a sample path of the discrete-time embedded MDP.

For any MDP $(\mathcal{S}, \mathcal{A}_s, c, P, \gamma)$, the standard Bellman equation for the value function $V_{n+1}(s)$ on the full state s can be written as:

$$V_{n+1}(s) = \min_{a \in \mathcal{A}_s} \left\{ c(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V_n(s') \right\}. \tag{51}$$

Separating the controllable state and the event $s = (x, e)$ and using the definition of event-based MDPs equation 5 we have

$$V_{n+1}(x,e) = \min_{a \in \mathcal{A}_{(x,e)}} \left\{ c[(x,e),a] + \gamma \sum_{(x',e')} P\left[(x',e') \mid (x,e),\ a\right] \cdot V_n(x',e') \right\} \tag{52}$$

$$= \min_{a \in \mathcal{A}_{(x,e)}} \left\{ c[(x,e),a] + \gamma \sum_{(x',e')} P_x\left[x' \mid (x,e),\ a\right] \cdot P_e(e' \mid x') \cdot V_n(x',e') \right\} \tag{53}$$

$$= \min_{a \in \mathcal{A}_{(x,e)}} \left\{ c[(x,e),a] + \sum_{x'} P_x\left[x' \mid (x,e),\ a\right] \cdot \underbrace{\gamma \sum_{e'} P_e(e' \mid x') \cdot V_n(x',e')}_{\triangleq V^*_n(x')} \right\} \tag{54}$$

$$= \min_{a \in \mathcal{A}_{(x,e)}} \left\{ c[(x,e),a] + \sum_{x'} P_x\left[x' \mid (x,e),\ a\right] \cdot V^*_n(x') \right\}, \tag{55}$$

where we define the value function $V^*_n(x)$ on the controllable state :

$$V^*_n(x) = \gamma \sum_e P_e(e \mid x) V_n(x,e). \tag{56}$$

$V^*_n(x)$ is the value of the state immediately after an action is taken but before the waiting time until the next event. Because there is a temporal gap between V^*_n and V_n, the discount factor must be applied at this stage to account for that delay.

We can always decompose the cost $c[(x,e),a]$ into two parts: (1) a cost $c(x)$ that depends only on the controllable state (e.g., the holding cost in the M/M/1 example), and (2) a cost $\tilde{c}[(x,e),a]$ that depends on the full state-action pair, namely

$$c[(x,e),a] = c(x) + \tilde{c}[(x,e),a].$$

Note that if the component $c(x)$ does not exist, we can always set $c(x) = 0$ and $\tilde{c}[(x,e),a] = c[(x,e),a]$.

Then the Bellman equation in equation 55 can be rewritten as

$$V_{n+1}(x,e) = c(x) + \min_{a \in \mathcal{A}_{(x,e)}} \left\{ \tilde{c}[(x,e),a] + \sum_{x'} P_{\mathrm{x}}\left[x' \,|\, (x,e),\ a\right] \cdot V_n^*(x') \right\}, \tag{57}$$

and the value function $V_{n+1}^*(x)$ can be rewritten as

$$\begin{aligned} & V_{n+1}^*(x) \\ = \quad & \gamma \sum_e P_{\mathrm{e}}(e \,|\, x) V_{n+1}(x,e) \qquad (58) \\ = \quad & \gamma \sum_e P_{\mathrm{e}}(e \,|\, x) \left[c(x) + \min_{a \in \mathcal{A}_{(x,e)}} \left\{ \tilde{c}[(x,e),a] + \sum_{x'} P_{\mathrm{x}}\left[x' \,|\, (x,e),\ a\right] \cdot V_n^*(x') \right\} \right] \\ = \quad & \underbrace{\gamma c(x)}_{\triangleq c'(x)} + \gamma \sum_e P_{\mathrm{e}}(e \,|\, x) \left[\min_{a \in \mathcal{A}_{(x,e)}} \left\{ \tilde{c}[(x,e),a] + \sum_{x'} P_{\mathrm{x}}\left[x' \,|\, (x,e),\ a\right] \cdot V_n^*(x') \right\} \right]. \end{aligned}$$

Therefore, we can go from $V_n^*(x)$ to $V_{n+1}^*(x)$ through the following three operators:

$$\begin{cases} T_{\mathrm{e}_j}\left[V_n^*(x)\right] = V_{n+1}(x, \mathrm{e}_j) = \min_{a \in \mathcal{A}_{(x,e)}} \left\{ \tilde{c}[(x,e),a] + \sum_{x'} P_{\mathrm{x}}\left[x' \,|\, (x,e),\ a\right] \cdot V_n^*(x') \right\} \\ T_{\texttt{unif}}\left[U_1(x), \ldots, U_\ell(x)\right] = \sum_{j=1}^{\ell} P(\mathrm{e}_j \,|\, x) \cdot U_j(x) \\ T_{\texttt{cost}}\left[U(x)\right] = c'(x) + \gamma \cdot U(x) \end{cases}$$

The operator-based Bellman equation on the value function $V_n^*(x)$ can be written as

$$V_{n+1}^*(x) = T_{\texttt{cost}}\left\{ T_{\texttt{unif}}\left(T_{\mathrm{e}_1}\left[V_n^*(x)\right], \ldots, T_{\mathrm{e}_\ell}\left[V_n^*(x)\right] \right) \right\}. \tag{59}$$

G Algorithms for Identifying Structural Results (Section 4.3)

In this section, we provide a detailed discussion of the algorithm used to identify structural properties of the solution from the operator graph (Section 4.3).

G.1 Why It Is Challenging to Identify Structural Results?

Given the operator graph, our goal is to deduce structural properties of the optimal policy and the value function $V^*(x)$. Consider the example from Section 3, further detailed in Appendix E.1. In that case, one can show that $V^*(x)$ is convex, which implies that the optimal acceptance policy is threshold. This follows from the fact that convexity is *propagated* through each operator in the graph. Consequently, convexity is preserved by the entire Bellman equation. Since convexity is a fixed property under this composition of operators, it must also be a property of the fixed point of the Bellman equation—namely, the optimal value function.

More generally, consider a problem where the operator graph consists of operators $(T_1, \ldots, T_k)$. For each operator, we are given a list of functional spaces (or properties) that it propagates. Our objective is to identify a common functional space propagated by all operators; the optimal value function must then belong to this space.

In practice, we do not have explicit lists of all propagated spaces, but only a set of primitive spaces from which additional spaces can be generated. New propagated spaces may be obtained by applying the two fundamental set operations, intersection and union. Indeed, if T propagates A and B, then it propagates both $A \cap B$ and $A \cup B$.

In principle, one could therefore consider the closure of the initial families under both intersection and union. However, because the initial spaces are only intersections of basis spaces, and because we are ultimately interested in the smallest common propagated space across all operators, it suffices to consider closure under intersection only.

A brief intuitive justification is as follows. Suppose a space of the form $A \cup B$ (with $A \neq B$) appears in the propagated closure for the operators. Writing this set in its Disjunctive Normal Form (a union of intersections), we obtain a union of terms, each of which must be propagated individually by all operators, since unions do not appear in the primitive families. If we now replace every union in this representation by an intersection, we obtain a new space that is propagated by all operators and is contained in the original set. Consequently, any common propagated space built using unions admits a smaller counterpart formed purely through intersections. As a result, the smallest common propagated space is built entirely from intersections, and closure under intersections is therefore sufficient.

Our problem is therefore to find the smallest element (under $\subseteq$) among all intersection-closures of the propagated spaces associated with each operator. This task is made more difficult by the presence of non-trivial inclusion relationships between these spaces, which may cause distinct expressions to represent the same underlying space.

To illustrate these challenges, consider the example discussed in Section 4.3, involving two operators T_1 and T_2 and six properties A through F:

$$\begin{cases} T_1 \text{ propagates } A \cap B,\ E \cap C,\ D, \\ T_2 \text{ propagates } C \cap A,\ D \cap F,\ B, \\ B \cap C \subset E. \end{cases}$$

Taking the closure under intersection, we obtain for T_1 the family of propagated spaces

$$\mathcal{P}_1 = \{A \cap B,\ E \cap C,\ D,\ A \cap B \cap D,\ E \cap C \cap E,\ A \cap B \cap E \cap C,\ A \cap B \cap E \cap C \cap D\}.$$

Similarly, for T_2 we obtain

$$\mathcal{P}_2 = \{C \cap A,\ D \cap F,\ B,\ C \cap A \cap B,\ D \cap F \cap B,\ C \cap A \cap D \cap F,\ C \cap A \cap D \cap F \cap B\}.$$

At first glance, the intersection $\mathcal{P}_1 \cap \mathcal{P}_2$ appears empty. However, using the inclusion $B \cap C \subset E$, we see that

$$A \cap B \cap C = A \cap B \cap C \cap E,$$

which shows that $A \cap B \cap C$ belongs to both families. Hence,

$$\mathcal{P}_1 \cap \mathcal{P}_2 = \{A \cap B \cap C\}.$$

In general, for a typical operator, the number of primitive propagated spaces may grow quadratically with the size of the state space, while the intersection-closure of these spaces can grow exponentially in the number of primitives. Consequently, a naive approach that explicitly computes every propagated space for each operator and then intersects them is computationally infeasible.

To address this, we introduce a dynamic programming algorithm that reduces both the time and memory complexity of the procedure. The following sections present a detailed discussion of this algorithm, including a proof of convergence, an analysis of its computational complexity and a running example.

G.2 Algorithm Description in Pseudo-Code

Algorithm 1 Find smallest common propagated space

```
Require: $\mathcal{O}$ : A set of operators for which we know propagation results.
Require: $\mathcal{B}$ : A set of basis function spaces.
Require: $P$ : The propagation results for all the operators in $\mathcal{O}$.
Require: $(T_1, \dots, T_J)$ list of operators in the graph.
Require: $R$ the non-trivial inclusion basis.
Ensure: $\mathcal{F}$ the smallest common propagated space.
 1: Create a mapping $m$ between $\{1, \dots, K\}$ and $\mathcal{B}$, with $K = \#\mathcal{B}$.
 2: Create the sets $\mathcal{P}_j \in \mathcal{P}(\{1, \dots, K\})$ for each $j$ based on $P$ and $m$.
 3: Create a dictionary $\mathcal{R}$ based on $R$ and $m$ such that if $\{i\} \subset^* U$ then $U \in \mathcal{R}[i]$.
 4: $\mathcal{P}_j^0 \leftarrow \mathcal{P}_j$ for all $j$
 5: $n \leftarrow 0$
 6: while $\{P\}_{n=0}^{\infty}$ does not converge do
 7:     $(p_k^n)_{k \leq J} \leftarrow \left(\bigcup_{p \in \mathcal{P}_j^n} p\right)_{k \leq J}$
 8:     for $j$ in $(1, \dots, J)$ do
 9:         $\mathcal{P}_j^{n+1} \leftarrow$Refine_propagated_space$(\mathcal{P}_j^n, (p_k^n)_{k \leq J}, \mathcal{R})$
10:     end for
11:     $n \leftarrow n + 1$
12: end while
13: $p^\infty \leftarrow \bigcup_{p \in \mathcal{P}_1^{n-1}} p$
14: Create $\mathcal{F}$ by mapping back $p^\infty$ to $\mathcal{B}$ using $m$
15: Return $\mathcal{F}$
```

Algorithm 2 Refine Propagated Spaces

```
Require: $\mathcal{P}_j^n$: Set of elements of $\mathcal{P}(\{1, \dots, K\})$
Require: $(p_k^n)_{k \leq J}$: List of sets of integers in $\{1, \dots, N\}$, $\mathcal{B}_k^n$ is the set of elements that appear in
    $\mathcal{P}_k^n$.
Require: $\mathcal{R}$: Dictionary corresponding to the non-trivial inclusion basis. If $U \subset^* \{i\}$ then $U \in \mathcal{R}[i]$
Ensure: $\mathcal{P}_j^{n+1}$: Refined list of spaces consistent across all operators. ($\mathcal{P}_j^{n\prime}$ with the notation of the
    subsection G.3.3)
 1: $\mathcal{P}_j^{t+1} \leftarrow \mathcal{P}_j^t$
 2: for each $a$ in $\mathcal{P}_j^t$ do
 3:     for each $i$ in $a$ do
 4:         if Do_$i$_covers_$\mathcal{P}_j^n$$(i, \mathcal{R}, p_j^n)$ then
 5:             break
 6:         end if
 7:         for each $p_k^n$ in $(p_k^n)_{k \neq j}$ do
 8:             if $i \notin p_k^n$ then
 9:                 Remove $a$ from $\mathcal{S}_j^{n+1}$
10:                 break
11:             end if
12:         end for
13:     end for
14: end for
15: return $\mathcal{S}_j^{n+1}$
```

Algorithm 3 Do i covers $\mathcal{P}_j^n$

```
Require: $i$: An integer
Require: $\mathcal{R}$: Non-trivial inclusion basis.
Require: $p_j^n$: Set of elements of $\mathcal{P}(\{1, \dots, K\})$, it is the set of elements that appear in $\mathcal{P}_j^n$.
Ensure: $covers$: Boolean indicating whether $i$ covers $\mathcal{P}_j^n$
 1: $covers \leftarrow$ False
 2: if $i$ is a key of $\mathcal{R}$ then
 3:     for $C$ in $\mathcal{R}[i]$ do
 4:         if $C \subset p_j^n$ then
 5:             $covers \leftarrow$ True
 6:             Break
 7:         end if
 8:     end for
 9: end if
10: return $covers$
```

G.3 Proof of Theorem 4.2

G.3.1 Mathematical Definition of the Problem

In the framework introduced by Koole (2007), structural properties of the optimal value function and policy can be derived from the propagation behavior of the operators forming the operator graph representation of the Bellman equation.

Formally, consider a graph of operators $T_1, \dots, T_k$, where n is the dimension of the state space, and value functions belong to $\mathbb{R}^{\mathbb{N}^n}$. Define a base family of K subspaces of $\mathbb{R}^{\mathbb{N}^n}$, denoted as $\mathcal{B} = \{B_1, \dots, B_K\}$. In this framework, these base subspaces are generated systematically, with details provided in ??.

From these base subspaces, we define the set of function spaces for which propagation results can be derived as:

$$\mathcal{S} = \text{span}\, \mathcal{B} = \Big\{ \bigcap_{i \in Q} B_i \mid Q \in \mathcal{P}(\{1, \dots, K\}) \Big\},$$

where $\mathcal{P}(\{1, \dots, K\})$ is the power set of $\{1, \dots, K\}$. We equip $\mathcal{S}$ with a non-trivial $\subset$ relationship. We discuss more precisely what it means bellow.

For each operator T_j, we can identify a set of propagated spaces $\mathcal{S}_j$, which is a subset of $\mathcal{S}$. The objective is to determine the smallest common space under $\subset$ across all $\mathcal{S}_j$:

$$\mathcal{F} = \min \bigcap_{j=1}^{N} \text{span}\, \mathcal{S}_j.$$

$\mathcal{F}$ is indeed the smallest element of $\mathcal{S}$ that is propagated through all the operators and therefore through the overall graph. From Theorem 4.1, we can conclude that $V^* \in \mathcal{F}$ and derive structural results for the optimal policies.

Definition of the non-trivial $\subset$ relationship.

The ordering relationship $\subset$ must be defined according to the following rules:

- **Base (trivial relationships):** The relationship starts from a set of trivial inclusion relationships, such as $A \cap B \subset B$.
- **Generation rule for non-trivial relationships:** We add a finite family of *non-trivial* inclusions of the form $A \subset \{i\}$. From these, all other needed inclusions must be generated by the following rule alone (without invoking the general transitivity rule to create new ones):

  $$\text{If } A \subset B \text{ and } C \subset D, \text{ then } A \cap C \subset B \cap D.$$

  This rule should be sufficient to produce a consistent ordering. We refer to this finite set of non-trivial inclusions as the *non-trivial inclusion basis*. A similar constraint hold for equalities, in particular we should not have to use the rule $(A \subset B) \wedge (B \subset A) \implies A = B$ and only the following ones :

1. **Intersection** If $A = B$ and $C = D$, then $\big(A \cap C\big) = \big(B \cap D\big)$.
2. **Augmentation:** If $A = B$ and $A \subset C$, then $\big(A \cap C\big) = B$.

$\subset$ is fully defined by the non-trivial inclusion basis.

The idea behind this constraint is that we can check if any inequality is verified in a constant time if we have the non-trivial inclusion basis and non-trivial equalities have a common form. It is used in lemma G.2 and in lemma G.3. We show that these rules are verified in our framework in section E.3.

G.3.2 Reformulation of the Problem

We reformulate the problem to better align it with an algorithmic approach:

- Replace $\mathcal{B}$ with $\mathcal{I} = \{1, \ldots, K\}$. In other words, each space is identified by an index.
- Replace $\mathcal{S}$ with $\mathcal{P}(\mathcal{I}) = \mathcal{P}(\{1, \ldots, K\})$.
- Replace $\subset$ with an ordering relationship $\subset^*$ over $\mathcal{P}(\mathcal{I})$. This extends the canonical inclusion relationship on $\mathcal{P}(\mathcal{I})$. For example, $\bigcap_{u \in U} B_u \subset \bigcap_{v \in V} B_v$ becomes $V \subset^* U$.
- The non canonical $\subset^*$ implies a non canonical equality relationship $=^*$ defined as : if $U \subset^* V$ and $V \subset^* U$ then $U =^* V$. Notably we can now have equalities that are non trivial, such as $(1, 2, 3) =^* (2, 3)$.
- For each relationship in the *non-trivial inclusion basis* $\{i\} \subset^* U$, we introduce a tuple representation $r = (i, U)$. Denote $\mathcal{R}$ as the set of all such inclusion relationships.
- Replace $\mathcal{S}_j$ with the corresponding $\mathcal{P}_j \subset \mathcal{P}(\mathcal{I})$, such that $\mathcal{S}_j = \{\bigcap_{i \in Q} B_i \mid Q \in \mathcal{P}_j\}$.

The closure under intersection, span $\mathcal{S}_j$, is equivalent to span $\mathcal{P}_j$, which is defined as the closure under two operations: *union* and the generation of new sets using the extended ordering relationship $\subset^*$. In other words if $U \cup W \in \text{span } \mathcal{P}_j$ and $V \subset^* U$, then the set $V \cup U \cup W$ is also included in span $\mathcal{P}_j$.

The problem now becomes finding the biggest common element for $\subset^*$ across all span $\mathcal{P}_j$:

$$\mathcal{I}_{\mathcal{F}} =^* \max\left(\bigcap_{j=1}^{N} \text{span } \mathcal{P}_j.\right)$$

Here, the intersection is taken with respect to the $=^*$ relationship.

This reformulation is advantageous for algorithmic purposes because the implicit relationships among functional spaces, captured by $\subset^*$ and $=^*$ in $\mathcal{P}(\mathcal{I})$, are made explicit through the set $\mathcal{R}$.

G.3.3 Some notations

Below we introduce several pieces of notation that will be used in our construction. Throughout, let $\mathcal{A}_1, \ldots, \mathcal{A}_N \subset \mathcal{P}(\mathcal{I})$, and let i be any index in $\mathcal{I}$.

- We say that i *appears in each family* $(\mathcal{A}_1, \ldots, \mathcal{A}_N)$ if, for every $j \in \{1, \ldots, N\}$, there exists at least one set $a \in \mathcal{A}_j$ such that $i \in a$. This notion naturally extends to a subset $I \subseteq \mathcal{I}$: we say that I *appears in each family* if all $i \in I$ *appear in each family* according to the above definition. If we only consider one family $\mathcal{A}$, we say that i *appears* in $\mathcal{A}$.
- For a particular $\mathcal{A}_j$, we say i *covers* $\mathcal{A}_j$ if there exists $(i, U) \in \mathcal{R}$ such that U appears in $\mathcal{A}_j$.
- We write $i \ \triangleleft \ \big(j, (\mathcal{A}_1, \ldots, \mathcal{A}_N)\big)$ if either i appears in each family $(\mathcal{A}_1, \ldots, \mathcal{A}_N)$, *or* i covers $\mathcal{A}_j$.

Finally, given $\mathcal{A}_1, \dots, \mathcal{A}_J \subset \mathcal{P}(\mathcal{I})$, for each $j \in \{1, \dots, J\}$ define

$$\mathcal{A}'_j \;=\; \Big\{ a \in \mathcal{A}_j \,\Big|\, \forall i \in a : \, i \,\triangleleft\, \big(j, \, (\mathcal{A}_1, \dots, \mathcal{A}_J)\big) \Big\}.$$

We then define the function

$$F\big(\mathcal{A}_1, \dots, \mathcal{A}_J\big) \;=\; \big(\mathcal{A}'_1, \, \dots, \, \mathcal{A}'_J\big).$$

G.3.4 The algorithm

A straightforward yet naive method would be to construct the set $\operatorname{span} \mathcal{P}_j$ for each j by exhaustively applying every closure rule, then intersect these sets at the end. However, due to the non-trivial inclusion relationships, this expansion can be both difficult to implement and prohibitively large in memory usage—scaling exponentially with the size of each $\mathcal{P}_j$.

Instead, we propose a more efficient procedure that avoids this blowup. We form a sequence of tuples

$$\{P^n\}_{n=0}^{\infty} \quad \text{with} \quad P^0 = (\mathcal{P}_1, \dots, \mathcal{P}_J) \quad \text{and} \quad P^{n+1} = F(P^n).$$

It will be shown below that this sequence converges to a stationary limit

$$P^{\infty} = (\mathcal{P}_1^{\infty}, \dots, \mathcal{P}_J^{\infty}).$$

We then define

$$p^{\infty} \;=\; \bigcup_{p \in \mathcal{P}_1^{\infty}} p$$

and prove that for each j,

$$p^{\infty} \;=^*\; \bigcup_{p \in \mathcal{P}_j^{\infty}} p, \quad \text{and} \quad p^{\infty} \;=^*\; \mathcal{I}_{\mathcal{F}} \;=^*\; \max\Big(\bigcap_{j=1}^{N} \operatorname{span} \mathcal{P}_j \Big).$$

In other words, the family of subspaces $\{B_k \mid k \in p^{\infty}\}$ constitutes the smallest space that is propagated through the entire operator graph.

G.3.5 Proof of convergence

Fixed points correspond to common propagated spaces.

Lemma G.1. *Let $P = (P_1, \dots, P_J)$ be a tuple of subsets in $\mathcal{P}(\mathcal{I})$ such that the corresponding function spaces of each P_j are propagated by operators T_j. Define $p_j = \bigcup_{p \in P_j} p$ for each j. If P is a fixed point of F, then $p_i =^* p_j$ for all i, j, and the set $\mathcal{F} = \{B_i \mid i \in p_1\}$ is a common propagated space across the operators.*

Proof. Since P is a fixed point of F, every index $i \in p_1$ must satisfy

$$i \,\triangleleft\, (1, \, (P_1, \dots, P_J)).$$

There are two ways this can happen:

1. i **appears in each family** $(P_1, \dots, P_J)$**.** In this case, i belongs to p_j for every j.

2. i **covers** P_1**.** Here, there exists $(i, U) \in \mathcal{R}$ such that $U \subset p_1$. By definition, $\{i\} \subset^* U$. Consequently, $p_1 \setminus \{i\} =^* p_1$.

Combining these observations, define

$$p'_1 \;=\; \{\, i \in p_1 \mid i \text{ appears in each family } (P_1, \dots, P_J) \}.$$

From the cases above, we see $p'_1 =^* p_1$, and in fact $p'_1 =^* p'_j$ for all j. Hence,

$$p_1 =^* p_2 =^* \cdots =^* p_J.$$

Finally, let $\mathcal{F}_j = \{B_i \mid i \in p_j\}$. Since each $\mathcal{F}_j$ is propagated by T_j and $\mathcal{F}_j =^* \mathcal{F}_i$ for all i, j, we conclude that $\mathcal{F} = \{B_i \mid i \in p_1\}$ is indeed a single common propagated space for all the operators. □

Equalities under $=^*$ share a useful structure.

Lemma G.2. *If $U =^* V$, then $U$and Vcan be decomposed as $U = U_1 \cup U_2$ and $V = V_1 \cup V_2$, where $U_1 = V_1$, $U =^* U_1$, and $V =^* V_1$.*

Proof. In our setting, and due to the specialized nature of the $\subset^*$ relation, *every* non-trivial equality under $=^*$ can be derived from a collection of trivial equalities by repeatedly applying two fundamental rules:

1. **Union** If $A =^* B$ and $C =^* D$, then $\big(A \cup C\big) =^* \big(B \cup D\big)$.
2. **Augmentation:** If $A =^* B$ and $A \subset^* C$, then $\big(A \cup C\big) =^* B$.

If the equalities used in these steps already satisfy the decomposition property of the lemma, then the newly derived equality also satisfies it. Since all *trivial* equalities fulfill this property at the outset, an induction argument ensures that *every* equality produced in this manner will do so as well. □

Common propagated spaces have a corresponding fixed point.

Lemma G.3. *Let's consider a common propagated space $\mathcal{F}$ across the operators T_j. There exists a fixed point $P = (P_1, \ldots, P_J)$ of F which corresponding propagated space (lemma G.1) is $\mathcal{F}$ and such that $P_j \subset \mathcal{P}_j$.*

Proof. Let's consider a common propagated space $\mathcal{F}$ and a corresponding representation $p \in \bigcap_{j=1}^{N} \text{span}\, \mathcal{P}_j$.. Lets p_j the representation of p in each span $\mathcal{P}_j$. Thanks to the previous lemma G.2 we can write $p_j = p_j^1 \cup p_j^2$ for all j such that $p_1^1 = p_2^1 = \cdots = p_J^1$. Now let j, and take $i \in p_j$. There are 2 possibilities :

- $i \in p_j^1$ and therefore i appears in each family $(\mathcal{P}_1, \ldots, \mathcal{P}_J)$.
- $i \in p_j^2$ and therefore $\{i\} \subset^* p_j^1$. And with arguments similar as in the lemma G.2 (relying on the specialized nature of $\subset^*$) we can say that i covers $\mathcal{P}_j$.

In other words $i \triangleleft \{j, (\mathcal{P}_1, \ldots, \mathcal{P}_J)\}$. We write each p_j as an union of elements of $\mathcal{P}_j$ and define $\mathcal{P}'_j$ the set of these elements. From the previous result we can conclude that $P' = (\mathcal{P}'_1, \ldots, \mathcal{P}'_J)$ is a fixed point of F. This conclude the proof. □

Theorem G.4. *Let $\{P^n\}_{n=0}^{\infty}$ be the sequence defined in the algorithm, where each $P^n = (\mathcal{P}_1^n, \ldots, \mathcal{P}_J^n)$. This sequence converges to a fixed point, and the corresponding function space is the smallest common space propagated by all operators.*

Proof. We note $P^n = (\mathcal{P}_1^n, \ldots, \mathcal{P}_J^n)$. For each j $\{\mathcal{P}_j^n\}_{n=0}^{\infty}$ is a decreasing sequence of subsets of the finite set $\mathcal{P}(\mathcal{I})$. Each one of them is then stationary after a certain point, therefore $\{P^n\}_{n=0}^{\infty}$ is itself stationary after a certain point.

We can now let P^∞ be the limit of this sequence. By the previous lemma G.1 we can define p_∞ and $\mathcal{F}_\infty = \{B_i \mid i \in p_\infty\}$ to be the corresponding common propagated space.

Suppose there is another common propagated space $\mathcal{F}$. By Lemma G.3, there exists a fixed point $P = (P_1, \ldots, P_J)$ corresponding to $\mathcal{F}$. A routine induction shows that any set a removed from $\mathcal{P}_j^n$ at some step of the algorithm can not appear in P_j. Therefore $\bigcup_{p \in P_j} \subset p_j^\infty$ and $\mathcal{F}_\infty \subset \mathcal{F}$. Hence, $\mathcal{F}_\infty$ is the smallest among all common propagated spaces. □

G.3.6 Complexity

Applying F to each $\mathcal{P}_j$ requires $\mathcal{O}(NJ^2)$ time per iteration, where $N = \max_j \sum_{p \in \mathcal{P}_j} \#p$ denotes the maximum total number of spaces across all propagated sets for any operator. In practice, the sequence $\{P^n\}$ typically converges in only a few iterations, ensuring that the overall runtime remains tractable. Additionally, the memory complexity is $\mathcal{O}(NJ)$, representing a significant improvement over the naive approach, which is exponential in both time and space.

This gain is already meaningful for small-scale problems. For instance, in a state space of dimension n, the size of the base set $\mathcal{B}$, and hence the quantity N, typically scales as n^2. When $n = 4$, the base set already contains $\#\mathcal{B} = 53$ elements.

G.3.7 A Running Example

In this section, we illustrate the algorithm using a running example. Consider the following setting:

$$\begin{cases} T_1 \text{ propagates } F \cap A,\ E \cap A,\ C,\ D \cap F,\ H, \\ T_2 \text{ propagates } H \cap C,\ D \cap F,\ A \cap B, \\ T_3 \text{ propagates } F \cap G,\ C \cap B \cap I,\ A \cap D,\ H, \\ D \cap C \subset E, \\ F \subset B, \\ A \subset G. \end{cases}$$

Using the notation from the formal proof, we start with the initiation ($n = 0$):

$$\begin{cases} \mathcal{P}_1^0 = \{F \cap A,\ E \cap A,\ C,\ D \cap F,\ H\}, \\ \mathcal{P}_2^0 = \{H \cap C,\ D \cap F,\ A \cap B\}, \\ \mathcal{P}_3^0 = \{F \cap G,\ C \cap B \cap I,\ A \cap D,\ H\}, \end{cases}$$

$$\begin{cases} p_1^0 = \{A, C, D, E, F, H\}, \\ p_2^0 = \{A, B, C, D, F, H\}, \\ p_3^0 = \{A, B, C, D, F, G, H, I\}. \end{cases}$$

First Iteration ($n = 1$). We now evaluate which propagated sets are preserved across all operators.

For T_1:

- $F \cap A$: $F, A \in p_2^0 \cap p_3^0 \Rightarrow$ keep.
- C: $C \in p_2^0 \cap p_3^0 \Rightarrow$ keep.
- $E \cap A$: $C \in p_2^0 \cap p_3^0$. $D, C \in p_1^0$ and $D \cap C \subset E \Rightarrow$ keep.
- $D \cap F$: $D, F \in p_2^0 \cap p_3^0 \Rightarrow$ keep.

Thus,

$$\mathcal{P}_1^1 = \{F \cap A,\ E \cap A,\ C,\ D \cap F,\ H\}.$$

For T_2:

- $H \cap C$: $H, C \in p_1^0 \cap p_3^0 \Rightarrow$ keep.
- $D \cap F$: $D, F \in p_1^0 \cap p_3^0 \Rightarrow$ keep.
- $A \cap B$: $A \in p_1^0 \cap p_3^0$, $F \subset B$ and $F \in p_2^0 \Rightarrow$ keep.

Hence,

$$\mathcal{P}_2^1 = \{H \cap C,\ D \cap F,\ A \cap B\}.$$

For T_3:

- $F \cap G$: $F \in p_1^0 \cap p_2^0$, $A \subset G$ and $A \in p_3^0 \Rightarrow$ keep.
- $C \cap B \cap I$: $I \notin p_1^0 \cap p_2^0$ and not involved in any relation $\Rightarrow$ discard.
- $A \cap D$: $A, D \in p_1^0 \cap p_2^0 \Rightarrow$ keep.
- H: $H \in p_1^0 \cap p_2^0 \Rightarrow$ keep.

Therefore,

$$\mathcal{P}_3^1 = \{F \cap G,\ A \cap D,\ H\}.$$

We now have :

$$\begin{cases} p_1^1 = \{A, C, D, E, F, H\}, \\ p_2^1 = \{A, B, C, D, F, H\}, \\ p_3^1 = \{A, D, F, G, H\}. \end{cases}$$

Second Iteration ($n = 2$). We now report only the spaces that are discarded in this iteration.

For T_1:

- C: $C \notin p_3^1 \Rightarrow$ discard.

$$\mathcal{P}_1^2 = \{F \cap A,\ E \cap A,\ D \cap F,\ H\}.$$

For T_2:

- $H \cap C$: $C \notin p_3^1 \Rightarrow$ discard.

$$\mathcal{P}_2^2 = \{D \cap F,\ A \cap B\}.$$

For T_3: We keep all the spaces, therefore :

$$\mathcal{P}_3^2 = \{F \cap G,\ A \cap D,\ H\}.$$

Update:

$$\begin{cases} p_1^2 = \{A, D, E, F, H\}, \\ p_2^2 = \{A, B, D, F\}, \\ p_3^2 = \{A, D, F, G, H\}. \end{cases}$$

Third Iteration ($n = 3$). **For T_1:**

- $E \cap A$: Previously kept due to $D \cap C \subset E$, but now $C \notin p_1^2 \Rightarrow$ discard.
- H: $H \notin p_2^2 \Rightarrow$ discard.

$$\mathcal{P}_1^3 = \{F \cap A,\ D \cap F\}.$$

For T_2: We keep all the spaces, hence :

$$\mathcal{P}_2^3 = \{D \cap F,\ A \cap B\}.$$

For T_3:

- H: $H \notin p_2^2 \Rightarrow$ discard.

$$\mathcal{P}_3^3 = \{F \cap G,\ A \cap D\}.$$

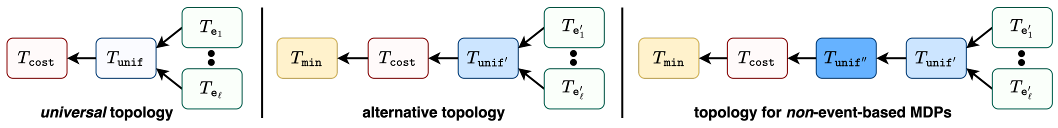
Update:

$$\begin{cases} p_1^3 = \{A, D, F\}, \\ p_2^3 = \{A, B, D, F\}, \\ p_3^3 = \{A, D, F, G\}. \end{cases}$$

Fourth Iteration ($n = 4$). At this stage, the propagated sets remain unchanged, indicating convergence. Thus, the largest common propagated space is:

$$A \cap B \cap D \cap F \cap G = A \cap D \cap F.$$

H Why Theorem 4.1 is Not Trivial? – Illustrative Examples



Universal topology for event-based MDPs, an alternative topology for an event-based MDP, and the topology for a non-event-based MDP.

In this section, we explain why the existence of a universal operator graph topology for event-based MDPs is not trivial. First, for an event-based MDP, it is possible to have operator graphs with alternative topologies. Second, for a non-event-based MDP, it may be impossible to construct an operator graph using the universal topology. We illustrate these two points through two examples. The summary of the results is illustrated in the figure above.

H.1 The M/M/1 Example With an Alternative Operator Graph Topology

For easy reference, we restate the Bellman equations for the M/M/1 example with controlled arrival and departure here:

$$V_{n+1}(x,\texttt{A}) = \min\left\{-r\cdot\frac{\lambda+\mu+\alpha}{\Lambda+\alpha} + V_n^*(x+1),\ c\cdot\frac{\lambda+\mu+\alpha}{\Lambda+\alpha} + V_n^*(x)\right\}, \tag{60}$$

$$V_{n+1}(x,\texttt{D}) = \min_{a_{\texttt{CD}}\in[0,1]}\left[g_r(a_{\texttt{CD}})\cdot\frac{\lambda+\mu+\alpha}{\Lambda+\alpha} + a_{\texttt{CD}}\cdot V_n^*(x) + (1-a_{\texttt{CD}})\cdot V_n^*(x-1)\right], \tag{61}$$

where $V_n^*(x)$ is the value function defined on the queuing state x:

$$V_n^*(x) \triangleq \frac{\rho_h(x)}{\Lambda+\alpha} + \gamma\left[\frac{\lambda}{\Lambda}\cdot V_n(x,\texttt{A}) + \frac{\mu}{\Lambda}\cdot V_n(x,\texttt{D}) + \left(1-\frac{\lambda+\mu}{\Lambda}\right)V_n(x,\varnothing)\right]. \tag{62}$$

Instead of the value function $V_n^*(x)$ on the queue length, we can decompose the Bellman equation on the standard value function $V_n(x,e)$ on the full state (x,e).

For example, the value function $V_{n+1}(x,\texttt{A})$ can be rewritten as

$$V_{n+1}(x,\texttt{A}) = T_{\texttt{min}}\Big\{T_{\texttt{cost},1}\left(T_{\texttt{unif}'}\left\{T_{\texttt{A}'}\left[V_n^*(x)\right], T_{\texttt{D}'}\left[V_n^*(x)\right], T_{\varnothing'}\left[V_n^*(x)\right]\right\}\right), \tag{63}$$

$$T_{\texttt{cost},0}\left(T_{\texttt{unif}'}\left\{T_{\texttt{A}'}\left[V_n^*(x)\right], T_{\texttt{D}'}\left[V_n^*(x)\right], T_{\varnothing'}\left[V_n^*(x)\right]\right\}\right)\Big\}, \tag{64}$$

with the modified event operators

$$T_{\texttt{A}'}\left[V_n^*(x)\right] = V_n(x,\texttt{A}),\quad T_{\texttt{D}'}\left[V_n^*(x)\right] = V_n(x,\texttt{D}),\quad T_{\varnothing'}\left[V_n^*(x)\right] = V_n(x,\varnothing), \tag{65}$$

a modified uniformization operator

$$T_{\texttt{unif}'}\left[U(x,\texttt{A}), U(x,\texttt{D}), U(x,\varnothing)\right] = \tag{66}$$

$$\frac{\rho_h(x)}{\Lambda+\alpha} + \gamma\left[\frac{\lambda}{\Lambda}\cdot U(x,\texttt{A}) + \frac{\mu}{\Lambda}\cdot U(x,\texttt{D}) + \left(1-\frac{\lambda+\mu}{\Lambda}\right)U(x,\varnothing)\right], \tag{67}$$

an action-dependent cost operator

$$T_{\texttt{cost},a}\left\{U[(x,e),a]\right\} = c[(x,e),a] + U[(x,e),a], \tag{68}$$

and a minimization operator

$$T_{\texttt{min}}\left\{U[(x,e),a]\right\} = \min_{a\in\mathcal{A}_{(x,e)}} U[(x,e),a]. \tag{69}$$

H.2 An Example of Non-Event-Based MDP With a Different Operator Graph Topology

An example of a non-event-based MDP is the M/M/1 queue with controlled arrival and service rate optimization.

Patients arrive according to a Poisson process with rate λ and wait to be served. The state $s = (x, e)$ has two components: the number of patients in the queue $x \in \mathbb{N}_+$ and the event $e \in \{\mathtt{A}, \mathtt{D}\}$, where $\mathtt{A}$ denotes arrival and $\mathtt{D}$ denotes departure. The action $a = (a_{\mathtt{CA}}, a_{\mathtt{RO}})$ consists of admission control ($\mathtt{CA}$) $a_{\mathtt{CA}} \in \{0, 1\}$ when a new patient arrives, where 0 denotes rejection and 1 denotes acceptance, and service rate optimization ($\mathtt{RO}$) $a_{\mathtt{RO}} \in \{\mu_1, \ldots, \mu_K\}$. The actions space is

$$\mathcal{A}_s = \left\{ \begin{array}{ll} \{0,1\} \times \{\mu_1, \ldots, \mu_K\} & s \in \{(x,e) \,|\, e = \mathtt{A}\} \\ \varnothing \quad \times \{\mu_1, \ldots, \mu_K\} & s \in \{(x,e) \,|\, e = \mathtt{D}\} \end{array} \right. . \tag{70}$$

When the system is not empty ($x > 0$), the state transition time is exponentially distributed with rate $\lambda(s, a) = \lambda + a_{\mathtt{RO}}$, and the state transition probability is

$$\hat{P}(s'|s,a) = \begin{cases} \mathbf{1}_{\{x'=x+a_{\mathtt{CA}}\}} \cdot \frac{\lambda}{\lambda + a_{\mathtt{RO}}}, & e = \mathtt{A}, e' = \mathtt{A} \\ \mathbf{1}_{\{x'=x+a_{\mathtt{CA}}-1\}} \cdot \frac{a_{\mathtt{RO}}}{\lambda + a_{\mathtt{RO}}}, & e = \mathtt{A}, e' = \mathtt{D} \\ \mathbf{1}_{\{x'=x\}} \cdot \frac{\lambda}{\lambda + a_{\mathtt{RO}}}, & e = \mathtt{D}, e' = \mathtt{A} \\ \mathbf{1}_{\{x'=x-1\}} \cdot \frac{a_{\mathtt{RO}}}{\lambda + a_{\mathtt{RO}}}, & e = \mathtt{D}, e' = \mathtt{D} \end{cases} . \tag{71}$$

The cost includes a one-time reward r for accepting a patient or a one-time penalty p of rejecting a patient, a holding cost per unit time $\rho_{\mathrm{h}}(x)$ that depends on the number of patients in the system, and a service cost per unit time $\rho_{\mathrm{s}}(a_{\mathtt{RO}})$ that depends on the service rate. So the cost can be calculated as

$$\hat{c}(s,a) = \begin{cases} -r + \frac{\rho_{\mathrm{h}}(x+1) + \rho_{\mathrm{s}}(a_{\mathtt{RO}})}{\lambda + a_{\mathtt{RO}} + \alpha}, & e = \mathtt{A}, a_{\mathtt{CA}} = 1 \\ c + \frac{\rho_{\mathrm{h}}(x) + \rho_{\mathrm{s}}(a_{\mathtt{RO}})}{\lambda + a_{\mathtt{RO}} + \alpha}, & e = \mathtt{A}, a_{\mathtt{CA}} = 0 \\ \frac{\rho_{\mathrm{h}}(x) + \rho_{\mathrm{s}}(a_{\mathtt{RO}})}{\lambda + a_{\mathtt{RO}} + \alpha}, & e = \mathtt{D} \end{cases} . \tag{72}$$

We can see from equation 71 that while the state transition probability can be decomposed, the probability of the event $P_{\mathrm{e}}(e \mid x, a)$ actually depends on the action. This is because the service rate affects the probability of the next event being an arrival or a departure.

In this case, we cannot use the universal topology for non-event-based MDPs.

I Detailed Descriptions of Examples in Section 5

In this sections we give more details on the examples discussed in section 5.

Example 3 (The MCTS succeeds to generate formulation with high structural expressiveness in complex problems)**.** Let us consider the following problem :

We aim to minimize the long-run average cost of operating our hospital. The hospital has 3 wards arranged sequentially, sharing a total capacity of 20 beds. Each ward has its own healthcare team and manages its own patients. On average, 7 patients arrive at the first ward per hour. The wards serve patients at average rates of 10, 5, and 2 patients/hour, respectively. Patients progress sequentially: from the first ward to the second, and then from the second to the third. After treatment in the third ward, patients leave the hospital. Incoming patients can be rejected, incurring a cost of 20. Additionally, patients can be directly transferred from one ward to the next before being served at an average rate of 3 patients/hour for each ward, with each transfer costing 5.

Key challenges: There are various types of events (controlled arrivals, departures, transfers) and implicit constraints (nonnegativity) on the state space.

The first level of the tree consists of defining the parameters of the problem, while the second level identifies the state space. This gives:

$$\begin{cases} n_{\text{beds}} = 20, \\ r_{\text{arrivals}} = 7, \\ r_{\text{service}} = (10, 5, 2), \\ r_{\text{transfer opportunity}} = 3, \\ c_{\text{refusal}} = 20, \\ c_{\text{transfer}} = 5, \end{cases} \qquad \begin{cases} x_1 + x_2 + x_3 \leq 20, \\ x_1, x_2, x_3 \geq 0. \end{cases}$$

Here, $x = (x_1, x_2, x_3)$ represents the number of patients in the respective wards. Notably the positive constraints were implicit.

The next step defines the events, their probabilities, and the available actions with their corresponding costs and effects on the state. Let $\Gamma = r_{\text{arrivals}} + r_{\text{service},1} + r_{\text{service},2} + r_{\text{service},3} + 3r_{\text{transfer opportunity}}$ and $(\varepsilon_i)_{1 \leq i \leq 3}$ the canonical base of $\mathbb{R}$. The events are as follows:

- **Patient arrival :**
$$\begin{cases} p_{\text{arrival}} = r_{\text{arrivals}}/\Gamma, \\ \mathcal{A}_{\text{arrival}} = \{a_{\text{accept}}, a_{\text{refuse}}\}, \\ P_{\text{arrival}}(x' \mid x, a_{\text{accept}}) = \mathbb{1}(x' = x + \varepsilon_1), \\ P_{\text{arrival}}(x' \mid x, a_{\text{refuse}}) = \mathbb{1}(x' = x), \\ c_{\text{arrival}}(x', x, a) = c_{\text{refusal}} \mathbb{1}(a = a_{\text{refuse}}). \end{cases}$$

- **Patient served ward 1 :**
$$\begin{cases} p_{\text{service},1} = r_{\text{service},1}/\Gamma, \\ \mathcal{A}_{\text{service},1} = \varnothing, \\ P_{\text{service},1}(x'|x) = \begin{cases} \mathbb{1}(x' = x - \varepsilon_1 + \varepsilon_2), & \text{if } x_1 > 0, \\ \mathbb{1}(x' = x), & \text{otherwise}, \end{cases} \\ c_{\text{service},1}(x', x) = 0. \end{cases}$$

- **Patient transferred ward 1 to ward 2:**
$$\begin{cases} p_{\text{transfer},1} = r_{\text{transfer opportunity}}/\Gamma, \\ \mathcal{A}_{\text{transfer},1} = \{a_{\text{transfer}}, a_{\text{keep}}\}, \\ P_{\text{transfer},1}(x' \mid x, a_{\text{transfer}}) = \mathbb{1}(x' = x - e_1 + e_2) \\ P_{\text{transfer},1}(x' \mid x, a_{\text{keep}}) = \mathbb{1}(x' = x) \\ c_{\text{transfer},1}(x', x, a) = c_{\text{transfer}} \mathbb{1}(a = a_{\text{transfer}}). \end{cases}$$

We do not detail the system dynamics for the remaining events: ”Patient served in ward 2”, ”Patient served in ward 3”, and ”Patient transferred from ward 2 to ward 3,” as they are similar to the examples illustrated above.

The next step is to identify the operational cost, which in this problem is equal to 0 (Maybe I should add one).

Using the dynamics of the system for each event, we can already derive the corresponding operators. In the last layer of the tree the LLM identifies the operators for which propagation results apply. For this problem, the identified operators are:

- **Patient arrival :**
$$T_{\text{CA},1}f(x) = \min\{c_{\text{refusal}} + f(x); f(x+e_1)\}$$
- **Patient served ward 1 :**
$$T_{\text{TD1},1}f(x) = \begin{cases} f(x-e_1+e_2), & \text{if } x_1 > 0, \\ f(x), & \text{otherwise.} \end{cases}$$
- **Patient transferred ward 1 to ward 2 :**
$$T_{\text{CTD},(1,2)}f(x) = \min\{c_{\text{transfer}} + f(x-e_1+e_2), f(x)\}$$
- **Patient served ward 3 :**
$$T_{\text{D1},3}f(x) = f((x-e_3)^+)$$

Similar operators can be derived for ”Patient served in ward 2” and ”Patient transferred from ward 2 to ward 3.”

For each of these operators we can automatically list the functional spaces they propagate, for instance $T_{\text{CA},1}$ propagate all the following spaces (see Appendix for the details of each one of them) :

$$\begin{aligned} &I,\ \text{UI},\ Cx(1),\ \text{Super}(1,2),\ \text{Super}(1,3),\ \text{Sub}, \\ &\text{Super}(1,2) \cap \text{SuperC}(1,2),\ \text{Super}(1,3) \cap \text{SuperC}(1,3), \\ &\text{Super}(1,3) \cap \text{SuperC}(3,1),\ \text{Super}(1,2) \cap \text{SuperC}(2,1), \\ &\text{Sub}(1,2) \cap \text{SubC}(1,2)\ \text{Sub}(1,3) \cap \text{SubC}(1,3), \\ &\text{Sub}(1,2) \cap \text{SubC}(2,1),\ \text{Sub}(1,3) \cap \text{SubC}(3,1),\ \text{MM} \end{aligned}$$

We also do this for T_{unif} and T_{cost}. Also, depending on the shape of the state space certain spaces must be dropped. Finally, we can run our second algorithm introduced in the subsection 4.3. For this problem we end up with the following propagated space :

$$I \cap \text{UI } \cap \text{MM}$$

From which we can extract automatically the following structural results :

1. **Controlled arrival in ward 1 :** let $\pi^*_{\text{CA}(1)} : \mathcal{S} \to \{0,1\}$ be the optimal acceptance policy in the first ward such that 0 is refusal and 1 acceptance.
   - $\pi^*_{\text{CA}(1)}$ is decreasing in the number of patients in the hospital.
   - $\pi^*_{\text{CA}(1)}$ is decreasing in the directions $(1,-1,0)$ and $(1,0,-1)$.
2. **Controlled departure from ward 1 to ward 2 :**, let $\pi^*_{\text{CTD}(1,2)} : \mathcal{S} \to \{0,1\}$ be the optimal departure policy such that 0 correspond to keeping the patient in the ward 1 and 1 is moving them to ward 2.
   - $\pi^*_{\text{CTD}(1,2)}$ is decreasing in the number of patient in the ward 2, ie in the direction $(0,1,0)$.
   - $\pi^*_{\text{CTD}(1,2)}$ is increasing in the number of patient in the ward 1, ie in the direction $(1,0,0)$.
3. **Controlled departure from ward 2 to ward 3 :** let $\pi^*_{\text{CTD}(2,3)} : \mathcal{S} \to \{0,1\}$ be the optimal departure policy such that 0 correspond to keeping the patient in the ward 2 and 1 is moving them to ward 3.

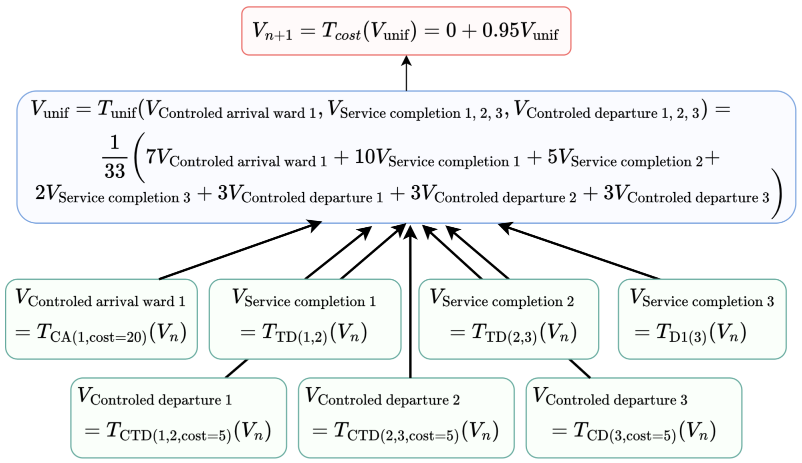

$V_{n+1} = T_{cost}(V_{\text{unif}}) = 0 + 0.95V_{\text{unif}}$
$V_{\text{unif}} = T_{\text{unif}}(V_{\text{Controled arrival ward 1}}, V_{\text{Service completion 1, 2, 3}}, V_{\text{Controled departure 1, 2, 3}}) =$
$\frac{1}{33}\Big(7V_{\text{Controled arrival ward 1}} + 10V_{\text{Service completion 1}} + 5V_{\text{Service completion 2}} +$
$2V_{\text{Service completion 3}} + 3V_{\text{Controled departure 1}} + 3V_{\text{Controled departure 2}} + 3V_{\text{Controled departure 3}}\Big)$
$V_{\text{Controled arrival ward 1}} = T_{\text{CA(1,cost=20)}}(V_n)$
$V_{\text{Service completion 1}} = T_{\text{TD(1,2)}}(V_n)$
$V_{\text{Service completion 2}} = T_{\text{TD(2,3)}}(V_n)$
$V_{\text{Service completion 3}} = T_{\text{D1(3)}}(V_n)$
$V_{\text{Controled departure 1}} = T_{\text{CTD(1,2,cost=5)}}(V_n)$
$V_{\text{Controled departure 2}} = T_{\text{CTD(2,3,cost=5)}}(V_n)$
$V_{\text{Controled departure 3}} = T_{\text{CD(3,cost=5)}}(V_n)$

Figure 5: Operator graph of Example 3

- $\pi^*_{\text{CTD}(2,3)}$ is decreasing in the number of patient in the ward 3, ie in the direction $(0, 0, 1)$.
- $\pi^*_{\text{CTD}(2,3)}$ is increasing in the number of patient in the ward 2, ie in the direction $(0, 1, 0)$.

In other words, the optimal policy of the problem is threshold along many different directions. These structural results and the optimal policy obtained by running a solver on the formulation are then communicated back to the user.

Key takeaways: Autoformulating an event-based MDP involves multiple steps, and our proposed algorithm effectively navigates these challenges in complex problems. Most of the time the resulting formulation has high structural expressiveness

Example 4. ▶ Two correct graphs of operators with different structural complexity

Let us consider the following problem :

*We aim to minimize the long-run average cost of operating our hospital. The hospital has 1 ward that manages 2 types of patients with **shared** healthcare teams. There are N_b beds in total. The average arrival rates of the patients are λ_1/hour and λ_2/hour respectively. A team take care of a patient with an average rate that depends on their type : μ_1/hour and μ_2/hour respectively. When a patient arrive we can refuse it, it occurs a cost of c_1 for the first type of patients and c_2 of the others.*

Key challenges: We cannot obtain structural results from the straightforward problem formulation. How to find an equivalent combination of operators that allow us to obtain structural results?

Straightforward problem formulation. The natural events of this problem are *Arrival of a patient of type 1*, *Arrival of a patient of type 2*, *Departure of a patient of type 1* and *Departure of a patient of type 2*.

This approach lead to the following operator graph :

$$V^*_{n+1} = T_{\text{cost}}\left\{(T_{\text{unif}}\left[T_{\text{CA}(1)}(V^*_n), T_{\text{CA}(2)}(V^*_n), T_{\text{D1}(1)}(V^*_n), T_{\text{D1}(2)}(V^*_n), V^*_n\right]\right\}$$

With probabilities in T_{unif} that depends on the state, you get for instance :

$$p_{\text{D}(1)} = \frac{\mu_1 n_1}{\lambda_1 + \lambda_2 + \mu N_b} \quad \text{with } \mu = \max(\mu_1, \mu_2).$$

Koole's results don't extend to probabilities that depend on the state in T_{unif}. We can't get any structural result from this formulation, even if it is a right one.

However, this formulation is actually equivalent to the following one:

$$V^*_{n+1} = T_{\text{cost}}\left\{T_{\text{unif}}\left[T_{\text{CA}(1)}(V^*_n), T_{\text{CA}(2)}(V^*_n), T_{\text{D}(1)}(V^*_n), T_{\text{D}(2)}(V^*_n)\right]\right\}.$$

This time the probabilities in T_{unif} are (with $\Gamma = \lambda_1 + \lambda_2 + \mu N_b$) :

1. $p_{\text{CA}(i)} = \lambda_i/\Gamma$
2. $p_{\text{D}(1)} = p_{\text{D}(2)} = \frac{1}{2}\big(1 - \frac{\lambda_1+\lambda_2}{\Gamma}\big)$

which don't depend on the state. The dependence in the state has been absorbed by the new T_D operators for which we have structural results :

$$T_{\text{D}(1)}f(x) = \frac{2\mu_1 n_1}{\Gamma - (\lambda_1+\lambda_2)} f((x-e_1)^+) + \left(1 - \frac{2\mu_1 n_1}{\Gamma - (\lambda_1+\lambda_2)}\right) f(x)$$

Indeed, with this formulation we can show that the following space is propagated through the Bellman equation :

$$I \cap \text{Cx } \cap \text{Super}$$

And we can deduce structural results from there.

Key Takeaways : Problem formulation and structural analysis are inherently connected, as certain valid formulations may not permit structural analysis.

Example 5. ► We don't have any formulation that reveals the structural results.

Let us consider the following problem :

We aim to minimize the long-run average cost of operating our hospital. The hospital has 3 wards arranged sequentially, with capacities of 5, 15, and 15 beds, respectively. Each ward has its own healthcare team and manages its own patients. On average, new patients arrive at rates of 3, 20, and 5 patients/day in the respective wards. The wards serve patients at rates of 10, 5, and 3 patients/day, respectively. After being served in the first or second ward, we can transfer to the next ward at a cost of 2 per transfer or keep them in the current ward. Patients served in the third ward leave the hospital. Additionally, ***patients can be moved back*** *from ward 2 to ward 1 at a rate of 3 patients/day or from ward 3 to ward 2 at a rate of 1 patient/day, each transfer incurring a cost of 2. Incoming patients can also be refused, incurring costs of 5, 10, and 15 for wards 1, 2, and 3, respectively.*

Key challenges: The solution to the problem exhibits structural properties, but these cannot be anticipated regardless of the choice of operator graph.

The autoformulation part of the algorithm managed to find a correct operator graph :

$$\begin{aligned} V^*_{n+1} = T_{\text{cost}}\big(T_{\text{unif}}\big(&T_{\text{CA},1}(V^*_n), T_{\text{CA},2}(V^*_n), T_{\text{CA},3}(V^*_n),\\ &T_{\text{CTD},(1,2)}(V^*_n), T_{\text{CTD},(2,3)}(V^*_n), T_{\text{CTD},(2,1)}(V^*_n),\\ &T_{\text{CTD},(3,2)}(V^*_n), T_{\text{D1},3}(V^*_n)\big)\big) \end{aligned}$$

The structural results observed experimentally (see Figure 6) cannot be predicted from the operator graph. Unlike Example 2, this issue cannot be resolved with a better formulation, as the structural results have not yet been theoretically established.

Key takeaways: The current method for identifying structural results fails on certain problems, as it depends on limited theoretical results.

Example 6 (Two wards with controlled jockeying)**.** Natural language description :

We aim to minimize the long-run average cost of operating our hospital. The hospital has two wards running in parallel, each managing its own patients with a dedicated healthcare team. The first ward can hold up to 5 patients, while the second can accommodate up to 10. Patients complete their treatment in the second ward before leaving the hospital. Each patient in the hospital cost 2/hour to the hospital. On average, 3 patients arrive at the first ward per hour, and 5 arrive at the second. New patients can be rejected, incurring a cost of 5 for the first ward and 10 for the second. The first ward operates at a frequency of 10 patients/hour, while the second operates at 5 patients/hour. Patients treated in the first ward can either remain there at no cost (but will require further care by the same team before leaving the ward) or be transferred to the second ward at a cost of 2. Additionally, patients can be transferred back from the second ward to the first at a frequency of 3 patients/hour, with each transfer costing 2. Refusing a transfer incurs no cost.

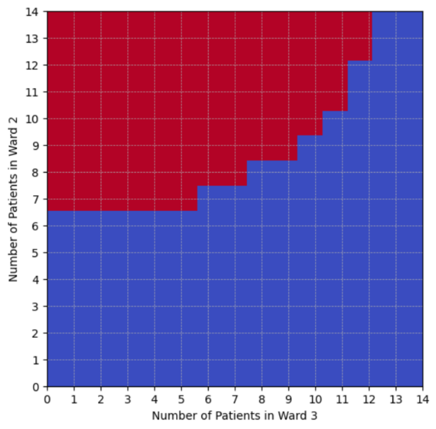


Figure 6: Optimal policy for a controlled jockeying problem across three wards with controlled arrival and unControlled departures in the last ward. One of the event is the opportunity to move a patient from ward 2 to ward 3, the possible actions are : *move* (in red) or *keep* (in blue). The graph shows the switching curve the optimal policy depending on the number of patients in the wards. The optimal policy is structured but we can not anticipate it with the results from Koole (2007) and therefore it's beyond the capacities of our algorithm.

The operator graph of the problem is illustrated 5. The Bellman equation propagate the following function space :

$$I \cap \text{Super } \cap \text{SuperC}$$

Let $x = (n_1, n_2)$ be the number of patients in the two wards. We have the following structural results for the optimal policy:

1. **Controlled arrival in ward 1 :** let $\pi^*_{\text{CA}(1)} : \mathcal{S} \to \{0, 1\}$ be the optimal acceptance policy in the first ward such that 0 is refusal and 1 acceptance.
   - $\pi^*_{\text{CA}(1)}$ is decreasing in the number of patients in the hospital.
   - $\pi^*_{\text{CA}(1)}$ is decreasing in the direction $(1, -1)$.
2. **Controlled arrival in ward 2 :** let $\pi^*_{\text{CA}(2)} : \mathcal{S} \to \{0, 1\}$ be the optimal acceptance policy in the first ward such that 0 is refusal and 1 acceptance.
   - $\pi^*_{\text{CA}(2)}$ is decreasing in the number of patients in the hospital.
   - $\pi^*_{\text{CA}(2)}$ is decreasing in the direction $(-1, 1)$.
3. **Controlled departure from ward 1 to ward 2 :**, let $\pi^*_{\text{CTD}(1,2)} : \mathcal{S} \to \{0, 1\}$ be the optimal departure policy such that 0 correspond to keeping the patient in the ward 1 and 1 is moving them to ward 2.
   - $\pi^*_{\text{CTD}(1,2)}$ is decreasing in the number of patient in the ward 2, ie in the direction $(0, 1)$.
   - $\pi^*_{\text{CTD}(1,2)}$ is increasing in the number of patient in the ward 1, ie in the direction $(1, 0)$.
4. **Controlled departure from ward 2 to ward 1 :**, let $\pi^*_{\text{CTD}(2,1)} : \mathcal{S} \to \{0, 1\}$ be the optimal departure policy such that 0 correspond to keeping the patient in the ward 2 and 1 is moving them to ward 1.
   - $\pi^*_{\text{CTD}(2,1)}$ is decreasing in the number of patient in the ward 1, ie in the direction $(1, 0)$.
   - $\pi^*_{\text{CTD}(2,1)}$ is increasing in the number of patient in the ward 2, ie in the direction $(0, 1)$.

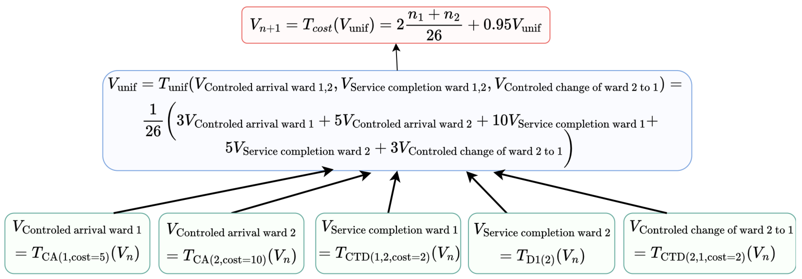

$V_{n+1} = T_{cost}(V_{\text{unif}}) = 2\frac{n_1+n_2}{26} + 0.95V_{\text{unif}}$
$V_{\text{unif}} = T_{\text{unif}}(V_{\text{Controled arrival ward 1,2}}, V_{\text{Service completion ward 1,2}}, V_{\text{Controled change of ward 2 to 1}}) = \frac{1}{26}\Big(3V_{\text{Controled arrival ward 1}} + 5V_{\text{Controled arrival ward 2}} + 10V_{\text{Service completion ward 1}} + 5V_{\text{Service completion ward 2}} + 3V_{\text{Controled change of ward 2 to 1}}\Big)$
$V_{\text{Controled arrival ward 1}} = T_{\text{CA(1,cost=5)}}(V_n)$
$V_{\text{Controled arrival ward 2}} = T_{\text{CA(2,cost=10)}}(V_n)$
$V_{\text{Service completion ward 1}} = T_{\text{CTD(1,2,cost=2)}}(V_n)$
$V_{\text{Service completion ward 2}} = T_{\text{D1(2)}}(V_n)$
$V_{\text{Controled change of ward 2 to 1}} = T_{\text{CTD(2,1,cost=2)}}(V_n)$

Figure 7: Operator graph of Example 5

J Details on the method

J.1 LLM enhanced MCTS

The Monte Carlo Tree Search (MCTS) formulates the problem in four steps, which correspond to four levels of the tree. A node in the first layer, denoted m_1, represents the parameters of the problem, such as the number and size of queues. A second-layer node m_2 represents the state variables and the constraints defining the state space. A third-layer node m_3 represents the possible events, their probabilities, the corresponding actions and operational costs. Finally, a fourth-layer node m_4 represents the operators associated with each event.

For a given problem description, each m_i is identified through the standard MCTS steps: *selection*, *expansion*, *evaluation*, and *backpropagation*, omitting the *simulation* step in this context. Details are given bellow. For a give node m_i we denote $\text{Child}(m_i)$ the children of the node in the tree.

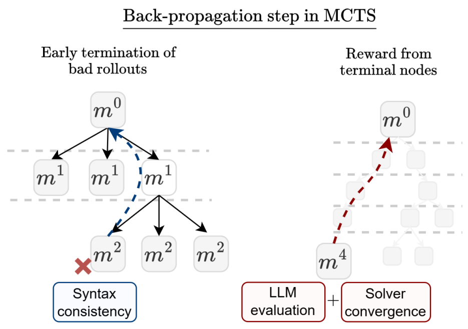

Back-propagation step in MCTS
Early termination of bad rollouts
Reward from terminal nodes
m^0
m^1
m^2
m^4
Syntax consistency
LLM evaluation
+
Solver convergence

Figure 8: **MCTS for constructing operator graph of event-based MDPs.** (1) Solver feedback complements LLM self-evaluation for more objective rewards. (2) Syntax checks at intermediate nodes detect errors early, preventing failed full rollouts.

J.1.1 Selection

The selection step guides the search towards promising regions of the tree. Starting from the root, the algorithm recursively selects child nodes using the Upper Confidence Bound for Trees (UCT): $m^*_{i+1} = \arg\max_{m_{i+1} \in \text{Child}(m_i)} \left(V(m_{i+1}) + \omega \sqrt{\frac{\ln N(m_i)}{N(m_{i+1})}} \right)$ Kocsis & Szepesvári (2006). This process continues until reaching an unexpanded node. Here, m^*_{i+1} is the selected child node, $V(m_{i+1})$ is its estimated value, $N(m_i)$ and $N(m_{i+1})$ are visit counts for the parent and child nodes respectively, and ω is an exploration constant. This formula balances exploitation (first term, favoring high-value nodes) with exploration (second term, favoring less-visited nodes).

J.1.2 Expansion

Upon reaching an unexpanded node m_i of depth i, we generate its child nodes through an expansion process. Unlike traditional MCTS, which operates within a predefined search space, our approach explores an open-ended hypothesis space of component formulations. To facilitate this expansion, we employ LLMs as adaptive hypothesis generators. These models, conditioned on the partial formulation constructed up to node m_i, propose potential formulations for the next component in the search process.

At each node m_i, the LLM generates potential child nodes m_i corresponding to next-step component formulations. This process follows the probability distribution:

$$p_\phi(m_{i+1}|m_{\leq i}, d) \tag{73}$$

where d represents the problem description and $m_{\leq i}$ represents the partial formulation constructed up to that depth.

The LLM is queried using a structured prompt consisting of three components: (1) the original problem description d, provided in natural language; (2) the partial formulation $m_{\leq i}$, represented in JSON format; and (3) level-specific instructions that define the expected output format and relevant constraints. Additionally, we instruct the LLM to return candidate formulations using the same structured dictionary format to ensure consistency across iterations.

For each node expansion, we sample H candidate formulations from the LLM's output distribution:

$$\text{Child}(m_i) = \{m_{i+1}^h \mid m_{i+1}^h \sim p_\phi(\cdot|m_{\leq i}, d), \forall h \in [H]\} \tag{74}$$

where m_{i+1}^h represents the h-th candidate formulation.

When generating new candidate formulations, we systematically verify their syntax consistency with the existing partial formulation by evaluating the mathematical expressions as the constraints or the probabilities of the events. This step allows us to immediately discard invalid options, ensuring coherence throughout the expansion process. If a candidate fails to meet syntax consistency requirements, we re-query the LLM for a revised formulation.

If the maximum number of retries is reached, we assume that the inconsistency is not solely due to the stochastic nature of the LLM but rather stems from an issue in the existing partial formulation—such as a missing variable definition in earlier steps. In such cases, we terminate the rollout and immediately backpropagate a score of 0 along the current branch.

J.1.3 Evaluation

After expanding a node, each newly created child node undergoes an initial evaluation to estimate its value, guiding subsequent selection in the search process. Assessing the correctness of a partial formulation relative to the original problem description is non-trivial, to address this challenge, we employ an LLM-based ranking evaluation for each set of child nodes, providing a more informed initial assessment.

Specifically we give the LLM the partial formulation till the current node $m_{\leq i}$ and let it rank the child nodes Child(m_i). The resulting ranks are then center-normalized to the interval $[0, 1]$, with the middle rank positioned at 0.5. We define the normalized score as $s(m_{i+1}^h)$, which is used to initialize the value of each child node:

$$V_{\text{prior}}(m_{i+1}^h) \leftarrow s(m_{i+1}^h). \tag{75}$$

This approach deviates from traditional Monte Carlo Tree Search (MCTS), which typically assigns uniform priors to newly expanded nodes. Instead, the LLM evaluates the formulations by incorporating optimization principles and problem-specific context, potentially capturing aspects such as formulation correctness, constraint feasibility, and alignment with the overall problem structure.

J.1.4 Backpropagation

Unlike conventional Monte Carlo Tree Search (MCTS), which typically simulates the problem to a terminal state after expanding a child node, our approach continues expansion until a terminal node m_t is reached. The resulting formalization $m_{\leq t}$ is evaluated against a baseline, typically the formalization obtained after one rollout. The LLM assigns a score between 0 and 1 based on its preference for the new formalization over the baseline. To mitigate bias in the LLM signal s_{LLM}, we also check whether the solver successfully converges on the formalization, setting $s_{\text{converged}} = 1$ if it converges and 0 otherwise. The final backpropagated score is then given by $s_{\text{LLM}} \times s_{\text{converged}}$.

The backpropagation process consists of updating the value of each node m_i along the current branch using the following update rule:

$$V_{\text{back}}(m_i) \leftarrow \frac{V_{\text{back}}(m_i) \cdot N(m_i) + s_{\text{LLM}} \times s_{\text{converged}}}{N(m_i) + 1} \tag{76}$$

where $N(m_i)$ denotes the number of times the value of m_i has been updated. After applying this update, we increment the count:

$$N(m_i) \leftarrow N(m_i) + 1. \tag{77}$$

K Desiderata of Autoformulation

K.1 Desiderata and Corresponding Challenges

The overarching objective of autoformulation is to autonomously solve problems expressed in natural language. This objective can be decomposed into three essential desiderata: (see Fig. 1 for how our framework fulfill them)

Accuracy. The ability to translate the natural language description into a suitable formal framework while preserving semantic accuracy. Autoformulation should correctly *formulate the problem.* (in the context of this paper output a MDP formulation that correctly reflects the problem description in natural language).

Computational Tractability: The resulting formalization must support efficient computation of a solution. For instance, autoformulation should *identify structures of the optimal policy* (e.g., the action is monotone in the state). The structures should be identified based on the formulation only, *before* the problem is solved. This facilitates in selecting low-complexity algorithms tailored for finding policies with certain structures (e.g., solvers for threshold policies).

Interpretability: Autoformulation should also be *interpretable* in two aspects. ▶ **Interpretability of formulation**: We should be able to trace each components of problem formulation back to the natural language problem description. ▶ **Interpretability of policies**: By identifying structural properties of the optimal policy, the autoformulator can explain the policy, making it easier for *non-technical* domain experts to understand and adopt the policy.

These desiderata come with corresponding challenges that must be addressed. We discuss them below, and use them in Section 5 to evaluate the performance of our proposed algorithm against these criteria.

Challenges in Accuracy
Achieving correct formalization is a non-trivial task, as it amounts to searching within a vast space of possible formulations. The main challenges include:

- ***Semantic Understanding:*** The system must correctly capture the underlying dynamics of the problem described in natural language. *For example, understanding that admitting a new patient to a hospital reduces the number of available beds*
- ***Parameter Identification:*** Relevant variables, constraints, and objective components must be identified and instantiated correctly. *For instance, the system should infer the arrival frequency of different types of patients to the hospital.*

- ***Syntactic Validity:*** The generated formulation must conform to the syntactic requirements of the chosen formal framework while preserving the original problem's intent. *For example, state updates should be expressed using syntactically valid expressions, such as correct Python formulas.*

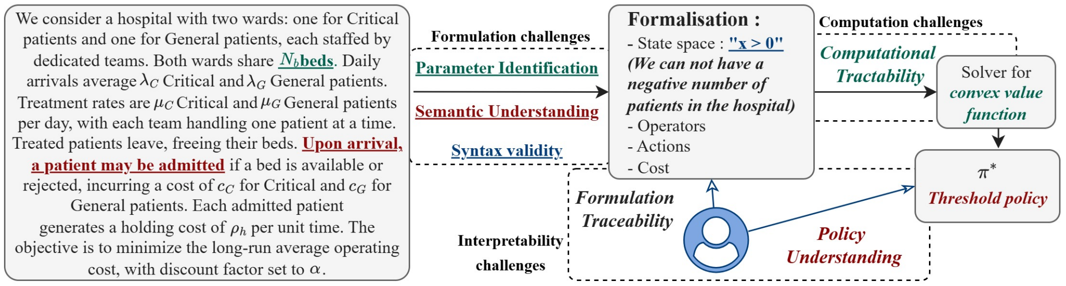
Challenges in Computational Tractability
Fulfilling the second desideratum goes beyond achieving a correct formalization: the chosen representation must also support efficient solving. In this work, we particularly emphasize the extraction of structural properties, which gives rise to two key challenges:

- ***Expressiveness of the Formulation:*** The formal representation must be sufficiently expressive to enable extraction of meaningful structural results.
- ***Structural Inference:*** Given a formalization, the system should be able to automatically identify structural properties that can guide or accelerate the solution process.

These two challenges are interconnected: the expressiveness of a formulation determines which structures can be extracted, while the usefulness of the formulation itself depends on the system's capacity to exploit these structures.

Challenges in Interpretability
For safety, usability, and insight, both the formulation and the solution should be interpretable. This is important for expert auditing and for practical deployment by non-expert users. The main challenges are:

- ***Formulation Traceability:*** Each element of the formalization should be traceable to a corresponding concept or statement in the original natural language problem description.
- ***Policy Understanding:*** The optimal policy for high-dimensional problems often behaves as a black box. Making its properties explicit enhances human understanding and trust.


We consider a hospital with two wards: one for Critical patients and one for General patients, each staffed by dedicated teams. Both wards share N_b**beds**. Daily arrivals average λ_C Critical and λ_G General patients. Treatment rates are μ_C Critical and μ_G General patients per day, with each team handling one patient at a time. Treated patients leave, freeing their beds. **Upon arrival, a patient may be admitted** if a bed is available or rejected, incurring a cost of c_C for Critical and c_G for General patients. Each admitted patient generates a holding cost of ρ_h per unit time. The objective is to minimize the long-run average operating cost, with discount factor set to α.

Formalisation :
- State space : **"x > 0"**
(We can not have a negative number of patients in the hospital)
- Operators
- Actions
- Cost

Figure 10: The challenges of autoformulation illustrated with an hospital example.

K.2 Typical Errors with Respect to These Challenges

In Section 5, we evaluate the extent to which our algorithm addresses the aforementioned challenges. Further details are provided below.

K.2.1 Errors in Accuracy.

While our method largely resolves the *Syntactic Validity* challenge and exhibits strong performance in *Parameter Identification*, our primary focus is on *Semantic Understanding*, where most errors tend to arise. Semantic errors can occurs in several ways :

- *Missing Constraints:* The algorithm may overlook implicit constraints in the problem description, such as the non-negativity of the number of patients in a hospital.
- *Incorrect Event Modeling:* It may introduce artificial events that do not exist in the actual problem. A common example is inventing an event to account for a holding cost, modeled as an event with frequency 1 per time unit in which the state remains unchanged but a cost is incurred.

- *Failure in Uniformization:* The algorithm may miscompute event probabilities when uniformizing the process. For instance, it sometimes fails to distinguish between sequential and parallel service models. If f denotes the service frequency per server:
  - In the single-server case (sequential), the probability of service is f/Γ.
  - In the multi-server case (parallel), the correct probability is xf/Γ', where x is the number of customers.

K.2.2 Errors in Computational Tractability and Interpretability.

Our algorithm exhibits limitations in both *Structural Expressiveness* and *Structural Inference*.

- *Limited Structural Expressiveness:* In some cases, the generated formalization lacks the expressive power needed to enable structural inference. It is illustrated and discussed with Example 1.
- *Structural Inference:* Given the proven performance guarantees of our dynamic programming algorithm (see Appendix G.3), the primary remaining bottleneck in structural inference lies in the incorrect labeling of operators. For example, consider an assembly line where two elements from two queues are combined to produce an item in a third queue. The correct operator is:
$$TV(x) = V(x'_1 = [x_1 - 1,\ x_2 - 1,\ x_3 + 1])$$
The algorithm may erroneously interpret this as a tandem departure operator:
$$T_{\mathrm{TD}(1,3)}V(x) = V(x' = [x_1 - 1,\ x_2,\ x_3 + 1])$$
which neglects the role of x_2 and leads to incorrect structural predictions.

Limitations of the Current Framework.
Example 2 highlights intrinsic limitations of the current framework for structural result extraction. Our approach relies on known theoretical propagation results for a fixed set of operators. In that example, we identify three possible reasons why structural results cannot be inferred:

1. The correct common propagated space has not yet been identified.
2. The appropriate operators to model the problem are missing from the current library and would need to be introduced along with corresponding propagation rules.
3. It is theoretically possible that the Bellman equation propagates a functional space even though none of the individual operators does—our current framework relies on a sufficient but not necessary condition, namely that *each* operator propagates the space.

To overcome these limitations, future work could involve extending the family of operators and enriching the library of propagation results. This can be done manually, following the methodology of Koole, or through automated discovery using machine learning techniques.

L Dataset

We constructed a dataset of 36 natural language descriptions of queueing control problems, varying in difficulty by state space size, state constraints, and number of event types. To assess performance in structure identification and support future research, the dataset includes three categories: (1) problems with provable structural results (e.g., Example 1); (2) problems with empirically observed, but unprovable, structures (e.g., Example 2); and (3) problems with no structural results. All problems address realistic issues from domains such as hospital management Bekker et al. (2017), telecommunications Koole & Mandelbaum (2002); Bhulai & Koole (2003); Bekker et al. (2011); Zhang et al. (2025c), freight dispatching Schwarz & Daduna (2006); Amjath et al. (2023), assembly lines Adeyinka & Kareem (2018), and traffic control Boon et al. (2023).

The problems are inspired by the literature and have each been manually designed and solved by an expert in OR. The ground truth consists of five randomly chosen states together with their optimal values. These optimal values were computed from the OR formulation using a general-purpose value iteration solver, with convergence assumed once the value changed by less than 0.05 between two

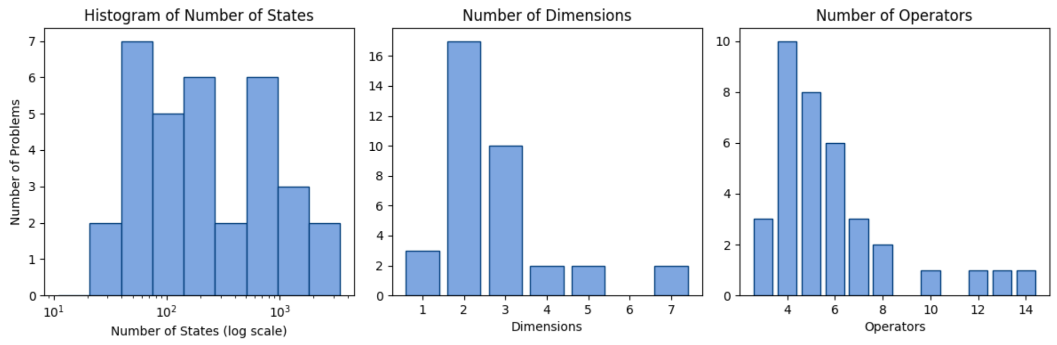


Figure 11: Distribution of some complexity measures across the dataset.

iterations. A formulation is therefore considered correct if its optimal value function matches the ground truth within a tolerance of 0.1.

Tab. 6 and Fig. 11 sum up the characteristics of the dataset.

Table 6: Overview of the dataset by domain.

Domain	**# Problems**	**With Structural Properties**	**Avg. # States**	**Avg. Dim.**	**Avg. # Events**
Hospital management	15	12	1017	3.3	6.7
Freight dispatching	9	6	335	2.2	5.9
Assembly lines	6	0	217	2.8	4.5
Traffic control	4	0	93	2.0	5.25
Telecommunications	2	2	726	2.5	5.0
Total	36	20 (56 %)	594	2.8	5.9

M Prompts

We aim to elicit the LLM to give the final output as a Python dictionary of the following format:

formalization_dict template

```
{
  "parameters": {
    "values": {},
    "descriptions": {}
  },
  "state_space": {
    "variables": {},
    "constraints": {}
  },
  "objective_function": {
    "operational_cost_per_unit_time": null,
    "discount_factor": null,
    "description": null
  },
  "events": {},
  "events_probabilities": {
    "uniformization_factor": null,
    "probabilities": {}
  },
  "operators": {}
}
```

In the following, we describe the prompts that ask the LLM to generate nodes in the Monte Carlo tree. As we will see, the prompts are completely application-agnostic and can be directly applied to problems in different domains.

We give the LLM a general context prompt at the beginning.

General context prompt

I have a sequential decision problem:

```
<<<PROBLEM DESCRIPTION>>>
```

I want to analyze this problem using the **Event-based Optimization Framework** introduced by Koole (2007). This framework models systems where actions are taken in response to random, uncontrollable events that occur over time. The framework's core components are **event operators**, which serve as the building blocks for defining the system's value function.

Framework Description

1. **Event Operators**: Event operators represent the dynamic transformations of the value function in response to specific system events. Formally:

$$T_i : V \to V, \quad \text{for } i = 0, \dots, k-1,$$

   where T_i maps a value function V from the state space to a new value function over the same space.

2. **Recursive Value Function**: The system's value function, V_n, is defined recursively to capture the sequential nature of decision-making:

$$V_n = \sum_{i=0}^{k-1} p_i T_i(V_{n-1}),$$

   where:
   - V_{n-1} is the value function from the previous step.
   - p_i is the probability of event i occurring at each step, satisfying $\sum_{i=0}^{k-1} p_i = 1$.
   - T_i represents the impact of event i on the system.
   - C is the operational cost.
   - α is the discounting factor.

This formalization captures the stochastic nature of the problem, where random events dictate the evolution of the system, and the value function reflects the accumulated system performance over time.

Objective Given this theoretical foundation, we need to formalize the problem by defining the following components:

```
<<<FORMALIZATION DICT>>>
```

The following prompt generates the first-level nodes m_1 of the Monte Carlo tree.

Parameters completion prompt

Task: Complete `formalization_dict` based on the problem description, you should complete the `"parameters"` field which consists of assigning constants to descriptive variable names. Only complete `"parameters"` and nothing else.

Guidelines:

1. Your primary responsibility is to define all the parameters from the problem description that will later be used to define the state space, objective function, events and operators.
2. You may include additional parameters in a format suitable for facilitating the subsequent tasks of defining the state space, objective function, events and operators.
3. For parameters that involve multiple indices (e.g. `x[i]` or `x[i, j]`), use the most appropriate data structure, such as lists, dictionaries, or dictionaries with tuple keys, to represent them.
4. For each parameter, include a clear, descriptive comment explaining its meaning.
5. Ensure that the parameter names (keys) are descriptive and intuitive.
6. The dictionary should contain two keys: `"values"` and `"descriptions"`.
   - `"values"` should contain the actual parameter values.
   - `"descriptions"` should contain the descriptions of the parameters.

Format: Return only the Python dictionary update (i.e., `formalization_dict["parameters"] = ...`) following the described requirements.

The following prompt generates the second-level nodes m_2 of the Monte Carlo tree.

State space completion prompt

Task: Complete the `"state space"` field in the `formalization_dict` based on the problem description. Specifically, define:

1. Variables: Populate the `"variables"` field to represent the system's state. Each key-value pair must adhere to the following structure:

```
<key>: {
   "description": <description>,
   "type": <type>,
   "iteration_space": <space>,
   "default_value": <default_value>
}
```

Guidelines:

- **Essential Variables Only**: Include only the strictly necessary variables to describe the system's state. Exclude costs, events, or redundant variables.
- **Less is better**: Due to the curse of dimensionality, keep the number of state variables minimal. If a variable can be derived from others, do not include it.
- **Symbolic Name**: Use unique, descriptive names that reflect the variable's role.
- **Description**: Clearly explain each variable's role in the system.
- **Type**: Either `"int"` or `"float"`.
- **Parameter Variables**: Use parameter-defined values directly (without using `parameters[...]`)
- **Iteration Space**: Use Python-style list comprehension syntax (e.g., `range(n)`). Use `None` for scalar variables.
- **Default Value**: Must be a single `int` or `float` to initialize the variable across its iteration space.
- **Consolidation**: Merge similar variables under a single key with an appropriate iteration space.

2. Constraints: Populate the `"constraints"` field to define boundaries of the state space. Each key-value pair must adhere to the following structure:

```
<constraint_key>: {
   "equation": <mathematical_equation>,
   "description": <description>
}
```

Guidelines:

- **Descriptive Constraints**: Use meaningful names.
- **Mathematical Description**: Use Python-like math expressions. Use list comprehensions when appropriate.
- **Equality and Inequality**: Capture valid bounds and implicit problem constraints.
- **Parameter Variables**: Refer directly to them, no nested `parameters[...]` syntax.
- **Indexed Variables**: Use bracket notation (e.g., `x[i]`).
- **Comments**: Each constraint should be preceded by a comment explaining its purpose.

Important Notes:

- If the problem has no explicit constraints, consider implicit ones.
- If no constraints apply, return: `formalization_dict["state space"]["constraints"] = {None: None}`

Return: Only the Python dictionary update (i.e., `formalization_dict["state space"] = ...`) following the described requirements.

The following prompt generates the third-level nodes m_3 of the Monte Carlo tree.

Objective function completion prompt

Task: Complete `formalization_dict` based on the problem description, you should complete the `"objective function"` field. Follow these requirements:

Define the operational cost and discount factor such that it adheres to the following structure:

```
"objective function" = {
   "operational cost":  <cost>,
   "discount factor":  <factor>,
   "description":  <description>
}
```

1. Operational Cost:

- Replace `<cost>` with the **operational cost** of the system. This represents the cost incurred *between events*, such as maintaining the system or executing ongoing operations.
- **Important:** Do not include costs triggered by events or actions — those go in the `"events"` field.
- Use parameter-defined variables, not hard-coded values. Express the cost as a string formula.
- **Default:** If not provided, use `"0"`.

2. Discount Factor:

- Replace `<factor>` with the system's **discount factor**, which determines the relative importance of future rewards.
- Use parameter-defined variables if mentioned. Otherwise, use the default value `0.95`.

3. Description:

- Replace `<description>` with a short explanation justifying the chosen operational cost and discount factor.

Return: Only the Python dictionary update (i.e., `formalization_dict["objective function"] = ...`) following these requirements.

The next few prompts generate the fourth-level nodes m_4 of the Monte Carlo tree.

Events completion prompt

Task: Complete `formalization_dict` based on the problem description, you should complete the `"events"` field. Follow these requirements:

Each key-value pair in the dictionary must adhere to the following structure:

```
<key>: {
  "description": <description>,
  "actions": {
    <action_key>: {
      "description": <action description>,
      "cost": <cost>,
      "state_change": <state_change>
    }
  }
}
```

Guidelines:

1. **Events:** Define the events that can occur in the system. An event is a random occurrence that changes the state, the cost, or triggers a need for action.
2. Each `<key>` must be a symbolic name representing a distinct event and will be used in the Python implementation of operators.
3. **Descriptive Events:** Use unique, symbolic names for each event.
4. **Event Description:** Replace `<description>` with a string describing the event's impact on the system.
5. **Actions:** Define the actions that can be taken in response to each event. If there is no decision involved, define only one action named `"default"`.
6. **Action Description:** Replace `<action description>` with a description of the action's role.
7. **Cost:** Replace `<cost>` with a string representing the formula for the cost of the event-action pair. Use parameter-defined variables.
8. **State Change:** Replace `<state_change>` with a list of equations (as strings) describing how state variables change due to the event and action.
9. **Constraints:** Do not repeat feasibility constraints — infeasible states automatically result in $V = +\infty$.
10. **Parameter Variables:** Use parameter-defined variables directly; do not reference them via `parameters[...]`.
11. **Indexed Variables:** Use bracket notation for indices (e.g., `x[0]`).
12. **One Event per Entry:** Do not merge events. Each event must have its own dictionary entry. Avoid undefined parameters (e.g., no generic `i` in event keys).

Return: Only the Python dictionary update (i.e., `formalization_dict["events"] = ...`) following these requirements.

Events probabilities completion prompt

Task: Complete `formalization_dict` based on the problem description, you should complete the `"events_probabilities"` field.

Requirements: Each key-value pair in the dictionary must adhere to the following structure:

`<key>: <probability>`

1. **Events Probabilities:** Define the probabilities of each event that can occur in the system. If needed, consider a uniformization framework.
2. Each `<key>` links to the corresponding event defined in the `"events"` section.
3. Replace `<probability>` by the probability of the event. Use parameter-defined variables instead of hard-coded values. Put the formula in a string format.
4. **Parameter Variables:** Use parameter-defined variables directly (do not reference them via `parameters[...]`).
5. **Indexed Variables:** For indexed state variables or parameters, use standard bracket notation (e.g., `x[i]`).

Return: Only the Python dictionary update (i.e., `formalization_dict["events probabilities"] = ...`) following these requirements.

Operators completion prompt

Task: Complete `formalization_dict` based on the problem description, you should complete the `"operator"` field.

Each operator is a 'sub Bellman' equation linking the optimal value function after the occurrence of the corresponding event (and the optimal action) to the value function before that.

List of Available Operators:

- `T_{A}`
  - **Description:** Arrival operator
  - **Definition:** $T_A(\texttt{state\_variable})\, f(x) = f(x + e_{\texttt{state\_variable}})$
  - **Parameters:** `['state_variable']`
- `T_{CA}`
  - **Description:** Controlled arrival operator
  - **Definition:** $T_{CA}(\texttt{state\_variable}, c_1, c_2)\, f(x) = \min(f(x) + c_1, f(x + e_{\texttt{state\_variable}}) + c_2)$
  - **Parameters:** `['state_variable', 'c_1', 'c_2']`
- `T_{D}`
  - **Description:** Departure operator
  - **Definition:** $T_D(\texttt{state\_variable})\, f(x) = f((x - e_{\texttt{state\_variable}})^+)$
  - **Parameters:** `['state_variable']`
- `T_{CD}`
  - **Description:** Controlled departure operator
  - **Definition:**

$$T_{CD}(\texttt{state\_variable}, c_1, c_2)\, f(x) = \begin{cases} \min(f(x) + c_1, \\ \quad f(x - e_{\texttt{state\_variable}}) + c_2), & \text{if } x_{\texttt{state\_variable}} > 0 \\ c_1 + f(x), & \text{otherwise} \end{cases}$$

  - **Parameters:** `['state_variable', 'c_1', 'c_2']`
- `T_{TD}`

- **Description:** Tandem departure operator
- **Definition:**

$$T_{TD}(\texttt{state\_variable\_1},\texttt{state\_variable\_2})\, f(x) = \begin{cases} f(x - e_{\texttt{state\_variable\_1}} \\ \quad + e_{\texttt{state\_variable\_2}}), & \text{if } x_{\texttt{state\_variable\_1}} > 0 \\ f(x), & \text{otherwise} \end{cases}$$

- **Parameters:** `['state_variable_1', 'state_variable_2']`

• `T_{CTD}`

- **Description:** Controlled tandem departure operator
- **Definition:**

$$T_{CTD}(\texttt{state\_variable\_1}, \texttt{state\_variable\_2}, c_1, c_2)\, f(x) = \begin{cases} \min(f(x) + c_1,\ f(x + e_{\texttt{state\_variable\_1}} + e_{\texttt{state\_variable\_2}}) + c_2), & \text{if } x_{\texttt{state\_variable\_1}} > 0 \\ c_1 + f(x), & \text{otherwise} \end{cases}$$

- **Parameters:** `['state_variable_1', 'state_variable_2', 'c_1', 'c_2']`

Field Format: Each key-value pair in the dictionary must follow this structure:

```
<key>: {
   "description": <description>,
   "operator": <operator>
}
```

Guidelines:

1. Replace `<key>` with the name of the event the operator corresponds to.
2. Replace `<description>` with a string that explains the operator's impact.
3. Replace `<operator>` with the selected operator and its parameters in string format (e.g., `"T_{CA}(i=x[1], c_1=1, c_2=2)"`).
4. If no operator fits, use `None`.
5. Use parameter-defined variables directly (not via `parameters[...]`).
6. Use bracket notation for indexed variables (e.g., `x[i]`).
7. For repeating patterns, use Python for-loops or list comprehensions where applicable.
8. Do **not** include event probabilities — they are handled elsewhere.

Return: Only the Python dictionary update (i.e., `formalization_dict["operators"] = ...`) following these requirements.