


Abstraction-Based Decision Making for Statistical Properties

Filip Cano ✉ 

Graz University of Technology, Austria

Thomas A. Henzinger ✉ 

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Bettina Könighofer ✉ 

Graz University of Technology, Austria

Konstantin Kueffner ✉ 

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Kaushik Mallik ✉ 

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

Sequential decision-making in probabilistic environments is a fundamental problem with many applications in AI and economics. In this paper, we present an algorithm for synthesizing sequential decision-making agents that optimize *statistical properties* such as maximum and average response times. In the general setting of sequential decision-making, the environment is modeled as a random process that generates inputs. The agent responds to each input, aiming to maximize rewards and minimize costs within a specified time horizon. The corresponding synthesis problem is known to be PSPACE-hard. We consider the special case where the input distribution, reward, and cost depend on input-output statistics specified by counter automata. For such problems, this paper presents the first PTIME synthesis algorithms. We introduce the notion of *statistical abstraction*, which clusters statistically indistinguishable input-output sequences into equivalence classes. This abstraction allows for a dynamic programming algorithm whose complexity grows polynomially with the considered horizon, making the *statistical case* exponentially more efficient than the general case. We evaluate our algorithm on three different application scenarios of a client-server protocol, where multiple clients compete via bidding to gain access to the service offered by the server. The synthesized policies optimize profit while guaranteeing that none of the server's clients is disproportionately starved of the service.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Computational pricing and auctions; Theory of computation → Abstraction

Keywords and phrases Abstract interpretation, Sequential decision making, Counter machines

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.2

Category Invited Talk

Supplementary Material

Software: <https://github.com/filipcano/abstraction-based-decision-making-FSCD24>
archived at `swb:1:dir:1113fa6039e8d3792e47a1fed24cc5210159ebef`

Funding This work is partly supported by the European Research Council under Grant No.: ERC-2020-AdG 101020093. It is also partially supported by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.



© Filip Cano, Thomas A. Henzinger, Bettina Könighofer, Konstantin Kueffner, and Kaushik Mallik;
licensed under Creative Commons License CC-BY 4.0

9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024).

Editor: Jakob Rehof; Article No. 2; pp. 2:1–2:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Sequential decision-making is a core algorithmic task in many AI-based planning and control problems with uncertain environments. The environment produces a sequence of inputs one at a time, and the decision-making agent needs to produce outputs on each of the inputs as soon as they arrive – without having seen the inputs from the future. We consider the finite-horizon setting, where the decision horizon, i.e., the total number of inputs to appear, is finite and fixed apriori.

The problem of computing an optimal decision-making agent in the sequential setting is known to be PSPACE-hard. Our contribution is to introduce a class of statistical properties for which the same problem can be solved in polynomial time. A property is called statistical if its satisfaction can be measured by keeping track of small statistics, typically a fixed set of states and a fixed set of counter values. The class of statistical properties not only includes every regular property but also includes richer quantitative properties like maximum and average response times. We present a new synthesis algorithm for sequential decision-makers whose constraint, objective function, and environment model can all be captured using statistical properties. The complexity of our algorithm, which is based on a notion of *statistical abstraction*, grows only polynomially with respect to the considered horizon.

The problem setting. The inputs to our synthesis algorithm are a probabilistic model of the environment generating a random sequence of inputs, a horizon length, a qualitative constraint function, and a quantitative objective function over the generated input-output sequence. The environment model, the constraint, and the objective function are required to be given as statistical properties, which guarantee that their outputs depend only on some small statistics about the past sequence of inputs and outputs. Since our statistical properties include regular properties, the qualitative constraint can be any finite-state constraint. Henceforth, we will refer to the constraint and the objective function as *cost constraint* and *reward*, respectively. We propose a synthesis procedure to compute a decision maker that, at each step, reads the current environment input and computes an output such that the expected reward is maximized over all possible futures that satisfy the cost constraint up to the given horizon. If no such future is possible, the decision maker outputs “fail.”

Example – Synthesis of responsive servers. We consider a simple client-server model, inspired by online advertisements [10], where multiple clients (i.e., the advertisers) compete via a bidding mechanism to access a certain service (i.e., the act of putting up the ads) hosted by the server (i.e., the advertisement publisher). A greedy server would always accept the highest bidder to maximize its profit, but this could potentially starve the weaker clients, monopolize the market, and ultimately result in overall bad ratings that are harmful to its business in the long run. We consider three different synthesis problems for designing servers that strike a balance between profit-making and maintaining acceptable service quality for all clients – both aspects being representable as statistical properties.

Synthesis problem I: Balanced servers. We say that a server is balanced if the individual frequencies of the clients getting access are similar. A *balanced server* ensures that no client experiences a significantly lower number of server accesses compared to the other clients. Given a time horizon t , a constant probability distribution over the prices that the clients will offer, and given a bound on the “imbalance,” how do we compute a server policy that will maximize the expected profit while keeping the imbalance within bounds in time t ?

Synthesis problem II: Maximally responsive servers. While the balanced server guarantees that each client would get a similar *total* number of accesses by the end of the horizon, the clients may end up waiting for longer periods. *Maximally responsive servers* minimize the maximal waiting time for each client. As a trade-off, the server may need to occasionally select the client offering a lower price. We quantify this trade-off using the *opportunity cost*, defined as the extra profit the server could make by selecting the client offering the highest price at a given stage. Given a time horizon t , a constant distribution over prices that the clients will offer, and a bound k on the total opportunity costs, how do we compute a server policy that is maximally responsive while making sure that the sum of opportunity costs after time t does not exceed k ?

Synthesis problem III: Clientele-aware servers. So far, the probability distribution over the offer prices by the clients was assumed to remain constant. The more realistic setting is when the clients adjust their prices based on their past experiences with the service: a dissatisfied client would lower the offer prices in the future whereas a satisfied client would be willing to pay more. Therefore, a far-sighted server policy should limit its initial rejection rates to keep up the expected offer prices in the future – even if this may need sacrificing the profit in the beginning. We consider the setting with just one client and assume that the server has a budget of N on the number of times the client can be accepted. Given a time horizon t and a model of how the past decisions affect the future price distributions, how do we compute a server policy that will maximize the expected profit while accepting the client’s request for at most N times?

Algorithmic solution and complexity. We present synthesis algorithms for the case that the probability distribution of the environment, the reward, and the cost constraint are all provided as statistical properties. For the general class of properties, the optimal decision of the agent at any point may need to consider every possible sequence of future inputs and outputs. This causes an expensive blow-up that is unavoidable in general, as the problem is known to be PSPACE-hard [12].

Our key insight is that for properties we call *statistical*, such as the maximum response time of a server, the computational blow-up can be avoided by using a statistical abstraction of the history. Such an abstraction of the observed input-output sequences combines statistically indistinguishable sequences into the same equivalence classes. For properties for which the number of equivalence classes grows polynomially with time, we obtain a *polynomial-time algorithm* for computing the optimal decision maker. Any property that can be described using a constant number of counters (like the maximum response time in the aforementioned client-server examples) and a constant number of states (like regular languages) falls into this category. In contrast, any order-dependent property (like a discounted sum) would have an exponentially growing number of equivalence classes. Note, that there are other common “statistics,” such as variance, which are more complicated to compute and do not lie within our class of statistical properties.

We show how to adapt the standard dynamic programming algorithm to work on the abstracted domain, rather than on the full history. We also show a correspondence between the size of the abstraction and the size of the representation of statistical properties using *counter automata*.

Experiments. We evaluated our method on the three examples described above. For each example, we compared the computational performances of the dynamic programming algorithm with and without using a statistical abstraction. Without the abstraction, the

algorithm runs out of memory (with a memory limit of 200 GB) for horizon lengths less than 10 in all cases. With the abstraction, however, the algorithm manages much larger instances, exceeding 400 in all cases and reaching up to 60000 for the balanced server.

Furthermore, we demonstrate how our decision-makers fare in simulations. For balanced servers, we demonstrate that our synthesized policy maintains balance while achieving performance similar to that of a greedy policy. Additionally, it significantly outperforms a policy synthesized to be balanced on expectation. For maximally responsive servers, we show how our method produces policies that can significantly reduce maximum waiting time, paying a comparatively small price in opportunity cost. Finally, we analyse the policies synthesized for clientele-aware servers for different budgets.

2 Sequential Decision Making for Statistical Properties

2.1 The Sequential Decision Making Problem

We first formulate a general problem statement for sequential decision making involving arbitrary *quantitative* properties. Afterwards, we will identify the important subclass of the problem where all the quantitative properties are actually statistical in nature, like the ones we described in Sec. 1.

We consider the alternating turn-based interaction of the decision maker with its uncertain environment. At each interaction phase, called a *stage*, the environment samples an action from a known probability distribution over the *input alphabet* \mathcal{X} , and the decision maker responds by generating an action from the *output alphabet* \mathcal{Y} , and both actions may depend on the past stages. Formally, the environment is modeled as *stochastic generators* of the form $\theta: (\mathcal{X} \times \mathcal{Y})^* \rightarrow \Delta(\mathcal{X})$, and the system is modeled as *transducers* of the form $\gamma: (\mathcal{X} \times \mathcal{Y})^* \times \mathcal{X} \rightarrow \mathcal{Y}$. Let Θ be the set of every stochastic generator and Γ be the set of every transducer. The interaction between θ and γ induces a probability distribution $\mathbb{P}_{\theta, \gamma}$ over the space of finite input-output sequences – called *traces* – as follows. For every stage $t > 0$ and for every trace $\vec{z}_t = z_1 \dots z_t \in (\mathcal{X} \times \mathcal{Y})^t$,

$$\mathbb{P}_{\theta, \gamma}(\vec{z}_t) := \prod_{i=1}^t \mathbb{P}_{\theta, \gamma}(z_i \mid \vec{z}_{i-1}),$$

where

$$\mathbb{P}_{\theta, \gamma}(z_i \mid \vec{z}_{i-1}) := \begin{cases} \theta(\vec{z}_{i-1})(x_i) & \text{if } \gamma(\vec{z}_{i-1}, x_i) = y_i, \\ 0 & \text{otherwise,} \end{cases}$$

with the convention that \vec{z}_0 represents the empty word. We will write $\text{Dom}(\mathbb{P}_{\theta, \gamma})$ to denote the domain of $\mathbb{P}_{\theta, \gamma}$, i.e., the set of traces with positive probabilities. Sometimes, we will call traces as *histories* to stress that the given input-output sequence consists of concrete inputs and outputs as observed in the past.

We consider a lexicographic specification for the transducer, formalized as follows. We consider the real-valued *reward function* $\text{rew}: (\mathcal{X} \times \mathcal{Y})^* \rightarrow \mathbb{R}$, and the binary *cost function* $\text{cost}: (\mathcal{X} \times \mathcal{Y})^* \rightarrow \{0, 1\}$ (where “1” could represent that the cost is below a given threshold). We define the expected value of the reward after stage t as $\mathbb{E}_{\theta, \gamma}^t[\text{rew}] := \sum_{\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t} \text{rew}(\vec{z}_t) \cdot \mathbb{P}_{\theta, \gamma}(\vec{z}_t)$. The lexicographic specification requires the cost to be 1 for every probable input sequence of the environment (“hard” objective) and the expected reward is maximized (“soft” objective), at the end of a given number of stages.

► **Problem 1.** Let $T \in \mathbb{N}$ be the given horizon length (i.e., the last stage index), θ be the given stochastic generator, \mathbf{rew} be the given reward function, and \mathbf{cost} be the given cost function. Let $\Gamma_{\text{feas}}^\theta \subseteq \Gamma$ be the set of feasible transducers fulfilling the cost constraint in the presence of θ , i.e.,

$$\Gamma_{\text{feas}}^\theta := \{\gamma \in \Gamma \mid \forall \vec{z}_T \in \text{Dom}(\mathbb{P}_{\theta, \gamma}) . \mathbf{cost}(\vec{z}_T) = 1\}. \quad (1)$$

Compute the *optimal* transducer γ^* that is feasible and is reward-optimal, i.e.,

$$\gamma^* = \arg \max_{\gamma \in \Gamma_{\text{feas}}^\theta} \mathbb{E}_{\theta, \gamma}^T[\mathbf{rew}]. \quad (2)$$

We will call the tuple $\langle \mathcal{X}, \mathcal{Y}, \theta, \mathbf{rew}, \mathbf{cost}, T \rangle$ a *problem instance*.

The finiteness of the horizon length is a standard choice in the literature on sequential decision making problems [9, 2] and is a natural choice in many practical situations. Naïvely lifting this finite-horizon restriction is technically tricky because it is unclear how $\Gamma_{\text{feas}}^\theta$ will be defined in the first place.

It follows from known results [12] that Prob. 1 is PSPACE-hard in general; see Sec. 3.1 for details. We present a PTIME algorithm for the special case where the functions θ , \mathbf{rew} , and \mathbf{cost} depend only on some particular statistics of histories.

2.2 Statistical Properties

In many practical instances of Prob. 1, including the motivating examples in Sec. 1, the functions \mathbf{rew} , \mathbf{cost} , and θ depend only on some aggregated statistic of the history, and the exact order of the inputs and outputs is unimportant. We formalize this in the following.

► **Definition 2** (Statistics). *Let \mathcal{W} be an alphabet and \mathcal{S} be an output domain. An \mathcal{S} -statistic over \mathcal{W} is a function $\mu: \mathcal{W}^* \rightarrow \mathcal{S}$.*

We will omit \mathcal{W} or \mathcal{S} whenever they are either clear from the context or are unimportant. Following is the key property of statistics that will be useful to us.

► **Definition 3** (Well-behaved statistics). *A statistic μ over \mathcal{W} is called well-behaved if for every $\vec{u}, \vec{v}, \vec{w} \in \mathcal{W}^*$, $\mu(\vec{u}) = \mu(\vec{v})$ implies $\mu(\vec{u}\vec{w}) = \mu(\vec{v}\vec{w})$.*

These concepts are illustrated using the following example.

► **Example 4.** Suppose we are given a sequence of the toss outcomes of a given coin, where heads and tails are represented as “1” and “0,” respectively. Following are examples of statistics over $\{0, 1\}$: total number of heads $\mu_1(\cdot)$, largest number of consecutive tails between any two heads $\mu_2(\cdot)$, average number of heads $\mu_3(\cdot)$, the mode statistic $\mu_4(\cdot)$, etc. Among these, it can be easily verified that μ_1 and μ_2 are well-behaved. The average statistic μ_3 is not well-behaved: The sequences 10 and 1010 have the same averages (which is $1/2$), but extending them with the sequence 11 gives us different averages ($3/4$ and $2/3$, respectively). The mode statistic μ_4 is also not well-behaved: the sequences 110 and 1110 have the same mode 1, but extending them with the sequence 00 gives us different modes (0 and 1, respectively).

Let μ be an \mathcal{S} -statistic over the alphabet \mathcal{W} . For every $t > 0$, the statistic μ induces an equivalence relation $\equiv_{\mu, t}$ on the set \mathcal{W}^t , defined as follows: $\vec{w}_t \equiv_{\mu, t} \vec{z}_t$ iff $\mu(\vec{w}_t) = \mu(\vec{z}_t)$. For instance, for the statistic μ_1 from Ex. 4 and for $t = 3$, we have $110 \equiv_{\mu_1, 3} 101 \equiv_{\mu_1, 3} 011$, because $\mu_1(110) = \mu_1(101) = \mu_1(011) = 2$ and each of them have length 3.

In the subsequent sections, the equivalence relation $\equiv_{\mu,t}$ will give way to a small abstraction of the set \mathcal{W}^t , where all the words in a given equivalence class will be abstracted by a single representative word from that equivalence class. Therefore, the larger the number of equivalence classes, the larger and more complex will be the abstraction. We formalize this as a measure of the complexity of statistics.

► **Definition 5** (Size of statistics). *Let $\mu: \mathcal{W}^* \rightarrow \mathcal{S}$ be a statistic. The size of μ is the function $size_\mu: \mathbb{N} \rightarrow \mathbb{N}$ mapping every t to the number of equivalence classes in \mathcal{W}^t induced by the equivalence relation $\equiv_{\mu,t}$.*

Consider the statistics μ_1 and μ_4 from Ex. 4. For any given t , it is easy to see that $size_{\mu_1}(t) = t + 1$ (because there can be $0, 1, \dots, t$ number of “1”-s) and $size_{\mu_4}(t) = 2$ (because the mode of any sequence can be either 0 or 1).

For a given $t \in \mathbb{N}$, let $\mathcal{S}_t \subseteq \mathcal{S}$ be the set of every valuation of μ on every word of length t , i.e., $\mathcal{S}_t = \mu(\mathcal{W}^t)$. A t -reconstructor of μ is any function $\kappa_t: \mathcal{S}_t \rightarrow \mathcal{W}^t$ such that if $\kappa_t(s) = \vec{w}_t$ then $\mu(\vec{w}_t) = s$. Observe that κ_t is not unique. For the statistic μ_1 in Ex. 4, we have $\mathcal{S}_t = \{0, 1, 2\}$, and one possible 2-reconstructor is given as $\kappa_2(0) = 00$, $\kappa_2(1) = 01$, and $\kappa_2(2) = 11$. The following claim follows immediately.

► **Proposition 6.** *Let $\mu: \mathcal{W} \rightarrow \mathcal{S}$ be a statistic and κ_t be a t -reconstructor of μ . Then for every $\vec{w} \in \mathcal{W}^t$, $\mu(\vec{w}) = \mu(\kappa_t \circ \mu(\vec{w}))$.*

We will now use well-behaved statistics to define the class of problem instances that are amenable to efficient computations and are our subject of study.

► **Definition 7** (μ -representability of functions). *Let μ be a \mathcal{S} -statistic over \mathcal{W} . The function $f: \mathcal{W}^* \rightarrow \mathcal{U}$ is μ -representable, if there exists a function $\hat{f}: \mathcal{S} \rightarrow \mathcal{U}$ such that for every $\vec{w} \in \mathcal{W}^*$, $f(\vec{w}) = \hat{f}(\mu(\vec{w}))$.*

We will later show that Prob. 1 can be efficiently solved if we can identify a well-behaved statistic μ such that θ , **rew**, and **cost** are μ -representable and μ is “small” in size. How to determine a small, well-behaved μ such that a given property f is μ -representable is a problem on its own. In Sec. 2.4, we present a pragmatic approach to quickly identify μ and its size when f is specified as a counter automaton. The problem of finding a small μ for f then boils down to the problem of finding a small counter automaton representing f .

2.3 Examples: Synthesis of Responsive Servers

Below, we describe how the examples from Sec. 1 can be formalized. We first introduce some common notation for the client-server model. Suppose that there are two clients, A and B , competing to access a resource hosted by a server. At each stage t , A and B concurrently submit their offer prices, a_t and b_t respectively, indicating the amount they are willing to pay for the service. Let a_t and b_t be non-negative integers, both less than or equal to C , where zero indicates that the respective client does not request the service at that stage. The server responds with the decision $d_t \in \mathcal{Y} = \{A, B\}$, representing whether client A or B get access. We model the pair of clients as the environment and the server as the system, i.e., $\mathcal{X} = [0; C] \times [0; C]$ and $\mathcal{Y} = \{A, B\}$.

► **Example 8** (Balanced server). Recall that the objective of the balanced server is to maximize profit while ensuring an even allocation of resources to its clients by the end of a given horizon. If the server initially acts greedily and accepts whoever offers the higher price, then it may later need to pay high opportunity costs (i.e., sacrifice profits to

balance out the past imbalances). We formulate the synthesis problem for the balanced server problem as follows. We assume that the distribution θ is known and remains fixed over time and that we have given an arbitrary trace $\vec{z}_t = (a_1, b_1)d_1 \dots (a_t, b_t)d_t \in (\mathcal{X} \times \mathcal{Y})^t$. We define the *profit* at stage i , denoted as $\text{Profit}(a_i, b_i, d_i)$, to be equal to a_i if $d_i = A$ and else equal to b_i (if $d_i = B$). Furthermore, we define the *imbalance* over \vec{z}_t , denoted as $\text{Imbalance}(\vec{z}_t)$, as the absolute difference of acceptance rates between the two clients, i.e., $\text{Imbalance}(\vec{z}_t) := \left| \sum_{i=1}^t \mathbf{1}(d_i = A \wedge a_i \neq 0) - \sum_{i=1}^t \mathbf{1}(d_i = B \wedge b_i \neq 0) \right|$. Then, the synthesis problem for balanced servers can be encoded using Prob. 1 by assigning $\text{rew}(\vec{z}_t) = \sum_{i=1}^t \text{Profit}(a_i, b_i, d_i)$ and $\text{cost}(\vec{z}_t) = 1$ iff $\text{Imbalance}(\vec{z}_t) \leq \delta$ for a given tolerance $\delta > 0$.

► **Example 9** (Maximally responsive server). The objective of the maximally responsive server is to minimize the maximum waiting time for each client while paying a bounded total amount of the sum of opportunity costs. As for the balanced server problem, we assume that the distribution θ is known and does not change over time and we have given an arbitrary trace $\vec{z}_t = (a_1, b_1)d_1 \dots (a_t, b_t)d_t \in (\mathcal{X} \times \mathcal{Y})^t$. We define the *maximum waiting time*, denoted as $\text{MaxWait}(d_1 \dots d_t)$, to be equal to the length of the largest subsequence of consecutive A 's or B 's in the sequence $d_1 \dots d_t$. Additionally, we define the *opportunity cost* at stage i , denoted as $\text{OppCost}(a_i, b_i, d_i)$, to be equal to $|a_i - b_i|$ if $a_i > b_i$ but $d_i = B$ or if $a_i < b_i$ but $d_i = A$. Then, the synthesis problem for maximally responsive servers can be encoded using Prob. 1 by assigning $\text{rew}(\vec{z}_t) = -\text{MaxWait}(d_1 \dots d_t)$ and $\text{cost}(\vec{z}_t) = 1$ iff $\sum_{i=1}^t \text{OppCost}(a_i, b_i, d_i) \leq k$ for a given budget $k > 0$, and is 0 otherwise.

► **Example 10** (Clientele-aware server). Clientele-aware servers account for the loss of clients due to dissatisfied clients who were rejected in the past. For simplicity, we assume that only client A is active and B is inactive; this can be achieved by simply setting $b_i = 0$ always. For simplicity, we will write (a_i, \cdot) to denote $(a_i, b_i = 0)$. The critical component in this example is the environment θ . We assume $\theta(\varepsilon)$, for the empty word ε , is the given initial distribution \mathbb{P}_0 over \mathcal{X} such that the probability of seeing the price a from the client A conditioned on $a \neq 0$ is fixed as \mathbb{P}_A , i.e., $\mathbb{P}_0((a, \cdot) \mid a \neq 0) = \mathbb{P}_A(a)$. We assume that a constant $\delta \in (0, 1)$ is given such that for every $\vec{z}_i = (a_1, \cdot)d_1 \dots (a_i, \cdot)d_i \in (\mathcal{X} \times \mathcal{Y})^i$, we have:

$$\theta(\vec{z}_i)((a = 0, \cdot)) = \mathbb{P}_0((a_0 = 0, \cdot)) + \max \left\{ 0, \min \left\{ 1, \delta \cdot \sum_{j=1}^{i-1} (\mathbf{1}(a_j \neq 0 \wedge d_j = A) - \mathbf{1}(a_j \neq 0 \wedge d_j \neq A)) \right\} \right\},$$

and $\theta(\vec{z}_i)((a, \cdot) \mid a \neq 0) = \mathbb{P}_A(a)$ for every $a \in [1; C]$. Intuitively, the probability of seeing “ $a = 0$ ” increases or decreases by δ for every rejection or acceptance of A , respectively. Conditioned on “ $a \neq 0$ ” being true, the probability of seeing a price a from A is fixed to $\mathbb{P}_A(a)$. The reward and cost functions are: $\text{rew}(\vec{z}_t) = \sum_{i=1}^t \text{Profit}(a_i, \cdot, d_i) = \sum_{i=1}^t a_i \cdot \mathbf{1}(d_i = A)$ and $\text{cost}(\vec{z}_t) = 1$ iff $\sum_{i=1}^t \mathbf{1}(d_i = A) \leq N$ for a given N .

2.4 Specifications using Counter Automata

Counter automaton is a rich framework for modeling functions that has access to both states and a finite set of counters. When rew , cost , and θ are given as counter automata, we show that there is a systematic procedure to extract the statistical complexity and the witness statistic μ with their respective μ -representations $\widehat{\text{rew}}$, $\widehat{\text{cost}}$, and $\widehat{\theta}$.

Our counter automata are adoptions of counter monitors introduced by Ferrère et al. [6]. A counter is an integer variable, which can be read and written according to relations and functions in the signature $S = \langle 0, +1, -1, \leq \rangle$. In particular, a *test* is a conjunction of atomic formulas over S and their negation, and an *update* is a mapping from variables to terms over S . The sets of tests and updates over a set of counters X are denoted as $\Phi(X)$ and $\Gamma(X)$, respectively.

► **Definition 11** (Counter automata). *Let Σ be a given input alphabet and D be an output domain. A counter automaton A on Σ and D is a tuple $\langle \Sigma, D, R, Q, \lambda, q_{\text{init}}, \delta \rangle$, where R is a finite set of registers, Q is a finite set of states, $\lambda: Q \times \mathbb{N}^R \rightarrow D$ is an output function, $q_{\text{init}} \in Q$ is the initial state, and $\delta \subseteq Q \times \Sigma \times \Phi(R) \times \Gamma(R) \times Q$ is a transition relation such that for every state $q \in Q$, input $\sigma \in \Sigma$, and valuation $v: R \rightarrow \mathbb{N}$, there exists a unique edge $(q, \sigma, \phi, \gamma, q') \in \delta$ with $v \models \phi$ satisfied.*

A run of the automaton on a given finite word $w = w_0 w_1 \dots w_n \in \Sigma^*$ is the unique sequence of transitions $(q_0, v_0) \xrightarrow{w_0} (q_1, v_1) \xrightarrow{w_1} \dots \xrightarrow{w_n} (q_n, v_n)$ such that $q_0 = q_{\text{init}}$, $v_0(r) = 0$ for every $r \in R$, and we write $(q, v) \xrightarrow{\sigma} (q', v')$ when there exists an edge $(q, \sigma, \phi, \gamma, q') \in \delta$ such that $v \models \phi$ and $v'(r) = v(\gamma(r))$ for every $r \in R$. We write $\text{final}_A(w)$ to denote the last configuration (q_n, v_n) . The semantics of a counter automaton A is given as $\llbracket A \rrbracket(w) = \lambda(\text{final}_A(w)) \in D$ where (q, v) is the final configuration of the run of A on w .

► **Proposition 12.** *Suppose $\varphi: \Sigma^* \rightarrow D$ is a function and A_φ is the equivalent counter automaton on Σ such that $\varphi(w) = \llbracket A_\varphi \rrbracket(w)$ for every $w \in \Sigma^*$. The function φ is μ -representable for the statistic μ defined as $\mu(w) := \text{final}_{A_\varphi}(w)$, and the μ -representation of φ is given by the output function of A_φ .*

When θ , **rew**, and **cost** are expressed using counter automata, Prop. 12 outlines a simple syntactic approach to extract the well-behaved statistic μ and the respective μ -representation. The output domains of the counter-automata for θ , **rew**, and **cost** are, respectively, \mathbb{R} , $\{0, 1\}$, and $\Delta(\mathcal{X})$. Fig. 1 shows examples of counter automata representations of some of the statistical properties from Ex. 8–10.

For any given counter automaton $A = \langle \Sigma, D, R, Q, \lambda, q_{\text{init}}, \delta \rangle$, the size of the underlying statistic μ as defined in Prop. 12 is given by:

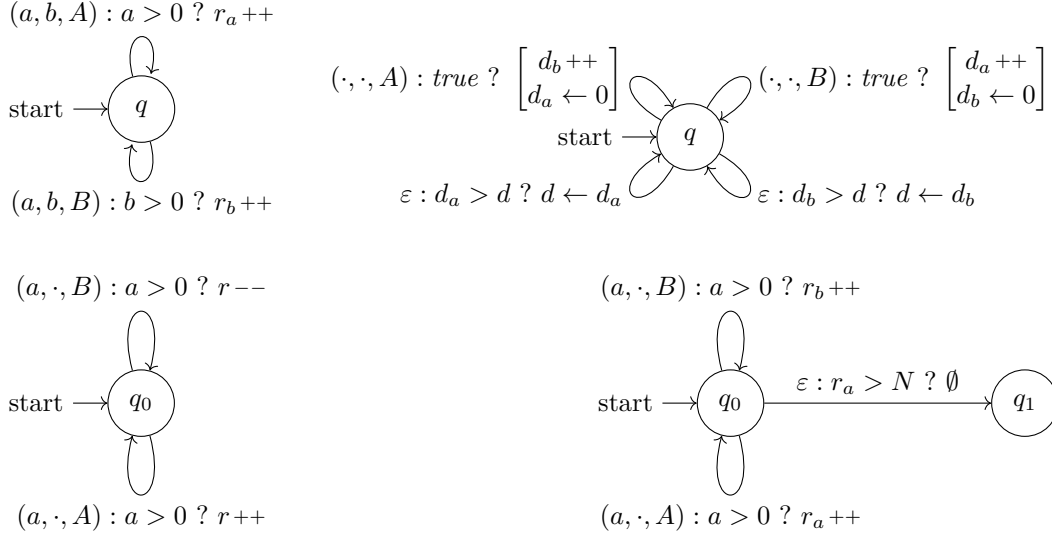
$$\text{size}_\mu(t) = |\{\text{final}_A(w) \mid w \in \Sigma^t\}| = |Q| \times \underbrace{t \times |R|}_P, \quad (3)$$

where P is the total number of counter values, obtained by using the fact that each counter will have a value in $[0; t - 1]$ after t stages.

3 Policy Synthesis Algorithms

3.1 Dynamic Programming

Prob. 1 can be solved using dynamic programming, whose time and space complexities would unfortunately grow exponentially with respect to the horizon T . Let \vec{z}_t be a given trace of length $t \in [0; T]$ and $x \in \mathcal{X}$ be an input; \vec{z}_0 is the empty word. We write $\mathbf{v}_t(\vec{z}_t)$ and $\mathbf{w}_t(\vec{z}_t, x)$ to denote the maximum expected rewards achievable in the remaining $T - t$ stages, while fulfilling the cost constraint, *before* and *after* revealing the next input x , respectively. We can compute the functions $\{\mathbf{v}_t(\cdot)\}_{t \in [0; T]}$ and $\{\mathbf{w}_t(\cdot, \cdot)\}_{t \in [1; T]}$ recursively as below:



■ **Figure 1** Counter automata representation of several properties used in Ex. 8, 9, and 10. TOP-LEFT represents **Imbalance** from Ex. 8: The counters r_a and r_b count the total numbers of accepted nonzero price requests from the clients A and B , respectively. For **Imbalance**, we use the output function $\lambda(q, r_a, r_b) = 1$ if $|r_a - r_b| \leq \delta$ and $\lambda(q, r_a, r_b) = 0$ otherwise. TOP-RIGHT represents **MaxWait** from Ex. 9: The counters d_a and d_b count the total numbers of stages since requests from, respectively, clients A and B were accepted (i.e., the current waiting times). The counter d keeps track of the current maximum waiting times for both clients. For **MaxWait**, we use the output function $\lambda(q, d_a, d_b, d) = d$. BOTTOM-LEFT represents the environment from Ex. 10: The counter r represents the difference between the numbers of acceptances and rejections of offers by A that were positive. For the environment, we use the output function $\lambda(q, r)$ that is a probability distribution over $[0; C]$ such that $\lambda(q, r)(0) = \mathbb{P}_0((a = 0)) + \max\{0, \min\{1, r\}\}$. BOTTOM-RIGHT represents the cost function from Ex. 10: The counters r_a and r_b represent the numbers of times A and B were accepted, respectively, given the price offered by A was positive. For the constraint, we use the output function $\lambda(q_0, r_a, r_b) = 1$ and $\lambda(q_1, r_a, r_b) = 0$.

$$\begin{aligned}
 t \in [0; T-1] : \quad \mathbf{v}_t(\vec{z}_t) &= \sum_{x \in \mathcal{X}} \mathbf{w}_t(\vec{z}_t, x) \cdot \theta(\vec{z}_t)(x), \\
 \mathbf{w}_t(\vec{z}_t, x) &= \max_{y \in \mathcal{Y}} \mathbf{v}_{t+1}(\vec{z}_t xy), \\
 t = T : \quad \mathbf{v}_t(\vec{z}_{t=T}) &= \begin{cases} -\infty & \text{if } \mathbf{cost}(\vec{z}_T) = 0, \\ \mathbf{rew}(\vec{z}_T) & \text{otherwise.} \end{cases}
 \end{aligned} \tag{standard-DP}$$

The sought optimal transducer γ^* of Prob. 1 is then obtained as: for every $t \in [0; T-1]$ and for every $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$, if $\mathbf{v}_t(\vec{z}_t) \neq -\infty$ then $\gamma(\vec{z}_t x) = \arg \max_{y \in \mathcal{Y}} \mathbf{v}_{t+1}(\vec{z}_t xy)$, and else $\gamma(\vec{z}_t x) = \text{FAIL}$. Note that even if $\mathbf{v}_0(\varepsilon) \neq -\infty$, for the empty word ε , the environmental uncertainty may force the system to a stage from where the cost constraint can no longer be satisfied, which is when γ would output **FAIL**.

As we unroll the recursion tree in (standard-DP) forward, we observe that every \mathbf{v}_t node has $|\mathcal{X}|$ children and every \mathbf{w}_t node has $|\mathcal{Y}|$ children. Therefore, the size of the entire tree will be $\mathcal{O}(|\mathcal{X}|^T \cdot |\mathcal{Y}|^T)$, causing an exponential blow-up in time and space complexities. In fact, from a classical result by Papadimitriou [12], it follows that Prob. 1 is PSPACE-hard. Note that the paper considers the setting without the cost function **cost**, which is a special case of our setting. The paper has proven this special case to be PSPACE-complete.

The difficulty of Prob. 1 stems from the *history dependence* of the stochastic generator θ , the reward function **rew**, and the cost function **cost**. Take for example the case where the output symbol selected at a given stage affects the distribution of θ at a future stage, for which building the entire recursion tree is unavoidable [12]. A similar situation arises when the reward and the cost are affected by all the past inputs and outputs and their exact order.

3.2 Simple Problem Instances

We take a slight detour and present a subclass of problem instances for which a PTIME algorithm exists; ideas from this algorithm will be useful in Sec. 3.4 when we will present special optimizations of our algorithm. We call the problem instance $\langle \mathcal{X}, \mathcal{Y}, \theta, \text{rew}, \text{cost}, T \rangle$ *simple* if the following hold:

Additive reward and cost: For every $t \in [1; T]$, let there be functions $r_t, c_t: \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$, assigning constant, history-independent reward and cost to each individual input-output pair, such that for every $\vec{z} = x_1 y_1 x_2 y_2 \dots$, $\text{rew}(\vec{z}) = \sum_t r_t(x_t y_t)$ and $\text{cost}(\vec{z}) = 1$ iff $\sum_t c_t(x_t y_t) \leq B$ for a given *budget* B .

History-independent environment: For every $t \in [0; T - 1]$, let $\bar{\theta}_t$ be a fixed probability distribution such that $\theta(\vec{w}_t) = \theta(\vec{z}_t) = \bar{\theta}_t$ for every $\vec{w}_t, \vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$.

Then we can modify the dynamic programming algorithm in (**standard-DP**) by replacing the history \vec{z} with just the cost b incurred in \vec{z} , and define the respective counterparts $\mathbf{v}'_t(b), \mathbf{w}'_t(b, x)$ of $\mathbf{v}_t(\vec{z}), \mathbf{w}_t(\vec{z}, x)$ as below:

$$\begin{aligned} t \in [0 : T - 1] : \quad \mathbf{v}'_t(b) &= \sum_{x \in \mathcal{X}} \mathbf{w}'_t(b, x) \cdot \bar{\theta}_t(x), \\ \mathbf{w}'_t(b, x) &= \max_{y \in \mathcal{Y}} [\mathbf{v}'_{t+1}(b + c_t(x, y)) + r_t(x, y)], \\ t = T : \quad \mathbf{v}'_t(b) &= \begin{cases} -\infty & \text{if } b > B, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (\text{simple-DP})$$

The sought optimal transducer γ^* of Prob. 1 is obtained as follows: for every $t \in [0; T - 1]$ and for every $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$ with $b = \sum_{i=1}^t c_i(x_i y_i)$, if $\mathbf{v}'_t(b) \neq -\infty$ then $\gamma(\vec{z}_t x) = \arg \max_{y \in \mathcal{Y}} \mathbf{v}'_{t+1}(b + c_t(xy))$, and else $\gamma(\vec{z}_t x) = \text{FAIL}$, where **FAIL** is defined as for the case of **standard-DP**, and indicates that the cost constraint cannot be satisfied in any possible future extension of the current trace.

Let us analyze the size of the recursion tree of (**simple-DP**). At every stage t , the number of $\mathbf{v}'_t(b)$ nodes is B and the number of $\mathbf{w}'_t(b, x)$ nodes is $B \cdot |\mathcal{X}|$, implying that the tree's size is $\mathcal{O}(T \cdot B \cdot |\mathcal{X}|)$, establishing the PTIME complexity.

In general, if even one of the three functions **rew**, **cost**, and θ is history-dependent, like in the examples of Sec. 2.3, the PTIME algorithm is not applicable. In the next section, we show a sub-class of Prob. 1 with statistical properties – with the dependence being only on a small statistic over the history instead of the exact history – for which efficient algorithms exist.

3.3 Statistical Abstraction

We now present the main contribution of this paper. Suppose we are given a problem instance $\langle \mathcal{X}, \mathcal{Y}, \theta, \text{rew}, \text{cost}, T \rangle$ such that there exists a well-behaved \mathcal{S} -statistic μ on $(\mathcal{X} \times \mathcal{Y})^*$ for which θ , **rew**, and **cost** are μ -representible. In this case, we no longer need to consider each history \vec{z}_t while using **standard-DP**, but rather we can combine histories that are statistically indistinguishable, i.e., equivalent with respect to $\equiv_{\mu, t}$ for a given t . This way

we obtain an abstraction-based implementation of **standard-DP**— called **statistical-DP**, where, for each stage t , $(\mathcal{X} \times \mathcal{Y})^t$ serves as the concrete domain of histories, \mathcal{S} serves as its abstraction, μ serves as the abstraction function, and κ_t serves as the concretization function. Following we give a key consistency property of the abstraction, whose proof follows from the well-behavedness of μ .

► **Proposition 13.** *For a fixed stage $t < T$ and an arbitrary history $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$ with $\mu(\vec{z}_t) = s$, it holds that for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we have $\mu(\vec{z}_t xy) = \mu(\kappa_t(s)xy)$.*

The above claim suggests that the statistic μ of an input-output trace can be computed sequentially and efficiently, without keeping track of the entire history but only from the value of the statistic in the last step. This enables us to only keep track of the value of the statistic μ in each recursive call, instead of the full histories and lets us lift the functions \mathbf{v}_t and \mathbf{w}_t to their abstract counterparts, respectively, $\hat{\mathbf{v}}_t$ and $\hat{\mathbf{w}}_t$ as follows. For every stage $t \in [0; T]$, we define $\hat{\mathbf{v}}_t: \mathcal{S} \rightarrow \mathbb{R}$ and $\hat{\mathbf{w}}_t: \mathcal{S} \rightarrow \{0, 1\}$ as below:

$$\begin{aligned}
 t \in [0; T-1]: \quad \hat{\mathbf{v}}_t(s) &= \sum_{x \in \mathcal{X}} \hat{\mathbf{w}}_t(s, x) \cdot \overbrace{\hat{\theta}(s)(x)}^{\text{I}}, \\
 \hat{\mathbf{w}}_t(s, x) &= \max_{y \in \mathcal{Y}} \overbrace{\hat{\mathbf{v}}_{t+1}(\mu(\kappa_t(s)xy))}^{\text{II}}, & (\text{statistical-DP}) \\
 t = T: \quad \hat{\mathbf{v}}_t(s) &= \begin{cases} -\infty & \text{if } \widehat{\text{cost}}(s) = 0 \\ \underbrace{\widehat{\text{rew}}(s)}_{\text{III}} & \text{otherwise.} \end{cases} \quad \leftarrow \text{IV}
 \end{aligned}$$

We use the annotations **I**, **II**, **III**, and **IV** in Sec. 3.4 to present further optimizations.

The sought optimal transducer γ^* of Prob. 1 is obtained as follows: for every $t \in [0; T-1]$ and every $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$, if $\hat{\mathbf{v}}_t(\mu(\vec{z}_t)) \neq -\infty$ then $\gamma(\vec{z}_t x) = \arg \max_{y \in \mathcal{Y}} \hat{\mathbf{v}}_{t+1}(\mu(\vec{z}_t xy))$, and else $\gamma(\vec{z}_t x) = \text{FAIL}$, where **FAIL** is as defined for the case of **standard-DP**, and indicates that the cost constraint cannot be satisfied in any possible future extension of the current trace.

Next we compare **statistical-DP** from above with the general dynamic programming scheme **standard-DP** for Prob. 1. The highlight of **statistical-DP** is that the functions $\hat{\mathbf{v}}_t$ and $\hat{\mathbf{w}}_t$ do not depend on the history \vec{z}_t (unlike \mathbf{v}_t and \mathbf{w}_t in **standard-DP**) anymore; rather they only depend on the statistical values of histories. If we interpret each $\hat{\mathbf{v}}_t(\cdot)$ and $\hat{\mathbf{w}}_t(\cdot, x)$ as vectors V_t and W_t , respectively, then the sizes of V_t and W_t are equal to the size $\text{size}_\mu(t)$ of the statistic μ on t , and not on the size of $(\mathcal{X} \times \mathcal{Y})^t$. Usually, $\text{size}_\mu(t)$ is substantially smaller than $|(\mathcal{X} \times \mathcal{Y})^t|$, and the computational savings with **statistical-DP** are significant. For instance, in Ex. 8 and Ex. 9, when the property is specified using the counter automata in Fig. 1, it follows from (3) that $\text{size}_\mu(t) = 1 \times t \times 4 = 4t$, which is significantly smaller than $|(\mathcal{X} \times \mathcal{Y})^t| = (|G| \times |C| \times |E|)^t = (2 \times 2c \times 2)^t = (8c)^t$ for sufficiently large t .

Theorem 14 demonstrates the correctness and the complexity of **statistical-DP**.

► **Theorem 14.** *Let $\langle \mathcal{X}, \mathcal{Y}, \theta, \text{rew}, \text{cost}, T \rangle$ be a problem instance and μ be a well-behaved statistic on $(\mathcal{X} \times \mathcal{Y})^*$ such that θ , **rew**, and **cost** are μ -representible. For every stage $t \in [0; T]$, and every $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$ with $\mu(\vec{z}_t) = s$ we have $\mathbf{v}_t(\vec{z}_t) = \hat{\mathbf{v}}_t(s)$. Furthermore, the computation of $\{\hat{\mathbf{v}}_t(\cdot)\}_{t \in [0; T]}$ requires $\mathcal{O}\left(|\mathcal{X}| \cdot |\mathcal{Y}| \cdot \sum_{t=1}^T \text{size}_\mu(t)\right)$ time and $\mathcal{O}\left(\sum_{t=1}^T \text{size}_\mu(t)\right)$ space.*

► **Remark 15 (Product statistic).** If θ , **rew**, and **cost** are represented using individual well-behaved statistics μ_θ , μ_{rew} , and μ_{cost} , respectively, the common statistic μ of Thm. 14 can be obtained by computing a product statistic as follows: For every \vec{z}_t , $\mu(\vec{z}_t) :=$

$(\mu_\theta(\vec{z}_t), \mu_{\text{rew}}(\vec{z}_t), \mu_{\text{cost}}(\vec{z}_t))$. If θ , rew , and cost are provided as counter automata, then μ can be extracted from their product automaton. Consequently, we obtain $\text{size}_\mu(t) = \text{size}_{\mu_\theta}(t) \times \text{size}_{\mu_{\text{rew}}}(t) \times \text{size}_{\mu_{\text{cost}}}(t)$.

Proof of Thm. 14. The first part of the proof is via backward induction on the stages. Throughout, we will use the trace $\vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$ for which $\mu(\vec{z}_t) = s$. To make the proof easy to follow, we show that both (i) $\mathbf{v}_t(\vec{z}_t) = \widehat{\mathbf{v}}_t(s)$ and (ii) $\mathbf{w}_t(\vec{z}_t, x) = \widehat{\mathbf{w}}_t(s, x)$ hold.

For the base case $t = T$, from Prop. 6 and the μ -representability of rew and cost , it follows that $\text{rew}(\vec{z}_T) = \text{rew}(\kappa_t \circ \mu(\vec{z}_T)) = \widehat{\text{rew}}(s)$ and $\text{cost}(\vec{z}_T) = \text{cost}(\kappa_t \circ \mu(\vec{z}_T)) = \widehat{\text{cost}}(s)$. Therefore, $\mathbf{v}_T(\vec{z}_T) = \widehat{\mathbf{v}}_T(s)$ holds as the base case for Claim (i). From the base case and Prop. 13 we obtain that, for every $t \in [0; T - 1]$, it holds that:

$$\begin{aligned} \mathbf{w}_t(\vec{z}_t, x) &= \max_{y \in \mathcal{Y}} \mathbf{v}_{t+1}(\vec{z}_t xy) = \max_{y \in \mathcal{Y}} \widehat{\mathbf{v}}_{t+1}(\mu(\vec{z}_t xy)) \\ &= \max_{y \in \mathcal{Y}} \widehat{\mathbf{v}}_{t+1}(\mu(\kappa_t(s)xy)) = \widehat{\mathbf{w}}_t(s, x). \end{aligned} \quad (4)$$

By using $t = T - 1$, we obtain $\mathbf{w}_{T-1}(\vec{z}_{T-1}, x) = \widehat{\mathbf{w}}_{T-1}(\mu(\vec{z}_{T-1}), x)$, which serves as the base case for Claim (ii).

Now suppose Claim (i) holds for an arbitrary stage $t + 1 \leq T$. Then the same derivation (4) can be used to show that Claim (ii) holds for the stage t . These two statements serve as the induction steps, and we will show that Claim (i) holds for stage t and Claim (ii) holds for stage $t - 1$. The following is the proof of the induction step for Claim (i) which holds due to the base case and the μ -representability of θ :

$$\mathbf{v}_t(\vec{z}_t) = \sum_{x \in \mathcal{X}} \mathbf{w}_t(\vec{z}_t, x) \cdot \theta(\vec{z}_t)(x) = \sum_{x \in \mathcal{X}} \widehat{\mathbf{w}}_t(s, x) \cdot \widehat{\theta}(s)(x) = \widehat{\mathbf{v}}_t(s). \quad (5)$$

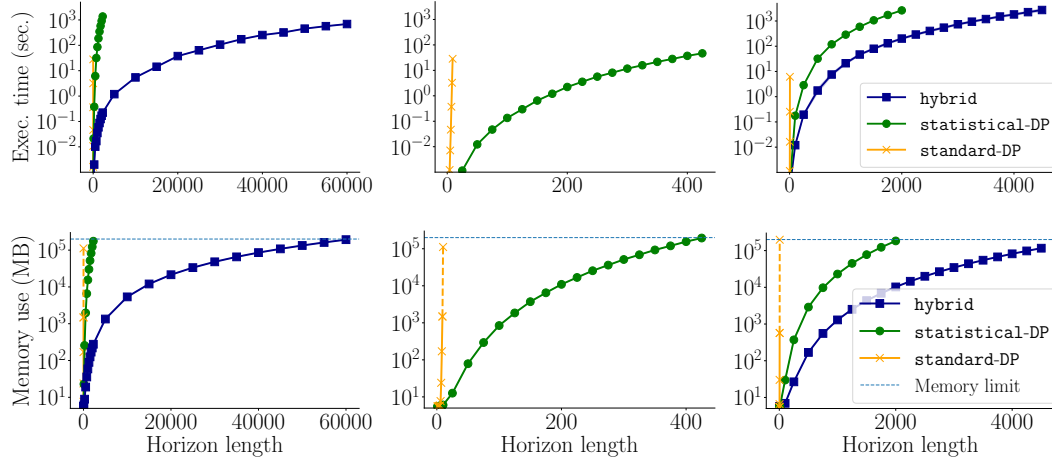
We can use the same derivation (4) by only substituting $t + 1$ with t , where the second inequality will now follow from (5), and obtain the proof of induction step for Claim (ii).

The time and space complexity bounds are established as follows. It follows from (3) that for each stage t , there are at most $\text{size}_\mu(t)$ distinct values of $s \in \mathcal{S}$ which can represent the whole set $(\mathcal{X} \times \mathcal{Y})^t$. Then, $\widehat{\mathbf{v}}_t(\cdot)$ will have at most $\text{size}_\mu(t)$ arguments. Therefore, there are $\mathcal{O}(\sum_{t=1}^T \text{size}_\mu(t))$ elements in $\{\widehat{\mathbf{v}}_t(\cdot)\}_{t \in [1; T]}$, for which we will need $\mathcal{O}(\sum_{t=1}^T \text{size}_\mu(t))$ space in total. On the other hand, for every distinct element in $\{\widehat{\mathbf{v}}_t\}_{t \in [1; T]}$, the recursion will have $|\mathcal{X}| \times |\mathcal{Y}|$ branchings ($|\mathcal{X}|$ branchings for the sum operator and $|\mathcal{Y}|$ branchings for the max). Therefore, we will need $\mathcal{O}(|\mathcal{X}| \cdot |\mathcal{Y}| \cdot \sum_{t=1}^T \text{size}_\mu(t))$ time in total for the computation. \blacktriangleleft

3.4 Additional Optimizations for Special Cases

We identify three special cases with potentially additional complexity improvements. We highlight conditions under which we can disregard each individual size, size_{μ_θ} , $\text{size}_{\mu_{\text{rew}}}$, and $\text{size}_{\mu_{\text{cost}}}$, one by one, when computing size_μ , as outlined in Remark 15. This optimization potentially reduces size_μ , providing synthesis algorithms with improved complexity.

Independent environmental distributions. Suppose θ has the property that for every $t \in [0; T - 1]$ and for every $\vec{w}_t, \vec{z}_t \in (\mathcal{X} \times \mathcal{Y})^t$, we have $\theta(\vec{w}_t) \equiv \theta(\vec{z}_t)$ (in distribution), which we write as θ_t . We can replace $\widehat{\theta}(s)(x)$ in (I) in **statistical-DP** with $\theta_t(x)$. Removing $\widehat{\theta}$ from **statistical-DP** means that we can ignore size_{μ_θ} while computing size_μ of the product statistic μ as in Rem. 15 (i.e., set $\text{size}_{\mu_\theta}(t) = 1$ for all t).



■ **Figure 2** Top row: plots of computation time (in log-scale) versus horizon length for Ex. 8, 9, and 10 (left to right). Bottom row: plots of memory usage (in log-scale) versus horizon length for Ex. 8, 9, and 10 (left to right).

Additive reward. Suppose rew has the property that for every $\vec{z}_t = x_1 y_1 \dots x_t y_t \in (\mathcal{X} \times \mathcal{Y})^t$, we can express the reward as $\text{rew}(\vec{z}_t) = \sum_{i=1}^t r(x_i y_i)$ for some positive, stage-invariant reward $r: \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$. In **statistical-DP**, we can replace $\hat{v}_{t+1}(\mu(\kappa_t(s)xy))$ in (II) with $\hat{v}_{t+1}(\mu(\kappa_t(s)xy)) + r(xy)$ and replace $\widehat{\text{rew}}(s)$ in (III) with “0,” like in the simple case described in Sec. 3.2. Removing $\widehat{\text{rew}}$ from **statistical-DP** means that we can ignore $\text{size}_{\mu_{\text{rew}}}$ while computing size_{μ} of the product statistic μ as in Rem. 15. Both Ex. 8 and Ex. 10 have additive rewards, which helped us to obtain faster computations in our experiments; see Sec. 4.1.

Cost is a safety constraint. Suppose cost is a stateless, boolean safety property, specified using a function $c: \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$ such that $\text{cost}(x_1 y_1 \dots x_t y_t) = 0$ iff there exists an $i \in [1; t]$ for which $c(x_i y_i) = 1$. In **statistical-DP**, we can replace (II) with “0” if $c(xy) = 1$, and remove (IV). Removing $\widehat{\text{cost}}$ from **statistical-DP** means that we can ignore $\text{size}_{\mu_{\text{cost}}}$ while computing size_{μ} of the product statistic μ as in Rem. 15.

4 Experiments

4.1 Comparison of Computational Performances

We considered problem instances with varying time horizons for all three of our examples, namely synthesis of balanced, maximally responsive, and clientele-aware servers, formalized in Ex. 8, 9, and 10, respectively. For each example, we compared the computational performances between **standard-DP** and **statistical-DP**, and for Ex. 8 and Ex. 10, we also compared the hybrid approach discussed in Sec. 3.4 for when the reward is additive. Fig. 2 compares the time and memory usage of our synthesis tool to solve each problem instance using different approaches. For all reported results we use a time limit of 1 hour and a memory limit of 200 GB. As expected, the time and memory usage for **standard-DP** grows exponentially with T in all cases, and the memory limit is reached way earlier compared to the **statistical-DP** approach. The hybrid approach, when available, performs even better than **statistical-DP**.

4.2 Analysis of Synthesized Policies

We demonstrate the quality of the policies synthesized using `statistical-DP` for each of the three examples that we consider. Tab. 1 summarizes the problem instances and Fig. 3 summarizes the aggregated outcomes of 100 simulations in each case.

Example – Balanced server. We compare the quality of the synthesized balanced server policy, formalized in Ex. 8 and from hereon referred to as the *balanced-by-construction* policy, with two baseline policies. The first baseline is a *greedy* policy, that always accepts the client with the highest price, and in the case of a tie, accepts the client that would make the imbalance smaller. The second baseline maximizes the expected profit, constrained to having a balanced execution in expectation. We call this the *balanced-on-average* policy. Fig. 3a shows that the *balanced-by-construction policy is balanced on every run*, whereas both the greedy policy and the balanced-on-average policy are unbalanced in the worst case. Besides, the average profit of the balanced-by-construction policy remains competitive (not shown in figures). In particular, on average the profit for the balanced-by-construction policy, the greedy policy, and the balanced-on-average policy were, respectively: 10719, 10723, and 8267. In this setting, the balanced-by-construction policy obtains almost the same profit as the greedy policy, while maintaining the balance constraint. The balanced-on-average policy obtains worse results both in terms of profit and balance.

Example – Maximally responsive server. We compare the performance of a maximally responsive server, formalized in Ex. 9, with the same input distribution and different budgets. As a baseline, we use a greedy policy that accepts the client offering the highest price. Fig. 3b shows the average and standard deviation of the maximal response time achieved by policies with different bounds on the total opportunity costs. In the experiments illustrated in Fig. 3b, the greedy policy obtains an average maximum response time of 24.4 time steps after a horizon length of 100 time steps. Our maximally responsive server with budgets 5, 15, and 25, respectively, achieves average maximum waiting time of 12.35, 7.94, and 5.55 while obtaining 98.75%, 96.25%, and 93.81% of the profit obtained by the greedy policy. Intuitively, the higher the budget on opportunity costs is, the more freedom the server gets to reduce the waiting time, though this impacts the overall profit.

Example – Clientele-aware server. We compare the performances of four different clientele-aware server policies (from Ex. 10) with four different constraints on the maximum number N of accepted requests. From the plot in Fig. 3c, we observe that the higher the value of N is, the lower is the initial price threshold, i.e., the higher is the acceptance rate in the beginning. This happens because for a higher value of N , the server has more freedom to initially accept more requests from the client (by lowering the threshold) to improve the chances of seeing better offer prices in future.

■ **Table 1** Where $q_{n,k}$ is the centered Binomial $_{n,k/n}$, i.e. $X - \mathbb{E}(X) \sim q_{n,k}$ with $X \sim \text{Binomial}_{n,k/n}$.

Example	Horizon	Price A PDF	Price B PDF	Objective	Constraint
8	1000	Binomial $_{20,0.49}$	Binomial $_{20,0.51}$	profit	imbalance ($\delta = 5$)
9	100	Binomial $_{6,0.3}$	Binomial $_{6,0.75}$	wait time	budget ($k \in \{5, 15, 25\}$)
10	500	$q_{6,1}$	-	profit	budget ($N \in \{50, 150, 250, 350\}$)

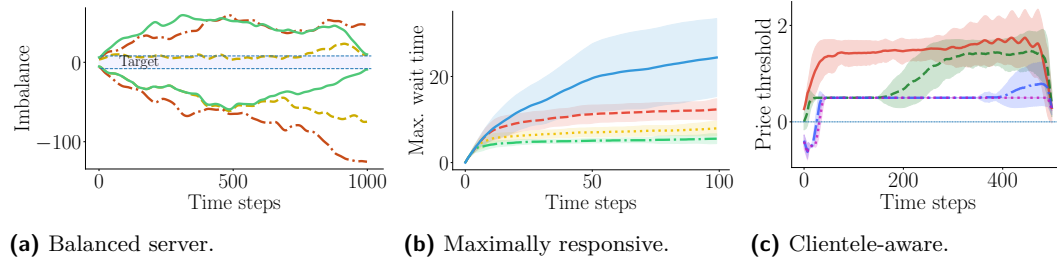


Figure 3 **Fig. 3a** depicts the minimal and maximal imbalance values over 100 simulations for the balanced-by-construction policy (—), the greedy policy (---), and the balanced-on-average policy (-.-). **Fig. 3b** depicts the maximal wait time values obtained over 100 simulations after deploying the maximally responsive server for a budget of 5 (---), 10 (....), and 25 (-.-). We also show for comparison, the maximal response time obtained by a server with a greedy policy (—). **Fig. 3c** depicts mean and standard deviation of the acceptance thresholds of the synthesized policy for various budget constraints: 50 (—), 150 (---), 250 (-.-), and 350 (....).

5 Related Work

The literature on bounded-horizon sequential decision-making broadly distinguishes between *model-free* and *model-based* instances of the problem, based on whether a model of the environment is available [14]. In this paper, we study the model-based problem.

For model-based problems, a common assumption is that the environment can be modeled with a Markov Decision Process (MDP) [14]. Hence, the synthesis problem reduces to finding the optimal policy in a finite MDP [13], for which extensive literature is available [7]. The size of the MDP can however be exponential in the horizon length, and solving the derived optimization problem is known to be PSPACE-hard [12]. Our statistical abstraction can be viewed as a small abstraction over the state space of this exponentially large MDP, which is only implicitly built and explored via **statistical-DP**. A large body of work on model-based sequential decision-making falls under the term *optimal stopping problems* [15]. The theory of optimal stopping revolves around the problem of choosing the ideal time to take a specific action to either maximize an expected reward or minimize an expected cost. Among those works, a variety of environment models – both Markovian [15] and non-Markovian [3] – and a variety of properties [1, 4, 11, 8] have been studied. To the best of our knowledge, the history-dependent statistical properties that we consider are beyond the reach of the existing algorithms in the optimal stopping literature. Moreover, due to the generality of our assumptions, common analytical tools, like competitive ratios [16], would fail to provide anything beyond trivial bounds.

For model-free problems, optimal policy synthesis algorithms are predominantly data-driven and rely on learning [18]. Under the assumption that the unknown state space is finite, PAC-style guarantees are possible [17]. The quality of algorithms in this area is often assessed using regret where the performance is compared against the model-based setting [5, 19]. We plan to consider this direction for future work, with an unknown environment and the goal to compute policies that fulfill the objectives with high probabilities.

6 Conclusion

We considered the sequential decision-making problem with uncertain environments where the objective of the decision-maker includes fulfillment of a statistical property over a finite horizon of a given length. Although the problem was known to be PSPACE-hard for general

properties (possibly non-stochastic), for statistical properties, we present a solution whose complexity grows only polynomially with respect to the horizon length. The crux of our approach is a novel statistical abstraction that clusters statistically indistinguishable traces of the system. Using a prototype implementation, we demonstrated the computational performance and effectiveness of our approach on three examples of designing server policies that need to act fairly towards its clients.

Several future directions exist. Firstly, the unbounded-horizon setting presents an interesting and non-trivial extension, given the inherent challenge of defining the feasible set of policies (Eq. 1) within this setting. Secondly, it will be valuable to consider the multi-objective problem where both **cost** and **rew** need to be optimized. The goal here will be to compute a Pareto-optimal solution. In contrast, our approach is currently limited to unconditionally fulfilling the qualitative constraint **cost**. Finally, our work establishes only the *sufficient* requirements on the properties (μ -representibility for a well-behaved statistic μ) which admit efficient abstractions. Determining the *necessary* requirements will be an interesting theoretical quest, which may improve our understanding of the complexity landscape of sequential decision-making problems.

References

- 1 Stefan Ankirchner, Maïke Klein, and Thomas Kruse. A verification theorem for optimal stopping problems with expectation constraints. *Applied Mathematics & Optimization*, 79:145–177, 2019.
- 2 Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.
- 3 Elena Bandini, Andrea Cosso, Marco Fuhrman, and Huy  n Pham. Backward SDEs for optimal control of partially observed path-dependent stochastic systems: a control randomization approach. *The Annals of Applied Probability*, 28(3):1634–1678, 2018.
- 4 Erhan Bayraktar and Song Yao. Optimal stopping with expectation constraints. *The Annals of Applied Probability*, 34(1B):917–959, 2024.
- 5 Yonathan Efroni, Shie Mannor, and Matteo Pirodda. Exploration-exploitation in constrained MDPs. *arXiv preprint*, 2020. [arXiv:2003.02189](#).
- 6 Thomas Ferr  re, Thomas A Henzinger, and Bernhard Kragl. Monitoring event frequencies. In *28th EACSL Annual Conference on Computer Science Logic (CSL)*. Schloss Dagstuhl – Leibniz-Zentrum f  r Informatik, 2020.
- 7 Abhijit Gosavi et al. *Simulation-based optimization*. Springer, 2015.
- 8 Sigrid K  llblad. A dynamic programming approach to distribution-constrained optimal stopping. *The Annals of Applied Probability*, 32(3):1902–1928, 2022.
- 9 Robert Kleinberg and S Matthew Weinberg. Matroid prophet inequalities and applications to multi-dimensional mechanism design. *Games and Economic Behavior*, 113:97–115, 2019.
- 10 S. Muthukrishnan. Ad exchanges: Research issues. In *Internet and Network Economics, 5th International Workshop (WINE)*, pages 1–12, 2009.
- 11 Aaron Zeff Palmer and Alexander Vladimirovsky. Optimal stopping with a probabilistic constraint. *Journal of Optimization Theory and Applications*, 175:795–817, 2017.
- 12 Christos H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.
- 13 David C Parkes and Satinder Singh. An MDP-based approach to online mechanism design. *Advances in neural information processing systems (NIPS)*, 16, 2003.
- 14 Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

- 15 Albert N Shiryaev. *Optimal stopping rules*, volume 8. Springer Science & Business Media, 2007.
- 16 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
- 17 Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. PAC model-free reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 881–888, 2006.
- 18 Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- 19 Andrew J Wagenmaker, Yifang Chen, Max Simchowitz, Simon Du, and Kevin Jamieson. First-order regret in reinforcement learning with linear function approximation: A robust estimation approach. In *International Conference on Machine Learning (ICML)*, pages 22384–22429, 2022.